과제5: DFS & Kruskal Algorithm

2022.1학기

※ 제출 관련 사항

(1) 제출기한: 2022. 6. 13(월), 오후 5시 (늦은 제출은 받지 않음)

(2) 제출파일

학번.c (또는 학번.cpp)

학번.pdf (보고서: 코드 실행 결과 캡쳐 + 자신의 말로 코드/알고리즘/시간복잡도 설명한 것. 두 가지 내용이 모두 있어야 한다.)

(3) 채점환경: 비주얼스튜디오2019 (기타 환경에서 코드 작성하였어도 비주얼스튜디오에서 컴파일 가능한지 반드시 확인할 것)

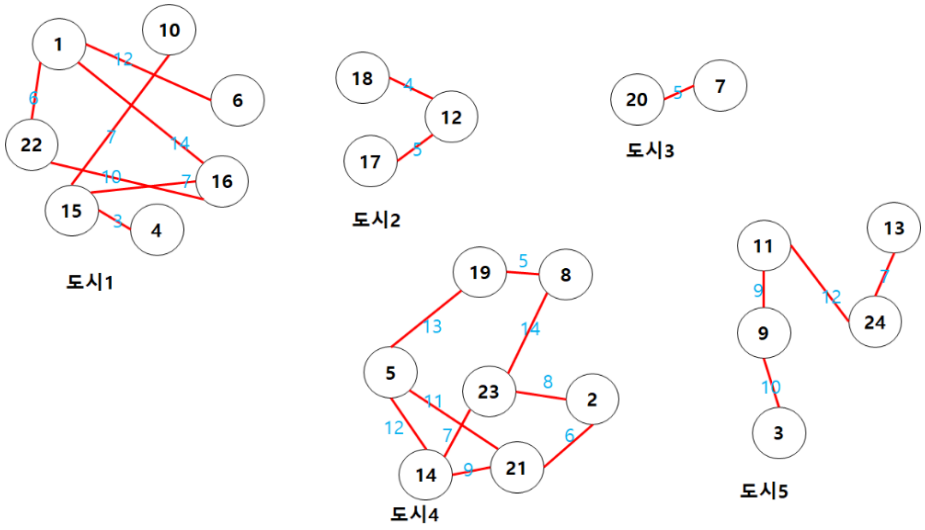
※과제 상세설명

아래 설명을 읽고 조건에 맞게 코드를 작성하고, 출력 결과를 내도록 한다.

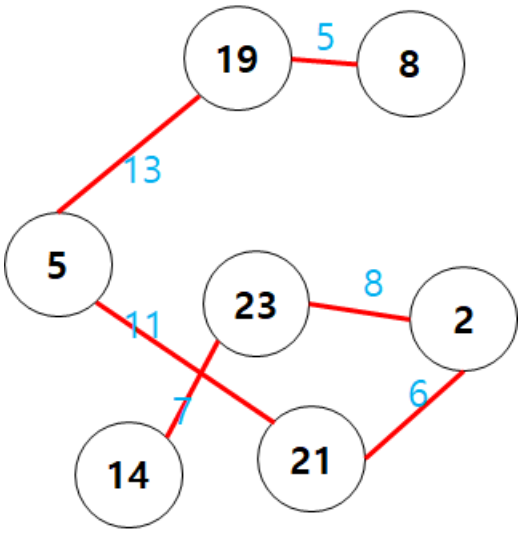
김영식군은 아파트 임장을 하러 가려고 한다. 총 N개의 아파트(node, vertex)들은 C개의 도시(disjoint graph)에 존재한다. 서로 다른 도시 사이에는 연결된 도로(edge)가 있을 수도 있고 없을 수도 있다. 모든 도로에는 이동시간(weight)이 표시되어 있다. 동일한 도시에 속한 아파트들은 도로로 연결되어 있지만, 서로 다른 도시에 속한 아파트들은 연결하는 도로가 존재하지 않는다.

김군의 시간은 한정적이기 때문에 최대한 많은 아파트가 속한 도시에 임장을 가야하며, 아파트가 가장 많은 도시가 2개 이상인 경우, 가장 비싼 아파트가 속한 도시에 가려고 한다. 아파트의 가격은 해당 아파트의 노드 번호이다. 도시 하나를 선택한 후에는 해당 도시에 속한 모든 아파트에 갈 수 있는 도로들을 선택하고자 한다. 다만 김군의 시간은 소중하기 때문에 선택한 도로의 이동시간의 총합은 최소가 되어야 한다.

DFS 알고리즘을 이용하여 도시의 개수를 찾고, Kruskal 알고리즘을 이용하여 이동시간의 총합이 최소가 되게 하는 도로를 찾아야한다. 아래 그림과 같은 그래프가 주어졌을 때, 아파트를 가장 많이 포함한 도시는 도시1과 도시4이다. (각각 7개) 한편 도시1의 최고가 아파트는 22이고, 도시4의 최고가 아파트는 23이다. 따라서 김군은 도시4를 선택한다.



도시 4 에 속한 모든 아파트를 갈 수 있는 도로의 이동시간 총합이 최소가 되게 선택하는 방법은 아래 그림과 같다.



도시4 MST

※데이터 입출력

입력 (A.txt)

```
24 23
1 6 12
1 16 14
1 22 6
2 21 6
2 23 8
3 9 10
4 15 3
5 14 12
5 19 13
5 21 11
7 20 5
8 19 5
8 23 14
9 11 9
10 15 7
11 24 12
12 17 5
12 18 4
13 24 7
14 21 9
14 23 7
15 16 7
16 22 10
```

-1번째 줄에는 전체 아파트(node, vertex) 개수 N과 전체 도로 개수(edge) E가 주어진다.

-2번째 줄부터 E+1번째 줄까지는 출발 아파트(start node), 도착 아파트(end node), 도로이동시간(weight) 순서로 edge 정보가 주어진다.

출력

```
5
7 9 23
50
```

-총 도시의 개수 (disjoint graph 개수)

-선택한 도시의 아파트 개수, 선택한 도시의 도로 개수, 선택한 도시의 최고가 아파트
(위의 예시에서 도시4에 해당함)

-선택한 도시의 최소 이동시간 총합 (위의 예시에서 도시4의 MST weight)

※조건

1. DFS(깊이우선탐색) 알고리즘 함수 작성 시 recursive가 아닌 iterative function으로 작성하고, 반드시 스택을 이용한다.
2. 스택 사용 시 전역변수를 사용하지 않고 주어진 구조체를 이용하며, init_stack, is_blank, push, pop 함수를 작성한다.
3. Graph 표현 방식은 adjacent matrix 또는 adjacent list 중 원하는 방법을 이용하고, 둘 중 하나만 사용하거나 모두 사용해도 무방하다.
4. MST(최소신장트리)를 구할 때에는 Kruskal 알고리즘을 사용하도록 하며, 해당 알고리즘 구현 시 사용할 정렬방식은 자유롭게 선택하도록 한다.
5. 출력 조건에 맞는 내용만 출력하도록 한다. 불필요한 내용이 포함되어 출력된 경우 감점될 수 있다.
6. 1~5 조건을 만족하는 선에서 주어진 template 코드(과제5.c)의 모든 내용은 자유롭게 수정이 가능하며 (main함수 포함), 단순 참고용이다.

코드 템플릿 (과제5.c)

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define MAX_APT_ON_ONE_CITY 40
#define MAX_CITY 10
#define MAX_STACK_SIZE 1000

typedef struct adjListNode* adjListPointer;
typedef struct adjListNode {
        int dest;
        int weight;
        adjListPointer next;
} adjListNode;

typedef struct adjList {
        int Nnode;
        adjListPointer* adj_head;
        adjListPointer* adj_tail;
} adjList;

typedef struct cityNode {
        int largest, Napt, Nedge;
        int apt[MAX_APT_ON_ONE_CITY];
} cityNode;

typedef struct stackList* stackPointer;
typedef struct stackList {
        int top;
        int* arr;
} stackList;

typedef struct edgeNode {
        int start, end, weight;
} edgeNode;

int** init_adj_matrix(int num_node);
adjList* init_adj_list(int num_node);
adjListPointer init_adj_list_node(int dest, int weight);
void insert_adj_list(adjList* graph, adjListPointer insert_node, int start_node);
void add_edges(FILE* fp, int** adj_matrix, adjList* adj_list, int num_edges);

int dfs(int** graph, int num_vertices, cityNode* city);

stackPointer init_stack(int size);
int is_blank(stackPointer stack);
void push(stackPointer stack, int element);
int pop(stackPointer stack);

cityNode find_most_apt_city(cityNode* city_list, int n);
edgeNode* find_and_sort_edge(adjList* adj_list, cityNode city);

int get_parent(int* parent, int x);
void union_parent(int* parent, int x, int y);
```

```
int find(int* parent, int x, int y);
int kruskal(cityNode city, edgeNode* edges, int total_num_node);

// bonus functions
void print_adj_matrix(int** adj_matrix, int num_node);
void check_adj_list(adjList* adj_list);
void check_city_and_apt(int n, cityNode* city_list);
void print_edges(cityNode city, edgeNode* edges);

/*=============================================================================================
==========================================*/
void main() {
        FILE* fp = fopen("A.txt", "r");

        int N, E;
        cityNode cities[MAX_CITY];

        fscanf(fp, "%d %d", &N, &E);

        int** graph_matrix = init_adj_matrix(N); // adj matrix 초기화
        adjList* graph_list = init_adj_list(N); // adj list 초기화
        add_edges(fp, graph_matrix, graph_list, E); // edge 추가 (adj matrix, adj list 각각)
        //print_adj_matrix(graph_matrix, N); // adj matrix 제대로 나오는지 출력
        //check_adj_list(graph_list); // adj list 제대로 나오는지 출력

        fclose(fp);

        int num_city = dfs(graph_matrix, N, cities);
        //check_city_and_apt(num_city, cities); // dfs 후에 city와 거기에 포함된 apt가 잘
나오는지 확인

        cityNode has_most_apt_city = find_most_apt_city(cities, num_city);
        edgeNode* edges = find_and_sort_edge(graph_list, has_most_apt_city);
        //print_edges(has_most_apt_city, edges); // 제대로 정렬되었는지 출력
        int mst_weight = kruskal(has_most_apt_city, edges, N);

        printf("%d₩n", num_city);
        printf("%d %d %d₩n", has_most_apt_city.Napt, has_most_apt_city.Nedge,
has_most_apt_city.largest + 1);
        printf("%d₩n", mst_weight);
}
/*=============================================================================================
==========================================*/

// bonus functions
void print_adj_matrix(int** adj_matrix, int num_node) {
        printf("   ");
        for (int i = 0; i < num_node; i++) printf("%2d ", i + 1);
        printf("₩n");
        for (int i = 0; i < num_node; i++) {
                printf("%2d ", i + 1);
                for (int j = 0; j < num_node; j++)
                        printf("%2d ", adj_matrix[i][j]);
                printf("₩n");
        }
```

```
}

void check_adj_list(adjList* adj_list) {
        for (int i = 0; i < adj_list->Nnode; i++) {
                adjListPointer cur = adj_list->adj_head[i];
                printf("node %d₩n", i + 1);
                for (cur = cur->next; cur != NULL; cur = cur->next) {
                        printf("%d %d₩n", cur->dest + 1, cur->weight);
                }
                printf("₩n");
                free(cur);
        }
}

void check_city_and_apt(int n, cityNode* city_list) {
        printf("number of cities: %d₩n₩n", n);
        for (int c = 0; c < n; c++) {
                printf("city %d₩nlargest apt: %d, number of apts: %d, number of
edges: %d₩napt list: ",
                        c + 1, city_list[c].largest + 1, city_list[c].Napt,
city_list[c].Nedge);
                for (int a = 0; a < city_list[c].Napt; a++) {
                        printf("%d ", city_list[c].apt[a] + 1);
                }
                printf("₩n₩n");
        }
}

void print_edges(cityNode city, edgeNode* edges) {
        for (int e = 0; e < city.Nedge; e++) printf("start: %d, end: %d, weight: %d₩n",
                edges[e].start + 1, edges[e].end + 1, edges[e].weight);
}
```