Name: Pheakdey Luk **Assignment 14**

ID: 986591

R-7.1 Draw a simple, connected, undirected, weighted graph with 8 vertices and 16 edges, each with unique edge weights. Identify one vertex as a “start” vertex and illustrate a running of Dijkstra’s shortest path algorithm on this graph.

**Answer:**

A

F

B

E

C

H

D

G

15

6

12

8

10

11

5

3

6

4

8

9

12

7

15

14

inf

inf

inf

inf

inf

inf

inf

inf

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H |
| A | 0 | **6A** | Inf | Inf | 15A | 10A | Inf | Inf |
| B | 0 | 6A | **11B** | Inf | **14B** | 10A | 10B | Inf |
| F | 0 | 6A | 11B | Inf | 14B | 10A | 10B | Inf |
| G | 0 | 6A | 11B | 22G | 14B | 10A | 10B | 24G |
| C | 0 | 6C | 11B | 22G | 14B | 10A | 10B | **20C** |
| E | 0 | 6C | 11B | 22G | 14B | 10A | 10B | 20C |
|  |  |  |  |  |  |  |  |  |

A

F

B

E

C

H

D

G

6

8

10

5

4

9

12

6

22

10

20

10

14

11

0

R-7-7 Draw a simple, connected, undirected, weighted graph with 8 vertices and 16 edges, each with unique edge weights. Illustrate the execution of Kruskal's algorithm on this graph. (Note there is only one minimum spanning tree for this graph.)

**Answer:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H |
| A | 0 | 6A | Inf | Inf | 15A | 10A | Inf | Inf |
| B | 0 | 6A | **5B** | Inf | **8B** | 10A | 4B | Inf |
| G | 0 | 6A | 5B | 12G | 8B | **3G** | 4B | 14G |
| F | 0 | 6A | 5B | 12G | 8B | 3G | **3G** | 14G |
| C | 0 | **5C** | 5B | 12G | 8B | 3G | 3G | **9C** |
| E | 0 | 5C | 5B | 12G | 8B | 3G | 3G | 9C |
| H | 0 | 5C | 5B | **7H** | 8B | 3G | 3G | 9C |

A

F

B

E

C

H

D

G

6

8

5

3

4

9

7

7

0

[{F,G}=3, {G,B}= 4,{B,C}=5,{B,A}=6,{D,H}=7, {B,E}=8, {C,H}=9]

So, total weight = 3+4+5+6+7+8+9=42

R-7.8 Repeat the previous problem for the Prim-Jarvik algorithm.

**Answer:**

|  |  |
| --- | --- |
| A-B | 6 |
| B-G | 4 |
| G-F | 3 |
| C-H | 9 |
| H-D | 7 |
| E-B | 8 |
| D-G | 12 |

# A

F

B

E

C

H

D

G

6

8

5

3

4

9

7

6 A

7H

3G

9C

4B

8B

5B

0

R-7-9 Repeat the previous problem for Baruvka's algorithm.

E

B

F

A

15

5

6

G

D

H

C

12

7

9

4

6

11

15

8

10

8

14

3

12

# A

F

B

E

C

H

D

G

6

8

5

3

4

9

7

Consider the following potential MST algorithms based on the generic MST algorithm. Which, if any, successfully computes a MST? Hint: to show that an algorithm does not compute an MST, all you need to do is find a counterexample. If it does, you need to argue why based on the cycle property and/or the partition property.

a.

Algorithm MST-a(G, w)

T ← edges in E sorted in nonincreasing order of edge weights  
for each e ∈ T do {each e is taken in nonincreasing order by weight }

if T – {e} is a connected graph then

T ← T – {e} {remove e from T}

return T

**Answer to the Q. a:**

This algorithm will not successfully computes MST, because we don't have any checking whether the edge that we remove will give result of more than 1 connecting component. If after we remove the edge, it will make additional connecting component cycle will still happen even though the number of edges are n-1.

b.

Algorithm MST-b(G, w)

T ←{}

for each e ∈E do { e is taken in arbitrary order } if T ∪{e} has no cycles then

T ←T ∪{e} {add e to T} return T

**Answer to the Q. b:**

This algorithm will not successfully computes MST, because since we take each edges in arbitrary order, we might take the edge with the maximum number, which makes it is not a minimum spanning tree.

c.

Algorithm MST-c(G, w)

T ←{}  
for each e ∈E do { e is taken in arbitrary order }

T ←T ∪{e} {add e to T}

if T now has a cycle C then

if e’ is the edge of C with the maximum weight then

T ←T −{e’} {remove e’ to T}

return T

**Answer to the Q. c:**

This algorithm will not successfully computes MST, because we don't have any checking whether the edge that we remove will give result of more than 1 connecting component. If after we remove the edge, it will make additional connecting component cycle will still happen even though the number of edges are n-1.

C-3-28 Describe how to implement the locator-based method before(l) as well as the locator-based method closestBefore(k) in a dictionary realized using a skip-list. What are the expected running times of the two locator-based methods in your implementation?

**Answer:**

To implement the method before and closestBefore in a skip-list, we have to make sure that the skip list has a pointer to the element before. Then find l, and then call the pointer down, until we go to the bottom element, and call the pointer before the element in the current pointer.

And the expected running time will be O(log n) because we might need to traverse all the element to find l and after that to get the before l, we need constant time, so it won't be counted, which will results O(log n)

C-5.1 A native Australian named Anatjari wishes to cross a desert carrying only a single water bottle. He has a map that marks all the watering holes along the way. Assuming he can walk k miles on one bottle of water, design an efficient algorithm for determining where Anatjari should refill his bottle in order to make as few stops possible. Argue why your algorithm is correct.

Guessing 1, the water holes are located along the road and we need to sort the closest refill water place.

Algorithm RefillWater (p, k)

Input : p is water refilling place, k miles for one bottle

Output : p2 is path with less stops for refill

PQ <- new heap priority queue

for i<--0 to p.size() do

PQ.insert(p.elemAtRank(i), p.elemAtRank(i)))

tem <-- 0

while ! PQ.isEmpty() do

tem <-- tem + PQ.removeMin()

if tem + PQ.peekMin() < k then

p2.insertElement(tem)

continue

p2.insertElement(tem)

tem <-- 0

return p2

Do this with two different assumptions:

1. Assume the watering holes are all located along a road
2. Assume the watering holes are spread over the whole desert, i.e., there is no road.