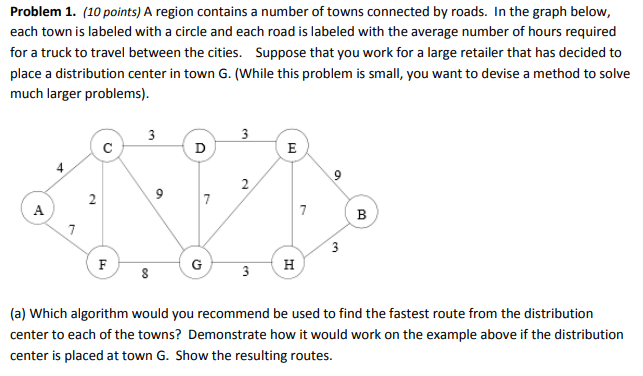
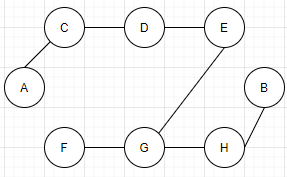
CS325

Guangyu Zhang

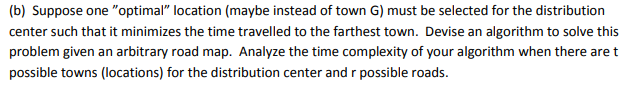




Since we know the source vertex which is , and destination is a vertex in I will use Dijkstra’s Algorithm. Using two sets , to contain the processed vertices and unprocessed vertices. Start from , now and find the shortest path from , which is , , next, Repeat the above steps, we have following: , here check , update the distance as , then, check , update distance then, check remain the , then then then

Therefore,

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



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 times Dijkstra’s algorithm, and for each time we have if it’s in decreasing key order, otherwise it can be .

Therefore, totally, we have complexity , or without the decreasing key order, it will be .



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 |

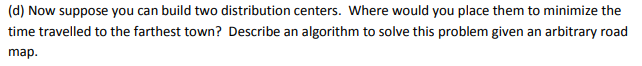
If G:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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.



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 , then pick two towns, recalculate the distance to create new sets, then find the minimum. This is

Therefore, totally, it’s .



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 , prim here is , so, totally, the running time of this algorithm is