Homework #6
Introduction to Algorithms/Algorithms 1
601.433/633
Spring 2018

Due on: Thursday, April 19, 5.00pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: Gradescope.
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

May 9, 2018

Problem 1 (15 points)

Let $G = (V, E)$ be a connected undirected graph with edge weights. For any spanning tree T of G, the **bottleneck** of T is the maximum edge weight in T. A **minimum bottleneck spanning tree (MBST)** of G is a spanning tree with minimum bottleneck over all spanning trees of G. The **bottleneck value** of a graph G is the bottleneck value of an MBST of G.

(a) (5 points)

Prove that a minimum spanning tree is also a MBST.
Answer: Let T be the MST, and consider the heaviest edge $(u, v) \in T$. We will argue that the bottleneck of any spanning tree is at least $w(u, v)$. In particular, let V_1 and V_2 be the vertices in the two connected subtrees produced when removing (u, v) from T . Since T is an MST, (u, v) must be the lightest edge crossing between V_1 and V_2. Every spanning tree must connect V_1 to V_2, so it must include some edge crossing the cut.

(b) (10 points)

Give an $O(E)$-time algorithm that given an input graph G and real value r, determines whether the bottleneck value of G is at most r. Prove the correctness and analyze the running time.

Answer: Choose an arbitrary starting vertex $s \in V$. Run a graph search(e.g.,BFS) from s, using only the edges with weights at most r. The bottleneck value is at most r if and only if the graph search reaches every vertex.

Said differently, construct a new graph $G' = (V, E')$, where $E' = \{(u, v) \in E | \omega(u, v) \leq r\}$. It is straightforward to construct G' in $O(V + E)$ time. Run BFS on G' from some arbitrary starting vertex. The rest is as above. $O(V + E)$ is $O(E)$ since the graph is connected.

Problem 2 (10 points)

Let $G = (V, E)$ be a graph with source x that represents a computer network topology. For each edge $e_i \in E$, its weight $w(e_i)$ represents the probability of failure of this edge $w(e_i) = p_i$, where $0 < p_i < 1$. All failures happen independently. Develop an algorithm to find the paths from x to all vertices with the lowest probability of failure in $O(|V|^2)$ time. Prove the correctness and provide running time analysis of your algorithm.

Answer:
Find-Lowest-Failure-Path(x,G)

- Create a new graph $G' = (V, E')$ such that $e_i \in E'$ if and only if $e_i \in E$, and such that $\omega(e_i) = -\log(1 - p_i)$ in G'.
- Run Dijkstra's algorithm on G' with source x.
- Return the path in G that corresponds to the shortest paths in G' according to Dijkstra's algorithm.

Proof of Correctness:
Let $F \in E$ be the x, y path in G with the lowest probability of failure. The goal of this algorithm is to find a path $F = e_{i_1} \ldots e_{i_k}$ such that each $e_{i_i} \in E$ and such that $e_{i_1} \ldots e_{i_k}$ forms a valid path from x to y where we have:

$$F = \text{argmin}_{e_{i_1} \ldots e_{i_k}} (1 - (1 - p_{i_1})(1 - p_{i_2})(1 - p_{i_{k-1}})(1 - p_{i_k})) \tag{1}$$

$$F = \text{argmin}_{e_{i_1} \ldots e_{i_k}} (1 - \prod_{e_{i_j} \in e_{i_1} \ldots e_{i_k}} (1 - p_{i_j})) \tag{2}$$

(1) comes directly from the problem definition. To make this notation a little more clear, (1) means that F is the valid x, y path $e_{i_1} \dots e_{i_k}$ such that the value of $(1 - (1 - p_{i_1})(1 - p_{i_2}) \dots (1 - p_{i_{k-1}})(1 - p_{i_k}))$ is at its minimum among all valid x, y paths $e_{i_1} \dots e_{i_k}$. (2) is then a restatement of (1) in more convenient notation. We can take out the 1 from the right side because it doesn't affect the value of F.

$$F = \text{argmin}_{e_{i_1} \dots e_{i_k}} \left(- \prod_{e_{i_j} \in e_{i_1} \dots e_{i_k}} (1 - p_{i_j})\right) \tag{3}$$

Switching the sign we can get:

$$F = \text{argmax}_{e_{i_1} \dots e_{i_k}} \left(\prod_{e_{i_j} \in e_{i_1} \dots e_{i_k}} (1 - p_{i_j})\right) \tag{4}$$

Because taking the log of a function does not change the locations of the functions minima or maxima, we can take the $\log$ of the term we are maximizing without affecting F. Therefore:

$$F = \text{argmax}_{e_{i_1} \dots e_{i_k}} \left(\log \prod_{e_{i_j} \in e_{i_1} \dots e_{i_k}} (1 - p_{i_j})\right) \tag{5}$$

By the product rule of logarithms, (5) is equivalent to:

$$F = \text{argmax}_{e_{i_1} \dots e_{i_k}} \left(\sum_{e_{i_j} \in e_{i_1} \dots e_{i_k}} \log(1 - p_{i_j})\right) \tag{6}$$

Finally, we can change the sign and get:

$$F = \text{argmin}_{e_{i_1} \dots e_{i_k}} \left(\sum_{e_{i_j} \in e_{i_1} \dots e_{i_k}} - \log(1 - p_{i_j})\right) \tag{7}$$

We can see then that the posed problem is identical to finding the path the minimizes the term in (7). (7) can be described as the shortest path in a graph where the graph weights are $-\log(1 - p_{i_j})$. The presented algorithm creates a graph G' from G where each edge with weight p_{i_j} is modified to have weight $-\log(1 - p_{i_j})$, then uses Dijkstra's algorithm to find the shortest path in this graph. The path found is the most reliable because (1) is mathematically equivalent to (7) and because Dijkstra's algorithm is correct. See the textbook for a proof of Dijkstra?s algorithm.

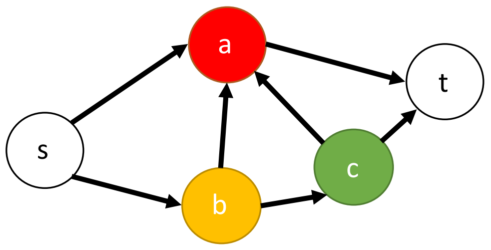
Running Time Analysis:
The time complexity of modifying each edge weight is $O(|E|)$, because each modification takes constant time. The run time of Dijkstra's algorithm depends on the

implementation of the priority queue that it uses, but in a naive implementation it is $O(|E|+|V|^2)$. See the textbook for a proof of Dijkstra's algorithm runtime. The total runtime is $O(|E|)+O(|E|)+O(|V|^2)=O(|V|^2)$ because the graph has at most $|V|^2$ edges.

Problem 3 (10 points)

Let $G=(V,E)$ be a directed and unweighted graph with nodes marked as s and t. All nodes (expect for s and t) in the graph are colored either red, yellow, or blue. Provide an algorithm which determines if there is a path (might be not simple path) from s to t which goes through both red and yellow vertices at least once, and if so outputs such a path. Full score will be given for running time $O(|V|+|E|)$. Prove the correctness and provide running time analysis.

For example, in the following graph, there exists such a path. Either $s\to b\to a\to t$ or $s\to b\to c\to a\to t$ will suffice.



Answer:

We are given a graph $G=(V,E)$, $s,t\in V$, and $f:V\to\{\mathcal{R},\mathcal{Y},\mathcal{B}\}$. Let R denote red vertices and Y denote yellow vertices.

In this problem, you need to consider two cases:

(1)$s\to R\to Y\to t$: go through a red vertex first and then a yellow vertex.

(2) $s\to Y\to R\to t$: go through a yellow vertex first and then a red vertex.

For case (1), we propose the following algorithm:

- First, use a depth first search from source s to find all paths from s to red vertices. Call the set of all such paths P_s, and the set of all such red nodes that are reachable from s as R'.
- Conduct a transpose of G to G', i.e. reverse all edges of G. Use a depth first search from t to find all paths from t to yellow vertices and store the reversed paths in P_t, and the set of such yellow nodes as Y'.

- Now we can get from s to R' and from Y' to t,so if we can find a path from some vertex of R' to some vertex of Y', we can construct a path from s to t. To do this, we will add a new vertex v_r that has an edge to every member of R' and a new vertex v_y which has an edge from every member of y'. If we can find a $v_r \to v_y$ path using depth first search, we can delete the first and last vertex to get a path from some member of R' to some member of Y'.

For case (2), we use the similar method as case (1).

Correctness:

The detailed correctness is omitted. The correctness relies on the correctness of DFS.

Running Time:

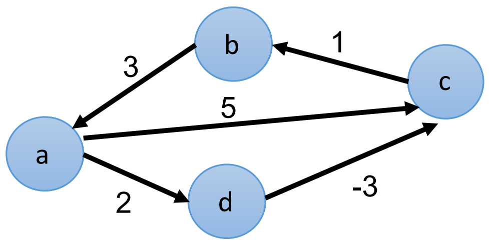
The step 1 and step 2 use DFS, which takes $O(|V| + |E|)$, and transpose of G is $O(|E|)$. Thus the running time is $O(|V| + |E|)$.

Problem 4 (15 points)

Let $G = (V, E)$ be a directed and weighted graph with weight function $w : E \to \mathbb{R}$. Design an $O(VE)$-time algorithm to find ***shortest destination weight*** $\delta^*(v)$ for each vertex $v \in V$, where $\delta^*(v) = min_{u \in V, u \neq v}\{\delta(u, v)\}$. Show the correctness and analyze the running time.

For example, in the following graph,

- for a, the shortest destination weight is 1.
- for b, the shortest destination weight is -2.
- for c, the shortest destination weight is -3.
- for d, the shortest destination weight is 2.



Answer:

- Method 1: Reverse the edges and use Bellman-ford from each v.
- Method 2: For each vertex v, add a new vertex s and an edge from s to v with very large weight (e.g. $>$ sum of all edge weights), and add edges from s to all other vertices with weights 0. Run Bellman-ford from s.

Correctness:
Omitted. Clearly the correctness of Bellman-ford.
Running Time:
Same as Bellman-ford.