600.363/463 Algorithms - Fall 2013
Solution to Assignment 8

(40 points)

24.3-6 We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Solution. In order to get the most reliable path, we aim to find a path p such that the product of the probabilities on that path is maximized. Let s be the source and t be the terminal, and let $p = \langle v_0, v_1, \cdots, v_k \rangle$, where $v_0 = s$ and $v_k = t$,then

$$p = \arg\max \prod_{i=0}^{k} r(v_{i-1}, v_i) \tag{1}$$

This problem can be easily converted to a single-source shortest path problem by letting the weight on edge (u, v) be $w(u, v) = -\log r(u, v)$. Since logarithm will not change the monotonicity, and a minus logarithm will convert a minimization problem into a maximization problem, a shortest path p on the converted graph satisfying

$$p = \arg\min \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

will satisfy (1) as well.

We use Dijkstra's algorithm to solve shortest path problem on the converted graph. The initialization of weights takes $O(E)$ time, and the rest are the same as Dijkstra's algorithm. Wlog we assume that the graph is sparse, i.e. $E = o(V^2/\log V)$, and all vertices are reachable from source, hence the algorithm runs in $O((V + E)\log V + E) + O(E) = O(E \log V)$. ■

24.3-7 Let $G = (V, E)$ be a weighted, directed graph with positive weight function $w : E \to \{1, 2, \cdots, W\}$ for some positive integer W, and assume that no two vertices have the same shortest-path weights from source vertex s. Now suppose that we define an unweighted, directed graph $G' = (V \cup V', E')$ by replacing each edge $(u, v) \in E$ with $w(u, v)$ unit-weight edges in series. How many vertices does G' have? Now suppose that we run a breadth-first search on G' . Show that the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstras algorithm extracts the vertices of V from the priority queue when it runs on G.

Solution. Let V be the set of vertices in G and V' be the set of inner vertices in G', therefore G' has set of vertices $V \cup V'$. For any edge (u, v) in G, since there are correspondingly $w(u, v)$ unit length edges in G', there will be $w(u, v) - 1$ inner vertices in G'. Therefore the total number of inner vertices in G' are:

$$|V'| = \sum_{(u,v)\in E} (w(u, v) - 1) = \sum_{(u,v)\in E} w(u, v) - |E|$$

And the total number of vertices in G' is

$$|V \cup V'| = |V| + |V'| = \sum_{(u,v)\in E} w(u, v) + |V| - |E|$$

To show that the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstras algorithm extracts the vertices of V from the priority queue when it runs on G, let u, v be vertices in V
Assuming that v is extracted from Q after u in Dijkstra's algorithm, and knowing that no two vertices have the same shortest-path weights from source vertex s, then $d[u] < d[v]$. In G', since each edge (u', v') on G are extended to be of length $w(u', v')$, in the BFS u is visited in the $d[u]$ step and v is visited in the $d[v]$ step, therefore v is visited after u.
On the other way, assuming that v is visited in BFS after u, then $v.depth > u.depth$, which indicates that $d[v] > d[u]$, therefore in Dijkstra's algorithm, v is extracted after u. ■

25.2-7 Another way to reconstruct shortest paths in the Floyd-Warshall algorithm uses values $\phi_{ij}^{(k)}$ for $i, j, k = 1, 2, \cdots, n$ where $\phi_{ij}^{(k)}$ is the highest-numbered intermediate vertex of a shortest path from i to j in which all intermediate vertices are in the set $\{1, 2, \cdots, k\}$. Give a recursive formulation for $\phi_{ij}^{(k)}$, modify the FLOYD-WARSHALL procedure to compute the $\phi_{ij}^{(k)}$ values, and rewrite the PRINT-ALL-PAIRS-SHORTEST-PATH procedure to take the matrix $\Phi = (\phi_{ij}^{(k)})$ as an input. How is the matrix Φ like the s table in the matrix-chain multiplication problem of Section 15.2?

Solution. $\phi_{ij}^{(k)}$ is used to record the predecessor of the vertex j on the shortest path from i to j for which all intermediate vertices are in the set $\{1, 2, \cdots, k\}$. It performs similar to the s table in matrix-chain multiplication. The recursive formula for $\phi_{ij}^{(k)}$ is:

$$\phi_{ij}^{(k)} = \begin{cases} \text{NIL} & \text{if } k = 0 \\ k & \text{if } k > 0 \text{ and } d_{ik}^{(k-1)} + d_{kj}^{(k-1)} < d_{ij}^{(k-1)} \\ \phi_{ij}^{(k-1)} & \text{if } k > 0 \text{ and } d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \geq d_{ij}^{(k-1)} \end{cases}$$

The modified FLOYD-WARSHALL and PRINT-ALL-PAIRS-SHORTEST-PATH procedures are shown below.

Algorithm 1: FLOYD-WARSHALL(W)

```
1  n = W.rows;
2  D^(0) = W;
3  Φ^(0) = NIL;
4  for k = 1 to n do
5  |  let D^(k) = (d_ij^(k)) be a new n × n matrix;
6  |  let Φ^(k) = (φ_ij^(k)) be a new n × n matrix;
7  |  for i = 1 to n do
8  |  |  for j = 1 to n do
9  |  |  |  if d_ik^(k-1) + d_kj^(k-1) < d_ij^(k-1) then
10 |  |  |  |  d_ij^(k) = d_ik^(k-1) + d_kj^(k-1);
11 |  |  |  |  φ_ij^(k) = k;
12 |  |  |  end
13 |  |  |  else
14 |  |  |  |  d_ij^(k) = d_ij^(k-1);
15 |  |  |  |  φ_ij^(k) = φ_ij^(k-1);
16 |  |  |  end
17 |  |  end
18 |  end
19 end
```

Algorithm 2: PRINT-ALL-PAIRS-SHORTEST-PATH(Φ, i, j)

```
1  if i == j then
2  |  print i;
3  end
4  else if φ_ij == NIL then
5  |  print "no path from " i " to " j " exists".;
6  end
7  else
8  |  PRINT-ALL-PAIRS-SHORTEST-PATH(Φ, i, φ_ij);
9  end
```

■

II Recall that in the Ford-Fulkerson algorithm, the residual capacity of a path, $c_f(p)$, is dened as the minimum of the capacities of the edges on the path (page 719). Design an ecient algorithm for computing the maximum value of $c_f(p)$, maximized over all the s to t paths in the residual network. What is the speed of the algorithm?

Solution. In this problem we aim to find the augmenting path that has the maximum capacity. We modify Dijkstra's algorithm to solve this problem. Note the following facts:

(a) at each vertex v other than s, we want to compute the maximum flow that can reach it.

(b) the residue capacity of an augmenting path is bounded by the minimum residue capacities of the edges on this path.

Fact (1) implies that we record the maximum flow that reaches v, therefore instead of extract-min, we use extract-max to get the vertex with the maximum key value; in addition, Fact (2) implies that in the relax procedure, instead of adding the weight to the path, we pick the smaller one from the currently computed capacity and the residue. Formally, let S be the current set and $g(u)$ be the currently computed maximum flow at u, then $\forall v \in V - S$, compute $g(v) = \max_{u \in S}\{\min(g(u), res(u, v))\}$, pick the v with the largest $g(v)$ value.

Algorithm 3: MAX-AUG-PATH(G,w,s)

```
1  for each v ∈ G.V do
2  |   v.g = 0;
3  |   v.π = NIL;
4  end
5  s.g = ∞;
6  S = ∅;
7  Q = G.V;
8  while Q ≠ ∅ do
9  |   u =EXTRACT-MAX(Q);
10 |   S = S ∪ {u};
11 |   for each v ∈ G.adj[u] do
12 |   |   if u.g < res(u, v) then
13 |   |   |   temp = u.g;
14 |   |   end
15 |   |   else
16 |   |   |   temp = res(u, v);
17 |   |   end
18 |   |   if v.g < temp then
19 |   |   |   v.g = temp;
20 |   |   |   v.π = u;
21 |   |   end
22 |   end
23 end
```

Compare to original Dijkstra's algorithm, lines 2-6 corresponds to INITIALIZE-SINGLE-SOURCE, line 10 corresponds to EXTRACT-MIN, and lines 13-23 corresponds to RELAX, in which lines $13 - 18$ corresponds to the addition operation, which also takes $O(1)$ time. Hence the running time of this algorithm is $O((V + E) \log V)$.

Remark: If one considers the way of listing all the augmenting paths and then choose the path with the largest capacity, it will lead to an algorithm running in exponential time since searching for all the augmenting path takes exponential time. ■