EECE 7205-Assignment 5

Sreejith Sreekumar: 001277209

December 6, 2018

1 Qn1 - Dijkstra's Algorithm

1.1 Code

```
#include <iostream>
#include <stdio.h>
#include <limits.h>


void output_to_console(int distance[], int vertices)
{
  int src = 0;
  printf("Vertex \t Distance from Source\n");
  for (int i = 1; i < vertices; i++){
      printf("%d \t %d\n", i, distance[i]);
    }
}


int minimun_distance_vertex(int distance[], bool shortest_path[], int vertices)
{

  int min = INT_MAX, min_index;
  for (int v = 0; v < vertices; v++) {
    if (shortest_path[v] == false && distance[v] <= min) {
      min = distance[v], min_index = v;
    }
  }

  return min_index;
}


void dijkstra(int** graph, int src, int vertices)
{

  int predecessor[vertices];
```

```
  int distance[vertices];
  bool shortest_path[vertices];

  for (int i = 0; i < vertices; i++){
    predecessor[0] = -1;
    distance[i] = INT_MAX;
    shortest_path[i] = false;
  }

  distance[src] = 0;
  for (int count = 0; count < vertices - 1; count++){
    int u = minimun_distance_vertex(distance, shortest_path, vertices);
    shortest_path[u] = true;
    for (int v = 0; v < vertices; v++) {
      if (!shortest_path[v] && graph[u][v] && distance[u] + graph[u][v] < distance[v]){
        predecessor[v] = u;
        distance[v] = distance[u] + graph[u][v];
      }
    }
  }

  output_to_console(distance, vertices);
}


int main()
{

  int source;
  int vertices;

  std::cout<<"Enter the number of vertices: ";
  std::cin>>vertices;

  int **_graph = new int*[vertices];

  for(int i=0; i<vertices; i++){
    _graph[i] = new int[vertices];
  }


  for(int i=0; i<vertices; i++){
    std::cout<<"Enter the connections for vertex "<<i<<" (Please separate weights by a space): ";
    for(int j=0; j<vertices; j++){
      std::cin>>_graph[i][j];
    }
  }
```

```
  std::cout<<"Enter the source vertex: ";
  std::cin>>source;

  dijkstra(_graph, source, vertices);

}
```

1.2 Output

```
Welcome to the Emacs shell

~/code/explore-algorithms-cpp/hw5 $ g++ dijkstra.cpp
~/code/explore-algorithms-cpp/hw5 $ ./a.out
Enter the number of vertices: 9
Enter the connections for vertex 0 (Please separate weights by a space): 0 4 0 0 0 0 0 8 0
Enter the connections for vertex 1 (Please separate weights by a space): 4 0 8 0 0 0 0 11 0
Enter the connections for vertex 2 (Please separate weights by a space): 0 8 0 7 0 4 0 0 2
Enter the connections for vertex 3 (Please separate weights by a space): 0 0 7 0 9 14 0 0 0
Enter the connections for vertex 4 (Please separate weights by a space): 0 0 0 9 0 10 0 0 0
Enter the connections for vertex 5 (Please separate weights by a space): 0 0 4 0 10 0 2 0 0
Enter the connections for vertex 6 (Please separate weights by a space): 0 0 0 14 0 2 0 1 6
Enter the connections for vertex 7 (Please separate weights by a space): 8 11 0 0 0 0 1 0 7
Enter the connections for vertex 8 (Please separate weights by a space): 0 0 2 0 0 0 6 7 0
Enter the source vertex: 0
Vertex          Distance from Source
1          4
2          12
3          19
4          21
5          11
6          9
7          8
8          14
~/code/explore-algorithms-cpp/hw5 $
```

2 Qn: Bellman-Ford Algorithm

2.1 Code

```
#include <limits.h>
#include <iostream>

using namespace std;

struct Edge{
  int source;
  int destination;
  int weight;
};


struct Graph{
  int node_count;
  int edges_count;
  struct Edge* edge;
};


void output_to_console(int distance[], int vertex_count)
{
  std::cout<<"\nNodes \tDistance from Source\n";
  for (int i = 0; i < vertex_count; ++i){
    std::cout<<i<<"\t"<<distance[i]<<"\n";
  }
}


struct Graph* build(int node_count, int edges_count)
{

  struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));
  graph->node_count = node_count;
  graph->edges_count = edges_count;

  graph->edge = (struct Edge*) malloc(graph->edges_count * sizeof(struct Edge));
  return graph;

}


void bellman_ford(struct Graph* graph, int source)
{
```

```
  int node_count = graph->node_count;
  int shortest_path[node_count];
  int edges_count = graph->edges_count;

  for (int i=0; i <node_count; i++)
    shortest_path[i] = INT_MAX;

  shortest_path[source] = 0;

  for (int i=1; i<=node_count-1; i++){
    for (int j=0; j <edges_count; j++){
      int u = graph->edge[j].source;
      int v = graph->edge[j].destination;
      int weight = graph->edge[j].weight;
      if (shortest_path[u] + weight < shortest_path[v])
        shortest_path[v] = shortest_path[u] + weight;
    }
  }

  for (int i=0; i <edges_count; i++){

    int u = graph->edge[i].source;
    int v = graph->edge[i].destination;
    int weight = graph->edge[i].weight;

    if (shortest_path[u] + weight < shortest_path[v]){
      std::cout<<"\n Graph contains negative edge cycle\n";
    }
  }

  output_to_console(shortest_path, node_count);

}


int main()
{

  int vertices,edges,source;

  std::cout<<"Enter number of vertices:";
  std::cin>>vertices;

  std::cout<<"Enter number of edges:";
  std::cin>>edges;

  std::cout<<"Enter source vertex ID:";
  std::cin>>source;
```

```
  struct Graph* graph = build(vertices, edges);
  std::cout<<"\nEnter edge attributes: Source, destination, weight respectively\n";

  for(int i=0;i<edges;i++){
    std::cout<<"Edge "<<i<<": ";
    std::cin>>graph->edge[i].source>>graph->edge[i].destination>>graph->edge[i].weight;
  }

  bellman_ford(graph, source);

}
```

2.2 Output

```
Welcome to the Emacs shell

~/code/explore-algorithms-cpp/hw5 $ g++ bellman-ford.cpp
~/code/explore-algorithms-cpp/hw5 $ ./a.out
Enter number of vertices:5
Enter number of edges:10
Enter source vertex ID:0

Enter edge attributes: Source, destination, weight respectively
Edge 0: 0 1 6
Edge 1: 0 2 7
Edge 2: 1 2 8
Edge 3: 1 4 -4
Edge 4: 1 3 5
Edge 5: 3 1 -2
Edge 6: 2 3 -3
Edge 7: 2 4 9
Edge 8: 4 0 9
Edge 9: 4 3 7

Nodes         Distance from Source
0        0
1        2
2        7
3        4
4        -2
~/code/explore-algorithms-cpp/hw5 $
```