ACM Study Sogang ACM-ICPC Team

Dynamic Programming

한상덕

INDEX

01

일반적인 동적 계획법

Dynamic Programming

복잡한 문제를 간단한 여러 개의 문제로 나누어 푸는 방법을 말한다.
이것은 부분 문제 반복과 최적 기본 구조를 가지고 있는 알고리즘을 일반적인 방법에 비해 더욱 적은 시간 내에 풀 때 사용한다.

기본적으로 수학적 귀납법과 크게 다르지 않다.

1. 정의 - f[i] = i번 째 피보나치 수
2. 초기화 - f[0] = 1
3. 수식 - f[i] = f[i-1] + f[i-2]

기본적으로 수학적 귀납법과 크게 다르지 않다.

1. 정의 - f[i] = i번 째 피보나치 수
2. 초기화 - f[0] = 1
3. 수식 - f[i] = f[i-1] + f[i-2]

그렇기 때문에 수식을 유도하는 방법은 초기값이 성립하는 것과

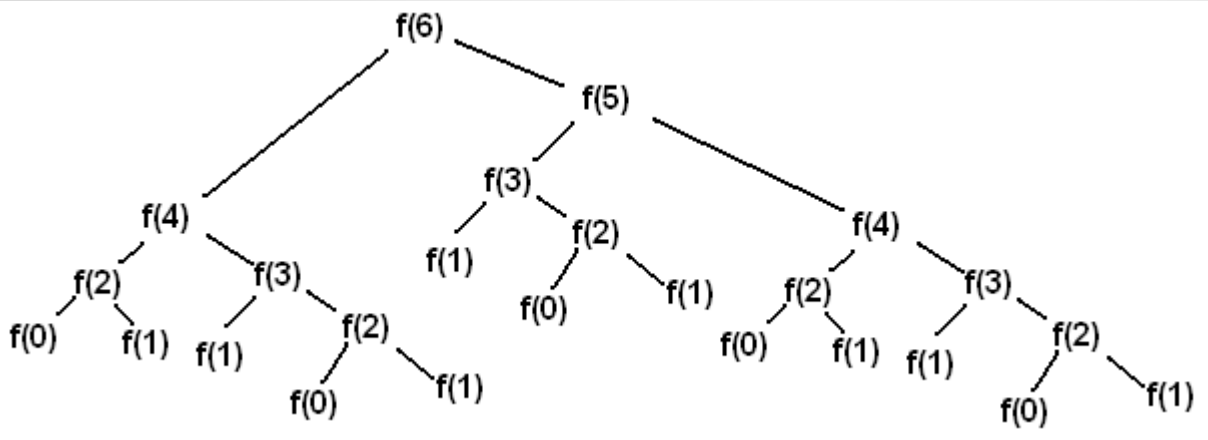
K번째 까지 성립할 때 K+1이 성립함을 초기화와 수식 단계에서

이루어지기 때문에 증명이 필요가 없음

피보나치 수

```
int fibo( int n ){
    if( n <= 1 ) return n;
    else return fibo( n-1 ) + fibo( n-2 );
}

int main(){
    int n;
    scanf("%d", &n);
    printf("%d₩n", fibo( n ) );
}
```

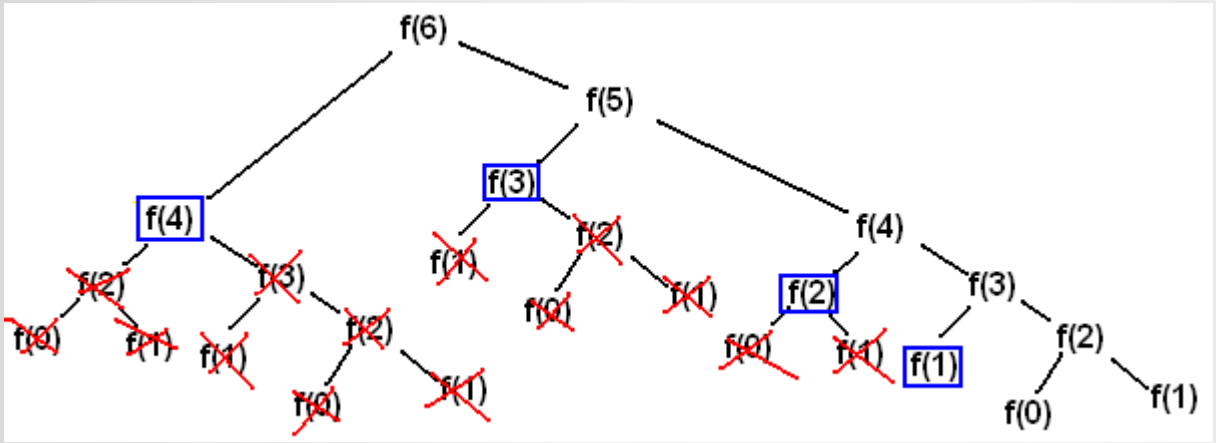


피보나치 수

```
int f[N];

int fibo( int n ){
    if( n <= 1 ) return n;
    else if( f[n] > 0 ) return f[n];
    else return f[n] = fibo( n-1 ) + fibo( n-2 );
}

int main(){
    int n;
    scanf("%d", &n);
    printf("%d₩n", fibo( n ) );
}
```



02

여러 종류의 접근법

Part 1 수식의 접근법

1. 이전 상태 -> 현재 상태를 유도
2. 현재 상태 -> 다음 상태를 유도

Example

동전 1

n가지 종류의 동전이 있다. 각각의 동전이 나타내는 가치는 다르다.
이 동전들을 적당히 사용해서, 그 가치의 합이 k원이 되도록 하고 싶다.
그 경우의 수를 구하시오. (각각의 동전은 몇 개라도 사용할 수 있다.)

제한

첫째줄에 n, k가 주어진다. ($1 \le n \le 100$, $1 \le k \le 10,000$)

다음 n개의 줄에는 각각의 동전의 가치가 주어진다.

Part 1 수식의 접근법

1. 이전 상태 -> 현재 상태를 유도

```
int main(){
    int n, k;
    scanf("%d %d", &n, &k);
    for(int i=0; i<n; i++) scanf("%d", &a[i]);
    d[0] = 1;
    for(int i=0; i<n; i++)
        for(int j=0; j<=k; j++)
            if( j-a[i] >= 0 )
                d[j] += d[j-a[i]];
    printf("%d", d[k]);
    return 0;
}
```

2. 현재 상태 -> 다음 상태를 유도

```
int main(){
    int n, k;
    scanf("%d %d", &n, &k);
    for(int i=0; i<n; i++) scanf("%d", &a[i]);

    d[0] = 1;
    for(int i=0; i<n; i++)
        for(int j=0; j<=k; j++)
            if( j+a[i] <= 10000 )
                d[j+a[i]] += d[j];

    printf("%d", d[k]);

    return 0;
}
```

어느 방법이 더 쉬울까?

상황에 따라 다름, 그렇기 때문에 두 가지 방법에 모두 익숙해지는 것이 중요

심지어 어느 한 쪽 방법으로만 풀리거나 시간복잡도가 줄어드는 경우가 있음

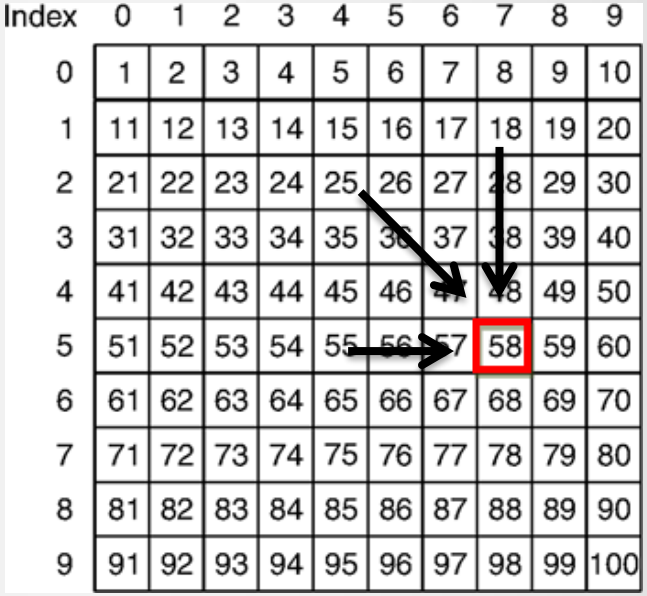
Part 2 수식의 다른 접근법

일반적인 수식

d[i] = d[i-1] + cost

d[i][j] = d[i-1][j-1] + cost

d[i] = d[i-k] + cost



Index	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

일반적으로 접근 순서가

옆 그림처럼 접근

하지만 꼭 저런 유형으로 수식을 정의할 필요는 없음

Ex) 대각선 수식

Part 2 수식의 다른 접근법 - Ex 대각선 수식

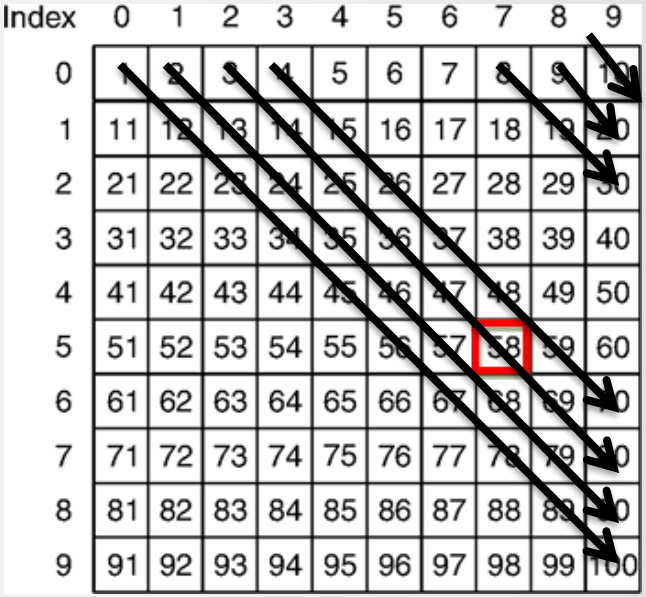
크기가 N×M인 행렬 A와 M×K인 B를 곱할 때 필요한 곱셈 연산의 수는 총 N×M×K번이다. 행렬 N개를 곱하는데 필요한 곱셈 연산의 수는 행렬을 곱하는 순서에 따라 달라지게 된다.

AB를 먼저 곱하고 C를 곱하는 경우 (AB)C에 필요한 곱셈 연산의 수는
5×3×2 + 5×2×6 = 30 + 60 = 90번이다.
BC를 먼저 곱하고 A를 곱하는 경우 A(BC)에 필요한 곱셈 연산의 수는
3×2×6 + 5×3×6 = 36 + 90 = 126번이다.

행렬 N개의 크기가 주어졌을 때, 모든 행렬을 곱하는데 필요한 곱셈 연산 횟수의 최소값을 구하는 프로그램을 작성하시오.
입력으로 주어진 행렬의 순서를 바꾸면 안 된다.

Part 2 수식의 다른 접근법

다음과 같이 대각선 순서로
접근을 할 수 있다.



1. 정의 d[i][j] = i번째 행렬부터 j번째 행렬까지 곱한 행렬의 최소곱셈
2. 초기화 d[i][i] = 0
3. 수식 d[i][j] = min(d[i][j], d[i][j-k] + d[j+1-k][j] + r[i]*c[j-k]*c[j])

Part 3 상태 공간에 필요한 모든 정보가 있을 필요는 없다.

- 일정 상태 정보를 이용하여 저장되어 있지 않은 상태를 유도할 수 있다.
- 이러한 방법을 이용하여 시간복잡도를 줄일 수 있음

Ex 경찰차

두 경찰차가 2차원 상에 존재하고 사건이 (x1, y1), (x2, y2), .. ,(xn, yn) 지점에서 순차적으로 발생하고 두 경찰차 중 적어도 한 대가 각 사건마다 출동되어야 하고 사건의 순서에 따라 이동되어야 할 때 두 경찰차의 이동 거리의 최소 거리

사건의 개수 : N $(1 \leq N \leq 1000)$

Part 3 상태 공간에 필요한 모든 정보가 있을 필요는 없다.

1. 정의 d[i][j][k] = i번째 사건까지 발생했고 첫 번째 경찰차가 j번째 사건이 발생한 사건에 위치하고 두 번째 경찰차가 k번째 사건이 발생한 위치에 있는 상태

2. 초기화 d[i][첫 번째 경찰차가 있는 위치][두 번째 경찰차가 있는 위치] = 0

3. 수식 d[i][i][k] = min(d[i][i][k], d[i][j][k] + cost(j, i))
   d[i][j][i] = min(d[i][j][i], d[i][j][k] + cost(k, i))

시간복잡도 : O(N^3)

Part 3 상태 공간에 필요한 모든 정보가 있을 필요는 없다.

1. 정의 d[i][j][k] = i번째 사건까지 발생했고 첫 번째 경찰차가 j번째 사건이 발생한 사건에 위치하고 두 번째 경찰차가 k번째 사건이 발생한 위치에 있는 상태

2. 정의 d[j][k] = 첫 번째 경찰차가 j번째 사건이 발생한 사건에 위치하고 두 번째 경찰차가 k번째 사건이 발생한 위치에 있는 상태

생각을 해보면 j, k 둘 중 하나는 i-1번째 사건 지점을 방문해야 됨

그렇기 때문에 상태 공간이 N^2으로 줄어들음

Part 3 상태 공간에 필요한 모든 정보가 있을 필요는 없다.

1. 정의 d[j][k] = 첫 번째 경찰차가 j번째 사건이 발생한 사건에 위치하고
   두 번째 경찰차가 k번째 사건이 발생한 위치에 있는 상태

2. 초기화 d[첫 번째 경찰차가 있는 위치][두 번째 경찰차가 있는 위치] = 0
3. 수식 d[i][j] = d[i-1][j] + cost(i-1, i)
   d[i-1][i] = d[i-1][j] + cost(j, i)
   d[i][i-1] = d[j][i-1] + cost(j, i)
   d[j][i] = d[j][i-1] + cost(i-1, I)

시간복잡도 : O(N^2)

Part 4 전처리로 시간을 줄일 수 있다.

동적 계획법 문제를 풀면서 가장 쉽게 시간을 줄일 수 있는 방법

D[i][j] = d[prev_i][cur_i] + cost

Cost 부분의 계산을 미리 계산하여 O(1)이나 작은 복잡도로 구하는 방법

대표적인 예가 S[i][j] = i번 째 행에서 1~j까지 열의 합

⇒ i번 째 열의 j~k번째 행의 합 = S[i][k] - S[i][j-1]

Ex) http://codeforces.com/contest/611/problem/C

THE END

감 사 합 니 다