Samuel Chavarria
411021334
Dijkstra’s Algorithm

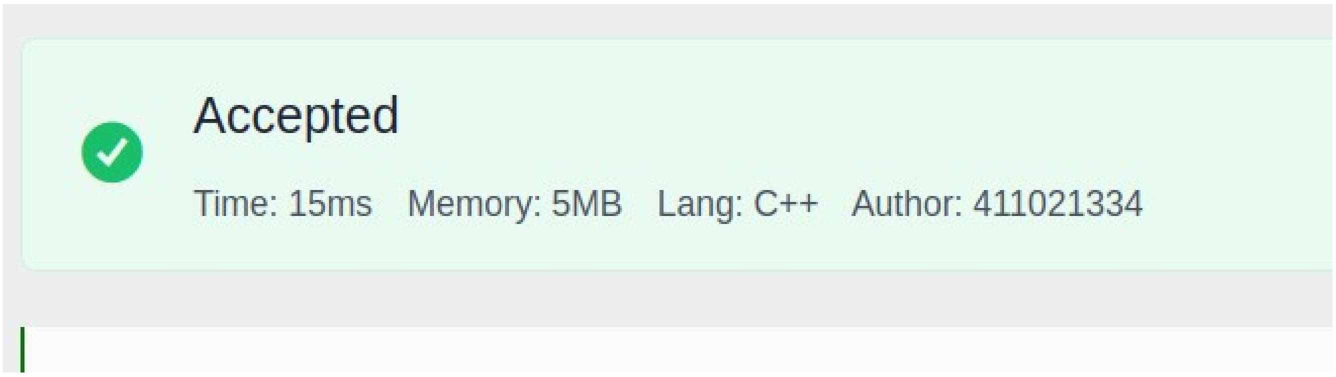
Description:

This problem employed Dijkstra’s algorithm directly, since it fit the description to the detail of the algorithm. Given that the problem was identical to the description, the algorithms was the most suitable for the solution. The only adaptation was that this solution problem contained an array with the visited vertices, it’s element number represented the vertex number, and it was an array of boolean values.

Time Complexity (worst):
O(|E| * log |V|)

Space Complexity:
O(|V|^2)

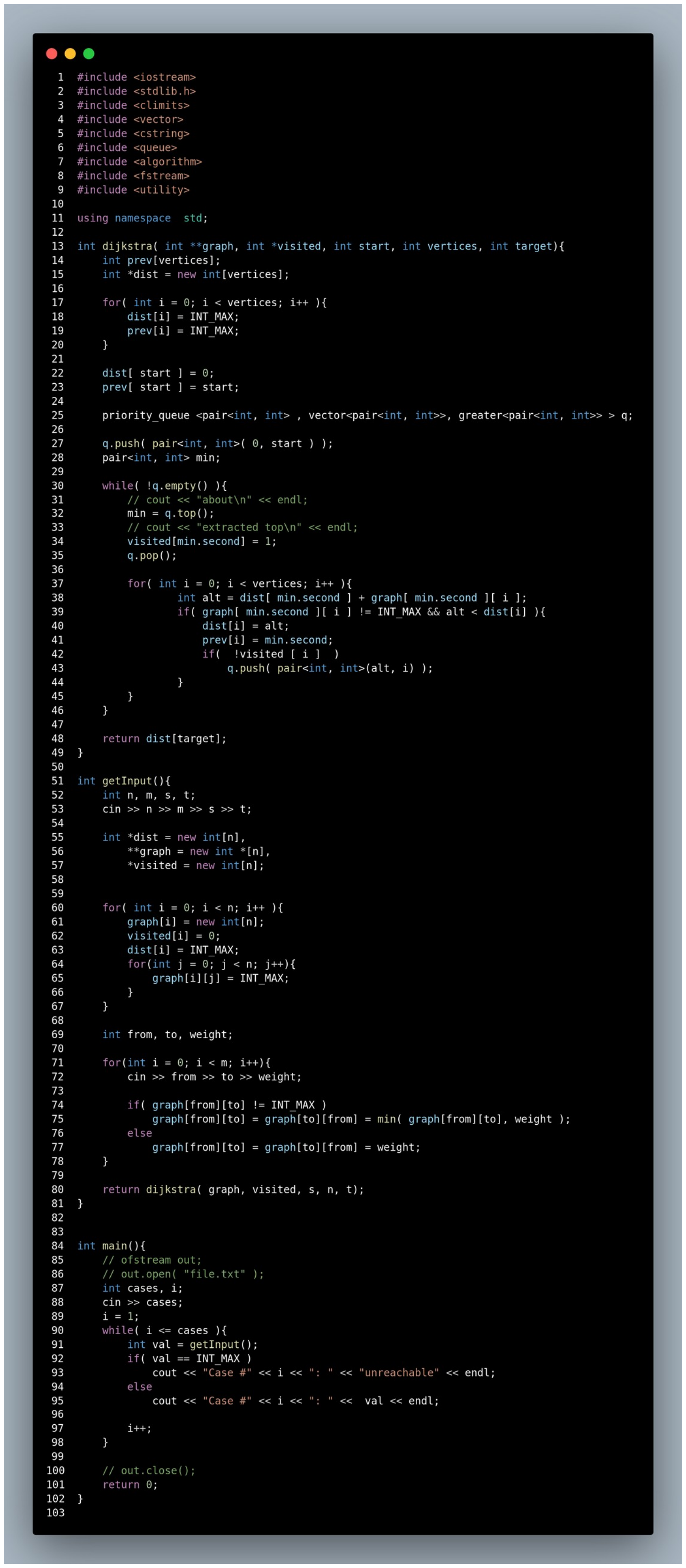
Acceptance statistics:



Accepted

Time: 15ms Memory: 5MB Lang: C++ Author: 411021334

Source code:



```cpp
#include <iostream>
#include <stdlib.h>
#include <climits>
#include <vector>
#include <cstring>
#include <queue>
#include <algorithm>
#include <fstream>
#include <utility>

using namespace  std;

int dijkstra( int **graph, int *visited, int start, int vertices, int target){
    int prev[vertices];
    int *dist = new int[vertices];

    for( int i = 0; i < vertices; i++ ){
        dist[i] = INT_MAX;
        prev[i] = INT_MAX;
    }

    dist[ start ] = 0;
    prev[ start ] = start;

    priority_queue <pair<int, int> , vector<pair<int, int>>, greater<pair<int, int>> > q;

    q.push( pair<int, int>( 0, start ) );
    pair<int, int> min;

    while( !q.empty() ){
        // cout << "about\n" << endl;
        min = q.top();
        // cout << "extracted top\n" << endl;
        visited[min.second] = 1;
        q.pop();

        for( int i = 0; i < vertices; i++ ){
                int alt = dist[ min.second ] + graph[ min.second ][ i ];
                if( graph[ min.second ][ i ] != INT_MAX && alt < dist[i] ){
                    dist[i] = alt;
                    prev[i] = min.second;
                    if(  !visited [ i ]  )
                        q.push( pair<int, int>(alt, i) );
                }
        }
    }

    return dist[target];
}

int getInput(){
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    int *dist = new int[n],
        **graph = new int *[n],
        *visited = new int[n];


    for( int i = 0; i < n; i++ ){
        graph[i] = new int[n];
        visited[i] = 0;
        dist[i] = INT_MAX;
        for(int j = 0; j < n; j++){
            graph[i][j] = INT_MAX;
        }
    }

    int from, to, weight;

    for(int i = 0; i < m; i++){
        cin >> from >> to >> weight;

        if( graph[from][to] != INT_MAX )
            graph[from][to] = graph[to][from] = min( graph[from][to], weight );
        else
            graph[from][to] = graph[to][from] = weight;
    }

    return dijkstra( graph, visited, s, n, t);
}


int main(){
    // ofstream out;
    // out.open( "file.txt" );
    int cases, i;
    cin >> cases;
    i = 1;
    while( i <= cases ){
        int val = getInput();
        if( val == INT_MAX )
            cout << "Case #" << i << ": " << "unreachable" << endl;
        else
            cout << "Case #" << i << ": " <<  val << endl;

        i++;
    }

    // out.close();
    return 0;
}

```