알고리즘 설계와 분석

HW 3: Minimum Spanning Tree 알고리즘의 구현

20211533 남정연

1. 과제 수행 환경

OS : Windows 10 Education

CPU : Intel(R) Core(TM) i5-9400 CPU @ 2.90GHz 2.90 GHz

RAM : 16.0GB

Compiler : Visual Studio 2022 Release Mode

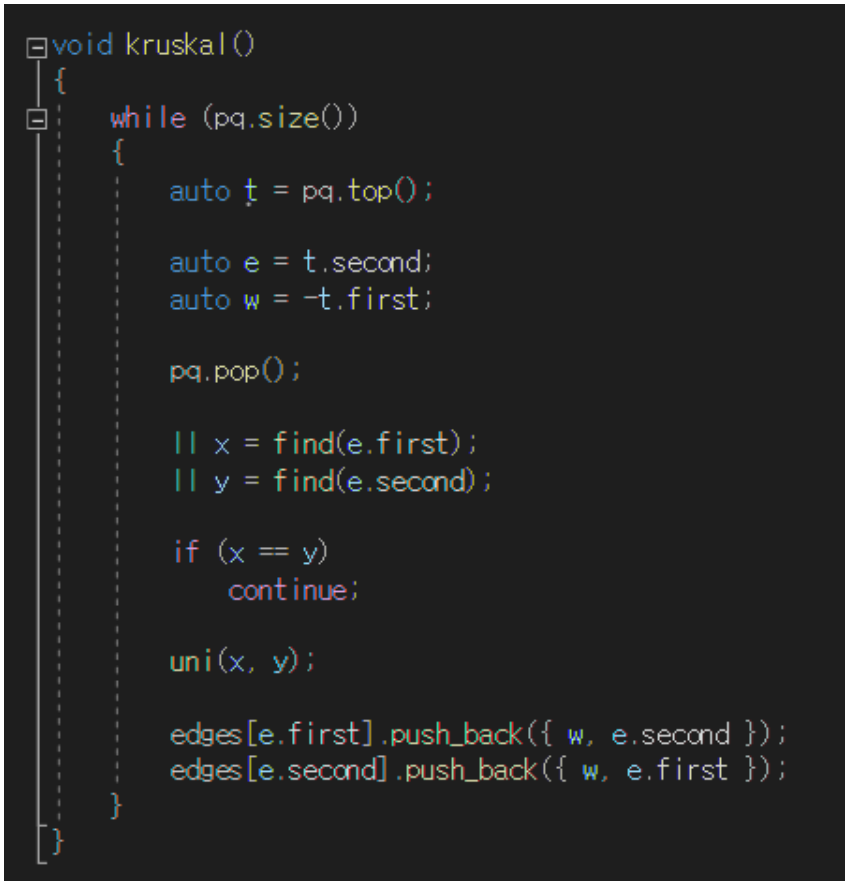
2.

가. Kruskal 알고리즘

Kruskal 알고리즘은 union-find 알고리즘과 priority_queue를 사용하여 구현하였다.

```
void uni(ll x, ll y)
{
    if (ra[x] < ra[y])
        swap(x, y);

    if (ra[x] == ra[y])
        ra[x]++;

    pa[y] = x;
}

ll find(ll x)
{
    if (pa[x] == x)
    {
        return x;
    }

    return pa[x] = find(pa[x]);
}
```

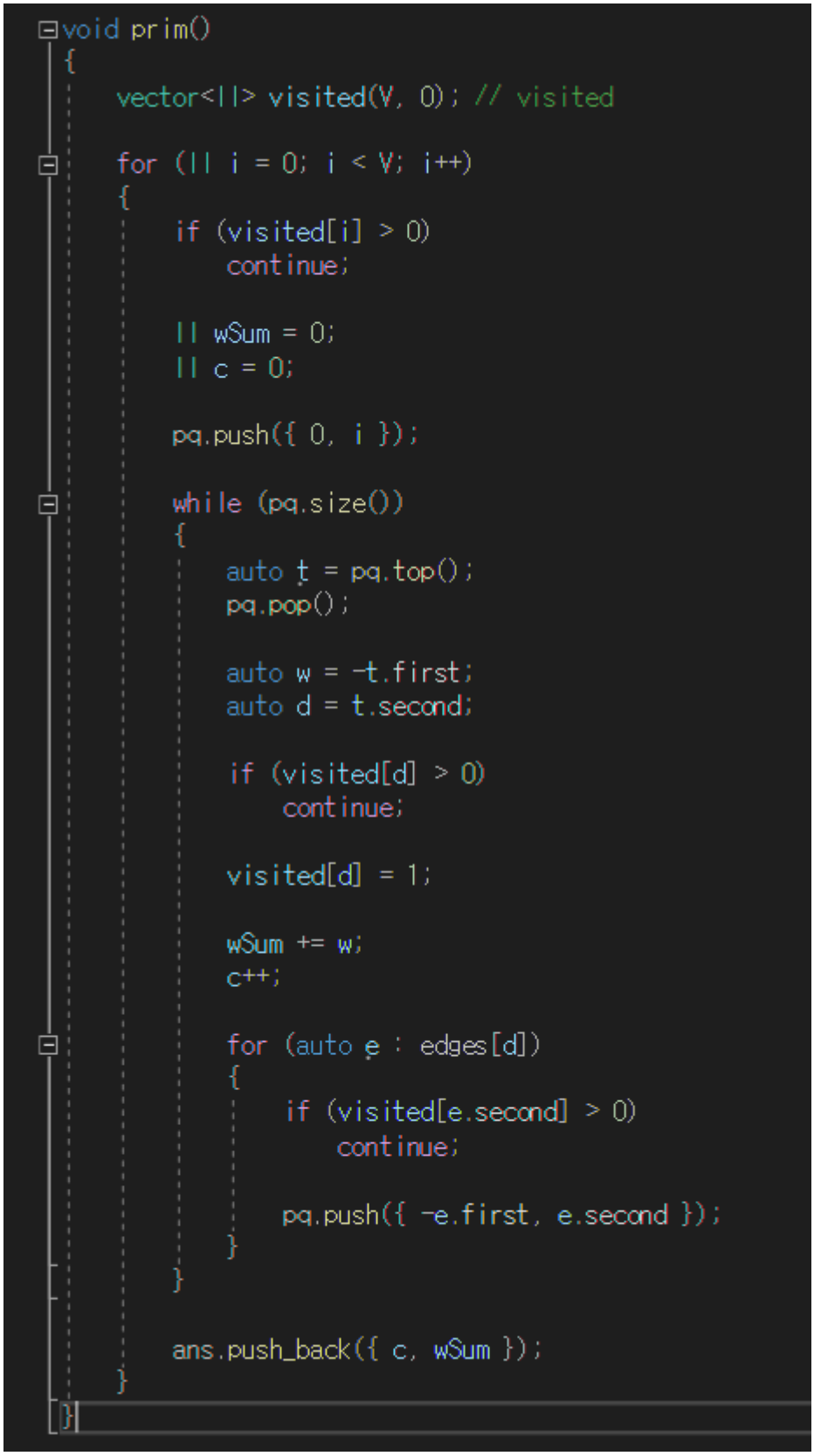
위는 disjoint_set을 구현하기 위한 union-find 알고리즘의 주요 원시 코드이다. union에 해당하는 uni 함수와 find에 해당하는 find함수, 각 set의 부모 set을 저장하는 pa 배열과 각 set의 rank 정보를 담은 ra 배열이 있다. uni 함수는 최적화 기법을 적용하여 rank가 더 큰 set에 작은 set을 합치도록 구현하였다. find 함수 또한 최적화 기법을 적용하여 set의 부모를 찾아가는 과정에서 pa 배열에 그 정보를 저장함으로써 트리의 높이를 낮혀주었다.



```
void kruskal()
{
    while (pq.size())
    {
        auto t = pq.top();

        auto e = t.second;
        auto w = -t.first;

        pq.pop();

        ll x = find(e.first);
        ll y = find(e.second);

        if (x == y)
            continue;

        uni(x, y);

        edges[e.first].push_back({ w, e.second });
        edges[e.second].push_back({ w, e.first });
    }
}
```

위는 Kruskal 알고리즘의 주요 구현코드이다. priority_queue pq에는 모든 간선들이 가중치 오름차순으로 저장되어 있다(음수 적용 기법 사용). pq에서 간선을 pop할 때 마다 동일한 disjoint_set에 있는지 확인하고 그렇지 않으면 두 set을 union하고 해당 간선을 선택했다(edges가 선택된 간선들이 담기는 adjacent list이다.). 이 과정의 시간 복잡도는 priority_queue가 heap에 기반을 두기 때문에 E개의 간선들을 모두 삽입하고 제거해야 하므로 O(E log E)이다.

나. Prim



```
void prim()
{
    vector<ll> visited(V, 0); // visited

    for (ll i = 0; i < V; i++)
    {
        if (visited[i] > 0)
            continue;

        ll wSum = 0;
        ll c = 0;

        pq.push({ 0, i });

        while (pq.size())
        {
            auto t = pq.top();
            pq.pop();

            auto w = -t.first;
            auto d = t.second;

            if (visited[d] > 0)
                continue;

            visited[d] = 1;

            wSum += w;
            c++;

            for (auto e : edges[d])
            {
                if (visited[e.second] > 0)
                    continue;

                pq.push({ -e.first, e.second });
            }
        }

        ans.push_back({ c, wSum });
    }
}
```

위는 Prim 알고리즘의 원시 코드이다. 주어지는 입력이 하나의 connected component로 이루어짐이 보장되지 않으므로, 방문하지 않은 모든 정점에 대하여 priority_queue를 사용하여 해당 정점으로부터 갈 수 있는 정점을 가중치가 낮은 간선을 선택하여 트리를 확장시켜 나갔다

(priority_queue에는 가중치 오름차순으로 정점이 담긴다). 해당 방법의 시간 복잡도는 priority_queue가 heap에 기반을 둠으로 간선 탐색에 E, 정점 삽입에 log V의 시간이 소요되어 O(E log V)의 시간이 소요된다.

3. 수행 결과

가. Kruskal

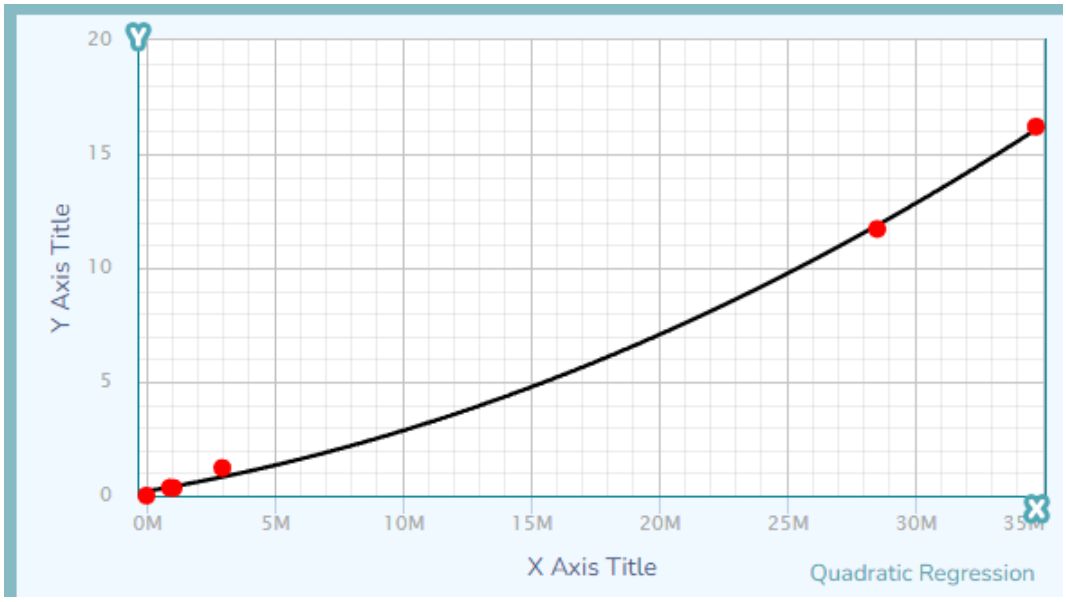
파일 이름	작동 여부	MST Weight	수행 시간 (초)	K scanned
HW3_com-amazon.ungraph	Yes	2729670156	0.356	334862
HW3_com-dblp.ungraph	Yes	2747895457	0.344	317079
HW3_com-lj.ungraph	Yes	28308045762	16.201	3997961
HW3_com-youtube.ungraph	Yes	14578691475	1.22	1134889
HW3_email-Eu-core	Yes	3110161	0.003	985
HW3_wiki-topcats	Yes	5351181035	11.71	1791488

나. Prim

파일 이름	작동 여부	MST Weight	수행 시간 (초)
HW3_com-amazon.ungraph	Yes	2729670156	0.209
HW3_com-dblp.ungraph	Yes	2747895457	0.238
HW3_com-lj.ungraph	Yes	28308045762	14.053
HW3_com-youtube.ungraph	Yes	14578691475	0.907
HW3_email-Eu-core	Yes	3110161	0.003
HW3_wiki-topcats	Yes	5351181035	10.808

4. 결과 분석

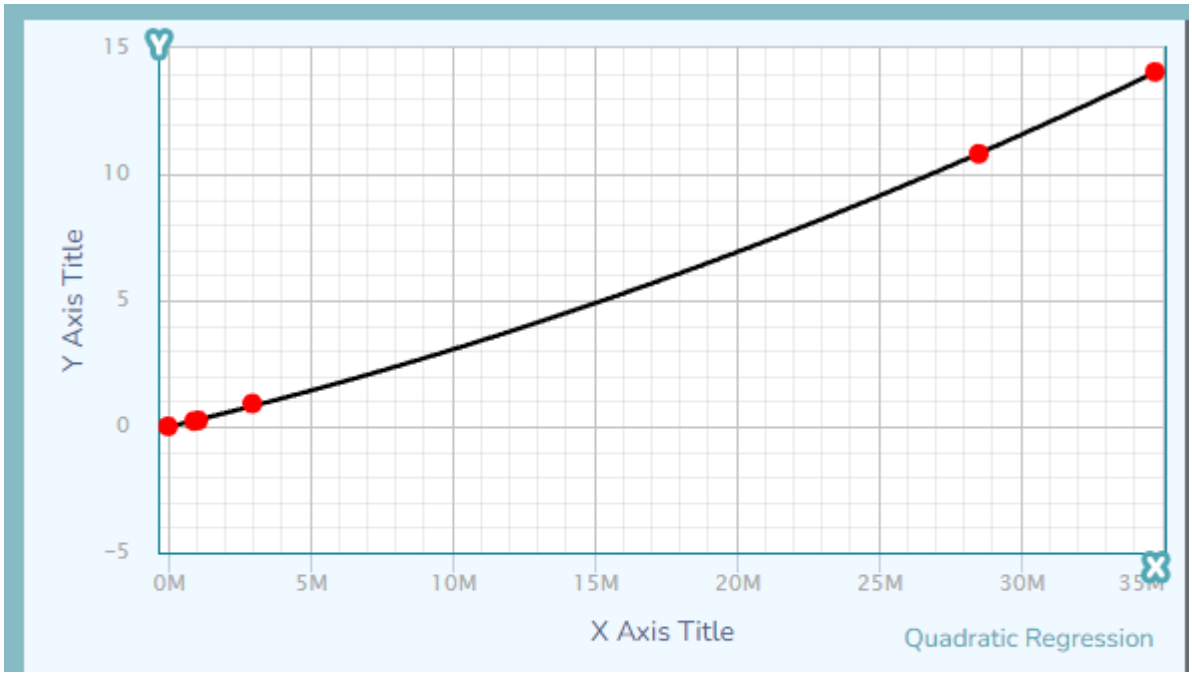
가. Kruskal



x축을 간선의 개수로 하여 Graph Fitting(Quadratic Regression)을 해본 결과 로그 함수의 그래프와

유사하게 나온 것을 알 수 있었으나 N log N 꼴의 그래프에 대해 Fitting을 할 수단이 없어 경향성의 유사만 체크할 수 있었다. 위 추세선을 통하여 log 그래프와 유사한 경향을 가짐으로 분석한 시간 복잡도가 어느 정도 정당함은 확인할 수 있었다.

나. Prim



Prim 방법 또한 x축을 간선의 개수로 하여 Graph Fitting(Quadratic Regression)을 해본 결과 로그함수의 그래프와 유사하게 나온 것을 알 수 있었으나 N log N 꼴의 그래프에 대해 Fitting을 할 수단이 없어 경향 성의 유사만 체크할 수 있었다. 위 추세선을 통하여 log 그래프와 유사한 경향을 가짐으로 분석한 시간 복잡도가 어느 정도 정당함은 확인할 수 있었다.