

Dynamic Programming I

이종서(leejseo)

강사 소개

- 요약: 학부생이자 직장인이자 대회 나가는 사람



이종서
(leejseo)

강사 소개

- KAIST 전산학부·수리과학과 휴학
- Software Engineer
 - (전) Moloco, Sendbird Software Engineer
 - (현) 차이코퍼레이션 근무 예정
- Competitive Programmer
 - ICPC Seoul Regional 10th Place
 - SCPC 5th Place Prize
 - Google Code Jam Round 3
 - Google Hash Code 15th Place



이종서
(leejseo)

다이나믹 프로그래밍

(1) 피보나치 수열의 재귀적 정의

$$f(x) = \begin{cases} 1, & x = 0, 1 \\ f(x-1) + f(x-2), & x > 1 \end{cases}$$

(2) 재귀 함수 구현

```
int f(int x) {  
    if (x <= 1) return 1;  
    return f(x-1) + f(x-2);  
}
```

다이나믹 프로그래밍

(1) 피보나치 수열의 재귀적 정의

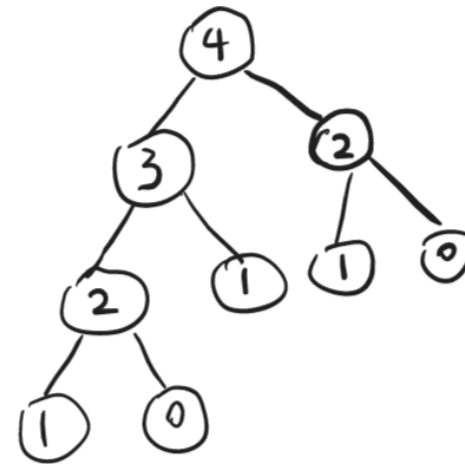
$$f(x) = \begin{cases} 1, & x = 0, 1 \\ f(x-1) + f(x-2), & x > 1 \end{cases}$$

(2) 재귀 함수 구현

```
int f(int x) {  
    if (x <= 1) return 1;  
    return f(x-1) + f(x-2);  
}
```

(3) 무엇이 문제일까?

$$T(n) = T(n-1) + T(n-2) + O(1)$$



다이나믹 프로그래밍

(4) 재귀적 구현의 문제점 극복

- 테이블을 만들어서 각 입력에 대한 답을 "메모"

```
int memo[MAXN];
int f(int x) {
    if (x <= 1) return 1;
    if (memo[x] != -1) return memo[x];
    return memo[x] = f(x-1) + f(x-2);
}
```

(5) 다이나믹 프로그래밍

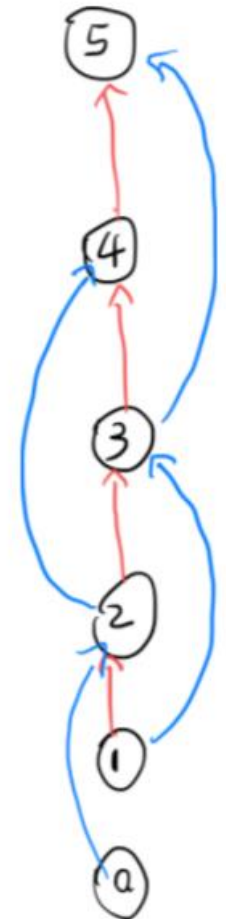
- (4)에서 테이블을 채우는 순서대로 루프를 이용해서 채움

```
int memo[MAXN];
memo[0] = memo[1] = 1;
for (int i=2; i<=x; i++) {
    memo[i] = memo[i-1]+memo[i-2];
}
```

다이나믹 프로그래밍

(6) 다이나믹 프로그래밍과 DAG

- 재귀함수의 입력 인자를 정점, 점화식에서의 참조 관계를 간선으로 표현하여 그래프를 만들면 DAG가 됨
- 재귀함수 with 메모이제이션은 테이블을 관리하며 그래프를 탐색하는 것과 같음
- 이 DAG의 위상정렬(들 중 하나)을 잘 찾고, 그걸 루프로 표현하여 채워 주면 그것이 DP
- 역추적?



다이나믹 프로그래밍 문제를 푸는 방법

- 최적해를 계산하기 위한 배열 등을 정의
 - 예) $F[i] = i$ 번째 피보나치 수
- 정의한 배열에 대해 재귀적인 수식을 세움
 - 예) $F[i] = F[i-1] + F[i-2]$
- 적절한 순서대로 재귀적인 수식을 이용해 배열을 채움
 - 예) `for (int i=2; i<=n; i++) F[i] = F[i-1] + F[i-2];`

다이나믹 프로그래밍 문제의 유형

- 유형 1: 여러 옵션 중 하나 선택 (예제 1, 2)
- 유형 2: 구간 DP (예제 3)
- 유형 3: 여러 변수를 이용하는 DP (예제 4)

예제 1: Longest Increasing Subsequence

- 문제: 수열 a_1, a_2, \dots, a_n 이 있을 때, 가장 긴 증가하는 부분수열의 길이를 구하여라.

예제 1: Longest Increasing Subsequence

- 문제: 수열 a_1, a_2, \dots, a_n 이 있을 때, 가장 긴 증가하는 부분수열의 길이를 구하여라.
- $D(i) = a_i$ 를 끝으로 하는 가장 긴 증가하는 부분수열의 길이
- $D(i) = \max\{D(j) \mid j < i \text{ and } a_j < a_i\} + 1$

예제 2: String Concatenation

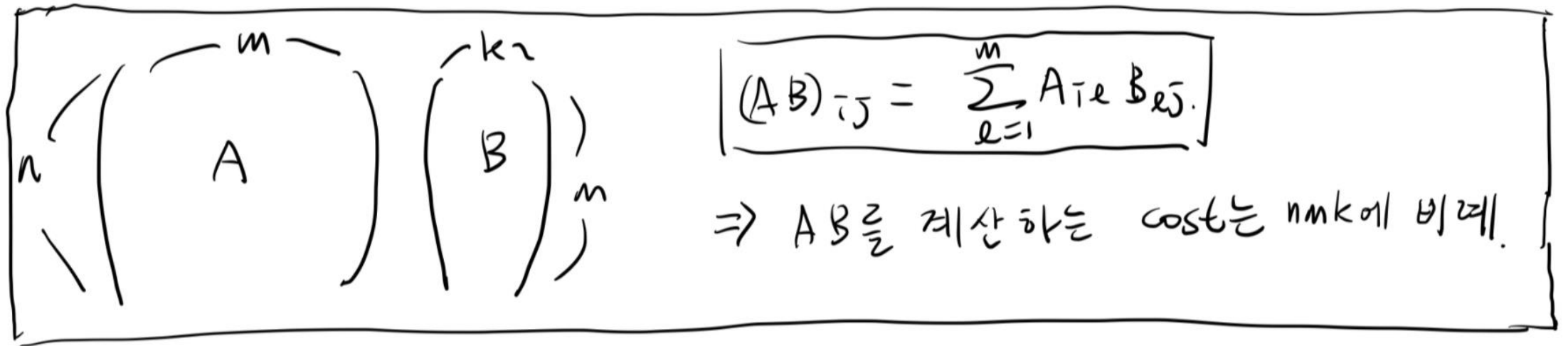
- 문제: n 개의 문자열 s_1, s_2, \dots, s_n 과 문자열 T 가 주어진다. 이 때, s_1, s_2, \dots, s_n 을 각각 0번 이상 사용하여 이어 붙여 T 를 만들 수 있는지 판별하여라.

예제 2: String Concatenation

- 문제: n 개의 문자열 s_1, s_2, \dots, s_n 과 문자열 T 가 주어진다. 이 때, s_1, s_2, \dots, s_n 을 각각 0번 이상 사용하여 이어 붙여 T 를 만들 수 있는지 판별하여라.
- $D(i) = T[1 \dots i]$ 를 만들 수 있는가? (true / false)
- $D(i) = \bigvee_{j=1}^i (T \text{ ends with } S_j \text{ and } D(i - \text{len}(S_j)))$

예제 3: Matrix Chain Multiplication

- 문제: 행렬 A_1, A_2, \dots, A_n 이 있다. A_i 의 크기는 $r_i \times c_i$ 이며, $r_{i+1} = c_i$ 이다. 이 때, $A_1 \cdots A_n$ 을 계산하기 위한 최소 cost를 구하여라.
- 순서가 상관 있을까? $A_1: 3 \times 100, A_2: 100 \times 5, A_3: 5 \times 5$ 일 때:
 - $(A_1 A_2) A_3$: 1575
 - $A_1 (A_2 A_3)$: 4000

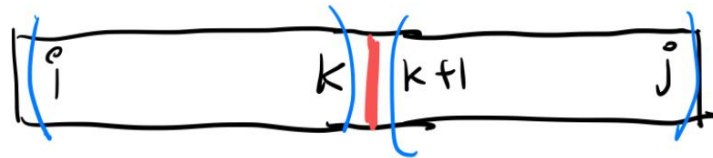

$$(AB)_{ij} = \sum_{l=1}^m A_{il} B_{lj}$$

$\Rightarrow AB$ 를 계산하는 cost는 nmk 에 비례.

예제 3: Matrix Chain Multiplication

- 문제: 행렬 A_1, A_2, \dots, A_n 이 있다. A_i 의 크기는 $r_i \times c_i$ 이며, $r_{i+1} = c_i$ 이다. 이 때, $A_1 \cdots A_n$ 을 계산하기 위한 최소 cost를 구하여라.

$D(i, j) = A_i A_{i+1} \cdots A_{j-1} A_j$ 를 계산하는 최소 비용.



$$\Rightarrow D(i, j) = \min_{i \leq k < j} (D(i, k) + D(k+1, j) + r_i c_k c_j).$$

예제 4: Longest Common Subsequence

- Example:

	0	1	2	3	4	5	6	7	8
S	/x/	A	B	C	C	D	A	B	C
T	/x/	A	C	A	D	C	/	/	/

Handwritten diagram illustrating the Longest Common Subsequence (LCS) between two strings S and T. The strings are S = A B C C D A B C and T = A C A D C. The diagram shows the alignment of characters and the path of the longest common subsequence (A C D C) highlighted with blue circles and arrows. Red slashes indicate mismatches or non-optimal paths.

예제 4: Longest Common Subsequence

- $D(i, j)$ = $S[1, i]$ 와 $T[1, j]$ 의 LCS 길이
- $D(i, j) = \max \{$
 - $D(i-1, j)$: $S[i]$ 를 안 쓰는 경우,
 - $D(i, j-1)$: $T[j]$ 를 안 쓰는 경우,
 - $D(i-1, j-1)+1$: $S[i] = T[j]$ 인 경우 $\}$