Knapsack Problem(배낭 문제)

어떤 도둑이 박물관에 들어와서 보석을 훔치려고 한다. 하지만 이 멍청한 도둑은 배낭을 1개 밖에 안들고 와서 이 배낭에 들어갈 수 있는 만큼의 보석밖에 가져갈 수 없다. 이 때, 도둑은 배낭에 보석들을 꽉 채웠을 때 값이 제일 많이 나가도록 보석들을 고를 것이다. 이 때 보석들의 값의 합을 어떻게 최대로 고르는가?

배낭 문제는 내가 들어보지는 못했지만 굉장히 유명한 문제인 것 같다. 2가지 분류로 나뉘는데 0-1 knapsack problem과 fractional knapsack problem이다. 전자는 보석들이 절대 쪼개질 수 없는 경우다. 예를 들어, 모나리자 같은 경우는 반으로 쪼갠다고 그 값어치가 반이 되는 것이 아니기 때문에 있는 그대로 배낭에 집어넣어야 한다. 하지만 금가루의 경우 fraction이 가능하기 때문에 fractional이라고 부르는 것이다. 종류가 2가지이듯이, 풀 수 있는 방법도 2가지 이다.

- **0-1 knapsack problem** : 쪼갤 수 없기 때문에 DP로 풀 수 있다. 모든 경우를 고려해야 한다.
- **Fractional knapsack problem** : 쪼갤 수 있기 때문에 Greedy로 풀 수 있다. 최선의 선택!!

물론 둘 다 Brute force로 풀 수 있지만 첫번째 문제의 경우 보석을 배낭에 집어 넣는 경우(0), 집어넣지 않는 경우(1)가 있기 때문에 n개의 보석이 있다고 하면 시간복잡도는 $\theta(2^n)$이 걸려 푸는데 너무 오래 걸린다. 후자는 아직 배우지 않았기 때문에 여기선 전자만 다루도록 하자.

DP의 점화식 정의

DP문제를 풀면서 느끼는 것은 Recurrence equation(점화식)을 얼마나 잘 정의하느냐에 따라서 풀 수 있고 없고가 결정되며 그걸 정의하는 것이 너무 어렵다는 것이다. 여기선 첫 번째 접근을 제시하는데 바로 S_k를 1~k번째 까지의 보석의 값어치 합을 최대로 하는 optimal substructure로 본다. 하지만 이는 S_{k-1}이 S_k의 optimal substructure가 되지 않는 반례들이 존재하기 때문에 안된다. 따라서 다른 정의를 해야 한다. 아래가 새로운 정의이다.

$$B[k,W] = \begin{cases} B[k-1,W] & \text{if } w_k > W \\ max(B[k-1,W], B[k-1,W-w_k] + b_k) & \text{if } w_k \leq W \end{cases}$$

$B[k,W]$은 k번째까지의 보석중에서 크기를 최대 W만큼까지 허용하는 값어치의 모든 경우를 말한다. 따라서 2가지 경우는 다음과 같이 해석된다.

- $w_k > W$: k번째 보석의 크기가 허용크기보다 큰 경우, k-1번째까지의 값어치가 적용됨.
- $w_k \leq W$: k번째 보석의 크기가 허용크기보다 작은 경우, 다시 2가지로 나뉜다.
  - k번째 보석이 들어가지 않는 경우 : $B[k-1,W]$
  - k번재 보석이 들어가는 경우 : $B[k-1,W-w_k] + b_k$

k번재 보석이 들어가는 경우가 살짝 헷갈렸는데 단순하게 생각해보자. k번째 보석이 들어간다는 것은 k-1번째에서 아직 k번째 보석이 들어가지 않은 경우에 k번째의 보석의 값을 더하는 것이다. 따라서 $W-w_k$를 해야 하는 것이다.

코드를 통한 풀이

```
#include <cstdio>
```

```c
// 보석의 최대개수를 100개, knapsack의 최대 크기를 100이라고 하자.
int knapsack[101][101];
int jewel[101][2]; // jewel[][0]=jewel의 크기, jewel[][1]=jewel의 값

int main(void)
{
    int n, k; scanf("%d %d", &n, &k);
    for (int i = 1; i <= n; i++) scanf("%d %d", &jewel[i][0], &jewel[i][1]);

    // jewel의 개수가 0이라면 knapsack에 들어간 jewel의 가치는 0
    for (int w = 0; w <= k; w++) knapsack[0][w] = 0;


    for (int i = 1; i <= n; i++) {
        // knapsack의 크기가 0이라면 어떤 jewel도 들어갈 수 없으므로 또한 0
        knapsack[i][0] = 0;

        // wi=jewel의 크기, bi=jewel의 가격(benefit)으로 설정
        int wi = jewel[i][0], bi = jewel[i][1];

        for (int w = 1; w <= k; w++) {
            if (wi <= w) {
                if (bi + knapsack[i - 1][w - wi] > knapsack[i - 1][w])
                    knapsack[i][w] = bi + knapsack[i - 1][w - wi];
                else
                    knapsack[i][w] = knapsack[i - 1][w];
            }
            else
                knapsack[i][w] = knapsack[i - 1][w];
        }
    }

    printf("최종 답 : %d\n", knapsack[n][k]);
    return 0;
}
```

딱히 별건 없다. 단순히 점화식을 코드로 구현한 것 뿐인데, 중요한 점은 여기선 knapsack 배열이 전역변수라 모두 0으로 초기화되지만 지역변수를 사용할 경우 base case를 주어야 dp를 사용할 수 있어서 0으로 초기화해야 된다는 점이다. 0으로 초기화하는 파트는 2개 부분인데, 다음과 같다.

- knapsack이 수용할 수 있는 크기가 0 : knapsack[i][0]인 경우
- jewel을 하나도 knapsack에 넣지 않은 경우 : knapsack[0][i]인 경우