

#3. 여러가지 문제 유형

나정휘

<https://justicehui.github.io/>

목차

DAG에서의 DP

트리에서의 DP

구간에 대한 DP

확률/기댓값 DP

게임 이론 + DP

자료구조 + DP

그리디 + DP

선형 점화식의 빠른 계산

DAG에서의 DP

DAG에서의 DP

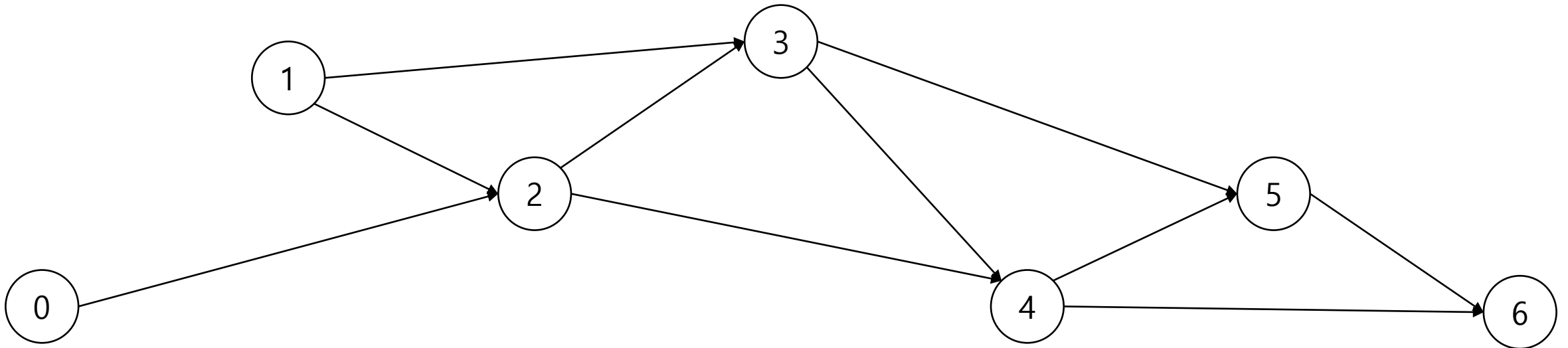
DAG와 DP의 관계

- DAG: 사이클이 없는 방향 그래프
 - 사이클이 없음
 - 위상 정렬을 할 수 있음
- DP: 작은 문제의 답을 이용해 큰 문제의 답을 구하는 방법
 - 작은 문제와 큰 문제 간의 연결 관계 → DAG
 - 부분 문제를 위상 정렬 순서대로 해결
 - 재귀 함수 + 메모이제이션

동적 계획법

DAG와 DP의 관계

- ex. 피보나치 수
 - 위상 정렬 상에서 앞선 문제의 답을 구해야 뒤에 있는 문제의 답을 구할 수 있음
 - 사이클 없음



동적 계획법

DAG와 DP의 관계

- 일반적인 그래프에서 풀지 못하는 문제를 DAG에서는 DP를 이용해 빠르게 해결할 수 있음
 - 최장 경로, S-T 경로의 개수
 - 일반적으로 다항 시간에 해결할 수 없지만 DAG에서는 $O(|V| + |E|)$
- 몇몇 DP 문제는 DAG로 생각해서 푸는 것이 편한 경우도 있음
 - 최장 공통 부분 문자열: DAG에서의 최장 경로
 - 동전 교환 경우의 수: DAG에서의 경로 개수
 - ...

DAG에서의 DP

BOJ 1005 ACM Craft

- DAG가 주어지면 정점 w 까지 가는 가장 긴 경로의 길이를 구하는 문제
- 단, 간선이 아닌 정점에 가중치가 붙어 있음
- 점화식
 - $D[x]$ = x 까지 가는 최장 경로의 길이
 - u 에서 v 로 가는 간선이 있다면 $D[v] \leftarrow D[u] + A[v]$
- 위상 정렬 순서대로 점화식을 계산하면 됨

DAG에서의 DP

```

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

int N, M, A[1010], C[1010], D[1010];
vector<int> G[1010];

void Solve(){
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1,s,e; i<=M; i++) cin >> s >> e, G[s].push_back(e), C[e]++;

    queue<int> Q;
    for(int i=1; i<=N; i++) D[i] = A[i];
    for(int i=1; i<=N; i++) if(!C[i]) Q.push(i);
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        for(auto i : G[v]){
            D[i] = max(D[i], D[v] + A[i]);
            if(--C[i]) Q.push(i);
        }
    }
    int T; cin >> T;
    cout << D[T] << "\n";
}

```

DAG에서의 DP

연습 문제

- BOJ 24888 노트 조각
- BOJ 16297 Eating Everything Efficiently

트리에서의 DP

트리에서의 DP

Tree DP

- Rooted Tree는 DAG
- 일반적인 구조
 - “ $D[v]$ = v 를 루트로 하는 서브 트리의 정답”으로 정의
 - 자식 정점들의 DP값을 잘 합치는 방법을 찾아야 함

트리에서의 DP

BOJ 15681 트리와 쿼리

- 루트가 있는 트리가 주어지면, 각 정점을 루트로 하는 서브 트리의 정점 개수를 구하는 문제
 - 각 정점의 (자손 정점 개수) + 1를 구하는 문제
- $D[v]$ = v 를 루트로 하는 서브 트리의 크기
 - v 의 자식 정점 $c_1, c_2, \dots, c_k$ 가 있다고 하면
 - $D[v] = D[c_1] + D[c_2] + \dots + D[c_k] + 1$
 - 리프 정점부터 크기를 구하면 됨

트리에서의 DP

```
● ● ●

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

int N, R, Q, S[101010];
vector<int> G[101010];

void DFS(int v, int b=-1){
    S[v] = 1;
    for(auto i : G[v]) if(i != b) DFS(i, v), S[v] += S[i];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> R >> Q;
    for(int i=1,u,v; i<N; i++) cin >> u >> v, G[u].push_back(v), G[v].push_back(u);
    DFS(R);
    for(int i=1,t; i<=Q; i++) cin >> t, cout << S[t] << "\n";
}
```

트리에서의 DP

BOJ 1949 우수 마을

- 정점마다 가중치가 있는 트리에서 정점을 몇 개 선택해야 함
- 이때 인접한 두 정점을 모두 선택할 수는 없음 (독립 집합)
- 선택한 정점들의 가중치의 합을 최대화
- 선형일 때의 문제를 먼저 풀어보자.
 - 배열 $A[]$ 에서 인접한 두 원소를 모두 선택하지 않으면서 선택한 원소의 합을 최대화
- $A[1..i-1]$ 만 고려했을 때의 정답을 알고 있을 때, $A[i]$ 를 추가했을 때의 정답을 구해야 함

트리에서의 DP

BOJ 1949 우수 마을

- 선형일 때의 문제를 먼저 풀어보자.
 - 배열 $A[]$ 에서 인접한 두 원소를 모두 선택하지 않으면서 선택한 원소의 합을 최대화
- 점화식
 - $A[1..i-1]$ 만 고려했을 때의 정답을 알고 있을 때, $A[i]$ 를 추가했을 때의 정답을 구해야 함
 - 즉, 점화식의 상태에서 마지막 원소의 선택 여부도 함께 저장해야 함
 - $D[i][0] = A[1..i]$ 만 고려했을 때 $A[i]$ 를 포함하지 않는 상황에서의 최대 점수
 - $D[i][1] = A[1..i]$ 만 고려했을 때 $A[i]$ 를 포함하는 상황에서의 최대 점수
 - $D[i][0] = \max(D[i-1][0], D[i-1][1])$
 - $D[i][1] = D[i-1][0] + A[i]$

트리에서의 DP

BOJ 1949 우수 마을

- 트리에서도 동일한 방법으로 해결 가능
- 점화식
 - $D[v][0]$ = v 를 루트로 하는 서브 트리에서 v 를 선택하지 않았을 때의 최대 점수
 - $D[v][1]$ = v 를 루트로 하는 서브 트리에서 v 를 선택했을 때의 최대 점수
 - $D[v][0] = \max(D[c_1][0], D[c_1][1]) + \max(D[c_2][0], D[c_2][1]) + \max(D[c_3][0], D[c_3][1]) + \dots$
 - $D[v][1] = A[v] + D[c_1][0] + D[c_2][0] + D[c_3][0] + \dots$

트리에서의 DP

```
● ● ●

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

int N, A[10101], D[10101][2];
vector<int> G[10101];

void DFS(int v, int b=-1){
    D[v][1] += A[v];
    for(auto i : G[v]){
        if(i == b) continue;
        DFS(i, v);
        D[v][0] += max(D[i][0], D[i][1]);
        D[v][1] += D[i][0];
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1,u,v; i<N; i++) cin >> u >> v, G[u].push_back(v), G[v].push_back(u);
    DFS(1);
    cout << max(D[1][0], D[1][1]);
}
```

트리에서의 DP

연습 문제

- BOJ 2213 트리의 독립집합
- BOJ 23089 사탕나무

구간에 대한 DP

구간에 대한 DP

BOJ 11049 행렬 곱셈 순서

- N개의 행렬이 주어짐
- 행렬의 순서를 바꾸지 않고 N개의 행렬을 모두 곱하는데 필요한 곱셈의 최소 횟수
- $A*B$ 행렬과 $B*C$ 행렬을 곱하면 $A*B*C$ 번의 곱셈이 필요하고, 그 결과는 $A*C$ 크기의 행렬
- 예시
 - 3개의 행렬 (5, 3), (3, 2), (2, 6)이 주어졌을 때
 - 앞에 있는 2개를 먼저 곱하면 $5*3*2 + 5*2*6 = 90$ 번
 - 뒤에 있는 2개를 먼저 곱하면 $3*2*6 + 5*3*6 = 126$ 번

구간에 대한 DP

BOJ 11049 행렬 곱셈 순서

- 이 문제는 최적 부분 구조가 성립함
 - 첫 번째부터 N번째 행렬을 최소 비용으로 곱하는 것은
 - 적당한 i 에 대해, $1 \sim i$ 번째 행렬을 곱한 행렬과 $i+1 \sim N$ 번째 행렬을 곱한 행렬을 곱하는 것과 동일
 - 따라서 $D[1][N] = \min\{ D[1][i] + D[i+1][N] + \text{곱셈 횟수} \}$
- $D[i][j] = i$ 번째 행렬부터 j 번째 행렬을 곱하는데 필요한 곱셈 횟수
 - $D[i][j] = \min\{ D[i][k] + D[k+1][j] + R[i]*C[k]*C[j] \}$
 - $D[i][j]$ 를 계산하기 위해 필요한 부분 문제는 항상 곱해야 하는 행렬이 더 적음
 - 따라서 $j - i$ 가 작은 부분 문제부터 계산

구간에 대한 DP

BOJ 11049 행렬 곱셈 순서

- 시간 복잡도
 - 부분 문제의 개수: $O(N^2)$
 - 각 부분 문제의 답을 구하기 위해 필요한 시간: $O(N)$
 - 따라서 전체 시간 복잡도는 $O(N^3)$
- $T(N) = \sum_{i=1}^N \sum_{j=i}^N (j - i) = \sum_{d=0}^N d(N - d) = N \sum d - \sum d^2$
- $= N^2(N + 1)/2 - N(N + 1)(2N + 1)/6$
- $\lim_{n \rightarrow \infty} \frac{T(N)}{N^3} = \frac{1}{2} - \frac{2}{6} = \frac{1}{6}$
- $\therefore T(N) \in \Theta(N^3)$

구간에 대한 DP

```
● ● ●

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

ll N, R[555], C[555], D[555][555];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> R[i] >> C[i];
    memset(D, 0x3f, sizeof D);
    for(int i=1; i<=N; i++) D[i][i] = 0;
    for(int i=2; i<=N; i++) D[i-1][i] = R[i-1] * R[i] * C[i];
    for(int d=2; d<=N; d++){
        for(int i=1; i+d-1<=N; i++){
            int j = i + d - 1;
            for(int k=i; k<j; k++){
                D[i][j] = min(D[i][j], D[i][k] + D[k+1][j] + R[i] * C[k] * C[j]);
            }
        }
    }
    cout << D[1][N];
}
```

연습 문제

BOJ 11066 파일 합치기

BOJ 12013 248 게임

BOJ 2449 전구

확률/기댓값 DP

확률/기댓값 DP

BOJ 15488 나이트가 체스판을 벗어나지 않을 확률

- $N \times N$ 크기의 체스판 위에 나이트가 있음
- 나이트는 체스판 밖으로 나갈 수 있지만, 한 번 나가면 다시 들어올 수 없음
- 나이트는 매번 이동할 수 있는 8가지 방법 중 균등한 확률로 한 가지를 골라 이동
- 나이트가 K 번 이동한 후에 체스판 위에 있을 확률
- 점화식
 - $D[k][x][y]$ = k 번 점프해서 (x, y) 에 도달할 확률
 - 시작 지점이 (i, j) 이면 $D[0][i][j] = 1$, 다른 모든 $D[0][*][*] = 0$
 - $D[k][x][y]/8 \rightarrow D[k+1][x+dx][y+dy]$

확률/기댓값 DP

```
#include <bits/stdc++.h>
using namespace std;
constexpr int di[] = {-2, -2, -1, 1, 2, 2, 1, -1};
constexpr int dj[] = {-1, 1, 2, 2, 1, -1, -2, -2};

int N, X, Y, K;
double D[55][55][55];
bool Bound(int i, int j){ return 1 <= i && i <= N && 1 <= j && j <= N; }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> X >> Y >> K;
    D[0][X][Y] = 1;
    for(int k=1; k<=K; k++){
        for(int i=1; i<=N; i++){
            for(int j=1; j<=N; j++){
                for(int d=0; d<8; d++){
                    int r = i + di[d], c = j + dj[d];
                    if(Bound(r, c)) D[k][r][c] += D[k-1][i][j] / 8;
                }
            }
        }
    }
    double R = 0;
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) R += D[K][i][j];
    cout << fixed << setprecision(20) << R;
}
```

연습 문제

BOJ 1344 축구

BOJ 20938 반짝반짝

게임 이론 + DP

게임 이론 + DP

BOJ 9655 돌 게임

- 테이블에 N개의 돌이 있고, 두 명의 플레이어가 번갈아 가면서 돌을 가져감
- 각 플레이어는 매번 돌을 1개 또는 3개 가져갈 수 있음
- 마지막 돌을 가져가는 사람이 승리
- 두 사람이 최선을 다해 플레이했을 때 이기는 사람을 구하는 문제

게임 이론 + DP

BOJ 9655 돌 게임

- 점화식
 - $D[i]$ = 돌이 i 개 있는 게임에서 먼저 돌을 가져가는 플레이어가 이기면 1, 지면 0
 - 만약 상대방을 $D[j] = 0$ 인 j 로 보내는 방법이 있다면 $D[i] = 1$
 - 상대방을 $D[j] = 0$ 으로 보낼 수 없다면 상대방은 $D[j] = 1$ 인 j 에서 시작하게 됨 $\rightarrow D[i] = 0$
- 시간 복잡도
 - 돌을 가져가는 방법의 수가 K 가지일 때 $O(NK)$

게임 이론 + DP



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

int N, D[1010] = {0, 1, 0, 1};

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    for(int i=4; i<1010; i++) D[i] = D[i-1] && D[i-3] ? 0 : 1;
    cin >> N;
    cout << (D[N] ? "SK" : "CY");
}
```

게임 이론 + DP

BOJ 11062 카드 게임

- 테이블에 N개의 카드가 일렬로 놓여 있고, 카드에는 점수가 적혀 있음
- 두 명의 플레이어가 서로 번갈아 가면서 가장 왼쪽이나 가장 오른쪽에 있는 카드를 가져감
- 모든 카드를 가져갈 때까지 게임을 진행하고, 가져간 카드에 적힌 수의 합이 더 큰 사람이 승리
- 두 플레이어는 서로 자신의 점수를 최대화하기 위해 최선을 다함
- 카드를 먼저 가져가는 사람의 점수를 출력

게임 이론 + DP

BOJ 11062 카드 게임

- 두 플레이어의 전략
 - 두 플레이어의 점수의 합은 일정함
 - 따라서 두 번째 플레이어는 첫 번째 플레이어의 점수를 최소화한다고 생각해도 됨
- 점화식
 - $D[s][e][f]$ = s..e번째 카드만 있는 게임에서 f번째 플레이어부터 시작했을 때 첫 번째 플레이어가 얻게 되는 점수
 - $f = 0$ 일 때는 결과를 최대화, $f = 1$ 일 때는 결과를 최소화하는 것이 목표

게임 이론 + DP

BOJ 11062 카드 게임

- 점화식

- $D[s][e][f]$ = s..e번째 카드만 있는 게임에서 f번째 플레이어부터 시작했을 때 첫 번째 플레이어가 얻게 되는 점수
- $f = 0$ 일 때는 결과를 최대화, $f = 1$ 일 때는 결과를 최소화하는 것이 목표
- $s > e$ 이면 $D[s][e][0] = D[s][e][1] = 0$
- $D[s][e][0] = \max\{ D[s+1][e][1] + A[s], D[s][e-1][1] + A[e] \}$
- $D[s][e][1] = \min\{ D[s+1][e][0], D[s][e-1][0] \}$
- $N - (e-s+1)$ 이 짝수면 $f = 0$, 홀수면 $f = 1$
- 따라서 $D[s][e]$ 만 있어도 됨

게임 이론 + DP



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

int N, A[1010], D[1010][1010];

int Solve(int s, int e){
    if(s > e) return 0;
    int &res = D[s][e]; if(res != -1) return res;
    int flag = (N - (e - s + 1)) % 2;
    if(flag == 0) return res = max(Solve(s+1, e) + A[s], Solve(s, e-1) + A[e]);
    else return res = min(Solve(s+1, e), Solve(s, e-1));
}

void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    memset(D, -1, sizeof D);
    cout << Solve(1, N) << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int TC; cin >> TC;
    for(int tc=1; tc<=TC; tc++) Solve();
}
```

연습 문제

BOJ 25949 Jar Game

자료구조 + DP

자료구조 + DP

BOJ 12015 가장 긴 증가하는 부분 수열 2

- LIS의 길이를 $O(N \log N)$ 에 구하는 문제
- $O(N^2)$ 은 쉬움
 - $D[i]$ = i 번째 원소로 끝나는 가장 긴 증가 부분 수열의 길이
 - $D[i] = \max_{j < i, A[j] < A[i]} (D[j] + 1)$
 - 이 방법은 더 최적화하기 어려움
- 점화식의 정의를 조금 바꿔보자.
 - $D[i][j]$ = 1.. i 번째 원소만 고려했을 때, 값이 j 인 원소로 끝나는 가장 긴 증가 부분 수열의 길이
 - $j = A[i]$ 이면 $D[i][j] = \max_{k < j} (D[i-1][k] + 1)$
 - $j \neq A[i]$ 이면 $D[i][j] = D[i-1][j]$
 - 수의 범위를 X 라고 하면 시간 복잡도는 $O(NX)$

자료구조 + DP

BOJ 12015 가장 긴 증가하는 부분 수열 2

- 점화식

- $D[i][j] = 1..i$ 번째 원소만 고려했을 때, 값이 j 인 원소로 끝나는 가장 긴 증가 부분 수열의 길이
- $j = A[i]$ 이면 $D[i][j] = \max_{k < j} (D[i-1][k] + 1)$
- $j \neq A[i]$ 이면 $D[i][j] = D[i-1][j]$
- 매번 더 큰 값으로 덮어쓰기 때문에 첫 번째 인덱스를 들고 있을 필요가 없음
 - 배낭 문제에서 차원 하나 없애는 방법과 동일
- $D[j] =$ 값이 j 인 원소로 끝나는 가장 긴 증가 부분 수열의 길이
- $A[i]$ 가 주어지면 $D[j] = \max_{k < j} (D[k] + 1)$ 로 갱신
- 구간의 최댓값을 구하는 작업이므로 세그먼트 트리로 계산 가능
- $O(N \log X)$

자료구조 + DP



```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 20;

int N, A[1010101], D[1010101], T[SZ<<1];

void Update(int x, int v){
    x |= SZ; T[x] = max(T[x], v);
    while(x >= 1) T[x] = max(T[x<<1], T[x<<1|1]);
}

int Query(int l, int r){
    l |= SZ; r |= SZ; int res = 0;
    while(l <= r){
        if(l & 1) res = max(res, T[l++]);
        if(~r & 1) res = max(res, T[r--]);
        l >>= 1; r >>= 1;
    }
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1; i<=N; i++) D[i] = Query(0, A[i]-1) + 1, Update(A[i], D[i]);
    cout << *max_element(D, D+N+1); // cout << T[1];
}
```

연습 문제

BOJ 13555 증가하는 부분 수열

BOJ 5466 상인

그리디 + DP

그리디 + DP

BOJ 13448 SW 역량 테스트

- T분 동안 진행되는 코딩 테스트에 N개의 문제가 출제됨
- i번 문제를 푸는데 R_i 만큼의 시간이 걸리고, t분에 맞았다면 $M_i - tP_i$ 점을 받게 됨
- 받을 수 있는 최대 점수를 구하는 문제
- 조금 더 쉬운 문제를 먼저 풀어보자.
 - 모든 문제를 해결해야 할 때 받을 수 있는 최대 점수
 - 늦게 풀수록 손해이므로 중간에 시간을 버릴 필요는 없음
 - 따라서 문제를 적당한 순서대로 정렬하고 순서대로 풀면 됨
 - 적당한 순서대로 정렬하는 방법은?

그리디 + DP

BOJ 13448 SW 역량 테스트

- 조금 더 쉬운 문제를 먼저 풀어보자.
 - 문제를 적당한 순서대로 정렬하고 순서대로 풀면 됨
 - 적당한 순서대로 정렬하는 방법은?
- exchange argument
 - 두 원소의 순서를 결정할 수 있고 그 순서 관계가 total ordering 형태이면 비교 함수로 정렬 가능
 - 따라서 두 원소 중 어떤 것이 앞에 오는지 결정하는 방법만 알면 됨
 - 그리디 기법 강의 참고

그리디 + DP

BOJ 13448 SW 역량 테스트

- exchange argument
 - 두 원소 중 어떤 것이 앞에 오는지 결정하는 방법만 알면 됨
- a를 먼저 풀고 b를 풀었을 때 얻는 점수
 - $M[a] - (x + R[a]) * P[a] + M[b] - (x + R[a] + y + R[b]) * P[b]$
- b를 먼저 풀고 a를 풀었을 때 얻는 점수
 - $M[b] - (x + R[b]) * P[b] + M[a] - (x + R[b] + y + R[a]) * P[a]$
- a가 b보다 앞에 있을 조건
 - $R[a] / P[a] < R[b] / P[b]$
 - $R[i] / P[i]$ 오름차순으로 정렬하면 됨

그리디 + DP

BOJ 13448 SW 역량 테스트

- 모두 해결할 필요는 없고 몇 개만 선택해도 된다면?
 - $R[a] / P[a] < R[b] / P[b]$ 이면 a와 b를 모두 해결할 때 a를 먼저 해결하는 경우만 생각해도 됨
 - 따라서 $R[i] / P[i]$ 오름차순으로 정렬하고, 그 순서대로 DP를 계산하면 됨
- 점화식
 - $D[i][j] = 1..i$ 번째 문제 중 몇 개를 선택해서 j분 동안 해결했을 때의 최대 점수
 - $D[i][j] = \max_{0 \leq k < i} \{ D[k][j-R[i]] + \text{score}(i, j) \}$

연습 문제

BOJ 26142 꺾이지 않는 마음 1

BOJ 24457 카페인 중독

선형 점화식의 빠른 계산

선형 점화식의 빠른 계산

선형 점화식

- 상수 계수를 가진 몇 개의 항의 합으로 구성된 점화식
 - 이전 몇 개의 항의 선형 결합으로 구성된 점화식
 - $a_n = \sum_{i=1}^k c_i a_{n-i} = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$
 - 피보나치 수열은 $k = 2, c_1 = 1, c_2 = 1$ 인 선형 점화식
- 선형 점화식의 n 번째 항을 계산하는 방법
 - $O(nk)$ - 단순 반복문
 - $O(k^3 \log n)$ - 행렬의 거듭제곱 이용
 - $O(k^2 \log n)$ - 특성다항식과 다항식의 나눗셈 이용 (키타마사법)
 - $O(k \log k \log n)$ - 키타마사법 + FFT
 - n 이 많이 커도 k 가 적당히 작으면 빠르게 계산할 수 있음

선형 점화식의 빠른 계산

행렬과 선형 점화식의 관계

- $a_n = \sum_{i=1}^k c_i a_{n-i} = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$
- 목표
 - 아래 수식을 만족하는 $k \times k$ 행렬 X 를 찾는 것

$$X \times \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_{n-k+1} \end{bmatrix}$$

- 행렬의 곱셈은 결합 법칙이 성립하므로 a_n 을 구하는 것은 $X^{n-k}[a_k; a_{k-1}; \cdots; a_1]$ 을 구하면 됨
- X^{n-k} 를 계산하는 것은 $O(\log n)$ 번의 행렬 곱셈으로 가능
- 따라서 초항 k 개만 알고 있다면 $O(k^3 \log n)$ 시간에 n 번째 항 계산 가능

선형 점화식의 빠른 계산

행렬을 만들어 보자.

$$\begin{bmatrix} ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & & ? & ? \\ ? & ? & ? & \cdots & ? & ? \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & \cdots & ? & ? \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

행렬을 만들어 보자.

- $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ 를 만드는 방법

$$\begin{bmatrix} ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & & ? & ? \\ ? & ? & ? & \cdots & ? & ? \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & \cdots & ? & ? \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

행렬을 만들어 보자.

- $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$ 를 만드는 방법: 점화식을 그대로 넣으면 됨

$$\begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{k-1} & c_k \\ ? & ? & ? & & ? & ? \\ ? & ? & ? & \cdots & ? & ? \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & \cdots & ? & ? \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

행렬을 만들어 보자.

- a_{n-1} 을 만드는 방법

$$\begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{k-1} & c_k \\ ? & ? & ? & & ? & ? \\ ? & ? & ? & \cdots & ? & ? \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & \cdots & ? & ? \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

행렬을 만들어 보자.

- a_{n-1} 을 만드는 방법: 이미 a_{n-1} 을 알고 있으므로 그냥 가져오면 됨

$$\begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{k-1} & c_k \\ 1 & 0 & 0 & & 0 & 0 \\ ? & ? & ? & \cdots & ? & ? \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ ? & ? & ? & \cdots & ? & ? \\ ? & ? & ? & \cdots & ? & ? \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

행렬을 만들어 보자.

- 나머지도 마찬가지로 그냥 가져오면 됨

$$\begin{bmatrix} c_1 & c_2 & c_3 & \cdots & c_{k-1} & c_k \\ 1 & 0 & 0 & & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \\ \vdots \\ a_{n-k+1} \\ a_{n-k} \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ \vdots \\ a_{n-k+2} \\ a_{n-k+1} \end{bmatrix}$$

선형 점화식의 빠른 계산

BOJ 11444 피보나치 수 6

- $n(\leq 10^{18})$ 번째 피보나치 수를 구하는 문제

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} f_1 = 1 \\ f_0 = 0 \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix}$$

선형 점화식의 빠른 계산

```

#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr ll MOD = 1e9+7;

struct Matrix{
    int n; vector<vector<ll>> a;
    Matrix(int n, int i=0) : n(n), a(n, vector<ll>(n)) {
        for(int k=0; k<n; k++) a[k][k] = i;
    }
    vector<ll>& operator [] (int i) { return a[i]; }
    const vector<ll>& operator [] (int i) const { return a[i]; }
    Matrix operator * (const Matrix &b) const {
        Matrix c(n); assert(n == b.n);
        for(int i=0; i<n; i++) for(int j=0; j<n; j++) for(int k=0; k<n; k++)
            c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % MOD;
        return c;
    }
};

Matrix Pow(Matrix a, ll b){
    Matrix res(a.n, 1);
    for(; b >= 1; a = a * a) if(b & 1) res = res * a;
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll N; cin >> N;
    if(N < 2){ cout << N; return 0; }
    Matrix M(2);
    M[0][0] = 1; M[0][1] = 1;
    M[1][0] = 1; M[1][1] = 0;
    M = Pow(M, N-1);
    cout << M[0][0];
}
```

연습 문제

BOJ 16467 병아리의 변신은 무죄

BOJ 14289 본대 산책 3