— This is a pledged take-home final. Do not discuss the final with anyone except the course staff. You can consult your personal course notes and the course website, but no other resources are allowed. All other forms of assistance are prohibited. In case of doubt, ask me.

— The exam is due WED MAY 11 1159P, 2016. Submit early/often. *No late submissions will be accepted!* Every problem must take exactly 2 pages. Your submission should be exactly 8 pages.

— Submit PDF solutions to `https://church.cs.virginia.edu/16s-4102/`.

— Prove answers. You will be graded on *clarity*, *correctness*, and *precision*.

PROBLEM 1 *Search Reduces to Decision*

Let language $L = \{$ pairs (G, k) where $G = (V, E)$ is a graph that has an independent set of size $\geq k$ nodes$\}$. Suppose you are given an algorithm D_L that decides the language L. That is, when given an instance (G, k), algorithm $D_L(G, k)$ returns "yes" when graph G has an independent set of size at least k, and "no" otherwise. Design and analyze an efficient polynomial time algorithm that uses D_L to find the largest independent set for any graph G. Your new algorthm $F_L(G)$ should output S which is the largest independent set of G.

PROBLEM 2 *Dispatch*

As the 911 dispatcher, you receive the positions $(p_1, p_2, \ldots, p_n)$ of n injured people as well as a measure of the severity of each person's injuries $(s_1, \ldots, s_n)$. This severity measure s_i tells you that you must deliver patient i to one of the k hospitals in your region within s_i minutes. You can use a good traffic model $d_j(p_i)$ to compute how many minutes it takes to deliver someone from position p_i to hospital j. You would like to distribute the n patients across the k hospitals so that each hospital gets $\lceil n/k \rceil$ patients. Write an efficient algorithm that determines when this is possible or not. Analyze the running time.

PROBLEM 3 *Short paths*

Let $G = (V, E)$ be a directed graph with edge weights $w(e)$ and no negative cycles.

(a) Write one sentence that explains the variable $\text{ASHORT}_{i,j,k}$ used in the All-pairs shortest path algorithm discussed in class.

(b) State the run time of the All-pairs shortest path algorithm discussed in class.

(c) Consider the following algorithm.

```
NEWALLPAIRS($G, w$)
1  Add a new node $s'$ to $G$. Add edges of weight 0 from $s'$ to every vertex $v \in V$.
   Call this new graph $G'$.
2  Run BELLMANFORD($G', s'$) to produce shortest path lengths $\delta(s', v)$.
3  For each $e = (x, y) \in E$, set $w'(e) \leftarrow w(e) + \delta(s', x) - \delta(s', y)$
4  For each $v \in V$, run DIJKSTRA($G, v, w'$) to compute $\delta(v, w)$ for all $w \in V$.
5  Set $d_{v,w} \leftarrow \delta(v, w) - \delta(s', v) + \delta(s', w)$
```

We aim to analyze why this algorithm works. The first step is to argue that the new edge weights w' that are defined in step (3) are all positive.

Prove that for all $e \in E$, $w'(e) \geq 0$.

(d) This explains why we can use the fast DIJKSTRA algorithm with weight w' in step (4) to compute shortest paths from node $v \in V$ to all other nodes in the graph. However, we must argue that the shortest paths under w' and under w will be the same shortest path.

Prove that for any pairs of nodes $u, v \in V$, if p is a shortest path from u to v with respect to weight function w', then p is also a shortest path from u to v with respect to weight function w.

(e) What is the running time of NEWALLPAIRS in terms of V and E? When does this algorithm run faster than the All-pairs algorithm discussed in class?

PROBLEM 4 *2-pass shortest tours*

There are n cities that I want to visit on my private jet. City i is located at (x_i, y_i). The traveling salesman problem would ask you to visit all n cities with the shortest tour as possible; but alas, that problem is NP-complete, and our private jet pilot also objects to arbitrary paths. To make filing our flight plan easier, our pilot has instead asked us to start from city 1, fly only eastward until we hit city n, and then turn around and fly back to city 1. However, as we fly east, we can fly as far North or far South as we like; and similarly, as we return west, we can fly as far North or South as we like. Hence, we call it the *2-pass tour*. Since we pay for private jet travel by the hour, we want to find the shortest route subject to this constraint. Devise and analyze an efficient algorithm to compute our 2-pass shortest tour.

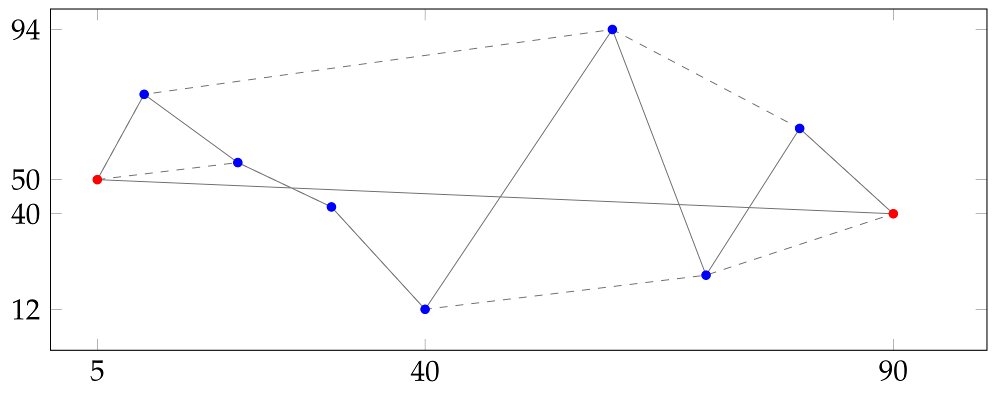

Figure 1: In this example, the start and end city are shown in red, and two possible 2-pass tours are shown in gray.