

# MST: Kruskal & Prim

1115 이정우

## 목차

1	알고리즘 개요	2
1.1	Kruskal 알고리즘 . . . . .	2
1.2	Prim 알고리즘 . . . . .	2
2	자료구조 상세	2
2.1	변수와 ADT . . . . .	2
2.2	구현 상세 및 핵심 코드 . . . . .	3
2.3	복잡도 분석 . . . . .	5
2.4	전체 코드 . . . . .	5
2.5	Prim . . . . .	7
3	활용 방안	8
4	인상깊었던 점	8
5	참고문헌	8

# 1 알고리즘 개요

## 1.1 Kruskal 알고리즘

Kruskal 알고리즘이란, 가중치가 있는 무방향 그래프에서 최소 신장 트리(MST)를 찾는 대표적인 방법으로, 모든 간선을 가중치 오름차순으로 정렬한 뒤, 가장 비용이 낮은 간선부터 하나씩 선택하면서 사이클이 생기지 않도록 Union-Find로 관리한다. 이 과정을 거치면 모든 정점을 최소 비용으로 연결할수 있으며, 불필요한 간선이 제외되어 최적의 연결 구조를 보장한다.

## 1.2 Prim 알고리즘

Prim 알고리즘은 그래프에서 최소 신장 트리(MST)를 찾는 방법 중 하나로, 시작 정점을 기준으로 매번 현재 트리에 연결된 간선들 중 가장 가중치가 작은 간선을 선택하여 새로운 정점을 추가하는 방식으로 진행된다. 즉, 트리에 속하지 않은 정점 중에서 이미 선택된 정점과 연결된 가장 작은 간선만 골라 확장해 나가며, 모든 정점을 포함할 때까지 반복한다.

# 2 자료구조 상세

## 2.1 변수와 ADT

### 2.1.1 Kruskal 변수

- `parent: any[size]`
  - 각 원소가 가진 대푯값을 표시하는 배열이다.
- `pq: tuple({int, int, int})`
  - 두 정점과 그 사이의 거리를 저장하는 우선순위 큐 배열이다.

### 2.1.2 Kruskal ADT

- `my_union() -> int`
  - 두 노드를 병합하는 함수이다.
- `find() -> int`
  - 노드의 대푯값을 반환한다.

### 2.1.3 Prim 변수

- `adj: vector<pair<int,int>>`
  - 두 노드 및 거리를 저장하는 변수이다.
- `pq: tuple({int, int, int})`
  - 두 정점과 그 사이의 거리를 저장하는 우선순위 큐 배열이다.

### 2.1.4 Exception

- 그래프가 연결되었지 않다면 MST를 구할수없다.

## 2.2 구현 상세 및 핵심 코드

### 2.2.1 my\_union()

a와 b를 매개변수로 받은 뒤 두 노드의 대푯값을 찾는 연산 및 사이클 검사를 하는 함수이다. 두 노드의 뿌리가 같아 사이클이 된다면 False, 아니면 True를 반환한다.

```
int my_union(int a, int b){
    int RootX = find(a);
    int RootY = find(b);
    if(RootX == RootY){
        return 0;
    }
    else if(RootX<RootY){
        parent[RootY] = RootX;
    }
    else{
        parent[RootX] = RootY;
    }
    return 1;
}
```

### 2.2.2 find()

노드의 대푯값을 반환하는 기능을 가진 함수이다. 노드 a의 대푯값이 자신을 가르키지 않으면 계속해서 재귀적으로 자신의 뿌리를 찾고 그렇지 않으면 최종값은 그 노드의 대푯값이 되므로 그 값을 반환한다.

```
int find(int a){
    if(parent[a]==a)return a;
    return parent[a] = find(parent[a]);
}
```

### 2.2.3 Kurskal

Union-Find 연산을 통해 두 노드간 거리가 가장 짧은값을 pop 연산 하여 사이클의 유무에 따라 MST의 최단거리를 구하는 코드이다.

```
while(!pq.empty()){
    int w, a, b;
    tie(w,a,b) = pq.top(); pq.pop();
    if(coun == V-1)break;
    if(my_union(a,b)){
        sum+=w;
        coun++;
    }
}
```

### 2.2.4 Prim

heapq에 있는 두 노드 사이의 거리가 가장 작은 값으로부터 중복연산 및 push, pop 연산을 통해 MST를 확장하여 최단거리를 구하는 코드이다.

```
pq.push({0,1});
while(!pq.empty() && V<N){
    int current = pq.top().second;
    int value = pq.top().first;
    pq.pop();
    if(visited[current])continue;
    visited[current] = 1;
    sum+=value;
    V++;
    for(int i=0; i<adj[current].size(); i++){
```

```

        int next = adj[current][i].first;
        int nextValue = adj[current][i].second;
        if(!visited[next])pq.push({nextValue, next});
    }
}

```

## 2.3 복잡도 분석

### 2.3.1 Kruskal 시간복잡도

- 간선이 M개라고 할때 넣고 빼는데 드는 시간이  $M * \log M$  정도이고, 점들을 합치는 Union-Find 과정은 거의 상수시간 이므로 무시할수있다. 그러므로 Kruskal의 시간복잡도는  $O(E \log E)$ 이다.

### 2.3.2 Prim 시간복잡도

- 간선(M)을 입력으로 받고 저장하는데  $O(M)$ 이 걸리고 우선순위 큐를 이용해 최소 가중치 간선을 반복 선택하는 과정에서 각 간선이 최대 한번씩 삽입, 삭제가 되므로  $M * \log M$ 이다. 그러므로 전체 시간 복잡도는  $O(M \log M) \approx O(M \log V)$ 이다.

## 2.4 전체 코드

### 2.4.1 Kruskal

```

#include <iostream>
#include <vector>
#include <queue>
#include <utility>
#include <tuple>
using namespace std;
priority_queue<tuple<int,int,int>, vector<tuple<int,int,int>>,greater<tuple<int,int,int>>> p

int parent[10001]={0,};
int sum = 0;
int coun = 0;

int find(int a){
    if(parent[a]==a)return a;
    return parent[a] = find(parent[a]);
}

```

```

}

int my_union(int a, int b){
    int RootX = find(a);
    int RootY = find(b);
    if(RootX == RootY){
        return 0;
    }
    else if(RootX<RootY){
        parent[RootY] = RootX;
    }
    else{
        parent[RootX] = RootY;
    }
    return 1;
}

int main(){
    int V, E;
    scanf("%d %d", &V, &E);
    for(int i=1; i<=V; i++){
        parent[i] = i;
    }
    for(int i=1; i<=E; i++){
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        pq.push({c,a,b});
        pq.push({c,b,a});
    }
    while(!pq.empty()){
        int w, a, b;
        tie(w,a,b) = pq.top(); pq.pop();
        if(coun == V-1)break;
        if(my_union(a,b)){
            sum+=w;
            coun++;
        }
    }
    printf("%d", sum);
}

```

## 2.5 Prim

```
#include <iostream>
#include <utility>
#include <vector>
#include <queue>
using namespace std;
priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;

vector<pair<int,int>> adj[100001];
int visited[100001]={0,};
long long int sum = 0;
int V = 0;
int main(){
    int N, M;
    long long int R = 0;
    scanf("%d %d", &N, &M);
    for(int i=1; i<=M; i++){
        int a, b, c;
        scanf("%d %d %d", &a, &b, &c);
        R+=c;
        adj[a].push_back({b,c});
        adj[b].push_back({a,c});
    }
    pq.push({0,1});
    while(!pq.empty() && V<N){
        int current = pq.top().second;
        int value = pq.top().first;
        pq.pop();
        if(visited[current])continue;
        visited[current] = 1;
        sum+=value;
        V++;
        for(int i=0; i<adj[current].size(); i++){
            int next = adj[current][i].first;
            int nextValue = adj[current][i].second;
            if(!visited[next])pq.push({nextValue, next});
        }
    }
    if(V<N){
        printf("-1");
        return 0;
    }
```

```
}  
printf("%lld", R-sum);  
return 0;  
}
```

### 3 활용 방안

#### 1. 전력 계통 분리 운영

- 송전망을 분리해야할때 Kruskal 및 Prim 알고리즘을 사용하면 가장 비싼 간선은 제거하되 연결된 송전망들을 최소비용으로 구축할수있다.

### 4 인상깊었던 점

이전부터 트리의 최단경로 같은 문제를 볼때면 “그냥 다익스트라 쓰면되는거 아닌가?” 라고 생각하여 실제로 구현해보니 답이 제대로 안나왔었는데 이번주에 Tree와 Kruskal, Prim을 배움으로써 왜 안되는지, 트리가 무엇인지 알게되어서 행복했다.

### 5 참고문헌

[알고리즘] 크루스칼(Kruskal)과 프림(Prim)