McGill University COMP360 Winter 2011 **Instructor: Phuong Nguyen**

Assignment 10 Solution

Question 1 (5pt) Recall the Bellman-Ford algorithm for the (general) Shortest Path problem. In this question you are asked to write a program that computes the total number of shortest st-paths in a given graph.

Formally, the input to your algorithm consists of a directed graph G and two vertices s and t in G. Each edge e of G is associated with a cost $c(e)$ that may be negative; however there is no cycle in G that has negative total cost. Your algorithm must output the total number of st-paths in G of minimum total cost. Note that you do not have to compute these paths.

Solution
The array: We modified the array used in the Bellman-Ford algorithm. Now each entry $A[v, i]$ in the array is a pair (c, k) where:

- c is the smallest cost of going from v to t using a path of length exactly i,
- and k is the total number of such paths.

Recurrence: Initially,

$$A[v, 0] = \begin{cases} (0, 1) & \text{if } v = t \\ (\infty, 0) & \text{if } v \neq t \end{cases}$$

For $i \geq 0$, for $v \in V$: Let $c = \min_{u \in V, (v,u) \in E} A\big(c(v, u) + A[u, i][1]$ Here $A[u, i][1]$ denotes the first in the pair $A[u, i]$. Let S be the set of neighbors u of v such that $c(v, u) + A[u, i][1] = c$. Then

$$A[v, i + 1] = \big(c, \sum_{u \in S} A[u, i][2]\big)$$

Program:

For $v \in V$ do $A[v, 0] \leftarrow (\infty, 0)$ End For
$A[t, 0] \leftarrow (0, 1)$
For i from 0 to $n - 1$ do
    For v in V do
        $c \leftarrow \min_{u \in V, (v,u) \in E} A\big(c(v, u) + A[u, i][1]$
        $k \leftarrow 0$
        For u in V such that $(v, u) \in E$ do
            If $c(v, u) + A[u, i][1] = c$ do $k \leftarrow k + A[u, i][2]$ End If
        End For

10. $A[v, i+1] \leftarrow (c, k)$

11. End For

12. End For

In line 5 above, the min is meant to be implemented as a for-loop.

To compute the total number of shortest st–paths we find the smallest cost amongst all $A[s, i]$, for $0 \le i \le n$, and sum up the corresponding numbers of paths.

1. $c \leftarrow \min_{0 \le i \le n} A[s, i][1]$
2. $k \leftarrow 0$
3. For i from 0 to n do
4. If $A[s, i][1] = c$ do $k \leftarrow k + A[s, i][2]$ End If
5. End For
6. return k

Again, in line above 1, the min is meant to be implemented as a for-loop.

Question 2 (10pt) (a) [5pt] Consider a directed grid graph G whose vertices are point (i, j) on the plane, for integers i, j: $0 \le i \le m$ and $0 \le j \le n$. The edges in G are horizontal and vertical grid edges that go from left to right and from bottom to top, together with diagonal edges in the direction from the lower-left corner $(0, 0)$ to the upper-right corner (m, n). In other words, the edges are:

$$\begin{aligned} ((i,j),(i,j+1)) \qquad & \text{for } 0 \le i \le m, 0 \le j \le n-1 \\ ((i,j),(i+1,j)) \qquad & \text{for } 0 \le i \le m-1, 0 \le j \le n \\ ((i,j),(i+1,j+1)) \qquad & \text{for } 0 \le i \le m-1, 0 \le j \le n-1 \end{aligned}$$

Each edge e of G is associated with a cost $c(e)$ which is a non-negative integer.

Given a path P in G from $(0, 0)$ to (m, n). Show how to modify the costs on the edges of G so that P is the unique minimum-cost path in G if and only if it is a minimum-cost path under the new cost function.

(b) [5pt] Give an algorithm that runs in time $\mathcal{O}(mn)$ and space $\mathcal{O}(m+n)$ that determines whether G has a unique minimum-cost path from $(0, 0)$ to (m, n). Justify the time and space complexity of your algorithm. Use (a) to argue that your algorithm is correct.

Solution

(a) Observe that the total number of edges in the path P is at most $m + n$. We first show how to define the new cost function which may have non-integer rational values. Then the costs can be scaled up simultaneously to become integers by multiplying with a common denominator.

We will increase the cost of each edge on P by a small amount, i.e. $\frac{1}{2(m+n)}$, and keep the cost of all other edges unchanged. Let c' denote the new cost function. We will prove now that P is the unique min-cost path under c iff P is a min-cost path under c'.

First, suppose that P is the unique min-cost path under c. Then note that the total increase in cost for P is at most

$$(m+n)\frac{1}{2(m+n)} = \frac{1}{2}$$

That is, $c'(P) \leq c(P) + \frac{1}{2}$. On the other hand, the total cost of all other paths do not decrease, i.e. $c'(P') \geq c(P')$ for all other paths P'. Because P is the unique min-cost under c we have

$$c(P') \geq c(P) + 1$$

for any other path P'. From these we have $c'(P') \geq c'(P) + \frac{1}{2}$. So P is a min-cost path under c'.

Second, suppose that P is not a min-cost path under c. This means that there is another path P' with $c(P') \leq c(P)$. Observe that the cost increase in P is the greatest, because no other path can contain all edges of P. Thus we have, in particular,

$$c'(P) - c(P) > c'(P') - c(P')$$

This gives

$$c'(P) > c'(P') + (c(P) - c(P') \geq c'(P')$$

Thus P is not a min-cost path under c'. QED

To define a new cost function that takes integer values, we let

$$c''(e) = 2(m+n)c'(e)$$

for all edges e.

(b) The algorithm is by defining a new cost function as above, then run a dynamic programming algorithm for computing the min-cost of going from $(0,0)$ to (m,n). Then compare this to the new cost of P: they are the same if and only if P is indeed the unique min-cost under the original cost function.

Question 3 (10pt) Consider the following problem. There are m machines $M_1, M_2, \ldots, M_m$. There are k types of job, and there are n jobs in total. (In general $n \geq k$, so there can be multiple jobs of the same type.) Each machine M_i is capable of processing a set of types of jobs, denoted by S_i. For example, if $S_2 = \{5, 9, 12\}$ then machine M_2 can process jobs of types 5, 9 and 12. Assume that each job requires one unit of time and must be processed by a single machine that is capable of processing it. Furthermore, each machine M_i has a total t_i units of time available. The problem is to schedule, whenever possible, all jobs on the machines in such a way that meet the described specification. Set up a flow network for solving this problem.

(a) Clearly specify the vertices, the edges, and the capacity on each edge of the network. Specify an algorithm for computing a maximum flow of the network.

(b) Give an algorithm that determines whether it is possible to schedule all jobs in such a way that satisfies the specification above, and if so, outputs such a schedule. (The output should be a list L_i for each machine M_i; this is the list of jobs that will be processed by the machine.)

(c) Prove that your algorithm in (b) is correct.

Solution

(a) The network has a vertex M_i for each machine M_i and T_j for each type j of jobs. If M_i can process a job of type T_j then there is an edge from M_i to T_j with capacity

$$c(M_i, T_j) = \min\left(t_i, n_j\right)$$

where n_j is the total number of jobs of type j.

There are also source s and sink t. For each machine M_i there is an edge from s to vertex M_i with capacity t_i, and for each job type j there is an edge from T_j to t with capacity n_j.

A maximum flow can be computed using Ford–Fulkerson algorithm.

(b) First run the Ford–Fulkerson algorithm to obtain a maximum flow f_{max}. If the value of this flow is less than n then we cannot schedule all n jobs. Otherwise, the list L_i of jobs for each machine M_i is obtained by looking at all vertices T_j such that $f_{max}(M_i, T_j) > 0$. The machine M_i will process $f_{max}(M_i, T_j)$ many jobs of type j.

(c) To prove the correctness of the algorithm, we argue that if there is a way of scheduling all n jobs, then the maximum flow value is n. In addition, if the maximum flow value is n, then there is a way of scheduling all n jobs.

First, suppose that we can schedule all n jobs. Then we can define a flow f of value n as follows.

- The flow on each edge (M_i, T_j) is the total number of jobs of type j that are processed by M_i.
- The flow on each edge (T_j, t) is n_j.
- The flow on each edge (s, M_i) is the total number of jobs scheduled on M_i.

It is easy to verify that this is a valid flow (i.e., it satisfies the Conservation and Capacity conditions). Also, the cut having t alone on one side has capacity exactly n. Thus the maximum flow is at most n. So the flow f above is a maximum flow.

Second, suppose that f is a maximum flow on the network, and f has value n. Then define a schedule as in (b). We can easily verify that it is a valid schedule (i.e., each machine M_i does not exceed its total time limit t_i, and for each job type j a total of n_j jobs are processed). QED