ch.09 해 탐색 알고리즘

9.1 백트래킹(Backtracking) 기법

해를 찾는 도중에 '막히면'(해가 아니면) 되돌아가서 다시 해를 탐색하는 기법

→ 문제의 조건에 따라 해를 깊이 우선 탐색(DFS)로 찾음

→ 최적화(optimization) , 결정(decision) 문제 해결 가능

최적값 또는 최대값 구함

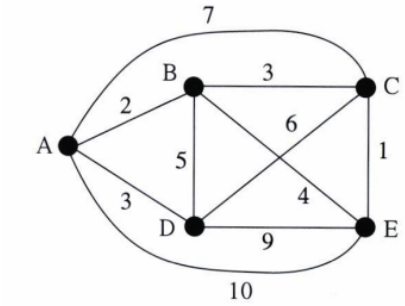
- 결정 문제 : 문제의 조건을 만족하는 해의 존재 여부 → yes or no

여행자 문제(TSP) → 해 탐색 알고리즘으로 해결

- bestSolution : 현재까지 찾은 가장 거리가 짧은 해 → (tour, tour의 거리)
- tour : 점의 순서(sequence)

```
tour = [시작점] //tour : 점의 순서(sequence)
bestSolution = (-, ∞) //bestSolution의 크기 -> 가장 큰 상수로 초기화

BacktrackTSP(tour)
if(tour가 완전한 해이면)
  if(tour의 거리 < bestSolution의 거리) //더 짧은 해를 찾았으면
    bestSolution = (tour, tour의 거리)
else{
  for(tour를 확장 가능한 각 점 v에 대해서){
    //현재 tour에 없는 점 -> 현재 tour의 가장 마지막 점과 간선으로 연결된 점

    newTour = tour + v //기존 tour의 뒤에 점 v를 추가 -> 확장 가능한 점 추가
    if(newTour의 거리 < bestSolution의 거리)
      BacktrackTSP(newTour) ⇒ 재귀호출
  } //만약 newTour > bestSolution -> 확장하여도 더 짧은 tour가 아니기에 가지치기
}
```

해가 완전한 해가 아닐 때 진행



tour = [A], bestSolution (-, ∞)

① bestSolution (tour)

newTour = [A, B] = 2 (경로가중치 2)

↓

bestSolution ([A, B])

② bestSolution ([A, B])

newTour = [A, B, C] = 2+3 = 5

↓

bestSolution ([A, B, C])

③ bestSolution ([A, B, C])

newTour = [A, B, C, D] = 2+3+6 = 11

↓

bestSolution ([A, B, C, D])

④ bestSolution ([A, B, C, D])

newTour = [A, B, C, D, E] = 2+3+6+9 = 20

↓

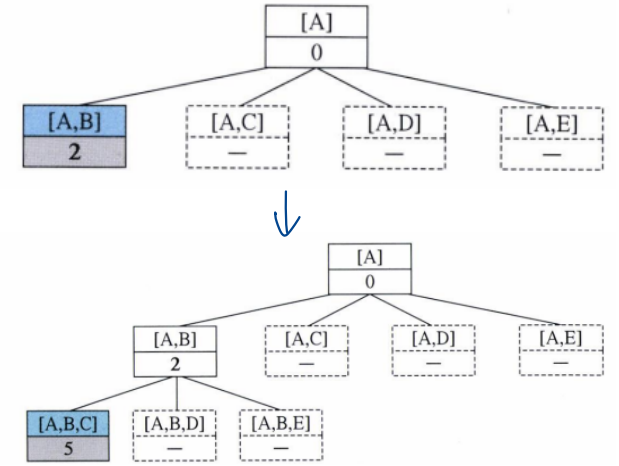
bestSolution ([A, B, C, D, E])

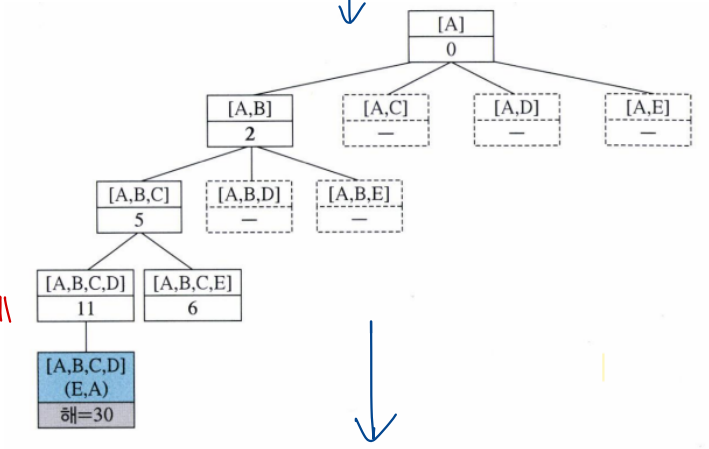
⑤ bestSolution ([A, B, C, D, E])

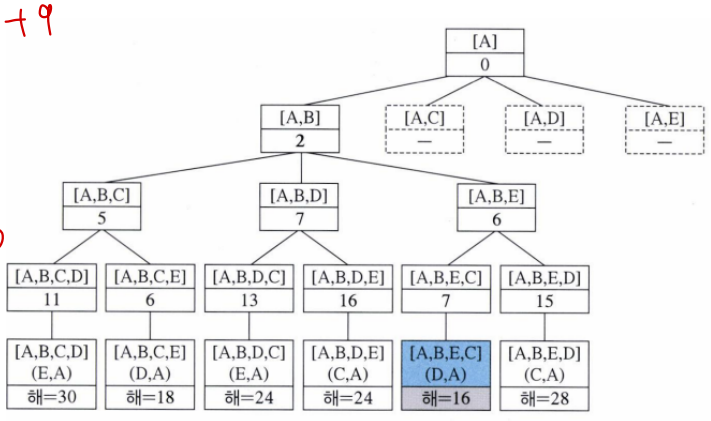
newTour = [A, B, C, D, E, A] = 30

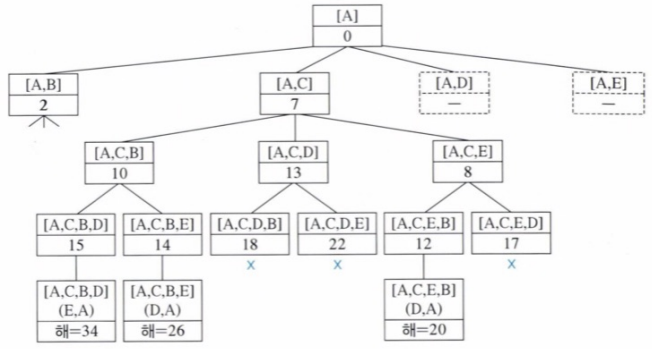
bestSolution ([A, B, C, D, E, A], 30)

첫번째 완전해 :





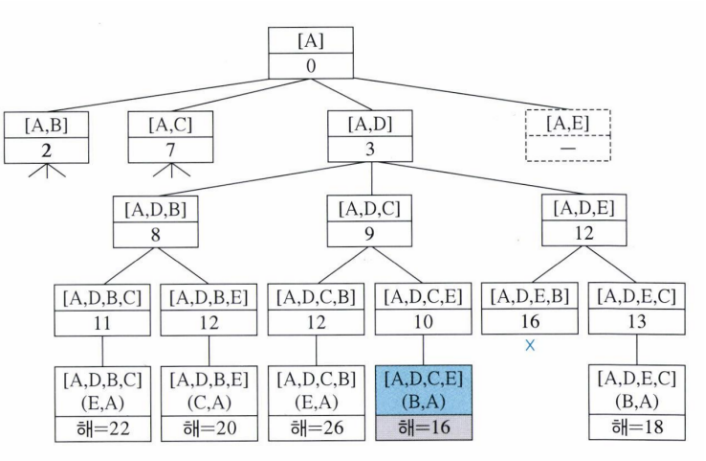




60보다 우수한 해는 탐색되지 X

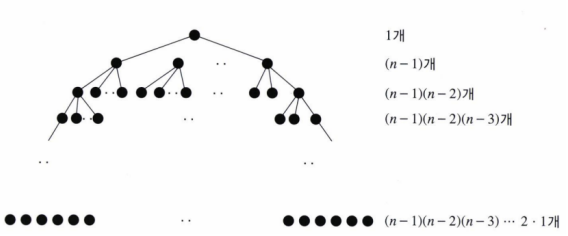
[A,E] 까지 다 탐색 후
16보다 우수한 해는 나오지 X

∴ 최종해 : [A,B,E,C,D,A]
16



Time Complexity

- state space tree의 node 개수에 비례
  - 최악의 경우 : 2^n개 → 지수 시간
    - 완전탐색의 시간복잡도와 같음.
  - but 일반적으로 '가지치기' ☆
    → 2^n 보다는 더 효율적



9.2 분기 한정 기법(Branch and bound)

backtracking → 트리에서 대부분의 노드를 탐색해야 하는 DFS (노드 우선순위 x)

branch and bound → 트리의 각 노드에 한정 값을 활용하여 우수한 한정 값 먼저 탐색하는 BFS

① 입력의 크기가 커져도 사용 가능

② backtracking보다 빠르게 해를 찾음

③ 최적화 문제를 해결하는데 적합

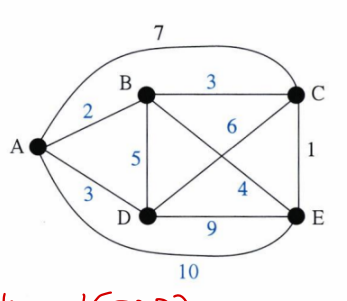
1. 최적해를 찾은 후에, 탐색한 노드의 한정값 ≥ 최적해 ⇒ 탐색 멈춤
2. 상태 공간 트리의 대부분의 노드 → 최적해 x
3. 최적해가 있을 만한 영역 먼저 탐색

```
BranchandBound(){
//S : 문제의 초기 상태
상태 S의 한정값을 계산
activeNodes = {S} //탐색되어야 하는 상태의 집합
bestValue = ∞ //현재까지 탐색된 해 중의 최솟값 -> 처음에는 가장 큰 수로 초기화

while(activeNodes != ∅){ //더 이상 탐색할 상태가 없다는 뜻
```

```
  Smin = activeNodes의 상태 중에서 한정값이 가장 작은 상태
  Smin을 activeNodes에서 제거한다.
  Smin의 자식(확장 가능)노드 "S'1, S'2, ... S'k"를 생성, 각각의 한정값 계산

  for i=1 to k{ //확장한 각 자식 S'i에 대해서
    if(S'i의 한정값 >= bestValue)
      S'i를 가지치기 //더 우수한 해가 없음
                  ①                ②
    else if("S'i가 완전한 해 "and "S'i의 값 < bestValue"){
      bestValue = S'i의 값
      bestSolution = S'i
    }
      else
      S'를 activeNodes에 추가
      //나중에 차례가 되면 S'i로부터 탐색을 수행하기 위해 추가
  }
}
```

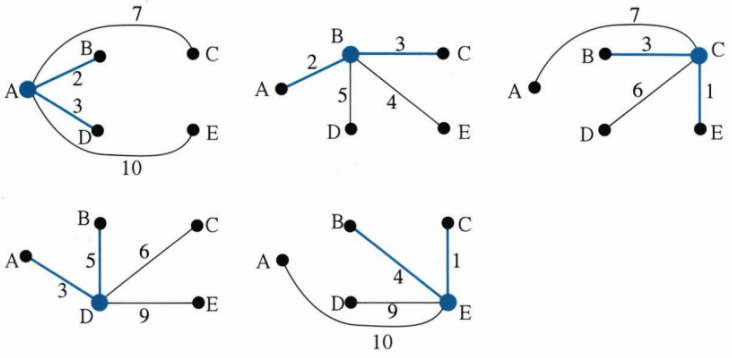
여행자 문제(TSP) 해결

1. 시작점에서 출발 → 모든 점을 1번만 방문 후 다시 시작점으로 돌아옴
2. 점 x의 한정값 : 시작점으로 돌아오는 경로의 예측 길이
   a. x에서 연결된 선분 중 가장 짧은 두 선분의 가중치의 평균의 합 → 예측 길이 계산 시 사용
      i. 가장 작은 점을 쓰지 않는 이유 : 한정값이 너무 작아져서 수행 오래 걸림
      ii. 큰 점을 쓰지 않는 이유 : 한정값이 너무 커져서 수행이 너무 빨리 됨

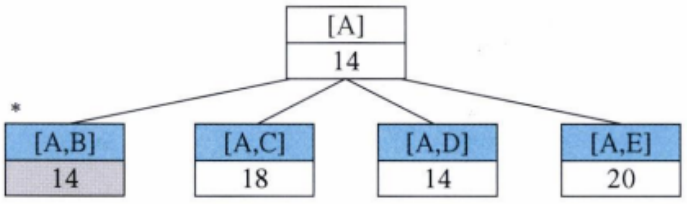


A B C D E

$[(2+3) + (2+3) + (1+3) + (3+5) + (1+4)] \times 1/2 = 27/2 = 14$



BranchandBound([A])
① activeNodes = {}
bestValue = ∞
Smin = [A]
activeNodes = ∅ ← 제거
$S_1 = [A,B]$
$S_2 = [A,C]$
$S_3 = [A,D]$
$S_4 = [A,E]$
$[A,B] = \left((2+3) + (2+8) + (1+3) + (3+5) + (1+4) \right) / 2 = 14$
↔ AB : 2
$[A,C] = ((2+7) + (2+3) + (1+7) + (3+5) + (1+4))/2 = 18$
↔ AC : 7



[A,D] = ((2+3)+(2+3)+(1+3)+(3+5)+(1+4)) / 2 = 14

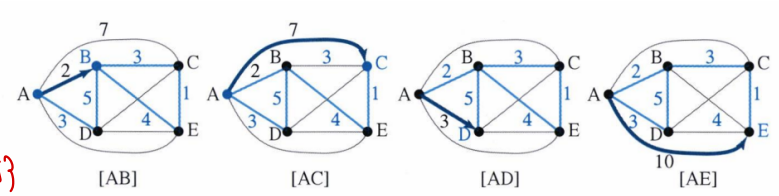
↔ AD : 3

[A,E] = ((2+10)+(2+3)+(1+3)+(3+5)+(1+10)) / 2 = 20

↔ AE : 10

모든 $S_1 \cdots S_4$ ⇒ 완전해 X

best Value보다 ↓



∴ activeNodes = {[A,B], [A,C], [A,D], [A,E]}

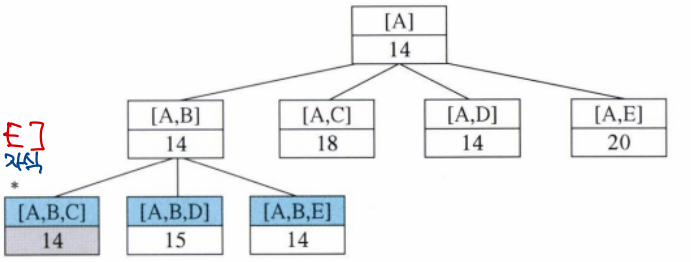
② activeNodes([A,B], [A,C], [A,D], [A,E])

S_{min} = [A,B] ([A,D]도 가능)

activeNodes = { [A,C], [A,D], [A,E])

$S_1 \cdots S_3$ = [A,B,C], [A,B,D], [A,B,E]

(각각, 각각, 각각, S_{min})



[A,B,C] = ((2+3)+(2+3)+(1+3)+(3+5)+((1+4)/2 = 14

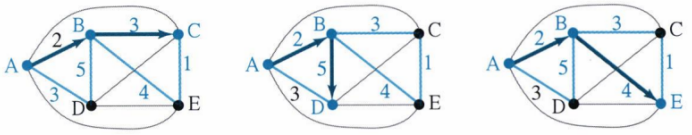
↔ AB + ↔ BC

2 3

[A,B,D] = ((2+3)+(2+5)+(1+3)+(3+5)+((1+4)) / 2 = 15

↔ AB + ↔ BD

2 5



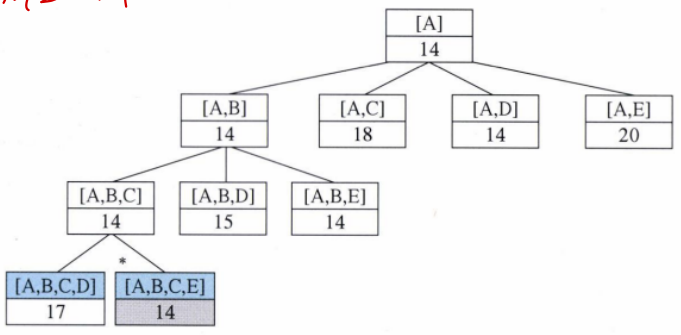
[A,B,E] = ((2+3)+(2+4)+(1+3)+(3+5)+((1+4))/2 = 14

↔ AB + ↔ BE

2 4

모든 $S_1 \cdots S_3$ = 완전해 X → best Value ↓

∴ activeNodes = { [A,C], [A,D], [A,E], [A,B,C], [A,B,D], [A,B,E] }

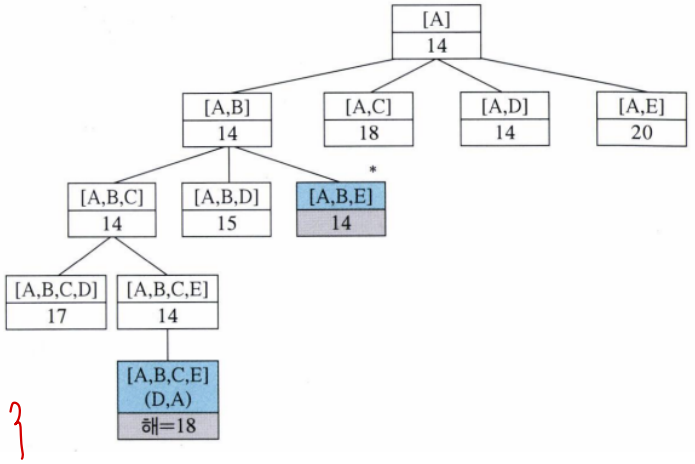


③ activeNodes = { [A,C], [A,D], [A,E], [A,B,C], [A,B,D], [A,B,E] }

S_{min} = [A,B,C] ([A,B,E], [A,D]도 가능)

activeNodes = { [A,C], [A,D], [A,E], [A,B,D], [A,B,E] }

$S_1 \cdots S_2$ = [A,B,C,D], [A,B,C,E]



④ activeNodes = { [A,C], [A,D], [A,E], [A,B,D], [A,B,E], [A,B,C,D], [A,B,C,E] }

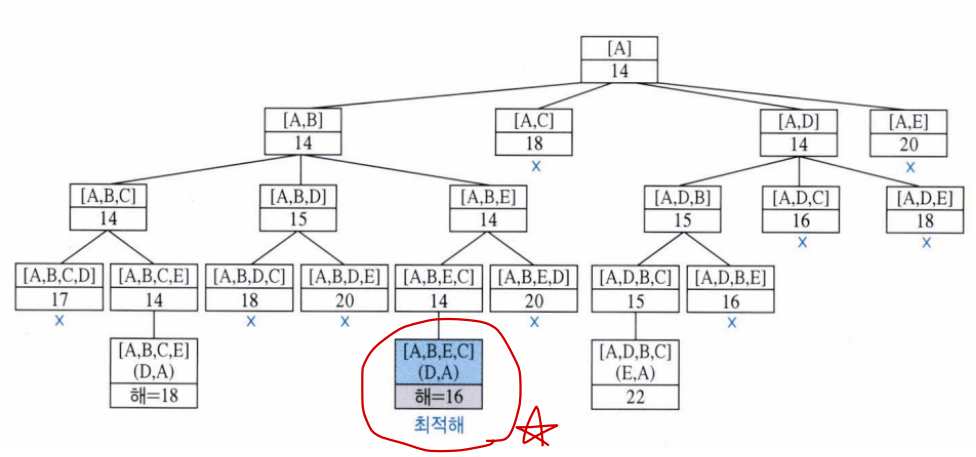
S_{min} = [A,B,C,E]

S_1 = [A,B,C,E,D]

↳ 완전해! best Value = 18

best Solution = [A,B,C,E,D,A]

: - - - -> [A,B,E] 혹은 [A,D] 부터 탐색 다시 시작



3. backtracking → 54개, branch and bound → 22개

   → 최적화 문제를 탐색 : 분기 한정이 훨씬 우수한 성능을 보임

   (한정값을 사용하여 최적해가 없다고 판단되는 부분은 탐색 x)