CSC 3102 - DATA STRUCTURES AND ALGORITHM II

Group Project: Mini Hackathon

Topic 3: What can we Find Out from this Map?

Dijkstra's Algorithm PROBLEM

Table of Contents

INTRODUCTION:

Recently, studies on traffic network have attracted a lot of attention from informatics and computer science areas. A traffic network is a traffic structure made up of a set of traffic road and a set of the dyadic ties between these cities or states. A huge amount of data and resources make it critical to analyze traffic network related problems.

This assignment illustrates and provide a visualization of the best and shortest path from one specific state to another that are connected. We decided to work on Malaysia map to provide the shortest distance from Sebak Bernam state to any other state of Malaysia. We would like to apply our knowledge learnt in Data structure II class to help Sebak Bernam people to find the optimal distance to any state in order to help them to be on time by spending less cost.

MAP USED IN OUR PROJECT:

Here is a map of Selangor Malaysia.

REAL LIFE PROBLEM:

A Sebak Bernam citizen want to calculate the shortest distance between Sebak Bernam and any other state such as Sepang, Hulu Langat, Klang, etc.… according to our map to optimize the time and cost of his vacation's trips.

WHY IS THE PROBLEM IMPORTANT TO BE SOLVED?

Nowadays, car's users demand efficient route planning in a dynamically changing environment. the existing navigation systems consist mainly in the analysis of static parameters such as the total route length or road type. Thus, in congested metropolises, the efficiency of route planning is very important in daily activities of public services such as police, fire brigades, medical emergency services or personal movement.

OVERALL IDEA ON HOW TO SOLVE THE PROBLEM.

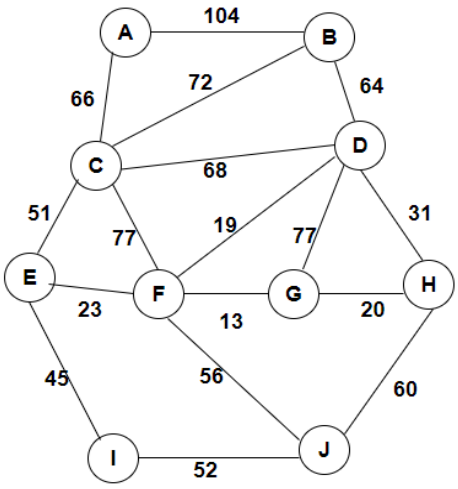
Let's simplify the map by converting it into a graph as below and naming all the states on the map with alphabetic upper letters and distance in kilometers.

States	Sabak Bernam	Hulu Selangor	Kuala Selangor	Gombak	Klang	Petaling	Kuala Lumpur	Hulu Langat	Kuala langat	Sepang
Notations	A	B	C	D	E	F	G	H	I	J

Real measurement of interconnected states

Interconnected states	Distance (Km)
A-B	104
A-C	66
B-C	72
B-D	645
C-D	68
C-E	51
C-F	77
D-F	19
D-G	77
D-H	31
E-F	29
E-I	45
F-G	13
F-J	57
G-H	20
H-J	60
I-J	52

GRAPH FROM OUR MAP



SOLVING TECHNIQUE AND ALGORITHM USED

By translating our real-life problem to a weighted graph, we will have ten (10) vertices and seventy (17) edges and assuming our distances as our weight, we do use Dijkstra's Algorithm to solve the shortest path from Sebak Bernam to any other state according to our map. So, since all our vertices or nodes are interconnected, this algorithm makes a tree of the shortest path from the starting node which is in our case here Sebak Bernam also called the source to all other nodes (points) which are in our case the other cities in the graph.

Here is the overall idea of to solve our problem with Dijkstra's Algorithm:

- Basically, the Dijkstra's algorithm starts from the source node which is in our case Sebak Bernam represented by 'A', and it examines the entire graph to determine the shortest path among that node and all the other nodes in the graph.
- The algorithm will conserve the track of the current known shortest distance from each node to the source node 'A' and updates the values if another shortest path is identified.
- Once the algorithm has determined the shortest distance from the source node 'A' to another node, the node will be marked as "visited node" and can be eventually added to the path.
- This process will continue until all the nodes in our graph are added to the path. as a result, a path will be created that connects our source node 'A' to all the other nodes in our graph which will be the plausible shortest path to get each node.

DIJKSTRA PSEUDO ALGORITHM

```
dist[S] ← 0                          // The distance to source vertex is set to 0.
Π[S] ← NIL                           // The predecessor of source vertex is set as NIL.
for all v ∈ V - {S}                  // For all other vertices
do dist[v] ← ∞                       // All other distances are set to ∞
Π[v] ← NIL                           // The predecessor of all other vertices is set as NIL.
S ← ∅                                // The set of vertices that have been visited 'S' is initially empty.
Q ← V                                // The queue 'Q' initially contains all the vertices.
while Q ≠ ∅                          // While loop executes till the queue is not empty
do u ← mindistance (Q, dist)         // A vertex from Q with the least distance is selected
S ← S ∪ {u}                          // Vertex 'u' is added to 'S' list of vertices that have been visited
for all v ∈ neighbors[u]             // For all the neighboring vertices of vertex 'u'
do if dist[v] > dist[u] + w(u,v)     // if any new shortest path is discovered
then dist[v] ← dist[u] + w(u,v)      // The new value of the shortest path is selected
return dist
```

SOLVING PROCESS STEP-BY STEP EXPLANATION BY DIAGRAM

Solving with Dijkstra's Algorithm Graph:

Using Dijkstra's Algorithm, find the shortest distance from source vertex 'A' to remaining vertices in the following graph.

Unvisited set: {A, B, C, D, E, F, G, H, I, J}

Visited set: { }

The two variables Π and d are created for each vertex and initialized as-

Π[A] = Π[B] = Π[C] = Π[D] = Π[E] = Π[F] = Π[G] = Π[H] = Π[I] = Π[J] = NIL

d[A] = 0

d[B] = d[C] = d[D] = d[E] = d[F] = d[G] = d[H] = d[I] = d[J] = ∞

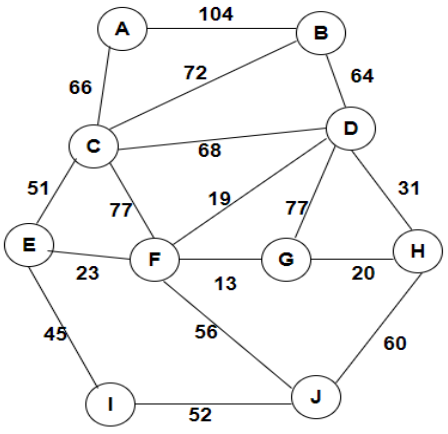
Vertex 'S' is chosen.

This is because shortest path estimate for vertex 'A' is least.

The outgoing edges of vertex 'A' are relaxed.

Step 1:

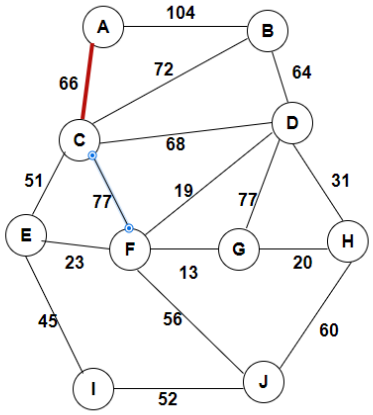
Tree vertices	Remaining vertices
A(-,0)	B(-,∞), C(-,∞), D(-,∞), E(-,∞), F(-,∞), G(-,∞), H(-,∞), I(-,∞), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	
J	∞	

Step 2:

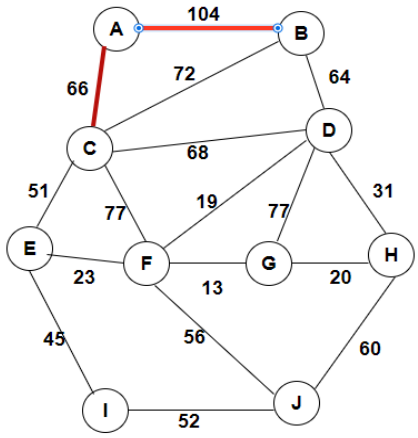
Tree vertices	Remaining vertices
A(-,0)	B(A,104), C(A, 66), D(-,∞), E(-,∞), F(-,∞), G(-,∞), H(-,∞), I(-,∞), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	
J	∞	

Step 3:

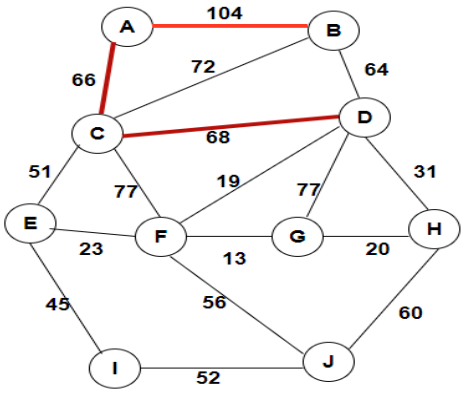
Tree vertices	Remaining vertices
C(A,66)	B(A,104), D(C,134), E(C,117), F(C,143), G(-,∞), H(-,∞), I(-,∞), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	143	C
G	∞	
H	∞	
I	∞	
J	∞	

Step 4:

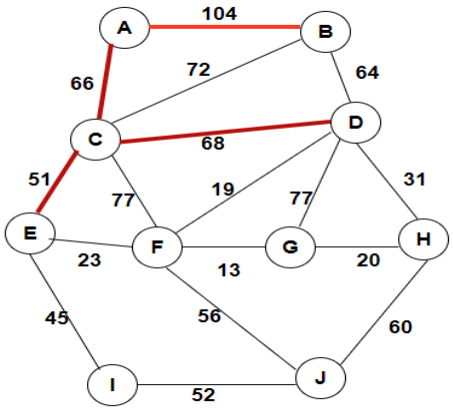
Tree vertices	Remaining vertices
B(A,104)	D(C,134), E(C,117), F(C,143), G(-,∞), H(-,∞), I(-,∞), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	143	C
G	∞	
H	∞	
I	∞	
J	∞	

Step 5:

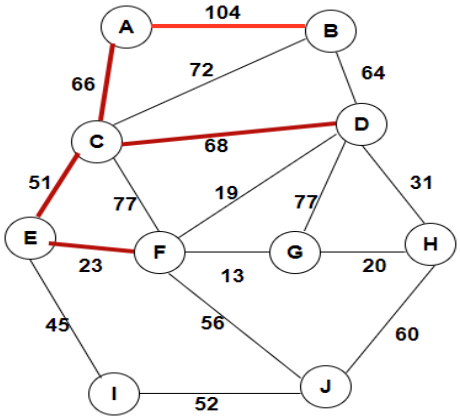
Tree vertices	Remaining vertices
D(C,134),	E(C,117), F(C,143), G(D,211), H(D,165), I(-,∞), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	143	C
G	211	D
H	165	D
I	∞	
J	∞	

Step 6:

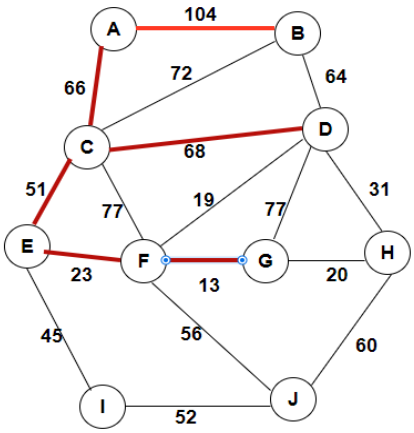
Tree vertices	Remaining vertices
E(C,117),	F(E,140), G(D,211), H(D,165), I(E,162), J(-,∞),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	211	D
H	165	D
I	162	E
J	∞	

Step 7:

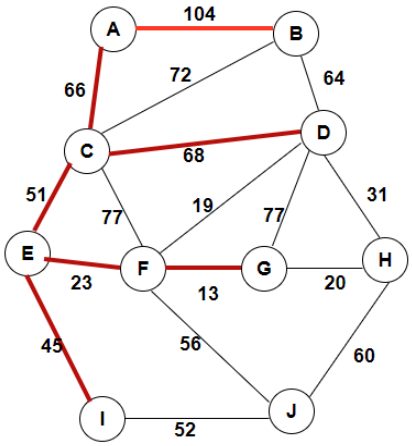
Tree vertices	Remaining vertices
F(E,140)	G(F,153), H(D,165), I(E,162), J(F,196),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	153	F
H	165	D
I	162	E
J	196	F

Step 8:

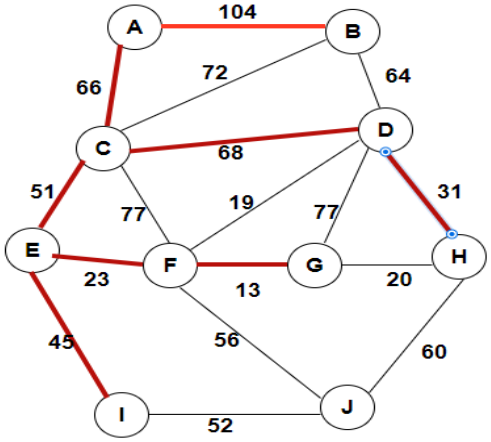
Tree vertices	Remaining vertices
G(F,153)	H(D,165), I(E,162), J(F,196),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	153	F
H	165	D
I	162	E
J	196	F

Step 9:

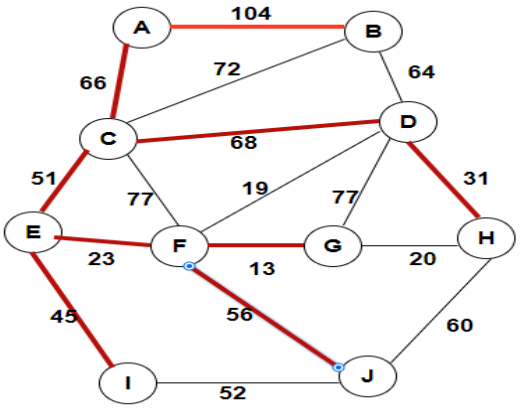
Tree vertices	Remaining vertices
I(E,162)	H(D,165), J(F,196),



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	153	F
H	165	D
I	162	E
J	196	F

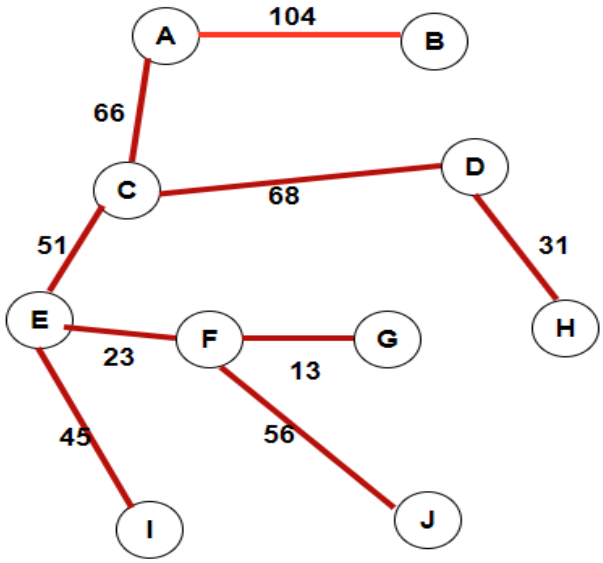
Step 10:

Tree vertices	Remaining vertices
H(D,165)	J(F,196)



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	153	F
H	165	D
I	162	E
J	196	F

Here is the final shortest distance from Sebak bernam to any other state according to our graph



Vertex	Shortest Distance	Prev
A	0	
B	104	A
C	66	A
D	134	C
E	117	C
F	140	E
G	153	F
H	165	D
I	162	E
J	196	F

Solving with Dijkstra's Algorithm in C++:

In the code below, an adjacency matrix is used for an undirected graph.

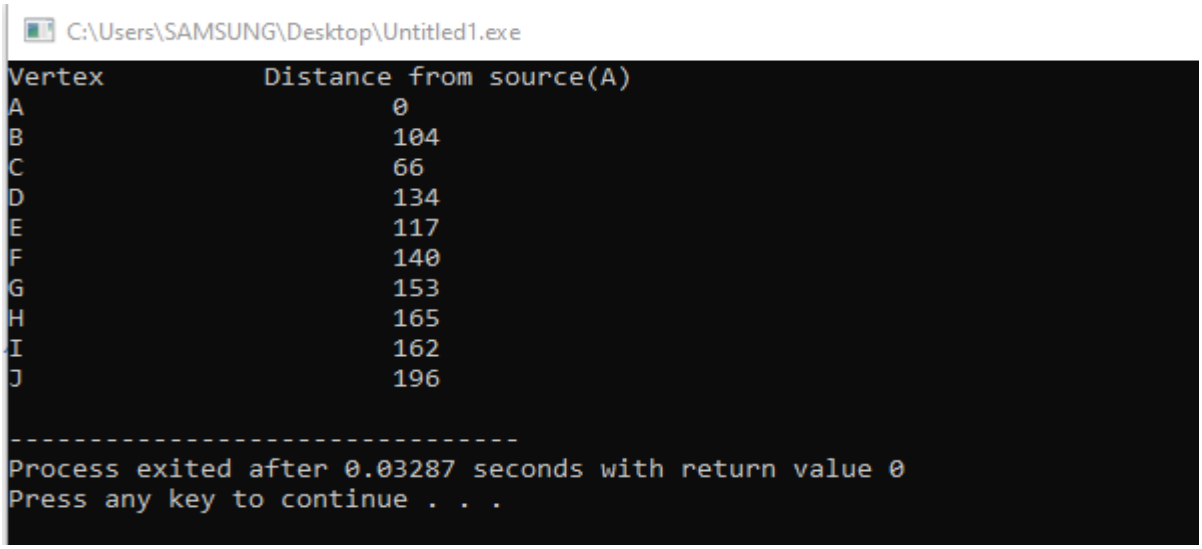
The vertex with the minimum distance, which is not included in the test, is searched in the minimumDist() method.

The time complexity of the algorithm is O ((|V| + |E|) log V) $O((|V|+|E|) logV)$ where E represents edges and V represents vertices. Similarly, space complexity is O (|V| + |E|) $O(|V|+|E|)$.

```
#include<iostream>

#include<climits>

using namespace std;

//this method returns a minimum distance for the
//vertex which is not included in Tset.

int minimumDist(int dist[], bool Tset[])

{ int min=INT_MAX,index;

for(int i=0;i<10;i++)

{ if(Tset[i]==false && dist[i]<=min)

{ min=dist[i];

index=i;

}

}return index;

} void Dijkstra(int graph[10][10],int src) // adjacency matrix used is 6x6

{ int dist[10]; // integer array to calculate minimum distance for each node.

bool Tset[10];// boolean array to mark visted/unvisted for each node.

// set the nodes with infinity distance // except for the initial node and mark
// them unvisited.

for(int i = 0; i<10; i++)

{ dist[i] = INT_MAX;

Tset[i] = false;

}

dist[src] = 0; // Source vertex distance is set to zero.

for(int i = 0; i<10; i++)
```

```
{
int m=minimumDist(dist,Tset); // vertex not yet included.
Tset[m]=true;// m with minimum distance included in Tset.
for(int i = 0; i<10; i++)
{
// Updating the minimum distance for the particular node.
if(!Tset[i] && graph[m][i] && dist[m]!=INT_MAX &&
dist[m]+graph[m][i]<dist[i])
dist[i]=dist[m]+graph[m][i];
}
}
cout<<"Vertex\t\tDistance from source"<<endl;
for(int i = 0; i<10; i++)
{ char str=65+i; // Ascii values for pritning A,B,C..
cout<<str<<"\t\t\t"<<dist[i]<<endl;
}
}int main()
{
int graph[10][10]={
{0, 104, 66, 0, 0, 0, 0, 0, 0, 0},
{104, 0, 72, 64, 0, 0, 0, 0, 0, 0},
{66, 72, 0, 68, 51, 77, 0, 0, 0, 0},
{0, 64, 68, 0, 0, 19, 77, 31, 0, 0},
{0, 0, 51, 0, 0, 23, 0, 0, 45, 0},
{0, 0, 77, 19, 23, 0, 13, 0, 0, 56},
{0, 0, 0, 77, 0, 13, 0, 20, 0, 0},
{0, 0, 0, 31, 0, 0, 20, 0, 0, 60},
{0, 0, 0, 0, 45, 0, 0, 0, 0, 52},
{0, 0, 0, 0, 0, 56, 0, 60, 52, 0}};
Dijkstra(graph,0);
return 0;
}
```

Output:



CONCLUSION

To summarize, at the minimum we can find Sebak bernam state by using Dijkstra algorithm if we are in any state in the map which provide the shortest path or distance between them. In this assignment, we first search the real measurement of the distances between the connected states on our map and discussed about the algorithm to be used. With all members agreement we finally chose Dijkstra algorithm as the efficient algorithm for our problem. We then turned to visibility the problem into graph and detailed the steps involved in deploying Dijkstra's algorithm. Finally, we described how to compute the shortest path by Dijkstra's code using C++ programming language, which takes O(n2) run time.

REFERENCES

1. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
2. https://www.hackerearth.com/blog/developers/kruskals-minimum-spanning-tree-algorithm-example/
3. https://www.algotree.org/algorithms/single_source_shortest_path/dijkstras_shortest_path_c++/
4. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
5. https://www.gatevidyalay.com/dijkstras-algorithm-shortest-path-algorithm/