HW4 Make MST with disjoint data structure in greedy Algorithm

2016\*\*\*\*

<실험환경>

OS: Windows10 Home

CPU: Intel(R) Core(TM) i3-5005U CPU @2.00GHz 2.00GHz

RAM: 4.00GB

Compiler: Visual Studio 2015 Community Release Mode (X64)

<실험 요약>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 파일 이름 | 작동여부 | MST weight | 수행 시간 (초) |  |
| HW4\_email-Eu-core.txt | YES | 3110161 |  | O(elogv) |
| HW4\_com-dblp.ungraph.txt | YES | 2747895457 |  | O(elogv) |
| HW4\_com-amazon.ungraph.txt | YES | 2729670156 |  | O(elogv) |
| HW4\_com-youtube.ungraph.txt | YES | 14578691475 |  | O(elogv) |
| HW4\_wiki-topcats.txt | YES | 5351181035 |  | O(elogv) |
| HW4\_com-lj.ungraph.txt | YES | 28308045762 |  | O(elogv) |

\*\* 주의사항

과제 요구사항에 따라 main.c 파일이 있는 경로에 ‘commands.txt’와 commands.txt 파일의 첫 번째 줄에 적힌 것과 동일한 파일명의 데이터 파일이 존재해야 한다. 또한, commands.txt 의 두 번째 줄에는 프로그램의 결과가 입력될 파일명을 적는다.

예) 첫 번째 줄에는 input.txt, 두 번째 줄에는 output.txt

\* 보고서와 함께 압축하여 제출한 프로젝트에는 commands.txt 파일을 비롯한 데이터 파일이 없습니다.

1. 그래프 표현 방법

Graph라는 구조체에 정점의 개수를 의미하는 V와 간선의 개수를 의미하는 E를 정수형태로 가지고 간선을 의미하는 edge를 포인터 형태로 갖는다. 간선을 의미하는 edge는 Edge라는 구조체로 표현하였고, src, dest, weight, root 총 네 개의 정수 데이터를 갖는다. Src와 dest는 해당 간선이 연결되어 있는 정점을 의미하며, weight는 해당 간선의 가중치를 의미한다. Root는 해당 간선에서 연결 가능한 최고 정점을 찾는다. 임의의 두 개의 간선에 대해 root 값이 서로 다르면, 두 간선은 서로 다른 connected component에 속한다는 것을 뜻한다. 그래프가 트리라고 가정한다면 최고 노드의 정점을 찾는다. Graph 구조체 안에 있는 edge를 통해 Edge 구조체로 접근이 가능하다.

<코드>

struct Edge {

int src, dest, root;

long long int weight;

};

struct Graph {

int V, E;

struct Edge \*edge;

};

1. 각 함수 설명과 시간복잡도
2. int find(struct subset subsets[], int i)

주어진 배열에서 i번째 항의 루트 노드를 찾는다. 자기 자신과 자신의 부모 노드의 값이 서로 같을 때까지 본 함수를 재귀적으로 호출한다. Minheap에서 최고 노드에 최소값을 위치하는 함수와 유사하다.

<코드>

int find(struct subset subsets[], int i) {

if (subsets[i].parent != i)

subsets[i].parent = find(subsets, subsets[i].parent);

return subsets[i].parent;

}

* 시간복잡도: O(logi)

1. void Union(struct subset subsets[], int x, int y)

주어진 배열 내의 두 아이템의 루트노드를 같도록 하여 두 아이템이 트리를 이루도록 한다. 이때, 배열에서 선택되는 두 아이템은 그래프 상에서 두 개의 정점에 해당하며, 두 정점이 하나의 간선으로 연결되어 있을 때 두 아이템을 하나의 트리로 만든다. 함수 내에서 각 정점에 대한 find 함수를 단독적으로 수행하기 때문에 시간복잡도는 O(logi)이다.

<코드>

void Union(struct subset subsets[], int x, int y) {

int xroot = find(subsets, x);

int yroot = find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

else if (subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

1. int comparing(const void\* a, const void\* b)

c 라이브러리에 내장되어 있는 qsort 함수를 사용하기 위한 정렬 기준을 제시한다. 각 간선의 weight를 중심으로 오름차순 정렬한다.

<코드>

int comparing(const void\* a, const void\* b) {

struct Edge \*a1 = (struct Edge\*)a;

struct Edge \*b1 = (struct Edge\*)b;

return a1->weight - b1->weight;

}

* 시간복잡도: O(1)

1. int comparing2(const void\* a, const void\* b)

각 edge의 root를 기준으로 오름차순하기 위한 qsort의 기준을 만든다.

<코드>

int comparing2(const void\* a, const void\* b) {

struct Edge \*a1 = (struct Edge\*)a;

struct Edge \*b1 = (struct Edge\*)b;

return a1->root - b1->root;

}

* 시간복잡도: O(1)

1. void MST(struct Graph\* graph)

주어진 그래프에 대해 크루스칼 알고리즘을 사용한 MST를 만든다. 우선 주어진 그래프의 간선 중 weight를 기준으로 quick sort를 수행한다. (O(eloge)) 이후, 각 정점의 연결 관계를 담는 배열을 만들고, 각 정점의 부모노드와 레벨을 셋팅한다. (O(v)) 그리고 그래프의 간선을 돌며 방금 초기화 한 정점에 간선 정보를 업데이트한다. 어떤 정점이 어떤 정점과 dusrufehldjdlTsmswlfmf disjoint 자료형으로 구성하는것. 전체 간선 개수 E를 돌며 각 정점에 대한 union 함수를 수행하기 때문에 시간복잡도는 O(elogv). 이렇게 구성한 간선 관계에서 전체 간선만큼 돌며 connected component 관계를 조사한다. (O(e)) 이렇게 조사된 각 component에서 v-1개 만큼의 간선을 선택해 mst를 구성한다. O(v)

<코드>

void MST(struct Graph\* graph) {

int V = graph->V;

int E = graph->E;

struct Edge \*result;

result = (struct Edge\*)malloc((V - 1) \* sizeof(struct Edge));

int e = 0, i = 0, n = 0;

qsort(graph->edge, graph->E, sizeof(graph->edge[0]), comparing);

struct subset \*subsets = (struct subset\*)malloc(V \* sizeof(struct subset));

for (int v = 0; v < V; v++) {

subsets[v].parent = v;

subsets[v].rank = 0;

}

for (e = 0; e < E; e++) {

struct Edge next\_edge = graph->edge[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

if (x != y) {

Union(subsets, x, y);

result[n++] = next\_edge;

}

}

for (int i = 0; i < n; i++) result[i].root = find(subsets, result[i].src);

qsort(result, n, sizeof(result[0]), comparing2);

int cnum = 1, vnum = 0, isolated = 0;

long long int weight = 0;

for (int i = 0; i < n - 1; i++) {

if (result[i].root != result[i + 1].root) cnum++;

}

for (int i = 0; i < V; i++) {

if (subsets[i].parent == i && subsets[i].rank == 0) isolated++;

}

FILE \*fp = fopen(output\_file, "w");

cnum += isolated;

fprintf(fp, "%d\n", cnum);

for (i = 0; i < n; i++) {

if (i > 0 && result[i].root != result[i - 1].root) {

fprintf(fp, "%d %lld\n", vnum + 1, weight);

vnum = 0;

weight = 0;

}

vnum++;

weight += result[i].weight;

}

fprintf(fp, "%d %lld\n", vnum + 1, weight);

for (i = 0; i < isolated; i++) fprintf(fp, "1 0\n");

fclose(fp);

}

* 시간복잡도: O(elogv)

1. 결론

메인에서는 mst 함수만을 호출하고, mst 함수의 전체 시간복잡도는 O(elogv)이므로, 전체 프로그램의 시간복잡도는 O(elogv)가 된다.