Submission instructions as in previous homeworks.

1 (100 PTS.) Break up!

You are given a vertical pile of n books. The books are numbered $1, 2, \ldots, n$ (in this order from the bottom of the pile). The ith book has weight w_i. Given a pile of books $[\![i \,..\, j]\!]$, you can pick an index $i \le k < j$, and break the pile into two piles $[\![i \,..\, k]\!]$, and $[\![k+1 \,..\, j]\!]$. Since the books are heavy, performing such an operation requires $\sum_{\ell=i}^{j} w_\ell$ units of energy, where w_ℓ is the weight of the ℓth book.

The task at hand is to break the given pile of n books into n piles of single books, using the above breakup operation. Note, that given a pile of books, the energy required to break it up into two piles is always the same – it is the total weight of the pile. In particular, moving repeatedly just the top book of the pile is probably not going to be the optimal solution.

For example, if the initial pile is B_0 is $[\![1 \,..\, 5]\!]$ a solution might be a sequence of commands:

- Break $[\![1 \,..\, 5]\!]$ into two piles $[\![1 \,..\, 2]\!]$ and $[\![3 \,..\, 5]\!]$.
- Break $[\![1 \,..\, 2]\!]$ into two piles $[\![1]\!]$ and $[\![2]\!]$.
- Break $[\![3 \,..\, 5]\!]$ into $[\![3 \,..\, 4]\!]$ and $[\![5]\!]$
- Break $[\![3 \,..\, 4]\!]$ into $[\![3]\!]$ and $[\![4]\!]$.

It is easy to verify that the total energy required to implement this solution is $2w_1 + 2w_2 + 3w_3 + 3w_4 + 2w_5$.

1.A. (70 PTS.) Let $\mathcal{E}$ be the minimum total energy required by any solution that breaks the given pile into singletons. Describe an algorithm, as fast as possible, that computes $\mathcal{E}$. Bound the running time of your algorithm.

Solution:

Given a set of books $[\![1 \,..\, n]\!]$, let's consider an algorithm $\textbf{minEnergy}(i, j)$ that outputs the minimum amount of energy it takes to split books $[\![i \,..\, j]\!]$ into $j - i + 1$ piles, given an array of weights $w[i..j]$. We have a simple base case when $i = j$; this would just be a single pile of one book, which can be split no further, so it would take 0 units of energy. In the recursive case, we need to consider all the possible ways to split books $[\![i \,..\, j]\!]$ into two piles $[\![i \,..\, l]\!]$ and $[\![l+1 \,..\, j]\!]$ and take the minimum total energy to split each of those two piles.

Stated as a recurrence, we have:

$$\textbf{minEnergy}(i,j) = \begin{cases} 0 & \text{if } i \ge j \\ \sum_{k=i}^{j} w[k] + \min_{i \le l \le j-1} \left\{ \textbf{minEnergy}(i,l) + \textbf{minEnergy}(l+1,j) \right\} & \text{otherwise} \end{cases}$$

We can memoize all possible values of $\textbf{minEnergy}(i, j)$ in a two-dimensional array $\text{dp}[1..n, 1..n]$ and the sum $\sum_{k=i}^{j} w[k]$ in another two-dimensional array $\text{dpsum}[1..n, 1..n]$. In terms of evaluation order, we can see that $\text{dp}[i, j]$ depends on entries $\text{dp}[k, j]$ and $\text{dp}[i, l]$ where $i < k < j$ and $i + 1 < l < j$.

Thus, we have the following algorithm:

```
minEnergy(w[1..n]):
  for i ← 1 to n do
    for j ← 1 to n do
      dp[i, j] ← 0
      dpsum[i, j] ← NONE
  for i ← n to 1 do
    for j ← i to n do
      if i ≠ j then
        if dpsum[i, j] = NONE then
          dpsum[i, j] ← sum(w[i..j])
        dp[i, j] ← dpsum[i, j] + min_{i≤k≤j−1} dp[i, k]+dp[k + 1, j]
  return dp[1, n]
```

Filling in an entry $\text{dp}[i, j]$ takes $O(n)$ time, since we must find the minimum of $j - i$ sums, which is bounded by n. So we have a total running time of $O(n^3)$. Since we keep two $n \times n$ arrays, the space complexity is $O(n^2)$.

1.B. (30 PTS.) Describe how to modify your algorithm in (A) so that it computes the solution itself (i.e., the sequence of breakup operations in the optimal solution). Describe the algorithm/procedure that outputs this solution once it is computed.

Solution:

To compute the solution itself, we can keep track of another table called 'splits', where $\text{splits}[i, j]$ will keep track of the start and end indices for the two piles that mark an optimal split for books $[\![i \,..\, j]\!]$. The sequence of piles can be obtained by following the indices in 'splits' until we reach piles of size 1.

```
minEnergy(w[1..n]):
  for i ← 1 to n do
    for j ← 1 to n do
      dp[i, j] ← 0
      dpsum[i, j] ← NONE
      splits[i, j] ← NONE
  for i ← n to 1 do
    for j ← i to n do
      if i ≠ j then
        if dpsum[i, j] = NONE then
          dpsum[i, j] ← sum(w[i..j])
        submin ← ∞
        for k ← i to j − 1 do
          s ← dp[i, k] + dp[k + 1, j]
          if s < submin then
            submin ← s
            minind ← k
        splits[i, j] ← [(i, minind), (minind+1, j)]
        dp[i, j] ← dpsum[i, j] + submin
  // Let's assume printCoords has access to splits as a global variable
  printCoords(1, n)
```

```
// prints the weights of the books in each intermediate pile
printCoords(i, j):
  print w[i..j]
  if i ≠ j then
    for x, y in splits[i, j] do
      printCoords(x, y)
```

---

2 (100 PTS.) Wireless routers.

You are given n locations $\ell_1, \ldots, \ell_n$ of people living on Red street. Here ℓ_i is the distance of the ith person from the beginning of the street in feet (Red street is straight), where $\ell_1 < \ell_2 < \cdots < \ell_n$.

We would like to install k wireless routers to serve these people need. Specifically, for a set of locations $Y = \{y_1, \ldots, y_k\}$, the cost of this solution to the ith customer is the distance of ℓ_i to its nearest wireless routers in Y to the power four. Formally, it is $\text{cost}(\ell_i, Y) = \min_{y \in Y}(y - \ell_i)^4 = (\ell_i - \mathsf{nn}(\ell_i, Y))^4$, where $\mathsf{nn}(\ell_i, Y)$ is the location of the nearest point to ℓ_i in Y. Indeed, the further a person's computer is from a wireless router, the stronger the signal his computer has to use to communicate with the router, and the energy of this signal grows as (say) a fourth power of the distance.

The ***cost*** of the solution Y is $\text{cost}(Y) = \sum_i \text{cost}(\ell_i, Y)$.

Given the n locations $\ell_1, \ldots, \ell_n$ and k, provide an algorithm, as fast as possible (in n and k), that computes the set $Y \subseteq \{\ell_1, \ldots, \ell_n\}$ of k routers, such that $\text{cost}(Y)$ is minimal. What is the running time of your algorithm? (Note, that the routers can be placed only in the given locations of the houses[1].)

Solution: This is the solution

```
Cost(L[0...n − 1]):
  cost ← 0
  leftRouter ← L[0]
  rightRouter ← L[n − 1]
  for i ← 1 to n − 2 do
    rightCost ← L[0] − L[i]
    rightCost ← rightCost·rightCost
    leftCost ← L[i] − L[n − 1]
    leftCost ← leftCost · leftCost
    if leftCost < rightCost then
      cost ← cost + leftCost · leftCost
    else
      cost ← cost + rightCost · rightCost
  return cost
```

Our algorithm works in the following way. It seeks to create the case where there are a substring of locations where the beginning and end locations have a router and none of the others do. Then we place a router at one of the middle locations and thus create two sub-problems that look exactly the same as this with one less router to place.

[1] The variant where the routers can be placed anywhere is slightly harder – you can think about this variant for fun. Then you can think about your life, and what you do for fun.

The first if statement covers the trivial case where there are more routers than locations, in which case it is easy to cover these locations such that the cost is 0.

The next if statement covers the case where we have no routers left to place. Then because we assume the first and last location have a router and none of the others do, it is easy to calculate the cost of this substring based on the cost function specified by the problem which is implemented in **Cost**.

Finally the inner for loops are the recursive step. We choose a location from the middle to place a router. That creates two substrings where the first and last location have routers. We recurse on all possibilities of distributing the remaining routers among the two substrings. Of course the results are cached in our memo table dp. This drastically improves the efficiency.

There is one final step for completeness where we place two imaginary routers infinitely far away from Red Street. Don't worry about this part, but that is what the first line does.

```
FindCosts(L[0...n − 1], n, k):
  L ← concat([−∞], L, [∞])
  dp[n + 3][n + 3][n + 3]
  for a ← 2 to n + 2 do
    for numRouters ← 0 to k do
      for c ← 0 to n + 2 − a do
        if numRouters ≥ a − 2 then
          dp[a][numRouters][c] ← 0
          continue
        if numRouters = 0 then
          subStrCost ← Cost(L[c..(c + a − 1)])
          dp[a][numRouters][c] ← subStrCost
          continue
        dp[a][numRouters][c] ← ∞
        numRoutersLeft ← numRouters −1
        for rss ← c + 1 to c + a − 2 do
          for leftRouters ← 0 to numRoutersLeft do
            rightRouters ← numRoutersLeft - leftRouters
            leftSubCost ← dp[rss − c + 1][leftRouters][c]
            rightSubCost ← dp[c + a − rss][rightRouters][rss]
            thisScenarioCost ← leftSubCost+rightSubCost
            minScenarioCost ← dp[a][numRouters][c]
            if thisScenarioCost < minScenarioCost then
              dp[a][numRouters][c] ← thisScenarioCost
  return dp[n + 2][k][0]
```

The running time of Cost is as follows. There is one for loop bounded by the length of the array L. Therefore Cost runs in $O(L)$.

The running time of **FindCosts** is as follows. There are three outer loops, two are bounded by n and one is bounded by k. Within the body of outer loops, there is a call to **Cost**. **Cost** runs linearly with the length of the array passed to it. Next come two for loops one bounded by n and the other bounded by k. Their body runs in O(1) time. Composed loops multiply the running time of their body by the product of their bounds. Therefore the running time of **FindCosts** will be $O(n^2 \cdot k\Big(n + n \cdot k\Big)) = O(n^3 \cdot k^2)$

3 (100 PTS.) A bridge too far.

You are planning a military campaign, and you had decided to break your army into two corps P and Q (a corps is a military unit bigger than a division). Corps P would occupy cities $p_1, p_2, \ldots, p_n$ (in this order), while the other corps Q would occupy cities $q_1, \ldots, q_m$ (in this order). The problem of course is that if P is in city p_i, and it is being attacked, then the second corps Q (which might be at city q_j), should be close enough to be able to provide support to P[2].

To this end, you are given numbers nm numbers, where $d(i,j)$ is the distance between p_i and q_j, for $i = 1, \ldots, n$ and $j = 1, \ldots m$. You are also given a threshold ℓ. You need now to schedule the movements of the two corps. Specifically, you need to output a schedule $(i_1, j_1), \ldots, (i_t, j_t)$. Here (i_k, j_k) denotes that in the beginning of the kth week of the campaign P would be at city p_{i_k}, and Q would be at city q_{j_k}.

Formally, the requirements on the computed schedule are the following:

(I) $i_1 = 1$ and $j_1 = 1$ (i.e., the campaign starts with P at p_1 and Q at q_1).

(II) $i_t = n$ and $j_t = m$ (i.e., the campaign ends with P at p_n and Q at q_m).

(III) For any k, $1 \leq k \leq t$, we have $d(i_k, j_k) \leq \ell$.

(IV) For any k, we have that either $i_{k+1} = i_k + 1$ or $j_{k+1} = j_k + 1$ (but not both – only one corps can move at any point in time). [As such, $t = n + m - 1$.]

3.A. (50 PTS.) Given the above input, provide an algorithm, as efficient as possible, that computes whether there is a feasible schedule that complies with all the above conditions. What is the running time of your algorithm? Argue shortly why your algorithm is correct (and why the stated running time is correct).

Solution:

Let's consider a function **feasible?**(k,t) that returns whether or not there exists a feasible schedule given that P is at city p_k and Q is at city q_t.

We can define a recurrence for this function as follows:

$$\textbf{feasible?}(k,t) = \begin{cases} \texttt{TRUE} & \text{if } k = n \text{ and } t = m \text{ and } d(k,t) \leq l \\ \texttt{FALSE} & \text{if } k > n \text{ or } t > m \text{ or } d(k,t) > l \\ \textbf{feasible?}(k+1,t) & \text{if } t = m \\ \textbf{feasible?}(k,t+1) & \text{if } k = n \\ \textbf{feasible?}(k+1,t) \vee \textbf{feasible?}(k,t+1) & \text{otherwise} \end{cases}$$

We can memoize the results of this function in a $n \times m$ array 'dp', where dp$[i,j]$ yields whether or not a feasible schedule exists from city p_i and q_j. To fill this table, notice that if you fix a cell, the value of that cell depends on the cells immediately to the right and below (assuming (0,0) is at the top left corner). Thus we must fill the table in reverse order, from n to 1 and m to 1.

Here, we assume that m, n, and d are global variables.

[2] Failing to have such close by support might end up in a military disaster – see the movie "a bridge too far".

```
feasible?(l):
  for i ← m down to 1 do
    for j ← n down to 1 do
      dp[i, j] ← FALSE
  for t ← m down to 1 do
    for k ← n down to 1 do
      if k = n and t = m and d(k, t) ≤ l then
        dp[t, k] ← TRUE
      else if k > n or t > m or d(k, t) > l then
        dp[t, k] ← FALSE
      else if t = m then
        dp[t, k] ← dp[t, k + 1]
      else if k = n then
        dp[t, k] ← dp[t + 1, k]
      else
        dp[t, k] ← dp[t + 1, k] ∨ dp[t, k + 1]
  return dp[1, 1]
```

Filling up the memo table takes $O(mn)$ time since there are mn entries, each of which requires only $O(1)$ time to compute. This algorithm also has an $O(mn)$ space complexity.

3.B. (10 PTS.) Describe how to modify the algorithm in (A) so that it outputs the feasible schedule.

Solution:

To output the feasible schedule, we can keep another $n \times m$ memo table 'sched' where sched$[i, j]$ contains a tuple representing the movement for the next week that would take the corps to the destination cities; i.e. sched$[i_k, j_k] = (i_{k+1}, j_{k+1})$.

Assume 'sched' is initialized to `NONE` for each element in the beginning. In line 12, we add a line below to store sched$[i, j] \leftarrow (t, k+1)$ if dp$[t, k+1] =$ `TRUE`. Similarly, in line 14, we add a line to store sched$[i, j] \leftarrow (t+1, k)$ if dp$[t+1, k] =$ `TRUE`. On line 16, we check both dp$[t+1, k]$ and dp$[t, k+1]$ and store the indices of the first of the two that contains `TRUE`, doing nothing if neither evaluates to `TRUE`.

To recover the schedule, we first check if dp$[1, 1]$ is `TRUE`. If so, we follow and print the indices starting from sched$[1, 1]$ until we reach sched$[m, n]$.

3.C. (40 PTS.) Modify your algorithm so that it computes and outputs the smallest value of ℓ, such that there is still a feasible schedule. What is the running time of your algorithm?

Solution:

Let $k = \max_{i,j} d(i, j)$. We can see that $0 \leq \ell_{\min} \leq k$. We can find k in $O(mn)$ time by looking at all pairs i, j over d. Now suppose we have an ℓ such that $0 \leq \ell \leq k$ and **feasible?**$(\ell) =$ `FALSE`. By requirement (III) above, we can see that **feasible?**$(\ell') =$ `FALSE` for all $\ell' < \ell$. Using this, we can find $\ell_{\min}$ by performing binary search over $[0, k]$.

```
minLFeasible():
  maxval ← −∞
  for i ← 1 to n do
    for j ← 1 to m do
      if d(i, j) > maxval then
        maxval ← d(i, j)
  return search(0,maxval)
```

```
search(a, z):
  if a ≥ z then
    return z
  mdpt ← ⌊(a + z)/2⌋
  if feasible?(mdpt) then
    return search(a, mdpt)
  else
    return search(mdpt+1, z)
```

This algorithm has a time complexity of $O(mn \log k)$ since there are $\log k$ calls to **feasible?**, which has $O(mn)$ running time.

HW 6 Due on **Wednesday, October 25, 2017 at 10am**

CS/ECE 374: Algorithms & Models of Computation, Fall 2017 Version: **1.4**

Submission instructions as in previous homeworks.

Unless stated otherwise, for all questions in this homework involving dynamic programming, you need to provide a solution with explicit memoization.

1 (100 PTS.) No one expects the Spanish inquisition.

(This problem is somewhat easier than the usual homeworks problems.)

It is 1492 in Spain (the Jews were just kicked out of Spain, and Columbus just landed in the Bahamas [an interesting year]) – unfortunately, the people in the town that you live in decided that you are a witch, and it is only a question of time till the inquisition would come to investigate[1]. You can avoid the investigation by paying bribes.

Fortunately, you got your hands on the organizational chart of the inquisition, which is a tree. The leafs are the investigators, and the internal nodes are the supervisors. A person that corresponds to a node u of this tree T, can be bought off by paying b_u Maravedís (the Spanish money at the time), and then they would suppress any investigation by anybody in their subtree.

Given a tree T as above with n nodes, the problem is to compute the set of people that you should bribe, so that all possible investigations are suppressed, and that the total sum paid is minimal. As example, consider the following tree (on the left, and a solution on the right).

1.A. (30 PTS.) State the recursive formula (or formulas if needed) for computing the desired quantity – and explain the logic behind it (in one sentence). What is the running time of your algorithm? (The running time should be as good as possible, as usual.)

Note, that you need only to compute the price of the optimal solution – there is no need to output the solution itself.

Solution:

Let $C(v)$ denote set of children of node v in the tree T.

Then, for a node v in the tree, we have:

$$\textbf{minBribeCost}(v) = \begin{cases} b_v & \text{if } v \text{ is a leaf} \\ \min\left(b_v, \sum_{u \in C(v)} \textbf{minBribeCost}(u)\right) & \text{otherwise} \end{cases}$$

Since a supervisor can suppress investigations in their subtree, we must compare the cost of bribing the supervisor and the minimum cost of stopping the investigation further down in the subtree.

The recurrence for the above algorithm is given by:

$$T(n) = \Theta(1) + \sum_i T(k_i),\ T(1) = \Theta(1)$$

where k_i denotes the size of the subtree for the ith child of the root node under consideration.

Assume that $T(n') \leq cn'$ for $n' < n$. Then we have:

[1] For once, somebody is expecting the Spanish inquisition: https://www.youtube.com/watch?v=QqreRufrkxM.

$$
\begin{aligned}
T(n) &\leq an + \sum_i T(k_i) \\
&\leq an + \sum_i ck_i = an + c\sum_i k_i \\
&= an + c(n-1) \\
&\leq cn \qquad\qquad \text{assuming } a \leq c
\end{aligned}
$$

Thus, the running time of the above algorithm is $O(n)$, where n is the number of nodes in the tree.

1.B. (70 PTS.) Describe a dynamic program to solve the problem. Your algorithm should be implemented using explicit memoization, and should not use recursion. Analyze the running time of your algorithm (you need to provide pseudo-code). What is the running time and space of your algorithm?

Here, assume the input is given in an array $Z[1 \,..\, n]$ of nodes. The first entry (i.e., $Z[1]$) is the root. Every node has a field b specifying the bribe, and a field ℓ specifying the number of children it has, and an array/list C of the indices of the children nodes. You can assume that for a node $Z[i]$, all its children are stored in locations in Z with index strictly larger than i.

(Again, no need to output the set realizing the solution – just the price of this solution – you should think about how to print out the optimal solution.)

Solution:

```
minBribeCost(Z[1..n]):
  for i ← n down to 1 do
    if Z[i].ℓ = 0 then
      Z[i].minCost ← Z[i].b
    else
      minChildrenBribe ← 0
      for c in Z[i].C do
        minChildrenBribe ← minChildrenBribe + Z[c].minCost
      Z[i].minCost = min(Z[i].b, minChildrenBribe)
  return Z[1].minCost
```

We can see that the time to calculate the minimum bribe cost for a tree rooted at node v depends on the number of children of v. Since a child node only contributes a constant amount of time for its parent node and since there is at most one parent for any given node, we can see that the running time of the algorithm is $O(n)$.

Since we keep track of a single numerical value for each node in the tree, the space complexity of this algorithm is $O(n)$.

---

2 (100 PTS.) Fishing for staircases.

A DAG is a directed graph with no cycles. Given a DAG G with n vertices and m edges, one can compute in $O(n+m)$ time an ordering of its vertices $v_1, \ldots v_n$, such that if there is an edges $(v_i, v_j) \in \mathsf{E}(\mathsf{G})$, then $i < j$. This is known as ***topological sort*** (or ***topological ordering***) of G, and you can assume you are given a black box that can implement this operation in $O(n+m)$ time. A vertex in a DAG is a ***sink*** if it has no outgoing edges.

2.A. (25 PTS.) The ***value*** of a vertex v in a DAG G, is the length of the longest path in DAG that starts in v. Describe a linear time algorithm (in the number of edges and vertices of G) that computes for all the vertices in G their value.

Solution:

Assuming that the input graph is represented by an adjacency list with singly-linked lists indexed from $[1..n]$, we have:

```
computeValues(G):
  G ← topologicalSort(G)
  for i ← 1 to n do
    dp[i] ← 0
  for i ← n down to 1 do
    maxPathLen ← 0
    for each vertex v_j in G[i] do
      dp[i] ← max(dp[i], 1+dp[j])
  return dp
```

By performing topological sort on G, we induce a partial ordering on the vertices such that the values of all out-neighbors of a vertex v can be computed before the value of v itself, which depends on the values of its out-neighbors.

Using an adjacency list, we can see that each vertex v requires $O(\deg(v)+1)$ time to iterate through the out-neighbors of v. Furthermore, **topologicalSort** runs in $O(n+m)$ time. Thus, the total running time is $O\Big(\sum_{k=1}^{n}\big(\deg(v_k)+1\big)\Big)+O(m+n)=O(m+n)$.

2.B. (25 PTS.) Prove that if two vertices $u, v \in \mathsf{V}(\mathsf{G})$ have the same value (again, G is a DAG), then the edges (u,v) and (v,u) are not in G.

Solution:

Proof by contradiction

Suppose $u, v \in \mathsf{V}(\mathsf{G})$ have the same value and there exists an edge between u and v. Since G is a DAG, either (u,v) or (v,u) is in G but not both. Without loss of generality, suppose we have an edge (u,v) in G. Let the value of vertex v be x and the value of vertex u be y. Since G is acyclic, we can see that the longest path starting from v cannot contain u. Thus, $y \geq x+1$ because we can simply include the edge (u,v) to the longest path from v to get a path of length $x+1$ starting from u. However, this is a contradiction of the assumption that $x=y$. Therefore, we can conclude that the edges (u,v) and (v,u) are not in G. ■

2.C. (25 PTS.) Using (B), prove that in any DAG G with n vertices, for any k, either there is a path of length k, or there is a set X of $\lfloor n/k \rfloor$ vertices in G that is ***autonomous***; that is, there is no edge between any pair of vertices of X.

Solution:

If there is no path of length k, then the length of any path in G is at most $k-1$. Furthermore, we can state that for any vertex $v \in \mathsf{V}(\mathsf{G})$, the length of the longest path from v is in the range $[0, k-1]$. By the result obtained from HW0 Q1A, we can see that there must be at least $\lceil n/k \rceil$ vertices with

longest paths of the same length. Let X be the set of these vertices. By (2B), we can conclude that there cannot be an edge between any two vertices in X. ■

2.D. (25 PTS.) Consider a set of P of n points in the plane. The points of P are in general position – no two points have the same x or y coordinates. Consider a sequence S of points $p_1, p_2, \ldots, p_k$ of P, where $p_i = (x_i, y_i)$, for $i = 1, \ldots, k$. The sequence S is ***staircase***, if either

- for all $i = 1, \ldots, k-1$, we have $x_i < x_{i+1}$ and $y_i < y_{i+1}$, or
- for all $i = 1, \ldots, k-1$, we have $x_i < x_{i+1}$ and $y_i > y_{i+1}$.

Prove using (C) that there is always a staircase of length $\lfloor \sqrt{n} \rfloor$ in P. Describe an algorithm, as fast as possible, that computes the longest staircase in P.

Solution:

Let two points $p_i = (x_i, y_i), p_j = (x_j, y_j)$ in P be connected by a (directed) edge (p_i, p_j) if $x_i < x_j$ and $y_i < y_j$. We can see that any path in this DAG describes a staircase by the first condition. By (C), we can see that there is either a path of length $\lfloor \sqrt{n} \rfloor$ or a set of autonomous vertices X of size $\lfloor \sqrt{n} \rfloor$ (where $k = \lfloor \sqrt{n} \rfloor$). If we have a path of length $\lfloor \sqrt{n} \rfloor$, then we are done, since the path denotes a staircase. Otherwise, we can see that if you order the points in X such that $x_i < x_{i+1}$ for all points $p_i \in$ X for $i = 1, \ldots, \lfloor \sqrt{n} \rfloor$, then $y_i > y_{i+1}$. This is true because two points $p_i = (x_i, y_i), p_j = (x_j, y_j)$, where $x_i < x_j$, in the DAG are connected by an edge (p_i, p_j) iff $y_i < y_j$. If there does not exist an edge between p_i and p_j, then it must be the case that $y_i > y_j$ (since the points are in general position), in which case the second case for staircase is satisfied.

Notice that given a set of points sorted by x-values, the longest staircase is given by the longest increasing subsequence of y-values or the longest decreasing subsequence of y-values, whichever is greater.

The following algorithm assumes that P is an array of points.

```
longestStaircase(P):
  // sort by increasing x value
  P ← mergeSortByX(P)
  lis, lenI ←longestIncreasingY(P)
  lds, lenD ←longestDecreasingY(P)
  return lis if lenI > lenD else lds
```

```
longestIncreasingY(P[1..n]):
  dp[i] ← 1 for i ∈ [1..n]
  prev[i] ← 0 for i ∈ [1..n]
  for i ← 2 to n do
    maxval ← 0
    maxind ← 0
    for j ← 1 to i do
      if P[j].y < P[i].y and dp[j] > maxval then
        maxval ← dp[j]
        prev[i] ← j
      dp[i] = 1 + maxval
  lisIndex ← 0
  maxLengthSubseq ← 0
  for i ← 1 to n do
    if dp[i] > maxLengthSubseq then
      maxLengthSubseq ← dp[i]
      lisIndex ← i
  staircase[i] ← 0 for i ∈ [1..maxLengthSubseq]
  for ctr ← maxLengthSubseq down to 1 do
    staircase[ctr] ← prev[lisIndex]
    lisIndex ← prev[lisIndex]
  return staircase, maxLengthSubseq
```

longestDecreasingY has an identical structure, except line 7 is changed to: **if** $\mathsf{P}[j].y > \mathsf{P}[i].y$ and $\mathrm{dp}[j] > \mathrm{maxval}$

In **longestIncreasingY** and **longestDecreasingY**, we need iterate over n^2 values to fill up our memo table. To retrieve the actual sequence, we follow $O(n)$ entries in our secondary table 'prev'. Thus, both **longestIncreasingY** and **longestDecreasingY** have running times of $O(n^2)$.

Our algorithm **longestStaircase** requires $O(n \log n)$ time to sort P, $O(n^2)$ time to compute the longest increasing and decreasing y-subsequences, and $O(1)$ time to return the maximum length subsequence between the two. Therefore, our algorithm runs in $O(n^2)$ time.

2.E. **(Harder + not for submission.)** Using the algorithm of (D), describe a polynomial time algorithm that decomposes P into a set of $O(\sqrt{n})$ disjoint staircases. Prove the correctness of your algorithm.

3 (100 PTS.) Independence in orderly graphs.

For a graph G with n vertices, assigning each vertex of G a unique number in $\{1, \ldots, n\}$ is a ***numbering*** of its vertices.

A graph G with a numbering $\mathsf{V}(\mathsf{G}) = \{1, \ldots, n\}$ is ***k-orderly***, if for every edge $ij \in \mathsf{E}(\mathsf{G})$, we have that $-k \leq i - j \leq k$ (note, that it is not true that if $|i - j| \leq k$ then ij must be an edge in the graph!). A different numbering of the vertices of G might change the value of k for which G is k-orderly. Here are a few examples of a 3-orderly graph:

3.A. (20 PTS.) For any given value of k, show an example of a graph that is not k-orderly for any numbering of its vertices. Prove the correctness of your example. For credit your graph should have a minimal number of edges (as a function of k).

Solution:

Consider a graph G' with $2k+2$ vertices, where one vertex v is connected by an edge to each of the other $2k+1$ vertices. This graph has $2k+1$ edges.

Let p be the numbering of vertex v. Then there are two cases:

- Suppose $p > k+1$. Then v is connected to a vertex numbered 1. Since $p - 1 > k$, G' is not k-orderly.
- Suppose $p \leq k+1$. Then v is connected to a vertex numbered $2k+2$. Since $p - (2k+2) \leq k+1-(2k+2) = -k-1$, G' is not k-orderly.

We have shown in both cases that G' cannot be k-orderly for any arbitrary k.

3.B. (80 PTS.) Given a k-orderly graph G (here you are given the numbering of the vertices, and the value k [or, you can compute it directly from the numbering]), consider the problem of computing the largest independent set of G. As a reminder, a set of vertices $X \subseteq \mathsf{V}(\mathsf{G})$ is ***independent***, if no pair of vertices of X are connected by an edge of G.

Provide an algorithm, as fast as possible, for computing the size of the largest independent set in G. What is the dependency of the running time of your algorithm on the parameter k?

In particular, for credit, your solution for this problem should have polynomial time for k which is a constant. For full credit, the running time of your algorithm should be $O(f(k)n)$, where $f(k)$ is some (potentially large) function of k.

For this question you can use implicit memoization.

Hint: (A) Think about the vertices as ordered from left to right as above. Start with $k = 1$. Then, solve the problem for $k = 2, 3, 4, \ldots$. Hopefully, by the time you hit $k = 5$ you would be able to describe an algorithm for the general case. Think about defining a recursive function for this problem that has $\approx k$ parameters.

Solution:

```
int getLookup#(start, bucketContents[1..n]){
        int lookup#
        for(int element=start, element<start+k+1; element++){
                if(bucketContents[i] == true) :
                        lookup# += 1 << element - start
        }
        return lookup#
}

// bucketContents is a 1 x n array of booleans
int R(graph[1..N][1..N], int start, bucketContents[1..N]){

        // If there are no more elements to consider.
        if start == N - K :
                return 0

        // Look for memoized answer
        memoized = memoTable[start][getLookup#(start, bucketContents)]
        if memoized != -1 :
                return memoized
        // Look for memoized answer


        // Shift window
        newElement = start+k+1
        bucketContents[start] = false
        // Shift window

        // Consider not adding new element to set
        dont_include_new = R(graph, start+1, bucketContents)

        independent = true
        for(int i = start+1; i < start+K+1; i++){
                // If the new element is connected to a member of the bucket
                // then the bucket Contents are an independent set.
                if i<1:
                    continue
                if 1 <= newElement <= n and graph[newElement][i] == true:
                        independent = false
                        break
        }

        include_new = 0
        if independent == true :
                // Consider when new element is added to the bucket
                bucketContents[newElement] = true
                include_new = 1 + R(graph, start+1, bucketContents)
                bucketContents[newElement] = false

        // Record the answer in the memo table.
```

```
        memoTable[start][getLookup#(start, bucketContents)]
                = max(include_new, dont_include_new)
        return memoTable[start][getLookup#(start, bucketContents)]

}

int findMaxIndependentSet(graph[1..n]) {
        for (int i = 1; i <= N; i++)
                bucketContents[i] = false
        return R(graph, -K, bucketContents)
}
```

Our algorithm is an optimization on the brute force $O(2^n)$ algorithm which generates the power set of the graph and does a linear search through the power set to find the largest independent set. I.e. checking every subset for the largest independent set.

The optimization we found was that given a window over a range of K nodes in the graph, if from this window we make a bucket filled with independent elements, a new element from outside the window, independent from the elements of the bucket, will be independent from everything before the window.

Using this fact, we created a symmetrical subproblems that consist of a window of size $K + 1$, an a starting node, followed by some other elements.

From two of these subproblems, one where we slide the window and add the next element to our bucket, and one where we slide the window and don't add the next element to the bucket, we can solve the original problem by taking the maximum value of the subproblem.

The coordinates of subproblems were the elements in the bucket, of which there could only be $2^{(K+1)}$, and the starting point of the window, of which there were n.

To check whether the next element could be added to the bucket, we had to know whether it was independent from the elements in the bucket. Therefore we checked the new element's adjacency matrix (we assume the graph is given to us as an adjacency matrix) for any of the elements in the bucket. This takes K lookups.

The base case is when we run out of new elements to add.

The starting case is where we invent imaginary node elements, none of which are in the bucket, and the first new node is 1.

That is how our algorithm works. Based on the formula for the runtime of a memoized algorithm based on the recurrence,

the algorithm runs in $O(f(K)n) = O(K \cdot 2^K \cdot n)$.

because there are two memoization parameters, one bounded by 2^K the other by n, and the recurrence body takes $O(K)$ to run.

CS 374 HW2

Charles Swarts

September 2017

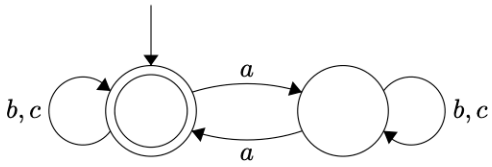
1

A

Question

All strings over $\{a, b, c\}$ in which every nonempty maximal substring of consecutive as is of even length.

Answer

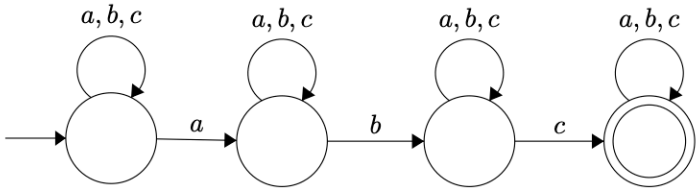


B

Question

$\Sigma^* a \Sigma^* b \Sigma^* c \Sigma^*$

Answer

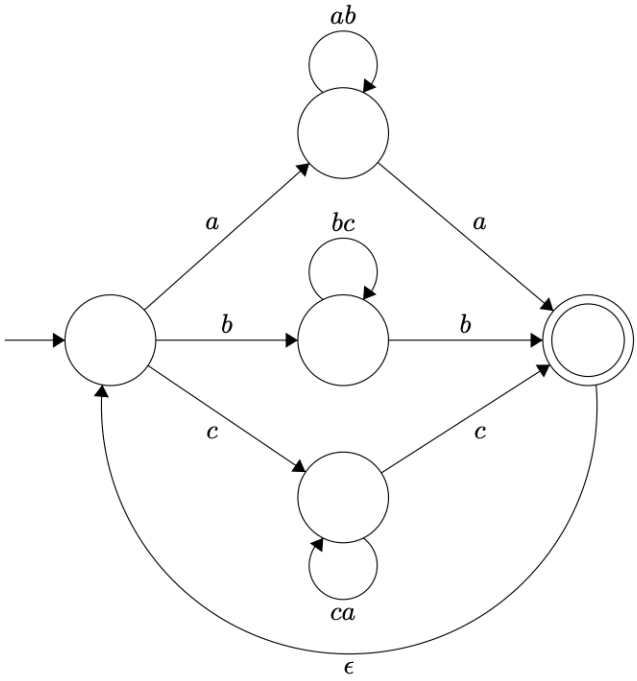


C

Question

$(a(a+b)^*a + b(b+c)^*b + c(c+a)^*c)^*$

Answer

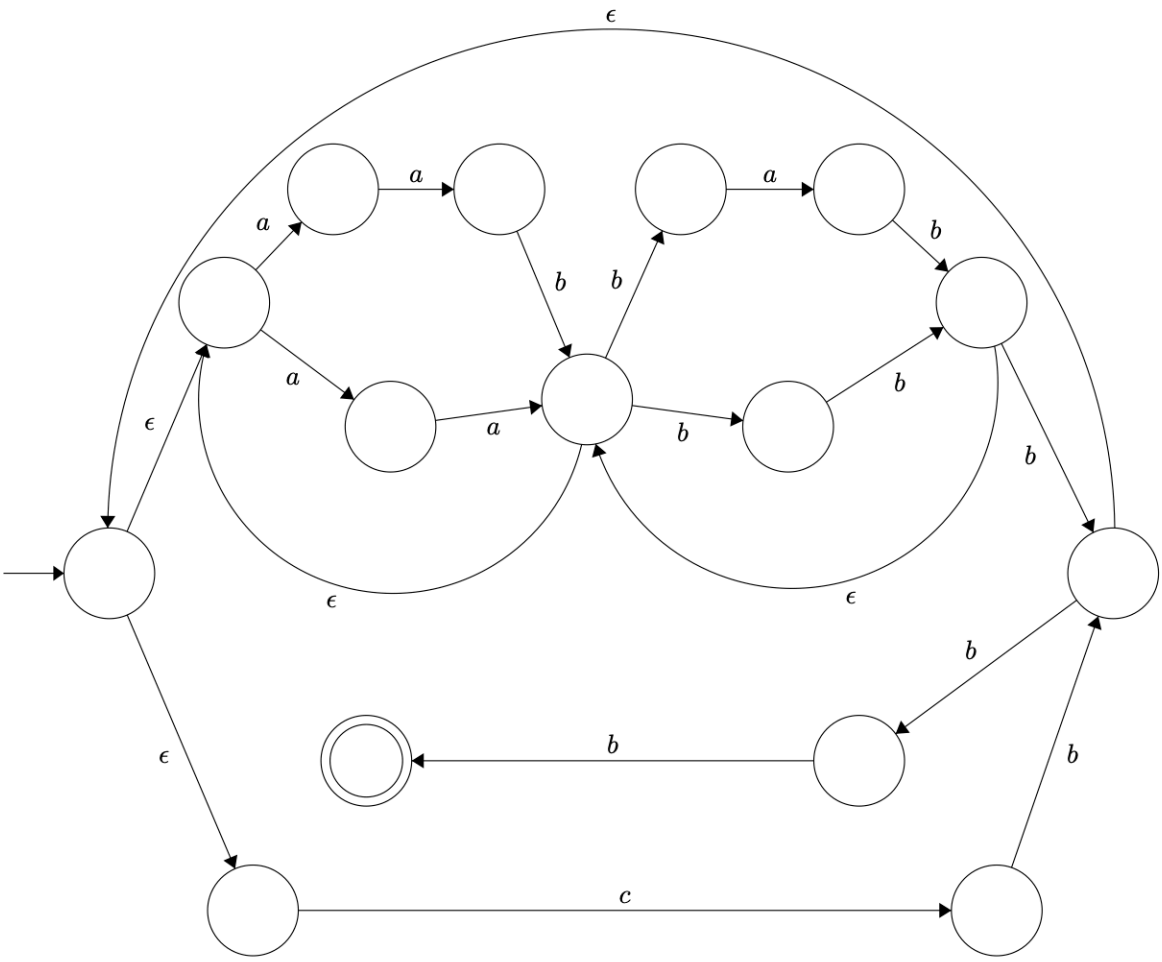


D

Question

$$\Big(\Big((aa+aab)^*(bab+bb)^*+c\Big)b\Big)^*+bb$$

Answer

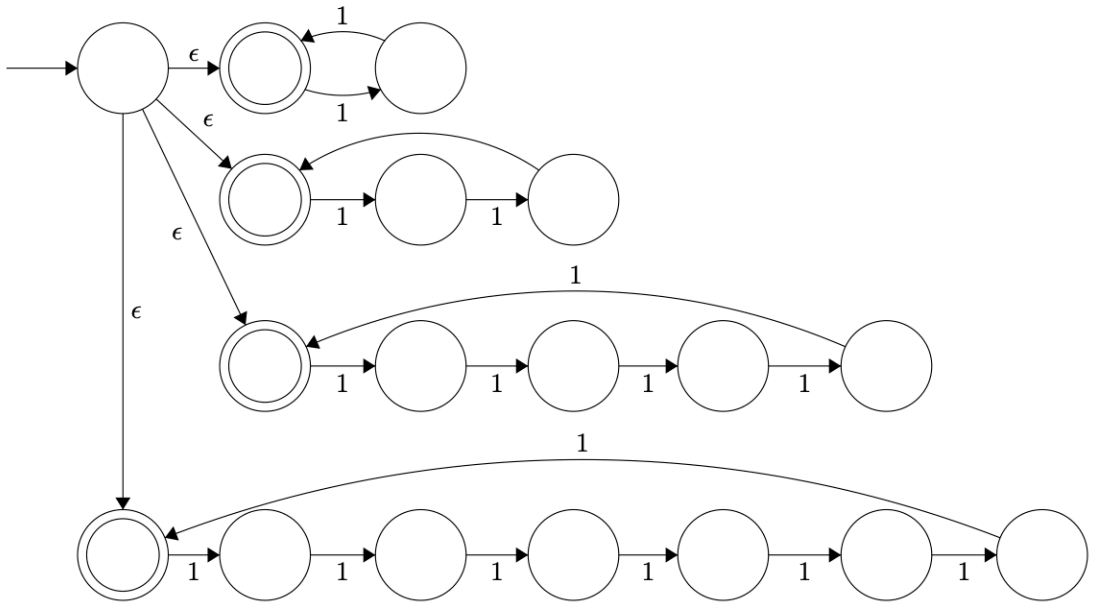


E

Question

All strings in 1^* of length that is divisible by at least one of the following numbers 2,3,5,7. For full credit your automata should have less than (say) 20 states.

Answer



F

Question

All strings in a^* of length that is NOT divisible by any of the following numbers 2, 3, 5, 7.

Answer

$$M = (Q, \Sigma, \delta, s, A)$$

$$\delta : Q \times \Sigma \to \mathcal{P}(Q)$$

$$A \subseteq Q$$

$$Q = \{q_i \mid i \in \mathbb{N},\ \ 0 \leq i < (2 * 3 * 5 * 7)\}$$

$$\Sigma = \{1\}$$

$$\delta(q_i, a) = \{q_{(i+1)\%(2*3*5*7)}\}$$

$$s = q_0$$

$$A = \{q_i \mid 2 \nmid i \text{ or } 3 \nmid i \text{ or } 5 \nmid i \text{ or } 7 \nmid i\}$$

2

A

Question

Consider the language $L_{\leq 1} = \{x \in \{0,1\}^* \mid \exists y \in L \text{ s.t. } d_H(x,y) \leq 1\}$. Describe in words what the language $L_{\leq 1}$ is.

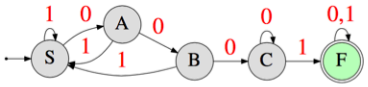
Answer

$L_{\leq 1}$ is every string that has a Hammind Distance of 1 or fewer from a string in L.

B

Question

Consider the following DFA M.



What is its language $L = L(M)$?

Answer

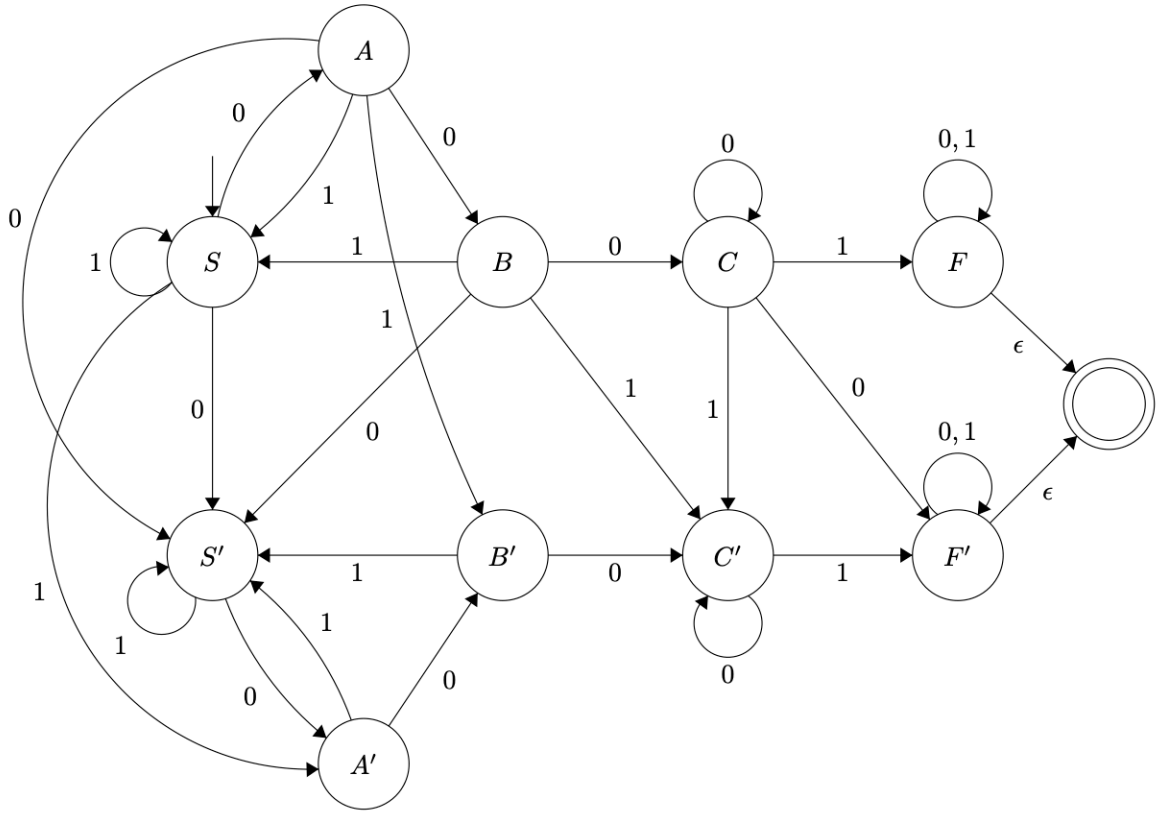
$\big((1+01)^*001\big)^*$ $\big((1+01)^*000\big)$ $0^*1(0+1)^*$

C

Question

By modifying the given DFA give above, describe an NFA that that accepts the language $L \leq 1$. Explain your construction.

Answer



This machine has M and a copy of M, M'. For every transition in M which goes from $(q_0, a) \to q_1$ there is a also a transition in my machine that goes from $(q_0, a') \to q_1'$. Thus this machine accepts any string in L, and accepts/tolerates one difference anywhere in the string from a string in L. Thus it tolerates any string with Hamming Distance 1 or 0 from a string in L.

D

Question

More generally, demonstrate that if a language $L \subseteq \{0,1\}^*$ is regular, then $L \leq 1$ is a regular language (for simplicity, you can assume $\epsilon \notin L$). Specifically, consider a DFA for L, and describe in detail how to modify it to an NFA for $L \leq 1$. (The description of the NFA does not have to be formal here.) Explain why the constructed NFA accept the desired language.

Answer

Since L is a regular language, it can be converted to an equivalent DFA M. We construct an NFA which has M and a copy of M, M'. For every transition in M which goes from $(q_0, a) \rightarrow q_1$ there is a also a transition in our constructed NFA that goes from $(q_0, a') \rightarrow \{q_1\}'$. Our constructed machine accepts any string in L, and accepts/tolerates one difference anywhere in the string from a string in L. Thus it tolerates any string with Hamming Distance 1 or 0 from a string in L. So the machine accepts any string in $L_{\leq 1}$. Since we can create a NFA for $L_{\leq 1}$, then that language is regular by the fact that an NFA can be converted into a regular expression, and so we can convert our constructed NFA into a regular expression which accepts $L_{\leq 1}$. By definition of regular expression, the language it accepts is regular.

E

Question

Prove, that for any constant k, the language $L_{\leq k}$ is regular. Your proof has to be formal and provide all necessary details. (I.e., you need to provide an explicit formal description of the resulting NFA for the new language, and prove that the NFA accepts the language $L_{\leq k}$).

Answer

Assume we have a regular language L and a natural number k. Where $L_{\leq k}$ is defined by the prompt. Also, since $L \subseteq \{0,1\}^*$, then L is a binary language. So, let $a' = 1$ if $a = 0$ and $a' = 1$ if $a = 0$.

Since L is regular, we know there is a DFA, $M = (Q, \Sigma, \delta, s, A)$ which accepts L.

Can construct an NFA N, defined as follows.

$$N = (Q_N, \Sigma, \delta_N, s_N, A_N)$$
$$\delta_N : Q_N \times \Sigma \to \mathcal{P}(Q_N)$$

$$Q_N = \{(q,i) \mid q \in Q \text{ and } i \in \mathbb{N} \text{ s.t. } 0 \leq i \leq k\}$$
$$s_N = (s, 0)$$
$$A_N = \{(a,i) \mid a \in A \text{ and } i \in \mathbb{N} \text{ s.t. } 0 \leq i \leq k\}$$

$$\delta_N(q,a) = \begin{cases} \{(\delta(p,a),i), (\delta(p,a'),i+1)\} & \text{if } q = (p,i) \text{ and } i < k \\ \{(\delta(p,a),i)\} & \text{if } q = (p,i) \text{ and } i = k \end{cases}$$

Now we prove that $L_{\leq k}$ is accepted by the NFA and only $L_{\leq k}$.

Lemma 1:

$$|z| < k \to \nexists z$$

We know if the Hamming Distance between two strings is greater than the length of one of those two strings, then one of those two strings doesn't exist, and is therefore is not in our language. □

Lemma 2: if v is a substring of a string in L, then $(\delta^*(q,v),i) \in \delta_N^*((q,i),v)$.

In any case of $\delta_N((q,i),a)$, $(\delta(q,a),i)$ is in the output set. So $\delta_N^*((s,0),v) = \delta_N(x_0 \in \delta_N(x_1 \in \delta_N(...), v_{|v|-1})$. So $(\delta^*(s,v),i) \in \delta^*((s,i),v)$. □

By Lemma 1, we know that $|z| \geq k$, and because of definition of Hamming Distance, $|z| = |y|$. So we can break y and z into

$$y = \left(\operatorname*{concat}_{i=0}^{d_H(y,z)-1} y_i a_i \right) y_{d_H(y,z)}$$
$$z = \left(\operatorname*{concat}_{i=0}^{d_H(y,z)-1} y_i a_i' \right) y_{d_H(y,z)}$$

Where each of the y_i are the equally positioned substrings where y and z match, and each a_i is a "bit" where they differ.

From our definition of δ_N we get that, for $i < k$

$$(\delta(q,a),i) \in \delta_N((q,i),a)$$

and

$$(\delta(q,a), i+1) \in \delta_N((q,i), a')$$

So starting from the same state, oposite input characters to δ_N produce outputs which contain N states with the same internal M state.

Lemma 2 also tells us that starting from two N states with the same internal M state, evaluating δ_N^* on the same string will produce an outputs which contain an N state with the same internal M state and the same integer they started out with.

So, from some starting state, evaluating δ_N on strings $y_i a_i$ and $y_i a_i'$ looks like

$$(\delta(q_i, y_i a_i), 0) \in \delta_N((q_i, 0), y_i a_i)$$

$$(\delta(q_i, y_i a_i), i+1) \in \delta_N((q_i, i), y_i a_i')$$

Which means after evalutaing all the $y_i a_i$ pairs of y and z, and then evaluating the suffix $y_(d_H(x,y))$, which by Lemma 2 will just advance the internal M state of each of our outputs, we get

$$(a \in A, 0) = (\delta^*(s,y), 0) \in \delta_N^*((s,0), y)$$
$$(a \in A, d_H(y,z)) = (\delta^*(s,y), d_H(y,z)) \in \delta_N^*((s,0), y)$$

Since $0 \leq d_H(y,z) \leq k$, both of these are valid states, and also Accepting states, so the NFA accepts $L_{\leq k}$

It also rejects strings not in $L_{\leq k}$.

Proof by contradiction. This means we assume the NFA accepts a string, z, which has a Hamming Distance $k+1$ from every string of length $|z|$ in L. For this to be the case, all output states of $\delta_N^*(s_N, z)$ must have their internal integer be $> k$, otherwise z would have had a Hamming Distance less than $k+1$ from some equal length string in L. But $\delta_N^*(s_N, z)$ cannot have an internal integer $\geq k+1$ because by definition of δ, if the internal count of an N state is k, then the option to deviate from any string in L is taken away because it can only make valid moves and keep the internal count at k. So this is a contradiction to our assumption that z has a Hamming Distance of $k+1$ from some string in L and still exists. So it must not exists, and therefore our NFA doesn't accept it. □

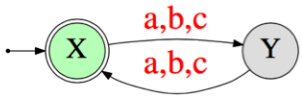
Since we can create a NFA for $L_{\leq k}$, then that language is regular by the fact that an NFA can be converted into a regular expression, and so we can convert our constructed NFA into a regular expression which accepts $L_{\leq k}$. By definition of regular expression, the language it accepts is regular. ■

3

A

Question

Let L be the language of the following DFA M. What is L?



Answer

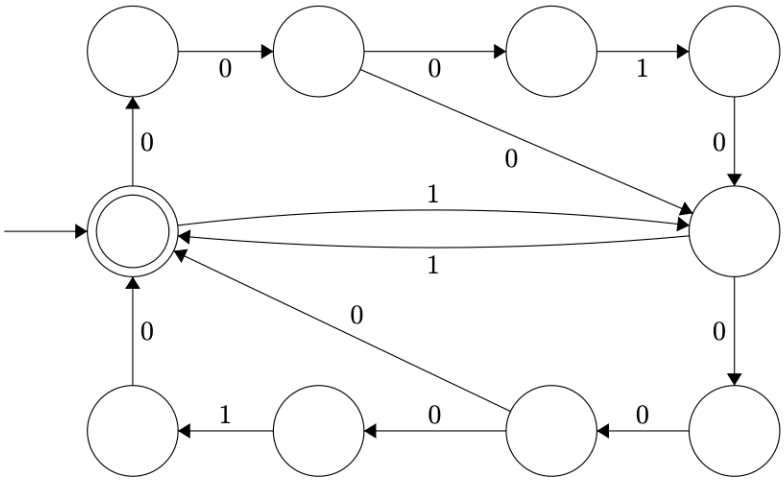
$((a+b+c)(a+b+c))^*$

B

Question

Working directly on the DFA M from (A) construct an NFA for the language $f(L)$. Here $f(L) = \{f(w) \mid w \in L\}$ is the code language. Where f is code from the above example.

Answer



C

Question

Let $L \subseteq \Sigma^*$ be am arbitrary regular language. Prove that the encoded language $f(L) = \{f(w) \mid w \in L\}$ is regular. Specifically, given a DFA $M = (Q, \Sigma, \delta, s, A)$ for L, describe how to build an NFA M' for $f(L)$. Give an upper bound on the number of states of M'. (I.e., You need to prove the correctness of your construction – that the language of the constructed NFA is indeed the desired language f(L).) (Rubric: Half the credit is for a correct construction, and the other half is for a correct proof of correctness.)

Answer

let us create a new function $g : \Sigma \times \mathbb{N} \to 0, 1$ which takes a symbol and a number and returns the numberth bit of the encoding of the symbol. e.g. $g(a, 1) = 0$ an $g(a, 4) = 1$

let us create a new function $h : \Sigma \to \mathbb{N}$ which takes a symbol and returns the length of the encoding.

$$M = (Q, \Sigma, \delta, s, A)$$

$$M' = (Q', \Sigma', \delta', s', A')$$

$$Q' = \{(q, \sigma, d, i) \mid q \in Q, \sigma \in \Sigma, d = \delta(q, \sigma), i \in \mathbb{N} \text{ s.t. } 1 \leq i \leq h(\sigma)\} \cup Q$$

This would put the upper bound of the number of states in M' at $|Q| + |Q| * \sum_{\sigma \in \Sigma} h(\sigma)$

$$\Sigma' = \{0, 1\}$$
$$s' = s$$
$$A' = A$$

$$\delta'(p, a) = \begin{cases} \{(q, \sigma, d, 1) \mid \sigma \in \Sigma, d = \delta(q, \sigma)\} & \text{if } p = q \text{ and } q \in Q \text{ and } a = \epsilon \\ \{d\} & \text{if } p = (q, \sigma, d, i) \text{ and } a = g(\sigma, h(\sigma)) \text{ and } i = h(\sigma) \\ \{(q, \sigma, d, i + 1)\} & \text{if } p = (q, \sigma, d, i) \text{ and } a = g(\sigma, i) \end{cases}$$

Proof that this is $f(L)$. If we want to prove this is $f(L)$, all we have to prove is that starting from any state in Q, (since both machines share Q as a subset of their states) that reading the code for a symbol, σ, into M' yields the same resulting state as reading σ into M.

Let us pick two arbitrary state q, p, in Q which are connected by a symbol $\sigma \in \Sigma$, such that $\delta(q, \sigma) = p$. Now we show $\delta'^*(q, f(\sigma)) = p$.

$$\delta'^*(q, f(\sigma)) = \delta'(\delta'(\delta'(...\delta'(q, g(\sigma, 1))...), g(\sigma, h(\sigma)))$$

Which, with the repeated application of δ' becomes

$$\delta'((q, \sigma, p, h(\sigma)), g(\sigma, h(\sigma)))$$

$$= \{p\}$$

a singleton set, so essentially just p. □

Now, because we have shown that there exists an NFA for $f(L)$, then we know a regular expression exists. Since a regular expression exists for $f(L)$, $f(L)$ is regular by definition of regular expression. ■

D

Question

Let $L \subseteq \{0,1\}$ be a regular language. Consider the decoded language $L(f) = \{w \in \Sigma^* \mid f(w) \in L\}$. Prove that $L(f)$ is a regular language. As above, given a DFA M for L, describe how to construct an NFA for $L(f)$. (Rubric: Half the credit is for a correct construction, and the other half is for a correct proof of correctness.)

Answer

$$M = (Q, \Sigma, \delta, s, A)$$

We construct an NFA M' to represent $L(f)$

$$M' = (Q, \Sigma', \delta', s, A)$$

$$\delta' : Q \times \Sigma \to \Sigma$$

$$A \subseteq Q$$

$$\Sigma = \text{an alphabet for which } f(\sigma \in \Sigma) \text{ is defined}$$

$$\delta'(q, a) = \{p \mid p = \delta^*(q, f(a))\}$$

Proof of correctness. All we have to do to prove this NFA, M', accepts $L(f)$ is to show that starting from any state, since both machines share the same state, that for some symbol $\sigma \in \Sigma$ reading in $f(\sigma)$ in M lands at the same state as reading σ in M'.

$$\delta^*(q, f(a)) = \delta'(q, a)$$

$$\delta^*(q, f(a)) = \{p \mid p = \delta^*(q, f(a))\}$$

$$\delta^*(q, f(a)) = \{\delta^*(q, f(a))\}$$

But since δ is from a DFA, $\delta^*(q, f(\sigma))$ only has one answer, so the set on the right is a singleton set, and essentially the equality holds. □

Now, because we have shown that there exists an NFA for $f(L)$, then we know a regular expression exists. Since a regular expression exists for $f(L)$, $f(L)$ is regular by definition of regular expression. ■

CS 374 HW3

Charles Swarts

September 2017

1

A

Question

Prove that the following language is not regular by providing a fooling set. You need to prove an infinite fooling set and also prove that it is a valid fooling set. The language is

$$L = \left\{ 0^k w \bar{w} 1^k \mid 0 \leq k \leq 3, w \in \{0,1\}^+ \right\}$$

Answer

To prove F is a fooling set, we want to show

$$\forall x, y \in F \; \exists z \text{ such that } xz \in L \oplus xz \in L$$

Here is my fooling set:

$$F = \{0^i \mid 1 \leq i\}$$

Proof it is valid. Suppose we take two elements of F, $x = 0^i$, and $y = 0^{\neg i}$. Then if $xz \in L$

$$xz = 0^i w \bar{w} 1^i$$

$$yz = 0^{\neg i} w \bar{w} 1^i$$

But as you can see, y is not in L, because it does not meet the definition. Therefore my set is a fooling set. ■

B

Question

Same as (A) for the following language. Recall that a run in a string is a maximal non- empty substring of identical symbols. Let L be the set of all strings in 0,1 that do not contain any two distinct runs of 0s of equal length. As an examples, L:

Answer

Here is my fooling set:

$$F = 0^*1$$

Proof it is valid. Consider two elements from the fooling set, $x = 0^i$ and $y = 0^{\neg i}$. Then we let $z = 0^i$, so

$$x = 0^i 10^i$$

$$y = 0^{\neg i} 10^i$$

So $y \in L$ and $x \notin L$, so F is a fooling set.

C

Question

Suppose you are given two languages L, L' where L is not regular, L' is regular, and L' is regular. Prove that $L \cup L'$ is not regular. Also, provide a counter-example for the following claim (it can be interpreted as an "inverse" of the above): Claim: Consider two languages L and L'. If L is not regular, L' is regular, and $L \cup L'$ is regular, then $L \cap L'$ is regular.

Answer

From Set Theory we know:

$$A \cap (B \setminus (A \cap B)) = \emptyset$$

And

$$A \cap C = \emptyset \rightarrow A \cup C \setminus C = A$$

So we take $(B \setminus (A \cap B))$ to be C

$$(A \cup \Big(B \setminus (A \cap B)\Big)) \setminus \Big(B \setminus (A \cap B)\Big) = A$$

We also know

$$A \cup (B \setminus (A \cap B)) = A \cup B$$

So an equivalent way to write this

$$\Big(A \cup (B \setminus (A \cap B))\Big) \setminus (B \setminus (A \cap B)) = A$$

Is this

$$\Big(A \cup B\Big) \setminus (B \setminus (A \cap B)) = A$$

If we plug in L for A and L' for B, we get

$$(L \cup L') \setminus (L' \setminus (L \cap L')) = L$$

Regular languages are closed under all of the operations on the left. So is $L \cup L'$, L', and $L \cap L'$ are regular then L must be regular. But L is not regular, and L', and $L \cap L'$ are regular, so $L \cup L'$ must not be regular.$\square$

To counter the claim, consider two Languages L is nonregular and $0, 1^*$. $L \cup 0, 1^* = 0, 1^*$. So their union is regular, but $L \cap 0, 1^* = L$. So their intersection is irregular. Thus there is a case where the claim is false, so the claim is false. $\square$

D

Question

Same as (A) for $L = \{0^{\lceil n \cdot lg(n) \rceil} \mid n \geq 3\}$, where $lgn = log_2(n)$.

Answer

Lemma1: There are infinite unique natural number outputs to

$$f(n) = \lceil (n+1) \cdot lg(n+1) \rceil - \lceil n \cdot lg(n) \rceil \quad \text{where } n \in \mathbb{N}$$

Proof:

We consider the real function:

$$g(x) = (x+1) \cdot lg(x+1) - x \cdot lg(x)$$

g is unbounded from above and monotone over the real numbers ≥ 1. $g(1) = 2$ Therefore for every natural number ≥ 2 is an output of the function.

Now if you consider a solution to $g(x) = n$ and a value $g(\lceil x \rceil)$.

$$g(x+1) \geq g(\lceil x \rceil) \geq g(x) \text{ since g is monotone}$$

$$g(\lceil x \rceil) \leq g(x) + \Big(g(x+1) - g(x)\Big)$$

And we know

$$g(x+1) - g(x) = (x+2)lg(x+2) - (x+1)lg(x+1) - \Big((x+1)lg(x+1) - xlg(x)\Big)$$

$$= (x+2)lg(x+2) + xlg(x) - 2(x)lg(x)$$

And we know

$$\frac{d}{dx}(x+2)lg(x+2) + xlg(x) - 2(x)lg(x) = \Big(log(x+2) + log(x) - 2log(x+1)\Big)/log(2)$$

$$\Big(log(x+2) + log(x) - 2log(x+1)\Big)/log(2) < 0$$

$$log(x+2) + log(x) - 2log(x+1) < 0$$

$$log((x+2)(x)) < log((x+1)^2)$$

$$log(x^2 + 2x) < log(x^2 + 2x + 1)$$

$$e^{x^2+2x} < e^{(x^2+2x+1)}$$

$$e^{x^2+2x} < e^{(x^2+2x)} \cdot e$$

$$1 < e$$

So the marginal gain of $g(x+1) - g(x)$ is smaller as x becomes larger. The marginal gain at 3 is $g(3+1) - g(3) < .5$. So $g(\lceil x \rceil)$ is within .5 of $g(x)$

Now we consider a function $h(x) = \lceil (x+1) \cdot lg(x+1) \rceil - \lceil x \cdot lg(x) \rceil$. Which is the same as $g(x)$, except that it has discritized the two components. There are 4 cases when we consider $h(x)$ against the equivalent input to $g(x)$. Both of the discrete components round up, and $h(x)$ is within 1 of $g(x)$, neither round up and $g(x) = h(x)$, or one of them round up and the other doesn't and the net affect is within 1 of $g(x)$.

So $g(x)$ is at most .5 away from $g(\lceil x \rceil)$, and $g(\lceil x \rceil)$ is at most .5 away from $h(\lceil x \rceil)$, so $f(x)$ is within 1.5 of $h(\lceil x \rceil)$, but because $g(x)$ and $h(\lceil x \rceil)$ produce natural number, they are within 1 of each other.

Since every natural number is an output of $g(x)$, and $h(\lceil x \rceil)$, is at most 1 away from $g(x)$, then $h(\lceil x \rceil)$ will produce at least every third natural number. Since the natural numbers are infinite, there are infinite and unique natural number outputs of $h(\lceil x \rceil)$. Since $f(n)$ and $h(\lceil x \rceil)$ are equivalent if $n = \lceil x \rceil$ and they are both over the same range, then $f(n)$ can produce infinite and unique natural number outputs. $\square$.

$$f(n) = \lceil (n+1) \cdot lg(n+1) \rceil - \lceil n \cdot lg(n) \rceil$$

We choose a fooling set

$$F = \{0^{\lceil nlg(n) \rceil} \mid k, n \in \mathbb{N} \text{ such that } f(n) = k, \text{ and } n \text{ is the lowest such number}$$

$$\text{where any element with a lower } n \text{ has a lower } k\}$$

We know the set is infinite because by Lemma 1, there are infinite natural number outputs of $f(n)$.

note: In lemma 1, we also proved $f(n)$ deviates no more than 1 from $g(x)$, So as n gets higher, $f(n)$ generally gets higher, so the added stipulation that lower values of n match lower values of $f(n)$ will not make the set finite.

Proof of validity:

$$x, y \in F$$

Without loss of generality, we say x is shorter. Then we take $z = 0^{k_x}$ where k_x is k from the definition of an element in F, but for x.

$$xz = 0^{\lceil n_x lg(n_x) \rceil} 0^{k_x}$$

$$yz = 0^{\lceil n_y lg(n_y) \rceil} 0^{k_x}$$

since $k_x = \lceil (n_x + 1) \cdot lg(n_x + 1) \rceil - \lceil n_x \cdot lg(n_x) \rceil$, we substitute

$$xz = 0^{\lceil n_x lg(n_x) \rceil} 0^{\lceil (n_x+1) \cdot lg(n_x+1) \rceil - \lceil n_x \cdot lg(n_x) \rceil}$$

$$yz = 0^{\lceil n_y lg(n_y) \rceil} 0^{\lceil (n_x+1) \cdot lg(n_x+1) \rceil - \lceil n_x \cdot lg(n_x) \rceil}$$

So

$$xz = 0^{\lceil (n_x+1) \cdot lg(n_x+1) \rceil}$$

$$yz = 0^{\lceil n_y lg(n_y) \rceil} 0^{\lceil (n_x+1) \cdot lg(n_x+1) \rceil - \lceil n_x \cdot lg(n_x) \rceil}$$

As you can see, xz is in the language.

We now explain why yz is not. It is because k is essentially the minimum number of zeros you have to add to get another string in L. We said y is longer than x, so it must have a greater n, and thus a greater k by definition. So when we add k_x 0s to y, it is enough to make y no longer in L but not enough to put it back in L. Thus $y \notin L$ and the fooling set is valid and infinite and the language is not regular. ■

2

Describe a context free grammar for the following languages. Clearly explain how they work and the role of each non-terminal. Unclear grammars will receive little to no credit.

A

Question

$$\{a^i b^j c^k d^l e^t \mid i, j, k, l, t \geq 0 \text{ and } i + j + k + l = t\}$$

Answer

For this question, I feel the shorthand is appropriate.

$$\begin{aligned} S &\to aSe \mid B \mid \epsilon \\ B &\to bBe \mid C \mid \epsilon \\ C &\to cCe \mid D \mid \epsilon \\ D &\to dDe \mid \epsilon \end{aligned}$$

This one works by allowing epsilon, because all the i-t could be 0. It runs on the principle that every time you add an a-d, you add an e. This keeps $i + j + k + l = t$. Then I add all the a's before the b's and b's before c's and c's before d's as prefixes because that ensures the correct ordering of the symbols. Each non terminal is a cascade which adds the next character in the sequence.

B

Question

$L = \{w \in \{0,1\}^* \mid \text{ there is a prefix } x \text{ of } w \text{ s.t.} \#_1(x) > \#_0(x)\}$

Answer

I also feel for this question, the shorthand is appropriate.

$$
\begin{aligned}
S &\to PW \\
P &\to PP \mid 01P \mid 10P \mid 0P1 \mid 1P0 \mid P01 \mid P10 \mid E \\
E &\to 1E \mid 1 \\
W &\to 1W \mid 0W \mid \epsilon
\end{aligned}
$$

This one works by immediately splitting the string into P for Prefix, and W for Whatever. The prefix works on the principle of making the string you want to avoid, and then twisting it at the end. It does this by adding an equal number of 0s and 1s each time, and in any configuration, then calling E for end, which adds 1s which make the prefix have more 1s than 0s, thus satisfying the condition. I also included PP in P so that I could call E in multiple places. W is whatever, because it contains any combination of 0s and 1s and epsilon.

3

$L = \{0^i 1^j 2^k \mid j = i + k\}$

A

Question

Prove that L is context free by describing a grammar for L.

Answer

I feel the shorthand is appropriate here.

$$
\begin{aligned}
S \to & BE \\
B \to & 0B1 \mid \epsilon \\
E \to & 1E2 \mid \epsilon
\end{aligned}
$$

This grammar works by immediately splitting the string into B for Beginning and E for End. The beginning and end add a 1 every time they add a 0 or 2, because that keeps $j = i + k$. The beginning deposits 0 followed by recurse followed by 1 to keep the order. Same for end but with 1 and 2.

B

Question

Prove that your grammar is correct.

Answer

If I can show that $B \rightsquigarrow^* 0^i 1^i \forall i \in \mathbb{N}$ and only that and $E \rightsquigarrow^* 0^k 1^k \forall i \in \mathbb{N}$ and only that, then, because my start immediately divides the string into B and E, I will have shown $L = L(G)$ where G is given in part A.

proof by induction on n, the number of times we derive from B. That B only derives all terminal strings of the form $0^i 1^i$.

base case, $n = 1$
$B \rightsquigarrow \epsilon$
$B \rightsquigarrow 0B1$
ϵ is in $= 0^0 1^0 = 0^{n-1} 1^{n-1}$ So this case is good. Our non-terminal string is $0^n B 1^n$.

inductive hypothesis: the terminal strings of $B \rightsquigarrow^{n=[1,n]}$ are of the form $0^{n-1} 1^{n-1}$, and the nonterminal strings are in the form $0^n B 1^n$.

inductive step: deriving B $n + 1$ times.

By the inductive hypothesis, the only non-terminal string of $B \rightsquigarrow^i$ is $0^n B 1^n$.
$0^n B 1^n \rightsquigarrow 0^n 1^n$ Which is a terminal string in the form specified by the inductive hypothesis, and the only non-terminal string is
$0^n B 1^n \rightsquigarrow 0^{n+1} B 1^{n+1}$ Which is in the form specified by the inductive hypothesis.

Thus, $B \rightsquigarrow^* \{0^i 1^i\} \forall i \in \mathbb{N}$ by a symmetric argument, $E \rightsquigarrow^* \{1^i 2^i\} \forall i \in \mathbb{N}$, so $L = L(G)$ ■

CS374 HW8

Charles Swarts

November 2017

1

Question

(This question was inspired by the game Open Flood [available as an app on android].) You are given a directed graph G with n vertices and m edges (here $m \geq n$). Every edge $e \in E(G)$ has a color $c(e)$ associated with it. The colors are taken from a set $C = \{1, ..., \xi\}$ (assume $\xi \leq n$), and every color $c \in C$, has price $p(c) > 0$.

Given a start vertex $s_0 = s$, and a sequence of $\Pi = \langle c_1, ..., c_l \rangle$ of colors, a compliant walk, at the ith time, either stays where it is (i.e., $s_i = s_i 1$), or alternatively travels a sequence of edges of color c_i that starts at s_i. Formally, if there is a path $\sigma \equiv (u_1, u_2), (u_2, u_3), ..., (u_{\tau 1}, u_\tau) \in E(G)$, such that $c(u_j, u_{j+1}) = c_i$, for all j, and $u_1 = s_i$, then one can set $s_{i+1} = u_\tau$. The price of Π is $p(\Pi) = \sum_{i=1}^{l} p(c_i)$.

Describe an algorithm, as fast as possible, that computes the cheapest sequence of colors for which there is a compliant walk in G from a vertex s to a vertex t. For full credit, your algorithm should run in $O(m \log m)$ time (be suspicious if you get faster running time). Correct solutions providing polynomial running time would get 50% of the points.

Answer

First we apply a transformation, createTransformedGraph() to the graph which creates a new graph. Then we apply Dijkstra. Dijkstra returns a **previous** array. We then run the **previous** through a one pass algorthm which converts that to Π.

To make this easier the following abstractions are used.

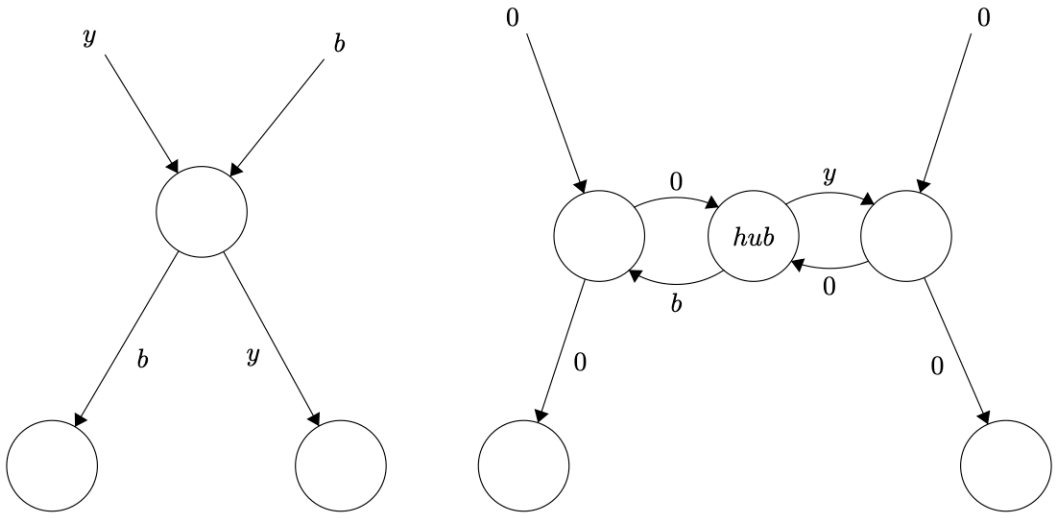
```
1: Graph:
2:     dict<(label, color),edgeList> nodes[]
3:
4: Edge:
5:     int color
6:     int cost
7:     (label,color) start
8:     (label,color) end
9:
10: Node
11:     int label
12:     int color
13:     Edge edgeList[]
```

```
1: createTransformedGraph(Graph oldGraph):
2:     newGraph = new Graph()
3:
4:     for each node in oldGraph.nodes:
5:         hub := Node(node.label,-1,[])
6:         newGraph.nodes[(node.label,-1)]=hub
7:         unique_colors := dict() //dictionary has O(n) to loop over all elements
8:                                 //and O(1) inserts and starts out with all entries as false.
9:
10:        for edge in node.edgeList:
11:            unique_colors[edge.color]=true:
12:        for color in unique_colors:
13:            platform_node := Node(node.label,node.color,[])
14:            newGraph.nodes[(node.label,node.color)].edgeList.add(platform_node)
15:            to_hub := Edge(node.color, node.color.cost, (platform_node.label,platform_node.color),(hub.label,hub.color))
16:            from_hub := Edge(node.color, 0, (hub.label,hub.color), (platform_node.label,platform_node.color))
17:            platform_node.edgeList.add(to_hub)
18:            hub.edgeList.add(to_hub)
19:            platform_node.edgeList.add(from_hub)
20:            hub.edgeList.add(from_hub)
21:
22:
23:    for each node in oldGraph.nodes:
24:        for edge in node.edgeList:
25:            newEdge = Edge(edge.color,0,(edge.start.label,edge.color),(edge.end.label,edge.color))
26:            newGraph.nodes[node.label,edge.color].edgeList.add(newEdge)
27:
```

Basically what this does is what the professor said. It makes every node into a subway station. Each of the colors represent a different subway train line. If a line goes to a particular station, then that station needs to have a platform for that line, and similarly if a train line of a certain color leaves the station, then the station needs a platform for that line. Thus we create a platform node for each unique train line ingressing or egressing. Then if you want to transfer lines, you have to pay a transfer fee which is the price of

the color of the line you are transfering to. This price is paid at the hub node which connects all platforms at a central location thus decreasing complexity. And except for this transfer fee, riding the subway if free.

More visually it makes arrangments on the left look like arrangements on the right.



Now we run through Dijkstra with the newGraph, the starting point (start,-1) with the ending point (end, -1) and this will be equivalent the cheapest subway path from the hub of the start to the hub of the end.

Dijkstra will return an array of **previous** which says which is an array of shortest path predecessor for each node. We take that and run it through

```
1: get_Π(previous[],start,end,num_V,color):
1:     if start == end:
2:         return []
2:     current := end
3:     Π_length := num_V
3:     Π := array[Π_length]
4:     Π[num_V] := color[previous[current]][current]
3:     while current != start:
4:         if color[previous[current]][current]!=Π[num_V]:
5:             num_V := num_V-1
6:             Π[num_V] := color[previous[current]][current]
5:         current := previous[current]
2:     return Π[num_V:Π_length]
```

Analysis: The transformation essentially looks at every edge twice and can add up to 4 internal edges for each edge. Otherwise the edges in the second outer loop are just the old edges shifted. Thus the new number of edges shared the same complexity with the old number of edges. However, the transformation has the potential to add a node for every edge, so the new number of nodes shared complexity with the number of edges. Thus when Dijkstra is run with a its running time of O(E+V log(V)) then V E and E=m, so this becomes O(m+m log(m)) = O(m log(m)).

The transformation looks through every edge and does constant work for each, so it runs in O(m). And get_Π runs in O(m) because it just backtracks through a 1 dimensional array of the previous node for each node, of which there are now m, doing constant time lookups.

Thus the full process takes O(m+m+m log(m)) = O(m log(m))

CS 411 HW1

Charles Swarts
swarts2@illinois.edu

Febuary 2017

1 Q1

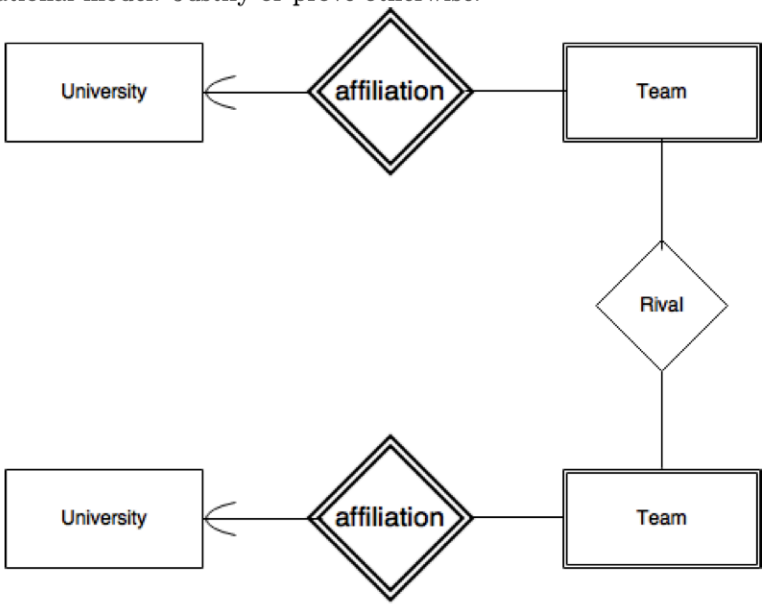
1.1 1

1. Consider the relation Person (Name, SSN, Age, Address, Gender). How many keys does the relation Person have? How did you arrive at your answer?

The Person relation will have between 1 and 5 keys, but most likely has 2. SSN is for sure a key because it is underlined, indicating it was created specifically to be a key. The other attributes could each be keys, but in the real world would not be. Since it is unlikely to find two Persons who share the same Name, Age, Address, and Gender, these four attributes could be used together as a key for most circumstances.

1.2 2

2. All relationships involving a weak entity set can be ignored while translating an ER diagram to a relational model. Justify or prove otherwise.



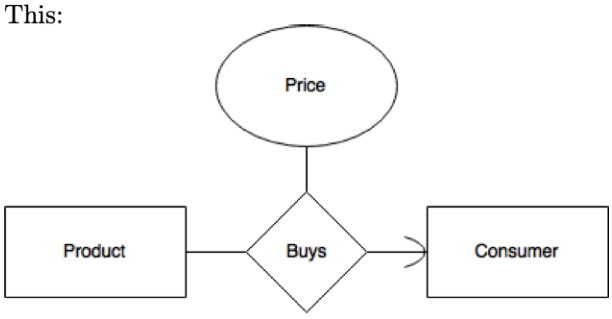
Rival is a relationship involving a weak entity set. When it is converted to the relational model, since it contains information that is not a part of either Team, it cannot be ignored.

1.3 3

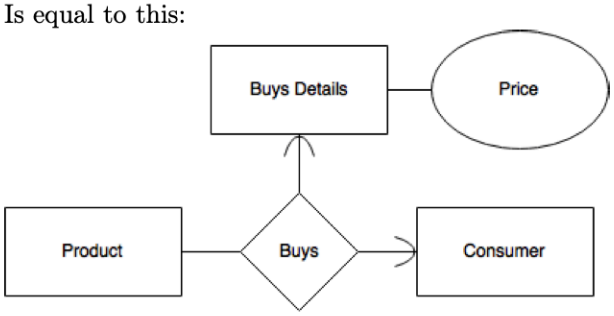
3. The expressiveness of ER models would reduce if we do not allow relationships to have attributes. Justify or prove otherwise.

The technical expressiveness of the ER model would not be affected by not allowing relationships to have attributes. This is because we can wrap any attributes into an entity and make that "details entity" as part of the relation. This works for any number of attributes.

This:



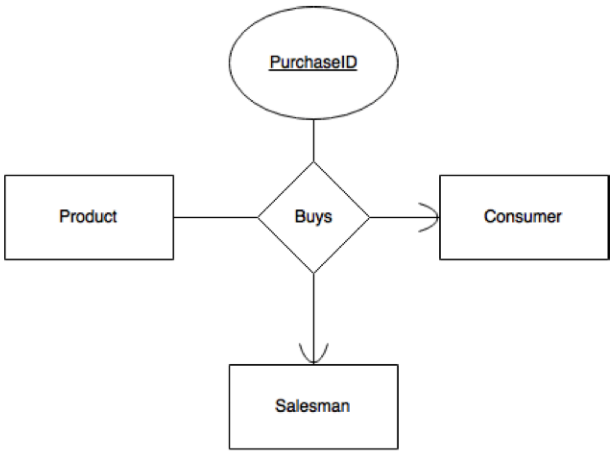
Is equal to this:



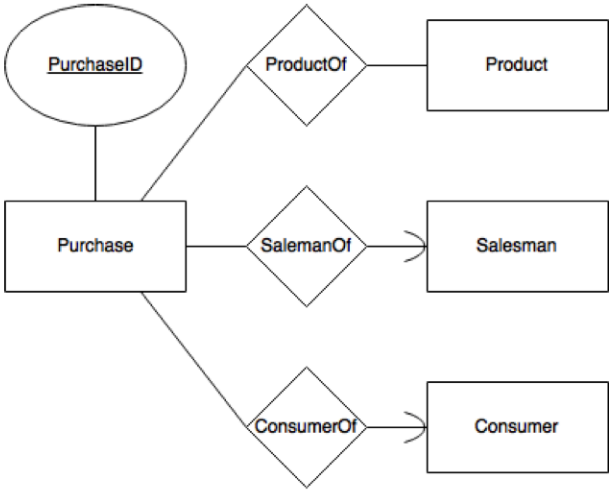
1.4 4

4. It is possible to transform a multiway relationship to multiple binary relationships without using weak entity sets. Justify or prove otherwise.

Yes it's absolutely possible. All we need to do is give the purchase details an ID.
We can transform this:



Into this:



1.5 5

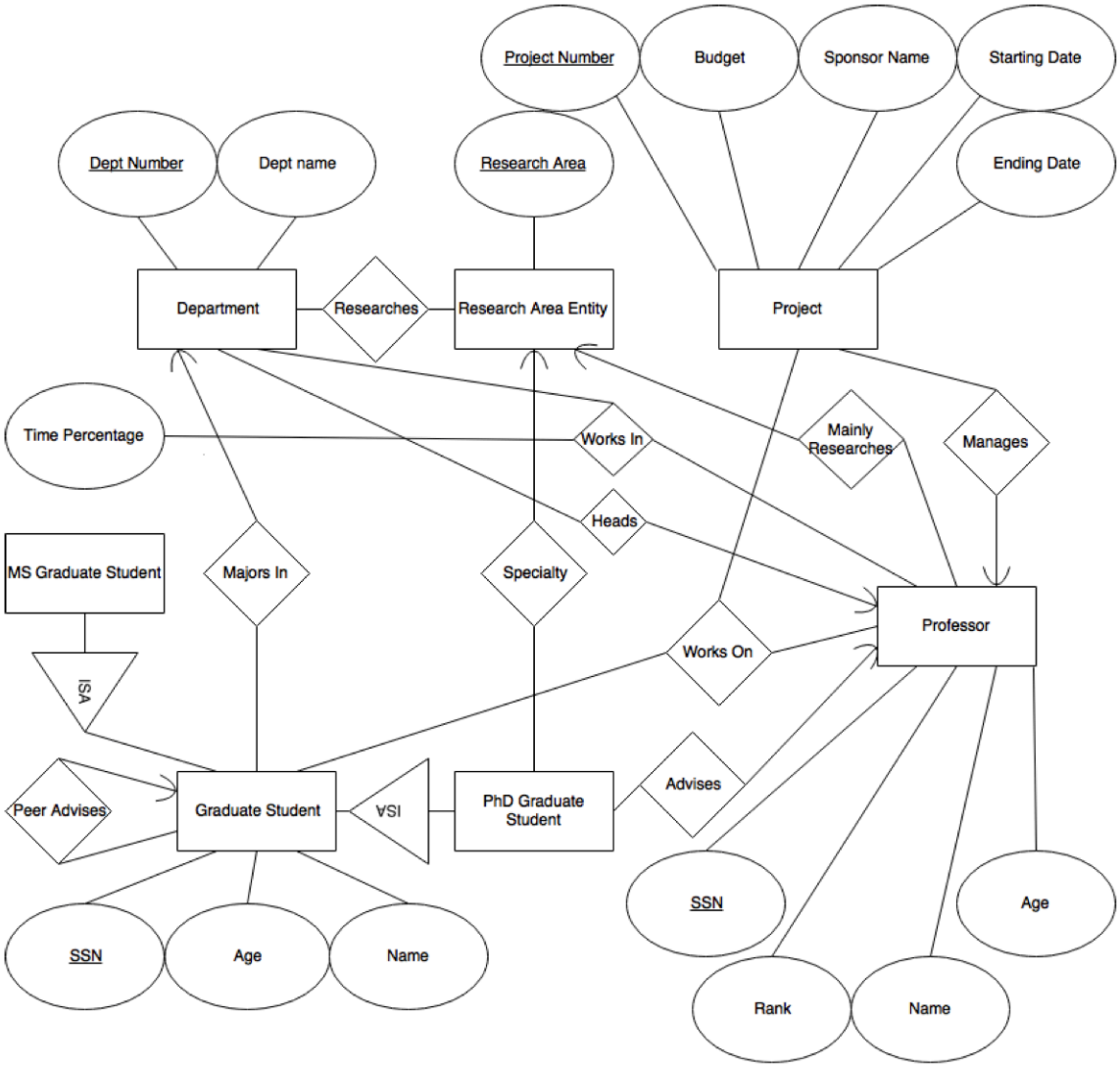
5. The reason why we prefer to combine the relation corresponding to an entity set A, with the relation corresponding to a relationship B—where B is a many-one relationship from A to another entity set—is because we want to improve the efficiency of queries involving A.
Doing it this way definitely saves space. It would also improve the efficiency of most queries involving A because the non-key information of A will be available in the singleton relation.

2 Q2

Consider the following information about a database of a university.

- Departments have a department number, department name, and many research areas.
- Professors have an SSN, a name, an age, a rank and a main research area.
- Projects have a project number, a sponsor name, a starting date, an ending date and a budget.
- Graduate students have an SSN, a name, and an age. They major in a single depart- ment.
- Graduate students can either be an MS or a PhD. PhD students need to determine their speciality/main research area.
- All PhD students have a professor as an advisor.
- Each project is managed by one professor.
- Each project is worked on by one or more professors, and one or more graduate stu- dents.
- Graduate students can work on multiple projects.
- Every department has a head, who is a Professor. 2
- Professors can work in one or more departments. For each department they work in, there is an associated time percentage.
- Graduate students have one major department in which they are working towards their degree
- Each graduate student has another senior graduate student as a mentor.

Design and draw an ER diagram that captures the aforementioned information. Indicate the key of each entity, as well as the multiplicity of your relationships. You are free to use annotation tools such as Mac Preview or Microsoft PowerPoint to draw the ER diagrams. Please do not include scanned pictures. You may want to check out draw.io. Note: state your assumptions clearly. Since there are many correct answers, your ER diagram will be evaluated considering your assumptions.



- I assumed that numeric attributes were made to be keys.
- I assumed that Research Area's are distinguishable.
- I assumed Graduate Students aren't Graduate Students unless they have a department, that Projects aren't Projects unless they have a professor managing them etc. and so I used the more rigid "exactly one" relationship multiplicity indicator to convey rigid academic bureaucracy.
- I assumed that while each graduate student has an adviser, that did not preclude a graduate student advising many other graduate students.
- I assumed that professors can work on more than one project.

3 Q3

3.1 1

1. Convert the ER model from the previous question to a relational model.

Here are the schemas of my Relational model:

Department(Dept Number, Dept Name, Heading Professor)

Department-Research(Dept Number, Research Area)

Project(ProjectNumber, Budget, Sponsor, Name, Starting Date, Ending Date, Managing Professor)

Professor(SSN, Age, Name, Rank, Research Area)

Graduate-Student-MS(SSN, Age, Name, Major Dept, Peer Adviser)

Graduate-Student-PhD(SSN, Age, Name, Major Dept, Peer Adviser, Specialty, Faculty Adviser)

Works On(Project Number, Participant SSN)

WorksIn(Dept Number, Professor SSN, TimePercentage)

3.2 2

2. Which approach did you use to convert the subclass entity set? Show us alternative schema designs.

I used the Object-Oriented approach. I split Graduate Student into two categories: Graduate-Student-MS and Graduate-Student-PhD.

Alternative Schemas for this would have been:

NULL-Values approach:
(assumes you can only either be a MS or PhD student)

Graduate-Student(SSN, Age, Name, Major Dept, Peer Adviser, Specialty, Faculty Adviser, isPhd)

Or

ER approach:

Graduate-Student(SSN, Age, Name, Major Dept, Peer Adviser, isPhd)
PhD-Extras(SSN, Specialty, Faculty Adviser)

3.3 3

3. Compare between all the designs you came up with in part 2. Talk about scenarios when each alternative would be a better choice in comparison to the rest.

The of the three approaches, the object oriented approach will do best on most searches because it pre-splits MS and PhD students into different groups, and both groups contain all information about the student.
A query where OO would beat the other two approaches is "find all PhD students who study nano-foobars and are age 26". Because the information would be in the same table, unlike the ER approach, and

would have prefiltered the PhD students, unlike the Null values approach.

A query where the null valued approach may be best would be "find the peer adviser for each student and the faculty adviser for each PhD student." Because all the information would be in the same table instead of in two tables like for the other two approaches.

A query where the ER approach may be best would "Find all Phd faculty advisors" because it would be in a table with less data compared to the other two approaches.

CS 425 HW4

Charles Swarts

May 2017

1 RPC/RMI

Experiment with an RPC compiler. We recommend using Apache Thrift, but you can also use, e.g., rpcgen or other RPC compiler. Using the interface definition language, define an interface for a key-value store supporting Put, Get, and Delete operations. Use the compiler to generate the stub and skeleton implementations of the protocol.

Include with your submission a printout of your interface definition, as well as **one page** each from the generated skeleton and stub files.

2 Two-phase Locking

a

Consider the following transaction. Show where the following happens:

- Read lock is acquired (R)
- Write lock is acquired (W)
- Read lock is upgraded to write lock (R→W)
- Lock is released (U)

`balance = A` (R on A)
`balance += B` (R on B)
`balance += C` (R on C)
`A -= 7` (R→W on A)
`D = balance - 7`(W on D; U on A,B,C,D)
`balance` here is a local variable; A, B, C, D are objects that are being read/updated by the transaction

b

Is the following interleaving of two transactions serially equivalent? Why or why not?

T1	T2
b1 = A	
	b2 = B
	b2 += C
	A = b2
	D = 10
b1 += D	
E = b1	

`b1` and `b2` are variables local to the transaction.

No they are not serially equivalent because if start with the values A→0;B→0;C→0;D→0;E→0 and go through T1 first then T2, we have E=0 at the end, but if we go through with the proposed ordering,

we get E=10. Therefore they are not serially equivalent, because this order does not produce the same results as serial ordering.

c

Consider the following two transactions:

```
T1:
b1 = A
b1 += B
D = b1
C = b1

T2:
b2 = B
b2 += C
D = b2
```

Show an interleaving that is impossible with two-phase locking when only exclusive (write) locks are used, but is possible when both shared (read) and exclusive locks are used.

T1	T2
b1 = A	
	b2 = B
b1+=B	
D=b1	
C=b1	
	b2+=C
	D=b2

d

Given the two above transactions, show an interleaving that results in a deadlock.

T1	T2
b1 = A	
	b2 = B
b1+=B	
D=b1	
	b2+=C
C=b1	
	D=b2

e

(5 points (bonus)) Show an interleaving that is impossible with two-phase locking when shared and exclusive locks are used, but is serially equivalent.

T1	T2
	X=20
Y=20	
X=20	
	Y=20

3 Distributed File Systems

In the Vanilla DFS we talked about during class, you will notice that there is no "open" and "close" API as POSIX, so there is no file descriptor. What is the advantage of not having a file descriptor?

Under the assumptions of Indy, in a file descriptive system, the server would have to maintain an underlying data-structure for the file descriptor. To maintain overall recover-ability, the server would have to ensure this data-structure is recoverable. So, the advantage of a file system without file descriptors is that it is faster because it doesn't have ensure this underlying data-structure is persistent. (Maintaining persistence is slow because it involves writing to disk.)

4 Two-phase Commit

a

Two-phase commit is essentially implementing consensus. Is it still using the crash-stop failure model?

From lecture - "The failure model of two-phase commits is actually sometimes called the 'crash-stop-recover' model." Based on that, I'd say Two-phase commit uses the crash-stop-recover model and not the crash-stop model.

b

Raft is a consensus algorithm we learned before. Under a slow network (messages timeout fre- quently), which algorithm is more likely to let a transaction go through, Raft or two-phase commit? Why?

Raft is more likely to let a transaction go through. This is because both algorithms make use of a coordinator, so the bandwidth is the same for both, so the network speed doesn't matter. What's different is that Raft only needs a majority, whereas 2PC needs unanimity. Also every node in 2PC has veto power, so that reduces 2PC's chances of letting a transaction through. Raft can also survive network partitions, so if the slow network is interpreted as failure of nodes, Raft is designed to keep working/heal quickly. For these reasons, I believe Raft will let more transactions through.

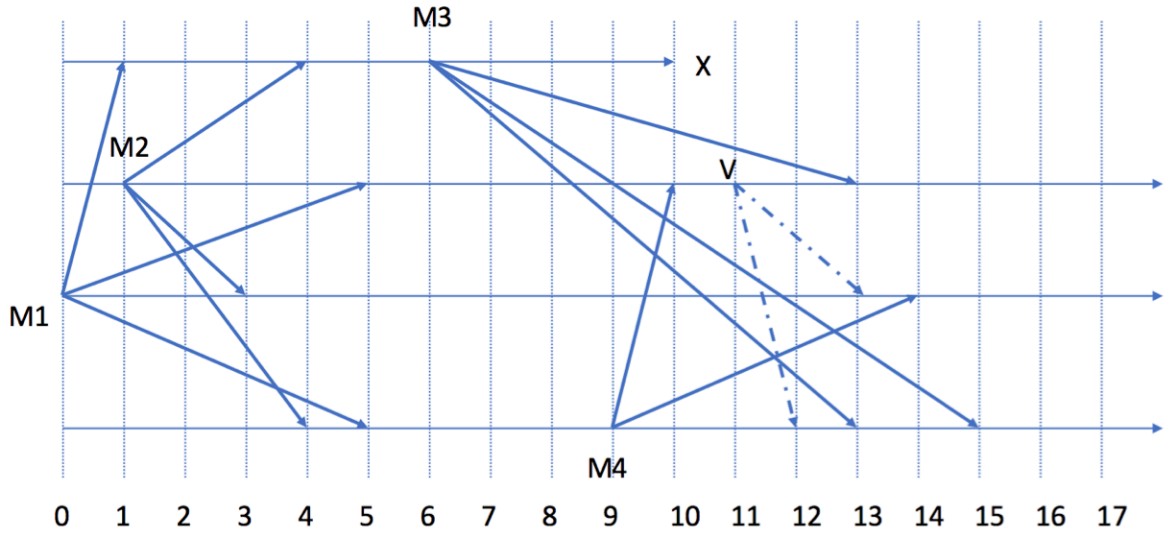
c

In a decentralized variant of the two-phase commit protocol the participants communicate directly with one another instead of indirectly via a coordinator. In phase 1, the coordinator sends its vote to all the participants. In phase 2, if the coordinator's vote is No, the participants just abort the transaction; if it is Yes, each participant sends its vote to the coordinator and the other participants, each of which decides on the outcome according to the vote and carries it out. Calculate the number of messages and the number of rounds it takes. What are its advantages and disadvantages in comparison with the centralized variant? Let's assume that network is fast, so messages won't timeout if machine doesn't crash.

This will take two rounds. One for each phase. It won't take more because if a node crashes, the others will just abort. There will be $(n-1) + (n-1)(n-1) = n(n-1)$ messages because each node sends its decision to every other node. The disadvantage of this system is it takes more bandwidth than regular 2PC. Other than that it's just as persistent. One advantage is that if the coordinator fails, then the rest of the nodes don't think they have been partitioned.

5 View synchronous communication

Consider the diagram below:

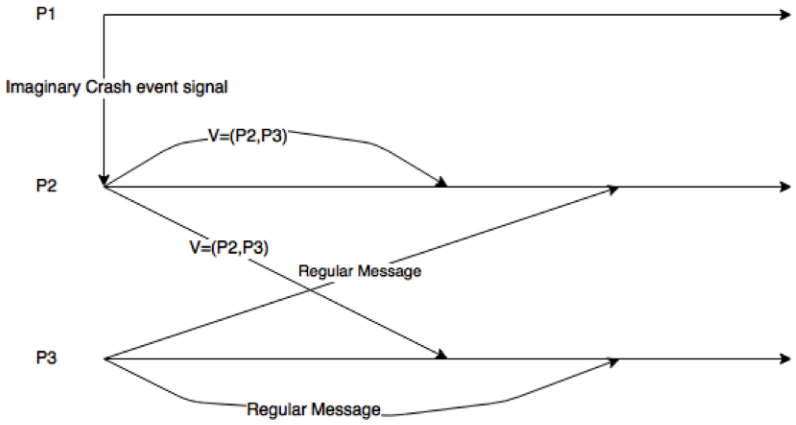


It shows communication among 4 processes which send 4 messages. The arrows shows when each message is sent by a process and when it is received by each other process. At time 10, the first process crashes (X). At time 11, process 2 detects this and sends a view change message (V) to the other two processes.

a

Consider that messages are delivered in causal order by following the rule that a received message is held back if message that causally precede this message have yet to be received. Once the missing messages are received, the messages get delivered immediately. (This is the same set up as in the first midterm.) View changes are treated as just another type of message and are delivered according to the same rules. Does this approach ensure view synchrony? If not, explain which of the requirements for view synchrony are violated.

No, here is a sequence which defeats causal ordering scenario.

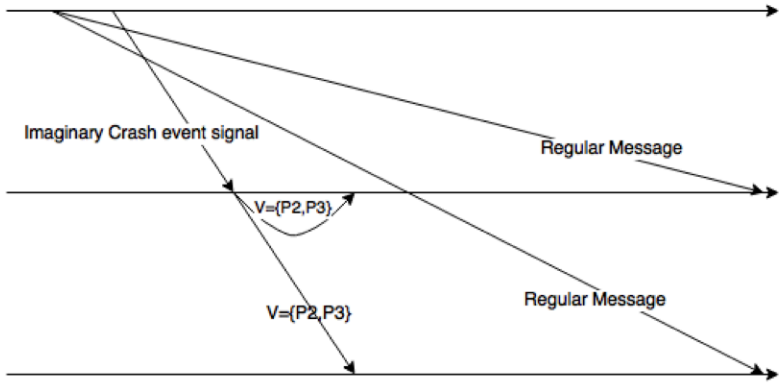


As can be seen in the counter example, the messages are causally ordered, but because the view change signal of P2 and the Regular Message of P3 are concurrent, there is nothing in causal ordering that prevents the Regular Message from coming after and being in the next view thus violating the "What happens in the View stays in the View."

b

Repeat the above part using a totally ordered delivery mechanism, where the order of messages is based upon the time that the first other process receives them. (E.g., for M1 this would be t = 1 when it is received by the first process.)

No, here is a sequence which defeats total ordering scenario.

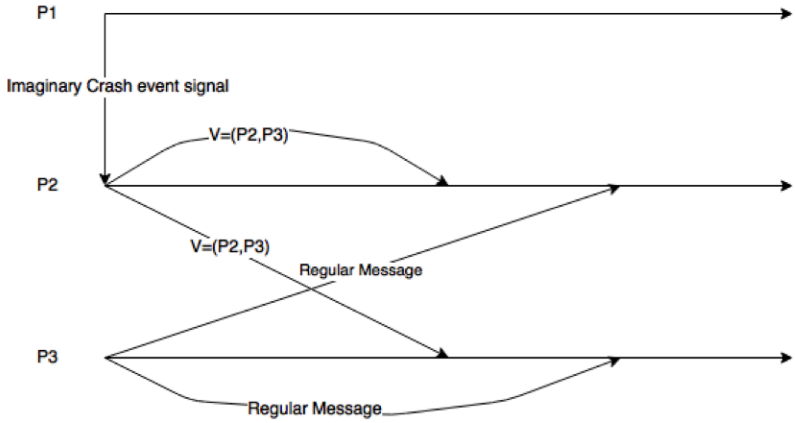


As can bee seen in the counter example, the messages are totally ordered, but the regular message from P1 gets delivered in the next view thus violating the "What happens in the View stays in the View."

c

Repeat the above part using a causal-total message delivery mechanism.

No, here is a sequence which defeats causal-total ordering scenario.



As can be seen, these messages are causally ordered and totally ordered, but still violate "What happens in the View stays in the View."

6 MapReduce

You are given a list of users and a corresponding list of their followers on a social network. Write a Map-Reduce pseudo code to find those pair of users (a,b), who have at least 20,000 followers and follow each other. You can have map-reduce chains to do this, however you are required to give a solution which uses the minimum number of maps and reduces.

The input will be of type List[Pair[[UserID],List[UserID]]]. E.g., in Python the input might look like:

```
   [ ("Alice", ["Bob","Carol","David"]),
     ("Bob", ["Alice", "David"]),
     ("Carol", ["Alice", "Evan"]),
     ("David", ["Alice", "Bob"]),
     ("Evan", ["Carol"])]

def mapFunc((Person,Followers)):
    if len(Followers)>20,000:
        emit (Person,Followers[i]).sort() for i in range(0,len(Followers))

def reduceFunc([key = (Person1,Person2)]):
    total = 0
    for number of times key appears in list
        total += 1
    if total==2:
        emit Person1
        emit Person2
```

This is my formula, with the functions defined above: (reduce reduceFunc (map mapFunc INPUT))

Interface Definition.

```
namespace py tutorial

service KeyValueStore
{
        string Get(1:string key);
        void Put(1:string key, 2:string value);
        void Delete(1:string key);
}
```

This is the skeleton.

```
class Processor(Iface, TProcessor):
    def __init__(self, handler):
        self._handler = handler
        self._processMap = {}
        self._processMap["Get"] = Processor.process_Get
        self._processMap["Put"] = Processor.process_Put
        self._processMap["Delete"] = Processor.process_Delete
    def process(self, iprot, oprot):
        (name, type, seqid) = iprot.readMessageBegin()
        if name not in self._processMap:
            iprot.skip(TType.STRUCT)
            iprot.readMessageEnd()
            x = TApplicationException(TApplicationException.UNKNOWN_METHOD, 'Unknown function %s' % (n
            oprot.writeMessageBegin(name, TMessageType.EXCEPTION, seqid)
            x.write(oprot)
            oprot.writeMessageEnd()
            oprot.trans.flush()
            return
        else:
            self._processMap[name](self, seqid, iprot, oprot)
        return True

    def process_Get(self, seqid, iprot, oprot):
        args = Get_args()
        args.read(iprot)
        iprot.readMessageEnd()
        result = Get_result()
        try:
            result.success = self._handler.Get(args.key)
            msg_type = TMessageType.REPLY
        except (TTransport.TTransportException, KeyboardInterrupt, SystemExit):
            raise
        except Exception as ex:
            msg_type = TMessageType.EXCEPTION
            logging.exception(ex)
            result = TApplicationException(TApplicationException.INTERNAL_ERROR, 'Internal error')
        oprot.writeMessageBegin("Get", msg_type, seqid)
        result.write(oprot)
        oprot.writeMessageEnd()
        oprot.trans.flush()
    def process_Delete(self, seqid, iprot, oprot):
        args = Delete_args()
        args.read(iprot)
        iprot.readMessageEnd()
        result = Delete_result()
        try:
            self._handler.Delete(args.key)
            msg_type = TMessageType.REPLY
        except (TTransport.TTransportException, KeyboardInterrupt, SystemExit):
            raise
        except Exception as ex:
```

This begins the stub.

```
class Client(Iface):
    def __init__(self, iprot, oprot=None):
        self._iprot = self._oprot = iprot
        if oprot is not None:
            self._oprot = oprot
        self._seqid = 0

    def Get(self, key):
        """
        Parameters:
         - key
        """
        self.send_Get(key)
        return self.recv_Get()

    def send_Get(self, key):
        self._oprot.writeMessageBegin('Get', TMessageType.CALL, self._seqid)
        args = Get_args()
        args.key = key
        args.write(self._oprot)
        self._oprot.writeMessageEnd()
        self._oprot.trans.flush()

    def recv_Get(self):
        iprot = self._iprot
        (fname, mtype, rseqid) = iprot.readMessageBegin()
        if mtype == TMessageType.EXCEPTION:
            x = TApplicationException()
            x.read(iprot)
            iprot.readMessageEnd()
            raise x
        result = Get_result()
        result.read(iprot)
        iprot.readMessageEnd()
        if result.success is not None:
            return result.success
        raise TApplicationException(TApplicationException.MISSING_RESULT, "Get failed: unknown result"

    def Put(self, key, value):
        """
        Parameters:
         - key
         - value
        """
        self.send_Put(key, value)
        self.recv_Put()

    def send_Put(self, key, value):
        self._oprot.writeMessageBegin('Put', TMessageType.CALL, self._seqid)
        args = Put_args()
        args.key = key
        args.value = value
        args.write(self._oprot)
```

CS 438 HW2

Charles Swarts

October 2017

1

Recall that BitTorrent uses a choking mechanism to allocate bandwidth to peers. It "un- chokes" four peers who have given it the best download performance (tit-for-tat), plus one of the remaining peers chosen at random (optimistic unchoking). The choice of four best peers, as well as the random choice are changed every 30 seconds.

Suppose Bob joins a BitTorrent swarm with 100 other peers. Let each peer, including Bob, have an upload speed of 10Mbps and unlimited download speed. Consider what happens in the phase of the protocol where for each pair of peers A and B, A has some blocks that B wants and vice versa; i.e, any peer can productively download data from any other peer.

A

Question

Suppose Bob wants to upload no data; i.e., be a free rider. What will be his average download speed?

Answer

Because Bob does not contribute, **in the long run** he will never be a well connected neighbor. Therefore his average download speed will come from optimistic unchoking.

For each altruistic peer in the torrent, there are 101 - self - 4 ·(well connected peers) = 96 peers to choose from randomly at any time. So the probability Bob is someone's optimistically unchoked random peer, $P(RP) = 1/96$. There are also 101-bob=100 altruistic peers in the torrent. Each altruistic peer always has 5 connections sharing a 10Mbps upload speed, 10Mbps/5 = 2Mbps upload to each connection. That leaves our final formula at

$$100(2\text{Mbps} \cdot \frac{1}{96}) = 2.08\bar{3}\text{Mbps}$$

B

Question

What will be the average download speed of each of the other peers?

Answer

Now we have also made the assumption that all uploading and downloading is useful. That means we can assume full output from the altruistic nodes. Each altruistic node outputs 10Mbps and there are 100 of them, so thats 1000Mpbs. We know on average, Bob will recieve 2.89Mbps. That leaves (1000-2.89) Mbps left for the other nodes. Since they are all altristic, we consider them semetric and therefore each get equal download bandwidth. Meaning each remaining node recieves on average

$$\frac{1000 - 100(2 \cdot \frac{1}{96})}{100} = \frac{479}{48} = 9.9791\bar{6}\text{Mbps}$$

Which is hilariously what you also get for

$$\frac{\binom{98}{3} \cdot \binom{96}{1} + \binom{98}{4}}{\binom{99}{4} \cdot \binom{96}{1}}$$

Which is funny because that formula assumes all configurations of a node are equally likely.

C

Question

Suppose that Bob runs a second client that pretends to be a separate peer, who also becomes a free rider. How fast can Bob download data now?

Answer

well, Bob is now has two peers so using the same logic as in A, that puts his download capacity at

$$100(2\text{Mbps} \cdot \frac{2}{97}) = 4\frac{12}{97} \approx 4.123\text{Mbps}$$

D

Question

Suppose Bob switches his two clients to the regular BitTorrent code and they both start uploading as well. What kind of aggregate download performance can he expect then?

Answer

All of the nodes are symmetric, this means on average they will upload the same ammount they download. This means that Bob's two clients each download 10 Mbps. That means his aggregate download is 20Mbps.

2

A

Question

Explain how a web server can carry on many HTTP connections simultaneously, even though all of the packets sent to it in all the sessions are all addressed to its TCP port 80. That is, how does the right packet get delivered to the right socket?

Answer

The packets all get sent to the right connection because a TCP connection is defined by a 4-tuple consisting of source IP, port, client IP, port. So because the clients will be connecting from unique source IP/port TCP can route the packets to the correct TCP connection.

B

Question

Are there any limitations on these simultaneous connections, such as, "a single host can only have one HTTP connection with this server at a time"? If yes, describe the limitations, if no, explain why there are no such limitations.

Answer

As defined by TCP, there is a limit if a client from every IP on every port tried to connect. For the same reason in A that source IP/port is enough for TCP to uniquely route packets to connections. In reality the limit would be on compute, buffer size, operating allowances for simultaneous connection requests.

3

Suppose within your web browser you click on a link to obtain a web page. The round-trip time to the server is 150 ms and your download speed is 2 Mbps.

A

Question

How long will it take you to download the HTML of the web page, which is 10 KB in size, from the time that you click on the link? (Assume that the necessary DNS infor- mation has been cached locally.) Ignore the sizes of the HTTP request and response headers for all parts in this question.

Answer

It will take 3/2RTT to establish a TCP connection and make the HTTP request, and 1/2RTT+FileSize/Bandwidth to get the page back.

$$2 \cdot 150\text{ms} + \frac{10\text{KB}}{2\text{Mbps}}$$

$$300\text{ms} + \frac{80\text{b}}{2\text{Kbps}}$$

$$300\text{ms} + 40\text{ms}$$

$$= 340\text{ms}$$

B

Question

Suppose the HTML file references 8 images, each of which is 100KB in size. How long will it take you to download the entire web page (starting from the click) using non-persistent HTTP connections?

Answer

It takes 1RTT to half establish a TCP connection and 1RTT + FileSize/Bandwidth to get each file since neither pipelining nor persistent connection is in place.

$$2 \cdot 9 \cdot \text{RTT} + \frac{\sum \text{file size}}{\text{Bandwidth}}$$

$$18 \cdot 150\text{ms} + \frac{10\text{KB} + 8 \cdot 100\text{KB}}{2\text{Mbps}}$$

$$2700\text{ms} + \frac{\text{ms} \cdot 8 \cdot (810)}{2}$$

$$2700\text{ms} + 3240\text{ms}$$

$$= 5940\text{ms}$$

C

Question

What if you are using persistent connections (without pipelining)?

Answer

Now each additional file only takes 1RTT+FileSize/Bandwidth to get recieved. So we just subtract (files-1)*RTT from the previous answer

$$5940 - 8 \cdot 150 = 5940 - 1200 = 4740\text{ms}$$

D

Question

What about if pipelining is also used on the persistent connection?

Answer

Now We pay only RTT for a file after the first, and FileSize/Bandwith penalty for each additional file, so we subtract (files-2)*RTT from the last answer.

$$4740 - 1050 = 3690\text{ms}$$

E

Question

Now consider a browser that uses non-persistent connections but will open up to 4 parallel connections to the server to download multiple objects simultaneously.

Answer

This means for the first file we pay 2RTT, and for every 4 additional files, we pay 2RTT.

$$1 \cdot 2\text{RTT} + 2 \cdot \text{RTT} + \sum \text{FileSizeBandwidth}$$

$$6 \cdot 150 + 3240\text{ms}$$

$$= 4140\text{ms}$$

4

Consider a 1 Gbps link that has a 100 ms round-trip time.

A

Question

How long will it take for the TCP connection to reach (approximately) full utilization using slow start? Assume one MSS is 1500 bytes.

Answer

Since this calculation is approximate, lets just assume the cwnd doubles every RTT. We know the connection can handle $1Gbps \cdot .1s / 1.5KBpS = 8.333KS$ That is, 8,333 segments per second. $log_2(8,333) \approx 13.024$ You reach full utilization, when you hit or cross the full utilization line. So, let's round up to 14 RTT or 1.4 seconds to reach full utilization.

B

Question

How long would it take for it to reach full utilization if AIMD were used, starting with a congestion window of 1 MSS?

Answer

Again, for this approximate calculation, we just assume with only additive increase that we send one more segment every RTT. That means 8334 (rounding up remember) - 1 * RTT is how long it will take. $8333 \cdot .1 = 833.3s$

5

5.1

Question

The TCP ACKs consume bandwidth. It would be better if the TCP receiver sends NACKs upon receiving out of order (or corrupted) packets.

Answer

FALSE: NACKs create the possibility of steady double transmissions which is much worse than the bandwidth consumed by TCP ACKs/without ACKs it would be impossible when to stop sending. Instead of NACKs, we send duplicate ACKS.

5.2

Question

Assume TCP is in the Slow Start phase, with SSThreshold as 32. At some time instant, the congestion window is 16. When the congestion window increases the next time, it will become 32.

Answer

FALSE: In a real system this is false, SS increases cwnd by 1MSS for every ACK. So next time it increases, it will increase to 17.

5.3

Question

A TCP socket creates an end to end connection between two devices.

Answer

FALSE: from the book "TCP is to extend IP's delivery service between two end systems to a delivery service between two processes." So TCP offers an end process to end process connection.

5.4

Question

The TCP sender has packets 20 to 30 in its congestion window (CW), all waiting for ACKs, when a time-out occurs. The sender will cut down CW to 1 and will gradually retransmit each of these packets.

Answer

FALSE: TCP will cut the cwnd in half and transmit based on that, eventually retransmitting all of these packets.

5.5

Question

Consider the case where a TCP sender transmits a file to a TCP receiver. A system ad- ministrator tells you that during this transfer, no timeout or dupACKs were recorded. In that case, the congestion window during the transfer should have never decreased (i.e., it should either remain same or increase).

Answer

TRUE: The finite state machine is pretty clear on this. Unless the file didn't transmit at all, cwnd must have either increased or stayed the same.

5.6

Question

The lower end of the TCP transmitters window is never greater than the lower end of the TCP receiver's window.

Answer

TRUE: assuming sender and receiver start at the same sequence number. The lower end of rwnd is the last byte received, and the lower end of the transsmision window the last byte sent but not ACK'd. Message is always received before it is ACK'd, so this is true.

5.7

Question

TCP can cope with any amount of losses, and hence, TCP should work without modifications on lossy wireless networks

Answer

TRUE: assuming lossy means some packets get through. From the book we know.

$$\text{average throughput of a connection} = \frac{1.22\text{MSS}}{\text{RTT}\sqrt{\text{L}}}$$

So TCP will work over any network that doesn't have total packet loss.

5.8

Question

The selective repeat receiver need not send an ACK if the received packet is less than its lower end of the (current) receive window.

Answer

FALSE: "In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged." Otherwise the sender will just keep sending packets for which the ACK got lost / the send window wouldn't move forward and progress would be halted.

6

Consider the Go-Back-N protocol with a send window size of N and a sequence number range of 4096. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that, the medium may drop packets but does not reorder messages.

A

Question

What are the possible sets of sequence number inside the sender's window at time t? Justify your answer.

Answer

The sender could have sent packets $[k-N, k-1]$ and is waiting for an ACK for each of them. Alternatively the sender could have gotten an ACK for $k-1$ and has sent $[k, k+N-1]$ and will definitely be waiting for their ACK because the reciever hasn't even seen k yet. Therefore the range of packets in the sender window is $[k-N, k+N-1]$.

B

Question

What are all possible values of the ACK field in the message currently propagating back to the sender at time t? Justify your answer.

Answer

The receiver has yet to see k, and GBN uses cumulative ACKs, so the return message can't have an ACK with $\geq k$. We know there has been an ACK for $k-N-1$, otherwise the sender wouldn't be able to send $k-1$. We also assumed packets don't get sent out of order, so if ACK $k-N-1$ was sent, then nothing before it will be in transit. This means the sequence numbers in the current return messages must be in the range $[k-N, k-1]$.

C

Question

With the Go-Back-N protocol, is it possible for the sender to receive an ACK for a packet that falls outside of its current window? Justify your answer with an example.

Answer

If we assume packets can be reordered, then yes. Imagine sender sends $k-N-2$ and $k-N-1$ and receives ACK $k-N-1$ back, but $k-N-2$ gets heavily delayed. Then the sender window moves to $[k-N, k-1]$, at which point ACK $k-N-2$ arrives and is not in the window.

7

One difficulty with the original TCP'S RTT estimator is the choice of an initial value. In the absence of any special knowledge of network conditions, the typical approach is to pick an arbitrary value, such as 3 seconds, and hope this will converge quickly to an accurate value. If this estimate is too small, TCP will perform unnecessary retransmissions. If it is too large, TCP will wait a long time before retransmitting if the first segment is lost. Also, the convergence might be slow.

A

Question

Choose $\alpha = 0.7$ and RTT-timeout$(0) = 1$ seconds, and assume all measured RTT values $= 0.5$ second with no packet loss. What is RTT-timeout(15)? Recall,

$$\text{RTT} - \text{timeout}(k+1) = \alpha \times \text{RTT-timeout}(k) + (1-\alpha) \times \text{RTT}(k+1)$$

Describe your solution approach (closed-form equation) AND provide the numerical result (approximate to 4th decimal place).

Answer

Since each EWMA is a linear combination of inputs, it can be put into a closed form. Let's abbreviate RTT-timeout as RTTT

We know

$$RTTT(0) = (\alpha)RTTT(0) + (1-\alpha)RTTT(0)$$

We then then unroll the formula

$$RTTT(k+1) = \alpha \cdot RTTT(k) + (1-\alpha) \cdot RTT(k+1)$$

$$= \alpha \cdot \Big(\alpha \cdot RTTT(k-1) + (1-\alpha) \cdot RTT(k)\Big) + (1-\alpha) \cdot RTT(k+1)$$

$$= \alpha \cdot \Big(\alpha \cdot \Big(\alpha \cdot RTTT(k-2) + (1-\alpha) \cdot RTT(k-1)\Big) + (1-\alpha) \cdot RTT(k)\Big) + (1-\alpha) \cdot RTT(k+1)$$

$$= \alpha^3 \text{RTTT}(k-2) + \alpha^2(1-\alpha)\text{RTT}(k-1) + \alpha(1-\alpha)\text{RTT(k)} + (1-\alpha)\text{RTT}(k+1)$$

The closed to this is

$$RTTT(k) = (\alpha)^k \cdot (\text{RTTT}(0)) + \sum_{i=1}^{k} (1-\alpha)\alpha^{k-i}\text{RTT}(k)$$

Plugging in for $k = 15$ gives 0.5024 seconds.

B

Question

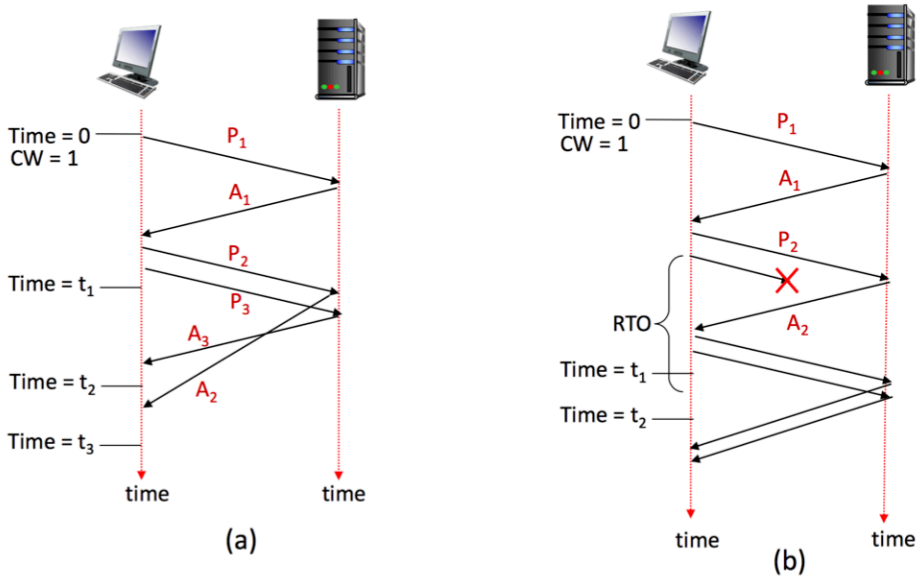
Using the same values as in above part, what happens if we use = 0.5 or = 0.9? Provide a numerical result for RTT-timeout(15) in both cases, then describe the effect of a larger or smaller on the RTT estimation procedure.

Answer

Plugging in $\alpha = .5$ gives .5000 seconds, and plugging in $\alpha = .9$ gives .6029 seconds. It is clear that the smaller alpha inputs will make convergence happen faster.

8

For this question, you will refer to Figure 1. Assume slow start, unless otherwise mentioned. Copy the timelines in your answer sheet and work on them.



8.1

Question

In Figure 1(a), what are the values of CW at times t1, t2, t3? How should the TCP transmitter react after receiving A3 and A2.

Answer

According to the FSM, since A_1, A_2, A_3 are all new acknowledgements, cwnd will increase by 1 after seeing each. Therefore at time t_1, cwnd=2. At t_2, cwnd=3. t_3, cwnd=4

8.2

Question

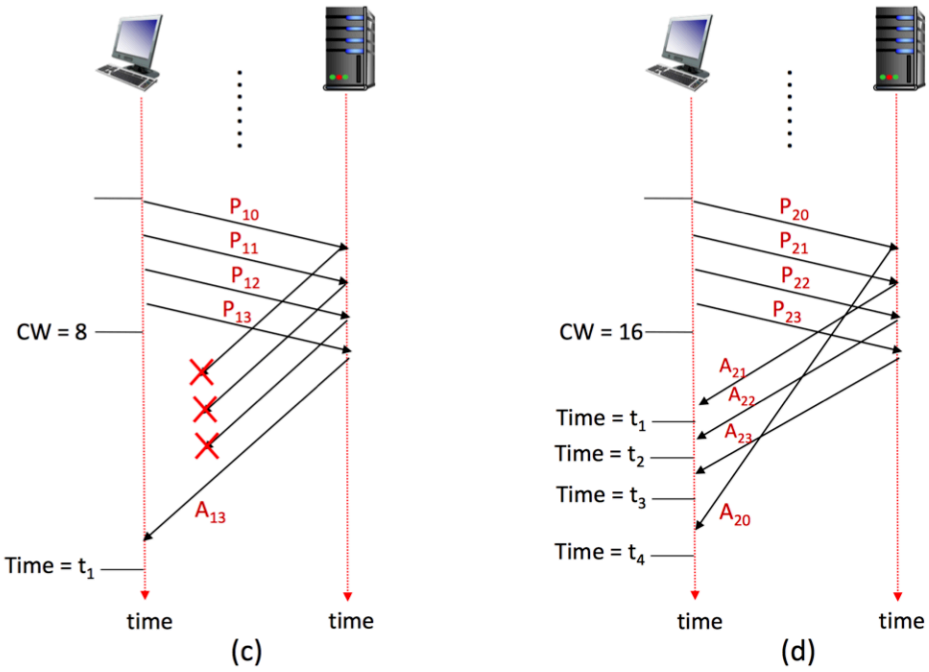
In Figure 1(b), assume TCP starts from CW=1. What should CW be at times t1 and t2? How should the TCP transmitter react after the timeout? How should the TCP transmitter react when each of the the last two ACKs (shown in the figure) arrive?

Answer

At time t_1 cwnd=3 because it sees two ACKs and starts at 1.

The transmitter should react to the timeout by setting ssthresh to $3/2 \cdot$ MSS. Setting cwnd to 1, settign dupACKcount to 0, and retransmitting P_3.

The transmitter should react to the P_4 by incrementing cwnd to 2 MSS, moving to Congestion Avoidance mode. Then upon seeing P_5, it should increase cwnd to 2.5 MSS.



8.3

Question

In Figure 1(c), what should CW be at time t1? How should the TCP transmitter react upon receiving A13?

Answer

Since it is in slow start and it sees a new ACK, it will increment cwnd.

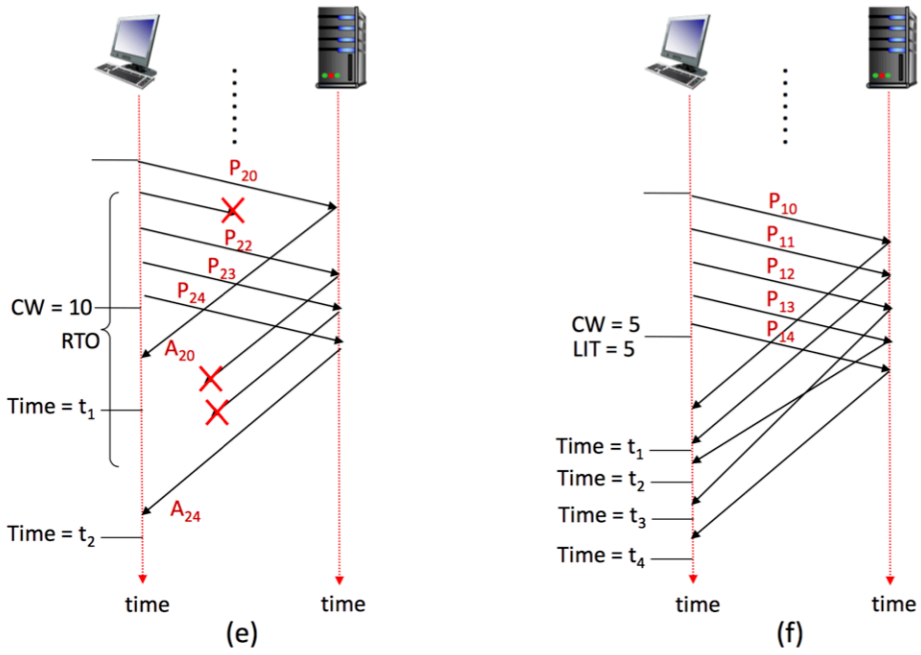
8.4

Question

In Figure 1(d), what should the CW be at times t1, t2, t3, and t4? How should the TCP transmitter react upon receiving each of the four ACKs?

Answer

Each ACK is new, so TCP will increment cwnd. Therefore
at t_1 cwnd=17
at t_2 cwnd=18
at t_3 cwnd=19
at t_4 cwnd=20



8.5

Question

In Figure 1(e), say that the first ACK that is shown to arrive at the TCP transmitter is A20. What should the CW be at times t1 and t2? How should the TCP transmitter react upon receiving A20, when the timeout occurs, and upon receiving the last shown ACK (just before t2)?

Answer

Since we are in Slow Start when A_20 arrives, we increment cwnd, so that

$$\text{at } t_1, \text{ cwnd} = 11$$

Then the timout occurs, so cwnd gets reset to 1. When A_24 arrives, then since it is a new ACK, we increment cwnd. Therefore

$$\text{at } t_2, \text{ cwnd} = 2$$

8.6

Question

In Figure 1(f), LIT is the acronym for linear increase threshold (AIMD). What should the values of CW be at times t1, t2, t3, and t4? How should the TCP transmitter react upon receiving each of the ACKs?

Answer

With AIMD, cwnd increases by

$$\frac{\text{MSS}}{\frac{\text{cwnd}}{\text{MSS}}}$$

Every time it sees a new ACK. So at

$$t_1 \text{ cwnd} = 5.4\text{MSS}$$

$$t_2 \text{ cwnd} = 5.6\text{MSS}$$

$$t_3 \text{ cwnd} = 5.8\text{MSS}$$

$$t_4 \text{ cwnd} = 6\text{MSS}$$

According to the FSM, the transmitter should only increment cwnd upon recieving the ACKs.

CS 473 HW2

Charles Swarts

February 2018

1

An intersection is connected via road to its parent intersection, its grandparent intersection, and all of its children intersections. Therefore an intersection can only not be lit if all the other intersections it is connected to are unlit.

The recursion:

$$r(g,p,u) = \min \begin{cases} \text{cost}(u) + \sum_{c \in C} r(p,1,c) & \textbf{where C} \text{ is the set of } u\text{'s children} \\ 0 \qquad\quad + \sum_{c \in C} r(0,0,c) & \textbf{if}(g = p = 1) \textbf{ where C} \text{ is the set of } u\text{'s children} \end{cases}$$

Note: The recursion covers the base case of an intersection having no children intersections.

The recursion would be called on the root intersection, r, with $g = 1, p = 1$ because it has no parent road or grandparent road to light.

Proof of correctness: By induction on tree level, l

Base case: we have a leaf node with at least one of the parent or grandparent node unlit.
Then this node must be lit. This cost is covered in the first branch of the recurrence. The second branch will not activate, therefore the cost of lighting this intersection will be the value returned. This is correct.

Base case: we have a leaf node with both parents lit.
This means the node can be unlit, which accrues no cost and is accounted for in the second branch. Since the cost of lighting an intersection is strictly positive, the first branch of the recursion will return a positive value. Therefore the result from the second branch, 0, will be chosen by the minimum function and returned as the correct value for this case.

Inductive hypothesis: for a node at level l, our recurrence function returns the optimal value for lighting this node's children trees, at level $l-1$, given whether the node at l is lit and whether the parent node at $l+1$ is lit.

Inductive step:
Case 1: one of the parent or grandparent node is unlit.
This node will look at the parent and grandparent node and if either one of them is unlit, it will become lit. It will return this cost, and the optimal cost of lighting the children nodes under such conditions. Thus it returns the optimal value of lighting the intersection tree starting at level l.
Case 2: both of the grandparent nodes are lit.
According to the if statement, both branches of the recursion are evaluated. Thus it compares the optimal cost of lighting this intersection and having all intersection subtrees optimally lit in that scenario, vs the cost of not having this intersection lit and having all intersection subtrees optimally lit in this scenario. Thus it will return the optimal scenario for either way.

Thus evaluating the recurrence at any level of the tree will return the optimal value of lighting that subtree with the initial starting conditions. □.

This can then be turned into an iterative algorithm.

```
given a post-order traversal Traversal = n1,n2,n3...nn

table[2][2][n];

for node in Traversal:

    sum = cost(node);
    for children of node:
        sum += table[0][1][children];
    table[0][0][node] = sum;

    sum = cost(node);
    for children of node:
        sum += table[1][1][children];
    table[0][1][node] = sum;
    table[1][1][node] = sum;

    sum = 0;
    for children of node:
        sum += table[0][0][children];
    table[1][1][node] = min( sum , table[1][1][node] );

return table[1][1][nn];
```

Runtime: During the body of the outer loop, each node looks at its children 3 times. That means there are $3n$ lookups in the table. For each lookup, the result is added to an accumulator in constant time. All other statements in the outer loop together happen in constant time. Thus the algorithm is $O(3n) = O(n)$.

2

a.

There exists a version of the numbers game where I win iff I do not use the greedy strategy.

Proof: This means there is a game where following the greedy strategy will lead me to lose, where if I were to follow a different strategy, I would win. Consider the following such game.

$$5, 99, 1, 1$$

Greedy strategy: I choose 5, my opponent chooses 99, I choose 1, they choose 1. I lose.

Alternate strategy: I choose 1, my opponent chooses 5, I choose 99, they choose 1. I win. □

b.

The algorithm to maximize the winnings in the game is to play every game and measure the difference in score. This can be done with the following recurrence on sequence X:

$$r(\text{start}, \text{end}) = \begin{cases} X_{\text{start}} & \textbf{if } \text{start} = \text{end} \\ \max(X_{\text{start}}, X_{\text{end}}) & \textbf{if } \text{start} + 1 = \text{end} \\ \max \begin{cases} X_{\text{start}} + r(\text{start}+2, \text{end}) & \textbf{if } X_{\text{start}+1} > X_{\text{end}} \\ X_{\text{start}} + r(\text{start}+1, \text{end}-1) & \textbf{if } X_{\text{start}+1} \leq X_{\text{end}} \\ X_{\text{end}} + r(\text{start}, \text{end}-2) & \textbf{if } X_{\text{start}} \leq X_{\text{end}-1} \\ X_{\text{end}} + r(\text{start}+1, \text{end}-1) & \textbf{if } X_{\text{start}+1} > X_{\text{end}} \end{cases} & \text{otherwise} \end{cases}$$

Proof of correctness of recurrence.

Proof by induction on l, the length of the sequence.

Base case: $l = 1$. That means the start of the sequence and the end of the sequence are the same. According to the recurrence, this element's value is the total possible sum. Since it is your turn, this result is easily seen to be correct. Base case: $l = 2$. That means the start and the end of the sequence are 1 apart. Our recurrence says the largest sum is the maximum value of the two remaining elements. Since it is your turn, this result is easily seen to be correct.

Inductive hypothesis: we assume the recurrence gives the correct sum for all subsequences of length $k - 2$.

Inductive step: given a sequence of length $k > 2$ then the there are only two ways proceed, either you choose the left or the right element. Then your opponent chooses the left or the right element greedily. Depending on which element they pick, you will be left with trying to maximize your sum in a smaller game over a subsequence of length $k-2$. Therefore the maximum sum will be the element you picked + the maximum sum you could get from a game over the resulting subset. Both of these are considered in the inner branching, and the maximum is returned, therefore the recursion will return the maximum sum possible to get from a sequence of length k.

Therefore by induction, our recursion will correctly give the maximum sum possible to achieve against a greedy opponent in a game of a sequence of any size l. □

This recursion can be converted to the following iterative algorithm.

```
number[] Sequence;

table [n][n];

// First base case;
for i in 0 to n-1:
    table[i][i] = Sequence[i];

// Second base case;
for i in 0 to n-2:
    table[i][i+1] = abs(Sequence[i] - Sequence[i-1])

for length in 2 to n-1:
    for start in 0 to n-1-length:
        end = start + length;

        takeLeft  = 0;
        takeRight = 0;

        if( Sequence[start+1] > Sequence[end] ){
            takeLeft  = Sequence[start] - Sequence[start+1];
            takeLeft += table[start+2][end]                ;
        }
        else if( Sequence[start+1] < Sequence[end]){
            takeLeft  = Sequence[start] - Sequence[end]    ;
            takeLeft += table[start+1][end-1]              ;
        }

        if( Sequence[start] < Sequence[end-1] ){
            takeRight  = Sequence[end] - Sequence[end-1]   ;
            takeRight += table[start][end-2]               ;
        }
        else if( Sequence[start] > Sequence[end-1] ){
            takeRight  = Sequence[end] - Sequence[start]   ;
            takeRight += table[start+1][end-1]             ;
        }
        table[start][end] = max(takeLeft,takeRight);

return table[0][n-1];
```

Runtime: The outer loop loops $n-3$ times. The inner loop loops n times. The body of the loop runs in constant time. thus the algorithm runs in $O((n-3)*n) = O(n^2)$.

3

This problem can be solved in polynomial time by dynamifying this recurrence.

$$r(\#s_1, \#s_2, \#s_3, \text{bin\#}, \text{usage}, \text{maxL}) =$$

$$\min \begin{cases} \text{maxL} & \textbf{if } \#s_1 = \#s_2 = \#s_3 = 0 \\ r(\#s_1, \#s_2, \#s_3, \text{bin\#} + 1, \text{cap}(\text{bin\#} + 1), \text{maxL}) & \textbf{if } \text{bin\#} < m - 1 \\ r(\#s_1 - 1, \#s_2, \#s_3, \text{bin\#}, \text{usage'}, \max(\text{maxL}, \text{usage'})) & \textbf{if } \#s_1 > 0, \text{usage}' < \text{cap}(\text{bin\#}) \textbf{ where } \text{usage'} = \text{usage} + s_1 \\ r(\#s_1, \#s_2 - 1, \#s_3, \text{bin\#}, \text{usage'}, \max(\text{maxL}, \text{usage'})) & \textbf{if } \#s_2 > 0, \text{usage}' < \text{cap}(\text{bin\#}) \textbf{ where } \text{usage'} = \text{usage} + s_2 \\ r(\#s_1, \#s_2, \#s_3 - 1, \text{bin\#}, \text{usage'}, \max(\text{maxL}, \text{usage'})) & \textbf{if } \#s_3 > 0, \text{usage}' < \text{cap}(\text{bin\#}) \textbf{ where } \text{usage'} = \text{usage} + s_3 \\ \infty & \textbf{if no other statements are true} \end{cases}$$

Argument for correctness:

This recurrence will attempt every configuration of putting the items in the buckets and will pick the best one.

Runtime: we have three variables to hold the number of items of each of the three sizes as parameters to the recurrence. There are n items so in the worst case their they could require $(n/3)^3$ loops to iterate for each of those loops. This is by the principle of maximum hypervolume of a hypercube with positive finite sides. (Ruta said in office hours that I could use this principle). Then there will be a loop for each of the bins. There are m bins. Then we have to loop over the range from 0 to the maximum usage. Since maxL is a the value stored in the table, we won't loop over it. Thus the total running time of the algorithm would be $O((n/3)^3 \cdot m \cdot \text{maximum capacity}) = O(n^3 \cdot m \cdot \text{maximum capacity})$.

Homework 8

Charles Swarts
Swarts2

March 2016

1 8.14

1.1 question

A Queue

A bus is supposed to arrive at a bus stop every hour for 10 hours each day. The number of people who arrive to queue at the bus stop each hour has a Poisson distribution, with intensity 4. If the bus stops, everyone gets on the bus and the number of people in the queue becomes zero. However, with probability 0.1 the bus driver decides not to stop, in which case people decide to wait. If the queue is ever longer than 15, the waiting passengers will riot (and then immediately get dragged off by the police, so the queue length goes down to zero). What is the expected time between riots?

Solution: I'm not sure whether one could come up with a closed form solution to this problem. A simulation is completely straightforward to write. I get a mean time of 441 hours between riots, with a standard deviation of 391. It's interesting to play around with the parameters of this problem; a less conscientious bus driver, or a higher intensity arrival distribution, lead to much more regular riots.

1.2 raw simulation code

```
    days_Simulated <- 1000000

intensity <- 4
max_Line_Length <- 15
prob_busDriver_flakes <- .1
timelength <- 1

#The strategy is to make a sample space for
#arrival rate of the people and the arrival
#of the bus driver. In simulating the line,
#I will make a sample space that includes
#probabilities for 0:max_Line_Length,
#(>max_line_length), because if that many
#people show up, the line automatically riots.
```

```
sampleSpace <- 0:(max_Line_Length+1)
sampleSpaceProbs <- c()
for(i in 0:max_Line_Length){
  sampleSpaceProbs <- c(sampleSpaceProbs,((intensity*timelength)^i)/(factorial(i))*exp((-1)*intensity*
}
sampleSpaceProbs <- c(sampleSpaceProbs, 1-sum(sampleSpaceProbs))


arrivals <- sample(sampleSpace,10*days_Simulated,replace=TRUE,prob=sampleSpaceProbs)


busSpace <- c(0,1)
busSpaceProbability <- c(.1,.9)


busComes <- sample(busSpace,10*days_Simulated,replace=TRUE,prob=busSpaceProbability)

peopleInLine <- 0
#riots is a list of time-lengths between riots.
riots <- c()
counter <- 1
#The riot Counter is the number of hours between riots.
riotCounter <- 0
for(i in 1:(days_Simulated*10)){
  riotCounter <- riotCounter+1
  peopleInLine <- peopleInLine+arrivals[i]

  if(peopleInLine>max_Line_Length){
    riots <- c(riots,riotCounter)
    riotCounter <- 0
    peopleInLine <- 0
    }
  if(busComes[i]==1){peopleInLine <- 0}

  #This code follows from the fact that the bus
  #runs 10 times/day and at night, people who
  #are waiting go home.
  if((counter%%10)==0) {peopleInLine <- 0}
  counter <- counter+1;
}

mean(riots)
sd(riots)
```

1.3 Answers

Simulating 1000000 days,

I got the mean time between riots was 491.7139 hours. And the Standard Deviation between time between riots was 492.683.

2 8.15

2.1 question

Inventory

A store needs to control its stock of an item. It can order stocks on Fri- day evenings, which will be delivered on Monday mornings. The store is old- fashioned, and open only on weekdays. On each weekday, a random number of customers comes in to buy the item. This number has a Poisson distribution, with intensity 4. If the item is present, the customer buys it, and the store makes 100; $otherwise, thecustomerleaves.Achevening atclosing, thestoreloses10$ for each unsold item on its shelves. The store's supplier insists that it order a fixed number k of items (i.e. the store must order k items each week). The store opens on a Monday with 20 items on the shelf. What k should the store use to maximise profits?

Solution: I'm not sure whether one could come up with a closed form solution to this problem, either. A simulation is completely straightforward to write. To choose k, you run the simulation with different k values to see what hap- pens. I computed accumulated profits over 100 weeks for different k values, then ran the simulation five times to see which k was predicted. Results were 21, 19, 23, 20, 21. I'd choose 21 based on this information.

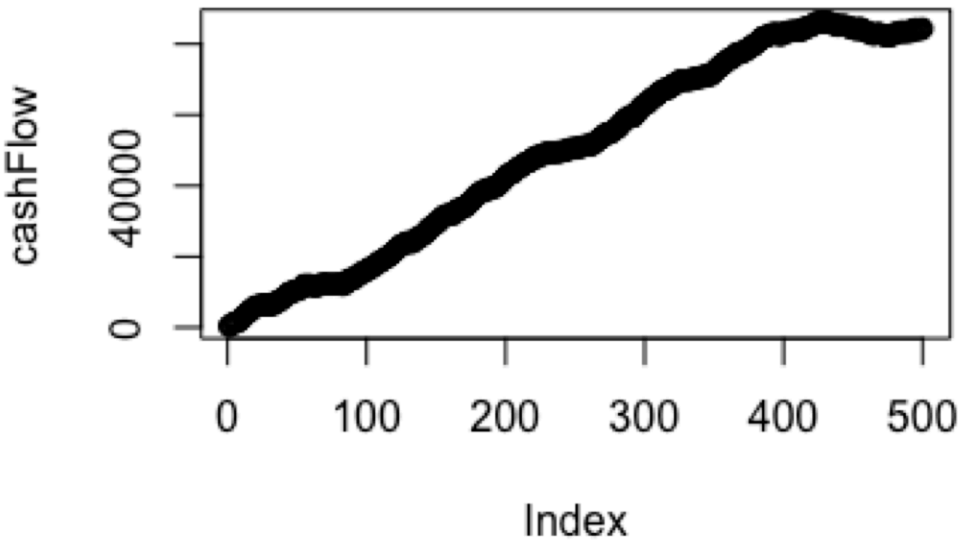
2.2 raw simulation code

```
    weeks_Simulated <- 100

original_stock <- 20
intensity <- 4
k <- 19

#My strategy for calculating the probabilities
#of the sample space for the number of
#customers who walk through the is to caluculate
#up to 2 * Expected[people who come in during the week]
#because I think it would be unlikely that that
#number of people walks into the store.
#The expected # of people for a poisson dist
#is intensity * time or 5*4 which is 20
#Therefore I will be calculating up to 40.

#numPeople()

prob_Num_People <- c()

for(i in 0:(intensity*5*2)){
prob_Num_People <- c(prob_Num_People,((intensity)^i)/(factorial(i))*exp((-1)*intensity))
}
prob_Num_People <- c(prob_Num_People,1-sum(prob_Num_People))

Num_People <- 0:(intensity*5*2+1)

people_by_Day <- sample(Num_People,weeks_Simulated*5,replace=TRUE,prob=prob_Num_People)
```

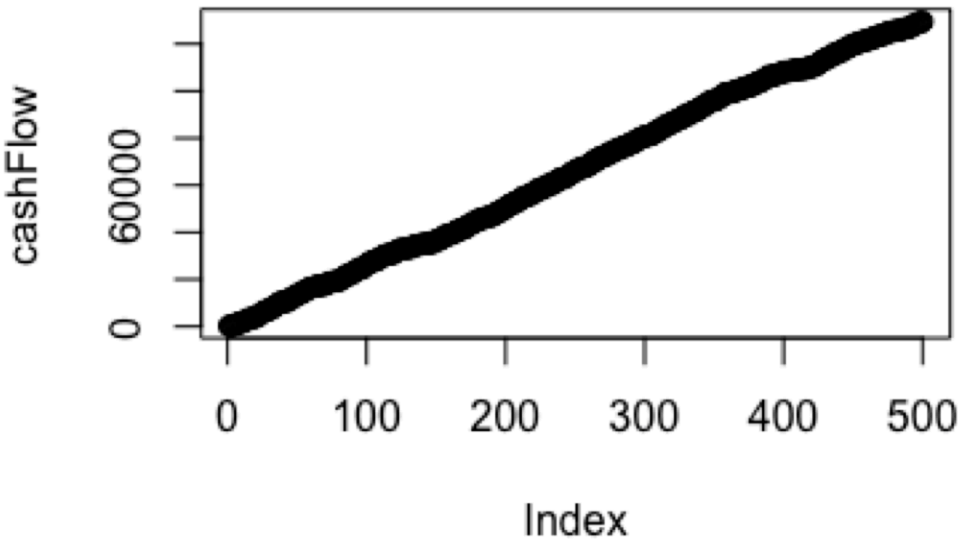
```
#people_by_Day

cashFlow <- c(0)

for(i in 1:(weeks_Simulated*5)){
  if(i!=1){
  if(people_by_Day[i]>original_stock)
  {
    cashFlow[i] <- (cashFlow[i-1]+100*original_stock)
    original_stock <- 0
  }
  else{
    cashFlow[i] <- (cashFlow[i-1]+100*people_by_Day[i])
    original_stock <- original_stock-people_by_Day[i]
  }
  }
  else{
    if(people_by_Day[i]>original_stock)
    {
      cashFlow[i] <- (100*original_stock)
      original_stock <- 0
    }
    else{
      cashFlow[i] <- (100*people_by_Day[i])
      original_stock <- original_stock-people_by_Day[i]
    }
  }
  cashFlow[i] <- (cashFlow[i]-10*original_stock)
  if(i%%5==0){original_stock <- original_stock+k}
}
plot(cashFlow)
people_by_Day
cashFlow[length(cashFlow)]
```

2.3 Answers

I did not throw product out at the end of the week, so when I plotted what it looks like when you order 20, I got plots like this:



where I only made $84400 after 100 weeks, and plots like this when I ordered 19 each week



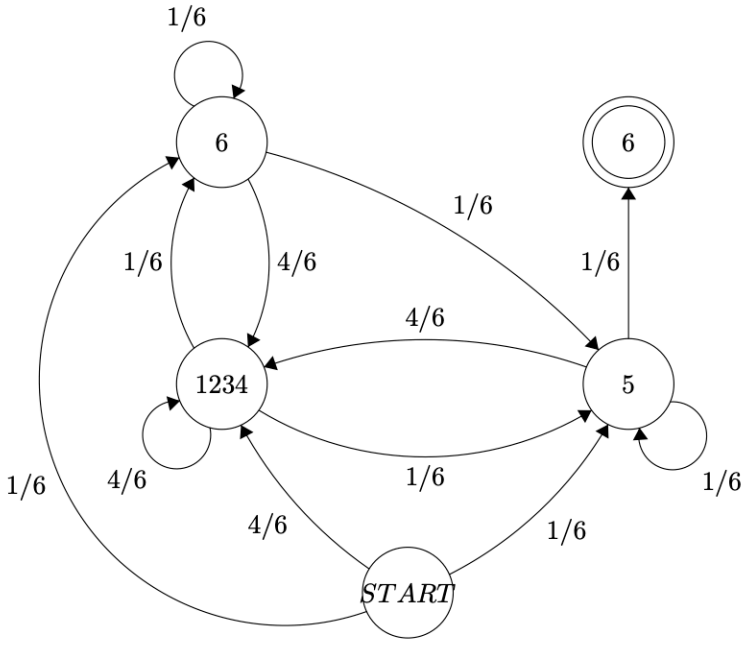
and made $129550
therefore I believe 19 to be the best amount to order each week.

3 8.2

3.1 question

8.2. Multiple die rolls: You roll a fair die until you see a 5, then a 6; after that, you stop. Write P(N) for the probability that you roll the die N times.
(a) What is $P(1)$?
(b) Show that $P(2) = (1/36)$.
(c) Draw a finite state machine encoding all the sequences of die rolls that you could encounter. Don't write the events on the edges; instead, write their probabilities. There are 5 ways not to get a 5, but only one probability, so this simplifies the drawing.
(d) Show that $P(3) = (1/36)$.
(e) Now use your finite state machine to argue that $P(N) = (5/6)P(N1) + (25/36)P(N2)$.

3.2 answer



(a). $P(1)$ according to the diagram is 0 because there is no walk of length one from START to 6. Alternative reasoning is that you can't get a five then a 6 in one role.

(b). $P(2)$ according to the diagram is 1/36 because there is only one walk of length 2 from START to 6 and one of the edges has a $P(E) = 1/6$ and the other has $P(E) = 1/6$ so to traverse both is $P(E) \cap P(E)$ which is P(E)*P(E) since they are independent, and $1/6 * 1/6 = 1/36$ Alternative reasoning is that you would have to get a 5 and then a 6 to end the game and the probability of getting a 5 then a 6 is $1/6 * 1/6 = 1/36$
(c) See diagram above
(d) $P(3)$ according to the diagram is 1/36 because the three walks that go from start to three are

$$START \rightarrow 5 \rightarrow 5 \rightarrow 6 = 1/6 * 1/6 * 1/6 = 1/216$$

$$START \to 6 \to 5 \to 6 = 1/6 * 1/6 * 1/6 = 1/216$$

$$START \to 1234 \to 5 \to 6 = 1/6 * 1/6 * 1/6 = 4/216$$

adding all the possibilities together makes $6/216 = 1/36$ This follows the general formula

$$P(n) = \sum_j P((n-1)|j)$$

Alternative reasoning is that in order to roll three before the game is over, you would have to get a 5 and a 6 on the second and third rolls respectively. It doesn't really matter the outcome of the first state, so you can say it could be "anything," and "anything" has a probability of 1. $P(2)$ has a probability of $1/36$ so $1 * 1/36 = 1/36$ which also begets the answer.

4 8.3

4.1 question

8.3. More complicated multiple coin flips: You flip a fair coin until you see either HTH orTHT, and then you stop. We will compute a recurrence relation for $P(N)$.

(b) Write $\sum_N$ for some string of length N accepted by this finite state machine. Use this finite state machine to argue that $\sum_N$ has one of four forms:

$$\mathbf{1.}TT \sum_{N-2}$$

$$\mathbf{2.}HH \sum_{N-2}$$

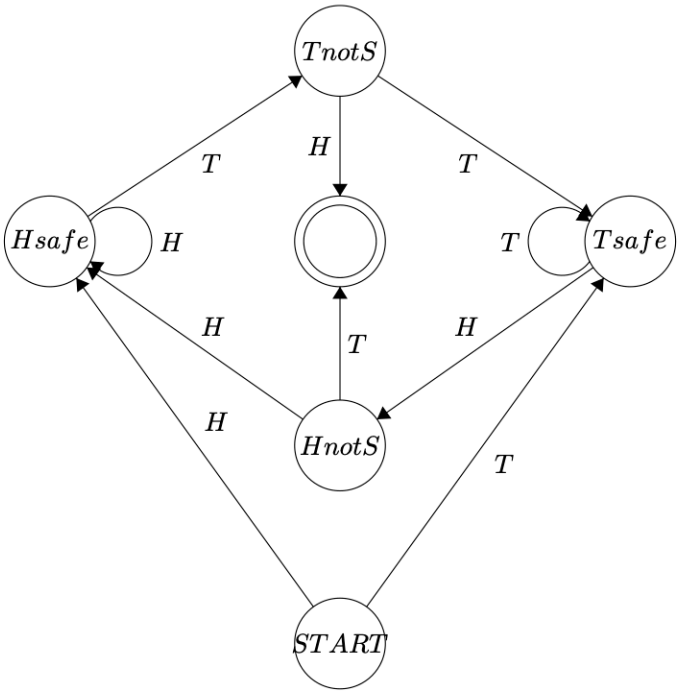
$$\mathbf{3.}THH \sum_{N-3}$$

$$\mathbf{4.}HTT \sum_{N-3}$$

(c) Now use this argument to show that

$$P(N) = (1/2)P(N-2) + (1/4)P(N-3).$$

4.2 answer



(b) So the triggers for the end state of this finite state machine are THT and HTH. And also I define the word "safe" to mean: can't cause the end state in one move. At the beginning, after the first coin flip you are safe, because if you draw a heads or a tail on the second flip, you still haven't triggered the end state.

Let's start with base cases, either the first is H or T, if it's H then, according to the state machine, the next sequence has to be either H, or TT for the sequence to be safe again. If you start out with T, then the next sequence has to be either T or HH before you know you are safe again.

Being safe means that you have recreated the conditions at the start for the last flip in the running sequence. IE you get the HTT scenario, it has now become like if Tails was the first outcome you got so that the two sequences availible to you are T and HH

It must be noted here that the $\sum_N$ implies that it is based off of what came before it. This is so you can never get the case $TTHTTH \sum N - 6$

Also, for each flip, the rest of the sequence decreases by one.
So to conclude all of this, there are four starting sequences that result,

$$1 : HH \sum_{N-2}$$

$$2 : TT \sum_{N-2}$$

$$3 : HTT \sum_{N-3}$$

$$4 : THH \sum_{n-3}$$

(c) So if we have the previous as the possible outcomes, we can break them down into their component parts which are:

1: The initial outcome. The initial outcome is boring because it is always safe, so the probability of it being correct is 1.

2:The second component which gives a 1/2 probability that the sequence is determined by the $P(N-2)$ or that it will go to the alternative.

3: In the alternative, we once again have a 1/2 chance that the variable lines up and the Probability is based on $P(N-3)$

That gives an overall probability for $P(N)$

$$P(N) = 1 * \frac{1}{2} * P(N-2) + 1 * \frac{1}{2} * \frac{1}{2} * P(N-3) = \frac{1}{2}P(N-2) + \frac{1}{4}P(N-3)$$

5 9.2

5.1 question

9.2. Fitting a Poisson Distribution: You count the number of times that the annoying "MacSweeper" popup window appears per hour when you surf the web. You wish to model these counts with a Poisson distribution. On day 1, you surf for 4 hours, and see counts of 3, 1, 4, 2 (in hours 1 through 4 respectively). On day 2, you surf for 3 hours, and observe counts of 2, 1, 2. On day 3, you surf for 5 hours, and observe counts of 3, 2, 2, 1, 4. On day 4, you surf for 6 hours, but keep only the count for all six hours, which is 13. You wish to model the intensity in counts per hour.
(a) What is the maximum likelihood estimate of the intensity for each of days 1, 2, and 3 separately?
(b) What is the maximum likelihood estimate of the intensity for day 4?
(c) What is the maximum likelihood estimate of the intensity for all days taken together?

5.2 answer

After reading the section, I found the maximum likelihood formula for the Poisson distribution is

$$\hat{\theta} = \frac{\sum_i n_i}{N}$$

(a)

$$\text{First day:} \frac{\sum_i n_i}{N} = \frac{(3+1+4+2)}{4} = \frac{10}{4} = 2.5 = \hat{\theta}$$

$$\text{Second day:} \frac{\sum_i n_i}{N} = \frac{(2+1+2)}{3} = \frac{5}{3} = 1.6\bar{6} = \hat{\theta}$$

$$\text{Third day:} \frac{\sum_i n_i}{N} = \frac{(3+2+2+1+4)}{5} = \frac{12}{5} = 2.4 = \hat{\theta}$$

(b) Because we are given the total occurrences for the forth day, we can treat that as the sum of all the individual periods. And we know there were 6 periods because that's how many hours were surfed.

$$\text{First day:} \frac{\sum_i n_i}{N} = \frac{13}{6} = \frac{13}{6} = 2.16\bar{6} = \hat{\theta}$$

(c) The formula should still hold over different days because Poisson distribution models independent events.

$$\text{Combined days:} \frac{\sum_i n_i}{N} = \frac{10+5+12+13}{4+3+5+6} = \frac{30}{18} = 1.6\bar{6} = \hat{\theta}$$

6 9.4

6.1 question

9.4. Fitting a Binomial Model: You encounter a deck of Martian playing cards. There are 87 cards in the deck. You cannot read Martian, and so the meaning of the cards is mysterious. However, you notice that some cards are blue, and others are yellow.
(a) You shuffle the deck, and draw one card. It is yellow. What is the maxi- mum likelihood estimate of the fraction of blue cards in the deck?
(b) You repeat the previous exercise 10 times, replacing the card you drew each time before shuffling. You see 7 yellow and 3 blue cards in the deck. What is the maximum likelihood estimate of the fraction of blue cards in the deck?

6.2 answer

(a) So to solve this problem, we will use the ML formula for the binomial to find the probability of getting blue. The formula is $\hat{\theta} = k/N$ where k is the number of times that result appears and N is the number of trials. For this we did one trial, so $N = 1$ and $k = 0$ because we got no blue cards.

$$P(\text{blue card}) = \hat{\theta} = \frac{k}{N} = \frac{0}{1} = 0$$

So the fraction of blue cards in the deck is equal to the probability of getting a blue card, in this case. Therefore according to the maximum likelihood estimate, THERE ARE NO BLUE CARDS IN THE DECK.

(b) Using the same reasoning as a, we can simply plug into the formula the number of blue cards:3 out of the number of trials:10 to get

$$P(\text{blue card}) = \hat{\theta} = \frac{k}{N} = \frac{3}{10} = 0.3$$

$$(\text{Est: Fraction of blue cards}) = .3$$

Homework 12

Charles Swarts
swarts2@illinois.edu

April 2016

1 Problem 1

1.1 question

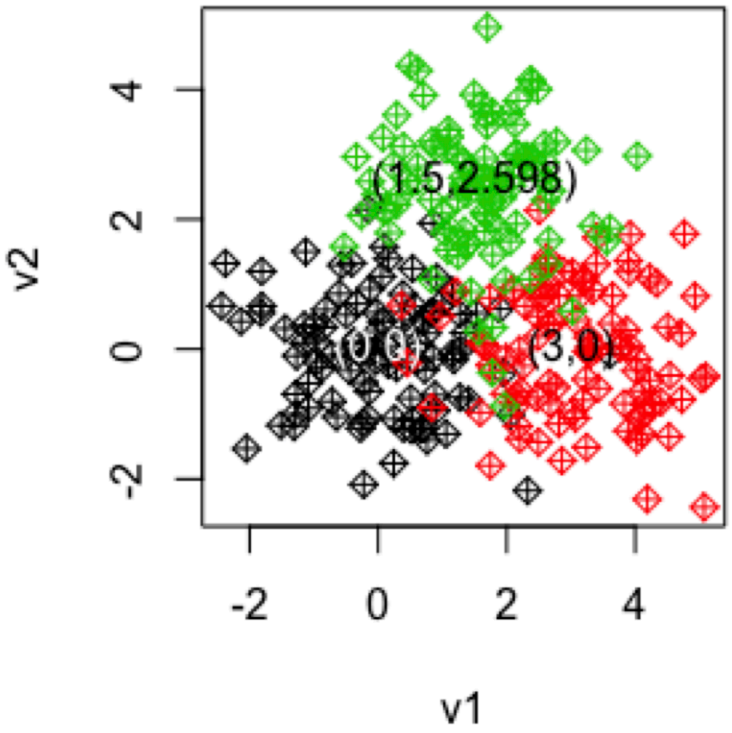
This time we're going to work on two different datasets.

Problem 1.

Generate three 2D Normal distributions with random mean vectors and unit variances. The Euclidean distance between the three random mean vectors shouldn't exceed 3. What that means is that you randomly define three 2D means, and your covariance matrices are all identity matrices. Draw 100 samples per distribution. Overall, you have 300 samples from three Gaussians. Pretend like that you don't know which sample belongs to which Gaussian.

(a) Scatterplot the samples. Use different colors or markers to distinguish samples from different distributions. Also, mark the positions of the original means. (b) Write your own code to do k-means clustering on this dataset. Show your estimated means on the scatterplot, and compare them to the original ones.

2 Answer



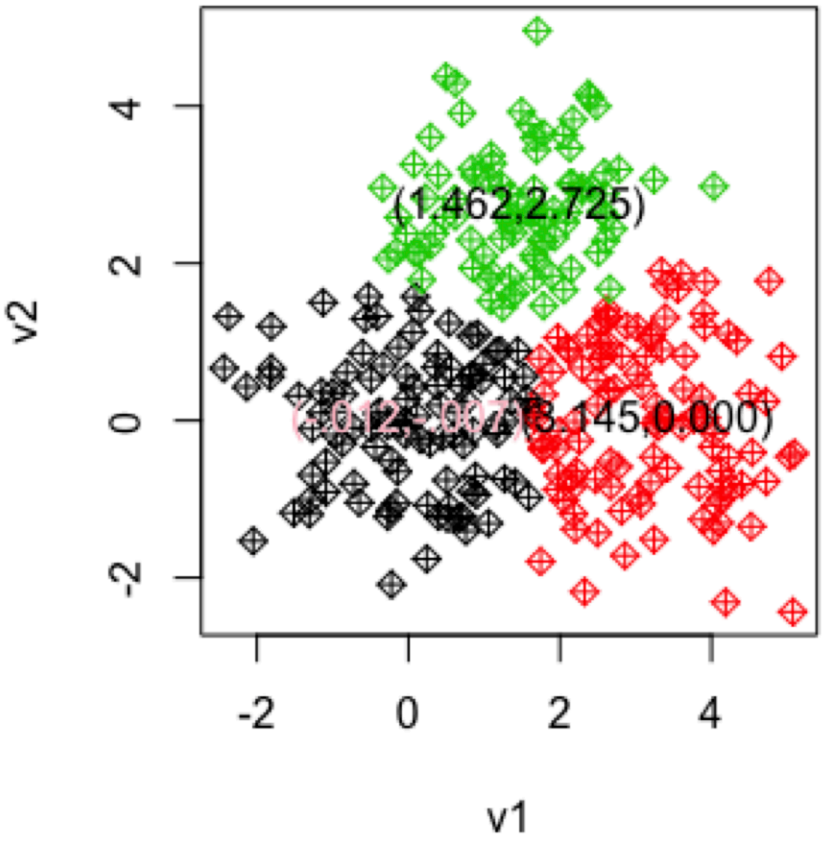
(a).

(b). My actual means were

1: $0, 0$
2: $3, 0$
3: $1.5, 2.598$
My estimated means were

1: $-0.012698883, -0.007750091$
2: $3.1458264815, 0.0007224473$
3: $1.462456, 2.725030$

They are pretty close to the mark in at least one dimension. 2 and 3 have one dimension that is off by .1.



2.1 Raw Code

```
    #For this question, I am picking the points (0,0)
#(3,0) and (1.5,2.598) each of which you will see
#have a euclidian distance of 3 from each other.

#Here the three 2D normal distributions are being
#drawn from.
v1 <- rnorm(100,0,1)
v2 <- rnorm(100,0,1)
v1 <- c(v1,rnorm(100,3,1))
v2 <- c(v2,rnorm(100,0,1))
v1 <- c(v1,rnorm(100,1.5,1))
v2 <- c(v2,rnorm(100,2.598,1))

#v3 keeps track of who's who.
```

```
v3<- factor(c(rep("x1",100),rep("x2",100),rep("x3",100)))

#Makes a dataframe to help keep track.
pointsFrame<- data.frame(cbind(v1,v2,v3))

#Here is the original plot before clustering.
plot(v2~v1,col=v3,pch=9)
text(0,0,"(0,0)",col="white")
text(3,0,"(3,0)",col=9)
text(1.5,2.598,"(1.5,2.598)",col=9)

#The points are here to keep track
#of if the algorithm has found the
#stable state.
oldPoints <- c(-1,-1,-1)

#newPoints first draws three random
#indices out of a hat.
newPoints <- sample(1:300,3,replace=FALSE)

#While not at a stable state.
while(newPoints[1]!=oldPoints[1]|newPoints[2]!=oldPoints[2]|newPoints[3]!=oldPoints[3])
{
  #This is how I keep track of the old points.
  oldPoints <- newPoints
  #Here is the space for the three clusters.
  clusterOne <- c()
  clusterTwo <- c()
  clusterThree <- c()

  #This sorts all threehundred points by cluster.
  for(i in 1:300)
  {
    #First the square distance from the k points
    #is calculated for every point in the field.
    SqrdistanceFrom1 <- (pointsFrame[i,1]-pointsFrame[oldPoints[1],1])^2
    SqrdistanceFrom1 <- SqrdistanceFrom1+(pointsFrame[i,2]-pointsFrame[oldPoints[1],2])^2
    SqrdistanceFrom2 <- (pointsFrame[i,1]-pointsFrame[oldPoints[2],1])^2
    SqrdistanceFrom2 <- SqrdistanceFrom2+(pointsFrame[i,2]-pointsFrame[oldPoints[2],2])^2
    SqrdistanceFrom3 <- (pointsFrame[i,1]-pointsFrame[oldPoints[3],1])^2
    SqrdistanceFrom3 <- SqrdistanceFrom3+(pointsFrame[i,2]-pointsFrame[oldPoints[3],2])^2

    #These if statements deposit the points in the
    #appropriate cluster.
    if(SqrdistanceFrom1<=SqrdistanceFrom2&SqrdistanceFrom1<=SqrdistanceFrom3)
    {
      clusterOne <- c(clusterOne,i)
    }
    else if(SqrdistanceFrom2<=SqrdistanceFrom3&SqrdistanceFrom2<=SqrdistanceFrom1)
    {
      clusterTwo <- c(clusterTwo,i)
    }
```

```
    else if(SqrdistanceFrom3<SqrdistanceFrom2&SqrdistanceFrom3<SqrdistanceFrom1)
    {
      clusterThree <- c(clusterThree,i)
    }
    #This warns me of dirty bugs.
    else
    {
      print("bug")
    }
  }

  #Next the means of the clusters are calculated.
  clusterOneMean <- mean(pointsFrame[clusterOne,1])
  clusterOneMean <- c(clusterOneMean,mean(pointsFrame[clusterOne,2]))

  clusterTwoMean <- mean(pointsFrame[clusterTwo,1])
  clusterTwoMean <- c(clusterTwoMean,mean(pointsFrame[clusterTwo,2]))

  clusterThreeMean <- mean(pointsFrame[clusterThree,1])
  clusterThreeMean <- c(clusterThreeMean,mean(pointsFrame[clusterThree,2]))

  #Finally the point in each cluster closest to the
  #mean is made to be the new k point for the cluster.
  newPoints[1]<- clusterOne[which.min((pointsFrame[clusterOne,1]-clusterOneMean[1])^2+
                            (pointsFrame[clusterOne,2]-clusterOneMean[2])^2)]
  newPoints[2]<- clusterTwo[which.min((pointsFrame[clusterTwo,1]-clusterTwoMean[1])^2+
                            (pointsFrame[clusterTwo,2]-clusterTwoMean[2])^2)]
  newPoints[3]<- clusterThree[which.min((pointsFrame[clusterThree,1]-clusterThreeMean[1])^2+
                            (pointsFrame[clusterThree,2]-clusterThreeMean[2])^2)]
  #Next the algorithm will reevaluate if it is in a stable state.
  #If not, it will continue.
}

#This is space for the new classifications.
clustersAfter <- rep(" ",300)

#Now the labels are reapplied.
#The awkward pairing makes sure the coloring lines up.
clustersAfter[clusterOne] <- "x2"
clustersAfter[clusterTwo] <- "x3"
clustersAfter[clusterThree] <- "x1"

#This factorized clustersAfter to make sure
#the clustering "sticks."
clustersAfter <- factor(clustersAfter,levels=c("x1","x2","x3"))

#And here is the plot of the data afterwards.
plot(v2~v1,col=clustersAfter,pch=9)
text(-0.012,-0.007,"(-.012,-.007)",col="pink")
text(3.1458,0,"(3.145,0.000)",col=9)
text(1.462,2.72,"(1.462,2.725)",col=9)
```

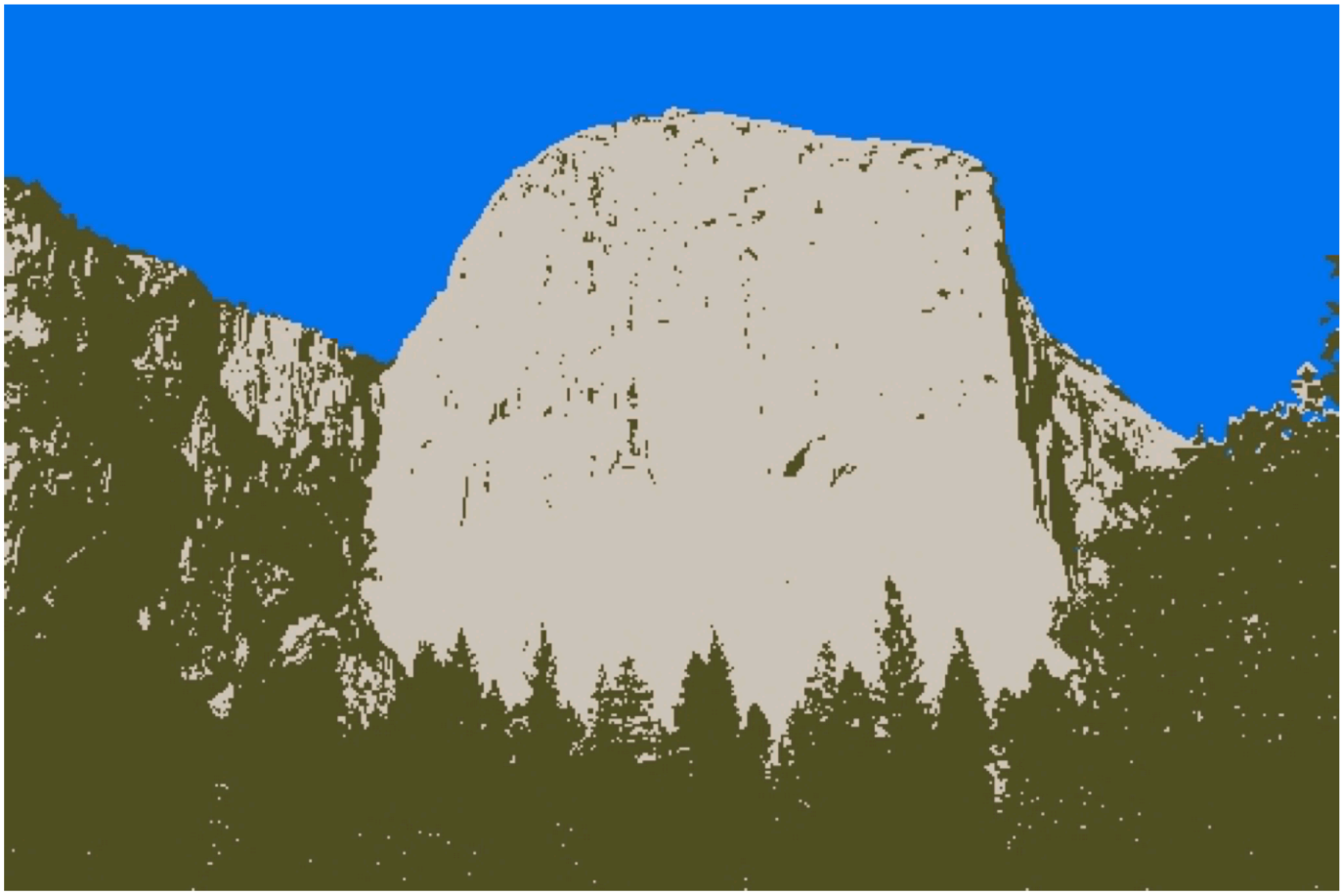
3 Problem 2

3.1 question

Problem 2.

Attached is a picture of El Capitan in the Yosemite National Park. Do the k-means clustering using your code to cluster the pixels into three groups: "sky", "rock", and "tree". Now, redraw the picture by using your three means you found (which means that every pixel can have to choose its color from the three mean values). Note that if you ignore all the positions of the pixels, you can reformulate the image into a matrix of (3 X of all pixels).

3.2 answer



Here is the finished product. It is a clusterized, or as I like to call it, "Warhol-ized" photo of the original. And I diffed it with the one on Piazza, they are different despite looking very similar. I am assuming that because the question did not say to label the photo, I don't have to. Anyways, the real story with this question is in the code. My code is generalized to any number of colors you wish to have, and it prevents infinite loops. Feel free to use it actually... if you want.

raw code

```
#This library is necessary to read and write jpegs
```

```
install.packages("jpeg")
library(jpeg)

#Read in the jpeg
myjpg <- readJPEG("/users/CharlesBSwarts/Desktop/elcapitan3.jpg",native=FALSE)

#Set the number of colors you want it to have
k <- 3

#Seperate the 3D array into 3 1D vectors.
RED <- as.vector(myjpg[,,1])
GREEN <- as.vector(myjpg[,,2])
BLUE <- as.vector(myjpg[,,3])

#Ponts is a vector of indices that are the mean/
#starter points of the next iteration of the
#k-means clustering.
newPonts <- c()
length(newPonts) <- k
oldPonts <- rep(-1,k)

#Need to pick k random points.
newPonts <- sample(1:(dim(myjpg)[1]*dim(myjpg)[2]),k,replace=FALSE)

#Now, clusters are stored in a jagged array,
#but R does not support jagged arrays except
#in lists of lists. So this clus is going to
#be the list that holds the other lists.
clus <- list()
length(clus) <- k

#DisSqr is the square distance of each point
#from the means.
DisSqr <- as.data.frame(matrix(ncol=k,nrow=dim(myjpg)[1]*dim(myjpg)[2]))

#clusMean is the mean of the values for each
#of the colors
clusMean <- as.data.frame(matrix(ncol=k,nrow=3))

#This stopper is here just in case we get an
#overly noisy photo or for some reason the
#algorithm begins to cycle.
stopper <- 0

#If the old points equal the new points, then
#we have found a minimum.
while(!all(oldPonts==newPonts)&stopper<100)
{
  #Store the new points as the old points.
  oldPonts <- newPonts
  #reset the clus list.
  clus <- list()
```

```
  #This calculates the square distance of every
  #point from every k point.
  for(i in 1:k)
  {
    DisSqr[,i] <- (RED-RED[oldPonts[i]])^2+(GREEN-GREEN[oldPonts[i]])^2+
                                          (BLUE-BLUE[oldPonts[i]])^2
  }

  #This calculates the minimum in every row
  values<- apply(DisSqr,1,min)

  #This sorts the respective points to their
  #respective cluster.
  for(i in 1:k)
  {
    clus[i] <- list(which(DisSqr[,i]==values))
  }

  #This calculates the mean value for each cluster
  for(i in 1:k)
  {
    clusMean[,i] <- c(mean(RED[unlist(clus[i])]),mean(GREEN[unlist(clus[i])]),mean(BLUE[unlist
  }

  #This chooses new starting k points based
  #on the mean value of each cluster.
  for(i in 1:k)
  {
    newPonts[i]<- unlist(clus[i])[which.min((RED[unlist(clus[i])]-clusMean[1,i])^2+(GREEN[unlist
  }
  #Increment the nasty stopper to avoid
  #infinite loops
  stopper <- stopper+1
}



#This creates vectors to be reinverted back
#into an image.
mapRED <- c()
mapGREEN <- c()
mapBLUE <- c()
length(mapRED) <- dim(myjpg)[1]*dim(myjpg)[2]
length(mapGREEN) <- dim(myjpg)[1]*dim(myjpg)[2]
length(mapBLUE) <- dim(myjpg)[1]*dim(myjpg)[2]


#This function puts the k point's color
#onto every pixel in its cluster.
for(i in 1:k)
{
```

```
mapRED[unlist(clus[i])] <- RED[newPonts[i]]
mapGREEN[unlist(clus[i])] <- GREEN[newPonts[i]]
mapBLUE[unlist(clus[i])] <- BLUE[newPonts[i]]
}


#This creates the final array for the writeJPEG
#function
imgArr <- array(dim=c(dim(myjpg)[1],dim(myjpg)[2],3))
imgArr[,,1] <- mapRED
imgArr[,,2] <- mapGREEN
imgArr[,,3] <- mapBLUE

#And this outputs the photo.
writeJPEG(imgArr,target="/users/CharlesBSwarts/Desktop/Warhol.jpg",quality=1)
```

MATH 347 HW7

Charles Swarts
swarts2@illinois.edu

October 2016

1 Q 14.5

(−) Find a counterexample to the following false statement. "If $a_n < b_n$ for all n and $\sum b_n$ converges, then $\sum a_n$ converges."

Premise: counter-example.

Let $a_n = -1$ and $b_n = 0$ then $a_n < b_n$ for all n and $\sum b_n$ converges like so

$$\sum_{n=1}^{\infty} b_n = \sum_{n=1}^{\infty} 0 = 0$$

But if we assume the series $\sum_{n=1}^{\infty} a_n$ converges to S

$$\sum_{n=1}^{\infty} a_n = \sum_{n=1}^{\infty} -1$$

$$S = \sum_{n=1}^{\infty} -1$$

$$S = -1 + \sum_{n=1}^{\infty} -1$$

$$S + 1 = \sum_{n=1}^{\infty} -1$$

$$S + 1 = S$$

$$1 = 0 \qquad ※$$

So $\sum a_n$ must be divergent.

Recap: we found an $\langle a \rangle$ and $\langle b \rangle$ where $a_n < b_n$ and $\sum b_n$ is convergent, but where $\sum a_n$ is divergent. ■

2 Q 14.12

If $a_n \to 0$ and $b_n \to 0$, then $\sum a_n b_n$. converges.

Premise: false due to counter-example.

Consider the sequence $c_n = \frac{1}{n}$. This sequence fails to produce a convergent series. Here is proof from the book

14.28. Example. The harmonic series. Consider $\sum_{k=1}^{\infty} 1/k$. To see that $\sum_{k=1}^{\infty} 1/k$ diverges even though $1/k \to 0$, we compare this with another divergent series whose terms approach 0. Let $\langle c \rangle = \frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{16}, \cdots$; here there are 2^{j-1} copies of $1/2^j$ for each $j \geq 1$. Since the copies of $1/2^j$ for each fixed j sum to $1/2$, for each $M \in \mathbb{N}$ the partial $\sum_{k=1}^{n} c_k$ exceeds M for large enough n, and $\sum_{k=1}^{n} c_k$. The last copy of $1/2^j$ in $\langle c \rangle$ is the $2^j - 1th$ term. Thus we have $1/k > c_k$ for every k. For each n, summing n of these inequalities yields $\sum_{k=1}^{n} 1/k > \sum_{k=1}^{n} c_k$. Hence $\sum_{k=1}^{n} 1/k$ also diverges. ■

(Mathematical Thinking Problem Solving and Proofs, D'Angelo and West, Second Edition, ISBN 0-13-014412-6, page 282)

So then take $a_n = 1/\sqrt{n}$ and $b_n = 1/\sqrt{n}$

Lemma: $\lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$
Lemma Premise: definition of limit

The definition of a limit states that

$$\forall \epsilon \in \mathbb{R},\ \epsilon > 0,\ \ \exists N \in \mathbb{N},\ \ \forall n \in \mathbb{N},\ \ n \geq N \Rightarrow \mid a_n - L \mid < \epsilon$$

We need to show

$$\mid \frac{1}{\sqrt{n}} - 0 \mid < \epsilon$$

Since $1/\sqrt{n} > 0$, it is sufficient to show

$$\frac{1}{\sqrt{n}} < \epsilon$$

$$\frac{1}{\epsilon^2} < n$$

Therefore N exists. □

So according to the lemma, $a_n \to 0$ and $b_n \to 0$

But $a_n * b_n = c_n$, and we know $\langle c \rangle$ doesn't produce a converging series. ■

3 Q 14.13

Prove that if $\langle a \rangle$ converges, then every subsequence of $\langle a \rangle$ converges and has the same limit as $\langle a \rangle$.

Premise: Proof by definition of limit and subsequence.

By definition of subsequence, a subsequence, $\langle As \rangle$, is a subsequence of a sequence, $\langle A \rangle$, if $As_i = A_{f(i)}$ where $f : \mathbb{N} \to \mathbb{N}$ is an increasing function.

Lemma: $f(i) \geq i$
Lemma Premise: induction on k

Base Case: $k = 1$

$$f(k) = f(1)$$

$f(1)$ must be a natural number, so $f(1) \geq 1$. ✓

Inductive Step
Hypothesis: $f(k) \geq k$

By definition of increasing function

$$k + 1 > k \quad \Rightarrow \quad f(k+1) > f(k)$$

$$f(k+1) > f(k) \geq k$$

$$f(k+1) > k$$

$$f(k+1) \geq k + 1$$

So by the principle of induction, $f(i) \geq i$ □

The definition of a sequence converging is the same as it having a limit. The definition of a limit, $L \in \mathbb{R}$, for an ordered sequence of real numbers, $\langle A \rangle$ is

$$\forall \epsilon \in \mathbb{R} \ , \epsilon > 0, \ \exists N \in \mathbb{N}, \ \forall n \in \mathbb{N}, \ n \geq N \Rightarrow \mid A_n - L \mid < \epsilon$$

So by the lemma,

$$f(N) \geq N$$

And by definition of increasing

$$n \geq N \quad \Rightarrow \quad f(n) \geq f(N) \geq N$$

So

$$n \geq N \Rightarrow f(n) \geq N \Rightarrow \mid A_{f(n)} - L \mid < \epsilon$$

$$n \geq N \Rightarrow f(n) \geq N \Rightarrow \mid As_n - L \mid < \epsilon$$

This means that any subsequence of $\langle A \rangle$ must also converge and have L as its limit. ■

4 Q 14.30

Let $\langle x \rangle$ be the sequence given by $x_1 = 1$ and $x_{n+1} = 1/(x_1 + \cdots + x_n)$ for $n \geq 1$. Prove that $\langle x \rangle$ converges, and obtain the limit.

Premise: Use MCT and infinity trick.

Lemma: First let us give a lower bound by showing that $\forall \, n, \ \ x_n > 0$

Lemma Premise: Proof by strong induction on k

Base Case: $k = 1 \quad x_k = x_1 = 1 \quad \checkmark$

Induction: Assume all x_1-x_k are positive, then their sum is positive.

$$x_{k+1} = \frac{1}{\sum_{i=1}^{k} x_i}$$

Since the the numerator and denominator on the RHS are positive, x_{k+1} is positive. $\square$

So we see there is a lower bound on x_n. Next we prove the x_n is decreasing $\forall\, n > 2$.

$$x_{n+1} < x_n$$

$$\frac{1}{\sum_{j=1}^{n} x_j} < \frac{1}{\sum_{i=1}^{n-1} x_i}$$

$$\frac{1}{\sum_{j=1}^{n-1} x_j + x_n} < \frac{1}{\sum_{i=1}^{n-1} x_i}$$

$$\sum_{i=1}^{n-1} x_i < \sum_{j=1}^{n-1} x_j + x_n$$

$$0 < x_n$$

Since the lemma showed all terms are positive, the last line is true. This implies the first line is also true. Since the next element in the sequence is always less than the current, x_n must be decreasing $\forall\, n > 2$.

By MCT, a decreasing sequence with a lower bound converges, so $\langle x \rangle$ converges.

Now we can use the infinity trick to find the limit.

$$\lim_{n\to\infty} x_n = \lim_{n\to\infty} \frac{1}{\sum_{i=1}^{n} x_i}$$

$$= \lim_{n\to\infty} \frac{1}{\sum_{i=1}^{n-1} x_i + x_n} = \lim_{n\to\infty} \frac{1}{\sum_{i=1}^{n} x_i + x_n}$$

So

$$\lim_{n\to\infty} x_n = \lim_{n\to\infty} \frac{1}{\sum_{i=1}^{n} x_i + x_n}$$

$$\lim_{n\to\infty} x_n = \lim_{n\to\infty} \frac{1}{\frac{1}{x_n} + x_n}$$

$$0 = \lim_{n\to\infty} \frac{1}{x_n(\frac{1}{x_n} + x_n)}$$

$$0 = \lim_{n\to\infty} \frac{1}{1 + x_n \cdot x_n}$$

$$\lim_{n\to\infty} 1 + x_n \cdot x_n = 1$$

$$\lim_{n\to\infty} x_n \cdot x_n = 0$$

$$\lim_{n\to\infty} x_n = 0$$

Therefore according to the MCT and the infinity trick, $\langle x \rangle$ converges to 0. $\blacksquare$

5 Q 14.44

Compute $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$. Use this to obtain upper and lower bounds on $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

Premise: Derive an easier partial sum formula. Then define a sequence from each of the given series and use them to give upper and lower bound on $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

Lemma: $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$

Lemma Premise: Proof by induction on k.

Base case: $k = 1$

$$\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1}$$

$$\sum_{i=1}^{1} \frac{1}{i(i+1)} = \frac{1}{1+1}$$

$$\frac{1}{2} = \frac{1}{2} \qquad \checkmark$$

Inductive Step: Hypothesis $\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1}$

$$\sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \sum_{i=1}^{k} \frac{1}{i(i+1)} + \frac{1}{(k+1)(k+2)}$$

$$= \frac{k}{k+1} + \frac{1}{(k+1)(k+2)} = \frac{k(k+2)+1}{(k+1)(k+2)}$$

$$= \frac{(k+1)^2}{(k+1)(k+2)} = \frac{k+1}{k+2}$$

Then by the principle of induction, the hypothesis of the lemma is true. $\square$

So using the lemma

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i(i+1)} = \lim_{n\to\infty} \frac{n}{n+1}$$

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = \lim_{n\to\infty} 1 - \frac{1}{n+1}$$

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1 - \lim_{n\to\infty} \frac{1}{n+1}$$

We know from class that

$$0 < \frac{1}{n+1} < \frac{1}{n}$$

$$\lim_{n\to\infty} 0 \leq \lim_{n\to\infty} \frac{1}{n+1} \leq \lim_{n\to\infty} \frac{1}{n}$$

$$0 \leq \lim_{n\to\infty} \frac{1}{n+1} \leq 0$$

$$\lim_{n\to\infty} \frac{1}{n+1} = 0$$

So

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1$$

Now that we know this, we can do a sequence analysis. We use the property of real numbers that

$$a < b \qquad c < d \qquad \Rightarrow \qquad a + c < b + d$$

So if we have two sequences, $\langle A \rangle$ and $\langle B \rangle$ where $\forall\, i \in \mathbb{N},\ A_i < B_i$ then the series $\sum_{i=0}^{\infty} A_i < \sum_{i=0}^{\infty} B_i$.

Let $\langle A \rangle$ be made of individual terms from $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$, and let $\langle B \rangle$ be made of individual terms from $\sum_{n=1}^{\infty} \frac{1}{n^2}$.

$$\begin{aligned} \frac{1}{i(i+1)} &< \frac{1}{i^2} \\ \frac{1}{(i+1)} &< \frac{1}{i} \\ 0 &< 1 \qquad \checkmark \end{aligned}$$

So

$$\begin{aligned} \sum_{i=0}^{\infty} A_i &< \sum_{i=0}^{\infty} B_i \\ 1 &< \sum_{n=1}^{\infty} \frac{1}{n^2} \end{aligned}$$

To get the other bound, we shift $\langle B \rangle$ one over.

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1} + \sum_{n=2}^{\infty} \frac{1}{n^2} = 1 + \sum_{n=1}^{\infty} \frac{1}{(n+1)^2}$$

Let $\langle B \rangle$ now be the individual terms of the series on the RHS.

$$\begin{aligned} \frac{1}{(i+1)^2} &< \frac{1}{i(i+1)} \\ \frac{1}{(i+1)} &< \frac{1}{i} \\ 0 &< 1 \qquad \checkmark \end{aligned}$$

So

$$\begin{aligned} \sum_{i=0}^{\infty} B_i &< \sum_{i=0}^{\infty} A_i \\ \sum_{n=1}^{\infty} \frac{1}{(n+1)^2} &< 1 \\ 1 + \sum_{n=1}^{\infty} \frac{1}{(n+1)^2} &< 2 \\ \sum_{n=1}^{\infty} \frac{1}{(n)^2} &< 2 \end{aligned}$$

So using the computation for $\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1$, and some facts about sequences, we derived an upper bound, 2, and a lower bound, 1, for $\sum_{n=1}^{\infty} \frac{1}{n^2}$. ■

MATH 347 HW8

Charles Swarts
swarts2@illinois.edu

November 2016

1 Q 1

the official definition of divisibility $a \mid b \coloneqq \exists\, m \in \mathbb{Z} : b = ma$

1.1 a

If $d \mid a$ and $d \mid b$, then $d \mid ax + by$ for any $x, y \in \mathbb{Z}$.

Premise: direct method.

Suppose it is the case that $d \mid a$ and $d \mid b$. And also suppose we are given $x, y \in \mathbb{Z}$
Then by definition of divisibility

$$d \mid a \coloneqq \exists\, m \in \mathbb{Z} : a = md$$

$$d \mid b \coloneqq \exists\, n \in \mathbb{Z} : b = nd$$

Then by the rules of standard algebra

$$ax + by = xmd + ynd$$

$$ax + by = (xm + yn)d$$

By closure (i.e. since the integers are closed under addition and multiplication i.e. an integer plus an integer is an integer, and an integer times an integer is an integer) $(xm + yn) \in \mathbb{Z}$

And again by definition of divisibility,

$$d \mid ax + by$$

■

1.2 b

If $a \mid b$ and $c \mid d$, then $ac \mid bd$.

Premise: direct method.

Suppose it is the case that $a \mid b$ and $c \mid d$.

Then by definition of divisibility,

$$a \mid b \coloneqq \exists\, m \in \mathbb{Z} : b = ma$$

$$c \mid d \coloneqq \exists\, n \in \mathbb{Z} : d = nc$$

Then by the rules of standard algebra

$$b \cdot d = ma \cdot nc$$

$$bd = mn \cdot ac$$

By closure $bd, ac, mn \in \mathbb{Z}$

By the definition of divisibility,

$$ac \mid bd$$

■

1.3 c

If $a \mid b$ and $c \mid d$, then $(a + c) \mid (b + d)$.

Premise: counter-example

Let $a = 1,\ b = 2,\ c = 3,\ d = 3$

By definition of divisibility:

$$a \mid b \coloneqq \exists\, m \in \mathbb{Z} : b = ma$$
$$c \mid d \coloneqq \exists\, n \in \mathbb{Z} : d = nc$$

Since $2 \in \mathbb{Z}$ and $b = 2a$, by definition $a \mid b$. Since $1 \in \mathbb{Z}$ and $d = 1c$, by definition $c \mid d$. However

$$(a + c) = 4 \qquad\qquad (b + d) = 5$$

So if $(a + c) \mid (b + d)$, by definition,

$$\exists\, l \in \mathbb{Z} : \qquad (b + d) = l(a + c)$$
$$\frac{(b + d)}{a + c} = l$$
$$\frac{5}{4} \in \mathbb{Z} \qquad\qquad ※$$

Hence we have found an example where $a \mid b$ and $c \mid d$, but ~~$(a + c) \mid (b + d)$~~ ■

2 Q 2

basic definition of congruence: $a \equiv b \bmod m \coloneqq \exists\, k \in \mathbb{Z} : a = b + km$

2.1 a

If $a \equiv b \bmod m$ and $c \equiv d \bmod m$, then $ac \equiv bd \bmod m$.

Premise: direct method.

Suppose $a \equiv b \bmod m$ and $c \equiv d \bmod m$.
Then by definition of congruence:

$$\exists\, j \in \mathbb{Z} : a = b + jm$$
$$\exists\, k \in \mathbb{Z} : c = d + km$$

Then by the rules of standard algebra

$$a \cdot c = (b + jm) \cdot (d + km)$$
$$ac = bd + (b \cdot km + jm \cdot d + jm \cdot km)$$
$$ac = bd + m \cdot (bk + jd + jkm)$$

By closure $(bk + jd + jkm) \in \mathbb{Z}$

So by definition of congruence $ac \equiv bd \bmod m$ ■

2.2 b

If $a \equiv b \bmod m$, then for any $k \in \mathbb{N}$, $a^k \equiv b^k \bmod m$.

Premise: induction on i

Base case: Suppose $a \equiv b \bmod m$. Then by the previous proof, using its framework, we let $c = a$ and $d = b$. The result is that $a^2 \equiv b^2 \bmod m$.

Inductive case: Suppose $a^i \equiv b^i \bmod m$. Then by the previous proof, using its framework, we let $c = a$ and $d = b$. The result is that $a^{i+1} \equiv b^{i+1} \bmod m$.

By the principle of induction, if $a \equiv b \bmod m$, then $a^k \equiv b^k \bmod m$. ■

3 3

Let $\mathbb{P}$ represent the set of all prime numbers. *Fermat's Little Theorem:* $p \in \mathbb{P} \Rightarrow a^p \equiv a \bmod p$

3.1 a

Fine the last decimal digit of 347^{101}.

Premise: Since we are using the base 10 number system, finding the congruence mod 10 from the set $\{0\} \cup [9]$ should give the last digit.

Firstly using result from 2.b, we see

$$347 \equiv b \bmod 10 \Rightarrow 347^{101} \equiv b^{101} \bmod 10$$

It is trivial to see that in this base,

$$347 \equiv 7 \bmod 10$$

So now we use the same process, using 2.a and 2.b

$$7^{101} \equiv 7 \cdot 7^{100} \equiv 7 \cdot 49^{50} \equiv 7 \cdot 9^{50} \equiv 7 \cdot 81^{25} \equiv 7 \cdot 1^{25} \equiv 7 \bmod 10$$

So the last digit must be 7.

3.2 b

Find the remainder of 347^{101} when divided by 101.

Premise: to find the remainder, we need to find the number from the set $\{0\} \cup [101]$ that is congruent to $347^{101} \bmod 101$.

Using Fermat's Little Theorem, we see that

$$347^{101} \equiv 347 \bmod 101$$

Which is most of the way, except $347 \notin \{0\} \cup [101]$

Luckily we know the definition of congruence

$$a \equiv b \bmod m \coloneqq \exists\, k \in \mathbb{Z} : a = b + km$$

So

$$347 \equiv 347 + -3(101) \bmod 101 \equiv 44$$

So the remainder is 44. ■

3.3 c

Using Fermat's Little Theorem, find a number between 0 and 12 that is congruent to 2^{100} modulo 13.

Premise: use Fermat's Little Theorem, 2.a.

Since we know Fermat's theorem, we know

$$2^{13} \equiv 2 \bmod 13$$

Using this fact and 2.a, we can compute

$$2^{100} \equiv 2^9 \cdot 2^{13} \cdot 2^{13} \cdot 2^{13} \cdot 2^{13} \cdot 2^{13} \cdot 2^{13} \cdot 2^{13}$$

$$\equiv 2^9 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \equiv 2^{13} * 2^3 \equiv 2 \cdot 2^3 \equiv 2^4 \equiv 16 \bmod 13$$

Luckily we know the definition of congruence

$$a \equiv b \bmod m := \exists\, k \in \mathbb{Z} : a = b + km$$

So

$$16 \equiv 16 + (-1) \cdot 13 \equiv 3 \bmod 13$$

So 3 is the number between 0 1nd 12 that is congruent to 2^{100}. ■

3.4 d

Find the last digit in the base 8 expansion of (i)9^{1000},(ii)10^{1000},(iii)11^{1000}.

Premise: Since we are converting to octal, we need to find the number which is congruent mod 8 and in the the set $\{0\} \cup [7]$.

3.4.1 i

By the definition of congruence

$$9 \equiv 9 + (-1)8 \equiv 1 \bmod 8$$

Since we know that, by 2.b

$$9^{1000} \equiv 1^{1000} \equiv 1 \bmod 8$$

So the last digit in octal will be 1.

3.4.2 ii

By the definition of congruence

$$10 \equiv 10 + (-1)8 \equiv 2 \bmod 8$$

Since we know that, by 2.b

$$10^{1000} \equiv 2^{1000} \equiv 8^{250} \bmod 8$$

By definition of congruence

$$8 \equiv 8 + (-1)8 \equiv 0$$

Since we know that, by 2.b

$$8^{250} \equiv 0^{250} = 0 \bmod 8$$

So the last digit in octal will be 0.

3.4.3 iii

By the definition of congruence

$$11 \equiv 11 + (-1)8 \equiv 3 \bmod 8$$

Since we know that, by 2.b

$$11^{1000} \equiv 3^{1000} \equiv 9^{500} \bmod 8$$

By the definition of congruence

$$9 \equiv 9 + (-1)8 \equiv 1 \bmod 8$$

Since we know that, by 2.b

$$9^{500} \equiv 1^{500} = 1 \bmod 8$$

So the last digit in cotal will be 1.