BU CLA CS 530: Analysis of Algorithms
Spring 1999

Selected Solutions for Problem Set 5

Problem 1. (Exercise 25.1-4, page 526, in [CLR])

Whenever RELAX sets π for some vertex v, it also reduces $d[v]$. Thus if $\pi[s]$ gets set to a non-NIL value, $d[s]$ is reduced from its initial value of 0 to a negative number. But $d[s]$ is the weight of some path from s back to s, which is a cycle including s. Thus there is a negative-weight cycle.

Problem 2. (Exercise 25.3-3, page 535, in [CLR])

As explained in the lecture (also proved in Lemma 25.12): For any vertex v, $d[v]$ attains its final value after k iterations of the main loop in the BELLMAN-FORD algorithm, where k is the length (number of edges) of any shortest-weight path from the source to v.

Hence, after m passes, BELLMAN-FORD can terminate. The problem is that we do not know m in advance (and that is why we cannot use m as one of the input arguments), so we cannot make the algorithm loop exactly m times and then terminates. But if we just make the algorithm stop when nothing changes any more, it will stop after $m+1$ iterations (i.e. after one iteration that makes no changes). The desired modification of BELLMAN-FORD is:

```
BELLMAN-FORD' (G, w, s)
    INITIALIZE-SINGLE-SOURCE (G, s)
    changes ← true
    while changes = true
          do changes ← false
             for each edge (u, v) ∈ E[G]
                 do RELAX' (u, v, w)

RELAX' (u, v, w)
    if d[v] > d[u] + w(u, v)
       then d[v] ← d[u] + w(u, v)
            pi[v] ← u
            changes ← true
```

The test for a negative-weight cycle (based on there being a vertex v for which $d[v]$ changes if another relaxation step is done after $|V|-1$ iterations) has been removed above, because this version of the algorithm will never get out of the **while**–loop unless $d[v]$ stops changing for every vertex v.

Problem 3. (Exercise 25.5-4, page 544, in [CLR])

We want to formulate the shortest-path weight from s to t as a linear program.

Let $\delta(u)$ be the shortest-path weigth from s to u. We want to find $\delta(t)$. δ must satisfy:

$$\delta(s) = 0 \ ,$$

$$\delta(v) - \delta(u) \leqslant w(u,v) \qquad \text{for all } (u,v) \in E \text{, by Lemma 25.3,}$$

where $w(u,v)$ is the weight of edge (u,v). Thus $x_v = \delta(v)$ is a solution for:

$$\begin{aligned} x_s &= 0 \ , \\ x_v - x_u &\leqslant w(u,v) \ . \end{aligned}$$

To turn this into a set of inequalities of the required form, replace $x_s = 0$ by $x_s \leqslant 0$ and $-x_s \leqslant 0$, i.e., $x_s = 0$. The constraints are now:

$$\begin{aligned} x_s &\leqslant 0 \ , \\ -x_s &\leqslant 0 \ , \\ x_v - x_u &\leqslant w(u,v) \ , \end{aligned}$$

which still has $x_v = \delta(v)$ as a solution.

However, δ is not the only solution for this set of inequalities. For example, if all edge weights are nonnegative, all $x_i = 0$ is a solution. To force $x_t = \delta(t)$ as required by the shortest-path problem, add the requirement to maximize (the objective function) x_t. This is correct because:

- $\max(x_t) \geqslant \delta(t)$ because $x_t = \delta(t)$ is part of one solution for the set of inequalities.
- $\max(x_t) \leqslant \delta(t)$ can be demonstrated by a technique similar to the proof of Theorem 25.17: Let p be a shortest path from s to t. Then by definition, $\delta(t) = \sum_{(u,v)\in p} w(u,v)$. But for each edge (u,v) we have the inequality $x_v - x_u \leqslant w(u,v)$, so that

  $$\delta(t) = \sum_{(u,v)\in p} w(u,v) \geqslant \sum_{(u,v)\in p} (x_v - x_u) = x_t - x_s \ .$$

  But $x_s = 0$, so that $x_t \leqslant \delta(t)$, as desired.

Note: Maximizing x_t subject to the above inequalities solves the single-pair shortest-path problem when t is reachable from s and there are no negative-weight cycles. But if there is a negative-weight cycle, the inequalities have no feasible solution, as demonstrated in the proof of Theorem 25.17; and if t is not reachable from s, then x_t is unbounded.

Problem 4. (Problem 26-1, page 576, in [CLR])

(a) Let T be the $|V| \times |V|$ matrix representing the transitive closure, such that $T[i,j]$ is 1 if there is a path from i to j, and 0 if not. Initialize T (when there are no edges in G) as follows:

$T[i,j] =$ **if** $(i = j)$ **then** 1 **else** 0

The matrix T can be updated as follows when an edge (u,v) is added to G:

```
TRANSITIVE-CLOSURE-UPDATE(u, v)
1.   for i ← 1 to |V|
2.        do for j ← 1 to |V|
3.              do if (T[i, u] = 1) and (T[v, j] = 1) then T[i, j] ← 1
```

This says that the effect of adding edge (u,v) is to create a path (via the new edge) from every vertex that could already reach u to every vertex that could already be reached from v. The above algorithm sets $T[u,v]$ to 1 because of the initial values $T[u,u] = T[v,v] = 1$.

The running time is $\Theta(V^2)$ because of the two nested loops.

(b) Consider inserting the edge (v_n, v_1) into the straight-line graph made up of the following edges:

$$(v_1, v_2) \;\; (v_2, v_3) \;\; (v_3, v_4) \ldots (v_{n-1}, v_n)$$

where $n = |V|$. Before (v_n, v_1) is inserted, only $n(n+1)/2$ entries in T are 1, namely all the entries on and above the main diagonal of T (why?). After (v_n, v_1) is inserted, the graph has a vertex in which every vertex can reach every other vertex, so all n^2 entries of T are 1. Hence, $n^2 - (n(n+1)/2) = \Theta(n^2) = \Theta(V^2)$ entries must be changed in T, so any algorithm to update the transitive closure must take at least $\Omega(V^2)$ time on this graph.

(c) The algorithm in part (a) would take $\Theta(V^4)$ time to insert all possible $\Theta(V^2)$ edges, so we need a more efficient algorithm in order for any sequence of insertions to take only $\mathcal{O}(V^3)$ total time.

To improve the algorithm, notice that the loop over j is pointless when $T[i, v] = 1$. That is, if there is already a path from i to v, then adding the edge (u, v) cannot make any new vertices reachable from i. The loop to set $T[i, j]$ to 1 for j such that there is a path from v to j is just setting entries that are already 1. Eliminate this redundant processing as follows:

```
TRANSITIVE-CLOSURE-UPDATE(u, v)
1.   for i ← 1 to |V|
2.       do if T[i, u] = 1 and T[i, v] = 0
3.           then for j ← 1 to |V|
4.                   do if T[v, j] = 1
5.                          then T[i, j] ← 1
```

This takes $\mathcal{O}(V^3)$ to update the transitive closure for any sequence of n insertions:

- There cannot be more than $|V^2|$ edges in G, so $n \leq |V^2|$.
- Summed over n insertions, the time to execute lines 1 and 2 is $\mathcal{O}(nV) = \mathcal{O}(V^3)$.
- Lines 3,4 and 5, which take $\Theta(V)$ time, are executed only $\mathcal{O}(V^2)$ times for n insertions. To see this, notice that lines 3-5 are executed only when $T[i, v] = 0$, and in that case line 5 sets $T[i, v] \leftarrow 1$, so the number of 0 entries in T is reduced by at least 1 each time lines 3-5 run. Since there are only $|V^2|$ entries in T, lines 3-5 can run at most $|V^2|$ times.
- Hence the total running time over n insertions is $\mathcal{O}(V^3)$.

Assaf Kfoury
Created: 99.02.18 Modified: 99.02.22