CMPS 102 Homework 2

Benjamin Gordon / bcgordon@ucsc.edu

5/8/2014

1 Problem 1

Since only one contestant may be in the pool at any given time, and each contestant takes exactly their projected swimming time, the swimming portion of the competition will *always take the same amount of time*, no matter how the contestants are ordered. This is true simply by the commutative property of mathematics, i.e. A+B = B+A. Therefore, since the running and biking times are allowed to overlap, it seems logical that the best way to order the contestants is to overlap these sections as much as possible. From these observations we can come up with a simple algorithm: The next contestant is the one whose projected biking and running times are the longest.

Since we don't have to wait until each contestant is done biking and running - just until they are done swimming - it is obvious that the more contestants we can have biking and/or running at the same time, the more efficient the algorithm and the faster the race will be over. Having the contestants with the longer running times at the end will extend the duration of the entire race due to a small number of contestants who are still running. Using the proposed algorithm ensures that the maximum number of contestants are running and/or biking at the same time.

2 Problem 2

Let a(n) be the total number of packages shipped after n trucks for our greedy algorithm a. Likewise, let b(n) be the total number of packages shipped after n trucks for some arbitrary non-greedy algorithm b. Let P(m, n) be the total number of remaining packages to ship for algorithm m after n trucks.

Base Case: Because the greedy algorithm packs the maximum number of possible packages onto each truck, and because the order in which packages are shipped cannot change, the only thing that can vary is the point at which trucks leave the station. Our greedy algorithm packs each truck until no further packages can fit; our arbitrary other algorithm does not necessarily do so. Therefore, $a(1) \geq b(1)$.

Inductive hypothesis: Now we will prove that for any n, if $a(n) \geq b(n)$, then $a(n+1) \geq b(n+1)$.

Saying $a(n) \geq b(n)$ is equivalent to saying that $P(a, n) \leq P(b, n)$. Since we know that algorithm b has at least as many packages remaining as algorithm a, we can essentially treat this as a separate problem, which is a subset of the original problem. Observe that the total weight that can be fit onto any given truck does not change based on the algorithm used. Therefore, since algorithm a always reaches the upper bound of weight per truck, and algorithm b does not always reach the upper bound of weight per truck, $a(n+1) \geq b(n+1)$ if $a(n) \geq b(n)$.

Therefore, for any number of trucks n, $a(n) \geq b(n)$. The last piece of this proof is to state the obvious: the greater the number of packages which can be fit onto each individual truck, the fewer trucks you will need overall.

3 Problem 3

Part A: This statement is TRUE. Here's why:

The algorithm for building an MST first sorts the edges by weight, then traverses the list of edges, adding each one that does not create a cycle, until it connects every vertex. Now, if we were to square all the weights of all the edges, the sorted order would still be the same. Therefore, since the graph is still the same shape, and the sorted list of edges is still in the same order, the resulting tree will also be the same.

Part B: This statement is FALSE. Here's a counterexample:

Let's say you have a graph with four vertices, A, B, C, and D. We are trying to find the minimum path from A to D. The edges are as follows:

AB: 5 AC: 1 BD: 6 CD: 9

On our starter graph, Dijkstra's algorithm would choose AC-CD as the minimum path from A to D (weight 10) over AB-BD (weight 11). However, when we square all the weights of all the vertices, AB-BD becomes the minimum path (weight $5^2 + 6^2 = 61$) over AC-CD (weight $9^2 + 1^2 = 82$).

4 Problem 4

Essentially, this problem is a MST problem. The twist, however, is that we don't know what the actual weights are. We don't know exactly how f increases numbers, but we do know that it is strictly increasing – that is, for any n, $f(n) < f(n+1)$. Therefore, though we don't know $f(S)$, we know that it has the same sorted order as S. This is similar conceptually to problem 3, part a; we get the same minimum spanning tree, regardless of the weights of the edges, as long as the sorted order of the edges remains the same. Therefore, it is possible to come up with such a subset S that meets the two requirements.

5 Problem 5

Essentially, this problem asks whether one can create any spanning tree of a graph, from any other spanning tree, by swapping one vertex at a time. If you can, then the resulting graph is connected.

Each vertex V in G can have 0, 1, or more than 1 edges connecting it to the graph. If 0, then the graph is not connected and we don't need to worry about it. If 1, then no matter how many changes to the spanning tree you make, that one edge must always be a part of the spanning tree, because it's the only way to include V.

If, however, V has more than 1 edge connecting it to the rest of the graph, then there are multiple ways to connect it to the spanning tree. As long as you remove one of V's edges from the spanning tree before adding a different one, you still maintain the property of not having any cycles. This is because the minimum spanning tree connects every edge to every other; by adding an edge to V without first removing one, it would necessarily create a cycle which includes V. However, by removing V's edge, we prevent the creation of a cycle. By doing this – removing one of V's edges from the spanning tree and adding a different one of V's edges to the spanning tree – we are creating neighbor spanning trees. That is, these two spanning trees have an edge between them in H. Because we can treat any node in G as vertex V, we can reconfigure the spanning tree into any configuration that is legal for a spanning tree to have, simply by removing one edge to a vertex, and adding a different edge to that vertex. Therefore, the graph H is connected.