알고리즘및실습

제13장 분기한정법

1. 개요

12장에서 전수조사 알고리즘의 성능을 기계적으로 개선할 수 있는 되추적 기법을 살펴보았다. 되추적은 전수조사 재귀 알고리즘의 호출 순서를 바꾸지 않고, 일부 호출을 가지치기하여 전수조사하는 경우의 수를 줄여 준다. **분기한정법**(branch-and-bound)은 더 많은 경우를 배제하기 전수조사하는 순서를 바꾼다. 전수조사 순서를 바꾸기 위해서는 조사하는 트리를 직접 만들어야 한다. 그다음 우선순위 큐를 이용하여 더 유망한 노드를 먼저 조사하는 방식을 사용한다. 이 측면에서 트리의 탐색은 너비 우선에 가깝다. 분기한정은 노드를 조사하여 노드 간의 유망 정도를 구분할 수 있어야 한다. 이 때문에 되추적과 달리 최적화 문제만 분기한정법을 적용할 수 있다. 예를 들어 12장에서 살펴본 부분집합의 합 문제는 분기한정법을 사용할 수 없다. 같은 레벨에 있는 두 개의 유망한 노드가 주어졌을 때 둘 중 더 유망한 노드가 존재하지 않는다. 12장에서 살펴본 n-Queens 문제, 그래프 색칠하기 문제, 해밀톤 회로 문제 모두 분기한정법을 사용할 수 있는 문제가 아니다.

분기한정은 되추적과 같은 상태 공간 트리를 사용하지만 되추적과 달리 직접 트리를 만들어야 한다. 직접 트리를 만든다고 전체 트리를 구축하는 것은 아니다. 상태 공간 트리의 루트 노드를 만들어 우선순위 큐에 포함하여 너비 우선 탐색을 변형하면 된다. 즉, 상태 공간 트리는 우선 순위 큐를 이용한 탐색을 통해 논리적으로 만들어지는 트리이다. 이렇게 우선순위 큐를 사용하면 가장 유망한 경로를 먼저 탐색하기 때문에 최적해를 되추적보다 빨리 찾을 수 있다. 최적해를 빨리 찾을수록 더 많은 경우를 배제할 수 있다. 하지만 여전히 상태 공간 트리를 순회해야 하므로 최악의 경우 시간 복잡도는 기존 전수조사 알고리즘과 같다. 보통 경우의 수가 지수 개가 존재하면 분기한정의 시간 복잡도는 지수시간이다.

2. 0-1 배낭 채우기 문제

10장에서 이 문제를 테뷸레이션 방식의 동적 프로그래밍으로 해결하였고, 12장에서는 되추적 기법을 이용하여 해결하였다. 이 절에서는 분기한정법을 이용하여 해결해 보고자 한다. 0-1 배낭 채우기 되추적 알고리즘은 무게당 이득을 기준으로 내림차순 정렬하여 물건을 조사하며, 빈틈없이 채우기 문제를 이용하여 탐색하고 있는 경로를 통해 얻을 수 있는 최대 이득을 조사한다. 분기한정은 되추적과 동일한 조건을 이용하여 가지치기를 한다. 차이점은 분기한정은 상태 공간 트리를 직접 만들어 순회하며, 순회할 때 우선순위 큐를 이용한다. 지금까지 탐색한 노드 중 가장 유망한 노드부터 조사한다. 이 때문에 이와 같은 알고리즘은 최고 우선 검색 분기 한정 가지치기(best-first search with branch-and-bound pruning)라 한다.

상태 공간 트리를 만들기 위해 각 노드가 유지해야 하는 값은 다음과 같다.

- `level`: 노드의 레벨이며, 이 레벨에서 고려해야 하는 물건 색인
- `currWeight`: 현 시점에서 배낭에 포함된 물건의 무게 합

- `currProfit`: 현 시점에서 배낭에 포함된 물건의 이득 합
- `bound`: 빈틈없는 배낭 채우기 문제를 이용하여 얻을 수 있는 이득의 상한값
- `included`: 지금까지 포함한 물건 정보 (논리형 배열)

우선 순위 큐를 이용한 0-1 배낭 분기한정 알고리즘은 알고리즘 13.1과 같다.

알고리즘 13.1 knapsack 분기한정 알고리즘

```
1:  function computeBound(items[], W, j, currProfit, currWeight)
2:      bound := currProfit
3:      totalWeight := currWeight
4:      while j <= n and totalWeight + items[j].w <= W do
5:          bound += items[j].v
6:          totalWeight += items[j].w
7:          ++j
8:      if j <= n then
9:          bound += (W − totalWeight) × items[j].v/items[j].w
10:     return bound
11: function knapsack(items[], W)
12:     sort(items)                                  ▷ 무게당 이득 기준으로 내림차순 정렬
13:     maxProfit := 0
14:     solution := null
15:     Q := empty priority queue                    ▷ bound 기준의 Node를 유지하는 우선순위 큐
16:     root := (0, 0, 0, computeBound(items, W, 1, 0, 0), [false] × N)
17:     Q.push(root)
18:     while not Q.isEmpty() do
19:         node := Q.pop()
20:         ++node.level
21:         node.currProfit += items[node.level].v
22:         node.currWeight += items[node.level].w
23:         node.include[node.level] := true
24:         if node.currWeight <= W and node.currProfit > maxProfit then
25:             maxProfit := node.currProfit
26:             solution := node.include
27:         for i := 1 to 2 do
28:             node.bound := computeBound(items, W,
                    node.level + 1, node.currProfit, node.currWeight)
29:             if node.bound > maxProfit then
30:                 Q.push(node.clone())
31:             node.currProfit −= items[node.level].v
32:             node.currWeight −= items[node.level].w
33:             node.include[node.level] := false
34:     return maxProfit, solution
```

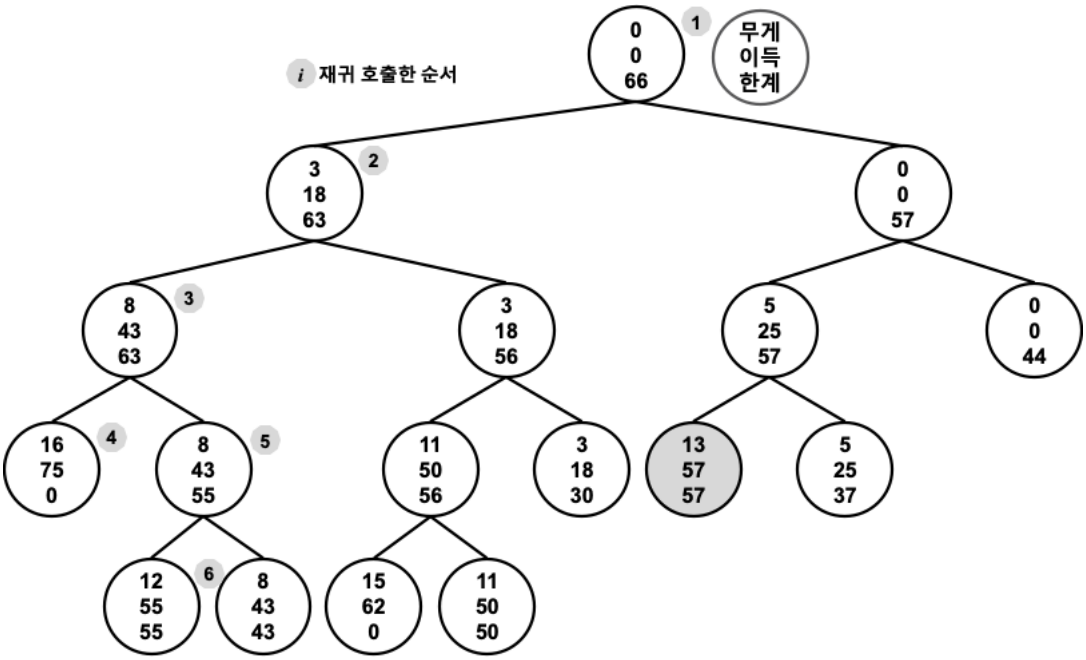
computeBound 함수는 0-1 배낭 채우기 되추적 알고리즘에서 사용하였던 것과 같은 것이다. 다음과 같이 4개의 물건이 주어지고,

$$[(12, 4), (32, 8), (25, 5), (18, 3)]$$

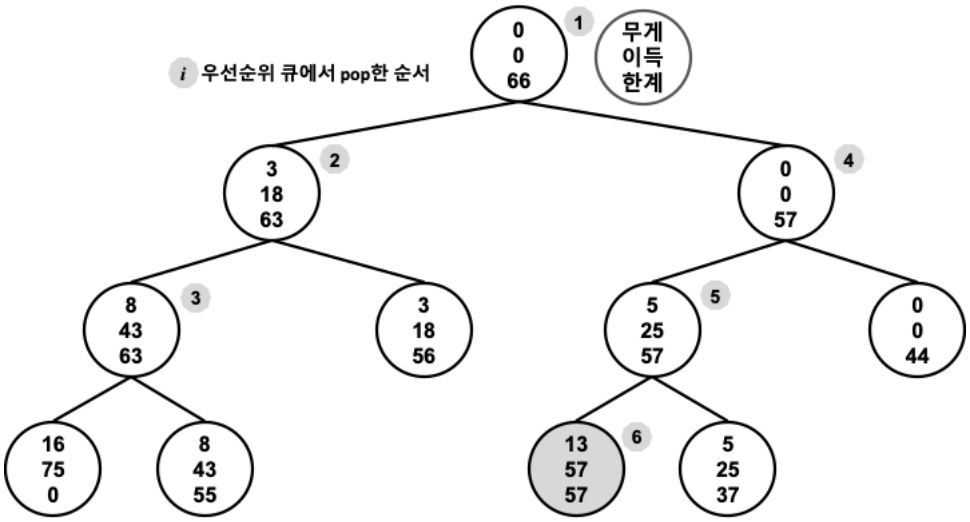
가방의 용량이 13일 때 12장의 0-1 배낭 되추적 알고리즘을 수행하였을 때 가지치기된 상태 공간 트리는 그림 13.1과 같으며, 같은 것을 알고리즘 13.1을 이용하여 수행하면 그림 13.2과 같은 가지치기된 상태 공간 트리를 얻게 된다.

무게당 이득 순으로 정렬하면 다음과 같다.

$$(18, 3, 6), (25, 5, 5), (32, 8, 4), (12, 4, 3)$$



<그림 13.1> 되추적 알고리즘을 이용한 가지치기된 상태 공간 트리



<그림 13.2> 알고리즘 13.1을 이용한 가지치기된 상태 공간 트리

따라서 빈틈없는 배낭 채우기로 채우면 $18 + 25 + 32(5/8) = 66$이 최대 이득이다. 두 상태 공간 트리의 차이는 문제의 사례마다 다를 수 있다. 분기한정이 추가로 배제하는 것이 거의 없을 수도 있다. 또 노드를 계속 복제하면서 진행하기 때문에 복제 비용도 무시하기 힘들다. 물론 재귀 호출 없이 반복문으로 동작하는 추가 이점도 있다.

3. 외판원 문제

외판원 문제는 11장에서 동적 프로그래밍을 이용하여 해결하였다. 이 절에서는 분기 한정법을 이용하여 보자. 외판원 문제에서 주어지는 그래프는 보통 완전 그래프이다. 따라서 모든 일주여행경로(투어)를 찾아 그 중에 가장 길이가 짧은 것을 선택하는 것은 지수 시간 비용이 소요된다. 전수조사를 할 때 가지치기를 할 수 있다면 성능을 개선을 할 수 있으며, 외판원 문제는 최적화 문제이므로 되추적보다는 분기한정법을 사용하는 것이 더 효과적이다.

분기한정법을 사용하기 위해서는 각 노드의 한계치를 계산할 수 있어야 한다. 노드의 한계치는 그 노드를 확장하여 얻을 수 있는 투어 길이의 하한값이다. 노드의 한계치가 지금까지 계산된 최적의 투어 길이보다 크면 이 노드는 유망하지 않다. 그래프가 주어졌을 때 한계치를 어떻게 계산할 수 있을까? 외판원 문제에서 그래프는 보통 가중치 방향 그래프이며, 완전 그래프이다. 따라서 그래프를 인접 행렬로 표현하는 것이 가장 효과적이다.

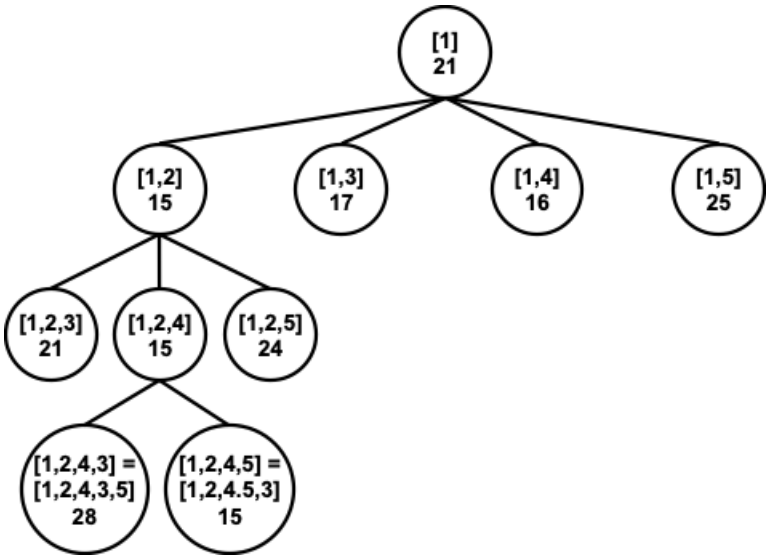
투어이기 때문에 경로의 간선 수는 n이며, 모든 노드마다 하나의 진출 간선으로 구성된다. 따라서 각 노드의 진출 간선 중 가장 작은 것을 모두 합한 것을 이용하여 한계치를 추정할 수 있다. 물론 이들 간선을 모은 것이 일주여행경로가 아닐 확률이 높지만 확실한 것은 이보다 더 작은 투어는 존재할 수 없다.

예를 들어 다음과 같은 5개 노드로 구성된 그래프가 주어졌다고 하자.

$$\begin{bmatrix} 0 & 4 & 6 & 2 & 9 \\ 8 & 0 & 8 & 5 & 9 \\ 3 & 7 & 0 & 4 & 3 \\ 6 & 10 & 8 & 0 & 1 \\ 8 & 2 & 2 & 5 & 0 \end{bmatrix}$$

v_1의 진출 간선 중 가중치가 가장 작은 갓은 $\min\{4, 6, 2, 9\} = 2$이고, v_2는 5, v_3는 3, v_4는 1, v_5는 2이다. 따라서 $2 + 5 + 3 + 1 + 2 = 13$을 분기한정 상태 공간 트리 루트 노드의 한계치로 계산하게 된다.

외판원 문제에서 시작 노드는 중요하지 않다. 1로 설정한 다음 그다음 노드로 2를 선택하였다고 하자. 그러면 이 간선의 가중치 4가 투어 길이 계산에 사용해야 한다. 그다음에는 노드 2, 3, 4, 5의 진출 간선을 이용하여 한계치를 추정해야 한다. 이때 2는 이미 방문하였기 때문에 2로 가는 진출 간선을 사용할 수 없고, 노드 2는 마지막 노드가 될 수 없으므로 2에서 1로 가는 간선은 선택할 수 없다. 그러면 추정치는 $4 + 5 + 3 + 1 + 2 = 15$가 된다.



<그림 13.3> 외판원 문제: 가지치기된 상태 공간 트리

$[1, 2, 3[$을 생각해 보면 $(1, 2) = 4$, $(2, 3) = 8$이고 v_3의 진출 간선은 3, v_4는 1, $v_5 = 5$이므로 $4+8+3+1+5 = 21$, $[1, 2, 4]$는 $(1, 2) = 4$, $(2, 4) = 5$, v_3는 3, v_4는 1, $v_5 = 2$이므로 $4+5+3+1+2 = 15$이다. $[1, 2, 4]$를 추가로 확장하여 보자. 그다음 노드로 3과 5가 가능하다. 그런데 둘 중 하나를 선택하면 그다음은 고정된다. 따라서 두 개의 노드가 남으면 투어 자체를 만들게 된다. 예를 들어 $[1, 2, 4, 3]$은 그다음 노드가 고정되므로 $[1, 2, 4, 3, 5]$와 같으며, 이 투어의 길이는 $4 + 5 + 8 + 3 + 8 = 28$이다. 전체 가지치기된 상태 공간 트리는 그림 13.3과 같다.

따라서 상태 공간 트리를 만들기 위해 각 노드가 유지해야 하는 값은 다음과 같다.

- `level`: 노드의 레벨이며, 이 레벨에서 고려해야 하는 노드 색인
- `bound`: 지금까지 경로를 포함하는 일주여행경로의 하한
- `included`: 지금까지 포함한 노드 정보
- `tour`: 지금까지 경로 정보

알고리즘 13.2 tsp 분기한정 알고리즘

```
1:  procedure computeBound(G, node)
2:      node.bound := 0
3:      for i := 2 to node.level + 1 do                          ▷ 실제 경로의 길이
4:          node.bound+= G[node.tour[i − 1]][node.tour[i]]
5:      minEdge := ∞
6:      for w := 2 to n do                  ▷ 경로 끝 노드가 아니면 노드 1로 진출하는 간선을 사용할 수 없음
7:          if node.included[w] then continue
8:          minEdge := min(minEdge, G[node.tour[−1]][w])
9:      node.bound+= minEdge
10:     for v := 1 to n do                                       ▷ 경로 길이 추정
11:         if node.included[v] then continue                    ▷ 방문 노드 제외
12:         minEdge := ∞
13:         for w := 1 to n do
14:             if w ≠ 1 and node.included[w] then continue
15:             minEdge := min(minEdge, G[v][w])
16:         node.bound+= minEdge
17: function tsp(G)
18:     minLength := ∞
19:     tour := null
20:     Q := []                                  ▷ bound 기준의 Node를 유지하는 우선순위 큐
21:     root := (0, _, [false] × N, [1])
22:     CallComputeBoundroot
23:     Q.push(root)
24:     while not Q.isEmpty() do
25:         node := Q.pop()
26:         if node.bound < minLength then
27:             for i := 2 to n do
28:                 if node.include[i] then continue
29:                 if G[node.tour[−1]][i] == INF then continue
30:                 next := node.clone()
31:                 ++next.level
32:                 next.tour.pushback(i)
33:                 if next.level = n − 2 then
34:                     complete the tour
35:                     tourLength := length(next)
36:                     if tourLength < minLength then
37:                         minLength := tourLength
38:                         tour := next.tour
39:                 else
40:                     ComputeBound(G, next)
41:                     if next.bound < minLength then
42:                         Q.push(next)
43:     return minLength, tour
```

매번 `node.included`를 이용하여 제외할 것을 검사하는 것보다 남은 노드 집합을 트리 집합 자료구조로 유지하는 것이 효과적일 수 있다.

4. 작업 할당 문제

작업 할당 문제는 이전 절에서 살펴본 외판원 문제와 매우 유사한 문제이다.

작업 할당 문제

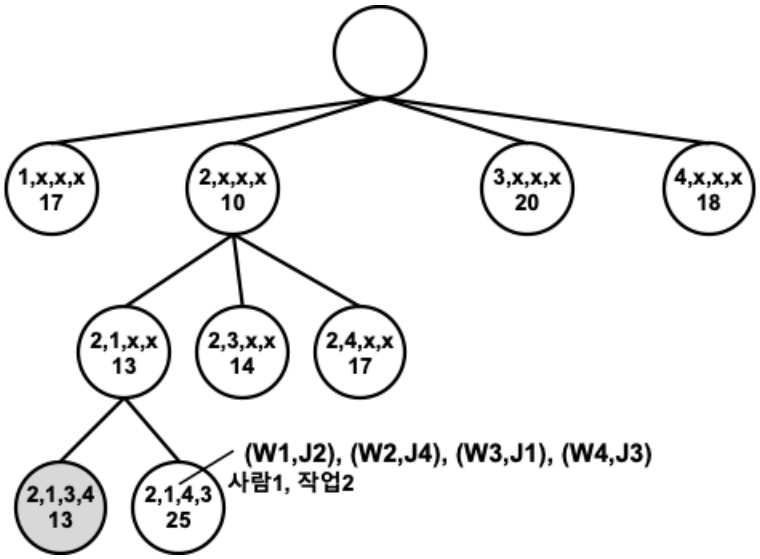
- 입력. n개 명의 사람이 n개의 작업을 수행할 때 소요되는 시간
- 출력. 전체 작업 완료 시간을 최소화하도록 각 사람에게 하나의 작업을 할당해야 함

이 문제의 총 경우의 수는 $n!$이다. 이 문제의 입력 데이터도 TSP 문제처럼 다음과 같이 2차원 배열로 표현할 수 있다.

$$\begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

사람 1이 작업 1, 작업 2, 작업 3, 작업 4을 완료하는데 필요한 시간은 첫 번째 행과 같다. 각 행에서 하나만 선택할 수 있고, 한 열에서 하나만 선택할 수 있다.

외판원 문제와 유사하게 분기한정법을 진행할 수 있다. 할당이 확정되지 않는 작업 중에서 최소 시간에 완료할 수 있는 작업을 선택하여 최소 한계를 추정한다. 위 예에서 각 행의 최소값 2, 3, 1, 4이므로 추정한 전체 작업 완료 시간은 9이다. 두 번째와 세 번째 사람에게 3번째 작업을 동시에 할당할 수 없으므로 이 할당은 유효하지 않은 할당이지만 유효한 할당 중 전체 작업 완료 시간이 이보다 더 적을 수 없다. 주어진 예제에 대해 분기한정법을 이용한 가지치기된 상태 공간 트리는 그림 13.4와 같다.



<그림 13.4> 작업 할당 문제: 가지치기된 상태 공간 트리

퀴즈

1. 분기한정법과 관련된 다음 설명 중 틀린 것은?

   ① BFS를 응용한 기법이기 때문에 되추적과 달리 실제 노드를 만들어 탐색해야 한다.
   ② 되추적과 달리 가장 유망한 노드부터 검색한다.

③ BFS를 응용한 기법이기 때문에 FIFO 큐를 사용한다.

④ 되추적과 마찬가지로 노드의 유망 여부를 검사하여 유망하지 않은 노드 아래로는 탐색하지 않는다.

2. 외판원 문제에서 그래프가 다음과 같이 주어졌을 때,

$$\begin{bmatrix} 0 & 18 & 2 & 4 & 15 \\ 2 & 0 & 15 & 19 & 17 \\ 1 & 20 & 0 & 12 & 20 \\ 10 & 8 & 3 & 0 & 17 \\ 4 & 13 & 16 & 19 & 0 \end{bmatrix}$$

분기한정법을 이용하여 최적 투어를 찾고자 한다. $1 \rightarrow 4 \rightarrow 3$까지 결정된 이후 최적 투어의 한계치는?

① 12

② 25

③ 33

④ 23

연습문제

1. n개의 영소문자로만 구성된 문자열 목록, 중복이 있을 수 있는 문자 목록, 각 문자의 점수가 주어진다. 문자 목록에 있는 문자를 최대 한 번만 사용하여 문자열 목록에 있는 문자열을 최대로 만들었을 때 합계 점수가 가장 높은 경우를 반환하라. 예를 들어 [“dog”,“cat”,“duck”,“good”], [‘a’, ‘a’, ‘c’, ‘c’, ‘d’, ‘d’, ‘k’, ‘o’, ‘o’, ‘o’], $[2, 0, 8, 6, 0, 0, 2, 0, 0, 0, 4, 0, 0, 0, 3, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$이 주어진 경우를 생각하여 보자. “dog” $= 6 + 3 + 2 = 11$, “good” $= 2 + 3 + 3 + 6 = 14$를 만들면 ‘d’ 문자를 다 사용했기 때문에 “duck”을 만들 수 없고, 두 문자열의 합계 점수는 25이다. 반면에 “good” $= 2 + 3 + 3 + 6 = 14$과 “duck” $= 6 + 1 + 8 + 4 = 19$을 만들면 합계 점수가 33이므로 답은 33이다.

2. 3×3 격자 모양의 보드가 주어진다. 이 보드의 각 셀은 1부터 8까지 수로 채워지며, 하나의 셀은 빈 셀이다. 초기 보드 모습과 최종 보드 모습이 주어지면 셀을 상하좌우로 옮겨 최종 모습이 되도록 바꾸어야 한다. 초기 보드 모습을 최종 보드 모습으로 바꾸는데 필요한 최소 이동을 출력하라.