06주차: Knapsack & IntervalDP

강사: 한경수

0. Knapsack 문제란?

대표적인 DP 문제중 하나인 Knapsack문제들에 대해 알아봅시다.

Knapsack 문제란?

배낭에 담을 수 있는 무게의 최댓값이 정해져 있고
일정 **가치**와 **무게**가 있는 짐들을 배낭에 넣을 때,
가치의 합이 최대가 되도록 짐을 고르는 방법을 찾는 문제

Knapsack 문제란?

**예시로는 무게가 W_i, 가치가 C_i인 보석들이 N개가 있다.
배낭이 최대 W_{max}무게 만큼의 물건을 담을 수
있다고 할 때, 가치가 최대가 되도록 보석을 담는 문제를
예시로 들 수 있다.**

Knapsack 문제의 종류

- **Fractional Knapsack Problem**
  1. 보석을 쪼개서 가방에 담는 것이 가능한 경우
  2. 보석들 중 무게 대비 가격이 가장 높은 것부터 담으면 된다.
- **0-1 Knapsack Problem**
  1. 여러 종류의 보석들이 각각 한 개씩만 있는 경우
  2. 가장 대표적인 Knapsack 문제
- **Unbounded Knapsack Problem**
  1. 여러 종류의 보석들이 무한 개 존재하며 원하는 만큼 담을 수 있는 경우

1. 0-1 Knapsack Problem

여러 종류의 보석들이 각각 한 개씩만 있다면?

0-1 Knapsack Problem

DP 배열의 정의

- DP[k][w] = 크기 w의 배낭안에 1~k번 보석까지 확인하여 담았을 때의 최대 가치
- 배낭의 최대 무게가 W_{max}라고 했을 때, k번째 보석까지 확인하여 DP[k][W_{max}]배열까지 채웠다고 가정한다면, 이제 k+1번 보석을 배낭에 넣어서 DP[k][w]를 채우려고 할 때, 두 가지 상황을 가정할 수 있다.
  - K+1번 보석을 담지 않는 경우
  - K+1번 보석을 담는 경우

0-1 Knapsack Problem

- **K+1번 보석을 담지 않는 경우**
  - 가방에 들어있는 보석이 변하지 않을 것이므로, DP[k+1][w]와 DP[k][w]의 값은 같을 것이다.
- **K+1번 보석을 담는 경우**
  - 가방에 K+1번 보석이 들어갈 자리가 필요하므로, 보석의 무게를 W_{k+1}이라 할 때, DP[k][w- W_{k+1}]에서 k+1번째 보석의 가치를 더해준 값이 DP[k+1][w]의 값이 될 것이다.

* DP[k][w- W_{k+1}] = w- W_{k+1}크기의 배낭에 k번째 보석까지 확인하여 담았을 때, 담을 수 있는 최대 가치를 담고 있는 칸

0-1 Knapsack Problem

- **K+1번 보석을 담지 않는 경우**
  - 가방에 들어있는 보석이 변하지 않을 것이므로, DP[k+1][w]와 DP[k][w]의 값은 같을 것이다.
- **K+1번 보석을 담는 경우**
  - 가방에 K+1번 보석을 넣기 전에 가방에 담긴 무게를 W_{k+1}이라 할 때, DP[k][w- W_{k+1}]에서 k+1번째 보석의 가치(C_{k+1})를 더해준 값이 DP[k+1][w]의 값이 될 것이다.

이 두 가지 경우에서 가능한 DP[k+1][w]의 값들 중, 최댓값을 대입해주면 된다.

DP[k+1][w] = max(DP[k][w],DP[k][w- W_{k+1}] + C_{k+1})

위 식의 규칙으로 DP table을 채워나가면 된다.

* DP[k][w- W_{k+1}] = w- W_{k+1}크기의 배낭에 k번째 보석까지 확인하여 담았을 때, 담을 수 있는 최대 가치를 담고 있는 칸

0-1 Knapsack Problem

```
int N, W;                              // N : 보석의 개수
                                       // W : 배낭의 크기
int _w[N_Max + 1], c[N_Max + 1];//_w[k] : k번째 보석의 무게
                                       //c[k] : k번째 보석의 무게
int DP[N_Max + 1][W_Max + 1];
for (int k = 0; k < N; k++) {
    for (int w = 1; w <= W; w++) {
        if (w >= _w[k])                // w - _w[k]가 음수가 되지 않도록 체크!
            DP[k+1][w] = max(DP[k][w], DP[k][w - _w[k]] + c[k+1]);
        else
            DP[k+1][w] = DP[k][w];
    }
}
```

연습문제

BOJ 1535
안녕

다음 페이지에는 문제의 풀이가 있습니다.
혼자 먼저 풀어보고 싶으신 분들은 조심하세요!

연습문제 – BOJ 1535 안녕

전형적인 0-1 knapsack 문제.
배낭의 최대 용량을 99(체력이 0이 되면 죽으니까)
인사할 때 잃는 체력을 보석의 무게,
인사할 때 얻는 기쁨을 보석의 가치로 생각한다.

연습문제 – BOJ 1535 안녕

```
int N, W;                          // N : 보석의 개수
                                   // W : 배낭의 크기
int _w[N_Max + 1], c[N_Max + 1];//_w[k] : k번째 보석의 무게
                                   //c[k] : k번쨰 보석의 무게
int DP[N_Max + 1][W_Max + 1];
for (int k = 0; k < N; k++) {
    for (int w = 1; w <= W; w++) {
        if (w >= _w[k])            // w - _w[k]가 음수가 되지 않도록 체크!
            DP[k+1][w] = max(DP[k][w], DP[k][w - _w[k]] + c[k+1]);
        else
            DP[k+1][w] = DP[k][w];
    }
}
```

위 코드에서 N과 N_Max를 사람의 수, W와 W_Max를 99라고 하고,

_w배열과 c배열에 각각 인사했을 때 잃는 체력과 얻는 기쁨들을 입력 받으면,

결과적으로 DP[N_MAX][W_MAX]칸의 값이 우리가 원하는 답이 된다.

2. Unbounded Knapsack Problem

여러 종류의 보석들이 무한개씩 있다면?

Unbounded Knapsack Problem

DP 배열의 정의

- DP[k][w] = 크기 w의 배낭안에 1~k번 보석까지 확인하여 담았을 때의 최대 가치
- 배낭의 최대 무게가 W_{max}라고 했을 때, k번째 보석까지 확인하여 DP[k][W_{max}]배열까지 채웠다고 가정한다면, 이제 k+1번 보석을 배낭에 넣어서 DP[k][w]를 채우려고 할 때, 두 가지 상황을 가정할 수 있다.
  - K+1번 보석을 담지 않는 경우
  - K+1번 보석을 1개 이상 담는 경우

Unbounded Knapsack Problem

- **K+1번 보석을 담지 않는 경우**
  - 가방에 들어있는 보석이 변하지 않을 것이므로, DP[k+1][w]와 DP[k][w]의 값은 같을 것이다.
- **K+1번 보석을 1개 이상 담는 경우**
  - 가방에 K+1번 보석이 들어갈 자리가 필요하므로, 보석의 무게를 W_{k+1}이라 할 때, DP[k+1][w- W_{k+1}]에서 k+1번째 보석의 가치를 더해준 값이 DP[k+1][w]의 값이 될 것이다.

* DP[k+1][w- W_{k+1}] = w- W_{k+1}크기의 배낭에 k+1번째 보석까지 확인하여 담았을 때, 담을 수 있는 최대 가치를 담고 있는 칸

Unbounded Knapsack Problem

- **K+1번 보석을 담지 않는 경우**
  - 가방에 들어있는 보석이 변하지 않을 것이므로, DP[k+1][w]와 DP[k][w]의 값은 같을 것이다.
- **K+1번 보석을 담는 경우**
  - 가방에 K+1번 보석이 들어갈 자리가 필요하므로, 보석의 무게를 W_{k+1}이라 할 때, DP[k+1][w- W_{k+1}]에서 k+1번째 보석의 가치를 더해준 값이 DP[k+1][w]의 값이 될 것이다.

무엇이 달라졌을까?

* DP[k+1][w- W_{k+1}] = w- W_{k+1}크기의 배낭에 k+1번째 보석까지 확인하여 담았을 때, 담을 수 있는 최대 가치를 담고 있는 칸

Unbounded Knapsack Problem

- **K+1번 보석을 담는 경우**
  - **0-1 knapsack의 경우, k+1번째 보석을 추가로 담을 때, k+1번째 보석이 가방에 없었다는 가정이 있다.(보석은 한 종류당 하나밖에 없으니까)**
    - 즉, DP[k][w- W_{k+1}]배열을 참조해야 한다.
  - **하지만 Unbounded Knapsack의 경우, k+1번째 보석을 추가로 담을 때, 가방 안에 k+1번째 보석이 이미 담겨 있을 수도 있다.**
    - 즉, DP[k+1][w- W_{k+1}]배열을 참조해야 한다.

Unbounded Knapsack Problem

- **K+1번 보석을 담는 경우**
  - 0-1 knapsack의 경우, k+1번째 보석을 추가로 담을 때, k+1번째 보석이 가방에 없었다는 가정이 있다.(보석은 한 종류당 하나밖에 없으니까)
    - 즉, DP[k][w- W_{k+1}]배열을 참조하느냐
  - 하지만 Unbounded knapsack의 경우, k+1번째 보석을 추가로 담을 때, 가방 안에 k+1번째 보석이 이미 담겨 있을 수도 있다.
    - 즉, DP[k+1][w- W_{k+1}]배열을 참조해야 한다.

즉, DP[k][w- W_{k+1}]를 참조하느냐, DP[k+1][w- W_{k+1}]을 참조하느냐 외에는 구현하는데 차이점이 없다.

Unbounded Knapsack Problem

- 0-1 knapsack의 경우, DP배열의 크기는 N*W이고, DP배열을 한 칸씩 모두 채워줘야 하므로 시간 복잡도 또한 O(NW)이다.

- 하지만, Unbounded Knapsack의 경우도 마찬가지로, 공간 복잡도와 시간 복잡도 모두 O(NW)이다.

Unbounded Knapsack Problem

- **공간 복잡도 줄이기**
  - 하지만 Unbounded Knapsack의 경우, 0-1 knapsack과 달리 1차원 배열로 구현할 수 있다.
  - DP[k+1][w]를 구하기 위해서 필요한 곳은 DP[k][w]와 DP[k][w-W_{k+1}]이다. 이를 DP[W_max]를 이용해 똑같이 구현할 수 있다.

Unbounded Knapsack Problem

- 전의 상황과 같이 k+1번째 보석을 담는 상황을 가정해 보자. 현재 DP[W_max]배열에는 k번째 보석까지 확인하여 담았을 때, 최대 가치를 저장하고 있을 것이다.
- 이제 k+1번째 보석을 담기 위해서 DP[0]부터 DP[W_max]까지 갱신하는데, DP[w]를 담을 때, 갱신하기 전에 이미 DP[w]에 들어있는 값은, w크기의 배낭에 k번째 보석까지 확인하여 담았을 때 최대 가치이다 (DP[k][w] 역할). DP[0]부터 갱신해서 DP[w]까지 왔으므로, DP[w-W_{k+1}]에는 w-W_{k+1}크기의 배낭에 k+1번째 보석까지 확인하여 담았을 때의 최대 가치가 담겨있을 것이다. (DP[k+1][w-W_{k+1}] 역할).

Unbounded Knapsack Problem

- EX) 총 배낭의 용량 : 8
- 1번째 보석의 무게: 2, 1번째 보석의 가격: 4
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7

DP	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0

Unbounded Knapsack Problem

- 1번째 보석까지 확인하여 DP테이블을 채워 넣으면 다음과 같다.
- 1번째 보석의 무게: 2, 1번째 보석의 가격: 4

DP	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0

Unbounded Knapsack Problem

- 1번째 보석까지 확인하여 DP테이블을 채워 넣으면 다음과 같다.
- 1번째 보석의 무게: 2, 1번째 보석의 가격: 4

DP	0	1	2	3	4	5	6	7	8
	0	0	4	0	0	0	0	0	0

Unbounded Knapsack Problem

- 1번째 보석까지 확인하여 DP테이블을 채워 넣으면 다음과 같다.
- 1번째 보석의 무게: 2, 1번째 보석의 가격: 4

DP	0	1	2	3	4	5	6	7	8
	0	0	4	4	8	8	12	12	16

Unbounded Knapsack Problem

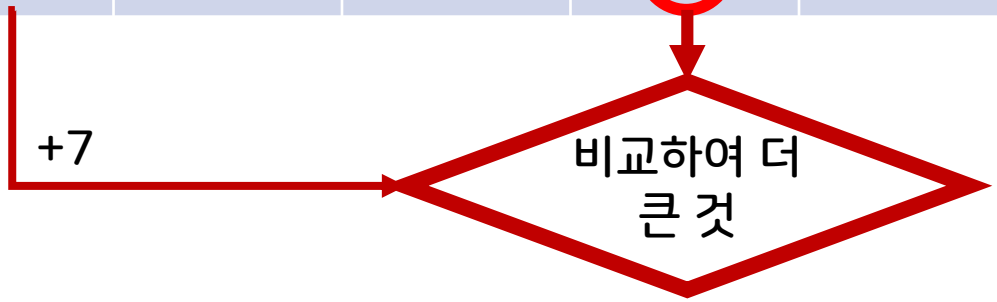
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 2번째 보석의 무게가 3이므로 DP[3]부터 갱신해 나갈 것이다.
- DP[3]을 갱신하는 상황에서 DP[3]에 이미 들어있는 값 '4'는 1번째 보석까지 확인하여 담았을 때의 최대 가치와 같다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	4	8	8	12	12	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- DP[3]에 원래 있던 값(4)와, DP[3-3]+7을 비교해준다

DP	0	1	2	3	4	5	6	7	8
	0	0	4	4	8	8	12	12	16



Unbounded Knapsack Problem

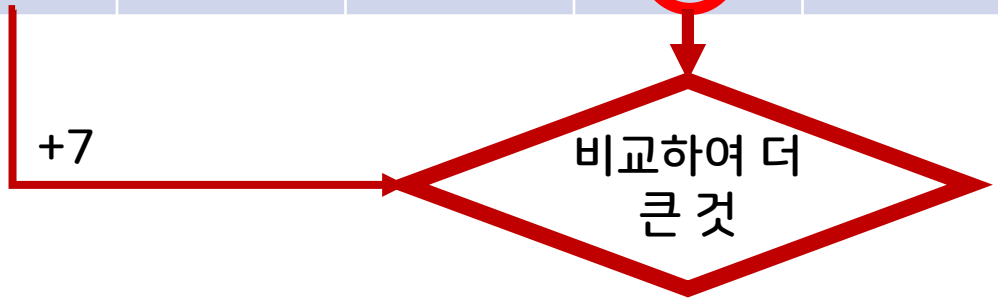
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- DP[3]에 원래 있던 값(4)와, DP[3-3]+7을 비교해준다

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	8	12	12	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	8	12	12	16



Unbounded Knapsack Problem

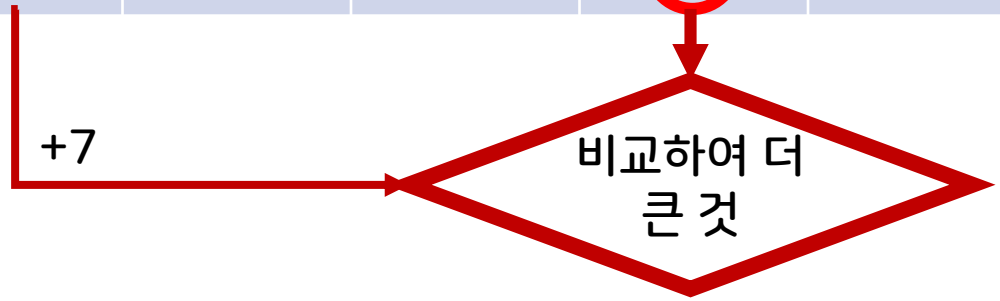
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	8	12	12	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	8	12	12	16



Unbounded Knapsack Problem

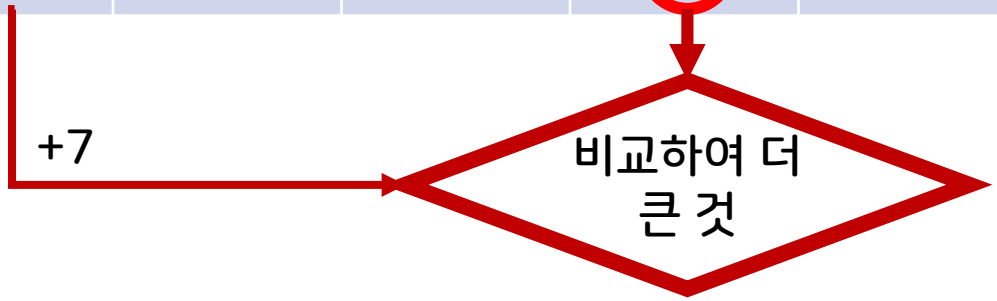
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	12	12	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	12	12	16



Unbounded Knapsack Problem

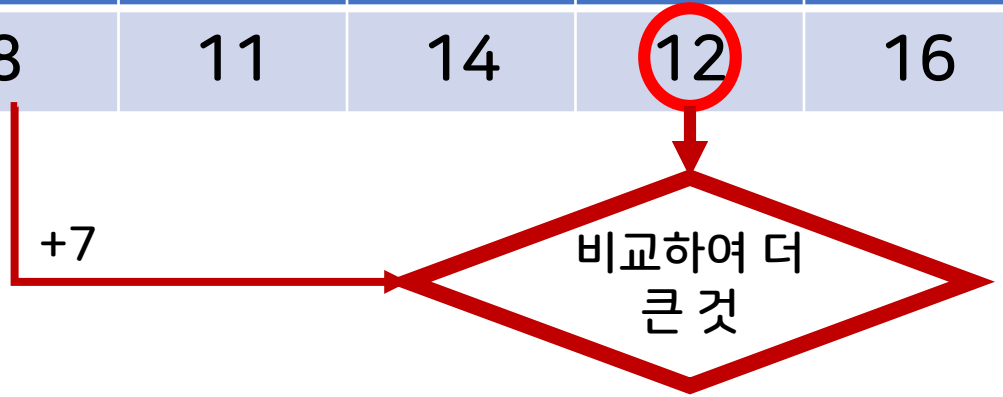
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	14	12	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	14	12	16



Unbounded Knapsack Problem

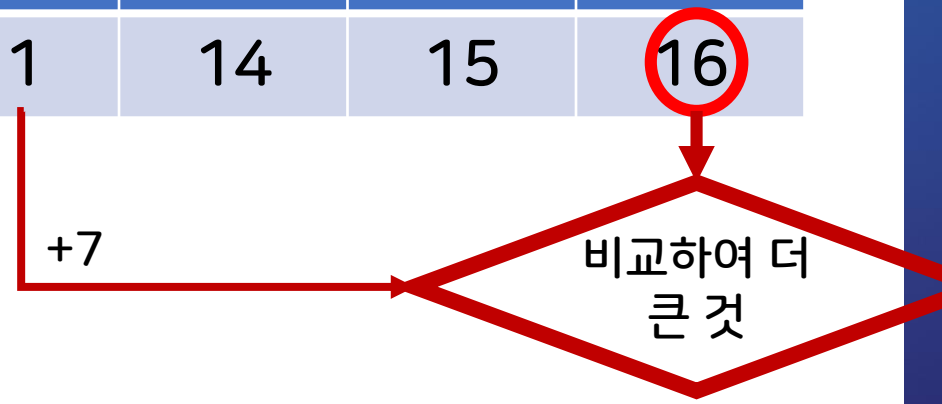
- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	14	15	16

Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	14	15	16



Unbounded Knapsack Problem

- 2번째 보석의 무게: 3, 1번째 보석의 가격: 7
- 이를 반복하여 DP[8]까지 갱신한다.

DP	0	1	2	3	4	5	6	7	8
	0	0	4	7	8	11	14	15	18

연습문제

BOJ 4781
사탕가게

다음 페이지에는 문제의 풀이가 있습니다.

혼자 먼저 풀어보고 싶으신 분들은 조심하세요!

연습문제 – BOJ 9084 동전

전형적인 **Unbounded knapsack** 문제.

배낭의 최대 용량을 상근이의 돈, 보석의 크기를 사탕의 가격, 보석의 가치를 사탕의 칼로리로 계산하면 된다.

이때, 입력되는 돈들의 값이 소수점 둘째자리의 수로 입력되는데, 입력 받은 소수가 P라고 할 때 돈을 (P*100+0.5)를 통해 int형으로 casting한 뒤 계산하여야 AC를 받을 수 있다.

3. Interval DP

또다른 DP의 종류 중 하나인, Interval DP에 대해 알아봅시다.

Interval DP란?

- **2차원 DP의 일종**
- **DP[i][j]가 i번째부터 j번째까지의 정보를 담고 있을 때, 이러한 종류의 DP를 Interval DP라고 한다.**

Interval DP가 어떤 경우에 필요한가?

- 일반적인 DP정의는 처리해야 하는 배열이 1차원이라면 DP도 1차원, 처리해야 하는 배열이 2차원이라면 DP배열도 2차원으로 정하는 것이 일반적이다.
- 1차원 배열을 처리하는 경우, DP[i]를 보통 0~i번째 까지 확인하였을 때, ~(최솟값, 최댓값 등)값을 저장한다고 정의한다.
- 하지만, 1차원 배열을 처리할 때, DP[i][j]를 배열의 i번째에서 j번째 배열까지 확인했을 때, ~값을 저장한다고 정의하는 것이다.

예시문제

BOJ 11066

파일 합치기

문제의 조건에 대한 설명은 생략하니

백준 문제 한번씩만 읽어 봐 주세요!

DP table의 정의

- 처리해야 하는 배열은 문자열 하나로, 1차원 배열이지만 dp는 2차원으로 정의할 것이다.
- DP[i][j]는 i번째 파일부터 j번째 파일을 합칠 때 필요한 최소 비용을 저장하는 배열이다.

- 문제의 예제입력에서 파일 4개(40,30,30,50)을 합치는 경우를 살펴보자.

예제 문제 – 11066번

- DP table의 정의에 따라 결국 우리가 원하는 답은 DP[1][4]에 저장되어있다.
- 마지막 1개의 파일을 만들 때, 아래 3가지 경우가 있다.
  - {40}/{30,30,50}
  - {40,30}/{30,50}
  - {40,30,30}/{50}

예제 문제 – 11066번

- 앞 슬라이드의 3가지 경우 중, {40}/{30,30,50}의 경우를 살펴보자.
- 우선 {40}이라는 파일의 경우, 다른 장들과 합쳐진 적이 없으므로, {40}이라는 파일을 만드는데 드는 비용은 0이다. 즉, DP[1][1]은 0이다.
- {30,30,50}의 파일을 만드는 최소 비용도 앞 슬라이드와 마찬가지로 2가지로 생각할 수 있다.
  - {30}/{30,50}의 경우
    - {30}을 만드는데는 0, {30,50}은 연속한 두 장이므로, 최소비용은 80이다.
    - 또한 {30}과 {30,50}을 합치는데 110의 비용이 더 드므로 총 190의 비용이 든다.
  - {30,30}/{50}의 경우
    - {30,30}의 경우는 60, {50}을 만드는 데는 0의 비용이 든다.
    - 또한 {30}과 {30,50}을 합치는데 110의 비용이 더 드므로 총 170의 비용이 든다.
  - 위의 두 경우를 비교했을 때, {30,30,50}의 파일을 만드는 데는 최소 170의 비용이 든다는 것을 알 수 있다.

예제 문제 - 11066번

- 앞 슬라이드의 3가지 경우 중, {40}/{30,30,50}의 경우를 살펴보자.
- 우선 {40}이라는 파일의 경우, 다른 장들과 합쳐진 적이 없으므로, {40}이라는 파일을 만드는데 드는 비용은 0이다. 즉, DP[1][1]은 0이다.
- {30,30,50}의 파일을 만드는 최소 비용도 앞 슬라이드와 마찬가지로 2가지로 생각할 수 있다.
  - {30}/{30,50}의 경우
    - {30}을 만드는데는 0, {30,50}은 연속한 두 장이므로, 최소비용은 80이다.
    - 또한 {30}과 {30,50}을 합치는데 110의 비용이 더 드므로 총 190의 비용이 든다.
  - {30,30}/{50}의 경우
    - {30,30}의 경우는 60, {50}을 만드는 데는 0의 비용이 든다.
    - 또한 {30}과 {30,50}을 합치는데 110의 비용이 더 드므로 총 170의 비용이 든다.
  - 위의 두 경우를 비교했을 때, {30,30,50}의 파일을 만드는 데는 최소 170의 비용이 든다는 것을 알 수 있다.

즉, {40}과 {30,30,50} 두 파일을 합쳐서
{40,30,30,50} 파일을 만드는 데는
0 + 170 + 150 = 320의 비용이 드는 것을 알 수 있다.

예제 문제 – 11066번

- 마찬가지의 방식으로 {40,30}/{30,50}의 경우와 {40,30,30}/{50}의 경우까지 비용을 구해주면, 첫번째는 비용이 300, 두번째는 비용이 310의 비용이 드는 것을 알 수 있다.
- 이 3가지의 경우 중 최소 비용이 드는 것은 {40,30}/{30,50}이므로, DP[1][4] = 300인 것을 알 수 있다.
- 즉, DP[i][j] = $\min(i \le k < j)\{\mathrm{DP[i][k]} + \mathrm{dp[k+1][j]}\} + (\mathrm{sum[j]} - \mathrm{sum[i-1]})$

어떻게 쪼개진 파일을 합쳐야 최소 비용이 되는지 결정

쪼개진 파일들을 합칠 때 드는 비용(일정)

- **sum[i]는 0~i번째 배열까지의 합을 의미함.**
- **sum[j]-sum[i-1]은 i에서 j번째 배열까지의 부분합을 의미함. 파일을 합칠 때 필요한 비용**