CS 330 – Spring 2019, Assignment 3

Problems due by 11:59PM on Tuesday, March 5

Please submit all homework problems as a single PDF on gradescope.

Problem 1 (20 pts)**.** *Chapter 4, Exercise 19, on p. 198.* Note that in your solution you will need to prove two things; that the graph sturcture you are looking for is appropriate to solve the problem and that your proposed algorithm is correct.

Answer:

The tree will have as many nodes as in grapg G and each parent node can have unlimited number of child nodes.

Define that all nodes are in set V and the initial tree T is empty.
Here is my Algorithm:
pick a random node v_0 and insert it to T as the root node
$while$ the size of T is less than the size of V:
find an edge $e*$ with the biggest bandwidth b_{e*} among all the edges that connects to any existing node v in T and has 1 node $v*$ outside of T.
insert the other node $v*$ into the tree T as a child node of the node v.
return T

Proof:

$Lemma$ 1: For every pair of parent node s and child node c, the edge e (p, c) is the path with the $best\ achievable\ bottleneck\ rate$.

Assume there is another path $p*$ that has a better $best\ achievable\ bottleneck\ rate$.

Since $b(p*) = min_{e* \in p*} b_{e*}$, suppose c should be connected to another parent node $s*$ and form a new edge $e*$ $(c, s*)$. Easily we known from algorithm $b_{e*} > b_e$.

By algorithm we can also known that among all the edges that connects to any existing node v in T, b_{e*} is the biggest bandwidth. Therefore, the new parent node $s*$ should always and already be inserted in T. Otherwise $p*$ has an edge with smaller bandwidth than the original path e and $b(p*) < b(e)$.

But from algorithm, we known that e is the edge with the biggest bandwidth b_e among all the edges that connects to any existing node v in T.

Contradiction. Proved.

Since T is a tree graph structure, there are no cycles. For each pair of 2 nodes a, b in T, there is only 1 unique path that connects them.

$Assumption$: for each pair of 2 nodes a, b in T, the path$p$ found in T is the path with $best\ achievable\ bottleneck\ rate$.

Proved by Contradiction:

Assume we have another path $p*$ that has a better $best\ achievable\ bottleneck\ rate$. That is, $b(p*) > b(p)$.

Assume p and $p*$ both share a part of path. Denote that part p_{shared}. Since $b(p*) > b(p)$, the bottleneck is not in p_{shared} otherwise we have a contradiction already.

Suppose p_{shared} ends in node sn. The next node for p is n and the next node for $p*$ is $n*$. By $lemma$ 1 and by algorithm we have $b(n) > b(n*)$ since the algorithm adds n to T. For now, $b(p) > b(p*)$. This continues for the rest nodes in p and $p*$. If there is at least 1 edge e in p that $b(e) < b(n*)$, it makes $b(p) < b(p*)$. Otherwise contradiction.

Suppose there is at least 1 edge e in p that $b(e) < b(n*)$, since our algrothim constructs p first, there must be at least 1 edge $e*$ in $p*$ that $b(e*) < b(e)$. However, this makes $b(p) > b(p*)$. Contradiction.

Therefore, this tree structure is approriate.

For correctness:

This algorithm constructs the tree T described above. Since the tree is proved to be appropriate, the algorithm is true.

For termination:

It inserts a node into T every time and ends as long as there is no free node left.

For runtime analysis:

Assume the original data is stored in adjcency list. We have n vertex and e edges.

It iterates n times. For each iteration, in the worst case we need to look up e edges.

Therefore, The time complexity is $O(ne)$.

Problem 2 (20 pts)**.** *Given a connected undirected graph $G(V, E)$ with positive weights on the edges. Assume that we found all shortest paths from a starting node s to the other vertices. (e.g. by using Dijkstra's or some other algorithm) The union of these paths forms a tree, which we will call the* shortest paths tree *or SPT at s. Note, that a graph may have multiple SPTs.*

1. *(10 pts.) Give an example of a weighted graph, whose minimum spanning tree (MST) differs from all its SPTs. In your answer make it clear what the edges and weights of the graph are. (e.g. legible picture or specify the edge list) Also specify what the MST and SPTs are. We recommend you find a small example as you need to list* **all** *possible trees.*

2. *(10 pts.) Prove that an MST and an SPT of G always have at least one edge in common.*

1)

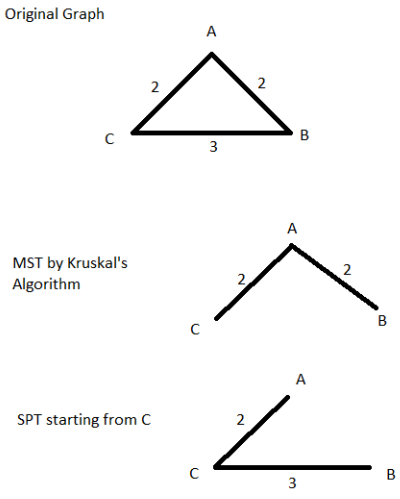

Figure 1:

2)

Suppose we have an MST M and an SPT S.

Suppose in S we have the starting node s. s is directly connected to at least 1 node. Denote that node by n. We have edge e (s, n). e is definitely in S.

If s has only 1 edge, since s is included in M, e must be in M. e is the common edge.
If s has multiple edges:
Since S starts from s, all these edges are in S.
Since s is included in M, among these edges, at least 1 will be selected. Call that edge $e*$, $e*$ is in M.
As mentioned before, $e*$ is also in S. So, $e*$ is the common edge.
Proved.

Problem 3 (20 pts)**.** *In this programming problem, we will be considering MSTs on complete, undirected graphs. A graph with n vertices is complete if all possible $\binom{n}{2}$ edges are present in the graph. Consider the following two types of graphs:*

- *Complete graphs on n vertices, where the weight of each edge is a real number chosen uniformly at random from $[0, 1]$.*
- *Complete graphs on n vertices, where the vertices are points chosen uniformly at random inside the unit square. (That is, the points are (x, y), with x and y each a real number chosen uniformly at random from $[0, 1]$.) The weight of an edge is the Euclidean distance between its endpoints.*

Your goal is to determine how the average weight of the minimum spanning tree grows as a function of n for each of these families of graphs. You will need to implement an MST algorithm and procedures that generate the appropriate random graphs. Run your program on at least five random graph instances for

$$n = 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192$$

and report the average for each value of n. (It is possible, that depending on your implementation you won't be able to run it for the largest graphs.) For each family of graphs, generate an appropriate figure that depicts your results. Clearly label the axes and think carefully about the most effective representation (should it be a bar chart, line chart, scatterplot, etc.). Also, interpret your results by giving (and plotting) two sample functions that characterize each of your depicted results. For example, your answer might be $f(n) = 2 \log n$, $f(n) = 1.5\sqrt{n}$, etc. Also, provide a few sentences of intuition for why the growth rate of your functions $f(n)$ are reasonable. Do this for both types of graphs separately.

As per usual, submit both the written answer (and charts) as well as your code as part of your pdf submission in Gradescope.

You will work on this assignment as individuals. However, the same collaboration policy applies as with any other assignment; you may discuss with classmates as long as the conclusions you write are your own and you disclose your collaboration.

Answer:
Code:

```
from random import *

def edges_generator_v1(n):
    # takes in the number of vertexes and generator the weight of all edges
    # for n vertexes we will have 1+2+3+...+n-2+n-1 = n(n-1)/2 edges
    # for type 1 only
    edges = []

    for i in range(n - 1):
        for j in range(i, n):
            l = uniform(0,1)
            edge = [l, i, j]
            edges.append(edge)
    return edges
```

```
15
def edges_generator_v2(n):
17      # points randomly pick from a unit square
        # for type 2 only
19      xy_list = [[uniform(0,1), uniform(0,1)] for i in range(n)]
        edges = []
21
        for i in range(n-1):
23          for j in range(1,n):
                first = xy_list[i]
25              next = xy_list[j]
                distance = ((first[0] - next[0])**2 + (first[1] - next[1])**2)**(0.5)
27              edge = [distance, i, j]
                edges.append(edge)
29      return edges

31  def is_cycle(nodesT, edge):
        # determines if by adding the edge into edgesT (a 2D list) we will create a cycle
33      # example of edgesT: [[1,2,3,4],[6,7]]
        mark1 = -1
35      mark2 = -1
        for i in range(len(nodesT)):
37          if edge[1] in nodesT[i]:
                mark1 = i
39          if edge[2] in nodesT[i]:
                mark2 = i
41          if mark1 is not -1 and mark2 is not -1:
                break
43      if (mark1 == mark2) and (mark1 is not -1):      # a cycle
            return True
45      elif (mark1 == mark2) and (mark1 is -1):     # a new edge
            nodesT.append([edge[1], edge[2]])
47      elif mark1 is -1:
            nodesT[mark2].append(edge[1])
49      elif mark2 is -1:
            nodesT[mark1].append(edge[2])
51      else:                                       # combine two clusters
            nodesT[mark1] += nodesT[mark2].copy()
53          del nodesT[mark2]
        return False
55
57  def minimum_spanning_tree(edges, num_nodes):
        # this function takes in a list of edges, number of nodes and returns the minimum weight
            of a MST. Kruskal's Algorithm.
        edges.sort()
59      # print(edges)
        nodes_in_T = [[edges[0][1], edges[0][2]]]     # a 2D list of all the nodes in clusters. e.
            g. [[1,2,3,4],[6,7]]
61      total_weight = edges[0][0]
        size_T = len(nodes_in_T[0])     # if a tree is formed, all nodes will be in 1 cluster. e.g
            . [[1,2,3,4,5,6,7]]
63
        for edge in edges[1:]:
65          if size_T < num_nodes:
                if is_cycle(nodes_in_T, edge):
67                  # print(nodes_in_T)
                    continue
69              else:
                    total_weight += edge[0]
71                  size_T = len(nodes_in_T[0])
                    # print(nodes_in_T)
73      # print(size_T)
```

```
    return total_weight


# now begin the stimulation
trials = [16, 32, 64, 128, 256]
for trial in trials:
    total_average_v1 = 0
    total_average_v2 = 0
    for i in range(100):
        edges_v1 = edges_generator_v1(trial)
        edges_v2 = edges_generator_v2(trial)
        mst_v1 = minimum_spanning_tree(edges_v1, trial)
        mst_v2 = minimum_spanning_tree(edges_v2, trial)
        total_average_v1 += mst_v1/trial
        total_average_v2 += mst_v2/trial
    average_v1 = total_average_v1/100
    average_v2 = total_average_v2/100
    s = "With n = " + str(trial) + ", \nthe average weight for type 1 is " + str(average_v1)
        + "\nthe average weight for type 2 is " + str(average_v2) + "\n"
    print(s)
```

MST.py

Here is some experiment of data from running my code:
With n = 16,
the average weight for type 1 is 0.06985280083570033
the average weight for type 2 is 0.1377528420051328

With n = 32,
the average weight for type 1 is 0.03629476252454324
the average weight for type 2 is 0.1046216101124182

With n = 64,
the average weight for type 1 is 0.01827287595010956
the average weight for type 2 is 0.07799533186712816

With n = 128,
the average weight for type 1 is 0.009012890117258456
the average weight for type 2 is 0.05612560076225726

With n = 256,
the average weight for type 1 is 0.00462717901592179
the average weight for type 2 is 0.03974057495104205

Chart by Excel and converted by Paint:

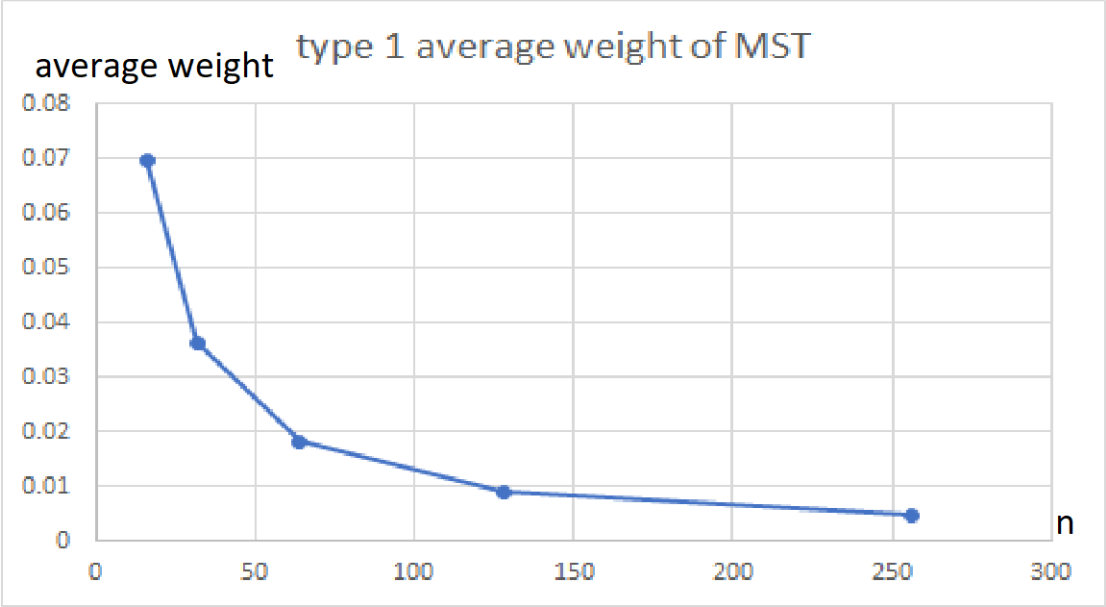

Figure 2:

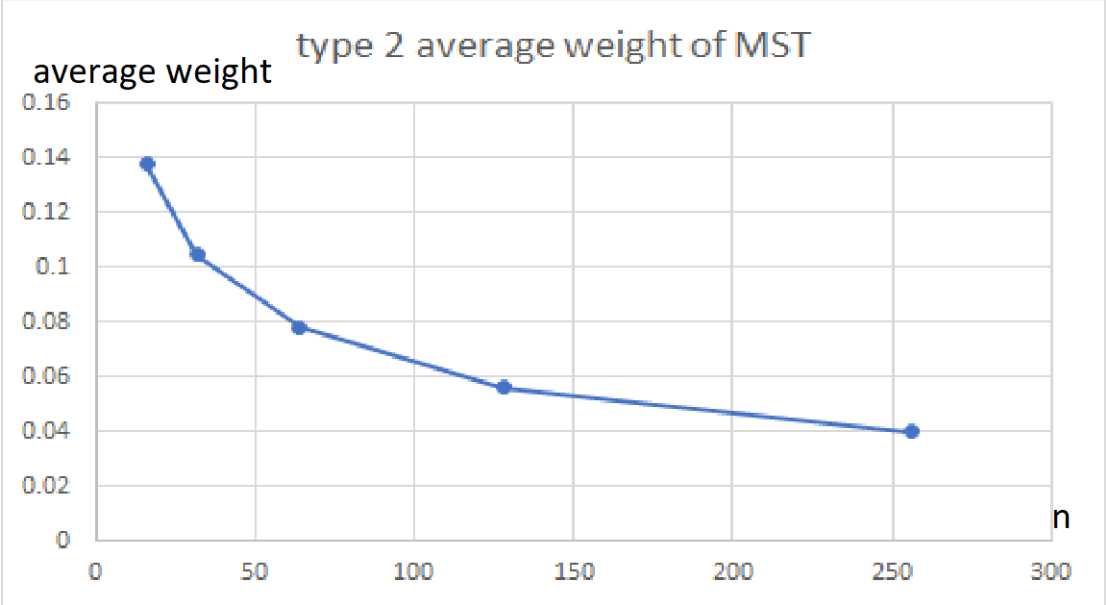

Figure 3:

As shown in the figures above, when n increases, the average weight of MST decreases. However, the average weight of MST for type 1 is always lower than that of type 2.

Approximately,

For type 1, $f(n) = 0.14\frac{1}{2}^{\frac{n}{16}}$.

This growth function is reasonable for the following reason. Since we pick the weight uniformly from $[0, 1]$, when sample space becomes bigger, we will have more and more edges that have extremely low weights (and more and more edges with high weights of course. But we can hopefully abandon them in MST). With these extremely short edges, the average weight decreases.

For type 2, $f(n) = 0.18\frac{3}{4}^{\frac{n}{16}}$.

This growth function is reasonable because as we have more and more vertexes, with random distribution, this is higher and higher probability that 2 points in a unit square are very close to each other, making an edge extremely short (and the weight becomes extremely low). With these extremely short edges, the average weight decreases.