1. **소스코드 및 주석**

#include <iostream>

#include <vector>

#include <queue>

#include <random>

#include <limits.h>

#include <algorithm>

using namespace std;

#define INF 99999999

random\_device rd;

mt19937\_64 rng(rd());

template<typename T>

T RandomNumber(T start, T end) {

uniform\_int\_distribution<T> dist(start, end);

return dist(rng);

}

class Edge {

public:

int node1, node2;

int weight;

Edge(int node1, int node2, int weight) {

this->node1 = node1;

this->node2 = node2;

this->weight = weight;

}

bool operator<(const Edge& edge) const {

return this->weight < edge.weight;

}

};

void printEdge(vector<Edge> v, int vertexCount) {

bool isSomenode = false;

int edgeCount = 0;

for (auto& i : v) {

if (!isSomenode) {

edgeCount += 1;

cout << edgeCount << ". random edge : (" << i.node1 << ", " << i.node2 << "), ";

isSomenode = true;

}

else {

cout << "(" << i.node1 << ", " << i.node2 << ") " << "weight : " << i.weight << endl;

isSomenode = false;

}

}

cout << endl;

cout << "Number of Vertices : " << vertexCount << endl;

}

void CreateGraph(vector<Edge>\* v, int\* vertexCount, int\* edgeCount) {

int node1 = 1, node2 = 1, weight = 1;

int duplicationCheck[11][11] = { 0, };

int vertex[11] = { 0, };

while (\*vertexCount < 10) {

if (RandomNumber(0, 1))

node1 = node2;

node2 = RandomNumber(1, 10);

while (node1 == node2) {

node2 = RandomNumber(1, 10);

}

weight = RandomNumber(1, 20);

if (duplicationCheck[node1][node2]) continue;

if (!vertex[node1]) {

vertex[node1] = 1;

(\*vertexCount)++;

}

if (!vertex[node2]) {

vertex[node2] = 1;

(\*vertexCount)++;

}

duplicationCheck[node1][node2] = duplicationCheck[node2][node1] = 1;

v->push\_back(Edge(node1, node2, weight));

v->push\_back(Edge(node2, node1, weight));

(\*edgeCount)++;

}

cout << "Random Matrix Generation!!" << endl;

printEdge(\*v, \*vertexCount);

}

// 다익스트라 + 경로 추적용 prev 배열 반환

pair<vector<int>, vector<int>> dijkstra(int start, int N, vector<vector<pair<int, int>>>& graph) {

vector<int> dist(N + 1, INF);

vector<int> prev(N + 1, -1);

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

dist[start] = 0;

pq.push({ 0, start });

while (!pq.empty()) {

int cur\_dist = -pq.top().first;

int cur\_node = pq.top().second;

pq.pop();

if (cur\_dist > dist[cur\_node]) continue;

for (auto& next : graph[cur\_node]) {

int nxt\_node = next.first;

int nxt\_dist = cur\_dist + next.second;

if (nxt\_dist < dist[nxt\_node]) {

dist[nxt\_node] = nxt\_dist;

prev[nxt\_node] = cur\_node;

pq.push({ -nxt\_dist, nxt\_node });

}

}

}

return { dist, prev };

}

// 경로 출력 함수

void printPath(int target, const vector<int>& prev, const vector<int>& dist) {

vector<int> path;

int cur = target;

while (cur != -1) {

path.push\_back(cur);

cur = prev[cur];

}

reverse(path.begin(), path.end());

for (int i = 0; i < path.size(); ++i) {

cout << path[i];

if (i < path.size() - 1) cout << " -> ";

}

cout << " (Cost: " << dist[target] << ")" << endl;

}

int main() {

vector<Edge> v;

int vertexCount = 0, edgeCount = 0;

CreateGraph(&v, &vertexCount, &edgeCount);

cout << "vCount : " << edgeCount << endl;

int maxNode = 10;

vector<vector<pair<int, int>>> graph(maxNode + 1);

for (const auto& edge : v) {

graph[edge.node1].push\_back({ edge.node2, edge.weight });

}

int start = 1;

pair<vector<int>, vector<int>> result = dijkstra(start, maxNode, graph);

vector<int> dist = result.first;

vector<int> prev = result.second;

cout << "\n[1번 정점에서 다른 모든 정점까지의 최단 경로]\n";

for (int i = 1; i <= maxNode; ++i) {

if (i != start && dist[i] != INF) {

printPath(i, prev, dist);

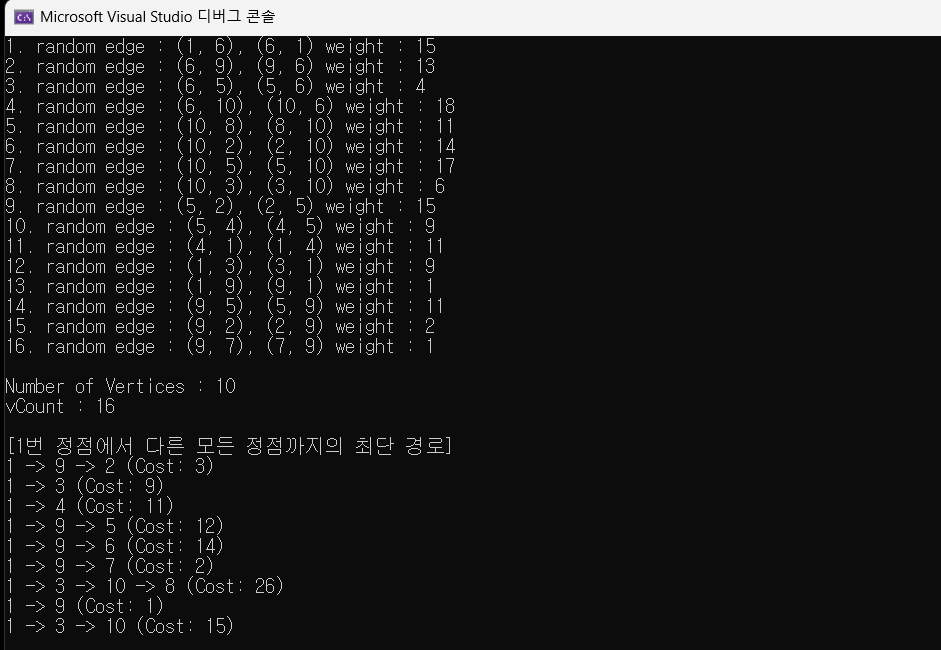
}

}

return 0;

}

1. **실행화면 캡처**



1. **고찰**

* 그래프에서 시작 노드에서 모든 노드까지의 최단 경로를 구하는 방식임을 알았습니다
  + 다익스트라를 쓰지않고 직접 손으로 그려보니 똑같은 것을 보아 직관적인 것 같습니다
* 우선순위 큐를 이용하여 우선적으로 가중치의 합을 구할때, priority\_queue<pair<int, int>> pq; 를 사용해야합니다
  + 유선순위 큐를 사용하여 노드가 많아질때, 시간복잡도가 줄어듭니다
* 간선이 많을때, 시간이 많이 걸리는 것같습니다