
알고리즘분석 실습 자료
12주차
Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

6장 분기한정법 (Branch-and-Bound)

실습프로그램

- ✓ 0-1 배낭문제: 분기한정 가지치기로 깊이우선검색
- ✓ 0-1 배낭문제: 분기한정 가지치기로 너비우선검색
- ✓ 0-1 배낭문제: 분기한정 가지치기로 최고우선검색
- ✓ 외판원문제

분기한정법(branch-and-bound)

- 특징:
  - ✓ 되추적 기법과 같이 상태공간트리를 구축하여 문제를 해결.
  - ✓ 최적의 해를 구하는 문제(optimization problem)에 적용할 수 있음.
  - ✓ 최적의 해를 구하기 위해서는 모든 해를 다 고려해 보아야 하므로 트리의 마디를 순회(traverse)하는 방법에 구애 받지 않음.
- 분기한정 알고리즘의 원리
  - ✓ 각 마디를 검색할 때 마다, 그 마디가 **유망(promising)**한지의 여부를 결정하기 위해서 한계값(**bound**)을 계산한다.
  - ✓ 그 한계치는 그 마디로부터 가지를 뻗어나가서(**branch**) 얻을 수 있는 해답값의 한계를 나타낸다.
  - ✓ 따라서 만약 그 한계값이 지금까지 찾은 최적의 해답값 보다 좋지 않은 경우는 더 이상 가지를 뻗어서 검색을 계속할 필요가 없으므로, 그 마디는 **유망하지 않다(nonpromising)**고 할 수 있다.

0-1 Knapsack Problem

- problem: $S = \{item_1, item_2, \ldots, item_n\}$

  w_i = weight of $item_i$

  p_i = profit of $item_i$

  W = maximum weight the knapsack can hold.

Determine a subset A of S such that $\sum_{item_i \in A} p_i$ is maximized subject to

$$\sum_{item_i \in A} w_i \le W$$

$$MAX \quad \sum_{item_i \in A} p_i$$
$$subject\ to \quad \sum_{item_i \in A} w_i \le W$$

or

$$MAX \quad \sum_{i=1}^{n} p_i x_i$$
$$subject\ to \quad \sum_{i=1}^{n} w_i x_i \le W$$
$$x_i = 0 \ \text{ or } \ 1, \quad for\ i = 1, n$$

0-1 배낭채우기: 알고리즘

- **알고리즘 스케치: 아이템은 무게당 가치가 감소하는 순서로 정렬 가정**
  - *profit* : 그 마디에 오기까지 넣었던 아이템의 값어치의 합.
  - *weight* : 그 마디에 오기까지 넣었던 아이템의 무게의 합.
  - *bound* : 마디가 수준 i에 있다고 하고, 수준 k에 있는 마디에서 총무게가 W를 넘는다고 하자. 그러면

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

이론적으로 현 상태에서 얻을 수 있는 최대 이익

$$bound = \left(profit + \sum_{j=i+1}^{k-1} p_j \right) + (W - totweight) \times \frac{p_k}{w_k}$$

처음 k-1개를 이용한 아이템의 값어치

k번째 아이템에 가용한 용량

k번째 아이템의 단위 무게당 값어치

[실습프로그램] 분기한정 가지치기로 깊이우선검색

```
def kp(i, profit, weight):
        global bestset
        global maxp

        if( weight <= W and profit >maxp):
            maxp = profit
            bestset = include[:]
# bestset을 그 순간의 include 값을 저장한다. 이후 include 일부 값이 변해도
# bestset의 값은 바뀌지 않음.

      구현
```

```
def promising(i,weight,profit):
    global maxp
```

구현

```
n=4
W=16
p=[40,30,50,10]
w=[2,5,10,5]
maxp=0
include =[0,0,0,0]
bestset = [0,0,0,0]
kp(-1,0,0)
print(maxp)
print(bestset)
```

```
90
[1, 0, 1, 0]
>>>
```

W=16, p=[40,30,50,10], w=[2,5,10,5]

분기한정 가지치기로 너비우선검색

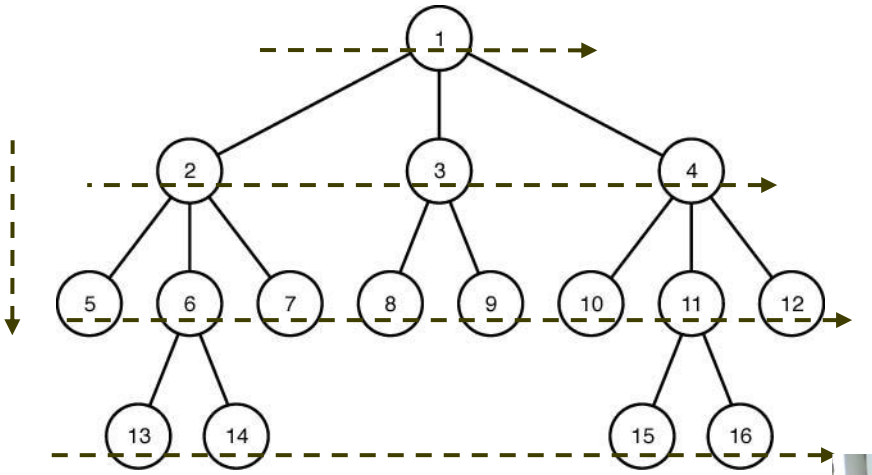
- 너비우선검색(breadth-first search)순서:

  (1) 뿌리마디를 먼저 검색한다.

  (2) 다음에 수준 1에 있는 모든 마디를 검색한다.
  (왼쪽에서 오른쪽으로)

  (3) 다음에 수준 2에 있는 모든 마디를 검색한다
  (왼쪽에서 오른쪽으로)

  (4) ...



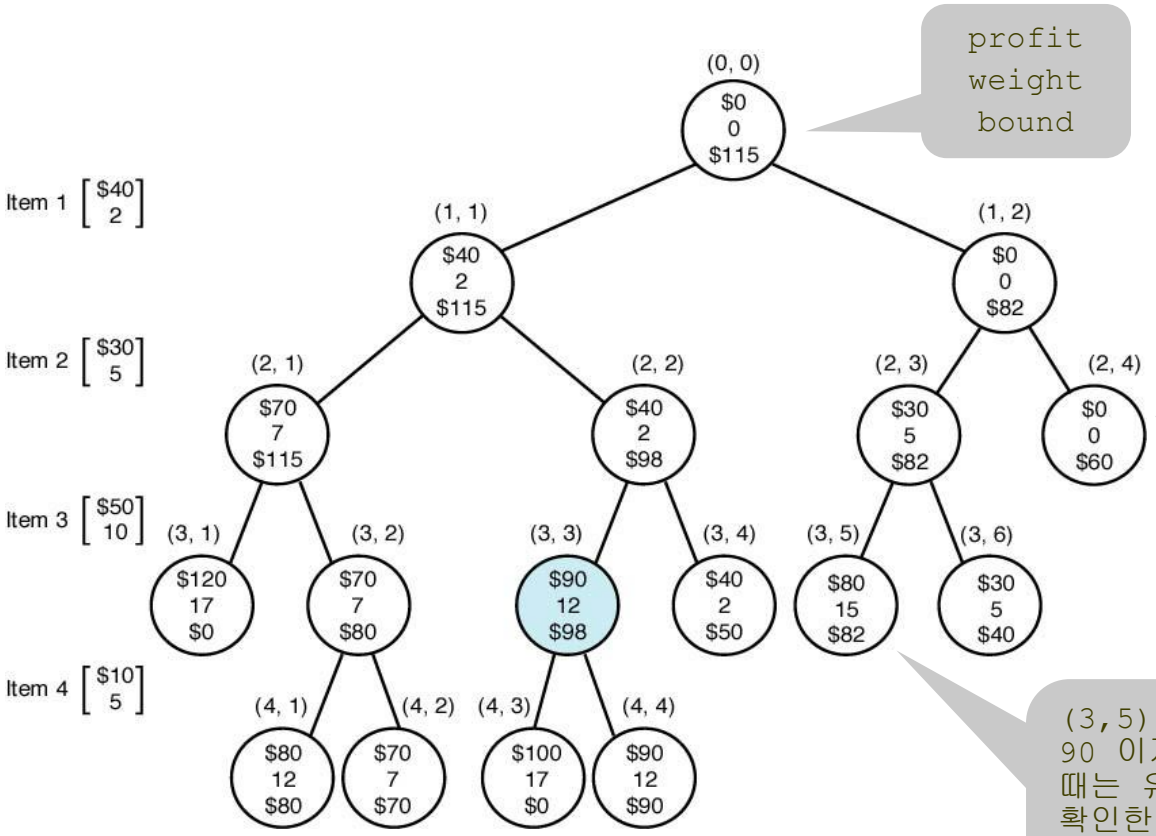
일반적인 너비우선검색 알고리즘

- **대기열(queue)**을 사용하여 구현

```
void breadth_first_search(tree T) {
        queue_of_node Q;
        node u, v;

        initialize(Q);
        v = root of T;
        visit v;
        enqueue(Q,v);
        while(!empty(Q)) {
          dequeue(Q,v);
          for(each child u of v) {
            visit u;
            enqueue(Q,u);
          }
        }
    }
```

(예 6.1)

- 앞의 예를 분기한정 가지치기 너비우선검색한 가지친 상태공간트리
- 검색하는 마디의 개수는 17. 깊이우선 알고리즘(13개)보다 좋지 않다!



i	p_i	w_i	p_i/w_i
1	$40	2	$20
2	$30	5	$6
3	$50	10	$5
4	$10	5	$2

W=16

(2,4)를 방문할 때는 이미 maxprofit이 70 이므로, (2,4)에서 비유망(60<70). (2,4)의 자식을 queue에 안 넣음

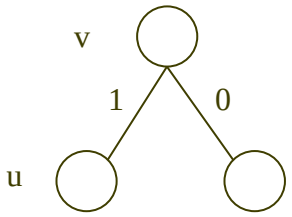
(3,5)를 방문할 때는 이미 maxprofit이 90 이지만, (2,3)에서 queue에 삽입할 때는 유망(70<82)하였으므로, 이 마디를 확인한다.

→ maxprofit은 자식마디를 실제 방문할 때 변할 수 있음.

분기한정 너비우선검색 알고리즘

```
void breadth_first_branch_and_bound(state_space_tree T, number& best) {
        queue_of_node Q;
        node u, v;

        initialize(Q);                          // Q는 빈 대기열로 초기화
        v = root of T;                          //뿌리마디를 방문
        enqueue(Q,v);
        best = value(v);
        while(!empty(Q)) {
            dequeue(Q,v);
            for(each child u of v) {            // 각 자식마디를 방문
              if(value(u) is better than best)
                best = value(u);
              if(bound(u) is better than best)
                enqueue(Q,u);
            }
        }
  }
```

```
void knapsack2(int n, const int p[], const int w[], int W, int& maxprofit){
    queue_of_node Q;
    node u,v;

    initialize(Q);
    v.level=0; v.profit=0; v.weight=0;
    maxprofit=0;
    enqueue(Q,v);
    while(!empty(Q)){
        dequeue(Q,v);
        u.level = v.level + 1;

        u.weight = v.weight + w[u.level];
        u.profit = v.profit + p[u.level];
α       if(u.weight <= W && u.profit > maxprofit)
            maxprofit = u.profit;
        if(bound(u) > maxprofit)
            enqueue(Q,u);

        u.weight = v.weight;
        u.profit = v.profit;
        if(bound(u) > maxprofit)
            enqueue(Q,u);
        }
}
```



u를 v의 자식마디로 만듬. u를 다음 아이템을 포함하는 자식마디로 놓음

u를 v의 자식마디로 만듬. u를 다음 아이템을 포함하지 않는 자식마디로 놓음. profit과 weight의 변화가 없으므로, α 부분 없음.

✓ dequeue(Q,v)후 v가 아직 유망한 지 점검가능. 여기서는 생략
✓ 만일 dequeue 후 유망점검이 있다면 (3,5) 이후는 바로 대상이 아닌 것으로 판단가능

[실습프로그램] 분기한정 가지치기로 너비우선검색

```
import queue

class Node:
        def __init__(self,level,weight, profit, include):
                self.level = level
                self.weight = weight
                self.profit = profit
                self.include = include
```

```
def kp_BFS():
        global maxProfit
        global bestset

        q = queue.Queue()

        구현


def compBound(u):


     구현


n=4
W=16
p=[40,30,50,10]
w=[2,5,10,5]
include=[0]*n
maxProfit =0
bestset=n*[0]
kp_BFS()
print(bestset)
```

W=16, p=[40,30,50,10],
w=[2,5,10,5]

```
[1, 0, 1]
>>>
```

[실습프로그램] 분기한정 가지치기로 최고우선검색

```
import queue

class Node:
        def __init__(self,level,weight, profit, bound, include):
                self.level = level
                self.weight = weight
                self.profit = profit
                self.bound = bound
                self.include = include
        def __cmp__(self, other):
                return cmp(self.bound, other.bound)
```

```
def kp_Best_FS():
        global maxProfit
        global bestset
        temp = n*[0]
        v = Node(-1,0,0, 0.0,temp)
        q = queue.PriorityQueue()


     구현

def compBound(u):
        if u.weight >=W:
                return 0
        else:
                 구현



# heap이 minheap이라 bound를 계산하여 -를 하여 리턴한다. 비교를 < maxProfit으로 수행한다.
n=4
W=16
p=[40,30,50,10]
w=[2,5,10,5]
include=[0]*n
maxProfit =0
bestset=n*[0]
kp_Best_FS()
print(bestset)
print(maxProfit)
```

W=16, p=[40,30,50,10],
w=[2,5,10,5]

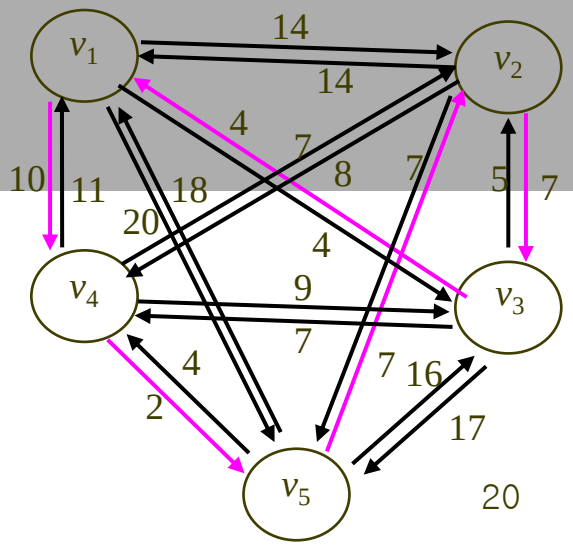
```
[1, 0, 1, 0]
-90
>>>
```

[실습프로그램] 외판원 문제

```
import queue

class Node:
        def __init__(self,bound ,path):
                self.bound= bound
                self.path = path
#        def __cmp__(self, other):
#                return cmp(self.bound, other.bound)

def TSP_Best_FS():
   global minLength
   global optTour
   path=[start]
   v = Node(0,path)

   v.bound = compBound(v)
   q = queue.PriorityQueue()
   q.put((v.bound,v.path))
   while not q.empty():
       v.bound, v.path= q.get()
       if(v.bound < minLength):
          if len(v.path)==n-1:
             A = v.path[:]
             B = [x for x in V if x not in A]
             C=[x for x in B if x !=start]
             if compLength(v.path)+W[v.path[n-2]][C[0]]+W[C[0]][start] < minLength:
                  minLength = compLength(v.path)+W[v.path[n-2]][C[0]]+W[C[0]][start]
                  optTour = v.path[:]+[C[0]]
          else:
             C=[ x for x in V if x not in v.path]
             for x in C:
                u = Node(0,v.path+[x])
                u.bound = compBound(u)
                print("path bound ", u.path, u.bound)
                if u.bound < minLength:
                  q.put((u.bound,u.path))
```

```
def compBound(u):
    global start
#  if len(u.path)==n-1:
#      print(" uPP ", u.path, u.path[n-2],compLength(u.path), W[u.path[n-2]][start])
#      return compLength(u.path)+W[u.path[n-2]][start]
#  else:
    tbound = 0
    A = u.path[:]
    B = [x for x in A if x != start]
    C= [x for x in V if x not in A]
    temp1=0
    tMin=10000
    for y in C:
        if W[u.path [ len(u.path)-1]][y] < tMin:
          tMin = W[u.path [ len(u.path)-1]][y]
    tbound += tMin
    for y in C:
       tMin = 10000
       D = [ x for x in C if x != y]
       D.append(start)
       for z in D:
         if W[y][z]<tMin:
           tMin=W[y][z]
       tbound += tMin
    return tbound+compLength(u.path)
```

```
def compLength(a):
  length =0
  for x in range (0,len(a)-1):
    length += W[a[x]][a[x+1]]
  return length

start =0
n=5
W=[[0,14,4,10,20],[14,0,7,8,7],[4,5,0,7,16],
  [11,7,9,0,2],[18,7,17,4,0]]
V=list()
for x in range(0,n):
  V.append(x)
optTour=list()
print(V)
minLength=10000
TSP_Best_FS()
print(minLength)
print(optTour)
```




```
[0, 1, 2, 3, 4]
path bound  [0, 1] 31
path bound  [0, 2] 22
path bound  [0, 3] 30
path bound  [0, 4] 42
path bound  [0, 2, 1] 22
path bound  [0, 2, 3] 27
path bound  [0, 2, 4] 39
path bound  [0, 2, 1, 3] 37
path bound  [0, 2, 1, 4] 31
path bound  [0, 2, 3, 1] 43
path bound  [0, 2, 3, 4] 34
path bound  [0, 3, 1] 45
path bound  [0, 3, 2] 38
path bound  [0, 3, 4] 30
path bound  [0, 3, 4, 1] 30
path bound  [0, 3, 4, 2] 48
30
[0, 3, 4, 1, 2]
>>>
```