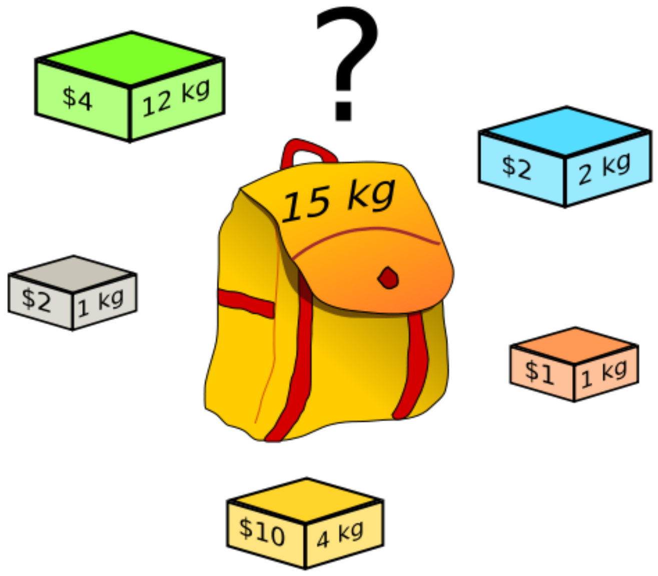
Knapsack Problem

커피동아리 권혁동

HANSUNG UNIVERSITY

Knapsack Problem

- Combinatorial optimization와 관련된 문제
  - 가방 용량은 제한되어 있음(K)
  - 물건의 수는 정해져 있음(N)
  - 넣고 싶은 물건의 가치는 클 수록 좋음(V)
  - 하지만 무게 때문에 모두 담을 수 없음(W)
  - 그렇다면 어떻게 물건을 담아야 하는가?
- 단순해 보이는 문제지만 NP-Complete에 속함
  - 암호의 기반 원리로 사용하기 좋음
  - 고전 컴퓨터에서 연산은 평범
  - **양자 컴퓨터 상에서 연산은 효율적**



Knapsack Problem

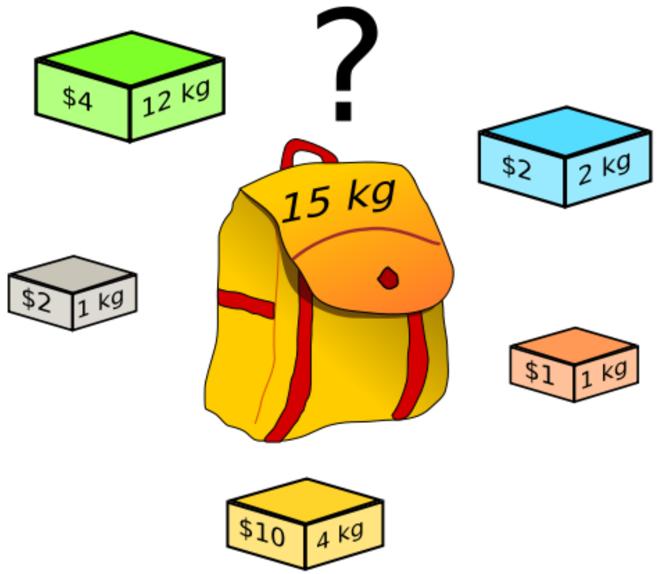
- 크게 두 가지 유형으로 분류
  - Fractional Knapsack Problem: 물건을 자를 수 있음
  - **0-1 Knapsack Problem: 물건을 자를 수 없음**
- 특별한 이유가 없다면 0-1 Knapsack Problem을 적용
- 다양한 풀이 방법이 존재하나, 효율적인 풀이 방법 고찰이 필요
  - O-notation에 따라서 분류

Knapsack Problem

- 각 원소 x_i는 분리될 수 없음 (0 또는 1만 가짐, 0-1 knapsack)
- 원소는 1부터 n까지 존재함
- 각 원소는 가중치(weight) w_i, 가치(value) v_i를 지님
- 원소를 수집할 수 있는 최대 가중치(capacity)는 W
- **가치가 가장 클 수 있는 조합**: $\sum_{i=1}^{n} v_i x_i$
- **제한 조건**: $\sum_{i=1}^{n} w_i x_i \leq W \; and \; x_i \in \{0,1\}$

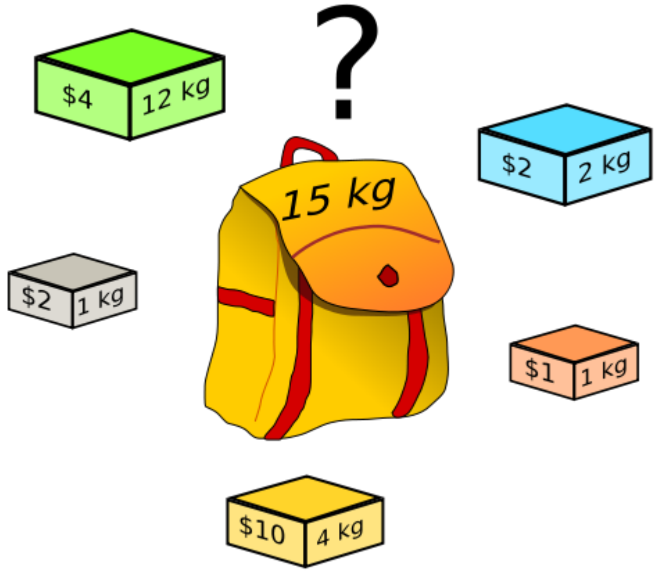
Knapsack Problem: Brute-force

- 가장 좋은 조합을 찾을 때까지 **모든 조합을 넣어보기**
  - 12kg → 2kg → X
  - 12kg → 1kg($2) → 1kg($1) → X
  - ...를 반복하면 언젠가는 정답에 도달
- 물건의 수가 N개라면 복잡도는 $O(2^n)$
  - 운이 좋다면 빨리 찾지만, 일반적으로 비효율적



Knapsack Problem: Greedy

- 가장 가치가 높은 물건을 먼저 택하는 방식
  - 4kg($10) → 2kg($2) → 1kg($2) → 1kg($1)
  - 하지만 아래 그림 상에서는 효과적이지 않음
  - Brute-force 보다는 좋지만, **최적을 보장하지 못함**
- 또는 무게당 가격을 먼저 택하는 방법도 존재
  - Fractional Knapsack의 경우에는 이 방법이 유효
  - 0-1 Knapsack은 무게당 가격 비율을 사용할 수 없음



Knapsack Problem: Dynamic Programming

- Principle of Optimality를 만족할 경우, DP를 적용 가능
  - 1. 어떤 큰 문제를 부분 문제로 분리 가능할 때,
  - 2. 부분 문제의 정답은 이를 포함한 큰 문제에 동일하게 적용
- DP는 이미 계산한 값을 저장(Memoization)하여 반복을 제거
- 피보나치 수열을 최적화 하는 경우가 이에 속함
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 … = $D[i] = D[i-1] + D[i-2]$
  - 100번째 피보나치 수열을 직접 계산하면 매우 오래 걸리나,
    메모리 상에서 중간 값인 99번째, 98번째 값을 호출하면 바로 계산 가능

$$\begin{array}{ccccccccccc} & & & & & 1 & & & & & \\ & & & & 1 & & 1 & & & & \\ & & & 1 & & 2 & & 1 & & & \\ & & 1 & & 3 & & 3 & & 1 & & \\ & 1 & & 4 & & 6 & & 4 & & 1 & \\ 1 & & 5 & & 10 & & 10 & & 5 & & 1 \end{array}$$

Knapsack Problem: Dynamic Programming

- **부분 문제(Subproblem)을 어떻게 정의**할 것인가?
- $m[i, w]$가 가장 원하는 이상적인 상태라고 정의 (A 집합)
  - 가중치(weight)가 가장 적으면서, 가치(value)가 가장 높은 상태
- 물건을 한 개도 안 고른다면?

→ $m[0, w] = 0$

- 물건을 넣다가, 현재 물건이 이전 물건의 가중치를 초과한다면?

→ $m[i, w] = m[i-1, w]$ 단, $w_i > w$

- 가장 가치 있는 물건을 넣어야 하기 때문에, 비교해서 **큰 값을 선택**

→ $m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i)$ 단, $w_i \le w$

Knapsack Problem: 암호학

- Knapsack Problem을 암호학에 적용할 경우
  → **Knapsack cryptosystems**
- Knapsack Problem이 NP-Complete로 분류되므로 안전함
  - Merkle-Hellman 공개키 암호화에 사용됨
  - 안전하지만 고전 컴퓨터 상에서는 사용하지 않음
- **양자 컴퓨터 시대에서 사용이 가능할 것으로 예상**
  - 양자 컴퓨터가 연산하기 어려운 분류이기 때문
  - Large integer factorization (X)
  - Discrete logarithms (X)