Algorithm Design Assignment 2

Jianan Lin, 662024667, linj21@rpi.edu

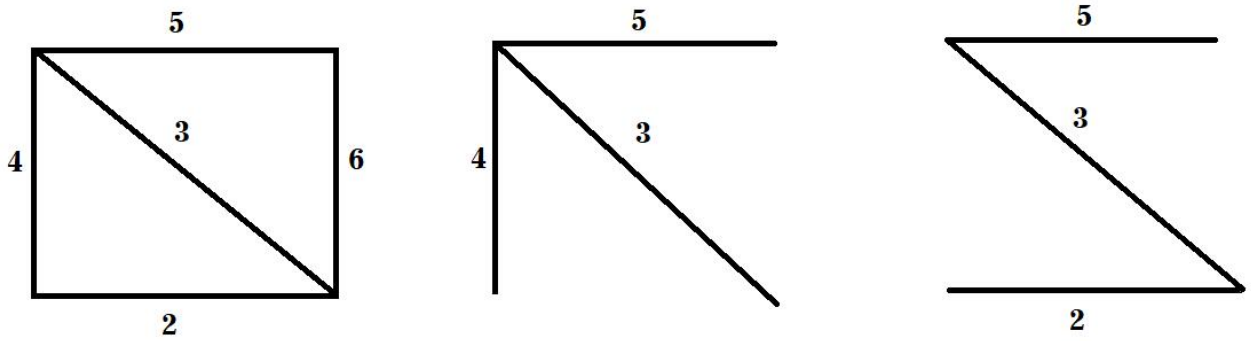
Problem 4.9. Let $G = (V, E)$ be a connected graph with n vertices, m edges with positive distinct costs. Let $T = (V, E')$ be a spanning tree of G. The bottleneck edge of T is the max edge cost. A spanning tree T of G is minimum bottleneck tree if there is no spanning tree T' of G with a less bottleneck edge cost.

(a). Is every minimum bottleneck spanning tree of G a minimum spanning tree?

(b). Is every minimum spanning tree of G a minimum bottleneck spanning tree?

Solution.

(a). No. The counterexample is shown below. The left is the entire graph with edge costs. The middle is a minimum bottleneck spanning tree, but it is not a minimum spanning tree. The right is a minimum spanning tree. In fact, a minimum spanning tree's cost is 10, but the middle tree's cost is 12.



(b). Yes. The proof is as follows.

Using contradiction proof, suppose a minimum spanning tree T with largest cost edge $e_0 = (p, q)$ of G is not a minimum bottleneck spanning tree. Suppose a minimum bottleneck spanning tree is T'.

Here, we use X to denote the set of vertices that can be reached from p without passing q in tree T (of course including p itself). In the same way, we use Y to denote the set of vertices that can be reached from q without passing p in tree T.

Notice that G is a connected graph, so there are some edges (one or more) connecting X and Y and we call it A. If we delete these edges, then there is no spanning tree. Therefore when we construct this minimum spanning tree T, we must include exactly one edge in A (If two or more, then there is a cycle, contradicted to the definition of a tree).

We should choose the one with the least cost (remember the cost is distinct) in A to construct T because if we change (p, q) into any other edge in A, we will still get a spanning tree. But notice (p, q)'s cost is larger than any edge in T', and T' obviously has an edge in A. Therefore (p, q) is not the edge with the least cost in A. So T is not a minimum spanning tree, which is contradicted with the conditions.

Therefore we prove the original proposition: every minimum spanning tree of G is a minimum bottleneck spanning tree.

Problem 6.4. There are n months for your group to work. In each month, you can choose one office from NY and SF with operating cost N_i and S_i respectively. If you move to another office, there is a moving cost M. Your goal is to design an algorithm to output the optimal sequence of the office in each month.

(a). Show the greedy algorithm in the textbook does not correctly solve this problem.

(b). Give an example in which every optimal plan must move at least thrice.

(c). Give an efficient algorithm which returns the cost of an optimal plan.

Solution.

(a). Let $M = 10$ and the operating cost is as follows, we can find that the optimal solution is "NY, NY, NY" with total cost 4, but the output of the algorithm is "NY, SF, NY" with total cost 23.

	Month 1	Month 2	Month 3
NY	1	2	1
SF	2	1	2

(b). Also let $M = 10$ and the operating cost is as follows. The only optimal solution is "NY, SF, NY, SF" with total cost 34 and 3 moves. Any solution with less than 3 moves will lead to a larger total cost because it must suffer from a situation with operating cost 100.

	Month 1	Month 2	Month 3	Month 4
NY	1	100	1	100
SF	100	1	100	1

(c). First we give the algorithm, then explain and analyze it.

Algorithm 1 DP

```
P, Q = N_1, S_1;
for all i ∈ [2, n] do
    T_1, T_2 = N_i + min(P, Q + M), S_i + min(P + M, Q);
    P, Q = T_1, T_2;
end for
return min(P, Q);
```

Since we use dynamic programming, we will explain the method. Let $OPT_N(i)$ and $OPT_S(i)$ denote the optimal total cost for the first i months, with working in NY and SF in month i respectively. Obviously $OPT_N(1) = N_1, OPT_S(1) = S_1$ Therefore we have the following relationship:

$$OPT_N(i) = N_i + \min\left(OPT_N(i-1), OPT_S(i-1) + M\right)$$
$$OPT_S(i) = S_i + \min\left(OPT_N(i-1) + M, OPT_S(i-1)\right)$$

Here, we should notice that we have exactly two choices in each month. Therefore when we find the recurrence, for example $OPT_i(N)$, then in the last month, we have two choices to work. If we worked in NY last month, then we do not need to have the moving cost this month. However, if we worked in SF last month, then we must add an extra moving cost. Then what we need to do is to choose one item with lower cost. When we want to get the ultimate optimal result, we only need to output $\min(OPT_N(n), OPT_S(n))$.

The time complexity is $O(N)$ obviously. And we do some optimization for the DP process: use variable P, Q instead of array OPT_N, OPT_S. Therefore the space complexity is $O(1)$.