4231 Homework 6

Eumin Hong (eh2890)

April 28, 2022

Problem 1

Problem 23-4 on Page 641: Alternative minimum-spanning-tree algorithms. For each of the three algorithms, either give a counterexample or prove that it always outputs a minimum spanning tree. Make sure your proof is written clearly and concisely. Also there is no need to describe efficient implementations of these algorithms.

23-4

In this problem, we give pseudocode for three different algorithms. Each on takes a connected graph and a weight function as input and returns a set of edges T. For each algorithm, either prove that T is a minimum spanning tree or prove that T is not a minimum spanning tree. Also describe the most efficient implementation of each algorithm, whether or not it computes a minimum spanning tree.

a. MAYBE-MST-A(G, w)

```
1: sort the edges into nonincreasing order of edge weights w
2: T = E
3: for each edge e, taken in nonincreasing order by weight do
4:     if T − {e} is a connected graph then
5:         T = T − {e}
6: return T
```

b. MAYBE-MST-B(G, w)

```
1: T = ∅
2: for each edge e, taken in arbitrary order do
3:     if T ∪ {e} has no cycles then
4:         T = T ∪ {e}
5: return T
```

c. MAYBE-MST-C(G, w)

```
1: T = ∅
2: for each edge e, taken in arbitrary order do
3:     T = T ∪ {e}
4:     if T has a cycle c then
5:         let e′ be a maximum-weight edge on c
6:         T = T − {e′}
7: return T
```

Problem 2

Problem 24-4 on Page 679: Gabow's scaling algorithm for single-source shortest paths.

24-4

A ***scaling*** algorithm solves a problem by initially considering only the highest-order bit of each relevant input value (such as an edge weight). It then refines the initial solution by looking at the two highest-order bits. It progressively looks at more and more high-order bits, refining the solution each time, until it has examined all the bits and computed the correct solution.

In this problem, we examine an algorithm for computing the shortest paths from a single source by scaling edge weights. We are given a directed graph $G = (V, E)$ with nonnegative integer edge weights w. Let $W = \max_{(u,v)\in E}\{w(u, v)\}$. Our goal is to develop an algorithm that runs in $O(E \lg W)$ time. We assume that all vertices are reachable from the source.

The algorithm uncovers the bits in the binary representation of the edge weights one at a time, from the most significant bit to the least significant bit. Specifically, let $k = \lceil \lg (W + 1)\rceil$ be the number of bits in the binary representation of W, and for $i = 1, 2, \ldots, k$, let $w_i(u, v) = \lfloor w(u, v)/2^{k-i}\rfloor$. That is, $w_i(u, v)$ is the "scaled-down" version of $w(u, v)$ given by the i most significant bits of $w(u, v)$. (Thus, $w_k(u, v) = w(u, v)$ for all $(u, v) \in E$.) For example, if $k = 5$ and $w(u, v) = 25$, which has the binary representation $\langle 11001\rangle$, then $w_3(u, v) = \langle 110\rangle = 6$. As another example with $k = 5$, if $w(u, v) = \langle 00100\rangle = 4$, then $w_3(u, v) = \langle 001\rangle = 1$. Let us define $\delta_i(u, v)$ as the shortest-path weight from vertex u to vertex v using weight function w_i. Thus, $\delta_k(u, v) = \delta(u, v)$ for all $u, v \in V$. For a given source vertex s, the scaling algorithm first computes the shortest-path weights $\delta_1(s, v)$ for all $v \in V$, then computes $\delta_2(s, v)$ for all $v \in V$, and so on, until it computes $\delta_k(s, v)$ for all $v \in V$. We assume throughout that $|E| \geq |V| - 1$, and we shall see that computing δ_i from δ_{i-1} takes $O(E)$ time, so that the entire algorithm takes $O(kE) = O(E \lg W)$ time.

a. Suppose that for all vertices $v \in V$, we have $\delta(s, v) \leq |E|$. Show that we can compute $\delta(s, v)$ for all $v \in V$ in $O(E)$ time.

b. Show that we can compute $\delta_1(s, v)$ for all $v \in V$ in $O(E)$ time.

Let us now focus on computing δ_i from δ_{i-1}.

c. Prove that for $i = 2, 3, \ldots, k$, we have either $w_i(u, v) = 2w_{i-1}(u, v)$ or $w_i(u, v) = 2w_{i-1}(u, v)+1$. Then, prove that

$$2\delta_{i-1}(s, v) \leq \delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1$$

for all $v \in V$.

d. Define for $i = 2, 3, \ldots, k$ and all $(u, v) \in E$,

$$\hat{w}_i(u, v) = w_i(u, v) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v)$$

Prove that for $i = 2, 3, \ldots, k$ and all $u, v \in V$, the "reweighted" value $\hat{w}_i(u, v)$ of edge (u, v) is a nonnegative integer.

e. Now, define $\hat{\delta}_i(s, v)$ as the shortest-path weight from s to v using the weight function $\hat{w}_i$. Prove that for $i = 2, 3, \ldots, k$ and all $v \in V$,

$$\delta_i(s, v) = \hat{\delta}_i(s, v) + 2\delta_{i-1}(s, v)$$

and that $\hat{\delta}_i(s, v) \leq E$.

f. Show how to compute $\delta_i(s, v)$ from $\delta_{i-1}(s, v)$ for all $v \in V$ in $O(E)$ time, and conclude that we can compute $\delta(s, v)$ for all $v \in V$ in $O(E \lg W)$ time.

Problem 3

Problem 25-2 on Page 706: Shortest paths in ϵ-dense graphs. Skip a). For a d-ary min-heap, INSERT takes time $O(\log_d n)$; EXTRACT-MIN takes time $O(d \cdot \log_d n)$; and DECREASE-KEY takes time $O(\log_d n)$. Check Chapter 6 and Problem 6-2 if you are interested in d-ary min-heaps. But for this problem, you may use these facts for free.

25-2

A graph $G = (V, E)$ is **ϵ-dense** if $|E| = \Theta(V^{1+\epsilon})$ for some constant ϵ in the range $0 < \epsilon \leq 1$. By using d-ary min-heaps (see Problem 6-2) in shortest-paths algorithms on ϵ-dense graphs, we can match the running times of Fibonacci-heap-based algorithms without using as complicated a data structure.

b. Show how to compute shortest paths from a single source on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(E)$ time. (*Hint:* Pick d as a function of ϵ.)

c. Show how to solve the all-pairs shortest-paths problem on an ϵ-dense directed graph $G = (V, E)$ with no negative-weight edges in $O(VE)$ time.

d. Show how to solve the all-pairs shortest-paths problem in $O(VE)$ time on an ϵ-dense directed graph $G = (V, E)$ that may have negative-weight edges but has no negative-weight cycles.

Problem 4

Problem 26.1 on Page 760: Escape problem.

26.1

An $n \times n$ ***grid*** is an undirected graph consisting of n rows and n columns of vertices, as shown in Figure 1. We denote the vertex in the ith row and the j th column by (i, j). All vertices in a grid have exactly four neighbors, except fo the boundary vertices, which are the points (i, j) for which $i = 1, i = n, j = 1$, or $j = n$.

Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in the grid, the ***escape problem*** is to determine whether or not there are m vertex-disjoint paths from the starting points to any m different points on the boundary. For example, the grid in Figure 1(a) has an escape, but the grid in Figure 1(b) does not.

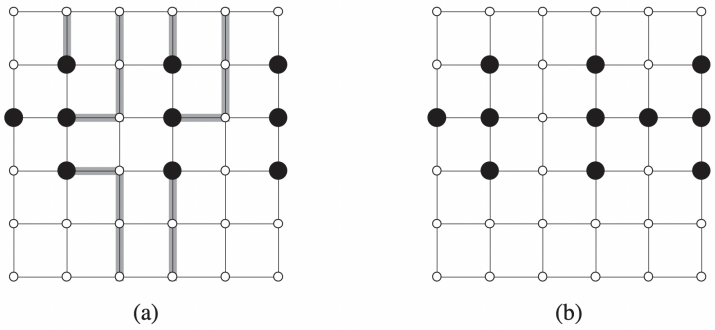

Figure 1: Grids for the escape problem. Starting points are black, and other grid vertices are white. **(a)** A grid with an escape, shown by shaded paths. **(b)** A grid with no escape.

a. Consider a flow network in which vertices, as well as edges, have capacities. That is, the total positive flow entering any given vertex is subject to a capacity constraint. Show that determining the maximum flow in a network with edge and vertex capacities can be reduced to an ordinary maximum-flow problem on a flow network of comparable size.

b. Describe an efficient algorithm to solve the escape problem, and analyze its running time.