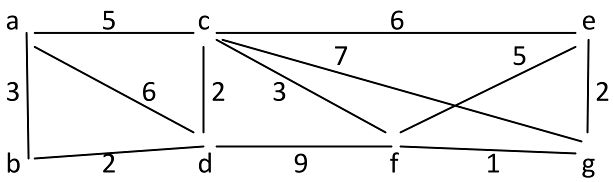
Dijkstra's Algorithm:

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.



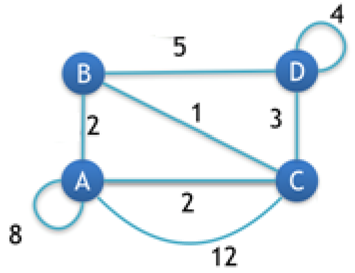
Source: a

v	a	b	c	d	e	f	g
a	0a	3_a	5_a	6_a	∞_a	∞_a	∞_a
b (Smallest)	0a	3_a	5_a	5_b (3+2)	∞_a	∞_a	∞_a
c (Smallest)	0a	3_a	5_a	5_b	11_c (5+6)	8_c (5+3)	12_c (5+7)
d (Smallest)	0a	3_a	5_a	5_b	11_c	8_c	12_c
f (Smallest)	0a	3_a	5_a	5_b	11_c	8_c	9_f (8+1)

Path from a to g: g ← f ← c ← a:

- a to c: 5
- c to f: 3
- f to g: 1

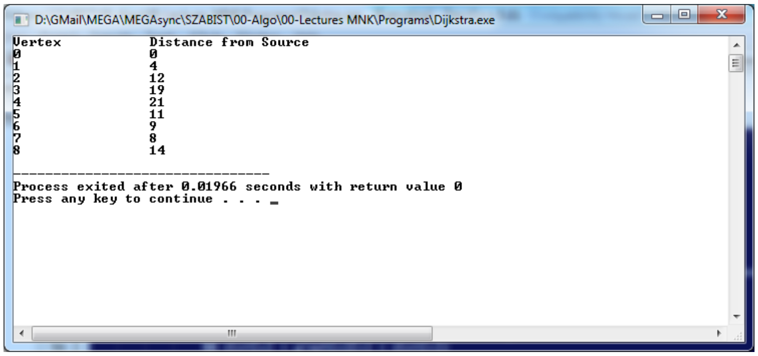
Task: Source is A



v	A	B	C	D
A	0a	2a	2a	∞a
B	0a	2a	min(2+1,2) = 2a	min(5+2, ∞) = 7b
C	0a	2a	2a	min(2+3,7) = 5c

```
// A C++ program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

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

#define V 9       // Number of vertices in the graph

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        sptSet[u] = true; // Mark the picked vertex as processed

        for (int v = 0; v < V; v++) // Update dist value of the adjacent vertices of the picked vertex.
            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
```

```
  }

  printSolution(dist);
}

int main()
{
  int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
              { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
              { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
              { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
              { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
              { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
              { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
              { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
              { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

  dijkstra(graph, 0);
  return 0;
}
```


D:\GMail\MEGA\MEGAsync\SZABIST\00-Algo\00-Lectures MNK\Programs\Dijkstra.exe

```
Vertex           Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14

--------------------------------
Process exited after 0.01966 seconds with return value 0
Press any key to continue . . . _
```

Task:

1. Draw the graph used in this program.
2. Apply the algorithm with different source nodes.