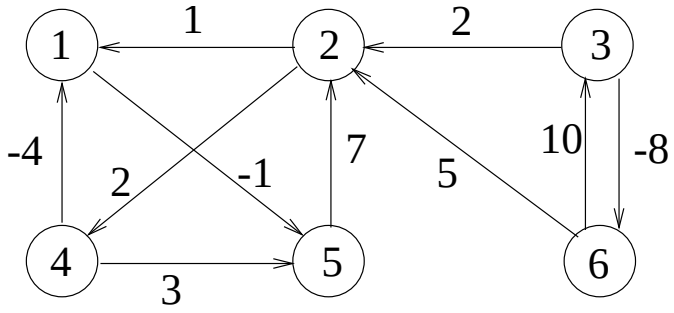
COMP 3711H – Fall 2016
Tutorial 9 - with Solution Sketches

1. Run the Floyd-Warshall algorithm on the weighted, directed graph shown in the figure. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.



Solution:

$$D^{(0)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(1)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(3)} = D^{(2)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^{(5)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^{(6)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

2 Let $G = (V, E)$ be Directed Acyclic Graph and s a vertex from which it's possible to get to all vertices. Show how to build a shortest (i.e., min-cost) path tree routed at s in $O(|V| + |E|)$ time.

Solution:

This is similar to the algorithm for finding a max-cost path for DAGs that we saw in class.

Let $n = |V|$ and $m = |E|$.

First topologically sort the vertices in $O(n+m)$ time (with s as the first vertex in the order) and relable them as $v_1, v_2, \ldots, v_n$ where v_i is the i'th vertex in the topological order. That is, if $(v_i, v_j) \in E$ then $i < j$

Next construct the in-adjacency lists. That is, for each vertex v_i we construct the list of all vertices that point to v_i. This construction can also be done in $O(n + m)$ time.

Let d_i be the cost of a shortest path from s to v_i. Then $d_1 = 0$ and, for $i > 1$,

$$\begin{aligned} d_i &= \min_{(v_j,v_i)\in E}(d_j + w_{j,i}) \\ &= \min_{j:(j<i) \text{ and } (v_j,v_i)\in E}(d_j + w_{j,i}) \end{aligned}$$

where $w_{j,i}$ is the cost of the edge (v_j, v_i). We saw the first equality when we first studied shortest paths. The 2nd equality follows from the properties of the topological order.

The algorithm is then to run through $i = 2, 3, \ldots, n$.
For each i run through V_i's in-adjacency list to calculate

$$\min_{j:(j<i)\,(\text{and})\,(v_j,v_i)\in E}(d_j + w_{j,i})$$

This algorithm takes $O(n + m)$ time.

3. (CLRS) Give an algorithm that takes as input a directed graph with positive edge weights, and returns the cost of the shortest cycle in the graph (if the graph is acyclic, it should say so). Your algorithm should take time at most $O(n^3)$, where n is the number of vertices in the graph.

Solution:

In $O(n^3)$ time run the Floyd-Warshall algorithm to find all the values $d_{i,j}$, the costs of the min cost path from v_i to v_j. The answer to the problem is

$$A = \min_{i,j}(d_{i,j} + d_{j,i})$$

Let OPT be the real cost of a minimum cost cycle. We need to show that $A = OPT$.

First note that every term $d_{i,j}+d_{j,i}$ is the cost of some cycle (start at i, follow the path of length $d_{i,j}$ to j and then follow the path of length $d_{j,i}$ back to i.). Since A is the minimum of some set of cycle costs, $OPT \leq A$.

Now suppose that we know some min-cost cycle C. *Let i', j' be any two points on the cycle. Let $d'_{i',j'}$ and $d'_{j',i'}$ be the costs on that cycle for the paths from i' to j' and from j' to i'. By definition*

$$d'_{i',j'} \geq d_{i',j'} \quad \text{and} \quad d'_{i',j'} \geq d_{i',j'}$$

Thus

$$OPT = cost(\mathrm{C}) = d'_{i',j'} + d'_{j',i'} \geq d_{i,j} + d_{j,i} \geq A$$

Thus, $A = OPT$.

The running time is $O(n^3)$ for the Floyd-Warshall algorithm and $O(n^2)$ for the rest, so the full running time is $O(n^3)$.

4. Assume that all edges have positive weight. Design an algorithm that will, for every pair of vertices, count the *number* of shortest paths between that pair.

Solution: If zero weight cycles existed then the number of shortest paths could be infinite. The reason for requiring all edges to have positive weight was to guarantee that no zero-weight cycles exist. This will guarantee that all shortest paths are simple and that there will thus be a finite number of shortest paths.

The short version of the solution is just to modify Floyd-Warshall slightly. Recall that Floyd-Warshall maintains a variable $d_{i,j}^{(k)}$ which is the length of the shortest path from i to j such that all intermediate vertices on the path (if any) are in the set $\{1, 2, ..., k\}$. Add another variable $N_{i,j}^{(k)}$ which will be the number of shortest paths from i to j such that all intermediate vertices on the path (if any) are in the set $\{1, 2, ..., k\}$.

Note that when running the code it checks if $pred[i, j] = k$. This occurs if

$$d_{i,j}^{(k)} = d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}.$$

If this is the case then set

$$N_{i,j}^{(k)} = N_{i,k}^{(k-1)} \cdot N_{k,j}^{(k-1)}$$

because any *shortest (i, k) path concatenanted with any shortest (k, j) path will give a shortest (i, j) path. If this does not occur for* any *k, then set*

$$N_{i,j}^{(k)} = N_{i,j}^{(k-1)}.$$

This gives a $O(n^3)$ algorithm. The space can be reduced from $O(n^3)$ down to $O(n^2)$ in the same way as in the regular Floyd-Warshall algorithm.

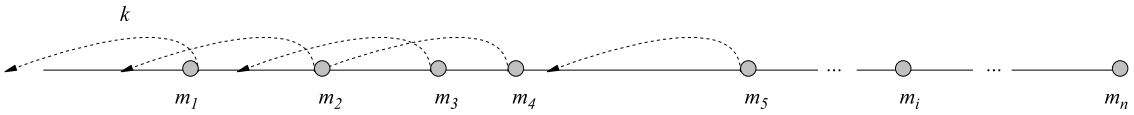
5. KFCC is considering opening a series of restaurants along the Highway. The n available locations are along a straight line; the distances of these locations from the start of the Highway are, in miles and in increasing order: $m_1, m_2, \ldots, m_n$. The constraints are as follows:

1. At each location, KFCC may open at most one restaurant.
   The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.

2. Any two restaurants should be at least k miles apart,
   where k is a given positive integer.

Give a dynamic programming algorithm that determines the locations to open restaurants which maximizes the total expected profit and analyze the running time of your algorithm.

Solution:



We define $T[i]$ to be the total profit from the best valid configuration using only locations from within $1, 2, \ldots, i$.

We also store $R[i]$ which is 1 if there is a restaurant at location i and 0 otherwise.

Case 1: Base case *If $i = 0$, then there is no location available to choose from to open a restaurant. So $T[0] = 0$.*

Case 2: General case *If $i > 0$, then we have two options.*

1. *Do not open a restaurant at location i*
   If no restaurant opened at location i, then the optimal value will be optimal profit from valid configuration using only location $1, 2, \ldots, i-1$. This is just $T[i-1]$.

2. *Open a restaurant at location i. Opening a restaurant at location i, gives expected profit p_i. After building at location i, the nearest location to the left where a restaurant can be built is c_i, where*

$$c_i = \max\{j \leq i \,:\, m_j \leq m_i - k\}.$$

   To obtain a maximum profit, we need to obtain the maximum profits from the remaining locations $1, 2, \ldots, c_i$. This is just $p_i + T[c_i]$.

Since these are the only two possibilities, we derive the following rule for constructing table T:

$$T[i] = \begin{cases} 0, & \text{if } i = 0 \\ \max\{T[i-1], p_i + T[c_i]\}, & \text{if } i > 0 \end{cases}$$

If $T[i] = T[i-1]$, then $R[i] = 0$; and $R[i] = 1$ otherwise.

Note: This immediately gives a $O(n^2)$ algorithm if we use brute force to find every c_i in $O(n)$ time. A little more thought shows that we could find $c_i = \max\{j : m_j \leq m_i - k\}$ in $O(\log n)$ time using binary search. This would give us a $O(n \log n)$ algorithm. The extra twist in this algorithm is that we will see soon that it's possible to calculate all of the c_i in $O(n)$ time, which will allow a $O(n)$ algorithm.

Note that for some values of i and c_i may not exist in which case, we set $c_i = 0$.

Algorithm to find optimal profit and locations to open restaurants.
Assumes c_i are known

Find-Optimal-Profit-And-Pos$(m_1, \ldots, m_n, p_1, \ldots, p_n, c_1, \ldots, c_n,)$

1: $T[0] = 0$
2: ***for*** $i = 1$ *to* n ***do***
3: *Not-Open-At-i* $= T[i-1]$; *Open-At-i* $= p_i + T[c_i]$
4: ***if*** *Not-Open-At-i* $>$ *Open-At-i* ***then***
5: $T[i] =$*Not-Open-At-i*; $R[i] =$*0*
6: ***else***
7: $T[i] =$*Open-At-i*; $R[i] =$*1*
8: ***end if***
9: ***end for***
10: ***return*** $T[n]$ *and* R;

Algorithm to compute $c_i = \max\{j : m_j \leq m_i - k\}$ for every i
uses fact that $0 = c_1 \leq c_2 \leq \cdots \leq c_n < n$.
Starts looking for c_i at $j = c_{i-1}$; increments j until finds correct value of c_i.

Note that algorithm runs in $O(n)$ time because line 5 can only be implemented $O(n)$ times!

Compute-ci$(m_1, \ldots, m_n, k)$

1: $c_1 = 0$;
2: **for** $i = 2$ to n **do**
3: $j = c_{i-1}$
4: **while** $(m_{j+1} \leq m_i - k)$ **do**
5: $j = j + 1$
6: **end while**
7: $c_i = j$
8: **end for**
9: **return** $c_1, \ldots, c_n$

Algorithm to report optimal locations to open restaurants

Report-Optimal-Locations$(R, c_1, \ldots, c_n,)$

1: $j = n$; $S = \emptyset$
2: **while** $j \geq 1$ **do**
3: **if** $R[j] = 1$ **then**
4: Insert m_j into S; $j = c_j$
5: **else**
6: $j = j - 1$
7: **end if**
8: **end while**
9: **return** S;

- **Compute-ci** takes $O(n)$ time to compute c_i for every i.
- Find **Optimal-Profit-And-Pos** takes $O(n)$ time to compute T and R.
- **Report-Optimal-Locations** takes $O(n)$ time to report the optimal locations for opening restaurants along the Highway.

Therefore, the overall running time is $O(n)$.