5. Greedy algoritm

5.0 Basics

5.1 Minimum spanning trees

5.2 Knapsack problem

5.3 Job sequencing with deadline

5.4 Optimal merge patterns

5.5 Huffman encoding

5.2 Knapsack problem

- Problem:
  - We are given n objects and a knapsack.
  - Object i has a weight w_i and a profit p_i, and the knapsack has a capacity M.
  - If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then the profit of $p_i x_i$ is earned.

5.2 Knapsack problem

- Problem:
  - The objective is to obtain a filling of the knapsack that maximizes the total profit earned.

$$\text{Maximize} \sum_{1 \le i \le n} p_i x_i \text{ subject to } \sum_{1 \le i \le n} w_i x_i \le M$$

  - The solution is a set (x_1, x_2, …, x_n)

5.2 Knapsack problem

- Example
  - n = 3, M = 20, $(p_1, p_2, p_3) = (25, 24, 15)$, $(w_1, w_2, w_3) = (18, 15, 10)$.
  - What is the solution (x_1, x_2, x_3) ?

5.2 Knapsack problem

- Solution strategy
  - At each step, we include that object which has the maximum profit per unit of capacity.
  - In the order of the ratio p_i / w_i.

5.2 Knapsack problem

- Knapsack algorithm

```
GREEDY_KNAPSACK ( int P[], int W[], int M, int X[], int n )
{
    sort P & W such that P[i]/W[i] >= P[i+1]/W[i+1];
    X ← NULL;
    cu ← M;          //     the remaining knapsack capacity
    for ( i = 0 to n )
        if ( W[i] > cu )
            break;
        X[i] = 1;
        cu ← cu – W[i];
    if ( i <= n )
        X[i] ← cu / W[i];
}
```

5.2 Knapsack problem

다음 설명 중 옳은 것을 모두 고르시오.

(a) Knapsack problem의 시간 복잡도는 O(n log n)이다 (n: 물체의 수).

(b) Knapsack problem의 공간 복잡도는 O(n)이다.

(c) Greedy algorithm은 Knapsack problem에서 어떤 경우에도 가장 많은 이익을 얻을 수 있도록 한다.

(d) 물체를 나눌 수 없는 경우에도 (x_i = 0 or 1만 가능) greedy algorithm은 Knapsack problem의 최대 이익을 보장한다.