HW4

Seyed Armin Vakil Ghahani
PSU ID: 914017982
CSE-565 Fall 2018
Collaboration with: Sara Mahdizadeh Shahri

September 24, 2018

Problem 1. Shortest Distances in a DAG

Solution

- (a) We know that because the graph is DAG, so it has a topological order that every edge is from left to right. Moreover, because vertex s has not any incoming edge and it has only outgoing edges, we can extract it from the topological order and insert it to the first of the list. It will not affect the main attribute of topological order that every edge is from left to right because this vertex has not any incoming edges and if we put it at the beginning of the list, all of its edges are from after this vertex and they are still from left to right. Suppose that the topological order is stored at $topo[1..n]$ array.
  Now, we can use dynamic programming to calculate the shortest distance from this vertex to any other vertices. Suppose an array $dis[1..n]$ that is going to store the shortest distance from s to i^{th} vertex in the *topo* array. By the definition, $dis[1]$ equals to the distance from s to s, and consequently, it is zero.

  For other vertices, we know that the shortest path from s to each vertex is going through one of its incoming vertices and the shortest distance to the incoming vertex plus the weight of the edge between them is the length of the path from s to this vertex that is going through this incoming vertex. Thus, if we get the minimum of these values, it will be the shortest distance from s to this vertex. If we calculate this value for each vertex from the leftmost vertex on the list to the rightmost vertex, the shortest distance to all of the incoming vertices for each vertex is calculated before and we can examine the shortest distance to this vertex according to the value of its incoming vertices.

- (b) To find the topological order of a DAG we should run a DFS that is $O(|V| + |E|)$. The following procedure to calculate the shortest distance to each vertex is a loop on every vertex. For each vertex, we are going through its incoming vertices and get the minimum value form its incoming vertices. Thus, the time complexity of this procedure is the sum of the incoming degree of each vertex that is $|E|$. So, the time complexity of this algorithm is $O(|V| + |E|)$.

- (c) Suppose that we have the correct value of shortest distance for the first vertex to i^{th} vertex in the topological order and we are going to calculate the shortest distance for the $i+1^{th}$ vertex. All of the incoming vertices of $i+1^{th}$ are from the vertices before it, and consequently, we know the shortest distance to each of them. Thus, the minimum path for the $i+1^{th}$ vertex can be calculated based on its incoming vertices as follows:

$$dis[i+1] = min_{j:topo[j]\rightarrow topo[i+1]} dis[j] + w[j][i+1]$$

So, if we start with node s and extend the vertices that their shortest distance is calculated from the leftmost vertex to rightmost vertex in the topological order, the shortest path to every node will be calculated.

Problem 2. Shortest Distances in a DAG

Solution

- (a) By the assumption that the root is in position 0 of the array, $Child(i,k)$ and $Parent(i)$ should be as follows:

$$Child(i,k) = i * d + k$$

$$Parent(i) = [\frac{i-1}{d}]$$

- (b) The number of vertices on height 1 is d and this number is multiplying by d. Hence, the number of vertices on height $h-1$ is d^{h-1}. However, the minimum number of vertices at height h is 1. As a result, the minimum number of vertices is

$$1 + \sum_{i=0}^{h-1} d^i = 1 + \frac{d^h - 1}{d-1}$$

- (c) The only thing that should be changed in the Heapify-up procedure on page 61 of textbook is the definition of parent function that should be like the parent definition in part a. The insert function should be changed as follows:

```
Insert(H, k) {
        H.size = H.size + 1
        key[A[H.size - 1]] = k
        Heapify-up(H, H.size - 1);
}
```

The running time of Heapify-up is changed to $\theta(log_d i)$ which i is the index of this element in the array. Hence, the time complexity of insert function is $\theta(log_d n)$ which n is the number of the elements in the heap.

- (d) The Heapify-down function should be changed as follows:

```
Heapify-down(H, i) {
        n = length (H)
        if (d*i+1 >= n)
                Terminate with H unchanged
        else {
                min_key = i
                for(j = d*i+1; j <= min(n-1, d*i+d))
                        if(key[H[min_key]] > key[H[j]])
                                min_key = j
        }
        if (min_key != i) {
                swap the array entries H [i] and H [min_key]
                Heapify-down(H, min_key)
        }
}
```

  Because each call of Heapify-down is $\theta(d)$, the time complexity of this function is the number of calling this function that is the height of the heap multiply by d. So, the time complexity of Heapify-down is $\theta(d * log_d n)$ which n is the number of elements in the array.

Problem 3. Modified lengths for MST and Shortest Paths

Solution

- (a) True. The order of edges will not changed by squaring the weight of the edges. As a result, the result of the Kruskal algorithm, which gives the minimum spanning tree by looking to the edges by their increasing order, will not change. In addition, because the weight of edges are distinct, the minimum spanning tree is unique and therefore the output of the Kruskal algorithm is this unique answer.

- (b) False. In the following example, the length shortest path between s and t in the left graph is 3. However, if we square the weight of edges, the minimum distance between s and t will become 8 and the shortest path is changed.

Problem 4. Greedily filling each truck is good

Solution I want to choose the 30% option for this question.