```
// ********************************************************************************************************************
// * Name : 이성재
// * Student ID : 20132651
// * Program ID : HW5_Kruskal_Main.cpp
// *
// * Algorithm Description
// * : 텍스트 데이터로 주어진 그래프 데이터를 링크드 리스트 형태의 데이터로 변환하고, 벡터 배열에 저장한 다음 Kruskal's Algorithm을 통해
// * : Minimum Spanning Tree 를 찾아내는 알고리즘입니다. 이 알고리즘은 다음과 같은 순서로 작동합니다.
// * : 1. 먼저 input.txt 파일에 저장된 그래프 데이터를 읽어와 Adjacency List 와 Vector 형태로 저장합니다.
// * : 2. 이렇게 저장된 그래프 데이터를 확인하기 위해 작성된 printList 함수를 호출하여 그래프 데이터를 출력합니다.
// * : 3. Sorting 알고리즘을 이용해 벡터 배열에 저장되어 있는 노드 형태의 그래프 데이터들을 Edge의 Cost에 따라 오름차순으로 정렬합니다.
// * : 4. 정렬 여부를 확인하기 위해 printSortedData 함수를 호출하여 그래프 데이터를 출력합니다.
// * : 5. 정렬된 그래프 데이터를 순차적으로 꺼내와 Cycle 여부를 확인하고, 사용 가능할 경우 kruskal 벡터에 저장합니다.
// * : 6. 최종적으로 sortKruskalMST 함수를 호출하여 저장된 kruskal 그래프 데이터를 정렬하고, 출력합니다.
// *
// * Variables
// * : Kruskal MST를 구현하기 위한 변수들은 헤더파일과 CPP파일에 정의되어 있습니다.
// * : Main 함수에서는 정의된 함수들의 호출만 이루어집니다.
// ********************************************************************************************************************

#include "HW5_Kruskal.hpp"

// ********************************************************************************************************************
// * function: int main
// * description
// * : 1. main 함수에서는 헤더파일에 정의된 함수들을 순차적으로 호출합니다. 이 때, 정렬된 데이터를 바탕으로 MST의 탐색이 이루어지므로,
// * :    함수가 호출되는 순서가 중요합니다.
// * : 2. 가장 먼저 input.txt 파일의 데이터를 Adjacency List와 Vector 로 변환하는 inputToArray(), arrayToList() 함수를 호출합니다.
// * : 3. 다음으로 저장된 데이터를 정렬하는 sortData 를 호출합니다. 이 때 저장되어있는 그래프의 모든 노드가 Cost에 따라 오름차순으로 정렬됩니다.
// * : 4. 정렬된 데이터는 graph 라는 이름의 벡터에 저장되며, 이는 kruskalMST() 함수에 의해 트리에 사용될 노드만 선택되어 다시 kruskal 벡터에 저장됩니다.
// * : 5. 최종적으로 kruskal 알고리즘에 의해 저장된 노드들은 다시 Cost에 따라 오름차순으로 정렬되며, printKruskalMST()함수에 의해 출력됩니다.
// ********************************************************************************************************************

int main(int argc, char const *argv[]) {

  inputToArray();
  arrayToList();
  printList();

  sortData();
  printSortedData();

  kruskalMST();
  sortKruskalMST();
  printKruskalMST();

  return 0;
}

// ********************************************************************************************************************
// * Name : 이성재
```

```
// * Student ID : 20132651
// * Program ID : HW5_Kruskal.hpp
// *
// * Description
// * 헤더파일 및 정의사항
// * : #ifndef / #define / #endif : HW5_Kruskal.hpp 헤더파일이 중복정의되지 않도록 설정합니다.
// * : iostream : 콘솔창에 결과를 출력하기 위한 헤더파일입니다.
// * : fstream : 입력을 텍스트 데이터 파일 형태로 받기 위한 파일 입출력 헤더파일입니다.
// * : vector : 그래프 데이터의 노드들을 손쉽게 다루기 위한 벡터 배열을 사용하기 위한 STL 헤더파일입니다.
// * : algorithm : 저장된 노드를 한 번에 정렬하기 위한 알고리즘 STL 헤더파일입니다.
// * : using namespace std : 기본 namespace 를 std 로 설정합니다.
// * : define MAX 6 : 현재 다루고자 하는 그래프가 6개의 점으로 이루어져 있으므로, MAX값을 6으로 저장합니다.
// *
// * Node 객체의 정의
// * : char myself : 두 점으로 이어져있는 하나의 그래프 노드 데이터에서, 자기 자신으로 칭하는 하나의 점을 문자 형태로 정의합니다.
// * : int edges : 하나의 그래프 노드에서, 두 점 사이의 거리를 이동하는 데 소요되는 비용을 정수 형태로 정의합니다.
// * : char linked : 두 점으로 이어진 노드중, 자기 자신을 제외한 다른 한 점을 문자 형태로 정의합니다.
// * : Node *next : Adjacency List로 표현하기 위해, 각각의 노드가 다음 노드를 가리키는 노드 포인터 변수 next를 가집니다.
// *
// * : Node (char myself, int edges, char linked) : 위에서 정의한 세 가지 요소를 받아 노드를 생성하는 생성자 함수를 정의합니다.
// * : bool operator < / > : 노드를 오름차순 혹은 내림차순으로 정렬하기 위한 operator를 정의합니다. 이 때 노드의 비교는
// * :                      노드의 비용인 edges 를 기준으로 하게 됩니다.
// ************************************************************************************************************************

#ifndef _KRUSKAL_H_
#define _KRUSKAL_H_

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#define MAX 6

using namespace std;

class Node{
public:
  char myself;
  int edges;
  char linked;
  Node *next = NULL;

  Node(char myself, int edges , char linked){
    this->myself = myself;
    this->edges = edges;
    this->linked = linked;
  }

  bool operator > (const Node &str) const{
    return (edges > str.edges);
  }
  bool operator < (const Node &str) const{
    return (edges < str.edges);
  }
};
```

```
// ********************************************************************************************************************
// * Name : 이성재
// * Student ID : 20132651
// * Program ID : HW5_Kruskal.hpp
// *
// * Description
// * 함수 프로토타입 정의
// * : 함수는 크게 1단계와 2단계로 나누어져 있습니다.
// * : 1단계에서는 파일에서 그래프 데이터를 저장 및 정렬하며, 2단계에서는 저장된 데이터로 최소 비용의 Spanning Tree
를 탐색합니다.
// *
// * : < 1단계 >
// * : inputToArray() : input.txt 데이터 파일을 읽어와 graph 벡터에 배열 형태로 저장합니다.
// * : arrayToList() : 저장된 데이터를 Adjacency Linked List 형태로 변환하여 저장합니다.
// * : printList() : 리스트 형태로 저장된 데이터를 출력합니다.
// * : sortData() : 가장 먼저 저장되었던 graph 벡더에 저장된 그래프 데이터를 sorting 알고리즘을 이용해 cost 순으로 정
렬합니다.
// * : printSortedData() : 벡터 내부에 정렬된 그래프 데이터를 순차적으로 출력합니다.
// * :
// * : < 2단계 >
// * : allVisited() : 모든 노드가 방문하였는지 체크하여, true 혹은 false를 반환하는 함수입니다.
// * : kruskalMST() : 앞에서 정의한 allVisited 함수를 계속해서 체크하며, 적은 비용 순으로 노드를 선택하는 함수입니다.
// * :              선택된 노드는 visited 배열의 해당 위치에 방문 여부가 체크되며, 이를 통해 Cycle이 발생하지 않습니다.
// * :              또한 선택된 노드는 최종적으로 kruskal 벡터 배열에 저장되며, 결과값으로 출력됩니다.
// * : sortKruskalMST() : 앞에서 kruskal 벡터 배열에 저장된 노드들을 cost 오름차순으로 정렬합니다.
// * : printKruskalMST() : 최종적으로 Kruskal's Algorithm 에 의해 선택된 Minimum Spanning Tree를 출력합니다.
// *
// ********************************************************************************************************************

void inputToArray();
void arrayToList();
void printList();
void sortData();
void printSortedData();

bool allVisited();
void kruskalMST();
void sortKruskalMST();
void printKruskalMST();

#endif
```

```
// ********************************************************************************************************************
// * Name : 이성재
// * Student ID : 20132651
// * Program ID : HW5_Kruskal.cpp
// * description : HW5_Kruskal.hpp 헤더 파일에서 구현된 프로토타입 함수를 구현합니다.
// ********************************************************************************************************************

#include "HW5_Kruskal.hpp"

// ********************************************************************************************************************
// * Variable
// *
// * : Node *adjList[MAX] : Adjacency List를 구현하여 각 시작점 노드를 저장하기 위한 노드 포인터 배열입니다.
// * : vector<Node> graph : 입력으로 들어온 그래프 데이터를 저장하기 위한 노드 벡터 배열입니다.
// * : vector<Node> kruskal : 최종적으로 MST에 사용될 노드들을 저장하는 노드 벡터 배열입니다.
// * : int visited[MAX] = {false} : MST 구현에서 Cycle을 체크하기 위한 visited 정수형 배열입니다. 초기화는 false로 되어 있습니다.
// ********************************************************************************************************************

Node *adjList[MAX];
vector<Node> graph;
vector<Node> kruskal;
int visited[MAX] = {false};

// ********************************************************************************************************************
// * function : void inputToArray()
// * description : input.txt 파일에 텍스트 형태로 저장된 그래프 데이터를 읽어와, 노드로 변환, 벡터 배열에 저장합니다.
// *
// * Variable
// * : int numCases : 그래프 데이터의 개수를 받아와, 그 횟수만큼 반복하기 위한 정수형 변수입니다.
// * : char myself : 텍스트 데이터의 첫 번째 값인 한 점을 받아와 임시로 저장하기 위한 문자형 변수입니다.
// * : int edges : 텍스트 데이터의 두 번째 값인 엣지의 Cost를 받아와 임시로 저장하기 위한 정수형 변수입니다.
// * : char linked : 텍스트 데이터의 세 번째 값인 다른 한 점을 받아와 임시로 저장하기 위한 문자형 변수입니다.
// * : Node temp : 앞에서 임시로 저장된 세 값을 하나의 임시 노드로 만들어 저장하기 위한 노드 변수입니다.
// * : graph[MAX] : 위에서 만들어진 노드를 순차적으로 저장하기 위한 노드형 벡터 배열입니다.
// *
// * Algorithm description
// * : 1. 먼저 변수들을 정의하고, 파일을 읽어오는 ifstream을 inStream으로 정의내립니다.
// * : 2. input.txt 파일을 열어 읽어오기를 시작합니다. 열리지 않는 경우에 대한 에러 처리를 작성합니다.
// * : 3. 가장 먼저 numCases 에 데이터의 개수를 받아오며, 이 값만큼 for 반복문을 진행합니다. ( 4 ~ 6 반복 )
// * : 4. inStream을 이용해 myself, edges, linked 변수에 각각 값을 저장합니다.
// * : 5. 저장된 세 가지 값으로 Node temp를 Node 생성자를 이용해 정의내립니다.
// * : 6. 만들어진 temp 노드를 graph 벡터 배열에 push_back 메소드를 이용해 저장합니다.
// * : 위의 반복문을 모두 마친 다음에는, graph 벡터 배열에 그래프 데이터가 input.txt 순서대로 저장됩니다.
// *
// ********************************************************************************************************************

 void inputToArray(){
  ifstream inStream;
  int numCases;
  char myself;
  int edges;
  char linked;

  inStream.open("input.txt");
  if (inStream.fail()) {
   std::cerr << "Input file opening failed." << '\n';
   exit(1);
  }
```

```
  inStream >> numCases;
  for (size_t i = 0; i < numCases; i++) {
    inStream >> myself >> edges >> linked;
    Node temp = Node(myself, edges, linked);
    graph.push_back(temp);
  }
}

// ********************************************************************************************************************
// * function : void arrayToList()
// * description : graph 에 저장된 배열 형태의 그래프 데이터를 Adjacency List 형태의 데이터로 변환합니다.
// *
// * Variable
// * : Node *temp : adjList 배열의 헤드값을 초기화 하기 위한 노드 포인터 변수입니다.
// * : Node *move : adjList 배열의 내부를 이동하며, 새로운 노드가 들어갈 위치를 찾아내기 위한 노드 포인터 변수입니다.

// * : adjList[MAX] : Adjacency List 형태로 그래프 데이터를 저장하기 위한 노드 포인터 배열입니다. 각 노드의 시작점 헤드가 들어있습니다.
// * : graph[MAX] : 앞에서 저장된 벡터 형태의 그래프 데이터를 활용하여 리스트로 변환합니다.
// *
// * Algorithm description
// * : 1. 먼저 리스트의 헤드 처리가 필요합니다. 헤드에는 해당 점을 표시하는 가짜 노드를 하나씩 넣습니다.
// * : 2. 예를 들어, A 2 B 노드가 들어갈 위치는 A 점이며, A 정보는 A 0 A 의 형태로 리스트의 헤드에 저장됩니다.
// * : 3. 그 다음으로 move 노드 포인터를 이용하여 리스트를 순회하고, 노드가 들어갈 위치를 정합니다.
// * : 3-1. 먼저 노드의 시작점을 받아와, 이에 해당하는 헤드 노드를 move에 저장합니다.
// * : 3-2. move 를 한 칸씩 옮기며, move -> next가 NULL이 아닌 값이 나타나면 이동을 중단합니다.
// * : 4. 해당 위치인 move -> next 에 새로운 노드를 생성하고, 받아온 값을 넣습니다.
// ********************************************************************************************************************

void arrayToList(){
  for (size_t i = 0; i < MAX; i++) {
    Node *temp = new Node('A' + i, 0, 'A' + i);
    temp->next = NULL;
    adjList[i] = temp;
  }

  Node *move;
  for (size_t i = 0; i < graph.size(); i++) {

    move = adjList[graph[i].myself - 'A'];
    while (move->next != NULL) {
      move = move->next;
    }
    move->next = new Node(graph[i].myself, graph[i].edges, graph[i].linked);
    move->next->next = NULL;
  }
}

// ********************************************************************************************************************
// * function : void printList()
// * description : 위에서 생성된 리스트 형태의 그래프 데이터를 출력합니다.
// *
// * Variable
// * : Node *move : adjList를 순회하기 위한 노드 포인터 변수입니다.
// * : adjList[] : 리스트 형태의 그래프 데이터가 저장되어 있는 노드 포인터 배열입니다.
// *
// * Algorithm description
```

```
// * : 1. for 반복문을 이용해 adjList의 헤드를 한 번씩 순회하고, while 반복문을 이용해 헤드에 연결된 노드들을 순회합니다.
// * : 2. 우선 adjList의 크기인 MAX만큼 for 반복문을 작성합니다.
// * : 3. move를 adjList 의 헤드값으로 설정한 다음, 해당 노드의 myself를 출력하여 어느 시작점인지 알려줍니다.
// * : 4. while 반복문을 이용하여 move -> next가 NULL 값이 아닌 동안 move를 이동하며 노드의 정보를 출력합니다.
// * : 5. 이 때, 끝점인 move -> linked 를 출력하고, 해당 노드의 코스트인 move -> edges 를 함께 출력합니다.
// * : 6. 이 과정이 모두 끝나면 NULL을 출력하여 끝을 알려줍니다.
// ********************************************************************************************************************

void printList(){
  std::cout << "***** Print Adjacency List *****" << "\n\n";
  Node *move;
  for (size_t i = 0; i < MAX; i++) {
    move = adjList[i];
    std::cout << move->myself << " :  " ;
    while (move->next != NULL) {
      move = move->next;
      std::cout << move->linked << " (Cost : " << move->edges << ")" << " -> ";
    }
    std::cout << "NULL" << "\n";
  }
  std::cout << '\n';
}

// ********************************************************************************************************************
// * function : void sortData()
// * description : graph에 저장되어 있는 그래프 데이터를 코스트 별로 정렬하기 위한 함수입니다.
// *
// * Variable
// * : graph[] : 벡터 형태로 저장되어 있는 그래프 데이터 노드 배열입니다.
// *
// * Algorithm description
// * : 1. algorithm 라이브러리의 sort 알고리즘을 사용합니다. graph 의 begin부터 graph의 end까지 진행합니다.
// * : 2. 이 때, 오름차순으로 정렬하기 위해 less<Node>() 를 작성합니다.
// * : 3. 추가적으로, 노드의 비교는 edges 의 비교로 이루어져야 하므로, 이를 클래스 내부에 정의해주는 과정이 필요합니다.
// ********************************************************************************************************************

void sortData() {
  sort(graph.begin(), graph.end(),less<Node>());
}

// ********************************************************************************************************************
// * function : void printSortedData()
// * description : 앞에서 정렬한 데이터를 정렬 확인하기 위해 출력하는 함수입니다.
// *
// * Variable
// * : graph[] : 벡터 형태로 저장된 그래프 데이터 노드 배열입니다. 앞에서 sortData() 함수가 호출되었다면 정렬된 상태입니다.
// *
// * Algorithm description
// * : 1. 벡터의 크기만큼 for 반복문을 통해 반복하며 순차적으로 출력합니다.
// ********************************************************************************************************************

void printSortedData() {
  std::cout << "***** Print Sorted Data *****" << "\n\n";
  for (size_t i = 0; i < graph.size(); i++) {
    std::cout << graph[i].myself << "   " << graph[i].edges << "   " << graph[i].linked << '\n';
  }
```

```
  std::cout << '\n';
}

// ********************************************************************************************************************
// * function : bool allVisited()
// * description :kruskal MST함수를 진행하는 중, 모든 노드가 방문되었는지 확인하여 참 / 거짓을 반환하는 함수입니다.
// *
// * Variable
// * : visited[] : 배열의 각 위치에 시작점 노드를 대응시키고, 해당 노드가 방문되었는지의 여부를 저장하는 배열입니다.
// *
// * Algorithm description
// * : 1. for 반복문을 이용해 visited 배열을 확인하며, 하나라도 false가 반복되면 false를 반환합니다.
// * : 2. for 반복문을 모두 통과하였다면 모든 값이 true이므로 true를 반환합니다.
// ********************************************************************************************************************

bool allVisited(){
  for (size_t i = 0; i < MAX; i++) {
    if (visited[i] == false) {
      return false;
    }
  }
  return true;
}

// ********************************************************************************************************************
// * function : void kruskalMST()
// * description : Kruskal's Algorithm의 원리를 이용해 Minimum Spanning Tree를 찾아내는 알고리즘입니다.
// *
// * Variable
// * : char first / last : 트리를 하나의 노드에서 시작하여, 앞뒤로 노드를 붙여나갑니다. 이 때 앞, 뒤의 점을 char형태로 저장합니다.
// * : Node firstNode / lastNode : 앞 뒤의 노드로 붙여나가는 과정에서, 현재 앞과 뒤의 노드가 어떤 노드인지 저장해두는 노드 변수입니다.
// * : Node dumpNode : first / last 노드의 초기값을 만들어주기 위한 dumpNode 노드 변수입니다.
// * : kruskal[] : 최종적으로 MST에 사용될 노드들을 kruskal 벡터 배열에 저장하도록 합니다.
// * : graph[] : kruskal 에 사용될 그래프 데이터를 가져올 노드 벡터입니다.
// *
// * Algorithm description
// * : 1. 먼저 사용될 변수들을 정의해 줍니다. 가장 비용이 적은 첫 번째 노드는 kruskal 에 무조건 사용되므로 먼저 저장합니다.
// * : 1-1. 이렇게 사용된 첫 번째 노드의 myself 시작점과 linked 끝점은 first 와 last 의 초기값으로 저장됩니다.
// * : 1-2. first / last 노드는 비용이 100으로 만들어진 dumpNode 로 초기화 됩니다.
// * : 2. kruskal 알고리즘은 모든 노드가 방문되면 종료되야 하므로, 앞에서 정의내린 allVisited 함수를 사용해 while문을 통해 반복시킵니다.
// * : 3. while문을 반복하며 가장 먼저 firstNode와 lastNode를 초기화 해 줍니다. 또한 first 와 last 를 확인하여 방문 표시해 줍니다.
// * : 4. 가장 먼저 first 와 last에 매칭될 노드를 찾아냅니다.
// * : 4-1. graph 벡터 배열을 전체 탐색하며 graph 의 노드 데이터의 myself 혹은 linked가 first와 같다면 해당 노드를 채택하여 firstNode에 저장합니다.
// * : 4-2. 이 때 주의할 점은 첫째로 myself 가 같다면 linked, linked가 같다면 myself 점을 확인하여 해당 점을 방문하였는지 확인해야 합니다.
// * : 4-3. first와 같은지 먼저 확인이 끝난 다음, last와 같은 노드를 찾아 같은 원리로 lastNode에 저장합니다.
// * : 5. 이 다음, firstNode와 lastNode의 방문이 끝나서 dumpNode가 최소 노드로 저장되었는지 확인합니다.
// * :    만약 모두 dumpNode가 저장되었다면 더 이상 최소값이 들어갈 수 없는 것이므로 while문을 종료합니다.
// * : 6. 마지막으로 선택된 firstNode와 lastNode 둘 중 하나를 kruskal 벡터 배열에 넣습니다.
// * : 6-1. 먼저 firstNode와 lastNode의 edges 를 비교하여 더 적은 비용을 가진 노드를 kruskal에 채택합니다.
// * : 6-2. 채택된 노드를 kruskal 벡터에 넣은 다음, 해당 first 혹은 last Node의 myself 가 동일했던 것인지, linked가 동일했던 것인지 확인합니다.
```

```
// * : 6-3. 확인된 위치의 반대 점을 first 혹은 last에 넣으며 다시 반복문의 처음으로 돌아가도록 합니다.
// ********************************************************************************************************************

void kruskalMST() {
  char first, last;
  kruskal.push_back(graph[0]);

  first = kruskal[0].myself;
  last = kruskal[0].linked;
  Node dumpNode = Node(0, 100, 0);
  Node firstNode = dumpNode;
  Node lastNode = dumpNode;

  while (!allVisited()) {
    firstNode = dumpNode;
    lastNode = dumpNode;
    visited[first - 'A'] = true;
    visited[last - 'A'] = true;

    for (size_t i = 0; i < graph.size(); i++) {
      if ((graph[i].myself == first && !visited[graph[i].linked - 'A']) || (graph[i].linked == first && !
visited[graph[i].myself - 'A'])) {
        firstNode = graph[i];
        break;
      }
    }
    for (size_t i = 0; i < graph.size(); i++) {
      if ((graph[i].myself == last && !visited[graph[i].linked - 'A']) || (graph[i].linked == last && !
visited[graph[i].myself - 'A'])) {
        lastNode = graph[i];
        break;
      }
    }
    if (firstNode.edges == dumpNode.edges && lastNode.edges == dumpNode.edges) {
      break;
    }
    if (firstNode.edges < lastNode.edges) {
      kruskal.push_back(firstNode);
      if (firstNode.myself == first) {
        first = firstNode.linked;
      }else{
        first = firstNode.myself;
      }
    }else{
      kruskal.push_back(lastNode);
      if (lastNode.myself == last) {
        last = lastNode.linked;
      }else{
        last = lastNode.myself;
      }
    }
  }
}

// ********************************************************************************************************************
// * function : void sortKruskalMST()
// * description : kruskal에 저장되어 있는 MST 데이터를 코스트 별로 정렬하기 위한 함수입니다.
// *
// * Variable
// * : kruskal[] : 최종적으로 MST에 사용될 노드들이 저장되어 있는 배열 변수입니다.
```

```
// *
// * Algorithm description
// * : 1. algorithm 라이브러리의 sort 알고리즘을 사용합니다. kruskal 의 begin부터 graph의 end까지 진행합니다.
// * : 2. 이 때, 오름차순으로 정렬하기 위해 less<Node>() 를 작성합니다.
// * : 3. 추가적으로, 노드의 비교는 edges 의 비교로 이루어져야 하므로, 이를 클래스 내부에 정의해주는 과정이 필요합니다.
// ********************************************************************************************************************

void sortKruskalMST() {
  sort(kruskal.begin(), kruskal.end(),less<Node>());
}

// ********************************************************************************************************************
// * function : void printKruskalMST()
// * description : 최종적으로 정렬된 kruskal MST 데이터를 출력하는 함수입니다.
// *
// * Variable
// * : kruskal[] : 벡터 형태로 저장된 kruskal MST 데이터 노드 배열입니다.
// *
// * Algorithm description
// * : 1. 벡터의 크기만큼 for 반복문을 통해 반복하며 순차적으로 출력합니다.
// ********************************************************************************************************************

void printKruskalMST(){
  std::cout << "***** Print Kruskal MST *****" << "\n\n";
  for (size_t i = 0; i < kruskal.size(); i++) {
    std::cout << kruskal[i].myself << "  "<< kruskal[i].edges << "  " << kruskal[i].linked << '\n';
  }
}

// ********************************************************************************************************************
// * Name : 이성재
// * Student ID : 20132651
// * Program ID : Makefile
// *
// * Algorithm Description
// * : 지금까지 헤더 파일의 사용이 잘못되었다는 것을 알고, makefile을 이용해 컴파일 하여 kruskal 알고리즘을 수행합니다.
// * : CC = g++ : g++ 컴파일러를 사용하여 c++ 파일과 헤더파일을 컴파일합니다.
// * : TARGET = HW5 : HW5 라는 이름의 타겟 파일을 생성합니다.
// * : SOURCE = HW5_Kruskal_Main.o HW5_Kruskal.o : 과제로 작성된 파일들의 오브젝트 파일 형태입니다.
// * : all : $(TARGET) : make 명령어가 호출되었을 때, TARGET을 작동시킵니다.
// * : $(TARGET) : $(SOURCE) $(CC) -o $@ $^ : TARGET에 대한 매크로를 설정합니다. 먼저 SOURCE 파일을 TARGET의 의존 객체로 설정합니다.
// * :                                    : 이에 대해 CC 컴파일 형태로 TARGET을 생성하며, 이 때 모든 의존파일은 SOURCE가 됩니다.
// * : clean : clean 명령어를 이용해 모든 오브젝트 파일과 타겟 파일을 삭제하도록 합니다.
// ********************************************************************************************************************

CC = g++
SOURCE = HW5_Kruskal_Main.o HW5_Kruskal.o
TARGET = HW5

all: $(TARGET)

$(TARGET): $(SOURCE)
	$(CC) -o $@ $^

clean:
```

```
	rm -rf *.o $(TARGET)
```

```
// ********************************************************************************************************************
// * Name : 이성재
// * Student ID : 20132651
// * Program ID : input.txt
// *
// * Algorithm Description
// * : 텍스트 형태로 작성된 그래프 데이터입니다.
// * : 첫 줄에는 그래프의 노드 개수가 나타납니다.
// * : 시작점 비용 끝점 의 형태로 노드가 개수만큼 반복되어 작성됩니다.
// *
// ********************************************************************************************************************
```

```
10
A 6 B
B 7 C
A 1 C
A 5 D
C 5 D
B 3 E
C 8 E
C 4 F
D 2 F
E 9 F
```

```
Sungjaeui-MacBook-Pro:HW5 sungjae$ ls
HW5_Kruskal.cpp         HW5_Kruskal.hpp.gch     HW5_report.cpp          input.txt
HW5_Kruskal.hpp         HW5_Kruskal_Main.cpp    Makefile
Sungjaeui-MacBook-Pro:HW5 sungjae$ make
c++    -c -o HW5_Kruskal_Main.o HW5_Kruskal_Main.cpp
In file included from HW5_Kruskal_Main.cpp:1:
./HW5_Kruskal.hpp:17:14: warning: in-class initialization of non-static data member is a C++11 extension [-Wc++11-extensions]
  Node *next = NULL;
             ^
1 warning generated.
c++    -c -o HW5_Kruskal.o HW5_Kruskal.cpp
In file included from HW5_Kruskal.cpp:1:
./HW5_Kruskal.hpp:17:14: warning: in-class initialization of non-static data member is a C++11 extension [-Wc++11-extensions]
  Node *next = NULL;
             ^
1 warning generated.
g++ -o HW5 HW5_Kruskal_Main.o HW5_Kruskal.o
Sungjaeui-MacBook-Pro:HW5 sungjae$ ./HW5
```

```
***** Print Adjacency List *****

A :  B (Cost : 6) -> C (Cost : 1) -> D (Cost : 5) -> NULL
B :  C (Cost : 7) -> E (Cost : 3) -> NULL
C :  D (Cost : 5) -> E (Cost : 8) -> F (Cost : 4) -> NULL
D :  F (Cost : 2) -> NULL
E :  F (Cost : 9) -> NULL
F :  NULL

***** Print Sorted Data *****

A  1  C
D  2  F
B  3  E
C  4  F
A  5  D
C  5  D
A  6  B
B  7  C
C  8  E
E  9  F

***** Print Kruskal MST *****

A  1  C
D  2  F
B  3  E
C  4  F
A  6  B
Sungjaeui-MacBook-Pro:HW5 sungjae$ make clean
rm -rf *.o HW5
Sungjaeui-MacBook-Pro:HW5 sungjae$ ls
HW5_Kruskal.cpp         HW5_Kruskal.hpp.gch     HW5_report.cpp          input.txt
HW5_Kruskal.hpp         HW5_Kruskal_Main.cpp    Makefile
Sungjaeui-MacBook-Pro:HW5 sungjae$
```