분기한정법
(Branch-and-Bound)

알고리즘 해석
강의 슬라이드 #6
도경구

도경구역, 알고리즘, 사이텍미디어,
1999

분기한정법(Branch-and-Bound)

- 특징:
  - 되추적 기법과 같이 상태공간트리를 구축하여 문제를 해결한다.
  - 최적의 해를 구하는 문제(optimization problem)에 적용할 수 있다.
  - 최적의 해를 구하기 위해서는 어짜피 모든 해를 다 고려해 보아야 하므로 트리의 마디를 순회(traverse)하는 방법에 구애받지 않는다.
- 분기한정 알고리즘의 원리
  - 각 마디를 검색할 때 마다, 그 마디가 유망한지의 여부를 결정하기 위해서 한계치(bound)를 계산한다.
  - 그 한계치는 그 마디로부터 가지를 뻗어나가서(branch) 얻을 수 있는 해답치의 한계를 나타낸다.
  - 따라서 만약 그 한계치가 지금까지 찾은 최적의 해답치 보다 좋지 않은 경우는 더 이상 가지를 뻗어서 검색을 계속할 필요가 없으므로, 그 마디는 유망하지 않다고 할 수 있다.

0-1 배낭채우기 문제

- 분기한정 가지치기로 깊이우선검색 (= 되추적)
  - 상태공간트리를 구축하여 되추적 기법으로 문제를 푼다.
  - 뿌리마디에서 왼쪽으로 가면 첫번째 아이템을 배낭에 넣는 경우이고, 오른쪽으로 가면 첫번째 아이템을 배낭에 넣지 않는 경우이다.
  - 동일한 방법으로 트리의 수준 1에서 왼쪽으로 가면 두 번째 아이템을 배낭에 넣는 경우이고, 오른쪽으로 가면 그렇지 않는 경우이다.
  - 이런 식으로 계속하여 상태공간트리를 구축하면, 뿌리마디로부터 잎마디까지의 모든 경로는 해답후보가 된다.
  - 이 문제는 최적의 해를 찾는 문제(optimization problem)이므로 검색이 완전히 끝나기 전에는 해답을 알 수가 없다. 따라서 검색을 하는 과정 동안 항상 그 때까지 찾은 최적의 해를 기억해 두어야 한다.

최적화 문제를 풀기 위한 일반적인 되추적 알고리즘

- ```
  void checknode(node v) {
     node u;
     if(value(v) is better than best)
        best = value(v);
     if(promising(v))
        for(each child u of v)
           checknode(u);
  }
  ```
  - ✓ best : 지금까지 찾은 제일 좋은 해답치.
  - ✓ value(v) : v 마디에서의 해답치.

2001-03-19 알고리즘 강의 슬라이드 6 5

0-1 배낭채우기: 알고리즘

- **알고리즘 스케치:**
  - ✓ Let:
    - *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

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

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

    - *maxprofit* : 지금까지 찾은 최선의 해답이 주는 값어치
  - ✓ w_i와 p_i를 각각 i번째 아이템의 무게와 값어치라고 하면, p_i / w_i 의 값이 큰 것부터 내림차순으로 아이템을 정렬한다. (일종의 탐욕적인 방법이 되는 셈이지만, 알고리즘 자체는 탐욕적인 알고리즘은 아니다.)
  - ✓ *maxprofit* := \$0; *profit* := \$0; *weight* := 0

2001-03-19 알고리즘 강의 슬라이드 6 6
  ```

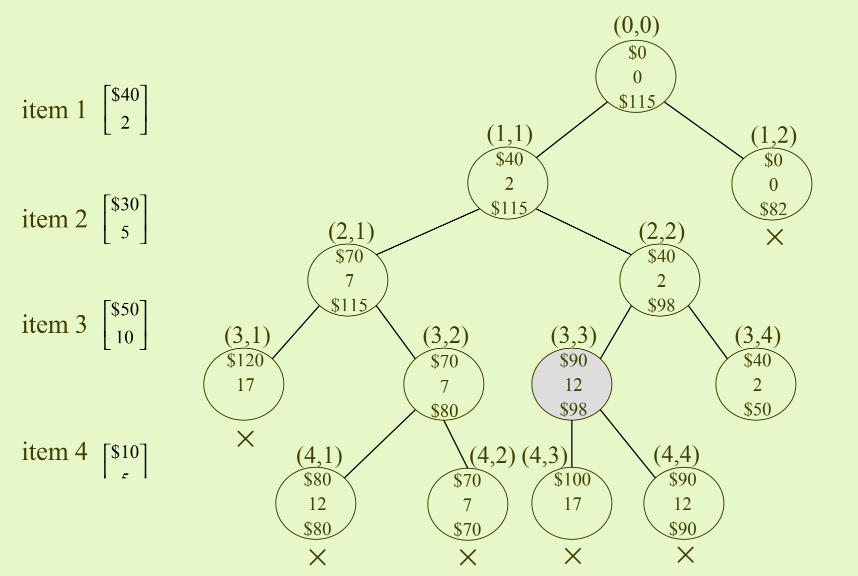
- ✓ 깊이우선순위로 각 마디를 방문하여 다음을 수행한다:
  1. 그 마디의 *profit*와 *weight*를 계산한다.
  2. 그 마디의 *bound*를 계산한다.
  3. *weight* < *W* and *bound* > *maxprofit*이면, 검색을 계속한다; 그렇지 않으면, 되추적.
- ✓ 고찰: 최선이라고 여겼던 마디를 선택했다고 해서 실제로 그 마디로부터 최적해가 항상 나온다는 보장은 없다.

- 보기: $n = 4$, $W = 16$이고

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

일 때, 되추적을 사용하여 구축되는 가지친 상태공간트리를 그려 보시오.

2001 -03 -19 알고리즘 강의 슬라이드 6 7



2001 -03 -19 알고리즘 강의 슬라이드 6 8

0-1 배낭채우기 알고리즘: 분석

- 이 알고리즘이 점검하는 마디의 수는 $\Theta(2^n)$이다.
- 위 보기의 경우의 분석: 점검한 마디는 13개이다. 이 알고리즘이 동적계획법으로 설계한 알고리즘 보다 좋은가?
  - ✓ 확실하게 대답하기 불가능 하다.
  - ✓ Horowitz와 Sahni(1978)는 Monte Carlo 기법을 사용하여 되추적 알고리즘이 동적계획법 알고리즘 보다 일반적으로 더 빠르다는 것을 입증하였다.
  - ✓ Horowitz와 Sahni(1974)가 분할정복과 동적계획법을 적절히 조화하여 개발한 알고리즘은 $O(2^{n/2})$의 시간복잡도를 가지는데, 이 알고리즘은 되추적 알고리즘 보다 일반적으로 빠르다고 한다.

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

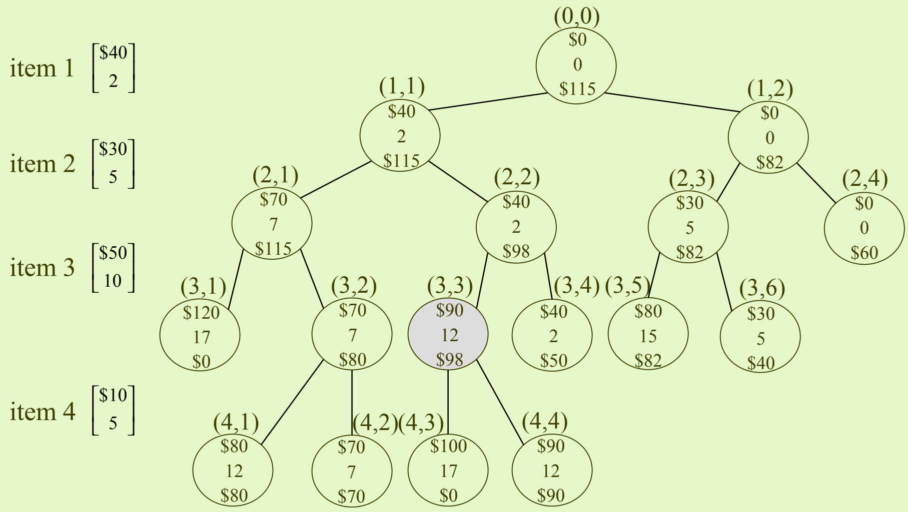
- 되부름(recursive) 알고리즘을 작성하기는 상당히 복잡하다.
따라서 **대기열(queue)**을 사용한다.

```
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);
    }
  }
}
```


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);
    }
  }
}
```

- 보기: 앞에서와 같은 예를 사용하여 분기한정 가지치기로 너비우선검색을 하여 가지친 상태공간트리를 그려보면 다음과 같이 된다. 이때 검색하는 마디의 개수는 **17**이다. <u>되추적 알고리즘보다 좋지 않다</u>!



2001 -03 -19 알고리즘 강의 슬라이드 6 13

분기한정 가지치기로 최고우선검색 (Best-First Search)

- 최적의 해답에 더 빨리 도달하기위한 전략:
  1. 주어진 마디의 모든 자식마디를 검색한 후,
  2. 유망하면서 확장되지 않은(unexpanded) 마디를 살펴보고,
  3. 그 중에서 가장 좋은(최고의) 한계치(bound)를 가진 마디를 확장한다.
- 최고우선검색(Best-First Search)은 너비우선검색에 비해서 좋아짐

2001 -03 -19 알고리즘 강의 슬라이드 6 14

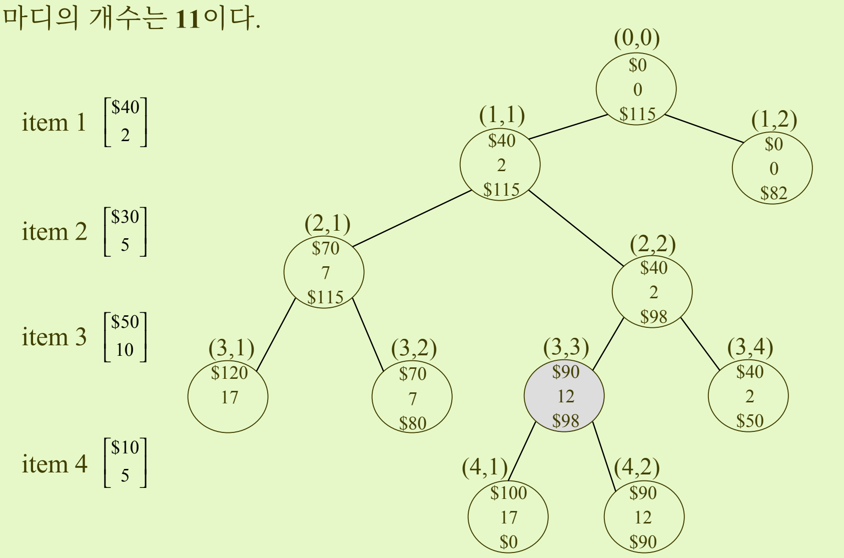
최고우선검색 전략

- 최고의 한계를 가진 마디를 우선적으로 선택하기 위해서 우선순위 대기열(Priority Queue)을 사용한다.
- 우선순위 대기열은 힙(heap)을 사용하여 효과적으로 구현할 수 있다.

분기한정 최고우선검색 알고리즘

```
void best_first_branch_and_bound(state_space_tree T,
                                 number best) {
  priority_queue_of_node PQ;
  node u,v;
  initialize(PQ);                   // PQ를 빈 대기열로 초기화
  v = root of T;
  best = value(v);
  insert(PQ,v);
  while(!empty(PQ)) {               // 최고 한계값을 가진 마디를 제거
    remove(PQ,v);
    if(bound(v) is better than best)      // 마디가 아직 유망한 지 점검
      for(each child u of v) {
        if(value(u) is better than best)
          best = value(u);
        if(bound(u) is better than best)
          insert(PQ,u);
      }
  }
}
```

- 보기: 앞에서와 같은 예를 사용하여 분기한정 가지치기로 최고우선검색을 하여 가지친 상태공간트리를 그려보면, 다음과 같이 된다. 이때 검색하는 마디의 개수는 **11**이다.



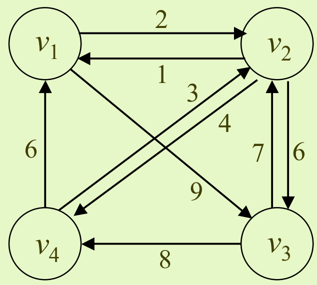
2001 -03 -19 알고리즘 강의 슬라이드 6 17

외판원 문제
(Traveling Saleswoman Problem)

- 외판원의 집이 위치하고 있는 도시에서 출발하여 다른 도시들을 각각 한번씩 만 방문하고, 다시 집으로 돌아오는 가장 짧은 일주여행경로(tour)를 결정하는 문제.
- 이 문제는 음이 아닌 가중치가 있는, 방향성 그래프로 나타낼 수 있다.
- 그래프 상에서 일주여행경로(tour, Hamiltonian circuits)는 한 정점을 출발하여 다른 모든 정점을 한번씩 만 거쳐서 다시 그 정점으로 돌아오는 경로이다.
- 여러 개의 일주여행경로 중에서 길이가 최소가 되는 경로가 최적일주여행경로(optimal tour)가 된다.
- 무작정 알고리즘: 가능한 모든 일주여행경로를 다 고려한 후, 그 중에서 가장 짧은 일주여행경로를 선택한다. 가능한 일주여행경로의 총 개수는 $(n - 1)!$이다.

2001 -03 -19 알고리즘 강의 슬라이드 6 18

예제: 가장 최적이 되는 일주여행경로는?



동적계획법을 이용한 접근방법

- V는 모든 정점의 집합이고, A는 V의 부분집합이라고 하자. 그리고 $D[v_i][A]$는 A에 속한 각 정점을 정확히 한번씩 만 거쳐서 v_i에서 v_1로 가는 최단경로의 길이라고 하자. 그러면 위의 예제에서 $D[v_2][\{v_3, v_4\}]$의 값은? (=20)
- 최적 일주여행경로의 길이:

$$D[v_1][V-\{v_1\}] = min_{2 \le j \le n}(W[1][j] + D[v_j][V-\{v_1, v_j\}])$$

일반적으로 표시하면 $i \neq 1$이고, v_i가 A에 속하지 않을 때, 다음과 같이 된다.

$$D[v_i][A] = min_{v_j \in A}(W[i][j] + D[v_j][A-\{v_j\}]) \text{ if } A \neq 0$$

$$D[v_i][0] = W[i][1]$$

- 연습으로 위의 보기의 그래프에서 최적일주여행경로를 찾아보시오. 즉, $D[v_1][\{v_2, v_3, v_4\}]$를 계산해 보시오.

동적계획 알고리즘

- 문제: 가중치(음수가 아닌 정수)가 있는 방향성 그래프에서 최적일주여행경로를 결정하시오.
- 입력:
  - 가중치가 있는 방향성 그래프
  - 그 그래프에 있는 정점의 개수 n.
  - 그래프는 행렬 W로 표시가 되는데, 여기서 $W[i][j]$는 v_i에서 v_j를 잇는 이음선 상에 있는 가중치를 나타낸다.
  - V는 그래프 상의 모든 정점의 집합을 나타낸다.
- 출력:
  - 최적일주여행경로의 길이 값을 가지는 변수 *minlength*
  - 배열 P (이 배열로부터 최적일주여행경로를 구축할 수 있다). $P[i][A]$는 A에 속한 각 정점을 정확히 한번씩 만 거쳐 v_i에서 v_1로 가는 최단경로 상에서, v_i 다음의 도달하는 첫번째 마디의 인덱스이다.

2001 -03 -19 알고리즘 강의 슬라이드 6 21

동적계획 알고리즘

```
void travel(int n,const number W[][],index P[][],
            number& minlength) {
  index i,j,k;
  number D[1..n][subset of V-{v1}];
  for(i=2; i<=n; i++)
    D[i][emptyset] := W[i][1];
  for(k=1; k<=n-2; k++)
    for(V-{v1})의 부분집합 중에서 k개의 정점을 가진 모든 부분집합 A)
      for(i=1이 아니고 vi가 A에 속하지 않는 모든 i){
        D[i][A] = minimum_{vj∈A}(W[i][j] + D[vj][A-{vj}]);
        P[i][A] = value of j that gave the minimum;
      }
  D[1][V-{v1}] = minimum_{2≤j≤n}(W[1][j] + D[vj][A-{v1}]);
  P[1][V-{v1}] = value of j that gave the minimum;
  minilength = D[1][V-{v1}];
}
```

2001 -03 -19 알고리즘 강의 슬라이드 6 22

동적계획 알고리즘의 분석

- **정리**: $n \geq 1$를 만족하는 모든 n에 대해서,

$$\sum_{k=1}^{n} k \begin{bmatrix} n \\ k \end{bmatrix} = n 2^{n-1}$$

증명:

$$\begin{bmatrix} n \\ k \end{bmatrix} = \frac{n!}{k!(n-k)!} = \frac{n}{k} \frac{(n-1)!}{(k-1)!(n-k)!}$$

$$= \frac{n}{k} \frac{(n-1)!}{(k-1)!(n-1-(k-1))!}$$

$$= \frac{n}{k} \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

$$k \begin{bmatrix} n \\ k \end{bmatrix} = k \frac{n}{k} \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} = n \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

$$\sum_{k=1}^{n} k \begin{bmatrix} n \\ k \end{bmatrix} = \sum_{k=1}^{n} n \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} = n \sum_{k=0}^{n-1} \begin{bmatrix} n-1 \\ k \end{bmatrix} 1^k 1^{n-1-k} = n(1+1)^{n-1} = n 2^{n-1}$$

동적계획 알고리즘의 분석

- 단위연산: 중간에 위치한 루프가 수행시간을 지배한다. 왜냐하면 이 루프는 여러 겹으로 쌓여 있기 때문이다. 따라서, 단위연산은 v_j의 각 값에 대해서 수행되는 명령문이다.(덧셈하는 명령문 포함)
- 입력 크기: 그래프에서 정점의 개수 n
- 시간 복잡도: 알고리즘에서 두 번째 for-루프가 시간복잡도를 좌우한다.

```
    for(k=1; k<=n-2; k++)
(1)   for(V-{v_i})의 부분집합 중에서 k개의 정점을 가진 모든 부분집합 A)
(2)     for(i=1이 아니고 v_i가 A에 속하지 않는 모든 i)
(3)       D[i][A] = minimum_{vj∈A}(W[i][j] + D[v_j][A-{v_j}]);
          P[i][A] = value of j that gave the minimum;
```

동적계획 알고리즘의 분석: 계속

- (1)번 루프는 $\begin{bmatrix} n-1 \\ k \end{bmatrix}$ 번 반복하고 (n -1개의 정점에서 k개를 뽑는 경우의 수), (2)번 루프는 n - k - 1번 반복하고 (v_1을 제외하고 A에 속하지 않는 정점의 개수), (3)번 루프는 A의 크기가 k이므로 k번 반복한다(A에 속한 정점의 개수). 따라서 시간복잡도는

$$T(n) = \sum_{k=1}^{n-2} (n-k-1)k\begin{bmatrix} n-1 \\ k \end{bmatrix}$$

- 공간복잡도: 배열 $D[v_i,A]$와 $P[v_i,A]$가 얼마나 커야 하는지를 결정하면 된다. V - $\{v_1\}$는 n - 1개의 정점을 가지고 있기 때문에, 이 배열은 2^{n-1}개의 부분집합 A를 가지고 있다. (n개의 아이템이 포함되어 있는 어떤 집합의 부분집합의 개수는 2^n이다.) 따라서 $M(n) = 2 \times n \times 2^{n-1} = n2^n \in \Theta(n2^n)$

2001 -03 -19 알고리즘 강의 슬라이드 6 25

외판원문제 알고리즘의 비교

- $n = 20$ 일 때,
  - ✓ 무작정 알고리즘: 각 일주여행경로의 길이를 계산하는데 걸리는 시간은 1μsec이라고 할 때, $(20 - 1)! = 19!\mu sec = 3857$년이 걸린다.
  - ✓ 동적계획법 알고리즘: 동적계획법 알고리즘의 기본동작을 수행하는데 걸리는 시간을 1μsec이라고 할 때, $T(20) = (20 - 1)(20 - 2)2^{20-3}\mu sec = 45$초가 걸리고, $M(20) = 20 \times 2^{20} =$ 20,971,520의 배열의 슬롯이 필요하다.

2001 -03 -19 알고리즘 강의 슬라이드 6 26

배열 P에서 최적여행경로 구축

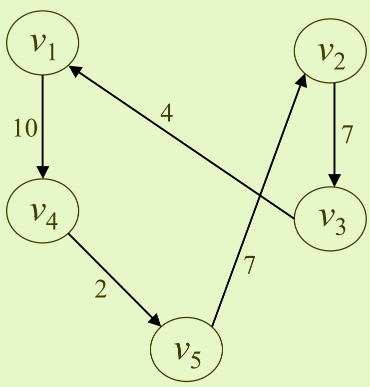
$$P[1,\{v_2,v_3,v_4\}]=3 \Rightarrow P[3,\{v_2,v_4\}]=4 \Rightarrow P[4,\{v_2\}]=2$$

- 따라서 최적 일주여행경로는 $[v_1, v_3, v_4, v_2, v_1]$.

외판원문제: 분기한정법

- $n = 40$일 때, 동적계획법 알고리즘은 6년 이상이 걸린다. 그러므로 분기한정법을 시도해 본다.
- **보기**: 다음 인접행렬로 표현된 그래프를 살펴보시오.

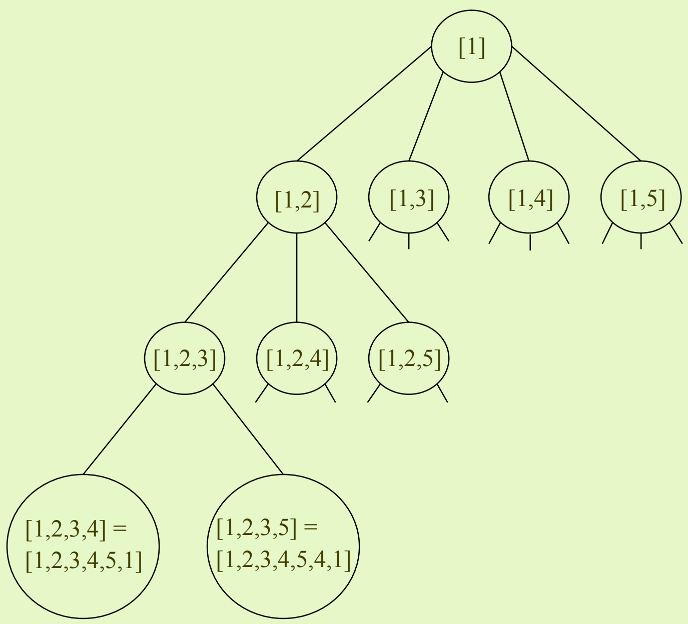
$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



상태공간트리 구축방법

- 각 마디는 출발마디로부터의 일주여행경로를 나타내게 되는데, 몇 개 만 예를 들어 보면, 뿌리마디의 여행경로는 [1]이 되고, 뿌리마디에서 뻗어 나가는 수준 1에 있는 여행경로는 각각 [1,2], [1,3], …, [1,5]가 되고, 마디 [1,2]에서 뻗어 나가는 수준 2에 있는 마디들의 여행경로는 각각 [1,2,3],…,[1,2,5]가 되고, 이런 식으로 뻗어 나가서 잎마디에 도달하게 되면 완전한 일주여행경로를 가지게 된다.
- 따라서 최적일주여행경로를 구하기 위해서는 잎마디에 있는 일주여행경로를 모두 검사하여 그 중에서 가장 길이가 짧은 일주여행경로를 찾으면 된다.
- 참고: 위 예에서 각 마디에 저장되어 있는 마디가 4개가 되면 더 이상 뻗어 나갈 필요가 없다. 왜냐하면, 남은 경로는 더 이상 뻗어 나가지 않고도 알 수 있기 때문이다.

2001 -03 -19 알고리즘 강의 슬라이드 6 29



2001 -03 -19 알고리즘 강의 슬라이드 6 30

분기한정 최고우선검색

- 분기한정 가지치기로 최고우선 검색을 사용하기 위해서 각 마디의 한계치를 구할 수 있어야 한다. 이 문제에서는 주어진 마디에서 뻗어 나가서 얻을 수 있는 여행경로의 길이의 하한(최소치)을 구하여 한계치로 한다. 그리고 각 마디를 검색할 때 최소여행경로의 길이 보다 한계치가 작은 경우 그 마디는 유망하다고 한다. 최소여행경로의 초기값은 ∞로 놓는다. 따라서 완전한 여행경로를 처음 얻을 때 까지는 한계치가 무조건 최소여행경로의 길이보다 작게 되므로 모든 마디는 유망하다.
- 각 마디의 한계치는 어떻게 구하나?

  [1,…,k]의 여행경로를 가진 마디의 한계치는 다음과 같이 구한다. Let:

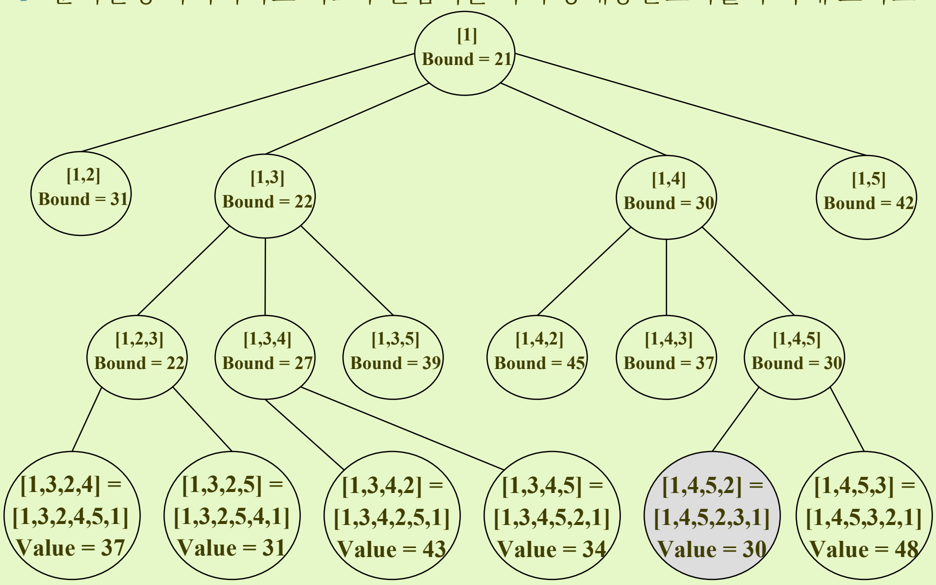
  $A = V$ - ([1,…,k] 경로에 속한 모든 마디의 집합)

  bound = [1,…,k] 경로 상의 총거리

  + v_k에서 A에 속한 정점으로 가는 이음선의 길이들 중에서 최소치

  + $\Sigma_{i \in A}$(v_i에서 $A \cup \{v_1\} - \{v_i\}$에 속한 정점으로 가는 이음선의 길이들 중에서 최소치)

- 분기한정 가지치기로 최고우선검색을 하여 상태공간트리를 구축해 보시오



분석

- 이 알고리즘은 방문하는 마디의 개수가 더 적다.
- 그러나 아직도 알고리즘의 시간복잡도는 지수적이거나 그보다 못하다!
- 다시 말해서 $n = 40$이 되면 문제를 풀 수 없는 것과 다름없다고 할 수 있다.
- 다른 방법이 있을까?
  - ✓ 근사(approximation) 알고리즘: 최적의 해답을 준다는 보장은 없지만, 무리 없이 최적에 가까운 해답을 주는 알고리즘.