Algorithms and Data — Problem Set 3

Nick Ippoliti

October 25, 2016

1 The Interval Scheduling Again

This approach is a greedy since it tries to make the locally optimal choice with each interval selection, i.e., it picks the one with the last start time that does not conflict with any of the others with the hope that by doing this repeatedly we arrive at a good global solution.

This does, in fact, yield an optimal solution. Let I be a set of intervals. We first pick an interval I_1 with start time S_1 such that S_1 is greater than all other start times. Then, we let I' be a subset of I such that no intervals in I' intersect with I_1. We repeat the process using I', and continue repeating in this way until no intervals are left.

Thus, in essence, we are picking intervals that begin closest to the maximum end time E_{max} of I and do not overlap any of our currently chosen intervals. The fact that we are choosing the interval with the start time closest to the maximum end time means that there can be no other interval or combination of intervals that do not intersect our current choices and minimize the distance between their start time and the max end time and that interval is the smallest interval that is compatible with our choices. That is to say, choosing in this way will give us a maximum number of non-ovelapping intervals in a given set of intervals.

2 Connected Components

Consider a graph with a single vertex and no edges. There is one connected component: the node itself. Thus: $n - k = 1 - 1 = 0$.

Consider a graph with n nodes and no edges, $e = 0$. This implies that there are k connected components currently and $k = n$. Drawing an edge between any connected components, which at this point are the vertices, increases the edge count e' by 1 and reduces the number of connected components k' by 1. This means the if this was done t times, we would now have $e' = t$ and $k' = n - t$. Simplifying, we get $e' = t = n - k'$. Since we can draw any number of edges between vertices within the same connected component, e' becomes an upper bound, $e' \geq n - k'$. Thus, the number of edges in a graph e will always be at least $n - k$.

3 MST + Shortest Paths

3.1 a

Replacing each edge cost by $\log(1 + c_e)$ will not change the minimum spanning tree. To see this, if we a have an ordered set of edge weights E, mapping $\log(1 + E_i)$ over each weight to produce E' will keep the ordering of E' and E the same. That is, for every element E'_i in E', $2^{E'_i} = E_i$. This is due to the fact that the log function is continuous and increasing for every number greater than 0. Since the ordering of the smallest edge weights would remain unchanged, the MST would remain the same.

3.2 b

Replacing each edge cost by $\sqrt{c_e}$ could change P. This is because square root does not effect numbers linearly, and thus does not effect the sums of path weights linearly. This would effect the path P, since finding the shortest path would involve calculating the total edge weights. For example, if we had a path from u to v containing edge weights 1, 1, and 1, and another path containing edge weight 4, we can see that the first would be faster. However, computing the square root would make the 4 into 2, causing the optimal path to change.

4 Most Reliable Communication Channel

If, for every edge weight w, we replace w with $w' = -\log(w)$, we can then run Djikstra's algorithm on the graph to find the most reliable communication channel. This works due to the fact that the for any real in $0 \leq r \leq 1$, the closer r is to 0 the more negative $\log(r)$ will be, and thus multiplying by -1 will give the most reliable paths the smallest values from $0 \leq w' \leq \infty$.

5 Minimum Spanning Tree

Assume we have a graph G with minimal spanning tree T. If we cut T between two vertices x and y, we get two new trees T_1 and T_2. If we consider the edges in G that join T_1 and T_2 — the edges crossing the cut — and find the lightest edge, that edge MUST be x, y or else T is not a minimum spanning tree.

Thus, assume every cut of G to have a unique light edge x, y and T to be an empty tree. For every cut, we add x, y to T if x or y or both are not in T. This will guarantee that T is a minimal spanning tree, and thus T exists.

Contradiction for the converse: Suppose we have a graph with vertices A, B, C and edges $w(A, B) = 1, w(B, C) = 1, w(A, C) = 2$. This graph has a minimal spanning tree containing $(A, B), (B, C)$. However, the cut that cuts the graph into B and A, C does not have a unique light edge crossing the cut.

6 Edge Disjoint Trees

Let S be the tree of shortest paths from s to every other vertex in a graph G. Let T be the minimum spanning tree of G. Since T is a minimum spanning tree, there must be some edge e with minimal weight connecting s to another vertex in T. Since S is a tree of shortest paths, the edge e must be contained in the set of edges connecting s to it's direct descendants in S. Thus, it is not possible for there to be an S and a G which do not contain any edges.