2023 SCCC 봄 #12

BOJ 28065. SW 수열 구하기

인접한 두 수의 차이는 차례대로 $N-1, N-2, \cdots, 2, 1$이 되어야 하므로 맨 앞에 오는 두 수는 $1, N$이 되어야 합니다. 이후 $2, N-1, 3, N-2, \cdots$ 를 붙이면 문제를 해결할 수 있습니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    deque<int> V(N);
    iota(V.begin(), V.end(), 1);
    for(int i=0; i<N; i++){
        if(i % 2 == 0) cout << V.front() << " ", V.pop_front();
        else cout << V.back() << " ", V.pop_back();
    }
}
```

BOJ 28067. 기하가 너무 좋아

두 삼각형 중 하나를 삼각형 하나를 돌리거나 뒤집었을 때 일치한다는 것은 두 삼각형이 합동이라는 것을 의미합니다. 따라서 합동인 것을 하나로 취급했을 때 만들 수 있는 삼각형의 개수를 구하는 문제입니다.

두 삼각형이 합동일 조건 중 하나로 SSS 합동이 있습니다. 사용할 수 있는 점이 최대 $11^2 = 121$개이므로 가능한 모든 삼각형을 만든 다음, 삼각형을 구성하는 변의 길이의 목록이 서로 다른 삼각형의 개수를 구하면 됩니다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using Point = pair<int, int>;

int CCW(const Point &p1, const Point &p2, const Point &p3){
    return (p2.x - p1.x) * (p3.y - p1.y) - (p3.x - p1.x) * (p2.y - p1.y);
}

int Dist(const Point &p1, const Point &p2){
    return (p2.x - p1.x) * (p2.x - p1.x) + (p2.y - p1.y) * (p2.y - p1.y);
}

int N, M;
vector<Point> V;
vector<array<int,3>> R;

array<int,3> f(int i, int j, int k){
    array<int,3> res;
    res[0] = Dist(V[i], V[j]);
    res[1] = Dist(V[j], V[k]);
```

```
    res[2] = Dist(V[k], V[i]);
    sort(res.begin(), res.end());
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; V.reserve((N+1) * (M+1));
    for(int i=0; i<=N; i++) for(int j=0; j<=M; j++) V.emplace_back(i, j);
    for(int i=0; i<V.size(); i++) for(int j=i+1; j<V.size(); j++) for(int k=j+1;
k<V.size(); k++) if(CCW(V[i], V[j], V[k])) R.push_back(f(i, j, k));
    sort(R.begin(), R.end());
    R.erase(unique(R.begin(), R.end()), R.end());
    cout << R.size();
}
```

BOJ 5021. 왕위 계승

s번째 사람의 혈통을 $D(s)$, s의 부모를 각각 A_s, B_s라고 하면, $D(s) = \frac{D(A_s)+D(B_s)}{2}$로 계산할 수 있습니다. 족보에 모순이 없기 때문에 그래프로 나타내면 DAG가 되고, 따라서 DP를 이용해 모든 사람의 혈통을 계산할 수 있습니다.

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

int N, M;
string S;
map<string, array<string,2>> G;
map<string, double> D;

double f(const string &s){
    if(s == S) return 1;
    else if(G[s][0].empty()) return 0;
    else if(D.find(s) != D.end()) return D[s];
    else return D[s] = (f(G[s][0]) + f(G[s][1])) / 2;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    for(int i=1; i<=N; i++){
        string a, b, c; cin >> a >> b >> c;
        G[a][0] = b; G[a][1] = c;
    }
    string res; double mx = -1;
    for(int i=1; i<=M; i++){
        string s; cin >> s;
        auto now = f(s);
        if(now > mx) res = s, mx = now;
    }
    cout << res;
}
```

BOJ 12944. 재미있는 숫자 놀이

A_i로 나누어 떨어지는 수들의 집합을 S_i라고 합시다. $|A_1 \cap A_2 \cap \cdots \cap A_N|$을 구해야 합니다.

합집합의 크기는 포함과 배제의 원리를 이용해서 구할 수 있습니다. 모든 홀수 개의 집합의 교집합의 크기를 더하고, 짝수 개의 집합의 교집합의 크기를 빼는 것으로 계산할 수 있고, 교집합의 개수를 세는 것은 A_i들의 최소공배수로 나누어 떨어지는 수의 개수를 구하면 됩니다.

교집합을 선택하는 2^N가지의 경우를 모두 확인해야 하고, 각 경우에서 $O(N \log X)$ 시간에 최소공배수를 구한 다음 $O(N)$ 시간에 교집합의 크기를 구해야 하므로 전체 시간 복잡도는 $O(2^N N \log X)$가 됩니다.

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

ll N, K, A[22], R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> K >> N;
    for(int i=0; i<N; i++) cin >> A[i];
    for(int bit=1; bit<(1<<N); bit++){
        ll now = 1;
        for(int i=0; i<N; i++){
            if(~bit >> i & 1) continue;
            now = now / __gcd(now, A[i]) * A[i];
            if(now > K) break;
        }
        if(now > K) continue;
        R += K / now * (__builtin_popcount(bit) % 2 ? 1 : -1);
   \ }
    cout << R;
}
```

BOJ 12950. 팰린드롬 보행

빈 문자열 또는 문자 1개로 구성된 문자열에서 시작해서, 매 단계마다 양쪽 끝에 동일한 문자를 하나씩 붙이는 방식으로 팰린드롬을 만든다고 생각합시다. 빈 문자열에서 시작하는 것은 (i, i)에서 시작하는 것, 문자 1개로 구성된 문자열에서 시작하는 것은 간선으로 연결된 두 정점 (u, v)에서 시작하는 것이라고 생각할 수 있습니다. 0번 정점에서 1번 정점으로 가는 팰린드롬 보행을 찾는 것은 매번 같은 문자를 갖고 있는 간선을 양쪽 끝에 붙이면서 $(0, 1)$로 이동하는 것과 동일합니다.

이런 관점에서 생각하면, 정점이 $O(N^2)$개, 간선이 최대 $O(N^4)$개인 그래프 위에서 최단 경로를 수행하는 것으로 문제를 해결할 수 있음을 알 수 있습니다. 다익스트라 알고리즘을 사용해서 $O(N^4 \log N)$에 해결해도 되고, BFS와 같은 방식을 이용해서 $O(N^4)$ 시간에 해결할 수도 있습니다.

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

int N, M, G[22][22], D[22][22];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
```

```
        int u, v; char c; cin >> u >> v >> c; u++; v++;
        G[u][v] = G[v][u] = c - 'a' + 1;
    }
    memset(D, 0x3f, sizeof D);
    priority_queue<tuple<int,int,int>, vector<tuple<int,int,int>>, greater<>> Q;
    for(int i=1; i<=N; i++) for(int j=1; j<=26; j++) Q.emplace(D[i][i]=0, i, i);
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) if(G[i][j]) Q.emplace(D[i]
[j]=1, i, j);
    while(!Q.empty()){
        auto [c,s,e] = Q.top(); Q.pop();
        if(c != D[s][e]) continue;
        if(s == 1 && e == 2){ cout << c; return 0; }
        for(int i=1; i<=N; i++){
            if(!G[i][s]) continue;
            for(int j=1; j<=N; j++){
                if(!G[e][j] || G[i][s] != G[e][j]) continue;
                if(D[i][j] > c + 2) Q.emplace(D[i][j]=c+2, i, j);
            }
        }
    }
    cout << -1;
}
```

BOJ 15807. 빛영우

(x, y)를 $(x+y, x-y)$로 바꾸면 모든 점 사이의 거리를 $\sqrt{2}$배로 늘리면서 좌표계를 45도 회전할 수 있습니다. 좌표계를 45도 회전시키면 스포트라이트를 비추는 것은 적당한 직사각형 영역에 1을 더하는 쿼리, 빛의 세기를 구하는 것은 특정 지점의 값을 구하는 쿼리가 됩니다. 플레인 스위핑을 해도 되고, 업데이트가 모두 주어진 다음에 쿼리를 한다는 점을 이용해서 2차원 누적 합을 사용해도 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 13;

struct Event{
    int x, y, v;
    Event() = default;
    Event(int x, int y, int v) : x(x), y(y), v(v) {}
    bool operator < (const Event &e) const { return tie(x,y,v) <
tie(e.x,e.y,e.v); }
};

int N, Q, R[101010], T[SZ];
vector<Event> E;
void Add(int x){ for(x+=3; x<SZ; x+=x&-x) T[x]++; }
int Get(int x){ int r=0; for(x+=3; x; x-=x&-x) r += T[x]; return r; }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,x,y; i<=N; i++) cin >> x >> y, E.emplace_back(x+y+3000, y-
x+3000, 0);
    cin >> Q;
    for(int i=1,x,y; i<=Q; i++) cin >> x >> y, E.emplace_back(x+y+3000, y-
x+3000, i);
    sort(E.begin(), E.end());
```

```
    for(int i=0; i<E.size(); i++){
        if(E[i].v == 0) Add(E[i].y);
        else R[E[i].v] = Get(E[i].y);
    }
    for(int i=1; i<=Q; i++) cout << R[i] << "\n";
}
```

BOJ 28077. 사탕 팔찌

문자열 S가 등장하기 위해서는 S 앞에 $S[1:]$로 끝나는 문자열이 와야 하고, S 뒤에 $S[0:-1]$로 시작하는 문자열이 와야 합니다. 즉, 문자열 S를 만드는 것은 $S[1:]$로 끝나는 문자열에서 $S[0:-1]$로 시작하는 문자열로 이동하는 것이라고 생각할 수 있습니다.

따라서 길이가 $K-1$인 모든 문자열을 정점으로 만든 다음, 길이가 K인 모든 문자열 S에 대해 $S[1:]$에서 $S[0:-1]$로 가는 간선을 만들면, 이 문제는 모든 간선을 정확히 한 번씩 사용하는 보행을 구하는 문제가 됩니다. 오일러 회로는 선형 시간에 구할 수 있으므로 제한 시간 안에 문제를 해결할 수 있습니다.

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

int N, K, U[11], In[404040], Out[404040];
vector<int> G[404040];
vector<string> V, E, R;
string S;

void Make(int dep, int len, vector<string> &vec){
    if(dep == len){ vec.push_back(S); return; }
    for(int i=0; i<N; i++) if(!U[i]) S += char(i+'A'), U[i] = 1, Make(dep+1,
len, vec), S.pop_back(), U[i] = 0;
}

void DFS(int v){
    while(!G[v].empty()){
        int nxt = G[v].back(); G[v].pop_back();
        DFS(nxt);
        R.push_back(V[v] + V[nxt].back());
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    Make(0, K-1, V); Make(0, K, E);
    sort(V.begin(), V.end());
    sort(E.begin(), E.end());
    for(const auto &s : E){
        int u = lower_bound(V.begin(), V.end(), s.substr(0, K-1)) - V.begin();
        int v = lower_bound(V.begin(), V.end(), s.substr(1, K-1)) - V.begin();
        G[u].push_back(v); Out[u]++; In[v]++;
    }
    for(int i=0; i<V.size(); i++) if(In[i] != Out[i]) { cout << "NO"; return 0;
}
    DFS(0); reverse(R.begin(), R.end());
    if(R.size() != E.size()){ cout << "NO"; return 0; }
    cout << "YES\n";
    for(auto i : R) cout << i << " ";
}
```

BOJ 17738. Voltage

이런 문제는 보통 tree edge와 back edge로 나눠서 생각하는 것이 편합니다.

back edge가 정답이 될 수 있는 상황을 생각해 봅시다. 색깔이 같은 두 정점을 연결하는 back edge가 2개 이상이면 tree edge를 없애지 않는 이상 이분 그래프가 될 수 없습니다. 따라서 back edge가 정확히 1개일 때만 back edge가 정답이 될 수 있고, 이때 정답이 1만큼 증가합니다.

이제 tree edge를 생각해 봅시다. tree edge가 두 정점 u, v를 연결하고, 일반성을 잃지 않고 $dep(u) < dep(v)$라고 합시다. 만약 u 밑에서 u 위로 올라가는 bad edge가 있으면 tree edge를 없애고 bad edge 하나를 tree edge로 뒤집은 다음 한쪽 컴포넌트의 색깔을 반전시키면 됩니다. 따라서 모든 bad edge가 u의 밑에서 위로 올라가는 형태이고, 밑에서 위로 올라오는 good edge가 없으면 간선 (u, v)는 정답이 될 수 있습니다. 반면, bad edge가 u 밑에 머물러 있으면 어떻게 해도 바꿀 수 없으므로 정답이 될 수 없습니다.

따라서 u, v를 연결하는 tree edge는 u 밑에서 출발하는 모든 bad edge가 u 위로 올라오고, u 밑에서 위로 올라오는 good edge가 없을 때만 정답이 될 수 있습니다. 트리 위에서 누적 합을 잘 사용하면 선형 시간에 문제를 해결할 수 있습니다.

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

int N, M, P[101010], D[101010], Good[101010], Bad[101010], GOOD, BAD, R;
vector<pair<int,int>> G[101010];

void Tarjan(int v, int b=-1){
    for(auto [i,id] : G[v]){
        if(id == b) continue;
        if(!P[i]) P[i] = id, D[i] = D[v] + 1, Tarjan(i, id);
        else if(D[v] <= D[i]) continue;
        else if(D[v] % 2 == D[i] % 2) Bad[v]++, Bad[i]--, BAD++;
        else Good[v]++, Good[i]--, GOOD++;
    }
}

void Sum(int v){
    int lst = -1;
    for(auto [i,id] : G[v]){
        if(P[i] == id && i != lst) Sum(i), Good[v] += Good[i], Bad[v] += Bad[i];
        lst = i;
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1,u,v; i<=M; i++) cin >> u >> v, G[u].emplace_back(v, i),
G[v].emplace_back(u, i);
    for(int i=1; i<=N; i++) sort(G[i].begin(), G[i].end()),
G[i].erase(unique(G[i].begin(), G[i].end()), G[i].end());
    for(int i=1; i<=N; i++) if(!P[i]) P[i] = -1, Tarjan(i, -1), Sum(i);
    R += BAD == 1;
    for(int i=1; i<=N; i++) R += P[i] != -1 && Bad[i] == BAD && Good[i] == 0;
    cout << R;
}
```

BOJ 18941. 평면그래프와 게임

간선을 끊는 쿼리를 먼저 생각해 봅시다. 평면 그래프에서는 $V-E+F=C+1$이 성립합니다. 간선을 하나 끊으면 E가 1만큼 감소하고, 이때 컴포넌트의 개수 C가 증가하기 위해서는 V, F 값이 바뀌지 않아야 합니다. 정점의 개수인 V는 항상 일정하므로 면의 개수인 F가 감소하는지 확인하면 됩니다. 즉, 간선 하나를 끊었을 때 F가 감소하면 컴포넌트가 분리되지 않은 것이고, F가 변하지 않으면 컴포넌트가 분리된 것입니다. F값의 변화는 dual graph를 이용해 관리할 수 있습니다.

두 정점 u, v를 잇는 간선이 없어지는 것으로 인해 컴포넌트가 분리되었다면 그래프는 u를 포함하는 컴포넌트와 v를 포함하는 컴포넌트로 분할됩니다. 연결성을 판별하기 위해 각 정점이 속한 컴포넌트의 ID를 관리할 건데, 둘로 나뉘게 되는 컴포넌트 중 크기가 더 작은 컴포넌트의 ID를 갱신하면 small to large 원리에 의해 각 정점의 ID가 최대 $O(\log N)$번만 바뀌게 됩니다. 모든 쿼리에서 ID를 바꾸는 연산은 $O(N \log N)$번만 수행하면 되므로 간선을 끊는 쿼리를 amortized $O(\log N)$ 시간에 처리할 수 있습니다.

크기가 더 작은 쪽만 색칠하기 위해서는 두 컴포넌트의 크기를 알아야 합니다. 단순하게 DFS/BFS를 돌리면 $O(N^2)$이 되기 때문에 당연히 안 되고, 스택/큐에 정점을 하나씩 추가하는 방식으로 구현해야 합니다. KOI 2016 고등부 3번 트리 문제와 비슷한 방식으로 해결할 수 있습니다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;
istream& operator >> (istream &in, Point &p){ return in >> p.x >> p.y; }

int CCW(const Point &p1, const Point &p2, const Point &p3){
    ll res = (p2.x - p1.x) * (p3.y - p1.y) - (p3.x - p1.x) * (p2.y - p1.y);
    return (res > 0) - (res < 0);
}

int N, M, Q, X, Y, P[202020];
Point A[101010];
vector<pair<int,int>> G[101010];
set<pair<int,int>> GS[101010];
map<pair<int,int>, int> EID;
int C[101010], ID[101010], Color, Cnt;

int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){ return Find(u) != Find(v) && (P[P[u]]=P[v], true); }

void AddEdge(int u, int v, int id){
    if(u > v) swap(u, v);
    EID[{u,v}] = id;
    G[u].emplace_back(v, id); GS[u].emplace(v, id);
    G[v].emplace_back(u, id); GS[v].emplace(u, id);
}

void RemoveEdge(int u, int v, int id){
    if(u > v) swap(u, v);
    EID.erase({u,v});
    GS[u].erase({v,id}); GS[v].erase({u,id});
}

void DualGraph(){
```

```
    iota(P, P+202020, 0);
    for(int i=1; i<=N; i++){
        sort(G[i].begin(), G[i].end(), [&](pair<int,int> e1, pair<int,int> e2){
            Point a = A[e1.first], b = A[e2.first];
            return (a > A[i]) != (b > A[i]) ? a > b : CCW(a, A[i], b) > 0;
        });
        for(int j=0; j<G[i].size(); j++){
            int k = j ? j - 1 : (int)G[i].size() - 1;
            Point p1 = A[G[i][k].first]; int u = G[i][k].second << 1 | 1;
            Point p2 = A[G[i][j].first]; int v = G[i][j].second << 1;
            u ^= p1 > A[i]; v ^= p2 > A[i];
            Merge(u, v);
        }
    }
}

void Init(int v){
    ID[v] = Color;
    for(auto [i,id] : G[v]) if(!ID[i]) Init(i);
}

void Divide(int a, int b){
    Color++; Cnt++;
    vector<pair<int,int>> stk[2];
    vector<int> vec[2];
    stk[0].emplace_back(a, 0);
    stk[1].emplace_back(b, 0);
    while(!stk[0].empty() && !stk[1].empty()){
        for(int i=0; i<2; i++){
            auto [v,lst] = stk[i].back(); stk[i].pop_back();
            vec[i].push_back(v); C[v] = Cnt;
            auto it = GS[v].lower_bound({lst,-1});
            if(it == GS[v].end()) continue;
            stk[i].emplace_back(v, it->first+1);
            if(C[it->first] != Cnt) stk[i].emplace_back(it->first, 0);
        }
    }
    if(stk[0].empty()) for(auto i : vec[0]) ID[i] = Color;
    else for(auto i : vec[1]) ID[i] = Color;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> Q >> X >> Y;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1,u,v; i<=M; i++) cin >> u >> v, AddEdge(u, v, i);
    DualGraph();
    for(int i=1; i<=N; i++) if(!ID[i]) Color++, Init(i);

    ll cnt = 0;
    for(int q=1; q<=Q; q++){
        int op, u, v; cin >> op >> u >> v;
        u = (u - 1 + cnt * X) % N + 1;
        v = (v - 1 + cnt * Y) % N + 1;
        if(u > v) swap(u, v);
        if(op == 1){
            if(!EID.count({u,v})) continue;
            int id = EID[{u,v}];
```

```
            RemoveEdge(u, v, id);
            if(!Merge(id<<1, id<<1|1)) Divide(u, v);
        }
        else if(ID[u] == ID[v]) cout << "A\n", cnt++;
        else cout << "D\n";
    }
}
```