Introduction to Algorithms Day 32
Massachusetts Institute of Technology 6.046J/18.410J
Singapore-MIT Alliance SMA5503
Professors Erik Demaine, Lee Wee Sun, and Charles E. Leiserson Handout 32

Problem Set 9 Solutions

MIT students: This problem set is due in lecture on *Day 32*.

Reading: Chapters 22 and 24.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation instructor and time, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 9-1. Do exercise 22.2-7 on page 539 of CLRS.

Solution:

Run BFS on any node s in the graph, remembering the node u discovered last. Run BFS from u remembering the node v discovered last. $d(u, v)$ is the diameter of the tree.

Correctness: Let a and b be any two nodes such that $d(a, b)$ is the diameter of the tree. There is a unique path from a to b. Let t be the first node on that path discovered by BFS.

If the paths p_1 from s to u and p_2 from a to b do not share edges, then the path from t to u includes s so

$$\begin{aligned} d(t,u) &\geq d(s,u) \\ d(t,u) &\geq d(s,a) \\ d(t,u) &\geq d(t,a) \\ d(b,u) &\geq d(b,a). \end{aligned}$$

Since $d(a, b) \geq d(u, b)$, $d(a, b) = d(u, b)$.

If the paths p_1 and p_2 do share edges, then t is on p_1. Since u was the last node found by BFS, $d(t, u) \geq d(t, a)$. Since p_2 is the longest path, $d(t, a) \geq d(t, u)$. Thus $d(t, a) = d(t, u)$ and $d(u, b) = d(a, b)$.

$d(a, b) \geq d(u, v)$ and $d(u, v) \geq d(u, b)$ so all three are equal. Thus $d(u, v)$ is the diameter of the tree.

Exercise 9-2. Do exercise 22.3-12 on page 549 of CLRS.

Solution:

Run DFS once from each vertex. The graph is singly connected iff all edges are tree or back. Time is $O(VE)$.

Exercise 9-3. Do exercise 22.4-3 on page 552 of CLRS.

Solution: Run a modified version of DFS where one tests to see if the edge (u, v) leads to a gray node v which is not u's parent.

Alternatively, one could run a modified DFS which only allows an edge to be examined once. i.e., once (u, v) has been examined, eliminate the edge (u, v) or (v, u) (the same edge) and continue looking for the gray node in the DFS. Essentially, this is the same method as the first procedure.

If the graph does contain a cycle then $E = O(V)$ and the running time is $O(V)$. If G does contain a cycle, it will be found after at most V edges and vertices have been examined. Thus the running time is always $O(V)$.

Exercise 9-4. Do exercise 24.1-4 on page 591 of CLRS.

Solution: There is a simpler answer than this one ...

The idea is to find a node in the negative-weight cycle, to set its weight to $-\infty$ and to run a BFS-like procedure on that node setting the d values of reachable nodes to $-\infty$.

```
BELLMAN-FORD-NEGATIVE-CYCLE-INFINITY(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  for i ← 1 to |V[G]| − 1
 3       do  for each edge (u, v) ∈ E[G]
 4                do  RELAX(u, v, w)
 5  for each edge (u, v) ∈ E[G]
 6       do  if d[v] > d[u] + w(u, v)
 7             then d[v] ← −∞
 8                  BFS'(G, v)
```

The BFS' procedure is the same as the ordinary BFS except that whenever a node is placed in the queue, it also has its d value set to $-\infty$.

Exercise 9-5. Do exercise 24.3-6 on page 600 of CLRS.

Solution: Maintain an array A indexed from 0 to W. The $A[i]$ are doubly linked lists. If $i < W$ then $A[i]$ contains the vertices with distance i from S. $A[W]$ contains vertices not reachable from S. Extract the minimum element from A by searching the elements of A in turn for a nonempty list and extracting an element from that list. $O(W)$ time. Relaxing an edge can be done in constant time. The algorithm runs in $O(VW + E)$ time.

```
DIJKSTRA'(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S ← ∅
 3  A[W] ← V[G] − {s}
 4  A[0] ← s
 5  while Q ≠ ∅
 6       do  u ← EXTRACT-MIN(A)
 7           S ← S ∪ {u}
 8           for each vertex v ∈ Adj[u]
 9                do  RELAX(u, v, w)
```

Exercise 9-6. Do exercise 24.5-7 on page 614 of CLRS.

Solution:

The appropriate sequence of steps relaxes the $|V| - 1$ edges of the predecessor subgraph G_π in the order of a DFS or BFS. The proof is by contradiction. If this sequence of relaxations does not result in $d[v] = \delta(s, v)$, then there is a shorter path from s to v. However, this path must be of length $< |V|$, because there are no negative-weight cycles. Thus, the predecessor subgraph must

include this path which is doesn't because then the relaxations on the graph would have given the optimal solution.

Problem 9-1. Running in Boston

To get in shape, you have decided to start running to work. You want a route that goes entirely uphill and then entirely downhill so that you can work up a sweat going uphill and then get a nice breeze at the end of your run as you run faster downhill. Your run will start at home and end at work and you have a map detailing the roads with m road segments (any existing road between two intersections) and n intersections. Each road segment has a positive length, and each intersection has a distinct elevation.

(a) Assuming that every road segment is either uphill or downhill, give an efficient algorithm to find the shortest route that meets your specifications.

Solution: Dijkstra's algorithm solves the single source shortest-paths problem on a general graph with non-negative edge weights in $O(m+n\log n)$ time. In this problem we can actually do better and solve it in $O(m+n)$ time.

The difference is that we must go uphill before we go downhill. With this constraint we know we have somewhere along the optimal path there will be a highest point. Call it h. A consequence of this path being optimal is that there exist no other points for which the length of the best uphill path from home to the point plus the length of the best downhill path from the point to work is shorter than the best paths to and from h. So if we could find the best uphill path to each point and the best downhill path from each point, we can do a linear scan through the points to find the one with the smallest sum and this tells us the optimal path.

Now we just have two subproblems of finding the single source shortest uphill paths to each point and the single goal shortest paths downhill from each point. Consider the uphill problem. We can solve this by throwing away all downhill edges, then because the path always moves uphill there can no longer be any cycles (since there are no level edges). Therefore we are dealing with a DAG and we discussed in class how to find the single source shortest paths in a DAG in linear time. Basically this involves putting a topological ordering on the verticies and then just computing the best paths in order. We can similarly solve the same downhill problem.

The total run time is $O(m)$ to produce each DAG. Then we solve two single source shortest path in a DAG problems which each run in $O(m+n)$ time. Finally, traversing the verticies to find the optimal peak vertex takes $O(n)$ time. Thus the total time is $O(m+n)$.

(b) Give an efficient algorithm to solve the problem if some roads may be level (i.e., both intersections at the end of the road segments are at the same elevation) and therefore can be taken at any point.

Solution: In this case, we can no longer replace each edge with a directed edge that assures us that no cycles exist. Therefore our best method in this general case is to use Dijkstra's algorithm after transforming the graphs as above replacing each level road with two directed arrows.

Problem 9-2. Karp's minimum mean-weight cycle algorithm

Let $G = (V, E)$ be a directed graph with weight function $w : E \to \mathbb{R}$, and let $n = |V|$. We define the ***mean weight*** of a cycle $c = \langle e_1, e_2, \ldots, e_k \rangle$ of edges in E to be

$$\mu(c) = \frac{1}{k} \sum_{i=1}^{k} w(e_i) \ .$$

Let $\mu^* = \min_c \mu(c)$, where c ranges over all directed cycles in G. A cycle c for which $\mu(c) = \mu^*$ is called a ***minimum mean-weight cycle***. This problem investigates an efficient algorithm for computing μ^*.

Assume without loss of generality that every vertex $v \in V$ is reachable from a source vertex $s \in V$. Let $\delta(s, v)$ be the weight of a shortest path from s to v, and let $\delta_k(s, v)$ be the weight of a shortest path from s to v consisting of *exactly* k edges. If there is no path from s to v with exactly k edges, then $\delta_k(s, v) = \infty$.

(a) Show that if $\mu^* = 0$, then G contains no negative-weight cycles and $\delta(s, v) = \min_{0 \le k \le n-1} \delta_k(s, v)$ for all vertices $v \in V$.

Solution: If there were a negative-weight cycle, then $\mu^* < 0$ because the minimum would have to be negative, therefore there are no negative weight cycles. Given that there are no negative weight cycles, then the shortest path will not take any cycles and can only be at most $n - 1$ edges long.

(b) Show that if $\mu^* = 0$, then

$$\max_{0 \le k \le n-1} \frac{\delta_n(s, v) - \delta_k(s, v)}{n - k} \ge 0$$

for all vertices $v \in V$. (*Hint:* Use both properties from part (a).)

Solution: First, we know $n - k$ is strictly positive because $k \le n - 1$. Then $\delta_n(s, v) - \delta(s, v) \ge 0$ because the shortest path when no negative weight cycles exist is going to cost more with n nodes than the shortest path with fewer nodes that is the actual shortest path.

(c) Let c be a 0-weight cycle, and let u and v be any two vertices on c. Suppose that the weight of the path from u to v along the cycle is x. Prove that $\delta(s, v) = \delta(s, u) + x$. (*Hint:* The weight of the path from v to u along the cycle is $-x$.)

Solution: We know $\delta(s, u) \le \delta(s, v) + x$ because the shortest path from s to u might use v. Alternatively we also know that $\delta(s, v) \le \delta(s, u) - x$ because the shortest path to v from s might go through u and around the zero weight cycle for a cost of $-x$. Therefore with these two inequalities we know $\delta(s, v) = \delta(s, u) + x$.

(d) Show that if $\mu^* = 0$, then there exists a vertex v on the minimum mean-weight cycle such that

$$\max_{0 \le k \le n-1} \frac{\delta_n(s,v) - \delta_k(s,v)}{n-k} = 0 \; .$$

(*Hint:* Show that a shortest path to any vertex on the minimum mean-weight cycle can be extended along the cycle to make a shortest path to the next vertex on the cycle.)

Solution: Get to the cycle along some shortest path and then extend the path along the cycle to make a shortest path of length n. If v is the vertex we end up at, then $\delta_n(s,v) = \delta(s,v)$. Then since for we took the shortest possible path to the cycle, there cannot exist any shorter path to the node with fewer steps, only equal path lengths.

(e) Show that if $\mu^* = 0$, then

$$\min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s,v) - \delta_k(s,v)}{n-k} = 0 \; .$$

Solution: We know that there exists some vertex with a maximum difference of 0, and all differences are greater than 0, so the minimum must be 0.

(f) Show that if we add a constant t to the weight of each edge of G, then μ^* is increased by t. Use this to show that

$$\mu^* = \min_{v \in V} \max_{0 \le k \le n-1} \frac{\delta_n(s,v) - \delta_k(s,v)}{n-k} \; .$$

Solution: Adding t to each edge increases μ^* by t. It also increases $\delta_n(s,v)$ by nt and decreases $-\delta_k(s,v)$ by kt. Manipulate, and both sides increase by t and the equation is maintained. Thus, by picking $t = -\mu^*$, we can use the previous part.

(g) Give an $O(VE)$-time algorithm to compute μ^*.

Solution: Compute $\delta_k(s,v)$ for $k = 0, 1, \ldots, n$ in $O(VE)$ time by evaluating the recurrence $\delta_{k+1}(s,v) = \min_u \delta_k(s,u) + w(u,v)$. In $O(V^2)$ time, determine the minimum of the maximum of the fraction.