Dijkstra.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

class MyComparator{
    bool reverse;
public:
    MyComparator(const bool & reverse=false){
        this->reverse = reverse;
    }
    bool operator() ( int & a, int & b) {
        if (reverse)
            return a < b;
        return a > b;
    }
};

typedef pair<int, int> iPair;
vector<vector<iPair> > graph (9, vector<iPair>());

void addEdge(int u, int v, int w){
    graph[u].push_back(make_pair(v, w));
    graph[v].push_back(make_pair(u, w));
}

void shortestPath(int src){
    priority_queue<iPair, vector<iPair>, greater<iPair> > pq;
    vector<int> dist(9, INT_MAX);
    pq.push(make_pair(0, src));
    dist[src] = 0;
    int u;
    while (!pq.empty()){
        u = pq.top().second;
        pq.pop();
        vector<iPair>::iterator it;
        for (it = graph[u].begin(); it != graph[u].end(); it++){
            int v = it->first;
            int w = it->second;
            if (dist[v] > dist[u] + w) {
                // Updating distance of v
                dist[v] = dist[u] + w;
                pq.push(make_pair(dist[v], v));
            }}}
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < 9; ++i) printf("%d \t\t %d\n", i, dist[i]);
}

int main(){
    addEdge(0, 1, 4);addEdge(0, 7, 8);addEdge(1, 2, 8);addEdge(1, 7, 11);addEdge(2, 3, 7);
    addEdge(2, 8, 2);addEdge(2, 5, 4);addEdge(3, 4, 9);addEdge(3, 5, 14);addEdge(4, 5, 10);
    addEdge(5, 6, 2);addEdge(6, 7, 1);addEdge(6, 8, 6);addEdge(7, 8, 7);
    shortestPath(0);
    priority_queue<int, vector<int>, MyComparator > pq(MyComparator(true));
    //priority_queue<int, vector<int>, grater<int> > pq;
    pq.push(4);pq.push(8);pq.push(2);
    while (!pq.empty()){
        int tmp = pq.top();
        pq.pop();
        cout << tmp << " ";
    }
    cout << endl; }
```