ICPC Sinchon

2022 Winter Algorithm Camp

3. 동적 계획법

서강대학교 임지환

2022 Winter Algorithm Camp

목차

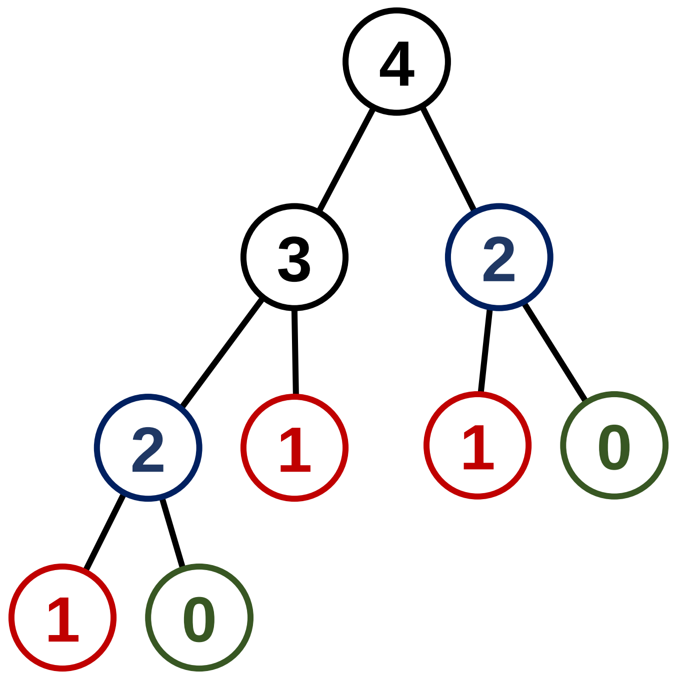
240

Introduction – 동적 계획법

* Dynamic Programming

- 전체 문제를 작은 **부분문제(subproblem)**로 나누어 푸는 방법

ex) $fib(n) = fib(n-1) + fib(n-2)$ -> $T(n) = T(n-1) + T(n-2) + O(1) \geq 2T(n-2) = c \cdot 2^{\frac{n}{2}}$



Introduction – 동적 계획법

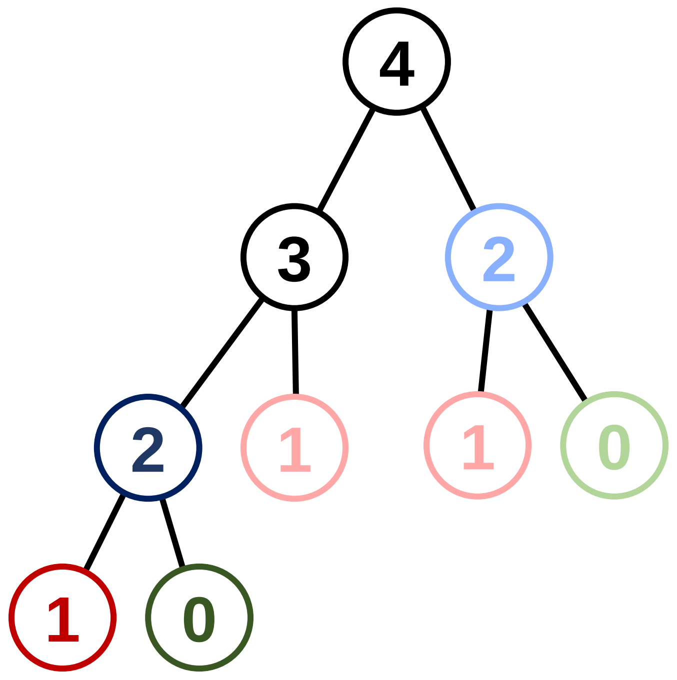
* Dynamic Programming

- **중복되는 부분문제(overlapping subproblem)**는 한 번만 계산

ex) $fib(n) = fib(n-1) + fib(n-2)$

* **메모이제이션 (memoization)**

- 결과를 저장하는 장소를 마련해두고,
  한 번 계산한 값을 저장해 뒀다가 **재활용**



Introduction – 동적 계획법

* Base Case

ex)

$$fib(n) = \begin{cases} n & n \leq 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$$

```
fib(n, d):
    if n <= 1
        return n
    if d[n] >= 0
        return d[n]
    d[n] = fib(n-1) + fib(n-2)
    return d[n]
```

11048. 이동하기

$N \times M$ 크기의 격자 a $(1 \le N, M \le 1{,}000, 0 \le a_{ij} \le 100)$

(r, c)에서 이동 가능 방향은 (r+1, c), (r, c+1), (r+1, c+1), 각 칸을 방문 시 칸에 쓰여진 값만큼 추가

(1,1) 에서 (N, M)으로 이동할 때 얻을 수 있는 경로 합의 최대?

11048. 이동하기

1. 무식한 방법부터 떠올려보자.

각 칸에서 이동할 수 있는 경우의 수: 3가지
$d(r,c)$: (r,c)로부터 출발하여 얻을 수 있는 합의 최대값
but, 모든 칸에 대하여 이동할 수 있는 경로의 가짓 수 ≤ 3^{nm}

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}
a_{41}	a_{42}	a_{43}

11048. 이동하기

2. 무식한 방법을 개선시켜보자.

$d(r,c)$는 어떠한 값들로부터 얻어질까?

$$d(r,c) = max\begin{cases} d(r+1,c) \\ d(r,c+1) \\ d(r+1,c+1) \end{cases} + a_{rc}$$

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}
a_{41}	a_{42}	a_{43}

11048. 이동하기

3. 재귀식의 종료조건

$$d(r,c) = max\begin{cases} d(r+1,c) \\ d(r,c+1) \\ d(r+1,c+1) \end{cases} + a_{rc}$$

$d(r,c)$: (r,c)로부터 출발하여 $(\boldsymbol{N},\boldsymbol{M})$**에 도달할 때까지** 얻을 수 있는 합의 최대값
∴ base case of (r,c) = (N,M),

Introduction – 동적 계획법

* 최적 부분구조(Optimal Substructure)

  - 전체 문제의 최적해가 부분 문제의 최적해를 포함

Approach

1. 최적해의 구조를 정의해본다.
2. 최적해의 값을 정의하기 위한 관계식(점화식)을 짜본다.
3. 부분 문제부터 최적해를 구해나가며 전체 문제를 구한다.

Approach – Complexity Analysis

* 관계식을 정의하는 과정에서의 시간복잡도 & 공간복잡도

- 공간복잡도: 존재하는 부분 문제의 개수
- 시간복잡도: 존재하는 부분 문제의 개수 × 한 부분 문제를 풀 때 필요한 반복문의 수행 횟수

240

11053. 가장 긴 증가하는 부분 수열

N 크기의 수열 a $(1 \le N \le 1{,}000, 1 \le a_i \le 1{,}000)$

가장 긴 증가하는 부분 수열의 길이를 구해보자.

11053. 가장 긴 증가하는 부분 수열

1. 최적해 구조 찾기

가장 긴 증가하는 부분수열로서 a_i를 선택해야 한다면:
1) a_i 이전에$(= j, j < i)$ $a_j < a_i$인 어떤 값이 있었다.
2) a_i 가 수열의 시작이다.

11053. 가장 긴 증가하는 부분 수열

2. 점화식 구하기

1) a_i 이전에$(= j, j < i)$ $a_j < a_i$인 어떤 값이 있었다.
2) a_i 가 수열의 시작이다.

$d(n)$: a_n으로 끝나는 증가하는 부분 수열의 최대 길이

$d(n) = \max(1, d(i) + 1)\ \ for\ \ d(i) < d(n),\ i < n$

11053. 가장 긴 증가하는 부분 수열

3. Complexity Analysis

$d(n) = \max(1, d(i) + 1) \;\; for \;\; d(i) < d(n), \; i < n$

부분 문제의 개수: n개 ($\because 1 \leq n \leq N$)
한 부분 문제를 풀 때 필요한 반복문의 수행 횟수: 최대 n번 ($\because i < n$)

Total Time Complexity: $O(N^2)$

9251. LCS

N, M 길이의 두 문자열 a, b $(1 \le N, M \le 1{,}000)$

모두의 부분 수열이 되는 수열 중 가장 긴 것(LCS; Longest Common Subsequence)의 길이?

9251. LCS

1. 최적해 구조 찾기

1) 문자열이 2개이기 때문에 하나의 문자열에 대한 index 관계로는 정의가 안된다.
2) 접두사에 한글자씩 append하면 원래 문자열이 된다.

9251. LCS

2. 점화식 구하기

두 문자열의 접두사 $a[0..i], b[0..j]$에 대하여

$$lcs(i,j) = \begin{cases} lcs(i-1,j-1)+1 & a[i]=b[j] \\ \max\big(lcs(i,j-1), lcs(i-1,j)\big) & else \end{cases}$$

11049. 행렬 곱셈 순서

행렬 N개와 각각의 크기 r_i, c_i $(1 \le N, \le 500, 1 \le r_i, c_i \le 500)$

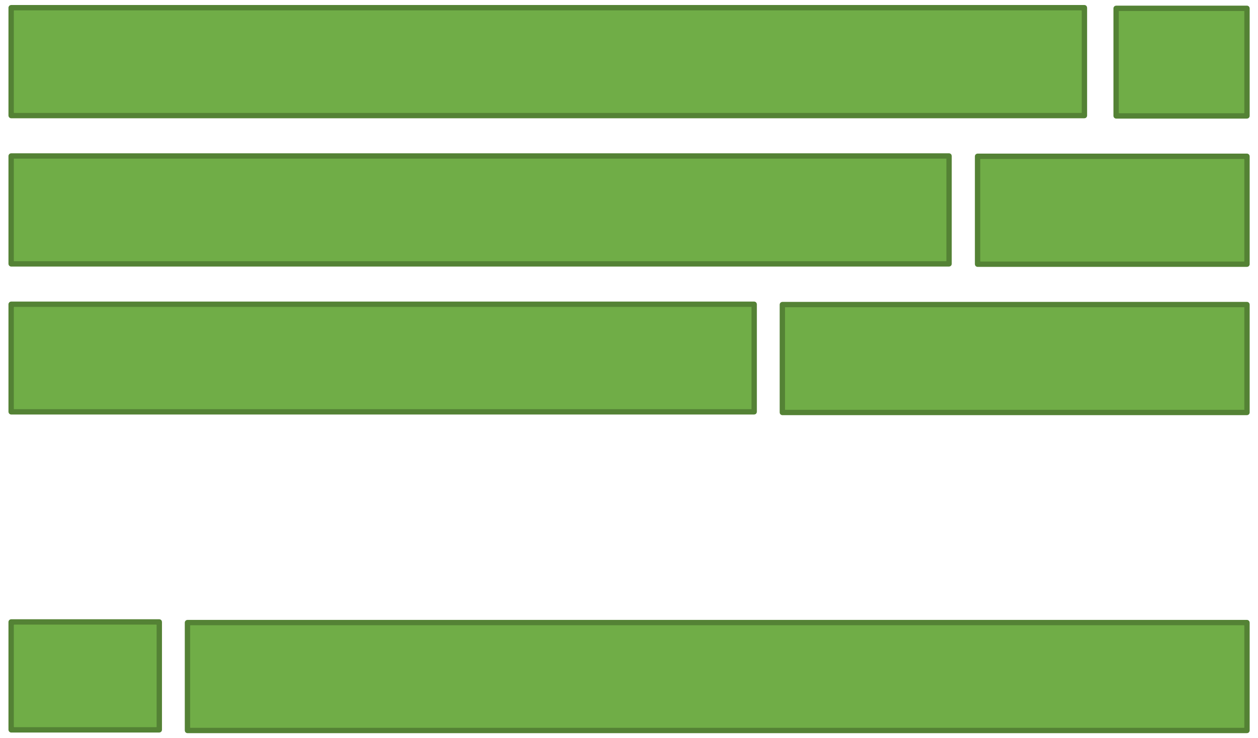
항상 순서대로 곱셈을 할 수 있는 크기만 주어짐

주어진 행렬들의 순서를 바꾸지 않고 모든 행렬을 곱하는데 필요한 곱셈 연산 횟수의 최솟값?

11049. 행렬 곱셈 순서

1. 최적해 구조 찾기

마지막 상황: 덩어리 2개가 있었고, 그 덩어리를 곱해야 하는 상황
마지막 상황 직전의 부분 문제: 왼쪽 덩어리 & 오른쪽 덩어리
계속해서 쪼개질 덩어리를 표현할 수 있는 방법: (L, R)



11049. 행렬 곱셈 순서

2. 점화식 구하기

$d(L, R)$: L번째 행렬부터 R번째 행렬까지 곱하는데 필요한 곱셈 연산 횟수의 최솟값
$C(i, j)$: i번째 행렬과 j번째 행렬을 곱하는데 필요한 곱셈 연산 횟수

$$d(L, R) = d(L, k) + d(k+1, R) + C(k, k+1)$$

11049. 행렬 곱셈 순서

3. Complexity Analysis

$$d(L,R) = d(L,k) + d(k+1,R) + C(k,k+1)$$

서로 다른 (L,R)쌍의 개수: $O(N^2)$
한 부분 문제를 풀 때 필요한 반복문의 수행 횟수: $O(R-L)$번 (최대 N번)

Total Time Complexity: $O(N^3)$

Appendix – Additional Topics

* Tracing

단순히 최적해만을 구해야 한다면 앞서 본 방식으로 끝나지만, 최적해에 도달하기 위한 최적 경로를 묻는다면 추가적인 작업이 필요합니다.
앞선 문제들에서 살펴본 것처럼, 전체 문제에 대한 최적해는 직전 상태들로부터 정의가 되므로 이를 통해 정의할 수 있습니다.

예를 들어 행렬 곱셈 순서(11049) 문제의 점화식을 본다면, 서로 다른 $d(L,k) + d(k+1,R) + C(k,k+1)$중 $d(L,R)$에 매칭되는 값이 있을 것이고, 이 때의 k가 마지막으로 계산되는 위치입니다. 분할된 각 부분에 대해 마찬가지 방식으로 분할되는 위치를 찾을 수 있습니다.

Appendix – Additional Topics

* Knapsack Problem

배열의 Index에 위치정보(i번째)가 아닌 다른 값을 넣어볼 수 있습니다.

배낭문제 (Knapsack Problem)는 배낭의 무게 한도, 각 물건의 개수, 무게 및 가치가 주어졌을 때 물건을 배낭에 챙겨서 가치를 최대한으로 얻어가고자 하는 문제입니다.

이전까지 봐왔던 문제들의 시간복잡도와 달리, 주어진 입력 값(물건의 개수)에 따른 다항시간으로 정의할 수 없으며, Index에 배낭의 Capacity 정보를 넣은, $O(nC)$만에 문제를 풀 수 있습니다.
이런 시간복잡도를 pseudo-polynomial time이라 합니다.

Appendix – Additional Topics

* Bitwise DP(or Bitmask DP, Bitfield DP, etc)

배열의 Index에 위치정보(i번째)가 아닌 상태를 넣어볼 수 있습니다.

Bitwise DP는 N개의 어떤 원소 집합의 상태(binary state)를 통해 상태 변화에 따른 전이를 표현함으로써 $O(N2^N)$과 같은 꼴의 시간복잡도를 갖는 문제입니다.

N이 15~20정도 되는, 매우 작지만 N!은 불가능한 범위가 주어졌을 때 사용해볼 법한 방법입니다.

일부 문제는 중급, 혹은 고급반에서 다루게 될 Maximum Flow 유형과 호환됩니다.

Appendix – Additional Topics

* DP with Data Structure

트리 자료구조는 그 자체로 재귀적인 구조를 갖습니다.
전체 트리와 서브 트리의 관계를 이용하여 트리 상에서의 동적 계획법 문제를 정의하고 풀 수 있습니다.

구조적인 부분 뿐만 아니라 일부 자료구조를 통해 동적 계획법의 시간복잡도 & 공간복잡도 최적화를 할 수도 있습니다.
쉬운 예로 훗날 뒤에서 배우게 될 deque를 이용하여 부분 문제의 값을 구하기 위해 look-up해야 하는 값들의 개수를 최소화할 수 있습니다.

Appendix – Additional Topics

* 분할정복을 이용한 거듭제곱(Exponentiation by squaring)

피보나치 수열의 값을 구하는데 N값이 매우 큰 값을 요구하면 어떨까요?

행렬을 통해 피보나치 수열의 관계를 나타낼 수 있고, 이와 유사한 비교적 적은 관계(= m개)로 구성된 선형 방정식 (linear equation)의 형태로 점화식이 정의될 경우, 그 관계를 행렬에 표현하고 행렬 거듭제곱을 빠르게 수행하여 DP값을 구할 수 있습니다.

N 거듭제곱을 수행하는데 시간복잡도는 $O(\log N)$, $M \times M$행렬 곱셈을 수행하는데 시간복잡도는 $O(M^3)$으로, 최종 $O(M^3 \log N)$에 N번째 항의 값을 구할 수 있습니다.

Appendix – Problems

필수문제

11726	2×n 타일링
1932	정수 삼각형
11055	가장 큰 증가 부분 수열
9177	단어 섞기
11049	행렬 곱셈 순서

연습문제

11727	2×n 타일링 2	2133	타일 채우기
11053	가장 긴 증가하는 부분수열	1915	가장 큰 정사각형
11048	이동하기	7346	유전자 함수
11051	이항 계수 2	11066	파일 합치기
9251	LCS		