COMP2007 Assignment1, 4603688148, Liam Jay-Ling Chiang

by Liam Chiang

Submission date: 27-Aug-2017 09:56PM (UTC+1000)

Submission ID: 840178175

File name: COMP2007,_Assignment1,_4603688148,_Liam_Jay-Ling_Chiang.pdf (31.59K)

Word count: 861

Character count: 3571

Question1:

a).
First of all, we have to construct the graph. In order to enhance the efficiency, we use hash table to store the adjacent list. The key of the hash table will be each node in the graph, and the value of the hash table will be a Set that stores all of its adjacent nodes.

Example case:

Hash table (HT)	
key	value
a1	{a1, a2, a3}
a2	{a1, a2, a4, a5}
a3	{a1, a3, a4, a7}
a4	{a2, a3, a4, a5, a6}
a5	{a2, a4, a5}
a6	{a4, a6, a7, a8}
a7	{a3, a6, a7}
a8	{a6, a8}

Assuming a1 is the starting point and a8 is the ending point (the target node).
We get the adjacent nodes Set s1 = {a1, a2, a3} from the hash table by giving the key. And loop through the Set to get the adjacent nodes Sets of each item in s1, so that we have s2 = {a1, a2, a4, a5} and s3 = {a1, a3, a4, a7} since a2 and a3 are not the target node so we will do the minus operation on Set s2 - s1 = {a4, a5}, s3 - s1 = {a4, a7}. Therefore, these two Sets contain the nodes can not be directly reached by a1. And then we put {a4, a5} and {a4, a7} into a queue.

Next, we start from the first Set in the queue, which is {a4, a5}. By looping through this Set, we do HT[a4] - HT[a2] and HT[a5] - HT[a2] and we get {a3, a6} and an empty Set and put them into the queue again.

Repeatedly, we retrieve the first Set in the queue, which is {a4, a7}. Since a4 has already been visited, we directly start from a7. HT[a7] - HT[a3] = {a6} and put {a6} into the queue.

And again, we get {a3, a6} from the queue. we do minus operation on HT[a3 - a4] and HT[a6 - a4] and we get a7 and a8 respectively. a8 is our target, so we can say a1 and a8 have connection.

b).
First of all, building the hash table which records all the adjacent lists of each node in the graph takes O(n). According to the hash table we constructed. Because each edge would not be checked by more than one time by using the minus operation on the Sets. Therefore, the time complexity would be O(m). Since in total the time complexity would be O(m + n) for checking the connectivity between each pair of vertices. If there are q pairs, then the final time complexity would be O(q * (m + n)).

c).
Upper bound: According to the Algorithm, the upper bound will be O(n) for constructing the hash table, since there is only one edge between the nodes

Question 2.

a).
At the first step, when we are constructing the array of input, meanwhile we will construct the hash table for those edges, whose key is a frozen set of vertices {vertexId, vertexId}, the value is the array [vertexId, vertexId, weight].

At the second step, after we have constructed the array of the selected edges, then we do the for loop in this array to remove the selected edges from the whole provided ones and update the current MST weight.

At the third step, we create a priority queue and put the rest of the edges (no selected edges) into the priority queue, so that they are sorted in the ascending order according to their weight.

At the final step, we loop through the priority queue by getting the edge with least weight during the round and check whether the edge would produce a cycle in the current graph. If not, updating the hash table, which represents the current graph, with this edge and update the current MST weight

b).
For the first step that has been mentioned above, the time complexity is $O(n)$.

For the second step, the time complexity is also $O(n)$ because by using the hash table, every time we can remove the selected edge by $O(1)$

For the third step, the time complexity is $O(m*\log n)$ because each insert to the priority queue would be $O(\log n)$.

For the final step, each time before we are adding an edge to the graph, we will use DFS to determine whether this edge will produce cycle or not in the graph. Therefore, for each check, the time complexity is $O(|V| + |E|)$ ($= O(m + n)$).

Over all, the time complexity is $O(m \log n)$.

c).
The upper bound, the best scenario will be:

a. the graph has exactly only one vertex with exactly one input
b. the graph has exactly only one vertex with exactly one output
c. for the rest of the vertices, there are only one input and one output

Therefore, there is no branches in the graph. Hence, the graph is just like a line.

So there is no any cycle would be produced by adding edges at either the end of the line.

The final step's time complexity would be only $O(1)$.

COMP2007 Assignment1, 4603688148, Liam Jay-Ling Chiang

GRADEMARK REPORT

FINAL GRADE

62/0

GENERAL COMMENTS

Instructor

PAGE 1

Comment 1

Q1: 5 + 0 + 1 + 15 = 21

Q2: 12 + 0 + 4 + 25 = 41

Total: 62

Comment 2

5/5

Comment 3

We need to assume worst case rather than average case insertion time. Please explain how you got this? What about the edges?

Comment 4

How long does your minus operation take

Comment 5

0/5

This is an analysis of time complexity, but we were really looking for an analysis of correctness here

Comment 6

Assuming each check is O(1)

Comment 7

1/5

This answer is different from the one in the previous part, and there are problems with the calculations.

It might have been easier to use BFS like we did in class

Comment 8

12/15

Good effort! This could be made a lot clearer by stating that you are using a variant to Kruskal's algorithm first. Also, please try to be clear about exactly what you mean "by update the MST weight" etc.

Comment 9

How?

Comment 10

What do you mean by "update the hash table with this edge"?

Comment 11

We need to assume worst case and not average case, and we would also be making m insertions

Comment 12

0/5

Please make sure you prove that your algorithm gives the right answer here, rather than the time complexity

Comment 13

Aren't there m edges in there, though/

Comment 14

4/10

Good effort, but how does dealing with A affect the run time?