CS 381 HW 5

Ankur Dhoot

October 31, 2016

Q1

After running DFS on the graph of figure 22.6 :

dfs(q)
    dfs(w)
        dfs(s)
            dfs(v)
    dfs(t)
        dfs(y)
        dfs(x)
            dfs(z)
dfs(r)
    dfs(u)

dfs(q) discovers (in order) w, s, v, t, y, x, z
dfs(r) discovers u
Thus, the start and finish times are :
q(1, 16)
r(17, 20)
s(3, 6)
t(8, 15)
u(18, 19)
v(4, 5)
w(2, 7)
x(11, 14)
y(9, 10)
z(12, 13)

After reversing G and running DFS on the vertices in order of decreasing finishing time:

dfs(r) discovers r
dfs(u) discovers u
dfs(q) discovers q, y, t
dfs(x) discovers x, z

dfs(w) discovers w, v, s

Thus, the SCCs produced are {r}, {u}, {q, y, t}, {x, z}, {w, v, s}.

Q2

Let N_{ij}^k denote the number of different shortest paths from i to j for which all intermediate vertices are in the set {1,2,...k}. Using the same convention for w_{ij} as the book, we get

$N_{ij}^0 =$

$$\begin{cases} 1 & w_{ij} < \infty \\ 0 & w_{ij} = \infty \end{cases}$$

since the number of shortest paths between i and j using no intermediate vertices is 1 if there exists an edge between i and j and 0 otherwise.

For k > 0, we can split N_{ij}^k into three cases:

$N_{ij}^k =$

$$\begin{cases} N_{ij}^{k-1} & d_{ij}^{k-1} < d_{ik}^{k-1} + d_{kj}^{k-1} \\ N_{ij}^{k-1} + \delta_{ij}^k & d_{ij}^{k-1} = d_{ik}^{k-1} + d_{kj}^{k-1} \\ \delta_{ij}^k & d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

where $\delta_{ij}^k = N_{ik}^{k-1}$ x N_{kj}^{k-1}

The reasoning is as follows:

1. If the shortest path between i and j using intermediate vertices {1,2,...k} doesn't use vertex k (i.e $d_{ij}^{k-1} < d_{ij}^{k-1} + d_{kj}^{k-1}$), the the number of shortest paths between i and j using intermediate vertices {1,2,...k} is just the number of shortest paths using intermediate vertices {1,2,...k-1}.

2. If $d_{ij}^{k-1} = d_{ik}^{k-1} + d_{kj}^{k-1}$, then the number of shortest paths from i to j using intermediate vertices {1,2,...k} is the number of shortest paths using intermediate vertices {1,2,...k-1} plus the number of shortest paths going through intermediate vertex k. The number of shortest paths going through vertex k is $\delta_{ij}^k = N_{ik}^{k-1}$ x N_{kj}^{k-1} (number of shortest paths from i to k multiplied by number of shortest paths from k to j) Obviously, k cannot be an intermediate vertex on a path from i to k or k to j (else we'd have a negative cycle contrary to our hypothesis) so we can use the set of intermediate vertices {1,2,...k-1} to calculate δ_{ij}^k.

3. If $d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1}$, then the number of shortest paths from i to j is just the number of shortest paths that go through k which is δ_{ij}^k.

Note that we can drop the superscripts. Why? If having dropped the superscripts, we were to compute and store N_{ik} or N_{jk} before using these values to compute δ_{ij}, then we might have one of the following situations:

$\delta_{ij}^k =$

$$\begin{cases} N_{ik}^{k} \text{ x } N_{kj}^{k-1} \\ N_{ik}^{k-1} \text{ x } N_{kj}^{k} \\ N_{ik}^{k} \text{ x } N_{kj}^{k} \end{cases}$$

However, k cannot be an intermediate vertex on any shortest path from i to k (else negative cycle contrary to hypothesis), so the number of shortest paths from i to k using intermediate vertices {1,2,...k} is just the number of shortest paths from i to k using vertices {1,2,...k-1}. Thus, $N_{ik}^{k} = N_{ik}^{k-1}$. Similarly, $N_{kj}^{k} = N_{kj}^{k-1}$. Thus, we can drop the superscripts.

Algorithm 1 Floyd-Warshall with total number of shortest paths

```
function FLOYD-WARSHALL(W)
    n = W.rows
    D = W
    Initialize N (n_ij = N^0_ij as described above)
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                if d_ij < d_ik + d_kj then
                    n_ij = n_ij
                else if d_ij = d_ik + d_kj then
                    n_ij = n_ij + n_ik x n_kj
                else
                    n_ij = n_ik x n_kj
                end if
                d_ij = min(d_ij, d_ik + d_kj)
            end for
        end for
    end for
    return D, N
end function
```

The runtime is clearly O(V^3) due to the three for loops and all computation inside the innermost for loop is O(1) depending only on previously computed values. The space required is O(V^2) since we dropped the superscripts and now only store the V x V matrices N and D, compared to the common implementation which uses O(V^3) space to store the matrices computed at each iteration.