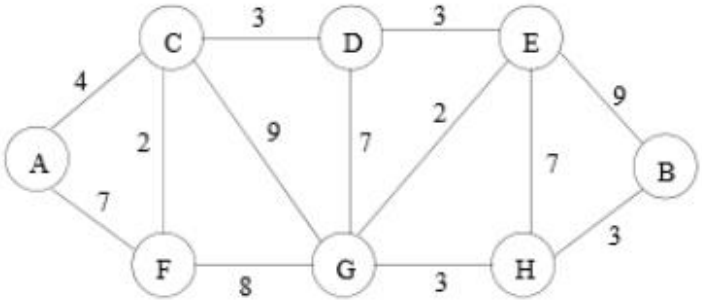
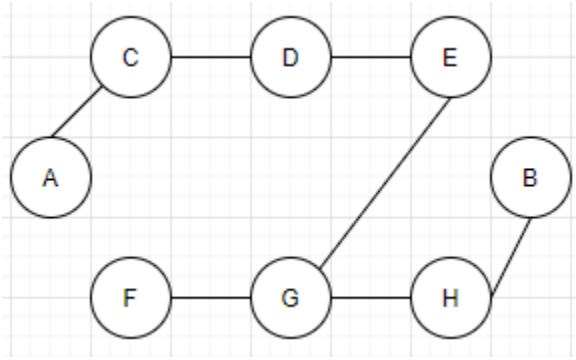
CS325

Guangyu Zhang

Problem 1. *(10 points)* A region contains a number of towns connected by roads. In the graph below, each town is labeled with a circle and each road is labeled with the average number of hours required for a truck to travel between the cities. Suppose that you work for a large retailer that has decided to place a distribution center in town G. (While this problem is small, you want to devise a method to solve much larger problems).



(a) Which algorithm would you recommend be used to find the fastest route from the distribution center to each of the towns? Demonstrate how it would work on the example above if the distribution center is placed at town G. Show the resulting routes.



Since we know the source vertex which is G, and destination is a vertex in $\{A, B, C, D, E, F, H\}$. I will use Dijkstra's Algorithm. Using two sets P, U, to contain the processed vertices and unprocessed vertices. Start from G, now $P = \{G\}$, and $U = \{A(x), B(x), C(9), D(7), E(2), F(8), H(3)\}$, find the shortest path from U, which is E, $P = \{G(0), E(2)\}$, next, Repeat the above steps, we have following: $P = \{G(0), E(2), D(2+3)\}$, here check $D(2+3) < D(7)$, update the distance as $P = \{G(0), E(2), D(5)\}$, then, $P = \{G(0), E(2), D(5), C(5+3)\}$, check $C(5+3) < C(9)$, update distance $P = \{G(0), E(2), D(5), C(8)\}$, then, $P = \{G(0), E(2), D(5), C(8), F(8+2)\}$, check $F(8+2) > F(8)$, remain the $F(8)$, $P = \{G(0), E(2), D(5), C(8), F(8)\}$, then $P = \{G(0), E(2), D(5), C(8), F(8), H(3)\}$, then $P = \{G(0), E(2), D(5), C(8), F(8), H(3), B(3+3)\}$, then $P = \{G(0), E(2), D(5), C(8), F(8), H(3), B(6), A(8+4)\}$

Therefore, $P = \{G(0), E(2), D(5), C(8), F(8), H(3), B(6), A(12)\}$

A	*12*	*GEDCA*
B	*6*	*GHB*
C	*8*	*GEDC*
D	*5*	*GED*
E	*2*	*GE*
F	*8*	*GF*
G	*0*	*G*
H	*3*	*GH*

(b) Suppose one "optimal" location (maybe instead of town G) must be selected for the distribution center such that it minimizes the time travelled to the farthest town. Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are t possible towns (locations) for the distribution center and r possible roads.

To find the optimal location, we want the path of longest path become minimum.

So, for each town, we run Dijkstra's algorithm once, and compare the longest path of results, select the town which has the minimum longest path.

In this algorithm, we run t times Dijkstra's algorithm, and for each time we have $O(t \log t + r)$ if it's in decreasing key order, otherwise it can be $O(t^2)$.

Therefore, totally, we have complexity $O(t^2 \log t + tr)$, or without the decreasing key order, it will be $O(t^3)$.

(c) In the above graph which is the "optimal" town to locate the distribution center?

Run Dijkstra's algorithm for each town, we have:

If A:

A	B	C	D	E	F	G	H
0	18	4	7	10	6	12	15

If B:

A	B	C	D	E	F	G	H
18	0	14	11	8	14	6	3

If C:

A	B	C	D	E	F	G	H
4	14	0	3	6	2	8	11

If D:

A	B	C	D	E	F	G	H
7	11	3	0	3	5	5	8

If E:

A	B	C	D	E	F	G	H
10	8	6	3	0	8	2	5

If F:

A	B	C	D	E	F	G	H
6	14	2	5	8	0	8	11

If G:

A	B	C	D	E	F	G	H
12	6	8	5	2	8	0	3

If H:

A	B	C	D	E	F	G	H
15	3	11	8	5	11	3	0

Therefore, E is the optimal location.

(d) Now suppose you can build two distribution centers. Where would you place them to minimize the time travelled to the farthest town? Describe an algorithm to solve this problem given an arbitrary road map.

To find the optimal location, we want the path of longest path become minimum.

If we can have two centers, in my algorithm, run Dijkstra's algorithm for each town as part c did, then then pick any two vertices, recalculate the distance to create new sets. Last, find the minimum of the farthest distance.

For example, if we pick A and B:

A	B	C	D	E	F	G	H
0	18	4	7	10	6	12	15
A	B	C	D	E	F	G	H
18	0	14	11	8	14	6	3

Get the minimum distance to each town, it will be:

A	B	C	D	E	F	G	H
0	0	4	7	8	6	6	3

In this algorithm, suppose we have t towns and r roads, we first do the same thing as part c did, which is $O(t^2 \log t + tr)$, then pick two towns, recalculate the distance to create new sets, then find the minimum. This is $O(t^2)$

Therefore, totally, it's $O(t^2 \log t + tr + t^2)$.

(e) In the above graph what are the "optimal" towns to place the two distribution centers?

If AB, AC, AD, AE, AF, AG, AH:

A	B	C	D	E	F	G	H
0	0	4	7	8	6	6	3
0	14	0	3	6	2	8	11
0	11	3	0	3	5	5	8
0	8	4	3	0	6	2	5
0	14	2	5	8	0	8	11
0	6	4	5	2	6	0	3

0	3	4	7	5	6	3	0

If BC, BD, BE, BF, BG, BH:

A	B	C	D	E	F	G	H
4	0	0	3	6	2	6	3
7	0	3	0	3	5	5	3
10	0	6	3	0	8	2	3
6	0	2	5	8	0	6	3
12	0	8	5	2	8	0	3
15	0	11	8	5	11	3	0

If CD, CE, CF, CG, CH:

A	B	C	D	E	F	G	H
4	11	0	0	3	2	5	8
4	8	0	3	0	2	2	5
4	14	0	3	6	0	8	11
4	6	0	3	2	2	0	3
4	3	0	3	5	2	3	0

If DE, DF, DG, DH:

A	B	C	D	E	F	G	H
7	8	3	0	0	5	2	5
6	11	2	0	3	0	5	8
7	6	3	0	2	5	0	3
7	3	3	0	3	5	5	0

If EF, EG, EH:

A	B	C	D	E	F	G	H
6	8	2	3	0	0	2	5

10	6	6	3	0	8	0	5
10	3	6	3	0	8	2	0

If FG, GH:

A	B	C	D	E	F	G	H
6	6	2	5	2	0	0	3
6	3	2	5	5	0	3	0

If GH:

A	B	C	D	E	F	G	H
12	3	8	5	2	8	0	0

Therefore, CH is the optimal location of two centers.

2.

Description:

Read how many cases in the txt, then for each case, read how many vertices in the current case, then read vertices into Ver[(x1, y1), (x2, y2),… (xn, yn)]. Next, calculate the distance between every two vertices, and store them into Edge[d1, d2, …, dn]. Finally, set an array to store vertice that included in MST,and others are in Ver[(x1, y1), (x2, y2),… (xn, yn)], from the first vertex, found its minimum distance to others, then include the minimum distance vertex in MST, remove it from the Ver[]. Repeat the steps until go through all the vertices.

Pseduocode:

```
Def deal_data:
    Read cases
    For i in cases:
        Read vertices
        For j in vertices:
            Read every vertices
        For m in vertices:
```

Calculate the distance of every two vertices

Store in edge[]

```
Def prim(vertices, edge[]):
    Key = [max]        # initial key
    P = []     #initial p
    Key[0]  = 0
    In_mst = [false] * vertices # initial in_mst
    For I in range (vertices):
        Find the minium edge and its vertex outside the in_mst to in_mst
        For j in range(vertices):
            If 0 < edge < key and in_mst is not in mst
                Set the vertex into the mst
                Store the weight of current edge
```

Running time:

In this algorithm, deal with data is $O(n^2)$, prim here is $O(n^2)$, so, totally, the running time of this algorithm is $O(n^2)$.