2023 SCCC 봄 #9

BOJ 27973. 지연 평가

이 문제는 등차 수열을 관리하는 문제입니다. 첫 번째 원소, 마지막 원소, 공차를 관리합시다.

모든 원소에 x를 더하는 쿼리는 첫 번째 원소와 마지막 원소를 x만큼 증가시키면 되고, x를 곱하는 쿼리는 첫 번째 원소, 마지막 원소, 공차에 모두 x를 곱하면 되고, 가장 작은 n개의 원소를 제거하는 것은 첫 번째 원소를 $d \times n$ 만큼 증가시키면 됩니다. 따라서 모든 쿼리를 상수 시간에 처리할 수 있습니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll L = 1, R = 1234567890123LL, D = 1;
    int Q; cin >> Q;
    for(int q=1; q<=Q; q++){
        ll op, v=0; cin >> op;
        if(op == 3){ cout << L << "\n"; continue; }
        cin >> v;
        if(op == 0) L += v, R += v;
        if(op == 1) L *= v, R *= v, D *= v;
        if(op == 2) L += v * D;
    }
}
```

BOJ 22968. 균형

높이가 h인 트리를 만들기 위해 필요한 정점의 최소 개수를 $D(h)$라고 정의합시다. 모든 $h > 0$에 대해 $D(h-1) < D(h)$를 만족한다는 것은 쉽게 알 수 있습니다.

높이가 h인 트리를 만들기 위해서는 최소한 한쪽 서브 트리의 높이는 $h-1$이어야 하고, 문제에 조건에 의해서 반대쪽 서브 트리의 높이는 $h-2$ 이상이어야 합니다. 따라서 $D(h) = D(h-1) + D(h-2)$가 성립합니다.

정점의 개수 V가 주어졌을 때 V개의 정점으로 만들 수 있는 최대 높이는 $D(h) \leq V$를 만족하는 가장 큰 h를 찾으면 됩니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    long long D[44] = {0, 1, 2};
    for(int i=3; i<44; i++) D[i] = D[i-1] + D[i-2] + 1;

    int T; cin >> T;
    while(T--){
        int V; cin >> V;
        int res = upper_bound(D+1, D+44, V) - D - 1;
        cout << res << "\n";
```

```
    }
}
```

BOJ 14678. 어그로 끌린 영선

시작 정점에서 출발해 짝수 번 이동해서 리프 정점에 도착할 때만 승리할 수 있습니다. 따라서 각 정점과 짝수만큼 떨어져 있는 정점의 개수를 구하면 문제를 해결할 수 있습니다.

아무 정점이나 하나 잡고 DFS를 수행하면서 각 정점의 깊이가 홀수인지 짝수인지 저장합시다. 각 정점의 정답을 구하는 것은, 그 정점과 깊이를 2로 나눈 나머지가 동일한 리프 정점의 개수를 구하면 되고, 이는 $O(N)$ 시간에 해결할 수 있습니다.

시작 정점의 차수가 1일 때를 주의해서 구현해야 합니다.

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

int N, D[1010101], C[2], R;
vector<int> G[1010101];

void DFS(int v, int b=-1){
    if(G[v].size() <= 1) C[D[v]]++;
    for(auto i : G[v]) if(i != b) D[i] = D[v] ^ 1, DFS(i, v);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,u,v; i<N; i++) cin >> u >> v, G[u].push_back(v),
G[v].push_back(u);
    DFS(1);
    for(int i=1; i<=N; i++) R = max(R, C[D[i]] - (G[i].size() == 1));
    cout << R;
}
```

BOJ 13283. Daruma Otoshi

전형적인 구간 DP 문제입니다.

$i, i+1, \cdots, j$번째 블록만 고려했을 때 없앨 수 있는 블록의 최대 개수를 $D(i, j)$라고 정의합시다.

만약 $i+1, i+2, \cdots, j-1$번째 블록을 모두 없앨 수 있고 $|A_i - A_j| \leq 1$이면 모든 블록을 없앨 수 있습니다. 따라서 이 경우에는 $D(i, j) = j - i + 1$입니다.

그렇지 않은 경우에는 $D(i, j) = \max_{i \leq k < j} \{D(i, k) + D(k+1, j)\}$로 계산할 수 있습니다.

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

int N, A[333], D[333][333];

void Solve(){
    for(int i=1; i<=N; i++) cin >> A[i];
    memset(D, 0, sizeof D);
    for(int i=2; i<=N; i++) D[i-1][i] = abs(A[i-1]-A[i]) <= 1 ? 2 : 0;
    for(int d=2; d<=N; d++){
        for(int i=1; i+d<=N; i++){
```

```
            int j = i + d;
            for(int k=i; k<j; k++) D[i][j] = max(D[i][j], D[i][k] + D[k+1][j]);
            if((j-i+1) % 2 == 0 && D[i+1][j-1] == j-i-1 && abs(A[i]-A[j]) <= 1)
D[i][j] = max(D[i][j], j-i+1);
        }
    }
    cout << D[1][N] << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    while(cin >> N && N) Solve();
}
```

BOJ 16156. 장애물 달리기

M번째 열에서 출발하는 multi source dijkstra를 돌리면서, 최단 경로가 어떤 source에서 출발했는지 관리하면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int di[] = {1, -1, 0, 0};
constexpr int dj[] = {0, 0, 1, -1};

ll N, M, A[555][555], D[252525], P[252525], R[555];
vector<pair<ll,ll>> G[252525];
inline int ID(int i, int j){ return (i - 1) * M + j; }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) cin >> A[i][j];
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            for(int k=0; k<4; k++){
                int r = i + di[k], c = j + dj[k];
                if(1 <= r && r <= N && 1 <= c && c <= M)
G[ID(r,c)].emplace_back(ID(i,j), A[r][c]);
            }
        }
    }
    memset(D, 0x3f, sizeof D);
    priority_queue<pair<ll,ll>, vector<pair<ll,ll>>, greater<>> Q;
    for(int i=1; i<=N; i++) Q.emplace(D[ID(i,M)]=0, ID(i,M)), P[ID(i,M)] = i;
    while(!Q.empty()){
        auto [c,v] = Q.top(); Q.pop();
        if(c == D[v]) for(auto [i,w] : G[v]) if(D[i] > c + w)
Q.emplace(D[i]=c+w, i), P[i] = P[v];
    }
    for(int i=1; i<=N; i++) R[P[ID(i,1)]] += 1;
    for(int i=1; i<=N; i++) cout << R[i] << "\n";
}
```

BOJ 1542. 체스 연습

Sprague grundy theorem을 이용해야 한다는 것은 쉽게 알 수 있지만, 모든 기물이 아닌 하나의 기물만 $(0,0)$에 이동시키는 것이 승리 조건이기 때문에 그런디 정리를 그대로 적용하는 것은 불가능합니다.

하나라도 $(0,0)$에 이동시키는 것이 승리 조건이라는 것은, 내 행동으로 인해 모든 기물이 $(0,0)$으로 이동 가능한 상태가 되었을 때 패배한다는 것과 같습니다. 따라서 $i=0$ 또는 $j=0$ 또는 $i=j$인 칸들의 그런디 수를 0으로 지정하면 문제를 해결할 수 있습니다.

처음부터 $i=0, j=0, i=j$인 칸이 주어지는 경우를 주의해서 구현해야 합니다.

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

int N=100, A[111][111], C[333];

void Init(){
    memset(A, 0, sizeof A);
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++){
            memset(C, 0, sizeof C);
            for(int k=0; k<i; k++) if(k != j) C[A[k][j]] = 1;
            for(int k=0; k<j; k++) if(i != k) C[A[i][k]] = 1;
            for(int k=0; k<min(i,j); k++) C[A[i-k-1][j-k-1]] = 1;
            for(; C[A[i][j]]; ) A[i][j]++;
        }
    }
}

bool Solve(){
    int K, R = 0, F = 0; cin >> K;
    for(int i=0; i<K; i++){
        int r, c; cin >> r >> c;
        if(r == 0 || c == 0 || r == c) F = 1;
        else R ^= A[r-1][c-1];
    }
    return F || R != 0;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    Init();
    int TC = 5;
    for(int tc=1; tc<=TC; tc++) cout << "DS"[Solve()] << "\n";
}
```

BOJ 18947. Tree Hull

트리 압축을 생각해 보면, DFS Ordering 상에서 인접한 두 정점을 잇는 경로에 속한 간선만 고려해도 충분하다는 것을 알 수 있습니다. 따라서 HLD와 세그먼트 트리를 이용해 각 간선이 몇 번 사용되었는지 관리하고, 1번 이상 사용된 간선들의 가중치 합을 구하면 $O(Q \log^2 N)$ 시간에 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int SZ = 1 << 19;
```

```
namespace segment_tree{
    ll sum[SZ], tree[SZ<<1], cnt[SZ<<1];
    void build(){
        partial_sum(sum, sum+SZ, sum);
    }
    void update(int l, int r, int v, int node=1, int s=0, int e=SZ-1){
        if(r < s || e < l) return;
        if(l <= s && e <= r) cnt[node] += v;
        else{
            int m = (s + e) / 2;
            update(l, r, v, node<<1, s, m);
            update(l, r, v, node<<1|1, m+1, e);
        }
        if(cnt[node]) tree[node] = sum[e] - (s ? sum[s-1] : 0);
        else tree[node] = s == e ? 0 : tree[node<<1] + tree[node<<1|1];
    }
    ll query(int l, int r, int node=1, int s=0, int e=SZ-1){
        if(r < s || e < l) return 0;
        if(l <= s && e <= r) return tree[node];
        int m = (s + e) / 2;
        return query(l, r, node<<1, s, m) + query(l, r, node<<1|1, m+1, e);
    }
}

ll N, Q, W[SZ];
int Top[SZ], Dep[SZ], Par[SZ], Sz[SZ], In[SZ];
vector<pair<int,int>> Inp[SZ];
vector<int> G[SZ];

void DFS0(int v, int b=-1){
    for(auto [i,w] : Inp[v]) if(i != b) Dep[i] = Dep[v] + 1, Par[i] = v,
G[v].push_back(i), W[i] = w, DFS0(i, v);
}
void DFS1(int v){
    Sz[v] = 1;
    for(auto &i : G[v]){
        DFS1(i); Sz[v] += Sz[i];
        if(Sz[i] > Sz[G[v][0]]) swap(i, G[v][0]);
    }
}
void DFS2(int v){
    static int pv = 0; In[v] = ++pv;
    for(auto i : G[v]) Top[i] = i == G[v][0] ? Top[v] : i, DFS2(i);
}
void Update(int u, int v, int w){
    for(; Top[u]!=Top[v]; u=Par[Top[u]]){
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        segment_tree::update(In[Top[u]], In[u], w);
    }
    if(In[u] > In[v]) swap(u, v);
    segment_tree::update(In[u]+1, In[v], w);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
```

```
    for(int i=1,u,v,w; i<N; i++) cin >> u >> v >> w, Inp[u].emplace_back(v, w),
Inp[v].emplace_back(u, w);
    DFS0(1); DFS1(1); DFS2(Top[1]=1);
    for(int i=1; i<=N; i++) segment_tree::sum[In[i]] = W[i];
    segment_tree::build();

    cin >> Q;
    set<pair<int,int>> S;
    for(int q=1; q<=Q; q++){
        char c; int v; cin >> c >> v;
        if(c == '+'){
            auto it = S.lower_bound(make_pair(In[v], v));
            if(it != S.begin() && it != S.end()) Update(prev(it)->second, it-
>second, -1);
            if(it != S.end()) Update(v, it->second, +1);
            if(it != S.begin()) Update(v, prev(it)->second, +1);
            S.emplace(In[v], v);
        }
        else{
            auto it = S.find(make_pair(In[v], v));
            if(it != S.begin()) Update(v, prev(it)->second, -1);
            if(next(it) != S.end()) Update(v, next(it)->second, -1);
            if(it != S.begin() && next(it) != S.end()) Update(prev(it)->second,
next(it)->second, +1);
            S.erase(it);
        }
        cout << segment_tree::tree[1] << "\n";
    }
}
```

BOJ 19560. Village

최소 비용을 구하는 것부터 생각해 봅시다. 리프 정점은 인접한 정점이 하나밖에 없기 때문에 그 정점으로 이동하는 것 외에는 다른 선택지가 없습니다. 두 정점의 위치를 교환하면, 그 다음부터는 두 정점이 없는 트리라고 생각해도 무방합니다.

따라서 DFS를 하면서 밑에 있는 정점부터 서로 교환하는 방식으로 최솟값을 구할 수 있습니다. 루트 정점이 문제가 될 수 있는데, 만약 루트 정점이 이동을 하지 않았다면 인접한 정점을 아무거나 하나 잡아서 교환하면 됩니다.

최대 비용을 구하는 것은 최소 비용에 비해 조금 어렵습니다. 정점이 아닌 간선의 관점에서 문제를 바라보면, 각 간선의 사용 횟수를 최대화하는 문제가 됩니다.

트리는 간선을 하나 제거하면 두 개의 컴포넌트로 분할되고, 분리된 두 컴포넌트의 크기를 a, b라고 하면 해당 간선은 최대 $\min(a, b)$번 사용할 수 있습니다. 따라서 모든 간선에서 $\min(a, b)$를 더한 것이 정답의 상한입니다. 과연 이 상한을 진짜 달성할 수 있는지 확인해 봅시다.

간선을 $\min(a, b)$번 사용하기 위해서는 한쪽에 있는 모든 사람이 반대쪽으로 넘어가야 합니다. 트리에서 이런 배정을 하는 가장 간단한 방법은, 한 정점을 루트로 하는 rooted tree로 만든 다음에 모든 정점을 자신이 속해있지 않은 다른 서브 트리로 옮기는 것입니다. 만약 모든 서브 트리의 크기가 $N/2$ 이하이면 항상 원래 있던 곳과 다른 서브 트리로 이동시킬 수 있고, 모든 서브 트리의 크기를 $N/2$ 이하로 만드는 정점은 센트로이드라는 이름으로 잘 알려져 있습니다.

따라서 센트로이드를 구한 뒤, 모든 정점을 다른 서브 트리로 이동시키면 문제를 해결할 수 있습니다.

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

```cpp
using ll = long long;

int N, S[101010], A[101010], B[101010], D[101010], P[22][101010];
vector<int> G[101010];

int GetSize(int v, int b=-1){
    S[v] = 1;
    for(auto i : G[v]) if(i != b) D[i] = D[v] + 1, P[0][i] = v, S[v] +=
GetSize(i, v);
    return S[v];
}

int GetCent(int v, int sz, int b=-1){
    for(auto i : G[v]) if(i != b && S[i] * 2 > sz) return GetCent(i, sz, v);
    return v;
}

void Small(int v, int b=-1){
    for(auto i : G[v]) if(i != b) Small(i, v);
    if(v == A[v] && b != -1) swap(A[v], A[b]);
    if(v == A[v] && b == -1) swap(A[v], A[G[v][0]]);
}

vector<int> Large(int rt, int st){
    vector<int> res;
    function<void(int,int)> dfs = [&](int v, int b){
        res.push_back(v);
        for(auto i : G[v]) if(i != b) dfs(i, v);
    };
    dfs(st, rt);
    return res;
}

int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    for(int i=0, k=D[u]-D[v]; k; i++, k>>=1) if(k & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; ~i; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1,u,v; i<N; i++) cin >> u >> v, G[u].push_back(v),
G[v].push_back(u);

    iota(A+1, A+N+1, 1);
    Small(1);

    int C = GetCent(1, GetSize(1));
    vector<int> V{C};
    for(auto i : G[C]){
        auto now = Large(C, i);
        V.insert(V.end(), now.begin(), now.end());
    }
    for(int i=0, j=N/2; i<N; i++, j=(j+1)%N) B[V[i]] = V[j];
    for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];
```

```
    ll X = 0, Y = 0;
    for(int i=1; i<=N; i++) X += D[i] + D[A[i]] - 2*D[LCA(i,A[i])];
    for(int i=1; i<=N; i++) Y += D[i] + D[B[i]] - 2*D[LCA(i,B[i])];

    cout << X << " " << Y << "\n";
    for(int i=1; i<=N; i++) cout << A[i] << " \n"[i==N];
    for(int i=1; i<=N; i++) cout << B[i] << " \n"[i==N];
}
```

BOJ 21086. Smol Vertex Cover

최대 매칭의 크기를 M, 최소 정점 덮개의 크기를 C라고 합시다. $M \leq C$인 것은 자명하기 때문에 $C = M$인 경우와 $C = M + 1$인 경우로 나눠서 생각하면 됩니다. 일단 Edmond's blossom algorithm을 이용해 $O(N^3)$ 시간에 최대 매칭을 구합시다.

만약 $C = M$이라면 각 매칭의 끝점 중 정확히 한 정점이 정점 덮개에 포함되어야 합니다. 또한 정점 덮개의 조건을 만족하기 위해서 각 간선의 끝점 중 하나 이상이 정점 덮개에 포함되어야 합니다. 두 조건 모두 2-SAT으로 모델링할 수 있으므로 $O(N^2)$ 시간에 판별할 수 있습니다.

이제 $C = M + 1$인 경우를 생각해 봅시다. 매칭마다 정점을 하나씩 선택한 다음 추가로 하나를 더 선택해야 합니다. 구체적으로, 매칭에 포함되지 않은 정점을 선택하거나, 한 매칭의 양쪽 끝점을 모두 선택해야 합니다.

매칭에 포함되지 않은 정점을 선택하는 것은 $N - 2M$가지의 경우를 확인하면 되고, 한 매칭의 양쪽 끝점을 모두 선택하는 것은 M가지 경우를 확인하면 됩니다. 두 케이스 모두 2-SAT으로 모델링할 수 있으므로 $O(N^3)$ 시간에 판별할 수 있습니다.

따라서 전체 문제를 $O(N^3)$ 시간에 해결할 수 있습니다.

```
vector<int> Check(int n, const vector<pair<int,int>> &edge, const
vector<pair<int,int>> &match,
                  int use_vertex, int use_match){
    // vertex cover condition
    two_sat::clear();
    for(const auto &[a,b] : edge) two_sat::addEdge(a+n, b),
two_sat::addEdge(b+n, a);

    // matching vertex
    vector<int> use(n+1);
    for(int i=0; i<match.size(); i++){
        const auto &[a,b] = match[i];
        if(i == use_match) two_sat::addEdge(a+n, a), two_sat::addEdge(b+n, b);
        else two_sat::addEdge(a, b+n), two_sat::addEdge(b, a+n);
        use[a] = use[b] = 1;
    }

    // not matching vertex
    for(int i=1; i<=n; i++) if(!use[i] && i != use_vertex) two_sat::addEdge(i,
i+n);
    if(use_vertex != -1) two_sat::addEdge(use_vertex+n, use_vertex);

    return two_sat::run(n);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
```

```
    int N, M; cin >> N >> M;
    if(M == 0){ cout << 0; return 0; }
    vector<pair<int,int>> E(M);
    for(auto &[a,b] : E) cin >> a >> b, blossom::addEdge(a, b);
    vector<pair<int,int>> Match = blossom::run(N);

    if(auto Cover = Check(N, E, Match, -1, -1); !Cover.empty()){
        cout << Cover.size() << "\n";
        for(auto c : Cover) cout << c << " ";
        return 0;
    }

    vector<int> U(N + 1);
    for(auto [a,b] : Match) U[a] = U[b] = 1;

    for(int i=1; i<=N; i++){
        if(U[i]) continue;
        if(auto Cover = Check(N, E, Match, i, -1); !Cover.empty()){
            cout << Cover.size() << "\n";
            for(auto c : Cover) cout << c << " ";
            return 0;
        }
    }

    for(int i=0; i<Match.size(); i++){
        if(auto Cover = Check(N, E, Match, -1, i); !Cover.empty()){
            cout << Cover.size() << "\n";
            for(auto c : Cover) cout << c << " ";
            return 0;
        }
    }

    cout << "not smol";
}
```