Advanced Algorithm HW8

Seungho Jang

November 2020

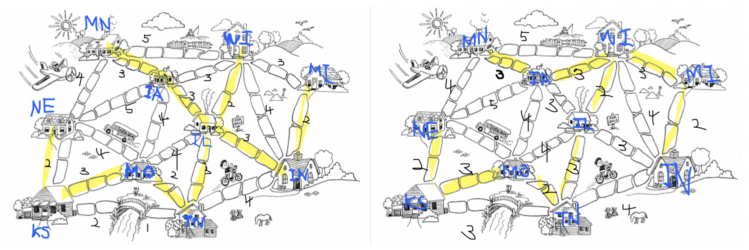
1 Q1

There are some assumption to solve this problem(muddy city problem).

1. Paving the bridge costs three times more than one stone
2. It's possible that there will be multiple minimal spanning tree.
3. This graph is an undirected graph

Based on the assumption, one of my intuition was to select and draw only with small weight(length) among states.

The figure below shows comparison between my intuition and implementation of spanning from the library(networkx).



(a) Intuition (b) Implementation

Figure 1: Muddy City Problem

The graph on figure 1-b) was redrawn or extracted from this source code : Muddy City Problem

Both of the graphs above show different trees, and it is hard to prove whether the intuition is correct as compared to the implementation. One way to prove is, to sum up, all the weights(lengths). The result of summing up all the weights on the intuition was (3+3+2+3+2+3+2+3+2=23), and the result of the implementation was (3+3+2+3+2+3+2+3+2=23). Therefore, the intuition is correct minimum spanning tree because of the assumption.

2 Q2

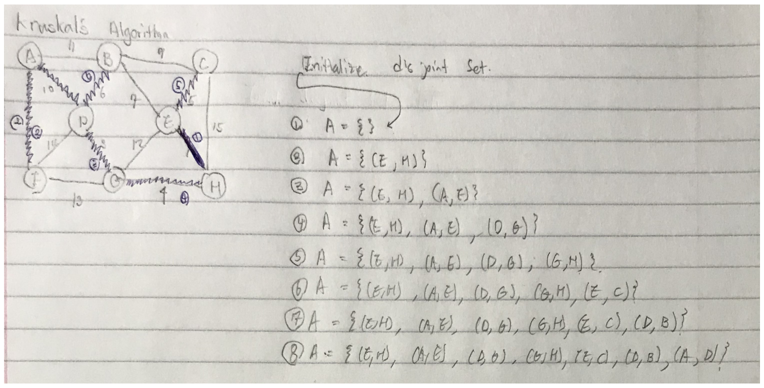
Comment : If I understood the problem correct, I think the question is asking how many arrangement of road can be.

Based on my understanding, there will be n(n-1) / 2. Because of how one vertex can go to n - 1 through n vertices. It's going to be n*(n-1) and repetition can be removed by dividing by 2.

3 Q3

The source code can be found on this link : Verification of the Kruskal's Algorithm As shown as the figures below, the answer was correct.

The work was done as shown in the figure 2.

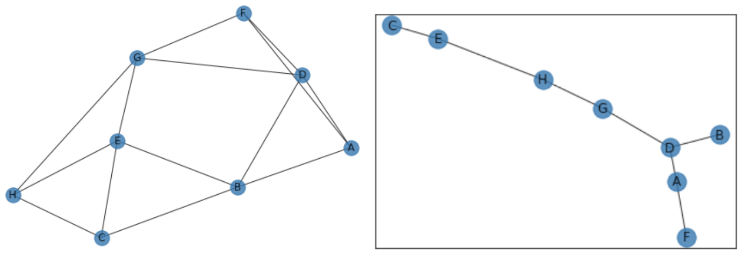

Kruskals Algorithm
A
B
C
D
E
F
G
H
11
9
10
6
7
5
15
14
8
12
1
13
4

Initialize disjoint Set.

① A = { }
② A = { (E, H) }
③ A = { (E, H), (A, E) }
④ A = { (E, H), (A, E), (D, G) }
⑤ A = { (E, H), (A, E), (D, G), (G, H) }
⑥ A = { (E, H), (A, E), (D, G), (G, H), (E, C) }
⑦ A = { (E, H), (A, E), (D, G), (G, H), (E, C), (D, B) }
⑧ A = { (E, H), (A, E), (D, G), (G, H), (E, C), (D, B), (A, D) }

Figure 2: Kruskal's Algorithm

The base graph and result of implementation was shown as figure below



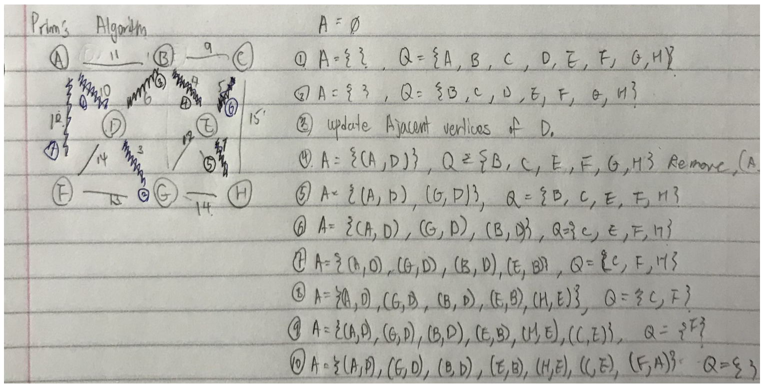
(a) Base Graph (b) Result of Kruskal's Algorithm

Figure 3: Kruskal's Algorithm

4 Q4

The source code can be found on this link : Verification of the Prim's Algorithm As shown as the figures below, the answer was correct.

The work was done as shown in the figure 4.



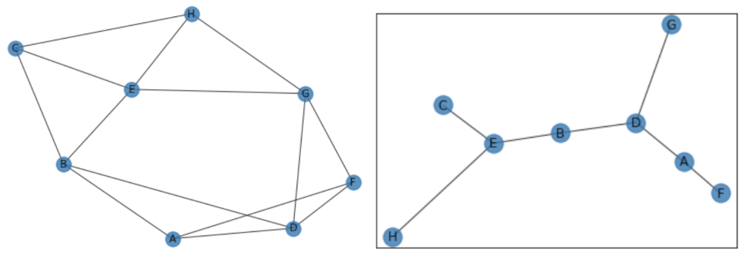
Prim's Algorithm

$A = \emptyset$

1. $A = \{\}$, $Q = \{A, B, C, D, E, F, G, H\}$
2. $A = \{\}$, $Q = \{B, C, D, E, F, G, H\}$
3. update Ajacent vertices of D.
4. $A = \{(A, D)\}$, $Q = \{B, C, E, F, G, H\}$ Remove (A
5. $A = \{(A, D), (G, D)\}$, $Q = \{B, C, E, F, H\}$
6. $A = \{(A, D), (G, D), (B, D)\}$, $Q = \{C, E, F, H\}$
7. $A = \{(A, D), (G, D), (B, D), (E, B)\}$, $Q = \{C, F, H\}$
8. $A = \{(A, D), (G, D), (B, D), (E, B), (H, E)\}$, $Q = \{C, F\}$
9. $A = \{(A, D), (G, D), (B, D), (E, B), (H, E), (C, E)\}$, $Q = \{F\}$
10. $A = \{(A, D), (G, D), (B, D), (E, B), (H, E), (C, E), (F, A)\}$ $Q = \{\}$

Figure 4: Prim's Algorithm

The result of implementation was shown as figure below



(a) Base Graph (b) Result of Prim's Algorithm

Figure 5: Prim's Algorithm

5 Q5

If we were to solve the time complexity of the Prim's algorithm using the binary min-heap, look at the following segments:

1. Segment 1: Initialization will takes O(V).
2. Segment 2(while loop) : The while loop will be executed until all the vertices explored ($|V|$ time). Then, the Extract-Min Operation takes O(lgV) by using binary heap. Then, the total time for calling Extract-Min will takes O($V * lgV$).

3. Segment 3(For loop) : In this for loop, it will explore all the edges(E), that means it will take O(E) times. Since the time it takes for Decrease-key for line 9 takes lgV, we can define lgV as t. Then the total time will be O(t*E).

Then the total time of Prim's algorithm using binary heap will be O($V*t+E*t$) where t is defined as lgV. Then it's going to be O($E*t$)(or $E*lgV$) because there are more edges than the vertices.

If we were to solve the time complexity of the Prim's algorithm using the Fibonacci heap, look at the following segments:

1. Segment 1: Initialization takes O(V) time.

2. Segment 2(while loop) : The while loop will be executed until all the vertices explored($|V|$ time). Then, the Extract-Min Operation takes O(lgV) by using binary heap. Then, the total time for calling Extract-Min will takes O(V*lgV).

3. Segment 3(For loop) : Since for loop will iterate through all edges, and the operation involved in line 11 is decrease-key, this will be O(1). That means the total time it will take in this segment is going to be O(E*1). Explanation are described below

The difference between Fibonacci heap and binary heap is that time complexity of Fibonacci are achieved by amortized analysis. What amortized analysis is if we have n sequence of operations(push, pop-min, decrease-key) starting from empty heap, push and decrease-key will do one operation at a time, then the rest of them will depend on pop-min for clean up.

Especially for the decrease key to explain the prim's algorithm using the Fibonacci heap, what it will do is to dump or slice the key(element) into root list. In other words, if key is decreased, the key itself will violate the heap property, so then dump it into root list, then do the pop-min(which it will do merge operation). Considering the decrease-key operation, moving that decreaed key(element) into root list takes O(1). That's why the segment 3 in the Fibonacci heap is differ from the binary heap.

So then the total time of the Prim's algorithm using the Fibonacci heap is O(E + Vt) where t is lgV, same as O(E + VlgV).

6 Q6

Yes, it is a true statement. In extreme case for dense graphs such complete graph, the number of edges is $(v(v-1)/2)$ which is almost V^2. In that case, the kruskal algorithm will show the time complexity of O(V^2*lg(V)) due to sorting all the weights in every edge. On the other hand, the O(V^2 + V*lg(V)) can be achieved by using prim's algorithm with the Fibonacci heap. For V^2 is just the constant value, then it become almost O(V*lgV). That means that this will prove the prim's algorithm is faster in dense graph.

7 Q7

I do not agree with the statement even though there are many analysis and assumption people can make. However, there are argumentation I can possibly make.

The reason we need to analyze the big-O and precisely choose an algorithm is to reduce the cost of advanced application of spanning tree. Also, since the network can be large and complex, so appropriate algorithms and analysis could allow us to minimize the latency among networks and reduce the time of execution.

8 Q8

Cut property can be applied for both Kruskal's algorithm and Prim's algorithm.

First we need to define what is cut property. Cut property is defined as following: Given any cut, minimum weight crossing edge is in the minimum spanning tree.

a) Kruskal's algorithm. Let's suppose that we have an edge e that has minimal cost(weight) and it doesn't belong to minimum spanning to tree. Also, another assumption would be all the edges are already sorted to find minimum spanning tree. Then if we were to add e into it, e must be placed before the specific edge that is connected to each vertices that were cut. In other word, there should be crossing edge that is connected between two vertices. Since this vertices is bigger value than the e due to e being minimum cost. That edge will be disconnected, and the new edge e will be part of minimum spanning tree.

b) Prim's algorithm. It is simpler to prove because prim's algorithm will select a random vertex and find the edge that is small weight(minimum cost). If e to be the smallest weight(minimal cost) and not belong to the minimal spanning tree, then another edge should be part of the minimal spanning tree. However, that will be contradiction because e is the smallest cost(weight) and it will make cyclic graph. Then cyclic means that it's not part of spanning tree, and it is not even minimum spanning tree.

Reference : The cut property