2025 겨울방학 알고리즘 스터디

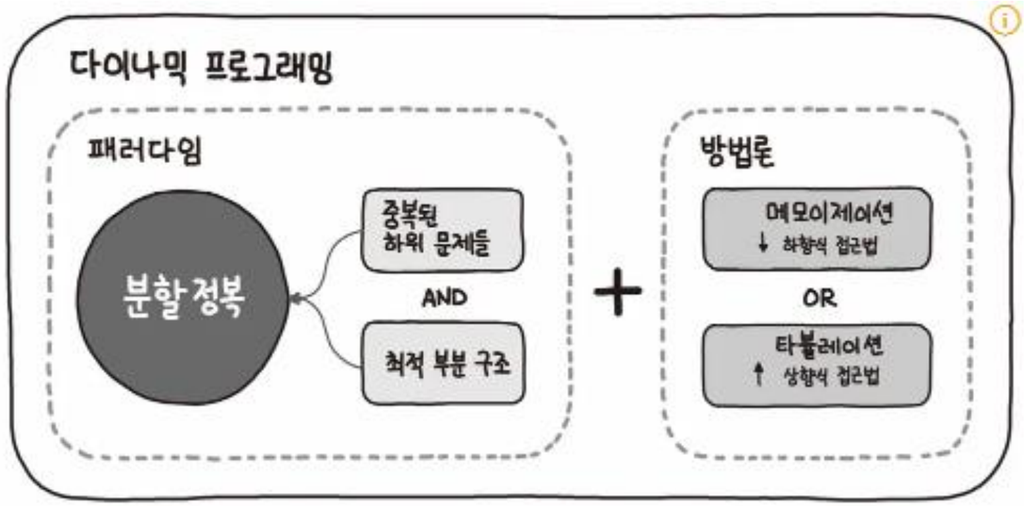
동적 계획법(Bottom-up)

컴퓨터 공학과 20230546 서보경

목차

동적 계획법이란?

동적 계획법(DP)은 복잡한 문제를 해결하기 위해 작은 부분 문제로 나누고, 그 결과를 저장하여 중복 계산을 방지하는 기법이다. 이를 통해 연산량을 줄이고 효율적으로 답을 구할 수 있다.
DP의 핵심 개념은 이전 결과를 활용하여 새로운 문제를 빠르게 해결하는 것이다. 동일한 계산을 반복하지 않도록 값을 저장(메모이제이션)하는 방식을 사용한다.



DP 풀이 방식은 분할정복과 비슷한 면이 상당히 많다.
분할 정복 : 독립적인 부분 문제 해결에 특화
DP : 중복되는 하위 문제를 저장하여 활용

동적 계획법의 특징 - 메모이제이션

지금부터 $\sum_{i=1}^{10} i$의 값을 한번 구해보자.

1까지는 1, 2까지는 3, 3까지는 6, 4까지는 10.....
10까지 55!!
➔ 어떤 식으로 구했나?? 물론 외우고 있는 사람도 있을 것이다.
하지만 더 큰 수에 대해서는 우리가 직접 세면서 더하는 방식을 많이 쓸 것이다.
그 큰 수에 대해서 누적이 가능한건, 직전 값에 대한 정보를 우리의 뇌가 일시적으로 저장하고 있기 때문!

메모이제이션

메모이제이션은 이전 계산 결과를 저장하여 중복 연산을 방지하는 기법이다. 이를 통해 불필요한 계산을 줄이고 더 효율적으로 문제를 해결할 수 있다.

DP는 점화식을 포함할 수 있지만, 반드시 점화식이 필요하지는 않다.
(비트마스크 / 트리 / 게임이론 DP는 일반적인 점화식으로 표현하기 어렵거나 비효율적일 수 있음)

DP의 꽃 = 메모이제이션! (문제의 최적 해가 부분 문제들의 최적 해로부터 구성될 수 있는 성질을 이용해서 최적화 하는 기법이 DP다!)

동적 계획법을 푸는 방법

동적 계획법(DP)의 기본 개념

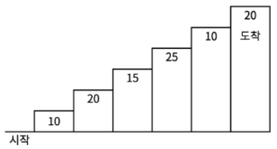
➔ 큰 문제를 작은 부분 문제로 나누고, 그 결과를 저장하여 중복계산을 방지하는 **최적화** 기법.

	바텀업(Bottom-up)	탑다운(Top-down)
방식	반복문 사용	재귀 사용
메모이제이션	있음 (테이블 갱신)	필수 (이미 계산된 값 저장)
호출 방식	작은 문제부터 차례대로 계산	필요할 때만 계산
재귀 오버헤드	O(1) (스택 부담 X)	O(N) 이상의 호출 스택 부담
코드 구조	배열을 기반으로 모든 부분 문제를 순차적으로 계산	논리적으로 문제를 나누어 해결
유리한 문제 유형	모든 부분 문제를 다 풀어야 하는 경우	특정 경우만 계산하는 경우
예시	계단 오르기 문제 / 배낭 문제 / LIS	행렬 곱셈 순서 / LCS / 트리 DP

이번 장에서는 바텀업 기반의 동적계획법 풀이를 배운다!
바텀업은 점화식 기반의 풀이가 상당히 많다.

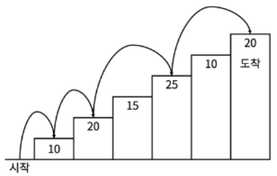
계단 오르기 문제

계단 오르기 게임은 계단 아래 시작점부터 계단 꼭대기에 위치한 도착점까지 가는 게임이다. <그림 1>과 같이 각각의 계단에는 일정한 점수가 쓰여 있는데 계단을 밟으면 그 계단에 쓰여 있는 점수를 얻게 된다.



<그림 1>

예를 들어 <그림 2>와 같이 시작점에서부터 첫 번째, 두 번째, 네 번째, 여섯 번째 계단을 밟아 도착점에 도달하면 총 점수는 10 + 20 + 25 + 20 = 75점이 된다.



<그림 2>

계단 오르는 데는 다음과 같은 규칙이 있다.

1. 계단은 한 번에 한 계단씩 또는 두 계단씩 오를 수 있다. 즉, 한 계단을 밟으면서 이어서 다음 계단이나, 다음 다음 계단으로 오를 수 있다.
2. 연속된 세 개의 계단을 모두 밟아서는 안 된다. 단, 시작점은 계단에 포함되지 않는다.
3. 마지막 도착 계단은 반드시 밟아야 한다.

따라서 첫 번째 계단을 밟고 이어 두 번째 계단이나, 세 번째 계단으로 오를 수 있다. 하지만, 첫 번째 계단을 밟고 이어 네 번째 계단으로 올라가거나, 첫 번째, 두 번째, 세 번째 계단을 연속해서 모두 밟을 수는 없다.

각 계단에 쓰여 있는 점수가 주어질 때 이 게임에서 얻을 수 있는 총 점수의 최댓값을 구하는 프로그램을 작성하시오.

입력

입력의 첫째 줄에 계단의 개수가 주어진다.

둘째 줄부터 한 줄에 하나씩 제일 아래에 놓인 계단부터 순서대로 각 계단에 쓰여 있는 점수가 주어진다. 계단의 개수는 300이하의 자연수이고, 계단에 쓰여 있는 점수는 10,000이하의 자연수이다.

출력

첫째 줄에 계단 오르기 게임에서 얻을 수 있는 총 점수의 최댓값을 출력한다.

너무 전형적인 유형인 계단 오르기 문제이다.
이 문제를 완전 탐색으로 풀 수 있을까?
➔ 불가능하다. (상태 공간 개수 2^n)

잘 보니, 어떤 i단계의 값은 이전 상태에 항상 종속적이다.
ex) 3번째에 도달 하기 위해선?
0 -> 1 -> 3 (O)
0 -> 2 -> 3 (O)
0 -> 1 -> 2 -> 3 (X)

종속적인걸 알았는데, 연속 3개는 안된다?
➔ i번째 값은 (i-1)에서 i로 직접 상태를 전이 시킬 수 없음.
(연속 3칸인지 판정 불가 하거나 구조가 복잡해짐)
➔ 즉, (i-3), (i-2)에서만 전이를 시킬 수 있다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	0	0	0	0	0	0

DP Table

일반적으로 계단 오르기 문제는 바텀업 DP의 접근법을 많이 이용한다.

바텀업 DP의 특징은 앞서 보았던 것 처럼 모든 DP 테이블을 채워 나가면서 문제를 해결하는 방식이다.
먼저, 점수를 입력받은 SCORE 배열과 상태전이를 시킬 DP Table을 준비한다.
또한 DP의 모든 테이블을 0으로 초기화한다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	0	0	0	0

DP Table

먼저 idx 1과 idx 2일때는 이전에 아무런 계단이 없었기 때문에 그대로 값을 업데이트 시키면 된다.
즉, idx가 1일때는 score[1]이 최대, idx가 2일때는 score[1] + score[2]가 최대가 된다.

문제 상황에서 나왔듯이 한 칸이나 두 칸을 연속해서 올라갈 수 있으며 연속된 세 칸을 올라가지 못하게끔 상태전이를 제한 시켜야한다.

전자의 조건은 만족하기 쉬워 보인다. 그러나 후자의 조건을 어떻게 충족할 수 있을까?

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	0	0	0	0

DP Table

가령, 현재 계단이 k번째 계단이라고 하자. 그렇다면 k계단이 가질 수 있는 점수는
dp[k] = dp[k-1] + score[k] 혹은 dp[k] = dp[k-2] + score[k]가 될 것이다.

그러나 아까도 언급했듯이 1칸씩 점프할 때 연속적으로 세칸을 올라가면 안된다는 제약이 발목을 잡는다.
2칸 올라가는건 연속 카운트가 초기화되어서 상관없지만, 1칸씩 올라갔을 때 우리가 몇 칸을 올라갔는지 따로 체크해주어야 할까?

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	0	0	0	0

DP Table

k-1 에서 k로 오기위한 방법을 구할 때 직접적으로 우리가 연속 카운트를 세지 않는 이상 그대로 전이를 시킬 수 없다. (k-2부터 계속 올라왔다면, 이 방식은 조건에 위배 됨)

따라서, k – 3 에서 k – 1로 2칸 점프를 하고 k로 올 때의 최댓값을 구해야한다.
이는 수식으로 dp[k] = dp[k-3] + score[k-1] + score[k]이다.

즉, 강제로 1칸 점프를 2번만 사용하도록 상황을 제한하는 것이다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	0	0	0	0

DP Table

최종적으로 점화식을 제시하자면 아래와 같다.

dp[k] = max(dp[k-2] + score[k], dp[k-3] + score[k-1] + score[k])

즉, 2칸 전에서 현재 칸으로 2칸 점프하는 것이 이득인지, 혹은 3칸 전에서 2칸 점프하고 바로 직전 칸과 현재 칸을 밟는게 이득인지를 비교하면서 테이블을 채워 나가면 된다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	35	0	0	0

DP Table

idx : 3

처음 인덱스는 3부터 시작한다.(이미 1과 2의 테이블을 채웠기 때문에)

A) 1번째에서 2칸 점프해서 3번째에 도달하는 값은 10 + 15 = 25 이다.
B) 0번째에서 2칸 점프한 뒤에, 1칸 점프해서 3번째에 도달하는 값은 0 + 20 + 15 = 35 이다.
이때, 35가 더 큰 값이므로 dp[3]의 값은 35로 정한다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	35	55	0	0

DP Table

idx : 4

인덱스의 위치가 4일 때 이다.

A) 2번째에서 2칸 점프해서 4번째에 도달하는 값은 30 + 25 = 55 이다.
B) 1번째에서 2칸 점프한 뒤에, 1칸 점프해서 4번째에 도달하는 값은 10 + 15 + 25 = 50 이다.
이때, 55가 더 큰 값이므로 dp[4]의 값은 55로 정한다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	35	55	55	0

DP Table

idx : 5

인덱스의 위치가 5일 때 이다.

A) 3번째에서 2칸 점프해서 5번째에 도달하는 값은 35 + 10 = 45 이다.
B) 2번째에서 2칸 점프한 뒤에, 1칸 점프해서 5번째에 도달하는 값은 20 + 25 + 10 = 55 이다.
이때, 55가 더 큰 값이므로 dp[5]의 값은 55로 정한다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	35	55	55	75

DP Table

idx : 6

인덱스의 위치가 6일 때 이다.

A) 4번째에서 2칸 점프해서 6번째에 도달하는 값은 55 + 20 = 75 이다.
B) 3번째에서 2칸 점프한 뒤에, 1칸 점프해서 6번째에 도달하는 값은 35 + 10 + 20 = 65 이다.
이때, 75가 더 큰 값이므로 dp[6]의 값은 75로 정한다. 고로 정답은 75 이다.

계단 오르기 문제

0	10	20	15	25	10	20

SCORE

0	10	30	35	55	55	75

DP Table

이번 문제에서 배울 수 있던 점을 요약하면 아래와 같다.

1 - dp Table을 통해 복잡한 상태 공간을 압축할 수 있다.
2 – 상태 전이를 통해 최적의 해를 구할 수 있다.
3 – 최적 부분 구조를 가지므로, 직전에 사용한 값을 재활용 할 수 있다.
4 – 제약 조건을 활용한 상태 전이 테크닉이 중요하다.
(이번 문제에서는 연속된 3칸 이동을 방지하기 위해 1칸 점프의 사용 조건을 특정 방식으로 강제했다.)
4번 테크닉은 상당히 많이 쓰이는 테크닉이므로 익혀두면 매우 유용하다.

배낭 문제

이 문제는 아주 평범한 배낭에 관한 문제이다.

한 달 후면 국가의 부름을 받게 되는 준서는 여행을 가려고 한다. 세상과의 단절을 슬퍼하며 최대한 즐기기 위한 여행이기 때문에, 가지고 다닐 배낭 또한 최대한 가치 있게 싸려고 한다.

준서가 여행에 필요하다고 생각하는 N개의 물건이 있다. 각 물건은 무게 W와 가치 V를 가지는데, 해당 물건을 배낭에 넣어서 가면 준서가 V만큼 즐길 수 있다. 아직 행군을 해본 적이 없는 준서는 최대 K만큼의 무게만을 넣을 수 있는 배낭만 들고 다닐 수 있다. 준서가 최대한 즐거운 여행을 하기 위해 배낭에 넣을 수 있는 물건들의 가치의 최댓값을 알려주자.

입력

첫 줄에 물품의 수 N(1 ≤ N ≤ 100)과 준서가 버틸 수 있는 무게 K(1 ≤ K ≤ 100,000)가 주어진다. 두 번째 줄부터 N개의 줄에 거쳐 각 물건의 무게 W(1 ≤ W ≤ 100,000)와 해당 물건의 가치 V(0 ≤ V ≤ 1,000)가 주어진다.

입력으로 주어지는 모든 수는 정수이다.

출력

한 줄에 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 출력한다.

또 하나의 Well-Known인 배낭문제다.
이 문제 또한 완전탐색으로 풀 수 없다.

이런 유형을 0/1 Knapsack이라고 하는데
어떤 물건을 넣을지, 말지의 2가지 선택만 가리키기 때문이다. (0 -> 안 넣는다 / 1 -> 넣는다)

어떻게 상태를 정의하고 테이블을 구성해서
최적해를 찾을 수 있을까?

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	0	0
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

배낭 문제는 보통 바텀업 방식으로 해결하는 경향이 강하다.

이 문제에서는 DP Table이 2차원이며, 먼저 모든 원소를 0으로 초기화한다.

그렇다면, 이 테이블을 어떤 방식으로 채울 수 있을까?

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	0	0
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

0/1 배낭 문제는 전형적인 DP 문제로, 상태 전이의 논리가 직관적이다.

아이템 정렬 없이 순차적으로 DP 테이블을 채워나가며, 이전 상태를 활용하는 구조이다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

무게가 5일 때는 1번 아이템을 넣을 수 없지만, 6 이상에서는 가능하다.

즉, 1번 아이템만 고려한다면 6 무게의 최대 가치는 13이 된다.

이처럼 첫 번째 행은 직전 상태가 없으므로 단순한 방식으로 채워진다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

이제 2번 아이템을 고려하자.

무게 4, 가치 8이며, 이전 상태를 활용해 테이블을 채운다.

즉, 최적해를 찾기 위해 이전 상태를 반드시 반영해야 한다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

점화식의 논리는 다음과 같다.

A) 현재 아이템이 넣을 무게보다 크면, 이전 상태 값을 그대로 가져온다.

B) 넣을 수 있다면, 기존 값과 (이전 최댓값 + 현재 아이템 가치) 중 큰 값을 선택한다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	0	0	0	0
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

B번 과정이 중요한 이유는, 현재 아이템을 넣을 때 이전 정보를 활용해야 하기 때문이다.

즉, 현재 무게를 i, 아이템 무게를 k라 할 때,
dp[i][j-1]=max(dp[i][j-1],dp[i−k][j-1]+item)

즉, 기존 상태를 유지할지, 새로운 아이템을 넣을지 비교해 최댓값을 선택한다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	8	8	13	13
3번	0	0	0	0	0	0	0	0
4번	0	0	0	0	0	0	0	0

DP Table

2번 아이템을 다시 보자.

0~3 무게는 그대로 유지한다.

4 이상에서는 이전 값과 (이전 최댓값 + 현재 가치)를 비교해 최댓값을 채운다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

무게 / 번호	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	8	8	13	13
3번	0	0	0	6	8	8	13	0
4번	0	0	0	0	0	0	0	0

DP Table

3번 아이템도 같은 방식으로 채운다.

무게 7에서 중요한 점은, 이전 상태의 값과 전이할 값을 비교하는 것이다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	8	8	13	13
3번	0	0	0	6	8	8	13	14
4번	0	0	0	0	0	0	0	0

DP Table

3번 아이템의 무게는 3이며, 무게 7을 만들려면 4의 상태를 참고한다.

즉, max(dp[4][2]+item[3],dp[7][2]) 를 비교하여 최댓값을 채운다.

이 경우 14가 되어, dp[7][2]=14 가 된다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	8	8	13	13
3번	0	0	0	6	8	8	13	14
4번	0	0	0	6	8	12	13	14

DP Table

같은 방식으로 4번 아이템도 전이하면 최대 가치는 14가 된다.

즉, 배낭 문제는 아이템을 선택할지 말지를 결정하며 DP 테이블을 채우는 과정이다.

배낭 문제

배낭 무게 : 7

번호	1번	2번	3번	4번
무게	6	4	3	5
가치	13	8	6	12

Item

번호 \ 무게	0	1	2	3	4	5	6	7
1번	0	0	0	0	0	0	13	13
2번	0	0	0	0	8	8	13	13
3번	0	0	0	6	8	8	13	14
4번	0	0	0	6	8	12	13	14

DP Table

설명이 다소 길었을 수 있지만, 핵심은 점화식이 아니라 상태 전이의 논리를 이해하는 것이다.

이전 상태를 그대로 유지할지, 새로운 상황을 추가할지 선택하는 과정이 중요하다.