

0/1 Knapsack Problem

- **Definition**
 - Input:
 - Vector p of n profits and a vector w of n weights and a total weight M
 - Output:
 - An optimal 0/1 vector of length x , where $x_i = 1$ indicates object i is in knapsack and $x_i = 0$ indicates object i is not in the knapsack
 - Maximizes the profit subject to the constraint that the total weight of the included objects is at most M

Notation

- $\text{Knap}(i, j, y)$
 - Value of optimal solution with respect to objects i through j using at most capacity y
 - $\text{Knap}(1, n, M)$
 - Value of the solution to our problem
- $g_j(y) = \text{Knap}(j+1, n, y)$
- $f_j(y) = \text{Knap}(1, j, y)$

Approaches

- **Forward**

- $$\text{Knap}(1, n, M) = \max \{ \text{Knap}(2, n, M), p_1 + \text{Knap}(2, n, M - w_1) \}$$
$$= g_0(M) = \max \{ g_1(M), p_1 + g_1(M - w_1) \}$$

- **Backward**

- $$\text{Knap}(1, n, M) = \max \{ \text{Knap}(1, n-1, M), p_n + \text{Knap}(1, n-1, M - w_n) \}$$
$$= f_n(M) = \max \{ f_{n-1}(M), p_n + f_{n-1}(M - w_n) \}$$

Backward Approach

- **Observe**

- $f_i(y) = \max \{f_{i-1}(y) , p_i + f_{i-1}(y - w_i)\}$

- **Also**

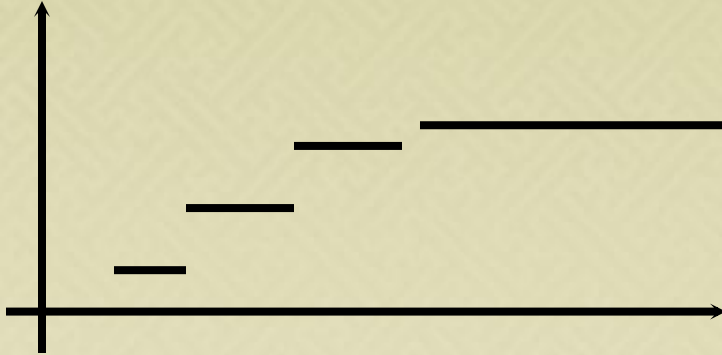
- $f_0(y) = 0$ for $y \geq 0$

- $f_0(y) = -\infty$ for $y < 0$

- Can compute f_1 from f_0 , f_2 from f_1 , and so on ...

Backward Approach

- $f_i()$ is an ascending step function



- $f_i()$ is specified by the jump pairs (p, w) where $f_i(y)$ changes
- $f_i()$ is buildable from $f_{i-1}()$
- Let S^i be the jump pair set for $f_i()$
 - $S^i = \{ (f_i(y_j), y_j) \mid y_j \text{ is a point of jumping (discontinuity)} \}$

Algorithm Details

- **Procedures**
 - Keep jump pairs sorted by associated profit
 - Will also be sorted by weight
 - $(0, 0)$ is the first jump pair
- **Compute**
 - Jump pair sets increasing order of f_i
 - Last pair in S^n tells you the value of the optimal solution

Algorithm

```
S[0] = {(0,0)}  
for (i = 1; i < n; ++i) {  
    Si1 = {(P, W) | (P - p[i], W - w[i]) | S[i-1] && W ≤ M}  
    S[i] = MergePurge(S[i-1], Si1)  
}  
for (i = n; n >= 1; --i) {  
    (P1, W1) = maxp { (P,W) | (P,W) in S[i-1]}  
    (P2, W2) = maxw { (P + p[i], W + w[i])  
                    | (P,W) in S[i-1], W + w[i] ≤ M}  
    if (P1 > P2)  
        x[i] = 0  
    else  
        x[i] = 1  
}
```

Implementation

- **Running time**
 - $|S[i]| \leq |S[i-1]|$
 - $|S[i]| \leq 2|S[i-1]|$
- **If $S[i]$'s are in sorted order then merging can be done in linear time**

$$\sum_{i=1}^n |S[i]| = \sum_{i=1}^n 2^i = 2^n - 1$$