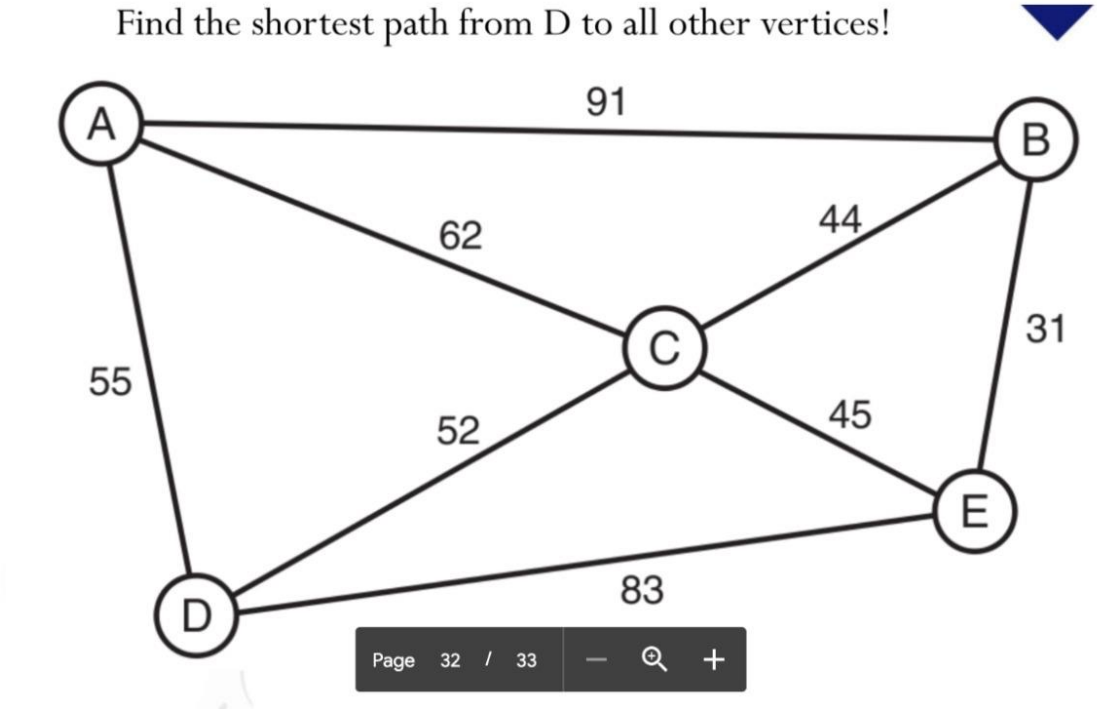
Dijkstra Algorithm to Find Shortest Path



First, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

Algorithm

1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3) While sptSet doesn't include all vertices

a) Pick a vertex u which is not there in sptSet and has minimum distance value.
b) Include u to sptSet.
c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Implementation

1. First, we import libraries and define number of vertices in the graph and store it in global variable.

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

// Number of vertices in the graph
const int V = 5;
```

2. We define graph representation as adjacency matrix and then, call Dijkstra procedure with D (vertex with index 3) as a source.

```
// Main Program
int main() {
    // Define adjacency matrix
    int graph1[V][V] = { {0, 91, 62, 55, 0},
                         {91, 0, 44, 0, 31 },
                         {62, 44, 0, 52, 45},
                         {55, 0, 52, 0, 83},
                         {0, 31, 45, 83, 0} };

    std::cout << "\nShortest path from D to all other vertices: " << std::endl;
    dijkstra(graph1, 3); // D is vertex with index 3, A -> 0, B -> 1, etc

    return 0;
}
```

Function implementation

Here we use 3 functions / procedures

1. Dijkstra procedure
   Procedure that implements Dijkstra's single source shortest path algorithm for a graph represented using adjacency matrix representation.

   First, we declare output array which stores shortest distance from source to all vertices and boolean array sptSet to store which vertex is included in shortest path tree. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not.

Then, we initialize all distance from source as infinity (max integer) and sptSet as false. Also change distance of source vertex to itself as 0.

Then, we find shortest path for all vertices with looping which we also use other function (minDistance) in the looping. We iterate for every number of vertices in the graph minus 1. Then we use minDistance function and store it in variable and set sptSet value for it as true.

We then iterate again for every number of vertices in the graph and update array dist value if it meets the conditions.

We then call other procedure (printSolution) to print out the result.

```
void dijkstra(int graph[V][V], int src) {
    int dist[V];
    // The output array. dist[i] will hold the shortest
    // distance from src to vertex i

    bool sptSet[V];
    // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to vertex 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);
        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)
            // 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];
    }

    // print the constructed distance array
    printSolution(dist);
}
```

2. minDistance function
   this function purpose is to find the vertex with minimum distance value, from the set of vertices not yet included in shortest path tree.

First we define min variable as infinity and declare min_index. Then we iterate for every number of vertices in the graph. If the vertex is not yet included in shortest path tree and its distance is less or equal to min value, we change min value and min_index

Last, we return min_index.

```
// 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[]) {
    // Initialize min value as infinity and declare min_index
    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;
}
```

3. printSolution
   First we define array of character to convert from index of vertex to its name.

   Then, we print the name of vertex and its distance to source for every vertex.

```
// A utility function to print the constructed distance array
void printSolution(int dist[]) {
    char arr[5] = {'A','B','C','D','E'};
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%c \t\t %d\n", arr[i], dist[i]);
}
```

Result

```
Shortest path from D to all other vertices:
Vertex          Distance from Source
A               55
B               96
C               52
D               0
E               83
```