SCV/2021

학교 탐방하기

- Minimum Spanning Tree -

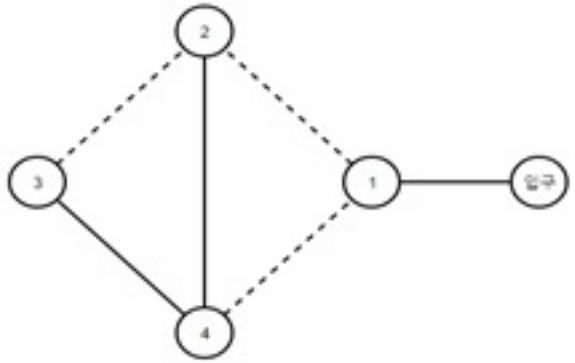
안태진(taejin7824@gmail.com)
GitHub(github.com/taejin1221)
상명대학교 소프트웨어학과
201821002

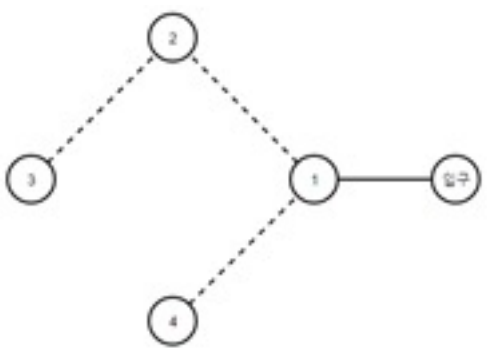
목차

Problem

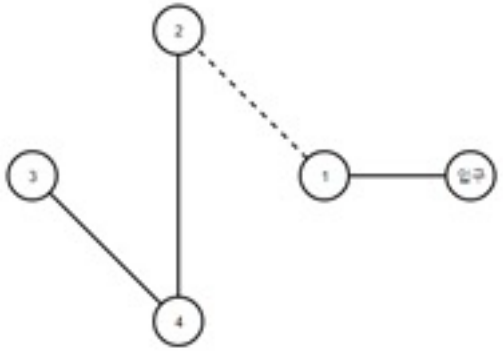
- 문제 설명
  - 점선은 오르막길, 실선은 내리막길
  - 모든 노드를 탐색하도록 간선을 연결
  - 선택된 오르막 간선의 제곱이 최종 cost

  - 최대 cost와 최소 cost의 차이를 구하여라
    - 오른쪽 그림은 1: 9, 2: 1
    - 따라서 차이는 8





1: 최대 cost



2: 최소 cost

목차

- Problem
- Approach
- Code

Approach

- Approach
  - 모든 노드를 최소한의 간선으로 연결 => MST
  - 최선의 선택
    - 내리막길을 최대한 선택하고 남는 간선은 오르막길로 선택
  - 최악의 선택
    - 오르막길을 최대한 선택하고 남는 간선은 내리막길로 선택
  - Prim? Kruskal?
    - 간선을 기준으로 내리막길/오르막길을 먼저 선택하는 방식이므로 Kruskal 선택

목차

- Problem
- Approach
- Code

Code

- 큰 그림
  - long long과 pair<int, int>는 이름이 길기 때문에 각각, ll, pii로 치환

```
struct Edge {
    int startNode, endNode;

    Edge(int s, int e): startNode(s), endNode(e) { }
};
```

  - Edge 구조체
    - 시작 정점과 도착 정점을 담는 구조체로써 간선 표현

  - UnionFindTree class
    - Kruskal은 Cycle체크가 필요함 따라서 효율이 좋은 Union find tree로 cycle detection

```
typedef long long ll;
typedef pair<int, int> pii;

class UnionFindTree;
struct Edge;
```

Code

- UnionFindTree class (1/2)
  - 멤버 변수
    - parent
      - 각 노드의 부모 정보를 지니는 배열
  - 생성자
    - parent를 size만큼 할당
    - 초기에는 부모가 자기 자신이도록 초기화
  - 소멸자
    - 동적 할당한 배열 제거

```
class UnionFindTree {
private:
    int* parent;

public:
    UnionFindTree(int size) {
        parent = new int[size];
        for (int i = 0; i < size; i++)
            parent[i] = i;
    }

    ~UnionFindTree() {
        delete[] parent;
    }
```

Code

- UnionFindTree class (2/2)
  - find method
    - target의 부모를 찾아 삼만리...
    - 찾으면서 부모 갱신 -> 최적화를 위해
  - unionTwoNode method
    - aRoot와 bRoot를 union하기
    - 그냥 더 작은 숫자가 부모가 되도록 함

```
    int find(int target) {
        return (parent[target] == target) ? target : parent[target] = find(parent[target]);
    }

    void unionTwoNode(int aRoot, int bRoot) {
        if (aRoot < bRoot)
            parent[bRoot] = aRoot;
        else
            parent[aRoot] = bRoot;
    }
};
```

Code

- getMST 함수 (1/2)
  - 이 문제를 풀기 위해 필요한 함수
    - 먼저 선택해야하는 간선들 먼저 선택한 후 남은 간선은 나중에 선택해야하는 간선들에서 선택
    - 따라서 firstEdges에 먼저 선택하는 간선, secondEdges에서는 나중에 선택해야하는 간선
      - 최선의 선택: firstEdges는 내리막 간선
      - 최악의 선택: firstEdges는 오르막 간선
  - 선택했을 때 Cycle이 안생기는 Edge 선택
  - 반환해야하는 값
    - firstEdges에서 선택된 간선의 수
    - secondEdges에서 선택된 간선의 수

```
pii getMST(int n, vector<Edge>& firstEdges, vector<Edge>& secondEdges) {
    UnionFindTree tree(n + 1);

    pii result = { 0, 0 };
    for (Edge& edge : firstEdges) {
        int aRoot = tree.find(edge.startNode), bRoot = tree.find(edge.endNode);
        if (aRoot != bRoot) {
            tree.unionTwoNode(aRoot, bRoot);
            result.first++;
        }
    }

    for (Edge& edge : secondEdges) {
        int aRoot = tree.find(edge.startNode), bRoot = tree.find(edge.endNode);
        if (aRoot != bRoot) {
            tree.unionTwoNode(aRoot, bRoot);
            result.second++;
        }
    }

    return result;
}
```

Code

- getMST 함수 (2/2)
  - 만약 간선이 연결하고 있는 두 정점의 부모가 다르면
    - cycle이 안생기는 것
    - 따라서 연결하고 union 해줌
    - 간선의 개수++

```
    for (Edge& edge : firstEdges) {
        int aRoot = tree.find(edge.startNode), bRoot = tree.find(edge.endNode);
        if (aRoot != bRoot) {
            tree.unionTwoNode(aRoot, bRoot);
            result.first++;
        }
    }
```

Code

- main function
  - 오르막 간선은 edges[0]에 내리막 간선은 edges[1]에 저장

```
vector<Edge> edges[2];
for (int i = 0; i < m + 1; i++) {
    int a, b, c;
    cin >> a >> b >> c;

    edges[c].push_back(Edge(a, b));
}
```

  - 최악의 선택, 최선의 선택으로 MST를 구하기

```
pii maxResult = getMST(n, edges[0], edges[1]);
pii minResult = getMST(n, edges[1], edges[0]);
```

  - 최악일 때와 최선일 때 차이 구하기

```
cout << (ll)maxResult.first * maxResult.first - (ll)minResult.second * minResult.second << '\n';
```

감사합니다!