Homework 3

Jack Wilburn

October 25th, 2021

1 Q1: Set Cover Analysis

1.1 Write down (i) the optimal solution, and (ii) the solution output by the greedy algorithm

1. The optimal solution is to take S_1 and S_2, this covers all the skills with just 2 people.
2. The greedy algorithm would select S_5, S_4, and S_3, a sub-optimal solution.

1.2 Describe an instance in which the ratio of the size of the solution produced by the greedy algorithm and the size of the optimal solution is Ω(log m)

Based on the figure and the fact that it a hint, I'm assuming the thing to do is to keep adding sets like S_3, S_4, and S_5, each time multiplying the number of elements in the new set by 3. Carrying on in this way, the optimal solution would always be 2 and the greedy solution would always choose the sets that like S_3, S_4, and S_5.

In this case, the total skills would be given by $m = 2 + \Sigma_{x=1}^{k}(3^x * 2)$ and the chosen number of sets for the greedy algorithm would be $log_3(m-1)$. The optimal solution would still be just 2, S_1 and S_2, thus the greedy algorithm is off by a factor of $\frac{log_3(m-1)}{2} \in \Theta(log_3(m))$

1.3 Prove that at least 50% of the skills are covered after k steps

Given the class lecture and the proof given in the lecture, we know that with the greedy algorithm the number of people required to cover all the skills is decreasing by a rate of 1/k where k is the optimal number of people to cover the skills. Given this knowledge we can show that the claim in the question is correct.

The number of uncovered skills at any time t is $|m| * (1 - \frac{1}{k})^k$, this comes from the fact that at each time period we're reducing the number of skills remaining by a factor of k (asserted in the class lecture). What we ultimately want to show is that $|m| * (1 - \frac{1}{k})^k \leq \frac{|m|}{2}$. That is, after k iterations, we have less than half the skills remaining to choose from. Let's simplify that equation to $(1 - \frac{1}{k})^k \leq \frac{1}{2}$. The left side of the equation is less than $\frac{1}{e}$ (which is less than $\frac{1}{2}$) for all real $k > 0$. Since the remaining uncovered skills after k steps is less than 50%, the number of skills covered is greater than 50% for any k.

2 Q2: Be lazy or be greedy?

2.1 Show how to "cast" the street surveillance problem as an instance of Set Cover, and write down the greedy algorithm

To cast this problem to the skill set coverage, let the nodes be people and the coverage areas (edges) be the skills. The greedy algorithm would then pick nodes based iteratively based on covering the most uncovered coverage areas.

Pseudo-code:

```
1. Pick the node that has the most uncovered coverage areas connected to it.
2. If not fully covered, go back to step one. If fully covered, stop.
```

Correctness and **time complexity** omitted per the question.

2.2 Prove that this set satisfies $|S| \leq 2k$, where k is the size of the optimal solution (i.e., the minimum number of nodes we need to place cameras at in order to monitor all the edges)

Each time we find an unmonitored edge and select it, adding the nodes to our set, we remove certain edges from consideration. The removed edges are all of those with 1 or more endpoints in our chosen set (so far). Thus, when the algorithm is complete, the edges that we chose that were unmonitored (call it R), share no endpoints. Additionally, we can't add any more edges, so every edge must be monitored by at least one node.

The edges that we collected above have a certain size, call it $|R|$. Since we're always taking both nodes on each end of the edge, our set of nodes in this solution is $2 * |R|$. Any solution to this problem must have at least size $|R|$, since at least one endpoint of the edges we selected must be in the optimal solution. Since there are no endpoints that are shared, there are $|R|$ different vertices as a minimum possible set. Thus an optimal solution must have $|R|$ vertices, whereas our lazy algorithm has $2 * |R|$ vertices.

Given that our problem statement says that there are k vertices in the optimal solution, our solution will have no more than $2 * k$ vertices through the above logic (basically rename R to k).

3 Q3: Finding triangles

3.1 Show how to do this in time $O\left(\sum_{i \in V} d_{in}(i) d_{out}(i)\right)$, where d_{in} and d_{out} refer to the in-degree and out-degree of a vertex respectively.

Pseudo-code:

```
1. Iterate through each vertex in the graph and if the vertex has both in and out edges consider that
   node for step 2, else continue onto another node. If there are no unvisited nodes remaining, report
   "no cycles" and quit.
2. For each in edge coming from vertex $x_i$ (for as many is as there are in edges), look at each vertex
   at the end of every out edge going to $y_j$ (for as many j s there are out edges).
3. Check if there is a connection from $y_j$ to $x_i$. If there is, we've found a cycle so report it and quit.
   If not, go back to 2 and pick different edge combinations to check.
```

Time complexity: The time complexity of this algorithm for one vertex, i, is $O\left(d_{in}(i) d_{out}(i)\right)$ since for each in edge we have to check every out edge. Thus we get that multiplication. Since we have to repeat this for all vertices, we get the sum from the problem statement, where we simply add that multiplication over all $i \in V$.

Correctness: This is an exhaustive search for cycles so it is correct. We check all possibilities from all nodes.

3.2 Show how to solve the problem in time $O(n^2.4)$.

Let the adjacency matrix be called A. Multiplying the same matrix together k times tells you how many paths there are between nodes i and j of length k in the graph at index i,j. Thus, if we're looking for cycles of length 3 (triangles) in A, we can raise A to the third power and check the diagonal to see if there are paths of length 3 that go from i to i for all $i \in A$

Pseudo-code:

```
1. Compute $A^3$
2. Check all diagonal elements of the computed matrix for a value greater than 0. If we find one
   report "triangle detected"
3. If there are no diagonal elements greater than 0, report "no triangles"
```

Correctness: This algorithm is correct based on the algebraic rules we're employing. That is the multiplication gives the number of paths of a given length in the position i,j.

Time complexity: This algorithm will take $O(2 * n^2.4) \in O(n^2.4)$ since we have to do 2 matrix multiplications.

4 Q4: Road tripping

4.1 Give an example in which the greedy algorithm is not optimal. I.e., give a set of song durations for which the above procedure uses more CDs than an optimal "packing"

Imagine a scenario where you have 8 songs, 4 of length 16 and 4 of 14. An optimal solution would be 2 16 minute songs and 2 14 minute songs on each disk. that would require 2 disks and uses the entire space available.

The greedy solution, however, chooses all of the 16's, first putting 48 minutes of audio onto disk one and 16 on disk 2. The remaining 14 minute tracks don't fit on disk one so start to fill disk 2, until the running time hits 58 minutes and 3 of the 14 minute songs are on the second disk. There is one final 14 minute song that would be written to a new disk, 3. Thus the greedy algorithm is not optimal.

4.2 Suppose you are told that all the song durations are at most 20 minutes. Now prove that the greedy algorithm is within a factor 3/2 of the "best possible" number of CDs

Let OPT be the optimal number of disks, and $D_1, D_2, ..., D_n$ be the disks required to hold all the tracks of durations $d_1, d_2, ..., d_k$. First we need to know that $OPT \geq \Sigma_{i \leq k} \frac{d_i}{60}$ since the sigma sum represents an optimal solution if the track all fit perfectly onto each disc with no gaps.

Now, the used duration of each disk must be at least 40 minutes for all but the last disk, because if there were less than 40 minutes remaining, we could fit any additional song onto that disk and the greedy algorithm would. Thus all disks are at least 2/3 filled except the last disk which comes with no guarantees. Thus we have

$\frac{2}{3} * (n-1) \leq \Sigma_{i \leq k} \frac{d_i}{60} \leq OPT$ which can be rearranged to $n \leq \frac{3*OPT}{2} + 1$ thus we've shown that the number of discs chosen in the case where each track is at most 20 minutes, is at most $\frac{3}{2}$ times the optimal number of disks.

5 Q5: Santa's tradeoffs

5.1 Prove that a locally optimal solution produced this way has a value that is at least (2/3) the optimum value.

Assume $\Sigma H_{i,l_i}$ is the solution found by the algorithm and $\Sigma H_{i,g_i}$ is the globally optimal solution, henceforth called LOC and OPT, respectively. We know that LOC optimizes the triplet distribution of presents such that $H_{i,l_i} + H_{j,l_j} + H_{k,l_k} \geq$ all other permutations of the 3 children and the three gifts. When might this algorithm go badly? If there were a couple of presents that were poorly assigned.

In the case that two presents were poorly assigned, this would imply there is some present g_i that would be better for child i that was assigned to another child (call them j). There would also be a poor assignment of child j's present, where they should have gotten g_j which may have been assigned to some child called k. To simplify a little, there are presents such that $g_i = l_j$ and $g_j = l_k$ In this case, the best thing to do, would be to swap the presents such that we have the following equation:

LOC + LOC + LOC = $H_{i,l_j} + H_{j,l_k} + H_{k,l_i} \geq H_{i,g_i} + H_{j,g_j} + H_{k,l_i}$

Now sum:

$\Sigma H_{i,g_i} + \Sigma H_{j,g_j} + \Sigma H_{k,l_i}$ = OPT + OPT + LOC

By the fact that all presents have a non-negative happiness:

LOC + LOC + LOC $\geq$ OPT + OPT + LOC $\geq$ OPT + OPT

Let's rearrange and divide to get:

3 * LOC $\geq$ 2 * OPT which implies LOC $\geq \frac{2}{3}$ OPT

Similar proof