20.12.30 동아리 풀이

선린인터넷고등학교 소프트웨어과
30610 나정휘
https://JusticeHui.github.io

0 문제 목록

- Z
- 수행 시간 - 2018 중앙대학교/숭실대학교 연합 대회 E번
- 완전 그래프의 최소 스패닝 트리
- Vacation Planning - USACO December 2013 Gold 1번
- Coronavirus Trend - ICPC 2020 Seoul Regional First Round D번
- Incomplete chess boards - TUD Contest 2005 9번
- 겹치는 선분 - TUD Contest 2005 3번
- Synchronization - JOI Open Contest 2013 2번

1 Z

재귀 함수 연습 문제입니다.
모든 칸을 Naive하게 방문하지 않고, 필요한 칸만 방문해야 합니다.

http://boj.kr/97c4e65ea0504ae6855d51e8a354e927

2 수행 시간

그래프를 잘 만든 뒤, 위상 정렬과 DP를 이용해 DAG의 최장 경로를 구하면 된다.

http://boj.kr/73f44336a6de459d821424959b306153

3 완전 그래프의 최소 스패닝 트리

메모리 제한이 16MB이다.

간선을 모두 저장할 수 없기 때문에 Kruskal Algorithm을 사용할 수 없다. 다른 알고리즘을 생각해봐야 한다.

MST를 구하는 다른 알고리즘으로는 Prim Algorithm과 Boruvka Algorithm이 있다. Boruvka는 자료가 많이 없으니 그냥 Prim을 구현하자.
당연히 Priority Queue 써서 구현하면 안 되고(Heap에 최대 $O(E)$개의 간선이 들어갈 수 있음), Naive하게 $O(N^2)$으로 구현하자.

http://boj.kr/44c0d2c3e19849a38ba17d8d97330b96

4 Vacation Planning

허브의 개수 K가 작으므로, (어떤 정점에서 각 허브로 최소 비용)과 (각 허브에서 다른 정점으로 가는 최소 비용)을 최소 비용을 각각 전처리하면 된다는 생각을 할 수 있다.
이 값들을 알고 있으면 (시작점에서 어떤 허브로 가는 최소 비용) + (어떤 허브에서 끝점으로 가는 최소 비용) 중 최솟값이 답이 되기 때문이다.

각 허브에서 다른 정점으로 가는 최소 비용은 단순히 Dijkstra Algorithm을 K번 돌리면 된다.
마찬가지로, 어떤 정점에서 각 허브로 가는 것은, 그래프의 간선 방향을 반대로 뒤집고 Dijkstra Algorithm을 돌리면 된다.

$O(M \log N)$ Dijkstra Algorithm을 K번 돌리고, 쿼리에 대한 답은 $O(K)$만에 구할 수 있으므로 전체 시간 복잡도는 $O(MK \log N + KQ)$가 된다.

http://boj.kr/c167b50579384c31baf7e105f5f8d719

5 Coronavirus Trend

DP 느낌이 강하게 난다. 점화식을 세워보자.

- $A_i = i$번째 원소까지 봤을 때, 마지막에 감소한 수열의 최대 길이
- $B_i = i$번째 원소까지 봤을 때, 마지막에 2번 감소한 수열의 최대 길이
- $C_i = i$번째 원소까지 봤을 때, 마지막에 증가한 수열의 최대 길이
- $D_i = i$번째 원소까지 봤을 때, 마지막에 2번 증가한 수열의 최대 길이

Segment Tree로 LIS를 구하는 것처럼 DP를 계산하면 $O(N \log N)$에 문제를 풀 수 있다.

http://boj.kr/8dcc4387b53f423d91adde017b05c009

6 Incomplete chess boards

격자 그래프는 이분 그래프다.

Maximum Bipartite Matching이 Perfect Matching인지 확인하면 된다.

http://boj.kr/b438958e512a46e888c5be96bcc169ae

7 겹치는 선분

직선은 $y = ax + b$로 표현할 수 있고, 두 직선의 a, b가 모두 같은 경우에만 겹칠 수 있다.
직선을 (a, b) 값 별로 나눈 다음, 각각에 대해 독립적으로 문제를 해결해도 충분하다.

(a, b)가 같은 경우, 간단한 스위핑을 통해 문제를 해결할 수 있다.

http://boj.kr/0c09704849bc4339a0961657c7197d46

8 Synchronization

한글 지문은 oj.uz에서 볼 수 있다:
https://oj.uz/problem/view/JOI13_synchronization

HLD + set / Euler Tour Trick + set을 이용하는 풀이가 존재하지만, 풀이를 생각하는 것과 구현이 모두 귀찮다.
약간의 사전지식이 있으면 훨씬 쉬운 풀이를 소개한다.

간선 제거 쿼리가 없는 문제를 생각해보자.
각 컴포넌트의 "정답"을 루트 정점에 저장하면서, 간선이 추가되면 컴포넌트를 Union Find를 통해 합치면 된다.
이때 각 컴포넌트의 "정답"은 단순히 컴포넌트의 크기가 된다.

간선을 끊는다면? **"Link Cut Tree를 쓰면 된다!"**라는 단순하고도 명쾌한 해답을 찾을 수 있다.
Union Find처럼 각 컴포넌트의 "정답"을 루트 정점에서 관리하면 된다.

하지만 간선을 끊는 쿼리가 있다면 두 컴포넌트를 합칠 때 정보의 교집합이 생길 수 있다. 이는 트리의 간선이 연결하는 정점들이 바뀌지 않기 때문에 쉽게 처리할 수 있다.
간선 $E = (u, v)$가 마지막으로 끊어지는 시점에서의 컴포넌트의 크기 $S(E)$를 알고 있다면, E가 다시 트리에 추가될 때 해당 컴포넌트의 "정답"은 $Ans(u) + Ans(v) - S(E)$가 된다.

간선을 끊고, 붙이고, 루트 정점을 구하는 연산은 모두 Link Cut Tree를 이용해서 처리할 수 있다.
다음 페이지에 Link Cut Tree의 뼈대 코드를 올렸으니 참고하면 된다.

http://boj.kr/27a85c1b8cd74345a4e84100b4a552f6

8.1 Link-Cut Tree

Usage: Link-Cut Tree 기본 코드 / BOJ 13539 트리와 쿼리 11
Time Complexity: Splay, Access: amortized $O(\log N)$
Link, Cut, GetLCA/Root/Par: depend on Access
Author: origin **sebinkim**, modify **Justice_Hui**

```
#include <bits/stdc++.h>
using namespace std;

struct Node{
    Node *l, *r, *p;
    Node() : l(nullptr), r(nullptr), p(nullptr) {}
    bool IsLeft() const { return this == p->l; }
    bool IsRoot() const { return !p || (p->l != this && p->r != this); }
    void Rotate(){
        if(IsLeft()){
            if(r) r->p = p;
            p->l = r; r = p;
        }
        else{
            if(l) l->p = p;
            p->r = l; l = p;
        }
        if(!p->IsRoot()) (p->IsLeft() ? p->p->l : p->p->r) = this;
        auto t = p; p = t->p; t->p = this;
    }
};
void Splay(Node *x){
    for(; !x->IsRoot(); x->Rotate()){
        if(!x->p->IsRoot()){
            if(x->IsLeft() == x->p->IsLeft()) x->p->Rotate();
            else x->Rotate();
        }
    }
}
void Access(Node *x){
    Splay(x); x->r = nullptr;
    for(; x->p; Splay(x)) Splay(x->p), x->p->r = x;
}
void Link(Node *p, Node *c){
    Access(c); Access(p);
    c->l = p; p->p = c;
}
void Cut(Node *x){
    Access(x);
```

```
    x->l->p = nullptr; x->l = nullptr;
}
Node* GetLCA(Node *x, Node *y){
    Access(x); Access(y); Splay(x);
    return x->p ? x->p : x;
}
Node* GetRoot(Node *x){
    Access(x); while(x->l) x = x->l;
    Access(x); return x;
}
Node* GetPar(Node *x){
    Access(x); if(!x->l) return nullptr;
    x = x->l; while(x->r) x = x->r;
    Access(x); return x;
}

int N, Q;
Node Tree[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    while(Q--){
        int op, a, b; cin >> op;
        if(op == 1) cin >> a >> b, Link(Tree+b, Tree+a);
        else if(op == 2) cin >> a, Cut(Tree+a);
        else cin >> a >> b, cout << GetLCA(Tree+a, Tree+b) - Tree << "\n";
    }
}
```