1. **소스코드 및 주석**
   1. **문제1**

#include <stdio.h>

#include <vector>

#include <random>

#include <iostream>

using namespace std;

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;

}

};

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

bool isSomenode = false;

int edgeCount = 1;

for (auto& i : v) {

if (isSomenode == false) {

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

isSomenode = true;

edgeCount += 1;

}

else {

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

isSomenode = false;

}

}

cout << endl;

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

cout << "Number of Edges : " << edgeCount << endl;

}

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

int node1=0, node2=0, weight=0;

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

for (auto& i : v) {

node1 = i.node1;

node2 = i.node2;

weight = i.weight;

matrix[node1][node2] = weight;

}

cout << endl;

cout << "Adjacency Matrix of Direct Graph" << endl;

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

for (int j = 1; j <= vertexCount; j++) {

if (matrix[i][j] == 0)

cout << " 0";

else if(matrix[i][j] < 10)

cout << " " << matrix[i][j];

else if (matrix[i][j] >= 10)

cout << " " << matrix[i][j];

}

cout << endl;

}

cout << endl;

}

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

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

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

int vertex[11] = { 0, };

node1 = RandomNumber(1, 10);

node2 = RandomNumber(1, 10);

while (\*vertexCount < 10) {

if (RandomNumber(0, 1))

node1 = node2;

else

node1 = node1;

node2 = RandomNumber(1, 10);

while (node1 == node2) {

node2 = RandomNumber(1, 10);

}

weight = RandomNumber(1, 20);

if (duplicationCheck[node1][node2] == 1)

continue;

if (vertex[node1] == 0) {

vertex[node1] = 1;

\*vertexCount += 1;

}

if (vertex[node2] == 0) {

vertex[node2] = 1;

\*vertexCount += 1;

}

duplicationCheck[node1][node2] = 1;

duplicationCheck[node2][node1] = 1;

(\*v).push\_back(Edge(node1, node2, weight));

(\*v).push\_back(Edge(node2, node1, weight));

}

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

printEdge(\*v, \*vertexCount);

}

int main() {

vector<Edge> v;

int vertexCount = 0;

CreateGraph(&v, &vertexCount);

printMatrixGraph(v, vertexCount);

return 0;

}

* 1. **문제2**

#include <stdio.h>

#include <vector>

#include <algorithm>

#include <random>

#include <iostream>

using namespace std;

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) {

for (auto& i : v) {

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

}

cout << endl;

}

void printEdge2(vector<Edge> v, int vertexCount) {

bool isSomenode = false;

int edgeCount = 1;

for (auto& i : v) {

if (isSomenode == false) {

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

isSomenode = true;

edgeCount += 1;

}

else {

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

isSomenode = false;

}

}

cout << endl;

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

cout << "Number of Edges : " << edgeCount << endl;

}

int check[11];

int getParent(int node) {

if (check[node] == node)

return node;

return getParent(check[node]);

}

void unionParent(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 < node2)

check[node2] = node1;

else

check[node1] = node2;

}

bool isCycle(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 == node2) return true;

else return false;

}

void CreateGraph(vector<Edge>\* v) {

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

int vertexCount = 0;

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

int vertex[11] = { 0, };

node1 = RandomNumber(1, 10);

node2 = RandomNumber(1, 10);

while (vertexCount < 10) {

if (RandomNumber(0, 1))

node1 = node2;

else

node1 = node1;

node2 = RandomNumber(1, 10);

while (node1 == node2) {

node2 = RandomNumber(1, 10);

}

weight = RandomNumber(1, 20);

if (duplicationCheck[node1][node2] == 1)

continue;

if (vertex[node1] == 0) {

vertex[node1] = 1;

vertexCount += 1;

}

if (vertex[node2] == 0) {

vertex[node2] = 1;

vertexCount += 1;

}

duplicationCheck[node1][node2] = 1;

duplicationCheck[node2][node1] = 1;

(\*v).push\_back(Edge(node1, node2, weight));

(\*v).push\_back(Edge(node2, node1, weight));

}

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

printEdge2(\*v, vertexCount);

}

int main() {

vector<Edge> v;

vector<Edge> mst;

CreateGraph(&v);

sort(v.begin(), v.end());

for (int i = 1;i <= 10;++i) {

check[i] = i;

}

int sum = 0;

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

if (!isCycle(v[i].node1, v[i].node2)) {

sum += v[i].weight;

unionParent(v[i].node1, v[i].node2);

mst.push\_back(v[i]);

}

}

cout<< endl;

cout << "Minimum Cost : " << sum << endl;

cout << "Minimum Spanning Tree" << endl;

printEdge(mst);

return 0;

}

* 1. **문제3**

#include <stdio.h>

#include <vector>

#include <algorithm>

#include <random>

#include <iostream>

using namespace std;

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) {

for (auto& i : v) {

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

}

cout << endl;

}

void printEdge2(vector<Edge> v, int vertexCount) {

bool isSomenode = false;

int edgeCount = 1;

for (auto& i : v) {

if (isSomenode == false) {

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

isSomenode = true;

edgeCount += 1;

}

else {

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

isSomenode = false;

}

}

cout << endl;

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

cout << "Number of Edges : " << edgeCount << endl;

}

int check[11];

int getParent(int node) {

if (check[node] == node)

return node;

return getParent(check[node]);

}

void unionParent(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 < node2)

check[node2] = node1;

else

check[node1] = node2;

}

bool isCycle(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 == node2) return true;

else return false;

}

void CreateGraph(vector<Edge>\* v) {

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

int vertexCount = 0;

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

int vertex[11] = { 0, };

node1 = RandomNumber(1, 10);

node2 = RandomNumber(1, 10);

while (vertexCount < 10) {

if (RandomNumber(0, 1))

node1 = node2;

else

node1 = node1;

node2 = RandomNumber(1, 10);

while (node1 == node2) {

node2 = RandomNumber(1, 10);

}

weight = RandomNumber(1, 20);

if (duplicationCheck[node1][node2] == 1)

continue;

if (vertex[node1] == 0) {

vertex[node1] = 1;

vertexCount += 1;

}

if (vertex[node2] == 0) {

vertex[node2] = 1;

vertexCount += 1;

}

duplicationCheck[node1][node2] = 1;

duplicationCheck[node2][node1] = 1;

(\*v).push\_back(Edge(node1, node2, weight));

(\*v).push\_back(Edge(node2, node1, weight));

}

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

printEdge2(\*v, vertexCount);

}

int main() {

vector<Edge> v;

vector<Edge> mst;

CreateGraph(&v);

sort(v.begin(), v.end());

for (int i = 1;i <= 10;++i) {

check[i] = i;

}

int sum = 0;

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

if (!isCycle(v[i].node1, v[i].node2)) {

sum += v[i].weight;

unionParent(v[i].node1, v[i].node2);

mst.push\_back(v[i]);

}

else {

cout << "edge (" << v[i].node1 << ", " << v[i].node2 << ") is excluded because it creates a cycle." << endl;

}

}

cout << endl;

cout << "Minimum Cost : " << sum << endl;

cout << "Minimum Spanning Tree" << endl;

printEdge(mst);

return 0;

}

* 1. **문제 3+ a**

#include <stdio.h>

#include <vector>

#include <algorithm>

#include <random>

#include <iostream>

using namespace std;

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) {

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

cout << " edge : (" << v[i].node1 << ", " << v[i].node2 << ") " << endl;

}

cout << endl;

}

void printEdge2(vector<Edge> v, int vertexCount) {

bool isSomenode = false;

int edgeCount = 1;

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

if (isSomenode == false) {

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

isSomenode = true;

edgeCount += 1;

}

else {

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

isSomenode = false;

}

}

cout << endl;

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

cout << "Number of Edges : " << edgeCount << endl;

}

int check[11];

int getParent(int node) {

if (check[node] == node)

return node;

return getParent(check[node]);

}

void unionParent(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 < node2)

check[node2] = node1;

else

check[node1] = node2;

}

bool isCycle(int node1, int node2) {

node1 = getParent(node1);

node2 = getParent(node2);

if (node1 == node2) return true;

else return false;

}

void CreateGraph(vector<Edge>\* v) {

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

int vertexCount = 0;

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

int vertex[11] = { 0, };

node1 = RandomNumber(1, 10);

node2 = RandomNumber(1, 10);

while (vertexCount < 10) {

if (RandomNumber(0, 1))

node1 = node2;

else

node1 = node1;

node2 = RandomNumber(1, 10);

while (node1 == node2) {

node2 = RandomNumber(1, 10);

}

weight = RandomNumber(1, 20);

if (duplicationCheck[node1][node2] == 1)

continue;

if (vertex[node1] == 0) {

vertex[node1] = 1;

vertexCount += 1;

}

if (vertex[node2] == 0) {

vertex[node2] = 1;

vertexCount += 1;

}

duplicationCheck[node1][node2] = 1;

duplicationCheck[node2][node1] = 1;

(\*v).push\_back(Edge(node1, node2, weight));

(\*v).push\_back(Edge(node2, node1, weight));

}

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

printEdge2(\*v, vertexCount);

}

int main() {

vector<Edge> v;

vector<Edge> mst;

CreateGraph(&v);

for (int i = 1;i <= 10;++i) {

check[i] = i;

}

int sum = 0;

cout << endl;

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

if(i==0) {

cout << "첫번째 생성된 edge : (" << v[i].node1 << ", " << v[i].node2 << ") " << "는 반드시 포함됩니다" << endl;

sum += v[i].weight;

unionParent(v[i].node1, v[i].node2);

mst.push\_back(v[i]);

}

else if (!isCycle(v[i].node1, v[i].node2)) {

sum += v[i].weight;

unionParent(v[i].node1, v[i].node2);

mst.push\_back(v[i]);

}

}

sort(v.begin(), v.end());

cout << "Minimum Cost : " << sum << endl;

cout << "Minimum Spanning Tree" << endl;

printEdge(mst);

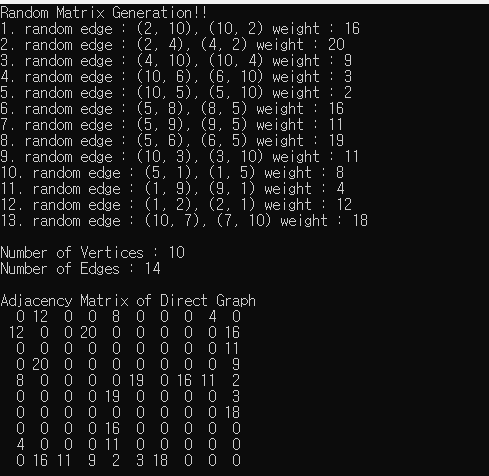
return 0;

}

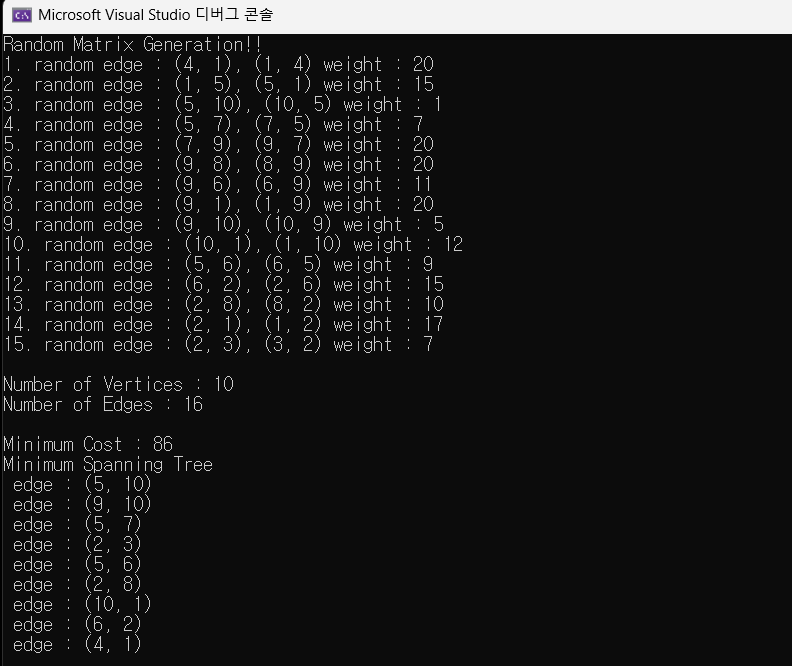
1. **실행화면 캡처**

* 10개의 1~10의 꼭지점을 생성한 후 연결 그래프가 생성될때까지 랜덤 edge를 생성하였습니다
* 가중치는 1~20의 랜덤값입니다
* 모든 출력이 정상적으로 계산되어 출력됨을 확인하였습니다

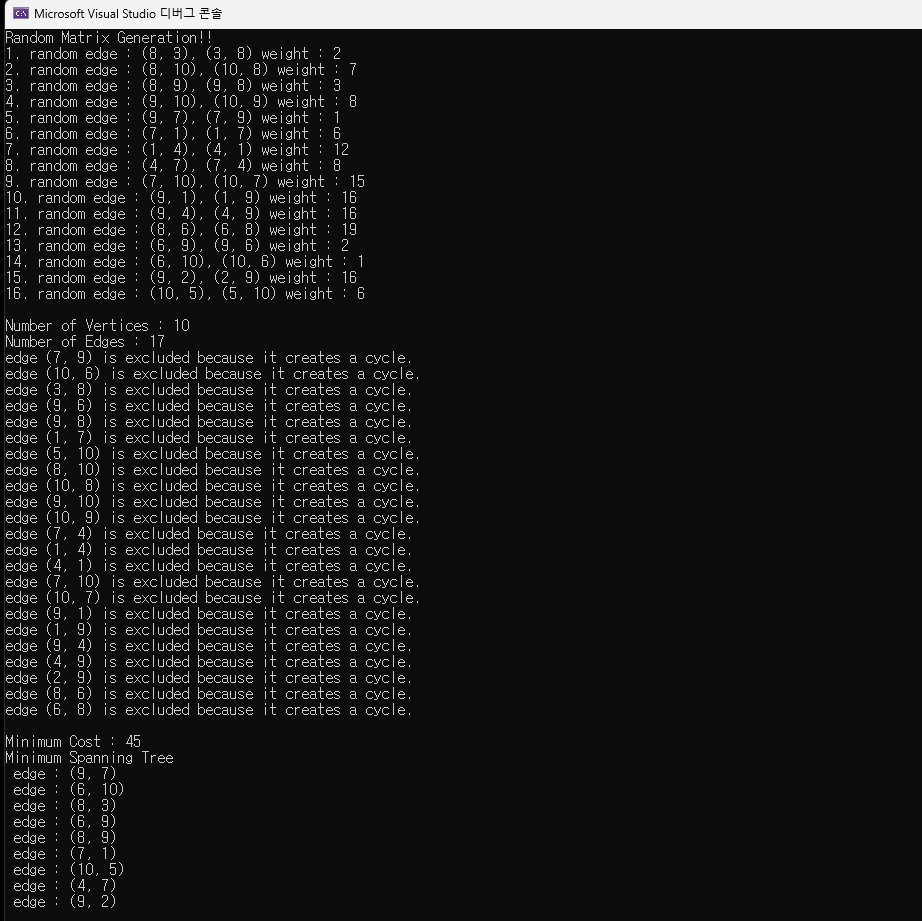
**2-1. 문제1 출력**



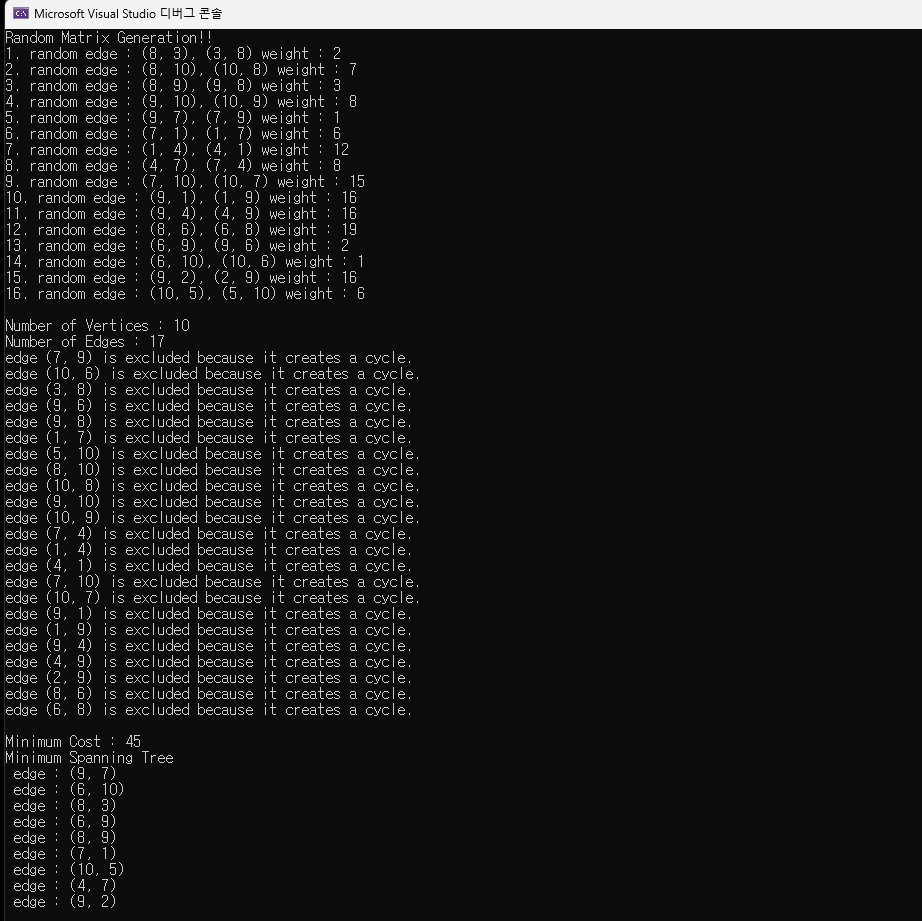
**2-2. 문제2 출력**



**2-3. 문제3 출력**



**2-4. 문제3+a 출력**



1. **고찰**

* 네비게이션에서 사용하면 효율적일거같다는 생각이 들었습니다
* MST 계산이 헷갈려서 실제로 그려보면서 확인해보니 크루스칼 알고리즘 방식이 직관적인 방식이라는 생각이 들었습니다
* Edge를 클래스로 생성하였지만 더 효율적인 자료구조가 있을거같다는 생각이 들었습니다
* 제코드를 좀더 리펙토링하면 더 효율적인 코드가 될 수 있을거같습니다