Written Homework #2

Richard Dizon
CS 4102: Algorithms
University of Virginia

September 9, 2016

1

The first step would be to create an interval from the locations $[p_1, p_{1+1}]$ which covers any points in the set P within the range, otherwise the elements p_1 to p_m where m is the index of the element of set P that exists in the interval. From there we can set the next interval from $[p_m, p_{m+1}]$ also covering any points within the range. This loop continues until all the points in the set P have been included in the interval.

The run-time for this algorithm should be $O(n)$ where n is the number of elements in set P. For each interval, all the points in the set should be covered and the algorithm linearly traverses the set, but for each of those intervals, the algorithm also checks to see which elements belong to the unit-interval, but not necessarily at another full linear check, which would make the run-time more close to $O(n^2)$ but instead will reduce down to simply $O(n)$.

2

Let S be the optimal solution such that $|S| = i$ where i is the minimal number of intervals. The maximum range we can get from the lowest point in the set to encompass as many other points in P as possible would be the range $[p_a, p_a + 1]$ where p_a is the lowest point in the step and append it to

S. We can then remove all the points from P where $P \cap S$ and can greedily continue creating ranges from the minimum remaining values of P until eventually, all the points that existed in P will live in the ranges existing in S.

3

Let the starting location be considered to be side A and the end location to be side B.

For all group sizes that are smaller than 3, the trivial solution would be for the group to be able to make it to side B in one trip.

For all group sizes that are 3 or larger, an efficient algorithm would be that for the *very first* trip, the two fastest people must cross the bridge first from side A to B and the fastest person will take the flashlight back from B to A. Every subsequent trip from A to B will be consist of the two slowest people from side A and again, the fastest person from side B will return the flashlight back to side A. This greedy decision will continue to loop until all people have successfully traveled from side A to B all within $O(nlogn)$ time.

4

An obvious greedy solution would be to attempt to always use the fastest person as a shuttle from both sides A and B, however this doesn't result in the optimal solution, rather it's more efficient to have to two slowest people travel together so that the fastest person won't have to drag the slowest people for nearly double the time it would take for the slowest people to travel together.

This leads us to our alternative greedy solution where after the initial two travels, every subsequent trip from A to B involves the two slowest people where ideally, the slower group of people will never have to risk another high-cost travel of going back from B to A.

5

For any odd value n where n was previously defined to be the amount of items in the dispute, no matter the decided private values, one party will always have a "higher-value" because one side will always have at least one more item than the other party thus returning $FALSE$.

Two algorithms come to mind when trying to discover a solution: one involves implementing a list which will be good for a small n but at suffer at larger sizes having a run-time of $O(n^2)$. The other one involves implementing a hash table which is considerably faster for large sizes given fast, constant time search methods. Additional time is used however for insertions involving the hash function but still simplifies down to $O(logn)$ since not all elements are visited.

Let h be equivalent to $\frac{n}{2}$ where h is considered to be the halfway index between each party's ranked list. The list implementation involves taking the first half of party B's list by means of a double for-loop. If there are and intersections then neither part has mutually disjoint sets and therefore one party will ultimately receive a higher value in the split and thus returning $FALSE$. If no collisions occur then both parties get mostly what they want and we can return $TRUE$. Although only half of both sets are being compared to each other, we still get a run time of $O(n^2)$ because mathematically, we have $\frac{n}{2} \cdot \frac{n}{2}$ computations which still simplifies down to $O(n^2)$ in big-O notation.

The hash implementation involves taking the first half of one party's list (otherwise $O(h)$ times) and running it through a hash function to be added to a table, each costing $O(1)$ computations. We then attempt to do hashed lookups for the first half of the *other* party's list and if there are any collisions, we can return $FALSE$) because the prized list will no longer be disjoint. If no collisions occur then we can return $TRUE$. At worst case, this algorithm will run at $O(n)$ time because ultimately 2 halves of n) hash functions will execute. At linear time, this is still considerably faster than the list implementation especially given large sets of items. The average case however will run at $O(logn)$ since the entire list halves will not be traversed when returning $FALSE$.

6

One idea might be to store file f across all the servers and although the act of adding the file to each server will be linear and make the cost of searching and accessing to be zero, this can prove to be costly given that the point of this problem is to minimize cost. What is most uncertain about this problem is the likelihood of trying to randomly access one of the servers to search for f only to find that it is not located there, and thus costing us. A better idea might be to determine which are the most costly servers and try to maximize the amount of servers we can place f in for the lowest cost.

To do this, we will have to mathematically calculate the maximum cost which would be the sum of all the cost of adding f to all of the servers. This calculation itself will cost us $O(n)$ time. A good start to maximizing the number of servers would be to try to add the file to servers who's sum cost is roughly equivalent to roughly half of the maximum cost. We'd then need to sort these costs and their corresponding servers, and add the file to those servers from the bottow up until the total cost thus far reaches half the maximum cost. What this does is that you will ideally have a greater likelihood of using less than the maximum cost by having at least half of the servers containing the file where only half the time, you will search in a server that does not contain f.

7

Given that a minimum spanning tree forms a tree connecting all the vertices of a graph whose sum of edge weights is as small as possible, Prim's algorithm creates an MST because its algorithm greedily selects an adjacent, un-visited vertex with the smallest edge weight to be the next vertex in the tree. A spanning tree is definitely constructed because no cycles are constructed from the algorithm but connects all the vertices and is considered minimum by quickly choosing the closest points.

To prove this by contradiction, we can suppose that there is a graph G such that an MST was not created, meaning that something along the way of the creation must have gone wrongly. At some vertex insertion, an un-optimal edge must have been selected. This creates a spanning tree that must be

larger than the previously given tree that we considered to be optimal, therefore Prim's Algorithm should return the minimum spanning tree following its greedy selection process.