Optimal Energy Conservation Measure (ECM) Selection

문제 정의 및 기본 특성

문제 정의

ECM	명칭	절감량	비용[백만]	탄소감축량
반송설비	고효율펌프교체	100	135	42.4
건축	바닥단열개선	100	135	42.4
	지붕단열개선	150	186	63.6
조명	고효율조명교체	100	135	42.4
	LED조명교체	150	186	63.6
	디밍제어	190	225	80.6
공조설비	외기유입량축소	100	135	42.4
	외기유입량축소	150	186	63.6
	고효율FCU교체	190	225	80.6
	고효율EHP교체	160	205	67.8
	효율VRF교체	200	242	84.8
냉방	스크류식냉동기	100	135	42.4
	냉각탑냉동기	150	186	63.6
난방	입형보일러	100	135	42.4
	주제철보일러	150	135	63.6
	고효율증기보일러	190	225	80.6
신재생	태양열	100	135	42.4
	지열	150	186	63.6
	태양광	190	225	80.6
	풍력	220	278	93.3

ECM	명칭	절감량	비용[백만]	탄소감축량
난방	주제철보일러	150	135	63.6
조명	디밍제어	190	225	80.6
반송설비	고효율펌프교체	100	135	42.4
합계		440	495	186.6

혹은 이득의 하한(lower bound)을 유지하며
비용을 최소화

문제 정의

- **입력:**
  - m개의 클래스(class)
  - 클래스 i에는 n_i개의 아이템, $i=1,2,\ldots,m$
  - $item_{i,j}$ 는 클래스 i에 속한 j번째 아이템
  - 각각의 $item_{i,j}$ 는 비용(cost) c_{ij} 와 이득(profit) p_{ij}를 가짐
  - 총비용의 상한 C
- **제약조건:**
  - 각 클래스에서 최대 1개의 아이템을 선택
  - 총 비용은 C를 초과할 수 없음
- **목적:**
  - 이득의 합이 최대가 되는 아이템 집합을 선택

Multiple Choice Knapsack Problem

Given $\{c_{ij}, p_{ij} \mid i = 1, \ldots, m, j = 1, \ldots, n_i\}$ and the cost bound C,

find $x_{OPT} = \text{argmax}_x \sum_{i,j} p_{ij} x_{ij}$

subject to

$$\sum_{i,j} c_{ij} x_{ij} \leq C$$

$$\sum_{j} x_{ij} \leq 1, \text{for } i = 1, \ldots m,$$

$$x_{ij} \in \{0, 1\}, \text{for } i = 1, \ldots, m, j = 1, \ldots, n_i$$

- 모든 클래스 i에 대해서 $n_i = 1$이면 **0-1 Knapsack 문제**
- 0-1 Knapsack 문제의 일반화이므로 당연히 NP-hard ← 최적해를 구하는 다항시간 알고리즘을 기대할 수 없음

Linear Relaxation: LMCKP

Given $\{c_{ij}, p_{ij} \mid i = 1, \ldots, m, j = 1, \ldots, n_i\}$ and the cost bound C,

find $x_{OPT} = \text{argmax}_x \sum_{i,j} p_{ij} x_{ij}$

subject to

$$\sum_{i,j} c_{ij} x_{ij} \leq C$$

$$\sum_{j} x_{ij} \leq 1, \text{for } i = 1, \ldots m,$$

$$\cancel{x_{ij} \in \{0,1\}, \text{for } i = 1, \ldots, m, j = 1, \ldots, n_i}$$

$$0 \leq x_{ij} \leq 1, \text{for } i = 1, \ldots, m, j = 1, \ldots, n_i$$

- **다항시간 greedy 알고리즘**
  - D. Pisinger, “A minimal algorithm for the multiple-choice knapsack problem,” Technical Report 94/25, DIKU, Univ. of Copenhagen, Denmark. 1994.
- **$OPT_{0\text{-}1} \leqq OPT_{fractional}$** ← Relaxed solution은 0-1 solution의 upper bound

상태공간트리의 탐색
Searching State Space Tree

State Space Tree

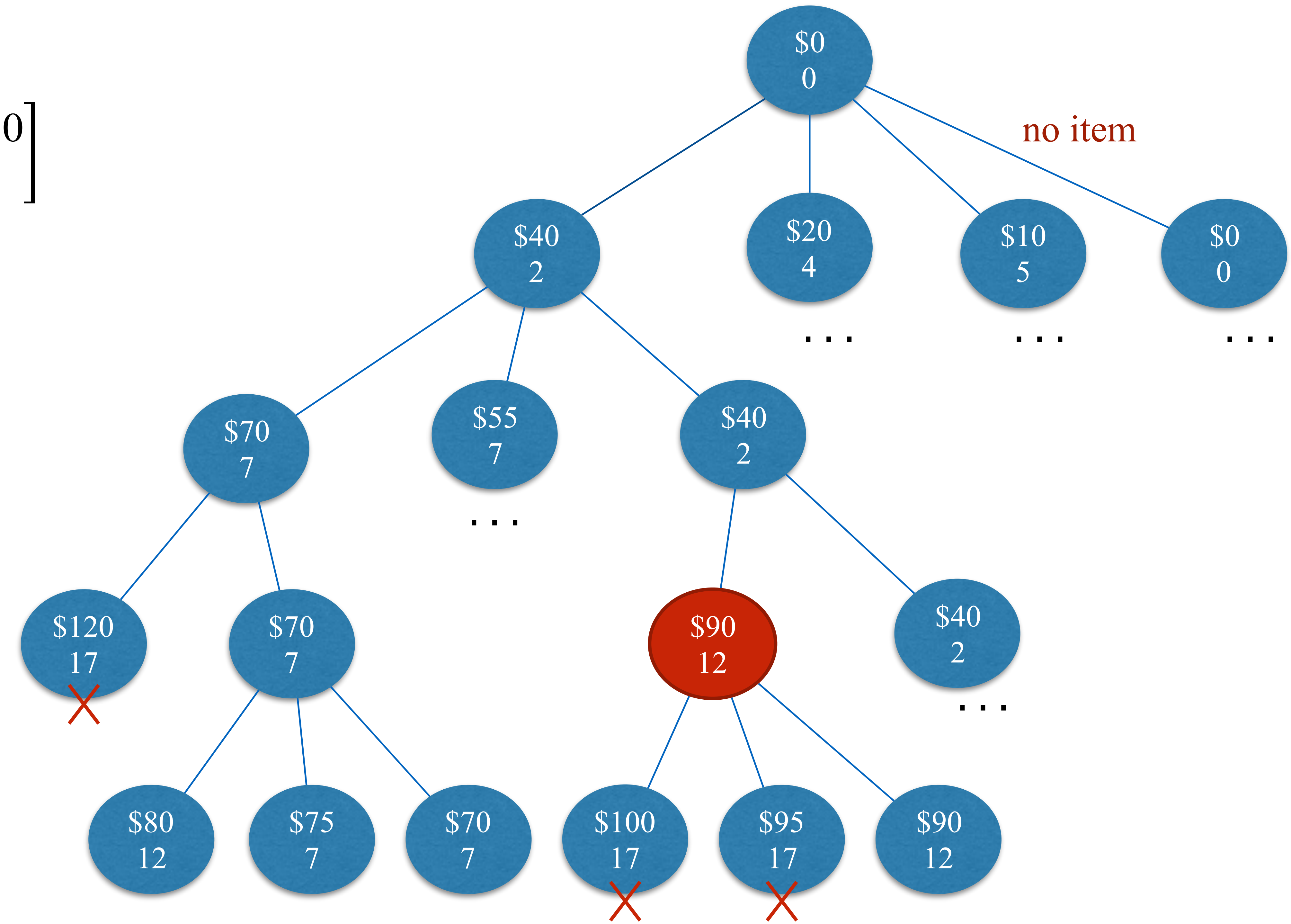
$C = 16$

$$Class1 \begin{bmatrix} \$40 & \$20 & \$10 \\ 2 & 4 & 5 \end{bmatrix}$$

$$Class2 \begin{bmatrix} \$30 & \$15 \\ 5 & 5 \end{bmatrix}$$

$$Class3 \begin{bmatrix} \$50 \\ 10 \end{bmatrix}$$

$$Class4 \begin{bmatrix} \$10 & \$5 \\ 5 & 5 \end{bmatrix}$$



트리의 크기: $O(\prod_{i=1}^{m} n_i)$

Pruning

- 상태 공간 트리의 모든 노드를 탐색할 필요는 없음
  - 가지치기(pruning)
  - 예를 들어 cost bound를 초과하면 그 노드의 subtree는 더 이상 탐색할 필요 없음
- 탐색과정에서 현재까지 발견한 최선의 해는 최적 해의 하한(lower bound)
- 각 노드에 대해서 달성 가능한 해의 상한(upper bound)을 계산
- 만약 어떤 노드의 달성 가능한 상한이 최적해의 하한에 미달하면 pruning
- Fractional 버전의 최적해는 최적해의 상한이 됨

Pruning

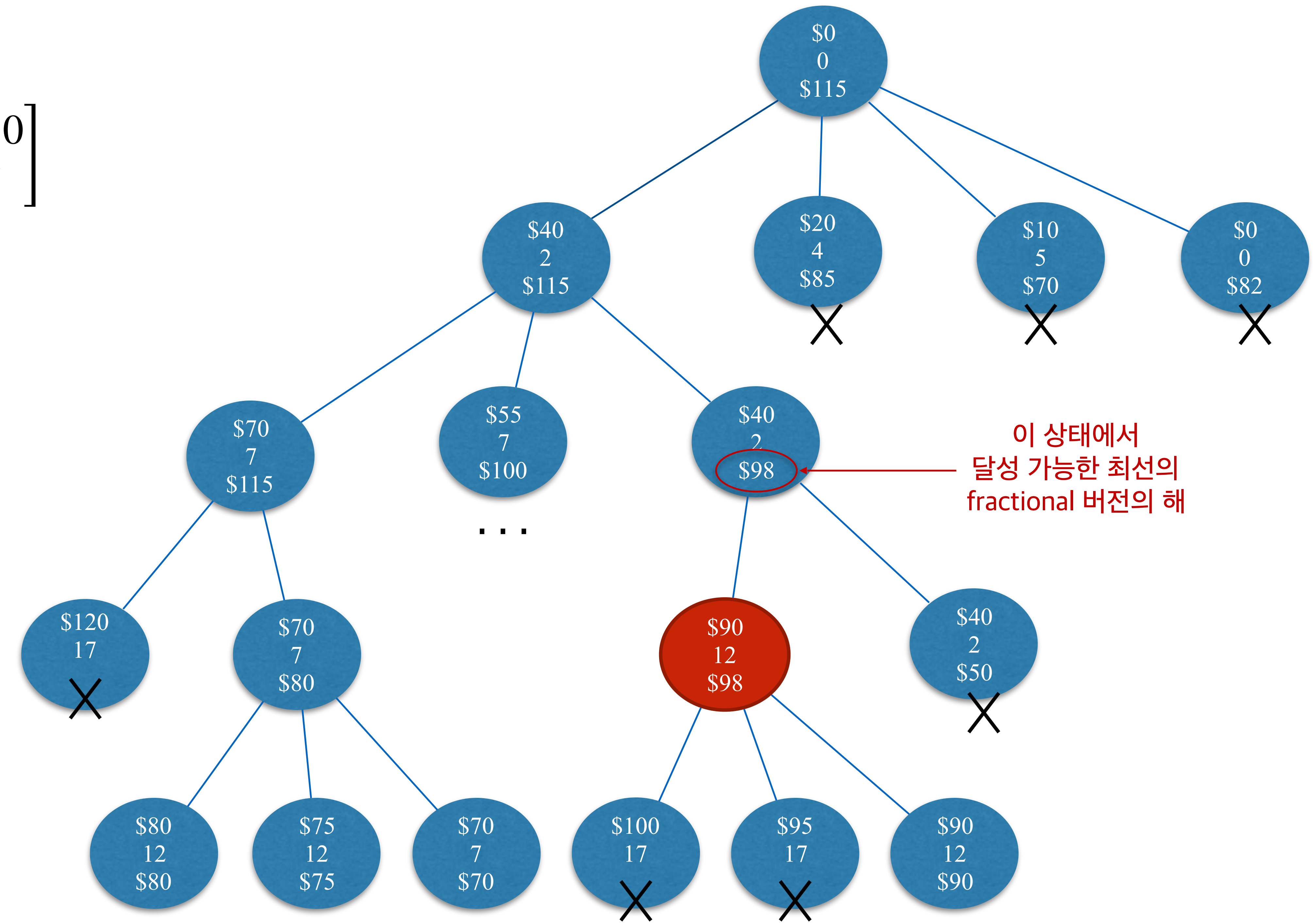
$$C = 16$$

$$Class1 \begin{bmatrix} \$40 & \$20 & \$10 \\ 2 & 4 & 5 \end{bmatrix}$$

$$class2 \begin{bmatrix} \$30 & \$15 \\ 5 & 5 \end{bmatrix}$$

$$class3 \begin{bmatrix} \$50 \\ 10 \end{bmatrix}$$

$$class4 \begin{bmatrix} \$10 & \$5 \\ 5 & 5 \end{bmatrix}$$



```
/* 각 카테고리 마다 cost와 profit이 각각 0인 dummy item을 하나씩 추가하였음 */

int maxprofit = 0;
bool selection[MAX];  /* initialized to false */

void mckp(int index, int profit, int cost) {
    /* class 0, 1,...,index-1 have already been decided */
    if (cost <= C && profit > maxProfit) {
        maxProfit = profit;
        backup current selection;
    }

    if (promising(index, profit, cost))  {        not promising하면 pruning
        for (int j = 0; j<n[index]; j++) {       dummy item도 포함하여 iterate
            selection[index] = j;
            mckp(index + 1, profit + p[index][j], cost + c[index][j]);
        }
    }
}
```

Promising Test

```
bool promising(int index, int profit, int cost) {
    if (cost >= C)  return false;
    double bound = profit;
    sort all items by the non-increasing order of profit/cost ratio;
    for each item [c, p] in the sorted order {
        if [c, b] belongs to classes {0,…,index-1}
            ignore it;
        if (cost + c > C) {
            bound += (C-cost)*b/c;
            break;
        }
        cost += c;
        bound += p;
    }
    return bound > maxprofit;
}
```

Relaxed
Knapsack의 해를 계산
(Not the optimal algorithm)

- 상태공간트리 탐색 알고리즘의 성능은 얼마나 빨리 최적 해에 근접하는 하한을 찾느냐에 달려있다.
- 상한이 높은 노드의 부트리(subtree)를 우선 탐색
- 우선순위 큐(priority queue)를 이용하여 구현

Dynamic Programming

순환식

- 각 클래스에는 비용 0, 이득 0인 dummy 아이템이 하나씩 있다고 가정
- **OPT(i, w)**: 허용 비용이 **w**일 때 클래스 **{1,2,...,i}**에 속한 아이템들만으로 얻을 수 있는 최대 이득
- 목적: **OPT(m, C)**

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ -\infty & \text{if } w < 0 \\ \max_{j=0}^{n_j}\{p_{i,j} + OPT(i-1, w - c_{i,j})\} & \text{otherwise} \end{cases}$$

Dynamic Programming

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ -\infty & \text{if } w < 0 \\ \max_{j=0}^{n_j}\{p_{i,j} + OPT(i-1, w-c_{i,j})\} & \text{otherwise} \end{cases}$$

OPT	1	2	⋯						w	⋯	C-1	C
0	0	0	0	0	0	0	0	0	0	0	0	0
1												
⋮												
i-1				(i-1,w-c_{i2})		(i-1,w-c_{i1})			(i-1,w)			
i									(i,w)			
⋮												
m												**Goal**

depends

row major order로 계산

```
/* dummy item 불필요, 클래스 인덱스는 1,…,M, 아이템 인덱스는 0,...,ni */

int mckp_dp() {
    int [][] opt = new int [M+1][C+1];
     for (int i=0; i<=M; i++) {     // i=0 means “no category to select”
        for (int w = 1; w<=C; w++) {
          if (i==0)
             opt[i][w] = 0, continue;

          opt[i][w] = opt[i-1][w];
          for (int j = 0; j<n[i]; j++) {
             if (c[i][j] <= w && p[i][j]+opt[i-1][w-c[i][j]] > opt[i][w])
                opt[i][w] = p[i][j] + opt[i-1][w-c[i][j]];
          }
       }
    }
    return opt[M][C];
}
```

시간 및 공간복잡도

- 의사다항(pseudo-polynomial) 시간 복잡도: O(NC), 여기서 $N = \sum n_i$
- C가 크면 시간 복잡도만이 아니라 공간 복잡도 O(mC)도 문제가 될 수 있음
- 최적해를 구하기 위해서 mC개의 모든 값이 필요한 것은 아님
- Memoization 기법을 이용하여 필요한 값만 계산하도록 구현 가능

Comparison

Comparison

m (n_{max}=8)	c_{max}	Backtrack (with Pruning)	Branch&Bound	Memoization	Dynamic Programming
20	10,000	0.01 ~ 10sec unstable	0.1~5Min very unstable	Implementation dependent, Not good when using HashMap or TreeMap	0.03 ~ 0.05
	100,000				0.1 ~ 0.2
	1,000,000				1.42 ~ 1.77
30	10,000	0.083sec ~ 1Min unstable	very unstable Often Out of Memory		0.049 ~ 0.081
	100,000				0.273 ~ 0.411
	1,000,000				2.97 ~ 3.341
40	10,000	Sometimes very fast, but mostly Timeout	Timeout Out of Memory		0.083 ~ 0.095
	100,000				0.613 ~ 0.744
	1,000,000				4.183 ~ 4.835
50	10,000				0.105 ~ 0.137
	100,000				0.756 ~ 0.823
	1,000,000				7.773 ~ 9.231
100	10,000				0.342 ~ 0.439
	100,000				2.74 ~ 3.627
	1,000,000				Out of Memory

$$C = \sum c_i / 16$$

Bounded Profit Version

- **획득해야할 이득의 하한을 주고 최소 비용을 구하는 문제** (dummy item 있다고 가정)

$$OPT(i,p) = \begin{cases} 0 & \text{if } i = 0 \text{ and } p \leq 0 \\ \infty & \text{if } i = 0 \text{ and } p > 0 \\ \min_{j=1}^{n_j}\{c_{i,j} + OPT(i-1, p - p_{i,j})\} & \text{otherwise} \end{cases}$$

Thank you.