//Dijkstra Algorithm

#define MAX 100

#define INF 1e6

vector<int>dist, G[MAX], W[MAX];

void printPath(int u) {

// extract information from ‘vi p’

if (u == s) {

printf("%d", s);

return;

} // base case, at the source s

printPath(p[u]); // recursive: to make the output format: s -> ... -> t

printf(" %d", u);

}

void dikjstra(int u, int destination, int nodes) {

//dist[v] contains the distance from u to v

dist.resize(nodes+1, INF);

dist[u] = 0;

//pq is sorted in ascending order according to weight and edge

priority\_queue<pair<int, int> > pq;

pq.push({0, -u});

while(!pq.empty()) {

int u = -pq.top().second;

int wu = -pq.top().first;

pq.pop();

//if we only need distance of destination, then we may return

if(u == destination)

return;

//skipping the longer edges, if we have found shorter edge earlier

if(wu > dist[u])

continue;

for(int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

int wv = W[u][i];

//path relax

if(wu + wv < dist[v]) {

dist[v] = wu + wv;

pq.push({-dist[v], -v});

}

}

}

}