CS180 — Algorithms and Complexity

HW#2 — Greed, Path Problems

Due: 11:55pm Sunday February 15, 2015

D. Stott Parker, Yuh-Jie Chen, Xiaoran Xu

stott@cs.ucla.edu, eyjchen@cs.ucla.edu, xrxu@cs.ucla.edu

Using CCLE, please upload your answers as a (very readable!) PDF document.

1. **Scheduling**

   Problem 4.13 in the [KT] textbook:

   > *"A small business – say, a photocopying service with a single large machine – faces the following scheduling problem. Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take t_i time to complete. Given a schedule (i.e., an ordering of the jobs), let C_i denote the finishing time of job i. For example, if job j is the first to be done, we would have $C_j = t_j$; and if job j is done right after job i, we would have $C_j = C_i + t_j$. Each customer i also has a given weight w_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i's job. So the company decides that they want to order the jobs to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.*
   >
   > *Design an efficient algorithm to solve this problem. That is, you are given a set of n jobs with a processing time t_i and a weight w_i for each job. You want to order the jobs so as to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} w_i C_i$.*
   >
   > ***Example.** Suppose there are two jobs: the first takes time $t_1 = 1$ and has weight $w_1 = 10$, while the second job takes time $t_2 = 3$ and has weight $w_2 = 2$. Then doing job 1 first would yield a weighted completion time of $10 \cdot 1 + 2 \cdot 4 = 18$, while doing the second job first would yield the larger weighted completion time of $10 \cdot 4 + 2 \cdot 3 = 46$."*

   Please do the problem above, but let $p_i = 2^{w_i}$ and then minimize $\prod_{i=1}^{n} p_i^{C_i}$ instead of minimizing $\sum_{i=1}^{n} w_i C_i$. Please prove the optimality of your algorithm.

2. **Degree Sequences**

   (a) Give a polynomial-time algorithm (in pseudocode) that determines, for a given *degree sequence* — a list of node degree values $d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ — whether there is an undirected graph $G = (V, E)$ that has these degrees.

   In other words, determine whether it is possible for the degrees of nodes in V to match this list, so that degree(v_i) = d_i. We do not permit edges from a node to itself, and any two nodes can be connected by at most one edge. (*Hint:* greed)

   (b) Determine the specific complexity of your algorithm (like $O(n), O(n \log n)$ or $O(n^2)$).

   (c) Using your algorithm, determine whether there is a graph with node degrees [5 4 3 2 2 2 2].

   (d) Can two non-isomorphic (topologically different) graphs have the same degree sequence? If your answer is 'yes', please explain why; if 'no', explain why not.

3. **Clique**

   A clique in an undirected graph $G = (V, E)$ is a completely-connected subset of vertices — i.e., a subset in which every pair of nodes is connected by an edge in E. The size of a clique is the number of vertices it contains. The *clique problem* is the problem of finding a clique of maximum size in a graph.

   Suppose someone gave us a *black box* that takes two inputs — an undirected graph, and an integer $k \leq n$ — and the output is TRUE or FALSE, telling us whether the graph contains a clique of size k. It is also amazingly fast, too, it runs in time $O(1)$, regardless of how large n is. (Black boxes are always fast like this.)

   Design an algorithm whose running time is polynomial in $O(V) = O(n)$ that uses this black box to find a clique of maximum size in a given undirected graph. You can vary the black box inputs any way you like, and can invoke it as many times as you like. In other words, using the black box as a 'subroutine', design a polynomial-time algorithm that solves the clique problem. Derive the time complexity of your algorithm.

4. **Shortest Paths**

(a) In the middle of p.140 in [KT], the text says:
"about Dijkstra's Algorithm and its analysis: ... the algorithm does not always find shortest paths if some of the edges can have negative lengths. (Do you see where the proof breaks?)"
Please explain how the proof of (4.14) on p.139 fails if edge lengths can be negative.

> **(4.14)** *Consider the set S at any point in the DFS algorithm's execution. For each $u \in S$, the path P_u is a shortest s–u path.*
>
> Proof. We prove this by induction on the size of S. The case $|S| = 1$ is easy, since then we have $S = \{s\}$ and $d(s) = 0$. Suppose the claim holds when $|S| = k$ for some value of $k \geq 1$; we now grow S to size $k+1$ by adding the node v. Let (u, v) be the final edge on our $s - v$ path P_v.
>
> By induction hypothesis, P_u is the shortest $s - u$ path for each $u \in S$. Now consider any other $s - y$ path P; we wish to show that it is at least as long as P_v. In order to reach v, this path P must leave the set S *somewhere*; let y be the first node on P that is not in S, and let $x \in S$ be the node just before y.
>
> the crux of the proof is very simple: P cannot be shorter than P_v because it is already at least as long as P_v by the time it has left the set S. Indeed, in iteration $k+1$, Dijkstra's Algorithm must have considered adding node y to the set S via the edge (x, y) and rejected this option in favor of adding v. This means that there is no path from s to y through x that is shorter than P_v. But the subpath of P up to y is such a path, and so this subpath is at least as long as P_v. Since edge lengths are nonnegative, the full path P is at least as long as P_v as well.
>
> This is a complete proof; one can also spell out the argument in the previous paragraph using the following inequalities. Let P' be the subpath of P from s to x. Since $x \in S$, we know by the induction hypothesis that P_x is a shortest $s - x$ path (of length $d(x)$), and so $\ell(P') \geq \ell(P_x) = d(x)$. Thus the subpath of P out to node y has length $\ell(P') + \ell(x, y) \geq d(x) + \ell(x, y) \geq d'(y)$, and the full path P is at least as long as this subpath. Finally, since Dijkstra's Algorithm selected v in this iteration, we know that $d'(y) \geq d'(v) = \ell(P_v)$. Combining these inequalities shows that $\ell(P) \geq \ell(P') + \ell(x, y) \geq \ell(P_v)$.

(b) In HW#1 you were given the graph of world airports and flight routes. In this problem we use the same data, but this time use distance measures in *miles* rather than in hops, using positions of the airports in the data given as (latitude,longitude)-pairs. Assuming the earth is a perfect sphere with radius 6371km $\approx$ 3959mi, for example, the Wikipedia article on *Great-circle distance* gives a formula:

> Let ϕ_s, λ_s; and ϕ_f, λ_f be the geographical latitude and longitude of two points ... and $\Delta\phi, \Delta\lambda$ their absolute differences; then $\Delta\widehat{\sigma}$, the *central angle* between them, is given by the *spherical law of cosines*: $\Delta\widehat{\sigma} = \arccos\left(\sin\phi_s \sin\phi_f + \cos\phi_s \cos\phi_f \cos\Delta\lambda\right)$. The distance d, i.e. the *arc length*, for a sphere of radius r and $\Delta\widehat{\sigma}$ ... is then $d = r\,\Delta\widehat{\sigma}$.

These distances are provided both as a distance matrix in `US_great_arc_distance.csv`, and an equivalent adjacency list representation in `US_adjacency_distance.csv`.
With these distances, the shortest-path distances between LAX and other major airports in the US (in miles), and print the *shortest-path tree* of *all shortest paths from LAX*, with distances. If you prefer, you can plot the shortest path tree, using (longitude,latitude) as (x,y) positions.
(Remember that this shortest path tree, using the terminology in section 4.4 of the [KT] text, is the set of edges selected by Dijkstra's algorithm when growing the set S, or equivalently, the union over all nodes v (except the source node s) of edges in the shortest path P_v from s to v.)

5. **Minimum Spanning Tree**

(a) Prove or disprove: if all the edge weights in a graph are distinct, then the MST is unique.

(b) Prove or disprove: in an undirected graph G, there can be a central node s for which the shortest path tree from s has lower total edge length than the minimum spanning tree.

(c) Using the distance values you derived in the previous problem, compute the Minimum Spanning Tree (MST) for the US airport network. Determine whether all the edge weights (distances, here) are distinct (however: all distance matrix values are integers).
Extract the path from Los Angeles (LAX) to Washington DC (IAD) through the MST. Compare this path with the shortest path to IAD in the shortest-path tree from LAX.

(d) Using the Maximal Separation Clustering approach described in section 4.7 of the text, find a clustering of the airports into four clusters using the MST. Specify the airports in each cluster. (Hint: this can be done visually.)

(e) A *congestion spanning tree* of an undirected graph G is a spanning tree in which the largest edge weight is minimum over all spanning tree of G. The value of the congestion spanning tree is the weight of its maximum-weight edge.

i. Argue that the every MST is also a congestion spanning tree.

ii. Given a linear-time algorithm that, given a graph G and an integer b, determines whether the value of the congestion spanning tree is at most b. In other words, determine whether the maximum edge cost c of the congestion spanning tree satisfies $c \leq b$.