

Problem Set 5—Due Tuesday 3:15 PM, March 3

Note: when giving an algorithm we are always most interested in as fast a solution as possible (at least up to big $O()$, and good constants even better).

(20) Problem 1 Consider the problem of planning the fastest plane trip from Sacramento to some far off location (say Shangri-la). Assume that you know for each airport the time each flight is scheduled to leave from that airport and the time it is scheduled to arrive. You may further assume that each flight will be exactly on schedule and that all flights are for one 24 hour period (and the same schedule repeats each day).

(a) Describe an efficient algorithm to find the route which gets you to Shangralla in the least time assuming you arrive at the Sacramento airport at 10AM. You may assume that if you arrive at an airport at time t , you can depart on any flight leaving after time t .

(b) If there are a airports and f_i flights leaving airport i , what is the run time of your solution to part a)?

(20) Problem 2 Suppose that want to send goods by truck from a production facility in Sacramento to a warehouse in Chicago. You want to make as few trips as possible (you have only one truck), and the critical issue is that each road segment (the piece of a road between two intersections) has a weight limit for the truck. Thus, for example, if a road segment has weight limit 10 tons, then you either have to put at most 10 tons of goods in the truck, or not use that road segment. You know the weight limit for each road segment.

Your goal is to find a route which allows the heaviest truck load, even if it is (much) longer.

(a) Describe an efficient algorithm to find such a route.

(b) If there are r road segments (which connect two intersections) and j intersections (where you can change roads), what is the running time of your solution?

(c) Let W be the weight limit of the route found in part a) (so every road segment on your route has weight limit W or more and this was the largest possible value of a route). Now describe how to find the shortest route (from Sacramento to Chicago) such that every road segment has weight limit W or more. Here the *length* of a route is number of intersections you pass through.

(15) Problem 3 You have a beach house which you are going to rent for the rest of the year. You have solicited bids from interested renters of the form: s_i, f_i, r_i where s_i is the day they want to start renting, f_i is the day they want to finish renting, and r_i is the rent they are willing to pay for their stay.

Only one renter at a time can use the beach house (so if two requested times overlap, only one of them can rent the house). You want to find a set of renters such that no two stays overlap and the total rental income you get is as large as possible.

For example, if the bids are: (Jan 2, March 10, 100), (Jan 10, Feb 7, 60), (Feb. 9, March 25, 90), (March 10, April 10, 80), the best solutions takes the bids of the first and last renters

for a total rent of 180. Note: if a rental ends on day f_i the next rental can start on day f_i (like a hotel, rentals end at noon, but don't start until 4PM).

(a) Describe how to use variation on shortest paths to find the best rental choices. Give an efficient algorithm to find the best set of renters and give the run time of your solution if there are b bids.

(b) Now suppose that you have two types of renters: clean and messy. Assume that when a renter submits a bid they also specify which they are (assume everyone is one or the other). After the first rental you may need to clean the beach house before the next renter uses it. If we go from a messy renter to a clean one, you need three days for cleaning (so a new rental starts on day $f_i + 3$ or later). From messy to messy or clean to clean now takes one day. From clean to messy no cleaning is required. Again describe how to find the best set of renters and analyze the running time.

(20) **Problem 4** The Bellman-Ford (BF) algorithm executes Relax on each edge $n - 1$ times. While this may be required in rare graphs, usually we can find the answer with far fewer Relax steps. We now explore an approach to improve typical performance. We will keep the same outer **for** loop (line 2 on page 588) but improve the inner loop.

(a) Consider two consecutive calls to RELAX(u, v, W) for a given edge (u, v) . Argue that if $d[u]$ has the same value for both calls, then the second call to RELAX is useless.

(b) Discuss now to modify BF so that in iteration $i + 1$ of the outer **for** loop, we only execute RELAX(u, v, W) if $d[u]$ changed in iteration i .

(c) Consider the graph in figure 24.4. How many RELAX operations will your algorithm of part b) perform on this graph (if the order of relaxing within the inner loop might vary, break ties using the order given in the caption for 24.4: e.g. if you will relax both (t, x) and (t, z) you relax (t, x) first). Compare this to the number used by BF.

(20) **Problem 5** Consider the problem of finding a shortest path from a start node s to a destination node t in a directed graph with positive edge weights.

(a) We can run Dijkstra's algorithm from s to solve this. In this case when can we terminate the algorithm and be sure we have the shortest path from s to t ?

(b) An alternate solution uses so-called *bidirectional* search. In this case we simultaneously run Dijkstra's algorithm from both s and t . Specifically, we first scan (look at the neighbors of) s and update their $d[u]$ value. Then we look at the nodes with arcs *directed to* t and update their distance estimate from t . To avoid confusion, let $d_t[u]$ denote the values for distance estimates from t . (note that when considering distances "from" t we are actually considering the graph with the directions reversed: if there is a node x with an edge to t of length 3, then we can set $d_t[x]$ to 3. Thus if we find a node y such that $d[y] = 10$ and $d_t[y] = 15$, we know there is a path of length 25 going through y).

Next we consider the unscanned vertex with the smallest $d[]$ value, and update its neighbors' $d[]$ values. Then we switch back to our other search, and select the unscanned vertex with the smallest $d_t[]$ value, and update the $d_t[]$ values of its neighbors (note that by "unscanned" we mean with respect to the search from s or t respectively).

We continue going back and forth between the two searches until we can be sure we have found a shortest path.

- i) Given an example graph where the first vertex scanned in both searches is **not** on the shortest s, t path.
- b) Prove that when some vertex is scanned in both directions we have found the shortest path. Specifically, the minimum value $d[u] + d_t[u]$ over all vertices u is the shortest path distance from s to t (where $d_s[]$ $d_t[]$ are the labels in the s and t searches).