#10-2. 동적 계획법

나정휘

https://justiceHui.github.io/

동적 계획법

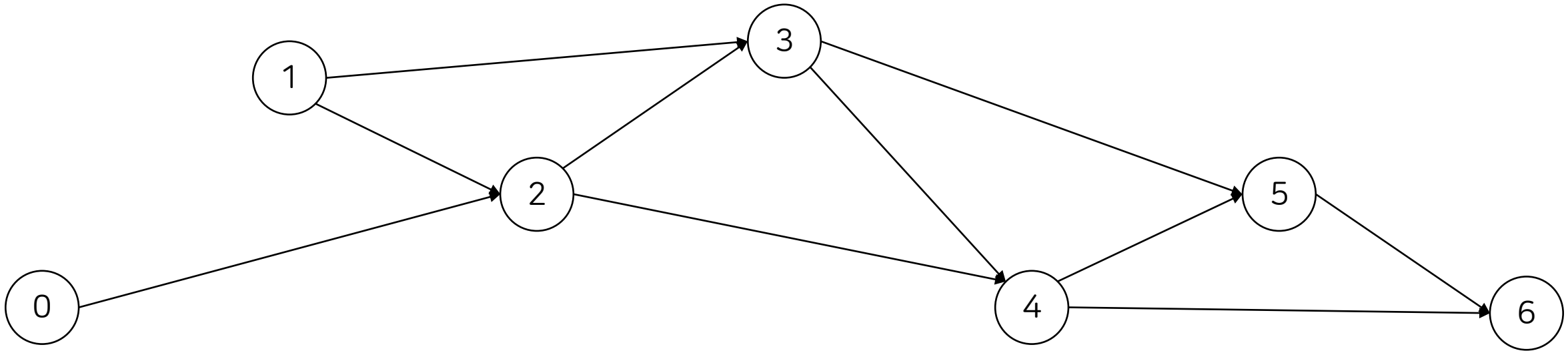
DAG와 DP의 관계

- DAG: 사이클이 없는 방향 그래프
  - 사이클 없음
  - 위상 정렬 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

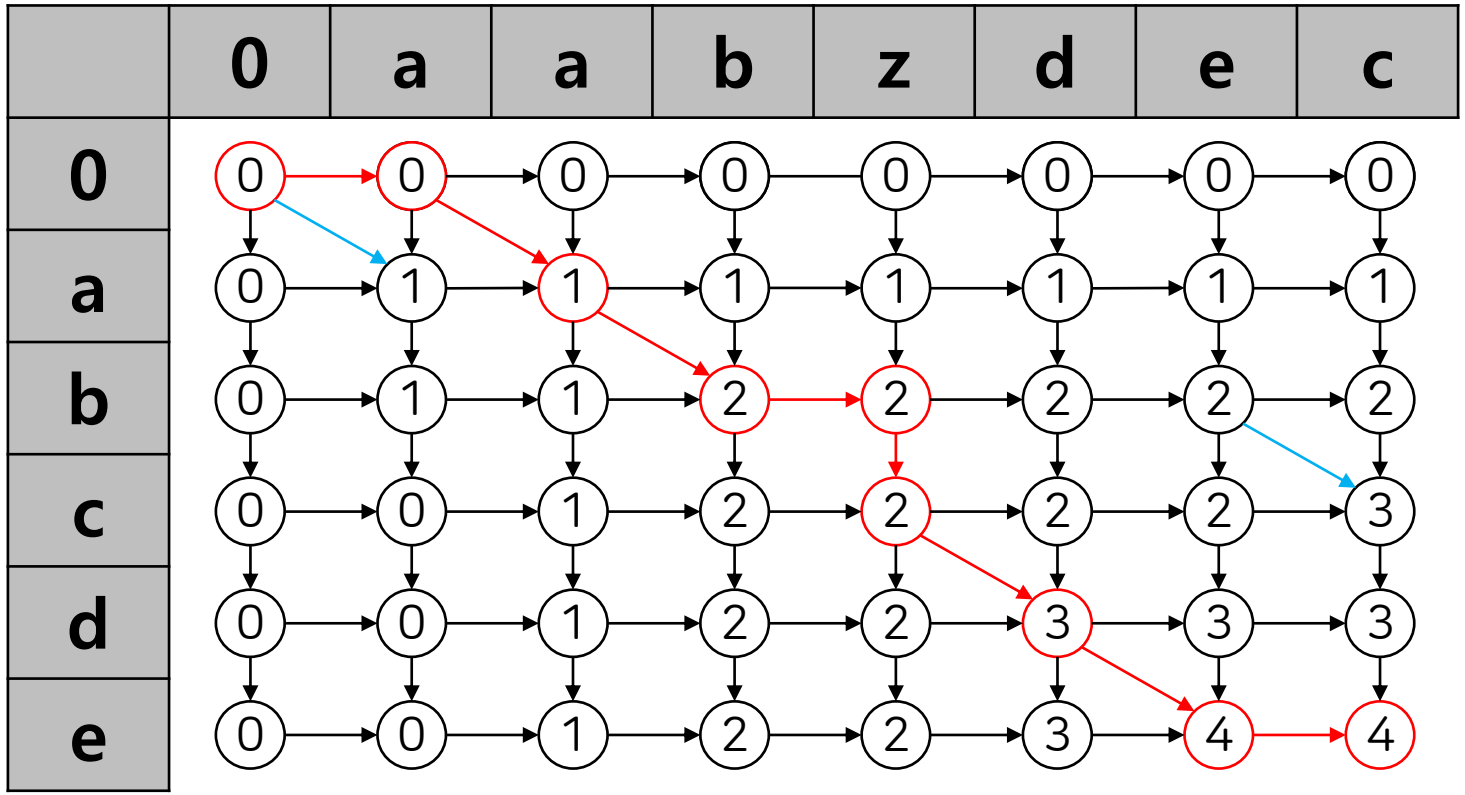
- ex. 피보나치 수
  - 위상 정렬 상에서 앞선 문제의 답을 구해야 뒤에 있는 문제의 답을 구할 수 있음
  - 사이클 없음



동적 계획법

DAG와 DP의 관계

- ex. LCS
  - (0, 0)에서 (N, M)으로 가는 최장 경로를 구하는 문제
  - 이 방식을 응용해서 분할 정복하는 게 Hirschberg's algorithm



동적 계획법

DAG와 DP의 관계

- ex. 동전 교환 경우의 수
  - N가지 종류의 동전으로 K원 만드는 방법
  - (0, 0)에서 (N, K)로 가는 경로의 개수를 구하는 문제

- ex. 최소 개수 동전 교환
  - N가지 종류의 동전을 최소한으로 사용해 K원 만드는 방법
  - (0, 0)에서 (N, K)로 가는 최단 거리를 구하는 문제

- 일반적인 그래프에서 풀지 못하는 문제를 DAG에서는 DP를 이용해 빠르게 해결할 수 있음
  - 최장 경로, S-T 경로의 개수 등
  - 일반적으로 다항 시간에 해결할 수 없지만 DAG에서는 O(|V| + |E|)

동적 계획법

BOJ 14567 선수과목

- N개의 과목과 M개의 선후수 과목 관계가 주어짐
- N개의 과목을 모두 듣기 위해 필요한 최소 학기를 구하는 문제

- in-degree = 0인 과목만 있으면 정답은 1
- in-degree ≠ 0인 과목으로 가는 간선의 시작점의 in-degree가 모두 0이면 정답은 2
- …

- 각 정점까지 가는 최장 경로를 구하는 문제
  - 경로의 시작점은 마음대로 정해도 됨

동적 계획법

BOJ 14567 선수과목

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

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e); In[e]++;
    }
    queue<int> Q;
    for(int i=1; i<=N; i++) if(!In[i]) Q.push(i), D[i] = 1;
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        for(auto i : G[v]){
            D[i] = max(D[i], D[v] + 1);
            if(!--In[i]) Q.push(i);
        }
    }
    for(int i=1; i<=N; i++) cout << D[i] << " ";
}
```

동적 계획법

BOJ 2637 장난감 조립

- 차원이 하나 더 늘어난 문제

BOJ 25953 템포럴 그래프

- 명시적으로 그래프를 구성하지 않고 푸는 문제

BOJ 18780 Timeline

- 그래프 모델링을 잘해야 하는 문제

질문?

동적 계획법

Tree에서의 DP

- Rooted Tree는 DAG

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

동적 계획법

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] = v를 루트로 하는 서브 트리의 크기
  - v의 자식 정점 c_1, c_2, … , c_k가 있다고 하면
  - $D[v] = D[c_1] + D[c_2] + … + D[c_k] + 1$
  - 리프 정점부터 크기를 구하면 됨

동적 계획법

```cpp
#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";
}
```

질문?

동적 계획법

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

동적 계획법

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]

동적 계획법

BOJ 1949 우수 마을

- 트리에서도 동일한 방법으로 해결 가능

- 점화식
  - D[v][0] = v를 루트로 하는 서브 트리에서 v를 선택하지 않았을 때의 최대 점수
  - D[v][1] = v를 루트로 하는 서브 트리에서 v를 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], C[c_3][1]) + ...
  - D[v][1] = A[v] + D[c_1][0] + D[c_2][0] + D[c_3][0] + ...

동적 계획법

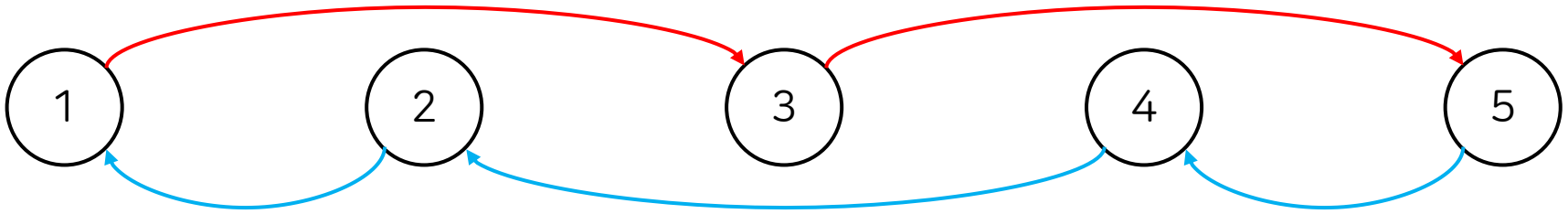
```cpp
#include <bits/stdc++.h>
using namespace std;

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]);
}
```

질문?

동적 계획법

BOJ 10272 현상금 사냥꾼 김정은

- 2차원 평면 위에 N개의 점이 주어짐
- 1번 점에서 출발해서 번호가 증가하는 순서대로 몇 개의 점을 거쳐 N번 점으로 이동한 뒤
- N번 점에서 출발해서 번호가 감소하는 순서대로 몇 개의 점을 거쳐 1번 점으로 돌아와야 함
- 각 점은 정확히 1번씩만 방문할 때, 이동 거리를 최소화하는 문제

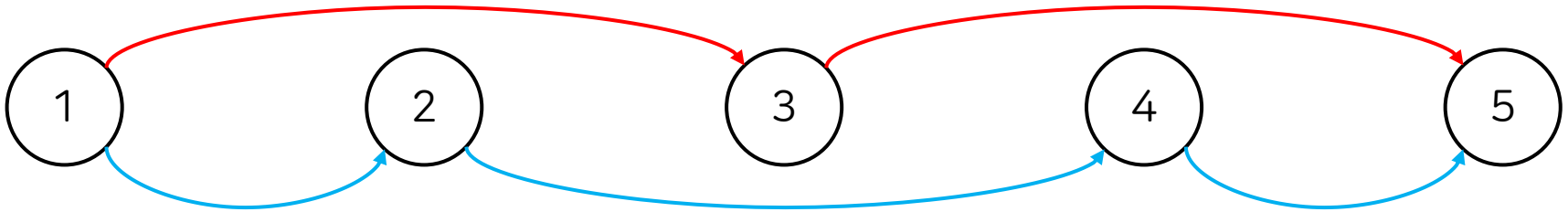


동적 계획법

BOJ 10272 현상금 사냥꾼 김정은

- 2차원 평면 위에 N개의 점이 주어짐
- 1번 점에서 출발해서 번호가 증가하는 순서대로 몇 개의 점을 거쳐 N번 점으로 이동한 뒤
- N번 점에서 출발해서 번호가 감소하는 순서대로 몇 개의 점을 거쳐 1번 점으로 돌아와야 함
- 각 점은 정확히 1번씩만 방문할 때, 이동 거리를 최소화하는 문제

- 돌아오는 경로를 뒤집으면 정점이 서로 겹치지 않도록 1 – N 경로를 2개 만드는 것이라고 생각할 수 있음
  - 왼발과 오른발이 모두 1번 점에 있는 상태에서 시작
  - N번 점을 제외한 다른 모든 점을 1번씩만 밟으면서 N 발자국 앞으로 나아가는 최소 비용



동적 계획법

BOJ 10272 현상금 사냥꾼 김정은

- 돌아오는 경로를 뒤집으면 정점이 서로 겹치지 않도록 1 – N 경로를 2개 만드는 것이라고 생각할 수 있음
  - 왼발과 오른발이 모두 1번 점에 있는 상태에서 시작
  - N번 점을 제외한 다른 모든 점을 1번씩만 밟으면서 N 발자국 앞으로 나아가는 최소 비용
  - 같은 발이 연속해서 2번 이상 이동할 수 있음에 주의

- D(i, j) = 왼발이 i번 점, 오른발이 j번 점에 있을 때, (N, N)까지 가는데 필요한 최소 비용
  - D(i, N) = Cost(i, N)
  - D(N, j) = Cost(j, N)
  - D(i, j) = min{ D(k, j) + Cost(i, k), D(i, k) + Cost(j, k) }
    - k = max(i, j) + 1

동적 계획법

BOJ 10272 현상금 사냥꾼 김정은

```
int N, X[555], Y[555];
double D[555][555];
double Dist(int i, int j){ return hypot(X[i]-X[j], Y[i]-Y[j]); }

double f(int i, int j){
    double &res = D[i][j];
    if(res >= 0) return res;
    if(i == N) return res = Dist(j, N);
    if(j == N) return res = Dist(i, N);
    int k = max(i, j) + 1;
    return res = min(f(k, j) + Dist(i, k), f(i, k) + Dist(j, k));
}

void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> X[i] >> Y[i];
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) D[i][j] = -1;
    cout << fixed << setprecision(20) << f(1, 1) << "\n";
}
```

질문?