알고리즘분석 실습 자료

11주차

Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

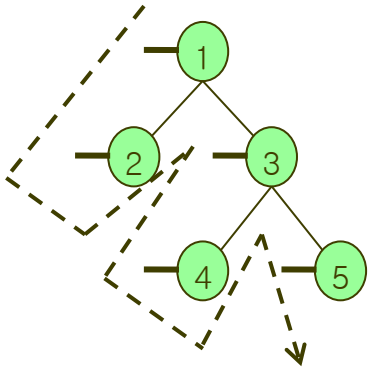
5장 되추적 (Backtracking)

실습프로그램

- ✓ 부분집합의 합
- ✓ m-coloring

트리 방문(tree traversal)

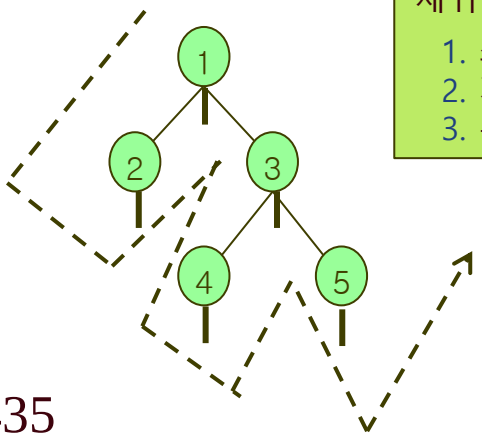
1. preorder



12345

재귀적으로
1. 자신 방문
2. 좌측 방문
3. 우측 방문

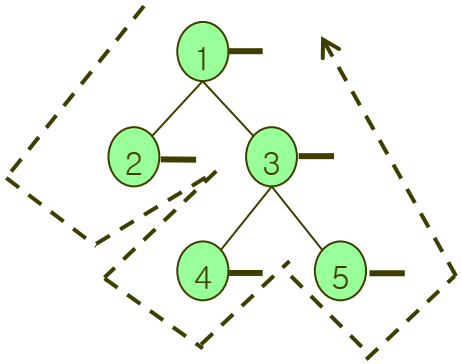
2. inorder



21435

재귀적으로
1. 좌측 방문
2. 자신 방문
3. 우측 방문

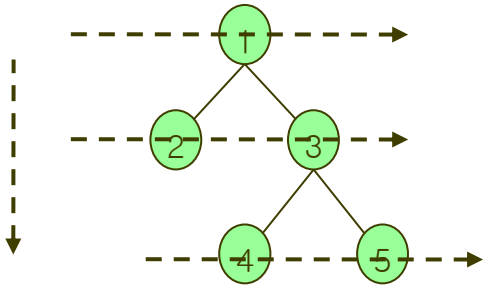
3. postorder



24531

재귀적으로
1. 좌측 방문
2. 우측 방문
3. 자신 방문

4. level order



12345

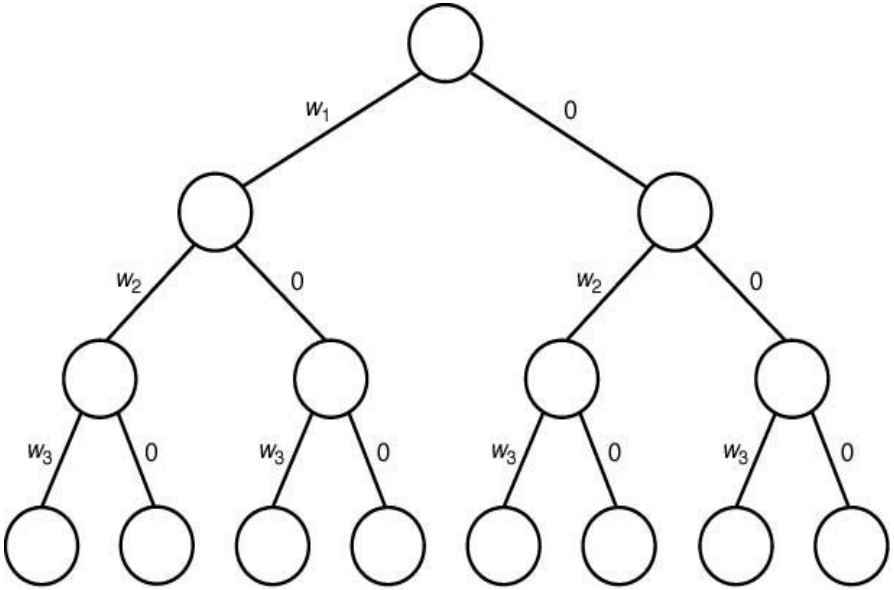
부분집합의 합 구하기(sum of subsets problem)

- n개의 item을 이용하여 item 들의 무게의 합이 W가 되는 부분집합을 구한다.

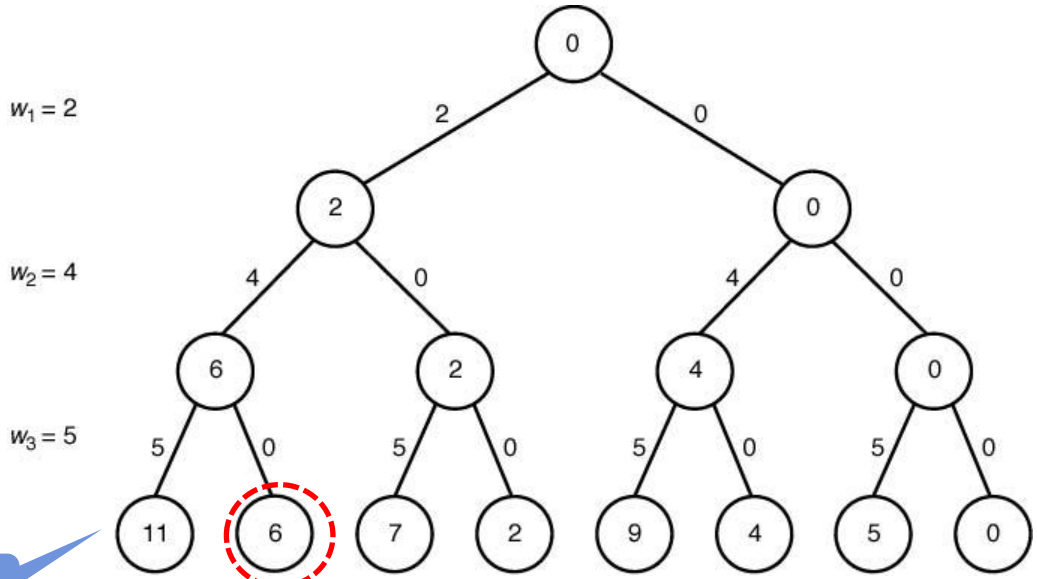
$$\sum_{i=1}^{n} w_i x_i = W$$

$$x_i = 0 \quad \text{or} \quad 1, \quad for \;\; i = 1, n$$

- For S={1,4,6,8}, select items so that sum of the subset is 5.



n=3, w_1=2, w_2=4, w_3=5, W=6



누적 무게

soution

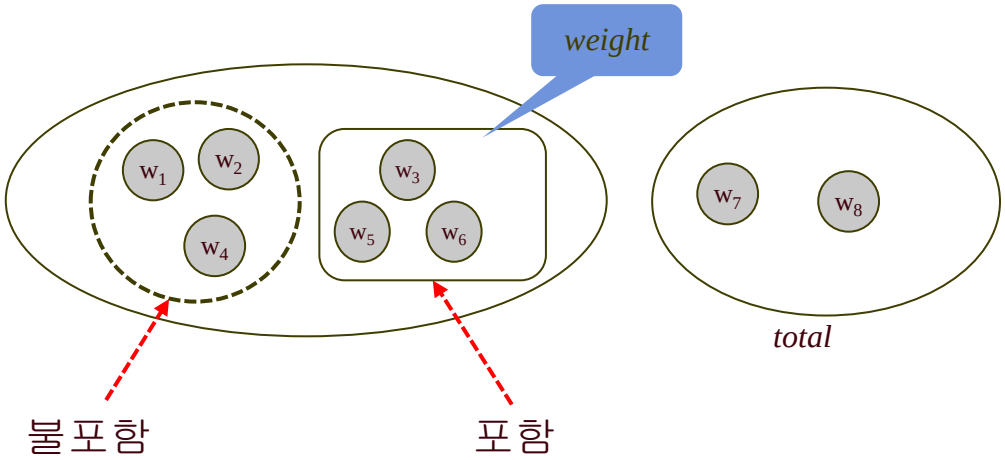
- 무게가 증가하는 순으로 데이터를 정렬→ 유망하지 않은지를 쉽게 판단할 수 있음. w_{i+1} 는 i수준에서 남아있는 가장 가벼운 아이템의 무게. w_{i+1}를 넣을 수 없으면 i+1 이후는 고려할 필요 없음.

- *weight*: 수준 i 의 마디까지 포함된 무게의 합

- *total*: 남아 있는 아이템의 무게의 총 합

- $weight + w_{i+1} > W$ *(if* weight $\neq W$*)* or
  $weight + total < W$ 이면 유망하지 않다.

At level 6,



n=4, W=13, w_1=3, w_2=4, w_3=5, w_4=6

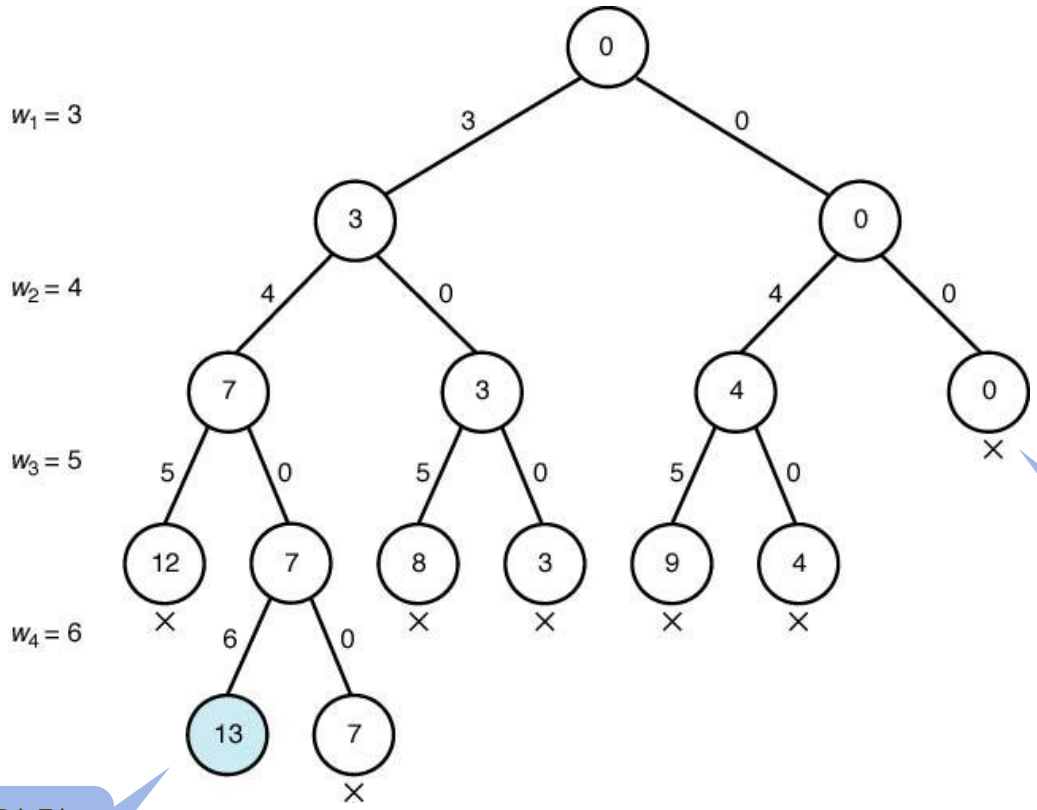


그림 5.9 가지친 상태공간트리. 총 15개의 마디 존재

```
void sum_of_subsets(index i, int weight, int total)
{
   if(promising(i))
       if (weight == W)
           cout << include[1] through include[i];
       else{
            include[i+1]="yes";
            sum_of_subsets(i+1, weight+w[i+1],total-w[i+1]);
            include[i+1]="no";
            sum_of_subsets(i+1, weight,total-w[i+1]);
            }
}
bool promising(index i){
  return((weight+total>=W)&&(weight == W ||weight+w[i+1]<=W);
}
```

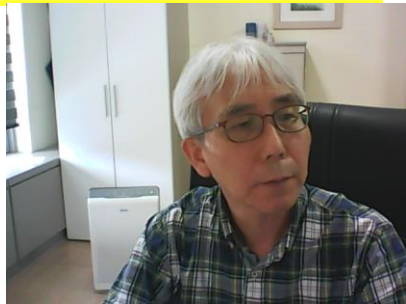
w[i+1] 포함

w[i+1] 불포함

not

- $weight + w_{i+1} > W$ (*if* weight $\neq W$) or $weight + total < W$ 이면 유망하지 않다.

- 최상위 호출: sum_of_subsets(0, 0,total)



[실습프로그램] 부분집합의 합

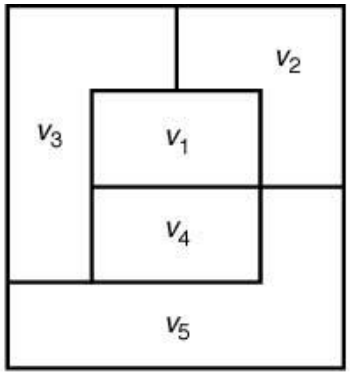
```
def promising(i,weight, total):

     구현


def s_s(i, weight, total, include):




     sum of subsets 구현



n=4
w=[1,2,4,6]
W=6
print("items =",w, "W =", W)
include = n*[0]
total=0
for k in w:
    total+=k
s_s(-1,0,total,include)
```

```
items = [1, 2, 4, 6] W = 6
sol [0, 1, 1, 0]
sol [0, 0, 0, 1]
>>>
```

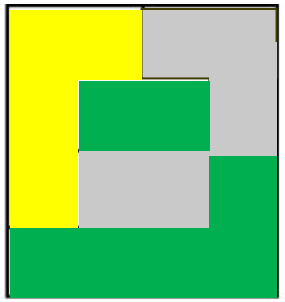
[연습문제] 부분집합의 합 프로그램을 다음 데이터를 이용해 수행

$S=\{1,2,\ldots,100\}$, $W=365$

그래프 색칠하기(graph coloring)

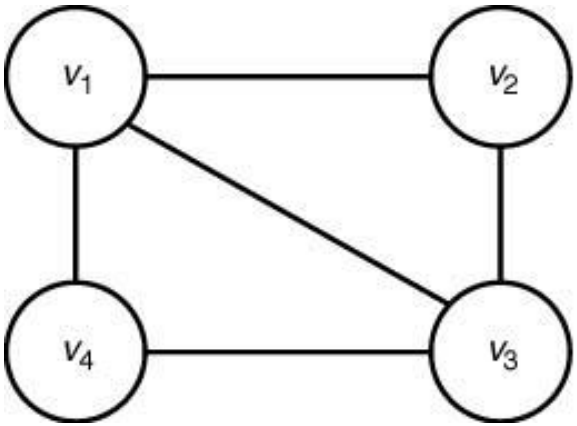


- 지도 칠하기(map coloring): 인접하는 지역을 구분하기 위해 색깔을 할당하는 문제
- 4개의 색깔이면 충분



m coloring problem

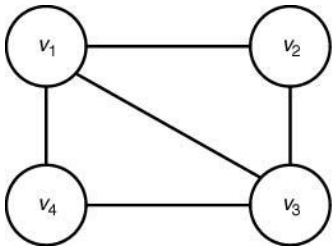
- **지도에 *m*가지 색으로 색칠하는 문제**
  - ✓ *m*개의 색을 가지고, 인접한 지역이 같은 색이 되지 않도록 지도에 색칠하는 문제

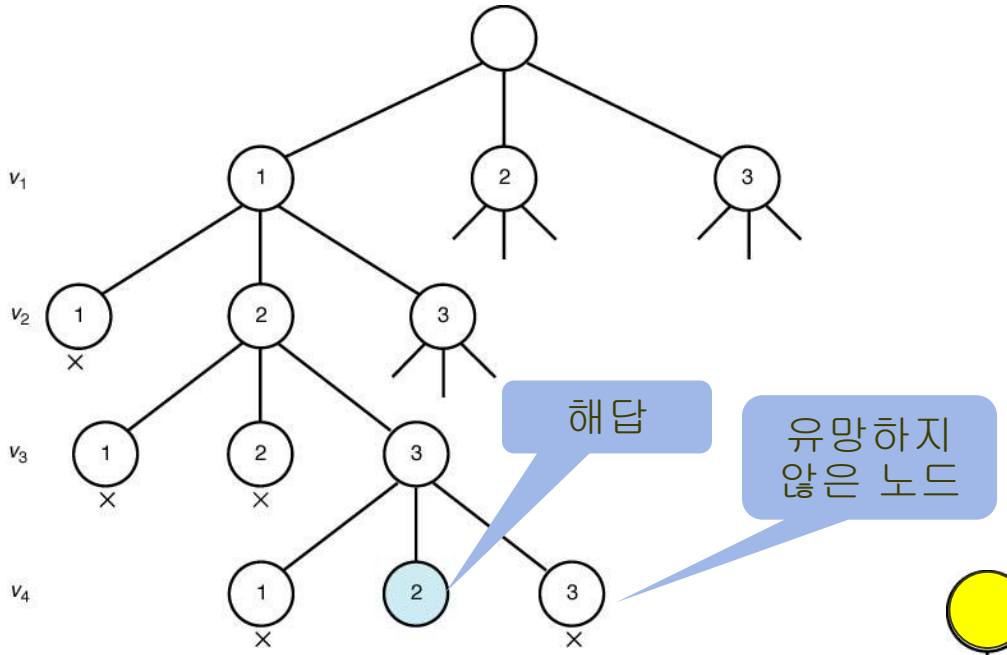


- 이 그래프에서 두 가지 색으로 문제를 풀기는 불가능하다.
- 세 가지 색을 사용하면 총 6가지의 해답을 얻을 수 있다.

그래프 색칠하기 되추적 해법

3종류 색깔 사용할 경우의 해 찾기





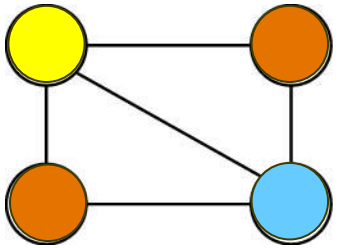


그림 5.12 가지친 상태공간트리

Fig 5.12 A portion of the pruned state space tree produced using backtracking to do a 3-coloring of the graph in Fig.5.10

```
void m_coloring(index i) {
    int color;

   if(promising(i))
       if(i==n)
           cout << vcolor[1] through vcolor[n];
       else
            for (color=1; color<=m; color++){
                  vcolor[i+1] = color;
                  m_coloring(i+1);
            }
}
bool promising(index i) {
   index j;
   bool switch;

    switch = true;
    j=1;
    while ( j<i  && switch){
         if(W[i][j] && vcolor[i]==vcolor[j]) // W[i][j]:연결표시.  T or F
                switch = false;
         j++;
     }
     return switch;
   }
```

vcolor[i]=노드i의 color

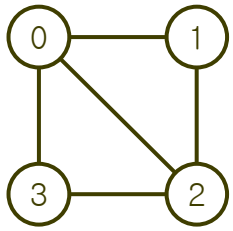
서로 인접한 것이 같은 색깔인지 확인.

연결되어 있는지

- 최상위 호출: m_coloring(0)

[실습프로그램] m-coloring

```
def color(i,vcolor):

      구현



def promising(i,vcolor):



      구현



n=4
W=[[0,1,1,1],[1,0,1,0],[1,1,0,1],[1,0,1,0]]
vcolor=n*[0]
m=3
color(-1,vcolor)
```




```
[1, 2, 3, 2]
[1, 3, 2, 3]
[2, 1, 3, 1]
[2, 3, 1, 3]
[3, 1, 2, 1]
[3, 2, 1, 2]
>>>
```