College of Computer & Information Science
Northeastern University
Spring 2019

CS5800
Algorithms
3 April 2019

Solutions to Practice Problems - Midterm 2

Problem 1. Alternating red-blue paths

Let $G = (V, E)$ be a directed graph in which each vertex has been assigned a color, either red or blue. A directed path in G is called an *alternating red-blue path* if and only if no two consecutive vertices on the path have the same color. Give an efficient algorithm that determines for *all* pairs of vertices u, v in V whether v is reachable from u via an alternating red-blue path.

Analyze the worst-case running time of your algorithm. The more efficient your algorithm is in terms of its worst-case running time, the more credit you will get.

Answer: The algorithm is simple. Remove all red-red and blue-blue edges. For each vertex u, run depth-first search from u to determine all vertices reachable from u.

Running time is $O(n(n + m))$.

One could do better in practice by first computing strongly connected components and then traversing the DAG obtained. But that does not yield a better running time in the worst case.

Problem 2. (6 points) Analysis of a directed graph

You are given a directed graph G and an integer weight $w(v)$ for each vertex v. For each vertex u, define $R(u)$ to be the set of all vertices that are reachable from u (by a directed path) in G.

For each vertex u, define the *reach* of u to be the largest weight of a vertex in $R(u)$. For instance, if the set of vertices that u can reach in G is $\{x, y, z\}$ and the weights of x, y, and z are 3, 9, and 4, respectively, then the reach of u is 9.

Design an algorithm that computes the reach of v, for each v in G. Analyze the worst-case running time of your algorithm. The more efficient your algorithm is in terms of its worst-case running time, the more credit you will get.

Answer: We present two algorithms.

- **Algorithm 1:**
  1. For each vertex u: run a depth-first search from u and put all the vertices reached in a list $R(u)$.
  2. For each vertex u: compute reach of u to be the maximum weight of a vertex in $R(u)$.

  The running time of the algorithm is $\Theta(n(n + m))$.

- **Algorithm 2:**
  1. Find strongly connected components of G and the dag D of the strongly connected components.

2. For each C in D: set $r(C)$ to be the maximum weight of a vertex in C.
3. Compute topological ordering of D.
4. For C in reverse topological order: set $r(C)$ to be the maximum of $r(C)$ and $r(C')$, over all C' such that C has an edge to C' in D.
5. For each u in C: set the reach of u to be $r(C)$.

Running time of steps 1, 3, 4 is $\Theta(n+m)$. Running time of steps 2 and 5 is $\Theta(n)$. So total running time is $\Theta(n+m)$.

Problem 3. Minimum Spanning Trees

For each of the following claims, indicate whether it is true or false. Briefly justify your answers.

(a) If T is a minimum spanning tree of a weighted undirected graph G, then any minimum-weight edge of T is also a minimum-weight edge of G.

Answer: True. Let e_1 and e_2 be minimum weight edge of T and G, respectively. If $w(e_1) > w(e_2)$, then consider the cycle formed in T when we add e_2. Every edge on this cycle has weight greater than $w(e_2)$. Removing any of these edges yields a spanning tree of lesser weight, a contradiction.

(b) Let T be a minimum spanning tree of an undirected graph G with positive weights on edges. Assume that all edge weights are distinct. Let (u, v) be any edge of T and C be any cut of G containing (u, v) (i.e., u is in one side of the cut and v in the other side). Then, (u, v) is the minimum-weight edge of C.

Answer: False. Consider the graph consisting of 3 vertices a, b, and c, with edges (a, b) and (a, c) of weights 1 and 2, respectively. The entire graph is an MST since there is only one spanning tree. Let C be the cut with a on one side and $\{b, c\}$ on the other. The edge (a, c) is in the cut but not the smallest weight edge.

Problem 4: Reliable Communication Channel

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 v 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 s and t.

Answer: The solution involves adapting Dijkstra's algorithm for shortest paths with the following variations:

- The reliability of a path P is computed as the *product* of the reliability associated with each edge along the path.
- The algorithm will greedily extend the path with the maximum reliability.

Here is the outline of the algorithm:

Dijkstra's Algorithm (G, s, t):
Let S be the set of explored nodes
for $u \in S$ **do**
  Comment: Store a reliability value $R(u)$ and a predecessor node $\pi(u)$
  $R(u) \leftarrow 0\ \pi(u) \leftarrow NIL$
end
Comment: Initially $S \leftarrow s, R(s) \leftarrow 1$, and $\pi(s) \leftarrow NIL$ **while** $S \neq V$ **do**
  Select a node $v \notin S$ with at least one edge from S for which
  $R'(v) = \max_{(u,v):u \in S} R(u) * r(u, v)$ is maximum
  Add v to S
  $R(v) \leftarrow R'(v)$
  $\pi(v) \leftarrow u$
end
Comment: The most reliable path to node t is obtained by tracing back the predecessor nodes from t back to s
PRINT-PATH(G, s, t)

5. Longest common subsequence of three sequences

Give an efficient algorithm to determine the longest common subsequence of three given sequences of length m, n, and p, respectively. Analyze the worst-case running time of your algorithm.

Answer: It may be tempting to solve this by first finding the longest common subsequence, say S, of X and Y, and then finding the longest common subsequence of S and Z. This strategy does not always work.

- for $i = 0$ to m, for $j = 0$ to n: $L[i, j, 0] = 0$
- for $i = 0$ to m, for $k = 0$ to p: $L[i, 0, k] = 0$
- for $j = 0$ to n, for $k = 0$ to p: $L[0, j, k] = 0$
- for $i = 1$ to m:
  - for $j = 1$ to n:
    - for $k = 1$ to p:
      - If $X[i] = Y[j] = Z[k]$ then $L[i, j, k] = L[i-1, j-1, k-1] + 1$.
      - else $L[i, j, k] = \max\{L[i-1, j, k], L[i, j-1, k], L[i, j, k-1]$.

Running time is $\Theta(nmp)$.