

## I2206

### Data Structures

### LS 7 : Dijkstra, Bellman-Ford and Prim Graph Algorithms

## 1 Dijkstra's Algorithm

A famous solution for the shortest path problem was developed by *Dijkstra*. Dijkstra's algorithm is a generalization of the BFS algorithm. The regular BFS algorithm cannot solve the shortest path problem as it cannot guarantee that the vertex at the front of the queue is the vertex closest to source  $s$ .

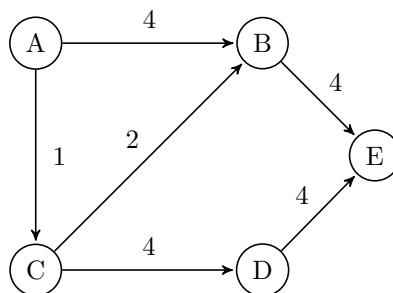
As in unweighted shortest path algorithm, here too we use the **Distance** table. The algorithm works by keeping the shortest distance of vertex  $v$  from the source in the **Distance** table. The value **Distance**[ $v$ ] holds the distance from  $s$  to  $v$ . The shortest distance of the source to itself is zero. The **Distance** table for all other vertices is set to -1 to indicate that those vertices are not already processed.

After the algorithm finishes, the **Distance** table will have the shortest distance from source  $s$  to each other vertex  $v$ .

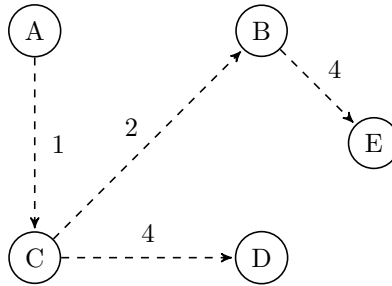
### Notes :

- Always pick the next closest vertex to the source.
- It uses priority queue to store unvisited vertices by distance from  $s$ .
- It does not work with negative weights.
- To represent weights in the adjacency list, each vertex contains the weights of the edges (in addition to their identifier).
- Instead of ordinary queue we use priority queue [distances are the priorities] and the vertex with the smallest distance is selected for processing.
- The distance to a vertex is calculated by the sum of the weights of the edges on the path from the source to that vertex.
- We update the distances in case the newly computed distance is smaller than the old distance which we have already computed.

*Example :* Dijkstra's algorithm can be used to find the shortest path from source A to the remaining vertices in the graph.



The final minimum cost tree which Dijkstra's algorithm generates is :



Given the following declaration,

```
typedef struct {
    int V;
    int E;
    int **Adj;
} Graph;
```

1. Write a function that reads a weighted graph.
2. Write a function that takes as parameters a weighted graph and an vertex, and returns the final minimum cost tree using Dijkstra's algorithm.
3. Write a main function to test your function.

```

C:\Users\antoun\desktop\documents\visual studio 2017\Projects\Co...
Enter the number of Vertices: 5
Enter the number of Edges: 6
Enter the edge and the weight: 0 1 4
Enter the edge and the weight: 0 2 1
Enter the edge and the weight: 1 4 4
Enter the edge and the weight: 2 1 2
Enter the edge and the weight: 2 3 4
Enter the edge and the weight: 3 4 4
0 should be reached from -842150451 of total weight 0
1 should be reached from 2 of total weight 3
2 should be reached from 0 of total weight 1
3 should be reached from 2 of total weight 5
4 should be reached from 1 of total weight 7

```

## 2 Bellman-Ford Algorithm

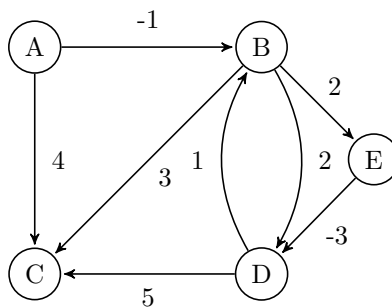
If the graph has negative edge costs, then Dijkstra's algorithm does not work. The problem is that once a vertex  $u$  is declared known, it is possible that from some other, unknown vertex  $v$  there is a path back to  $u$  that is very negative. In such a case, taking a path from  $s$  to  $v$  back to  $u$  is better than going from  $s$  to  $u$  without using  $v$ .

A combination of Dijkstra's algorithm and unweighted algorithms will solve the problem.

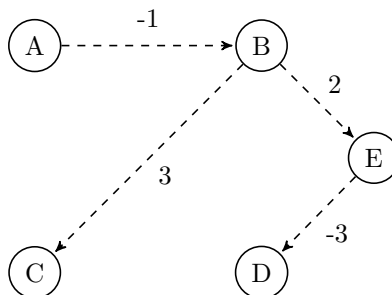
Initialize the queue with  $s$ . Then, at each stage, we DeQueue a vertex  $v$ . We find all vertices  $W$  adjacent to  $v$  such that,  $\text{distance to } v + \text{weight}(v, w) < \text{old distance to } w$ .

We update  $w$  old distance and path, and place  $w$  on a queue if it is not already there. A bit can be set for each vertex to indicate presence in the queue. We repeat the process until the queue is empty.

*Example :* Bellman-Ford algorithm can be used to find the shortest path from source A to the remaining vertices in the graph.



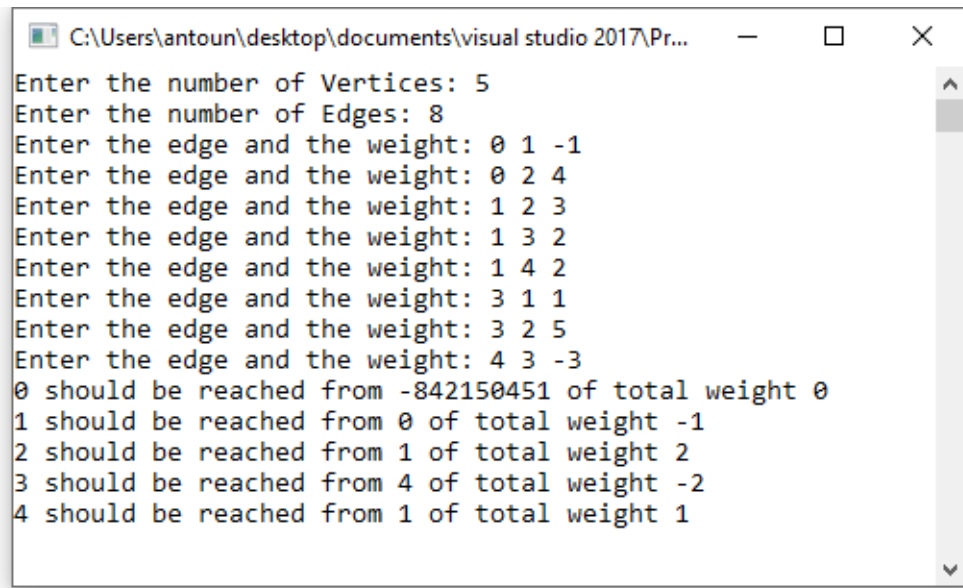
The final minimum cost tree which Bellman-Ford algorithm generates is :



Given the following declaration,

```
typedef struct {
    int V;
    int E;
    int **Adj;
} Graph;
```

1. Write a function that reads a graph containing negative weights.
2. Write a function that takes as parameters a weighted graph and an vertex, and returns the final minimum cost tree using Bellman-Ford algorithm.
3. Write a main function to test your function.



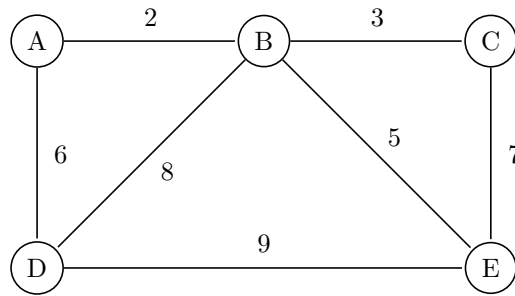
```
C:\Users\antoun\Desktop\documents\visual studio 2017\Pr...
Enter the number of Vertices: 5
Enter the number of Edges: 8
Enter the edge and the weight: 0 1 -1
Enter the edge and the weight: 0 2 4
Enter the edge and the weight: 1 2 3
Enter the edge and the weight: 1 3 2
Enter the edge and the weight: 1 4 2
Enter the edge and the weight: 3 1 1
Enter the edge and the weight: 3 2 5
Enter the edge and the weight: 4 3 -3
0 should be reached from -842150451 of total weight 0
1 should be reached from 0 of total weight -1
2 should be reached from 1 of total weight 2
3 should be reached from 4 of total weight -2
4 should be reached from 1 of total weight 1
```

### 3 Prim's Algorithm - Minimal Spanning Tree

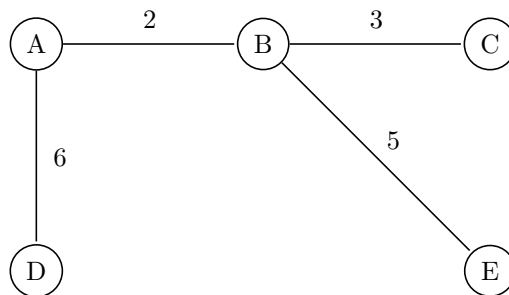
A minimum spanning tree of an undirected graph  $G$  is a tree formed from graph edges that connect all the vertices of  $G$  with minimum total cost (weights). A minimum spanning tree exists only if the graph is connected.

Prim's algorithm is almost the same as Dijkstra's algorithm. As in Dijkstra's algorithm, in Prim's algorithm we keep the values distance and paths in the distance table. The only exception is that since the definition of distance is different, the updating statement also changes a little. The update statement is simpler than before.

*Example :* Prim's algorithm can be used to find the Minimal Spanning Tree of the graph.



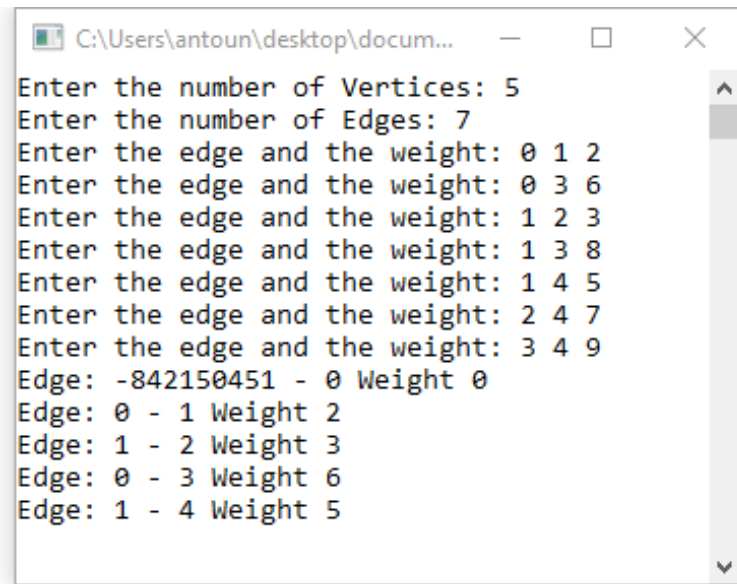
The Minimal Spanning Tree which Prim's algorithm generates is :



Given the following declaration,

```
typedef struct {
    int V;
    int E;
    int **Adj;
} Graph;
```

1. Write a function that reads an undirected weighted graph.
2. Write a function that takes as parameters an undirected weighted graph and an vertex, and returns the final Minimal Spanning Tree using Prim's algorithm.
3. Write a main function to test your function.



A screenshot of a Windows Notepad window titled "C:\Users\antoun\desktop\docum...". The window contains a text-based interface for entering graph data. It prompts the user to enter the number of vertices (5) and edges (7). Then, it asks for 7 edges, each with two vertices and a weight. The input is as follows:

```
Enter the number of Vertices: 5
Enter the number of Edges: 7
Enter the edge and the weight: 0 1 2
Enter the edge and the weight: 0 3 6
Enter the edge and the weight: 1 2 3
Enter the edge and the weight: 1 3 8
Enter the edge and the weight: 1 4 5
Enter the edge and the weight: 2 4 7
Enter the edge and the weight: 3 4 9
Edge: -842150451 - 0 Weight 0
Edge: 0 - 1 Weight 2
Edge: 1 - 2 Weight 3
Edge: 0 - 3 Weight 6
Edge: 1 - 4 Weight 5
```