```
*********************************************************************************
* Name : 차윤성
* Student ID : 20163162
* Program ID : Hw#6
* Description : 파일로부터 각각의 그래프 별로 가중치 배열을 입력 받고 기준 정점으로부터 각 정점까
*               지의 최단거리를 구하는 프로그램이다.
* Algorithm
* - 파일로부터 Cost Matrix를 입력 받고 그 그래프의 정점의 개수 및 그래프에서의 최대 가중치(기준 정
*   점으로부터 직접적으로 연결되어 있지 않을 경우)를 초기화 해준다,
* - printCostMatrix 함수를 통해 그래프 별 Cost Matrix를 출력한다.
* - 배열 found의 각 인덱스 값을 전부 false로 초기화해준다.
* - 배열 distance의 각 인덱스 값을 매개변수로 받은 기준정점 v에서부터의 각 정점으로 가는 거리로 초
*   기화 해준다.
* - 배열 found와 distance의 v번째 인덱스의 값을 각각 true와 0으로 초기화해준다.
* - 이로써 현재 집합에는 기준정점 v와 v에서 v까지의 거리가 0으로 설정된 것이다.
* - 최단 거리를 찾기 전 초기 Distance를 출력해준다,
* - ChooseNode 함수를 통해 현재 정점으로부터 이동할 다음 정점을 선택하고 u에 저장한다.
* - found[u]를 true로 초기화해준다. 집합에 포함되었다는 의미이다.
* - found의 마지막 인덱스까지 검사한다.
* - found의 w번재 인덱스가 집합에 포함되어 있지 않고, 기준정점으로부터 현재 이동해 있는 정점 u까
*   지의 거리와 정점u에서 정점w까지의 거리의 합이 기준정점으로부터 w까지의 최단거리보다 작을 경우
*   w까지의 최단거리를 이 두 값의 합으로 초기화시켜준다.
* - distance의 마지막 인덱스까지 검사가 끝나면 다음 정점을 선택하고, 정점의 개수 - 1만큼 반복한다.
* - ChooseNode는 다음번으로 이동할 정점 minNode를 반환하는 함수이다.
* - 최소 거리를 저장할 변수 min을 먼저 그래프의 최대 가중치로 초기화한다.
* - distance의 정점의 개수까지의 인덱스를 검사하면서 현재 기준정점 으로부터 정점 i까지의 최소
*   거리보다 min이 큰 값을 가지고, found 배열에서 i번재 인덱스의 값이 true가 아닐 경우, 즉 정점을
*   들리지 않았을 경우 min의 값을 현재 기준정점으로부터 정점 i까지의 최소 거리 distance[i]로 초기화
*   해주고, minNode를 정점 i로 초기화한다.
* - distance의 마지막 인덱스까지 반복검사 한 후 검사가 종료되면 현재 distance 중 최소값을 가지는
*   인덱스를 minNode에 저장하고, minNode를 반환한다.
* Variables
* - found : 현재 정점이 집합에 포함 되어있는지 없는지 검사하기 위한 bool형 배열 변수이다.
* - distance : 기준정점으로부터 각 인덱스별 정점까지의 최단 거리가 저장되는 int형 배열 변수이다.
* - cost : 파일로부터 주어진 Cost Matrix가 저장될 int형 배열 변수이다.
* - numOfVertex : 주어진 정점의 총 개수가 저장될 int형 변수이다.
* - MAX : Cost Matrix에서 주어진 최대값이 저장되는 int형 변수이다.
* - u : 현재 정점 이후 다음번 정점이 저장될 int형 변수이다.
* - initCostMatrix1 : TestCase1에 대한 Cost Matrix를 초기화 해주는 함수이다.
* - initCostMatrix2 : TestCase2에 대한 Cost Matrix를 초기화 해주는 함수이다.
* - printCostMatrix : Cost Matrix를 출력하는 함수이다.
* - printDistance : 현재 distance 배열을 출력하는 함수이다.
* - shortestPath1 : TestCase1에 대한 shortest path를 구하는 함수이다.
* - shortestPath2 : TestCase2에 대한 shortest path를 구하는 함수이다.
* - chooseNode : 다음번에 몇 번 정점을 고를지 검사하는 함수이다.
*********************************************************************************
#include <iostream>
#include <fstream>

using namespace std;
*********************************************************************************
* class : ShPath
* description : ShPath는 Cost Matirx를 토대로 shortest path를 구하고 저장하고 있는 객체이다.
* variables
* - found : 현재 정점이 집합 안에 포함 되어있는지 없는지 검사하는 배열이다.
* - distance : 기준 정점으로부터 인덱스 별 정점까지의 최단 거리가 저장되는 배열이다.
* - cost : Cost Matrix가 저장되는 2차원 배열이다.
* - numOfVertex : 정점의 총 갯수가 저장되는 변수이다.
* - MAX : cost배열 내 최댓값이 저장되는 배열이다.
*********************************************************************************
class ShPath {
private:
        bool *found;    // 정점이 현재 집합 안에 포함 되어있는지 없는지 검사하는 배열
        int *distance;  // 기준정점으로 부터 V[index]까지의 최단 거리가 저장될 배열
        int **cost;     // 주어진 Cost Matrix가 저장될 2차원 배열
```

```
	int numOfVertex;		// 정점의 총 갯수
	int MAX;		// cost 배열 내 최댓값(기준 정점에서 도달할 수 없는 경우)
public:
	ShPath() {
		found = new bool[100];
		distance = new int[100];
	}
	void initCostMatrix1();	// testcase 1
	void initCostMatrix2();	// testcase 2
	void printCostMatrix();
	void printDistance();
	void shortestPath1(int v);		// testcase 1
	void shortestPath2(int v);		// testcase 2
	int chooseNode();
};
*****************************************************************************************
* function : initCostMatrix1
* description : initCostMatrix1의 사용목적은 TestCase1에 대해 Cost Matrix를 초기화해주는 것이다.
*****************************************************************************************
void ShPath::initCostMatrix1() {
	ifstream infile;
	infile.open("data1.txt", ios::in);

	infile >> numOfVertex;

	cost = new int*[numOfVertex];
	for (int i = 0; i < numOfVertex; i++) {
		cost[i] = new int[numOfVertex];
		for (int j = 0; j < numOfVertex; j++) {
			infile >> cost[i][j];
		}
	}
	MAX = 100;
	infile.close();
}
*****************************************************************************************
* function : initCostMatrix2
* description : initCostMatrix2의 사용목적은 TestCase2에 대해 Cost Matrix를 초기화해주는 것이다.
*****************************************************************************************
void ShPath::initCostMatrix2() {
	ifstream infile;
	infile.open("data2.txt", ios::in);

	infile >> numOfVertex;

	cost = new int*[numOfVertex];
	for (int i = 0; i < numOfVertex; i++) {
		cost[i] = new int[numOfVertex];
		for (int j = 0; j < numOfVertex; j++) {
			infile >> cost[i][j];
		}
	}
	MAX = 10000;
	infile.close();
}
```

```
*****************************************************************************************
* function : printCostMatrix
* description : printCostMatrix의 사용 목적은 Cost Matrix를 출력하는 것이다.
*****************************************************************************************
void ShPath::printCostMatrix() {
	cout << "\t";
	for (int i = 0; i < numOfVertex; i++) {
		cout << "V" << i << "\t";
	}
	cout << endl;
	for (int i = 0; i < numOfVertex; i++) {
		cout << "V" << i << "\t";
		for (int j = 0; j < numOfVertex; j++) {
			cout << cost[i][j] << "\t";
		}
		cout << endl;
	}
}
*****************************************************************************************
* function : printDistance
* description : printDistance의 사용 목적은 distance배열을 출력하는 것이다.
*****************************************************************************************
void ShPath::printDistance() {
	for (int i = 0; i < numOfVertex; i++) {
		cout << distance[i] << " ";
	}
}
*****************************************************************************************
* function : shortestPath1
* description : shortestPath1의 사용 목적은 TestCase1에 대해서 매개변수로 받은 기준정점 v에 대해
*		각 정점에 대한 최단거리를 구하는 것이다.
* variables
* - i,w : 배열의 인덱스를 가리키는 변수이다.
* - u : 다음번에 선택된 정점을 가리키는 변수이다.
*****************************************************************************************
void ShPath::shortestPath1(int v) {
	int i, u, w;
	for (i = 0; i < numOfVertex; i++) {
		found[i] = false;
		distance[i] = cost[v][i];
		// 기준 정점(V(v))로 부터 직접적으로 거치는 정점들에 대한 가중치 초기화
	}

	found[v] = true; // 기준 정점을 집합에 포함시킴
	distance[v] = 0; // 기준 정점에 대한 가중치 0으로 초기화
	for (i = 0; i < numOfVertex - 1; i++) {
		cout << "  Dist : ";
		printDistance();

		u = chooseNode();		// 다음번으로 거칠 정점 u를 찾는다
		found[u] = true;
		cout << "\t(Selected Node : " << u << ")" << endl;
		for (w = 0; w < numOfVertex; w++) {
			if (!found[w]) {  // 정점 V(w)가 집합에 포함되어있지 않으면
				if (distance[u] + cost[u][w] < distance[w])
				// 현재 정점 u에서부터 다음 정점 w까지 가중치의 합이
				// 이전까지 찾은 기준정점으로부터 정점 w까지의 가중치보다 작으면
					distance[w] = distance[u] + cost[u][w];
			}
		}
	}
}
```

```
*****************************************************************************************
* function : shortestPath2
* description : shortestPath2의 사용 목적은 TestCase2에 대해서 매개변수로 받은 기준정점 v에 대해
*               각 정점에 대한 최단거리를 구하는 것이다.
* variables
* - i,w : 배열의 인덱스를 가리키는 변수이다.
* - u : 다음번에 선택된 정점을 가리키는 변수이다.
*****************************************************************************************
void ShPath::shortestPath2(int v) {
        int i, u, w;
        for (i = 0; i < numOfVertex; i++) {
                found[i] = false;
                distance[i] = cost[v][i];
        }

        found[v] = true;
        distance[v] = 0;
        cout << "Beginning    Dist : ";
        for (i = 0; i < numOfVertex - 1; i++) {

                printDistance();
                cout << endl;

                u = chooseNode();
                found[u] = true;

                for (w = 0; w < numOfVertex; w++) {
                        if (!found[w]) {
                                if (distance[u] + cost[u][w] < distance[w])
                                        distance[w] = distance[u] + cost[u][w];
                        }
                }
                cout << "Select(" << u << ")-> Dist : ";
        }
        printDistance(); // 결과값 도출
        cout << endl;
}
*****************************************************************************************
* function : chooseNode
* description : chooseNode의 사용 목적은 다음번으로 선택될 정점을 구하는 것이다.
* variable
* - min : 최단거리가 저장되고 어떤 거리가 제일 작은지 비교하기 위한 변수이다.
* - minNode : 최단거리가 저장되어있는 정점을 반환해줄 변수이다.
*****************************************************************************************
int ShPath::chooseNode() {
        int min, minNode;
        min = MAX;      // cost배열 내 최댓값으로 초기화

        for (int i = 0; i < numOfVertex; i++) {
                if (distance[i] < min && !found[i]) {
                        // 기준정점으로 부터 정점 i까지의 값이 min보다 작고
                        // i번째 정점이 현재 집합에 포함되있지 않을 때
                        min = distance[i];
                        minNode = i;
                }
        }
        return minNode;
}
```

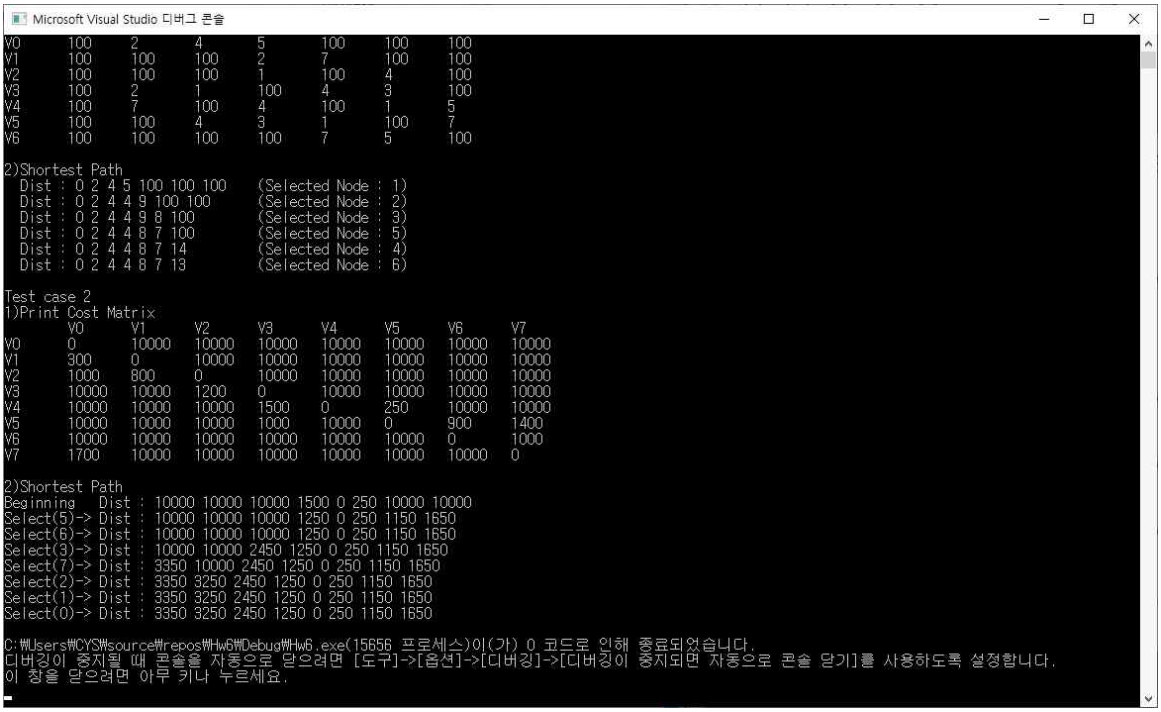
```
int main() {
	ShPath sh, sh1;
	cout << "Test case 1" << endl;
	sh.initCostMatrix1();
	cout << "1)Print Cost Matrix" << endl;
	sh.printCostMatrix();
	cout << "\n2)Shortest Path" << endl;
	sh.shortestPath1(0);

	cout << endl;

	cout << "Test case 2" << endl;
	sh1.initCostMatrix2();
	cout << "1)Print Cost Matrix" << endl;
	sh1.printCostMatrix();
	cout << "\n2)Shortest Path" << endl;
	sh1.shortestPath2(4);
}
```

****

* Output



****