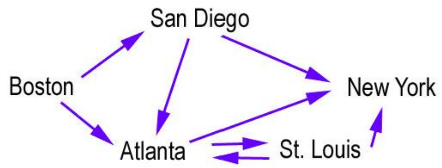
Combinatorial Optimization

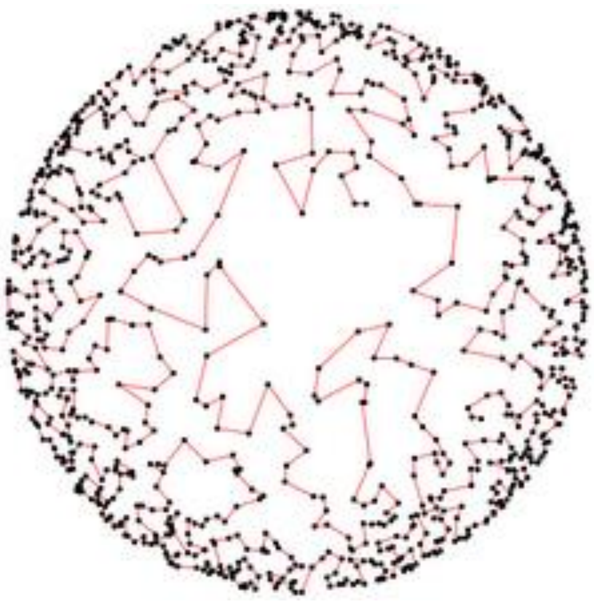
Combinatorial Optimization (조합최적화 문제)

- 정 의
  - Combinatorial optimization is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects, where the set of feasible solutions is discrete or can be reduced to a discrete set. - Wikipedia
- 종류
  - Typical combinatorial optimization problems are
    - the travelling salesman problem ("TSP"),
    - the minimum spanning tree problem ("MST"),
    - and the knapsack problem.
- 관련 분야
  - Combinatorial optimization is related to operations research, algorithm theory, and computational complexity theory.
- 응용분야
  - It has important applications in several fields, including
    - Artificial Intelligence,
    - Machine Learning,
    - Auction Theory,
    - Software engineering,
    - Applied mathematics
    - Theoretical computer science etc.

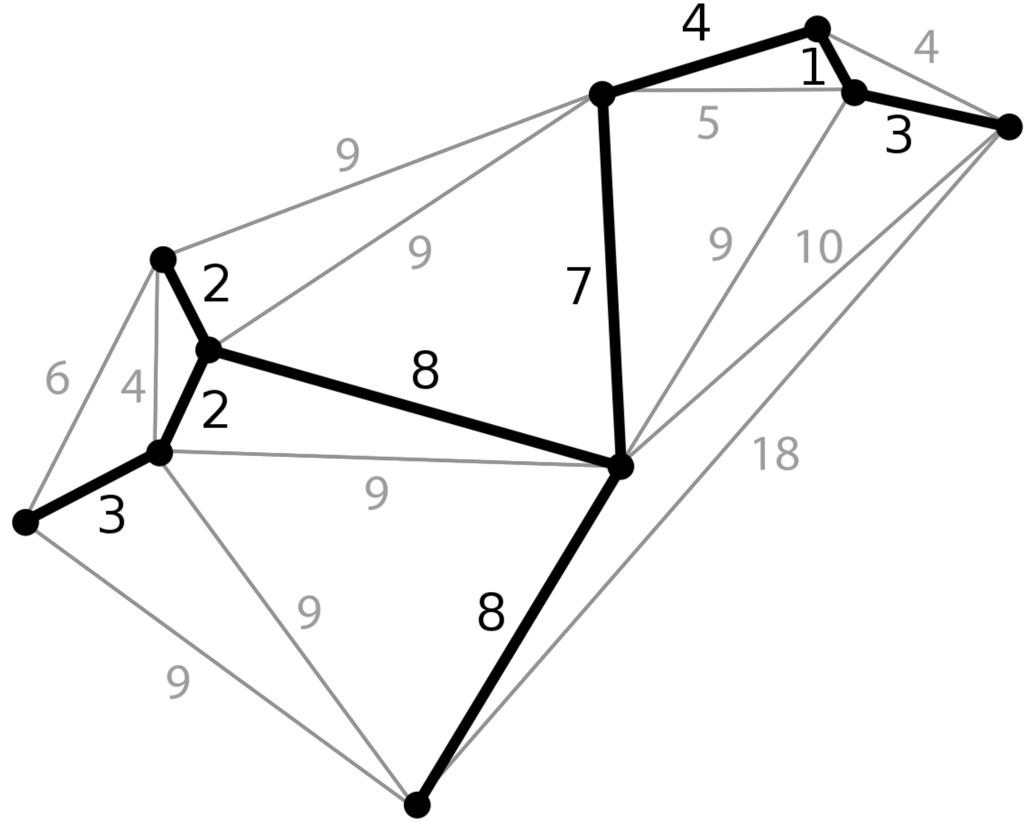
Traveling Salesman Problem(TSP)

- TSP (Traveling Salesman Problem)
- 들러야 할 도시들과 도시들 간의 거리가 인풋으로 주어지고, 모든 도시들을 들를 수 있는 최단거리를 찾는 문제

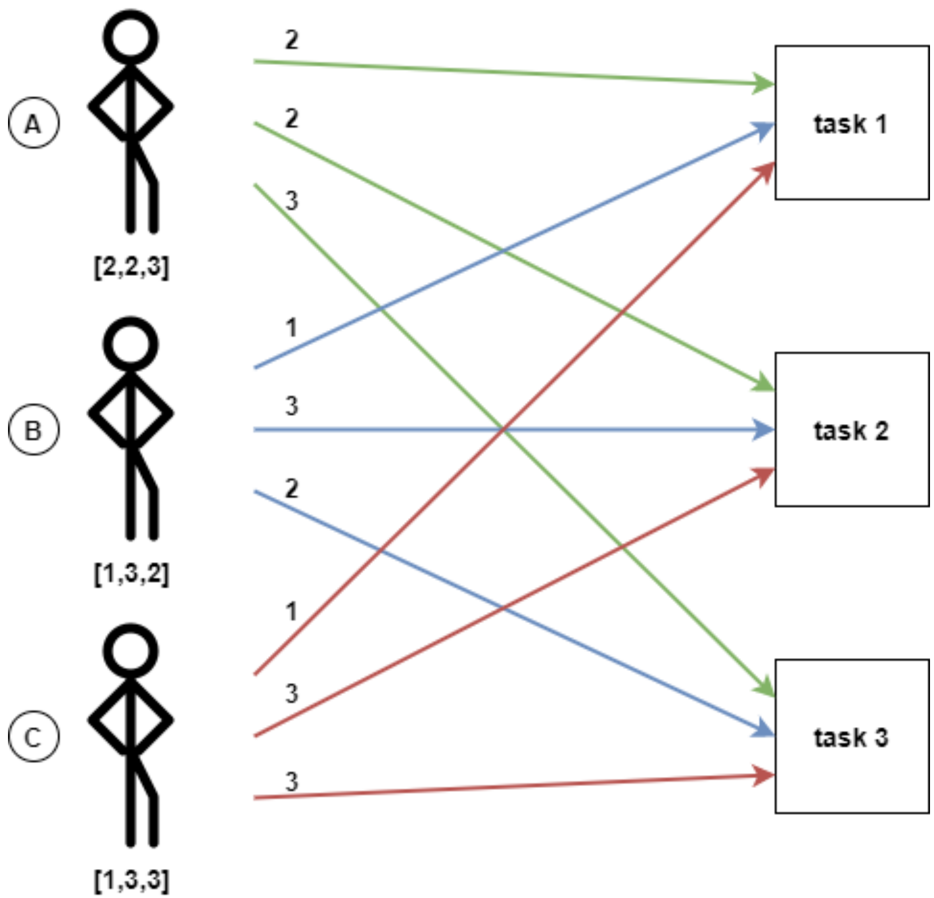




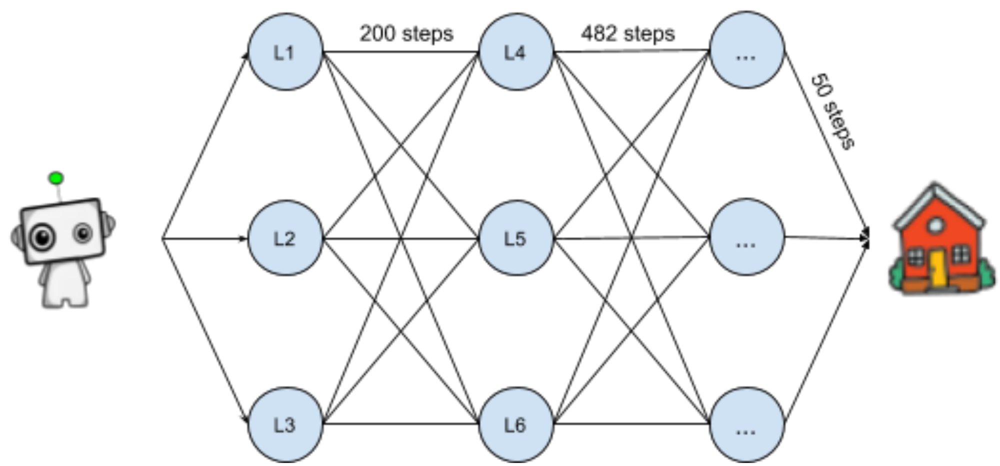
Minimum Spanning Tree



Assignment Problem(할당문제)



Robot Step Problem(로봇경로문제)



- 로봇의 환경
    - 위치 및 기타 정보가 이미 알려진 장애물들이 놓여진 정적인 환경
    - 또는 정보가 알려지지 않은 장애물이 갑작스럽게 나타나는 동적인 환경
- 목표지점까지 최적의 경로
    - 주어진 환경에서 장애물들을 효율적으로 회피
    - 최단거리, 최소비용, 안정성 등을 고려해경로를 탐색
    - 로봇의 핵심기술 중 하나

상태 공간 트리 (State Space Tree)

- 문제 해결 과정의 중간 상태들을 모두 Node로 구현해 놓은 트리
  - 경우의 수를 트리 형태로 표시
- 상태 공간 트리의 Leaf Node는 해당 문제의 해(Solution)에 해당
  - Optimum(최적해)는 Leaf Node 어딘가에 위치해 있음
  - 이를 효율적으로 찾아내는 것이 목표.
- 대표적인 해 탐색 알고리즘
  - 백트래킹
  - 분지한정법

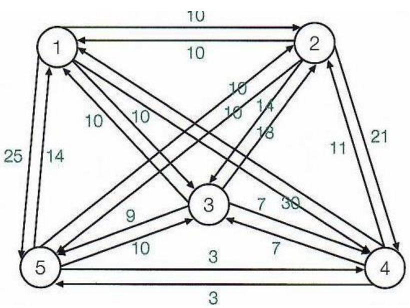
Brute Force (주먹구구식 탐색)

상태 공간 트리에 존재할 수 있는 모든 Leaf Node들을 계산하여 모든 경우의 수를 조사하는 방법
대부분의 경우에서, 허용치를 넘어선 나쁜 성능을 보임

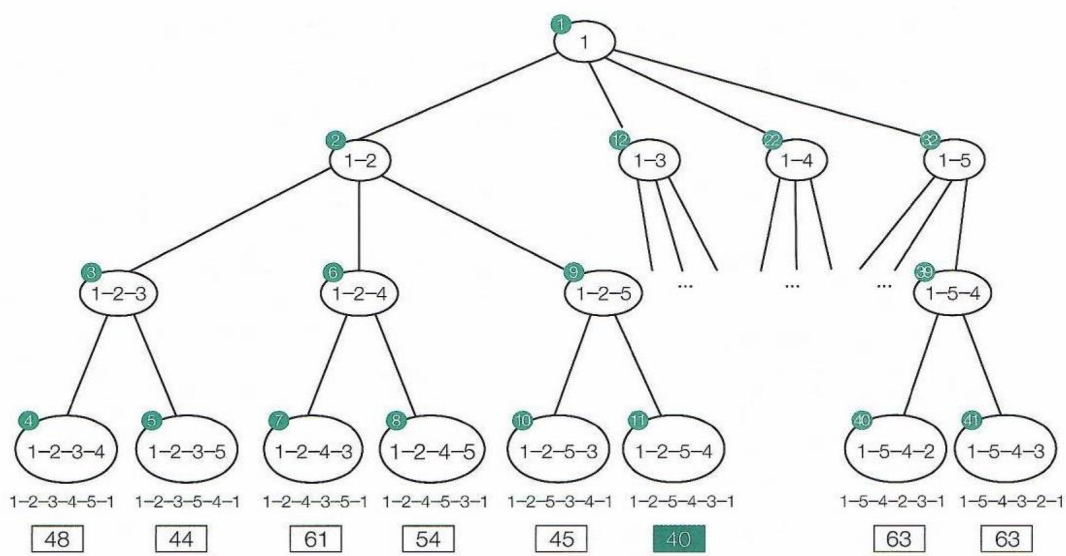
상태 공간 트리 (State Space Tree)의 예

- 여행자 문제
  - 모든 도시를 방문하고 출발 도시로 돌아오는 최단경로 찾기 문제

	1	2	3	4	5
1	0	10	10	30	25
2	10	0	14	21	10
3	10	18	0	7	9
4	8	11	7	0	3
5	14	10	10	3	0

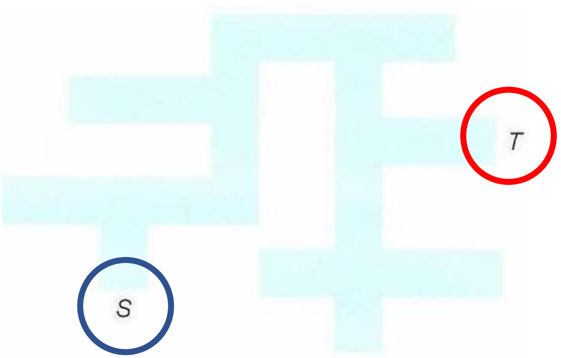


여행자문제

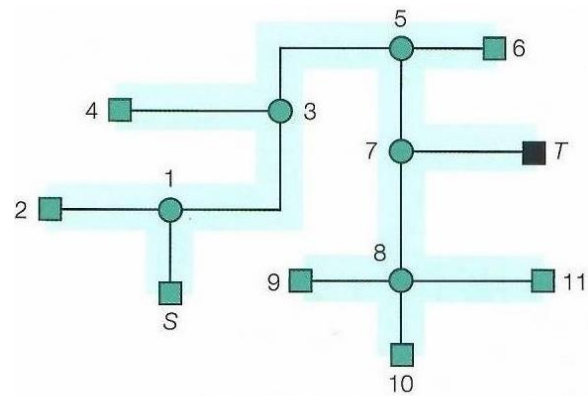


상태 공간 트리

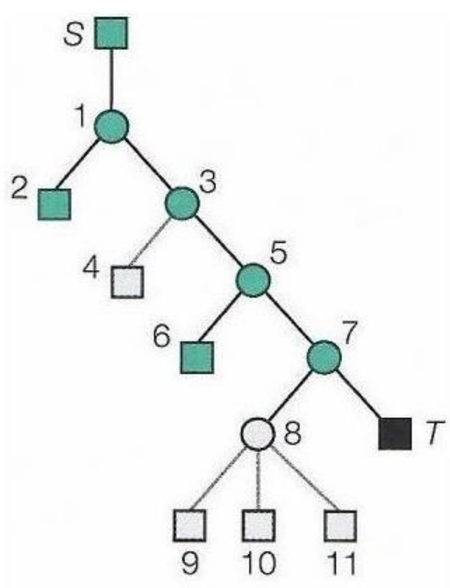
Mazing Problem | 미로 찾기 문제



S는 시작지점, T는 목표 지점



경로 중 하나를 선택해야 하는 분기점(O 표시)을 표시



시작 지점 S를 Root로 하는 상태 공간 트리

- 탐색
  - 회색 도형들은 운 좋게 방문하지 않고 탐색이 종료된 노드
  - 운이 좋으면 시행착오를 겪지 않고 T에 도달할 수 있고
  - 운이 나쁘면, 모든 노드들을 다 방문한 후에 T에 도달할 수도 있다.

제 9 장 해 탐색 알고리즘

9.1 백트래킹 기법

- 백트래킹 (Backtracking, 후퇴) 기법
  - 상태공간트리를 이용
  - 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법
    - DFS(Depth First Search (깊이 우선 탐색)

↔ 최적이든 아니든 가능해를 하나 찾아보자
얘보다 나쁜 건들은 찾아볼 필요가 없다

- 백트래킹 기법의 적용
  - 최적화 (optimization) 문제
  - 결정 (decision) 문제
  - 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'가 답하는 문제
    - 미로 찾기
    - 해밀토니안 사이클 (Hamiltonian Cycle) 문제
    - 컬러링 문제
    - 부분 집합의 합 (Subset Sum) 문제 등

여행자 문제(TSP)를 위한 백트래킹 알고리즘

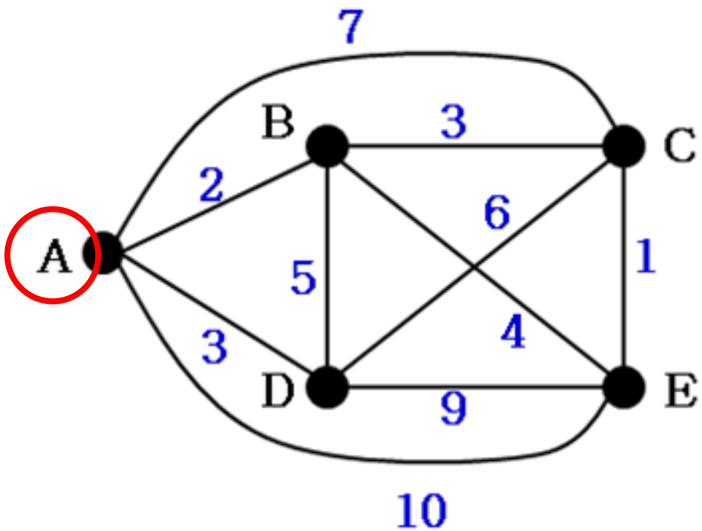
- 알고리즘에서 bestSolution은 현재까지 찾은 가장 우수한 (거리가 짧은) 해,
  - 2개의 성분 (tour, tour의 거리)으로 나타냄
- Tour는 점의 순서 (sequence)

- 초기화
  - tour = [시작점]   // tour는 점의 순서 (sequence)
  - bestSolution = (tour, ∞) // tour는 시작점만 가지므로 그 거리는 가장 큰 상수로 초기화

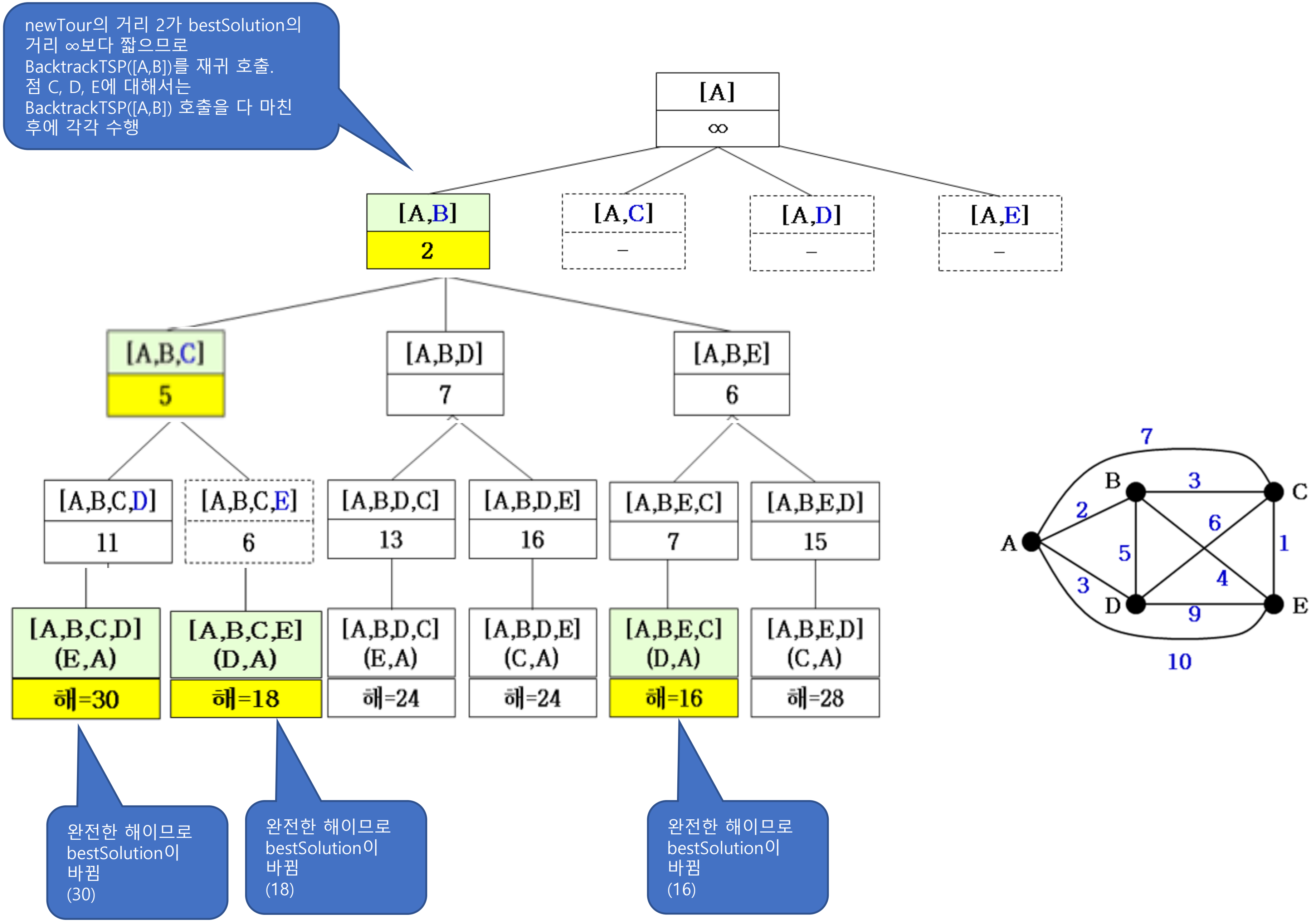
```
BacktrackTSP(tour)
1. if (tour가 완전한 해이면)
2.       if (tour의 거리 < bestSolution의 거리)   // 더 짧은 해를  찾았으면
3.             bestSolution = (tour, tour의 거리)
4.   else {
5.          for (tour를 확장 가능한 각 점 v에 대해서) {
6.                    newTour = tour + v        // 기존 tour의 뒤에  v 를 추가
7.                     if (newTour의 거리 < bestSolution의 거리)
8.                          BacktrackTSP(newTour)
            }
      }
```

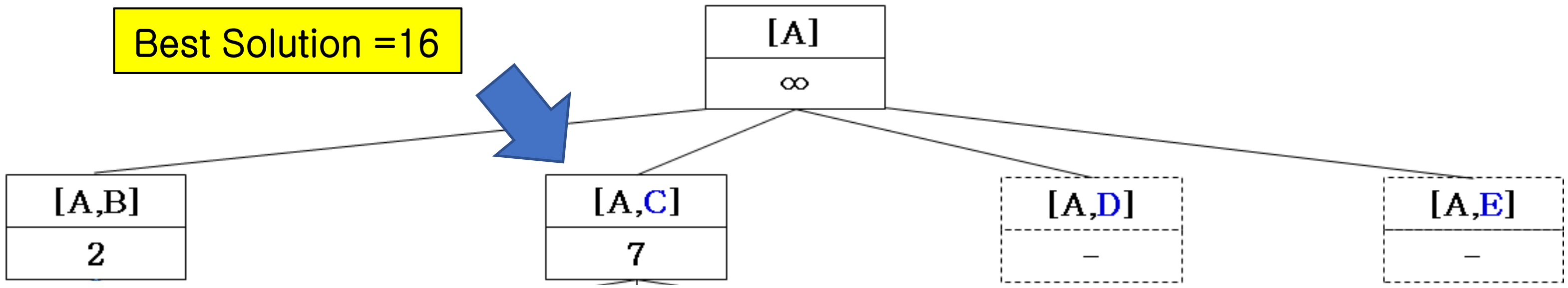
알고리즘의 수행 예제

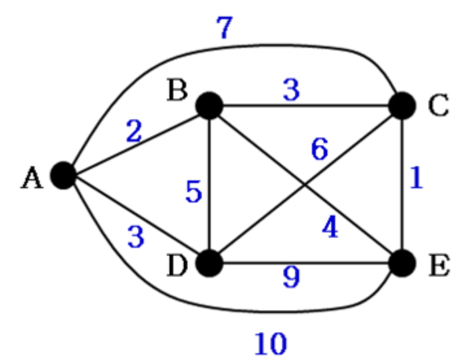
- 다음의 그래프에 대한 BacktrackTSP 알고리즘의 수행과정 (점 A=시작점)
  - 모든 노드를 방문하고 다시 A로 돌아오는 경로 찾기

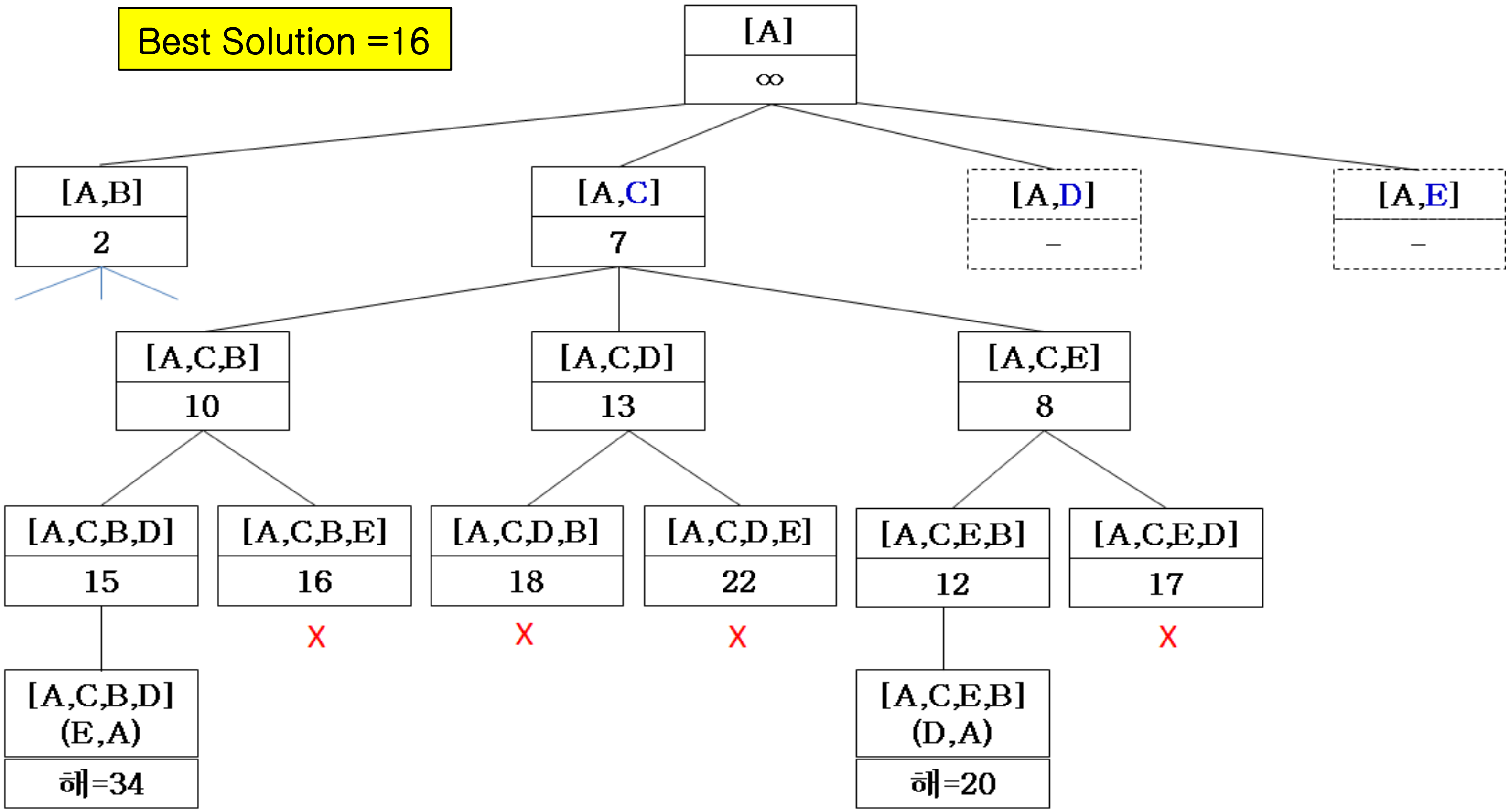


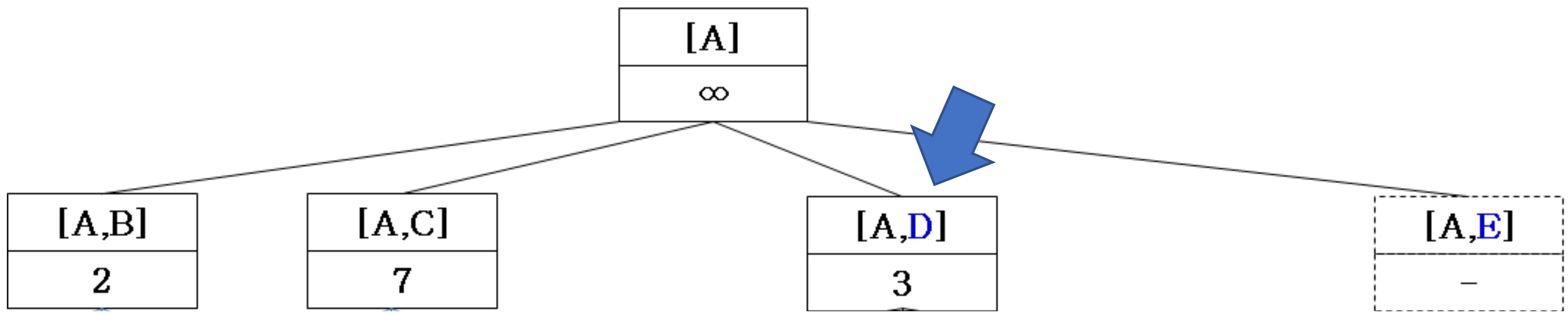
- 시작점이 A이므로, tour=[A]이고, bestSolution=([A],∞)이다.
- BacktrackTSP(tour)를 호출하여 해 탐색 시작

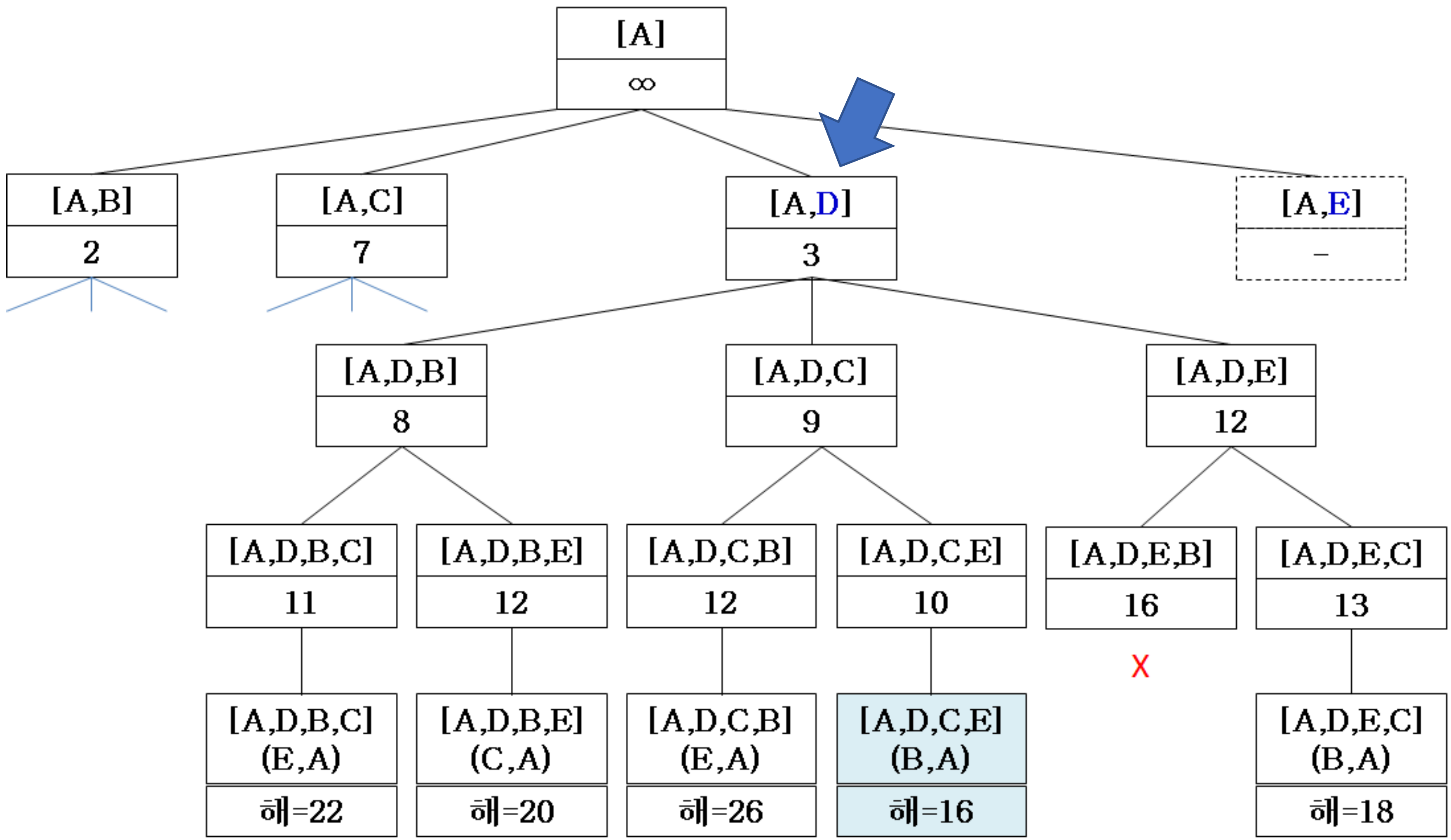

newTour의 거리 2가 bestSolution의 거리 ∞보다 짧으므로 BacktrackTSP([A,B])를 재귀 호출. 점 C, D, E에 대해서는 BacktrackTSP([A,B]) 호출을 다 마친 후에 각각 수행
[A]
∞
[A,B]
2
[A,C]
–
[A,D]
–
[A,E]
–
[A,B,C]
5
[A,B,D]
7
[A,B,E]
6
[A,B,C,D]
11
[A,B,C,E]
6
[A,B,D,C]
13
[A,B,D,E]
16
[A,B,E,C]
7
[A,B,E,D]
15
[A,B,C,D] (E,A)
해=30
[A,B,C,E] (D,A)
해=18
[A,B,D,C] (E,A)
해=24
[A,B,D,E] (C,A)
해=24
[A,B,E,C] (D,A)
해=16
[A,B,E,D] (C,A)
해=28
완전한 해이므로 bestSolution이 바뀜 (30)
완전한 해이므로 bestSolution이 바뀜 (18)
완전한 해이므로 bestSolution이 바뀜 (16)
A
B
C
D
E
7
3
2
6
1
5
4
3
9
10


Best Solution =16
[A]
∞
[A,B]
2
[A,C]
7
[A,D]
–
[A,E]
–


7
B
3
C
2
6
A
1
5
3
4
D
9
E
10

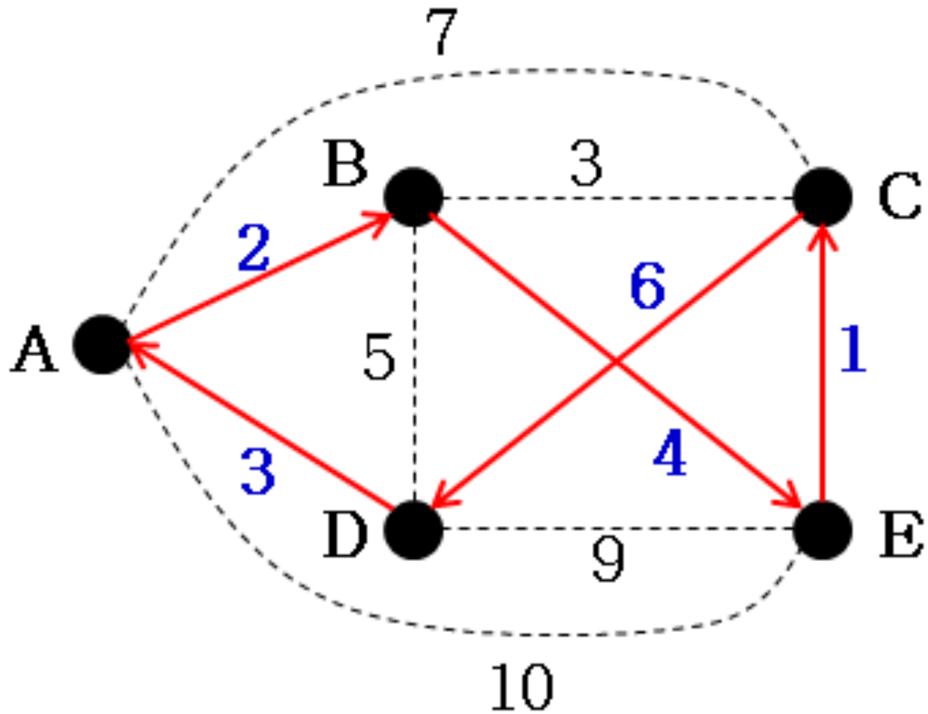

Best Solution =16
[A]
∞
[A,B]
2
[A,C]
7
[A,D]
–
[A,E]
–
[A,C,B]
10
[A,C,D]
13
[A,C,E]
8
[A,C,B,D]
15
[A,C,B,E]
16
X
[A,C,D,B]
18
X
[A,C,D,E]
22
X
[A,C,E,B]
12
[A,C,E,D]
17
X
[A,C,B,D]
(E,A)
해=34
[A,C,E,B]
(D,A)
해=20


[A]
∞
[A,B]
2
[A,C]
7
[A,D]
3
[A,E]
–


[A]
∞
[A,B]
2
[A,C]
7
[A,D]
3
[A,E]
–
[A,D,B]
8
[A,D,C]
9
[A,D,E]
12
[A,D,B,C]
11
[A,D,B,E]
12
[A,D,C,B]
12
[A,D,C,E]
10
[A,D,E,B]
16
[A,D,E,C]
13
X
[A,D,B,C] (E,A)
해=22
[A,D,B,E] (C,A)
해=20
[A,D,C,B] (E,A)
해=26
[A,D,C,E] (B,A)
해=16
[A,D,E,C] (B,A)
해=18

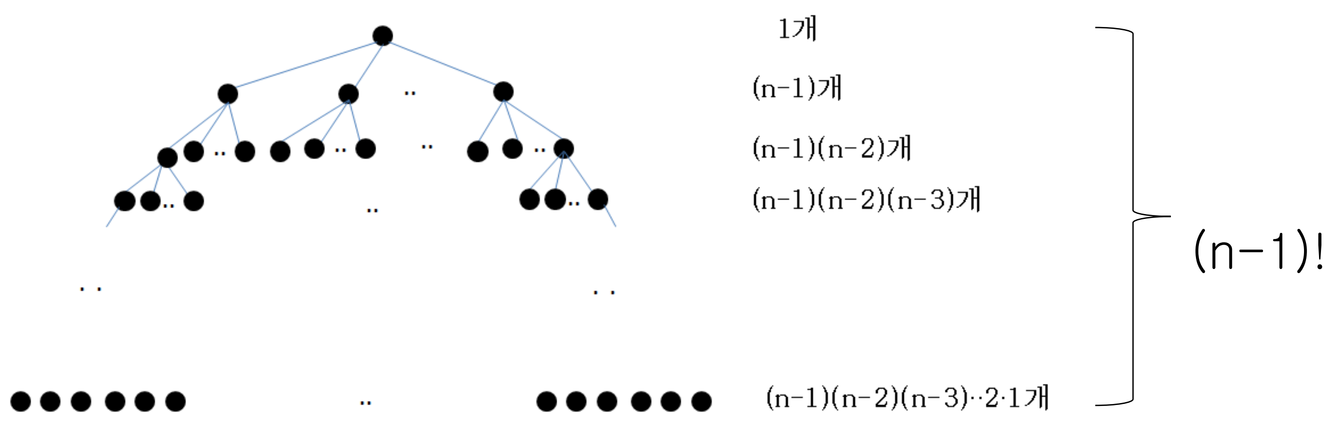
- 마지막으로 tour=[A,D]에 대해서 탐색을 수행하여도 bestSolution보다 더 우수한 해는 발견되지 않음

- 따라서 최종해= [A,B,E,C,D,A]이고, 거리=16이다.



시간복잡도

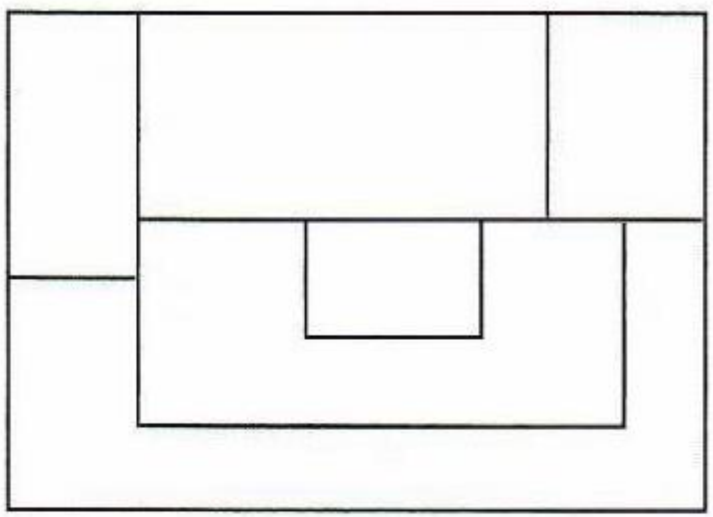
- Backtracking 알고리즘의 시간 복잡도는 상태 공간 트리의 노드 수에 비례
- n개의 점이 있는 입력 그래프에 대해서 BacktrackTSP 알고리즘이 탐색하는 최대 크기의 상태 공간 트리



- 최악의 경우 모든 경우를 다 검사하여 해를 찾는 완결 탐색 (Exhaustive Search)의 시간복잡도와 같음
- 그러나 일반적으로 백트래킹 기법은 ‘가지치기(pruning 혹은 fathom)’를 하므로 완결 탐색보다 훨씬 효율적임

Graph Coloring Problem 그래프 색칠 문제

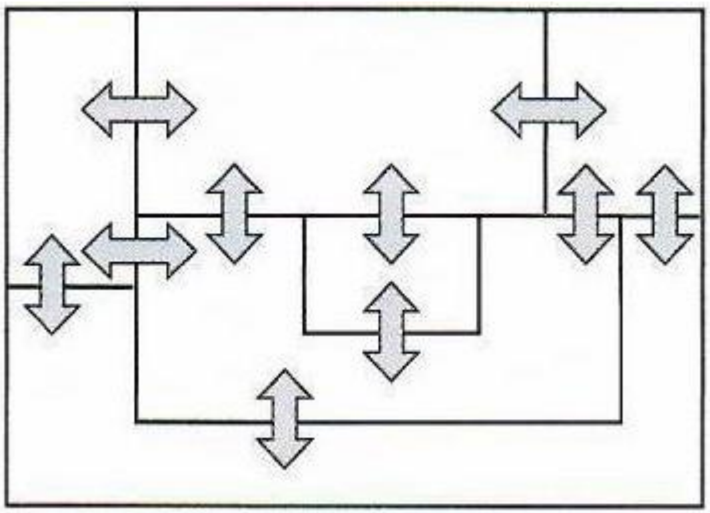
- 주어진 그래프에 k개의 색상을 사용하여 면을 색칠
- 인접한 정점에는 같은 색깔이 칠해지지 않도록 그래프의 모든 면을 색칠
  - 최소 개수의 색으로 모든 면을 칠하는 문제



(a) 지도

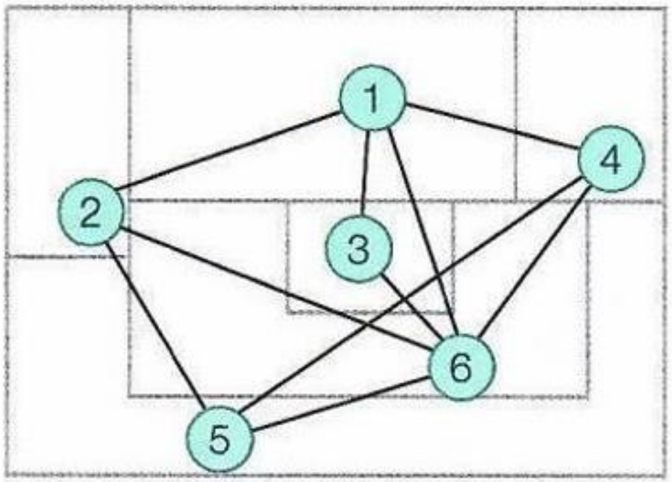
문제의 변환

- 주어진 지도에서 각 면들의 인접 관계를 화살표로 표현해 보자



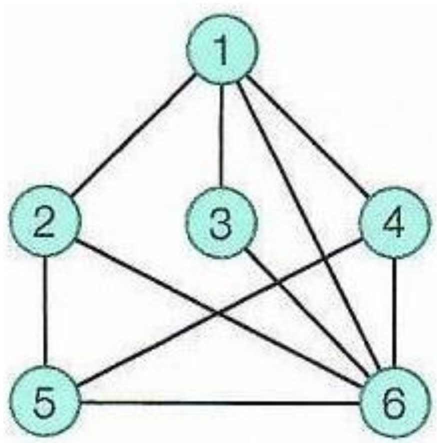
(b) 구역 간의 인접 관계

- 각각의 평면들은 그래프의 노드
- 각각의 화살표들은 정점들을 잇는 간선으로 표시해 보면



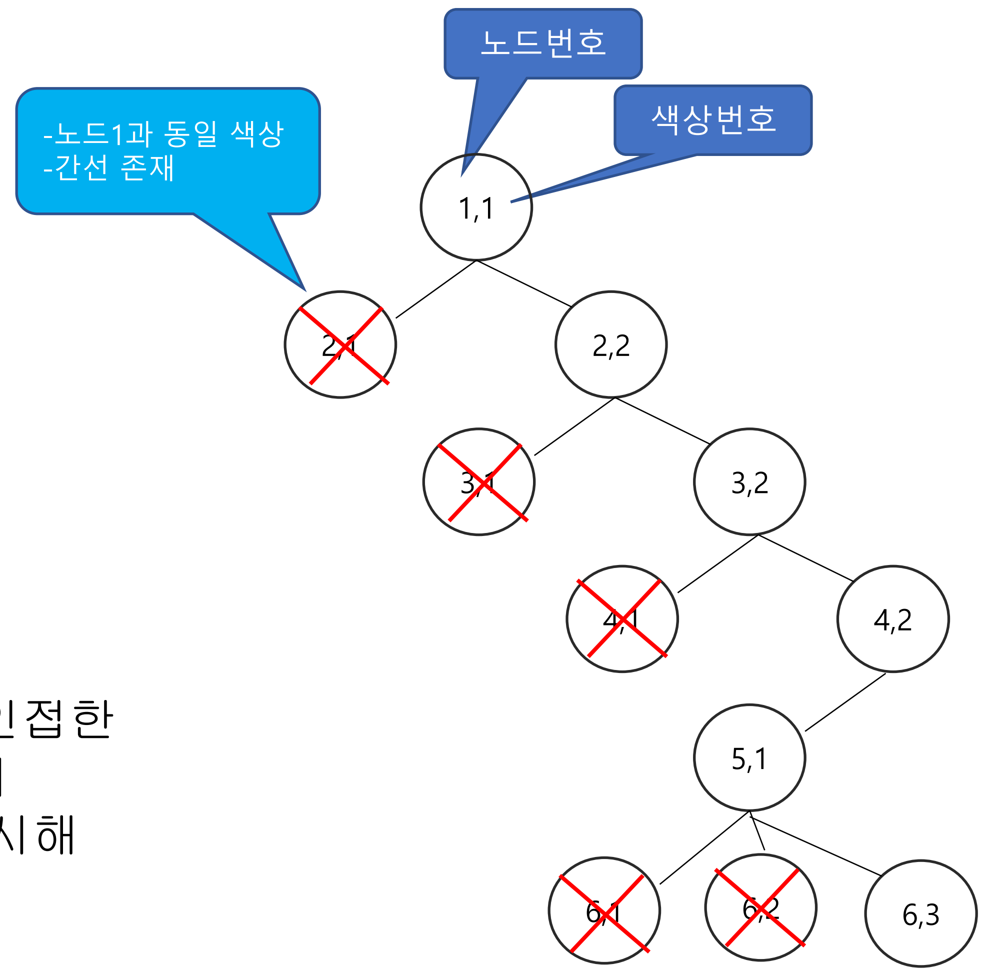
(c) 연결 관계를 정점과 간선으로 나타낸 것

상태공간트리 작성



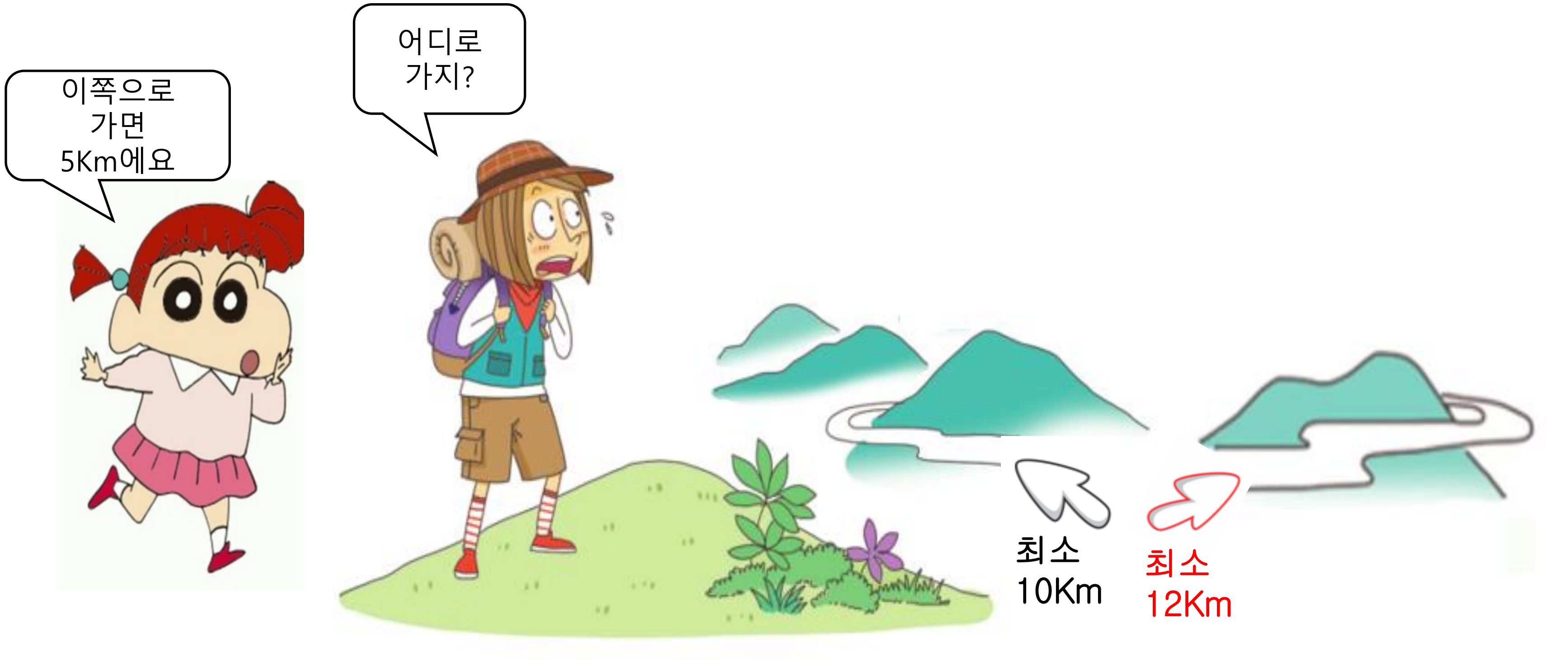
(d) (c)와 동일한 그래프

백트래킹 알고리즘을 통해 인접한 노드끼리는 같은 색을 칠하지 않도록 하는 경우의 수를 표시해 보자



Solution = {(1,1), (2,2), (3,2), (4,2), (5,1), (6,3)}

9.2 Branch-and-Bound Method
(분기한정법)


이쪽으로 가면 5Km에요
어디로 가지?
최소 10Km
최소 12Km

분기 한정 (Branch-and-Bound) 기법

- 백트래킹 기법은 깊이 우선 탐색수행
- 최적화 문제에 대해서는 최적해가 상태 공간 트리의 어디에 있는지 알 수 없으므로, 트리에서 대부분의 노드를 탐색하여야 함
- 입력의 크기가 커지면 해를 찾는 것은 거의 불가능
- 분기 한정(Branch-and-bound) 기법은 이러한 단점을 보완하는 탐색 기법

탁월한

- **분기 한정 기법의 효율적인 탐색 원리**

- 분기 한정 기법은 상태 공간 트리의 각 노드 (상태)에 특정한 값 (한정값)을 부여
  - 한정값 : Lower Bound. 가장 이상적인 값으로 가능하지 않은 해일 수도 있음
  - 한정값을 계산하는 방법은 문제에 따라 다르다.
  - 해를 찾은 후에, 탐색하여야 할 나머지 노드의 한정값이 지금까지의 최선해의 값과 같거나 나쁘면 더 이상 탐색하지 않는다
  - 노드의 한정값을 활용하여 가지치기를 함으로서 백트래킹 기법보다 빠르게 해를 찾음

- 최적해가 있을 만한 영역을 먼저 탐색한다
  - 분기 한정 기법에서는 가장 우수한 한정값을 가진 노드를 먼저 탐색하는 최선 우선 탐색 (Best First Search, BFS)으로 해를 찾음

- 상태 공간 트리의 대부분의 노드가 문제의 조건에 맞지 않아서 해가 되지 못한다.

Branch and Bound Method

- 절차
  - 문제를 몇 개의 Category로 분류(branch)하고
  - 최적해가 될 수 없는 Category는 통째로 제외(cut 혹은 fathom)하는 방법

① Branch(가지치기)
  - 해결 가능한 모든 해결책을 고려
  - 모든 범주를 부분적 해결책(조건)에 의해 두 개 혹은 그 이상의 수의 범주로 나누어 해를 구한다.

② Bound (한정값, 최소화 문제 기준)
  - Lower bound (최대화 문제의 경우에는 Upper Bound)
    - Branch 단계에서 생성된 Category 에 있는 모든 방법 중 최저 한계치
    - 실현 가능하지 않은 해(infeasible solution)여도 무방함
  - Best Solution
    - 지금까지 얻은 문제 해결법 중 실현 가능한 최고의 해 (best feasible solution)
      - 이 것보다 크면 최적해가 될 수 없다는 뜻
      - 해를 구하는 도중 더 좋은 방법을 발견했을 때 Best Solution의 값은 지금까지 나온 실행가능한 해 중 최고의 해를 새로운 Best Solution으로 함

③ Cut (Fathom, Pruning, 가지치기)

- 지금까지 얻은 최고의 해, 즉, Best Solution보다 큰 lower bounds를 갖는 branch를 없애는 과정
  - 최선을 다해 줄여도 다른 branch의 현재까지의 해보다 더 크다면?
  - 더 이상 탐색 필요 없음

④ 반복

- 더 이상의 검사할 Category가 없을 때 까지 반복
- 지금까지 얻은 최고의 해결책은 최적의 해결책

Branch and Bound Method의 절차

1. Make a branch
2. Compute a Lower bound for each category produced
3. Compare each lower bound with the best solution obtained so far, and may cut
4. If no more branches are possible , you have an optimal solution and may stop
5. Otherwise, consider the category with the smallest lower bound, and go to step 1 above

알고리즘

Branch-and-Bound(S)

1. 상태 S의 한정값을 계산한다.
2. activeNodes = { S } // 탐색되어야 하는 상태의 집합
3. bestValue = ∞ // 현재까지 탐색된 해 중의 최소값
4. while (activeNodes ≠∅) {
5. S_{min}= activeNodes의 상태 중에서 한정값이 가장 작은 상태
6. S_{min}을 activeNodes에서 제거한다.
7. S_{min}의 자식 (확장 가능한) 노드 S'_1, S'_2, …, S'_k를 생성하고, 각각의 한정값을 계산한다.
8. for i=1 to k { // 확장한 각 자식 S'_i에 대해서
9. if (S'_i의 한정값 ≥ bestValue)
10. S'_i를 가지치기한다. // S'_i로부터 탐색해도 더 우수한 해가 없다.
11. else if (S'_i가 완전한 해이고 S'_i의 값 < bestValue)
12. bestValue = S'_i의 값
13. bestSolution = S'_i
14. else
15. S'_i를 activeNodes에 추가한다. // 나중에 차례가 되면 S'_i로부터 탐색을 수행

}

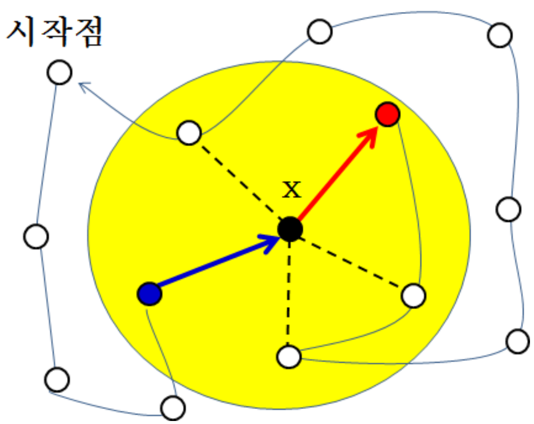
}

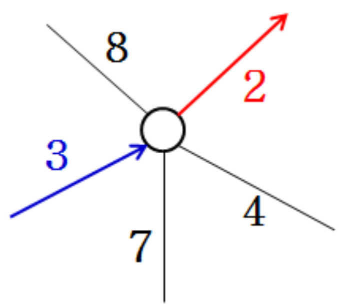
TSP를 분기 한정 기법으로 해결하는 과정

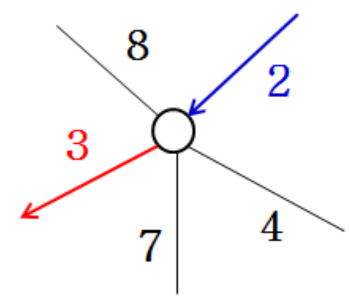
- 한정값 계산을 위해서 여행자 문제의 조건
  - 해는 주어진 시작점에서 출발하여 모든 다른 점을 1번씩만 방문하고 시작점으로 돌아와야 한다.
  - 이러한 경로 상의 1개의 점 x를 살펴보면, 다른 점에서 점 x로 들어온 후에 점 x를 떠나 또 다른 점으로 나간다.

한정값(Lower Bound)의 계산 방법:

- 점 x로 들어올 때와 나갈 때 사용되는 선분의 가중치를 활용
- X 노드에서 가장 이상적인 순회 방법
  - 가중치 3인 선분으로 들어와서 가중치 2인 선분으로 나가든지 (왼쪽 그림)
  - 가중치 2인 선분으로 들어와서 가중치 3인 선분으로 나가든지 (오른쪽 그림)
  - 두 경우 모두 최소의 비용으로 이 점을 방문하는 것
- 노드 x에서의 한정 값 연결된 선분 중에서 가중치가 가장 작은 두 선분의 가중치의 합의 1/2
  - 한 점에서 나가는 선분은 인접한 (다른) 점으로부터 들어오는 선분과 동일하기 때문

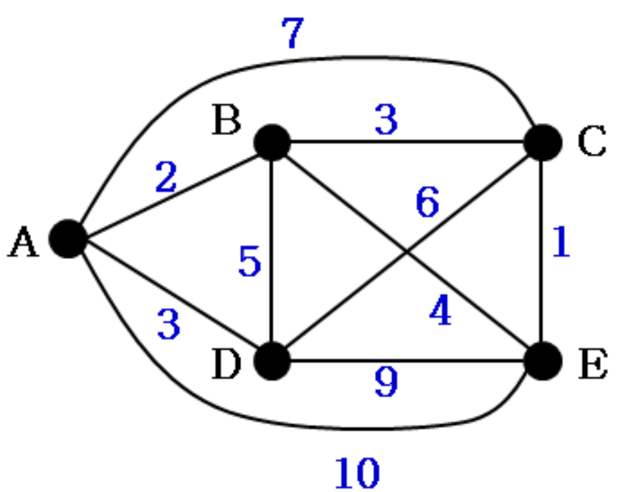






Branch-and-Bound 알고리즘 수행과정

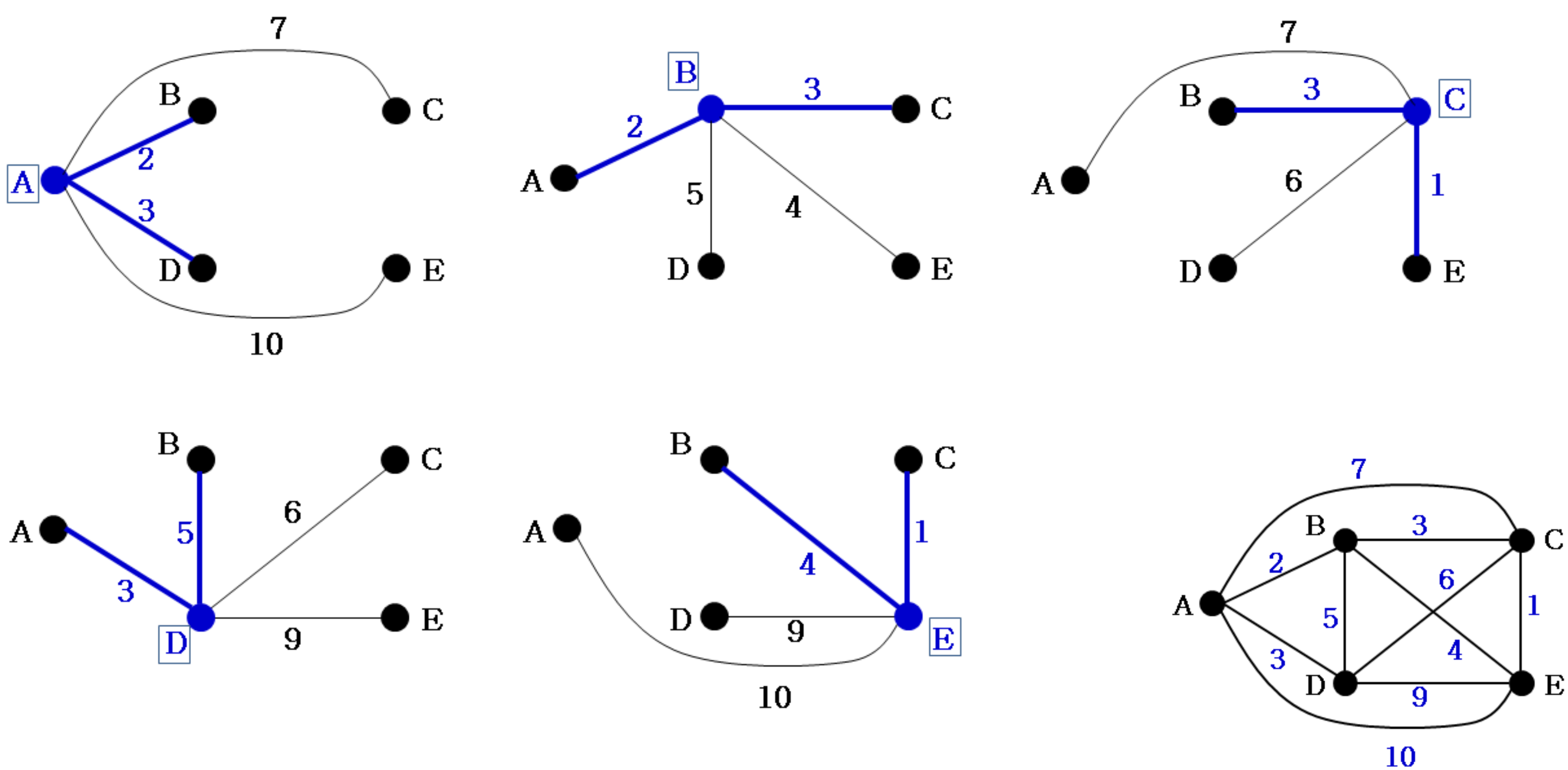
- A=시작점
- 초기 상태= [A]
- Branch-and-Bound([A])를 호출하여 탐색 시작



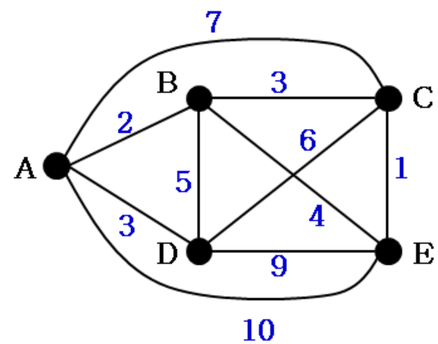
- 초기 상태 [A]의 한정값(Lower Bound) 계산
  - 각 점에 인접한 선분의 가중치 중에서 가장 작은 2개의 가중치의 합을 구한 다음에, 모든 점의 합의 1/2을 한정 값으로 정한다.

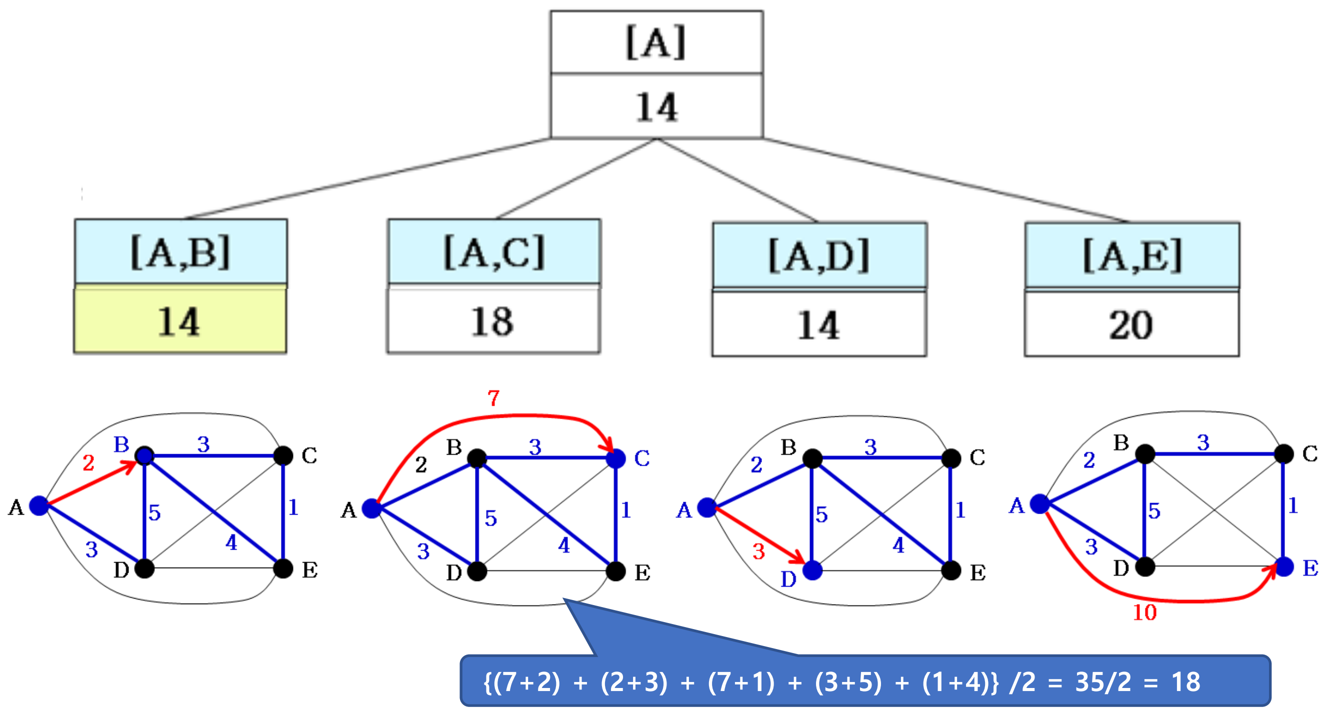
A B C D E

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

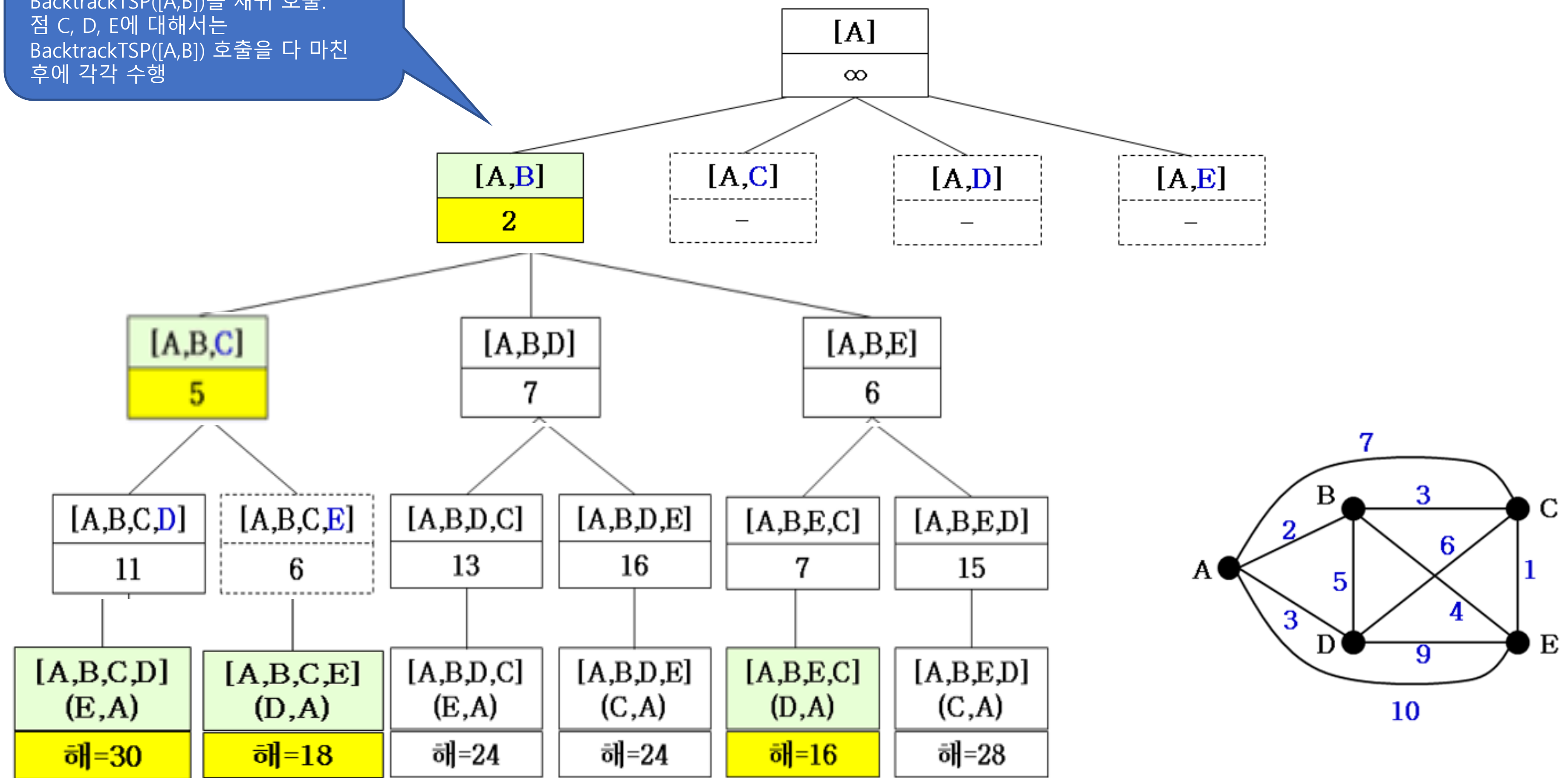


- Branch하고 Bound 구하기
  - 상태 [A])의 자식 상태 노드를 생성(Branch)하고, 각각 한정값을 구한다
  - Branch
  - 자식 노드는 두 번째 방문하는 점이
    - B인 상태 [A,B]
    - C인 상태 [A,C],
    - D인 상태 [A,D],
    - E인 상태 [A,E]





newTour의 거리 2가 bestSolution의 거리 ∞보다 짧으므로 BacktrackTSP([A,B])를 재귀 호출. 점 C, D, E에 대해서는 BacktrackTSP([A,B]) 호출을 다 마친 후에 각각 수행



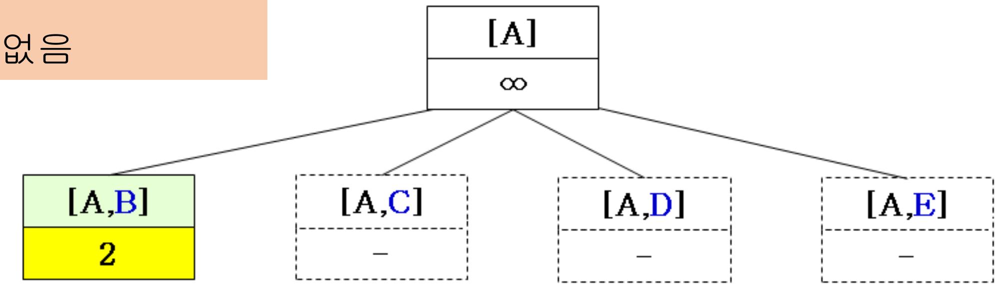
완전한 해이므로 bestSolution이 바뀜 (30)

완전한 해이므로 bestSolution이 바뀜 (18)

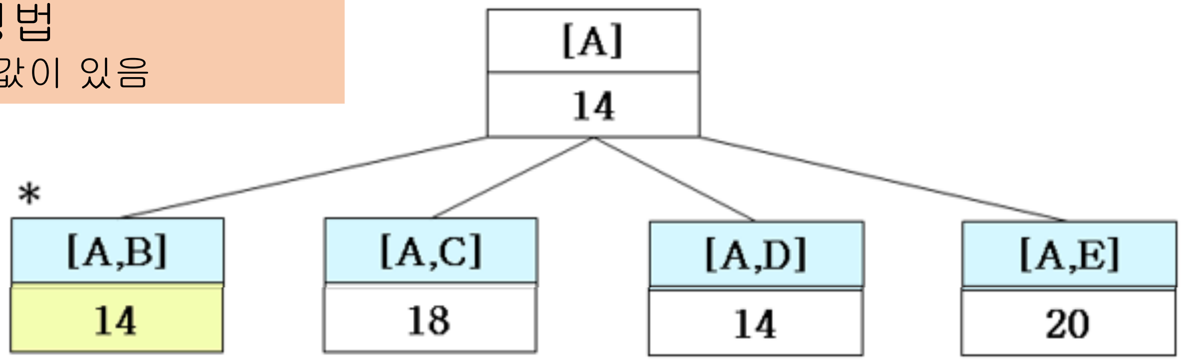
완전한 해이므로 bestSolution이 바뀜 (16)

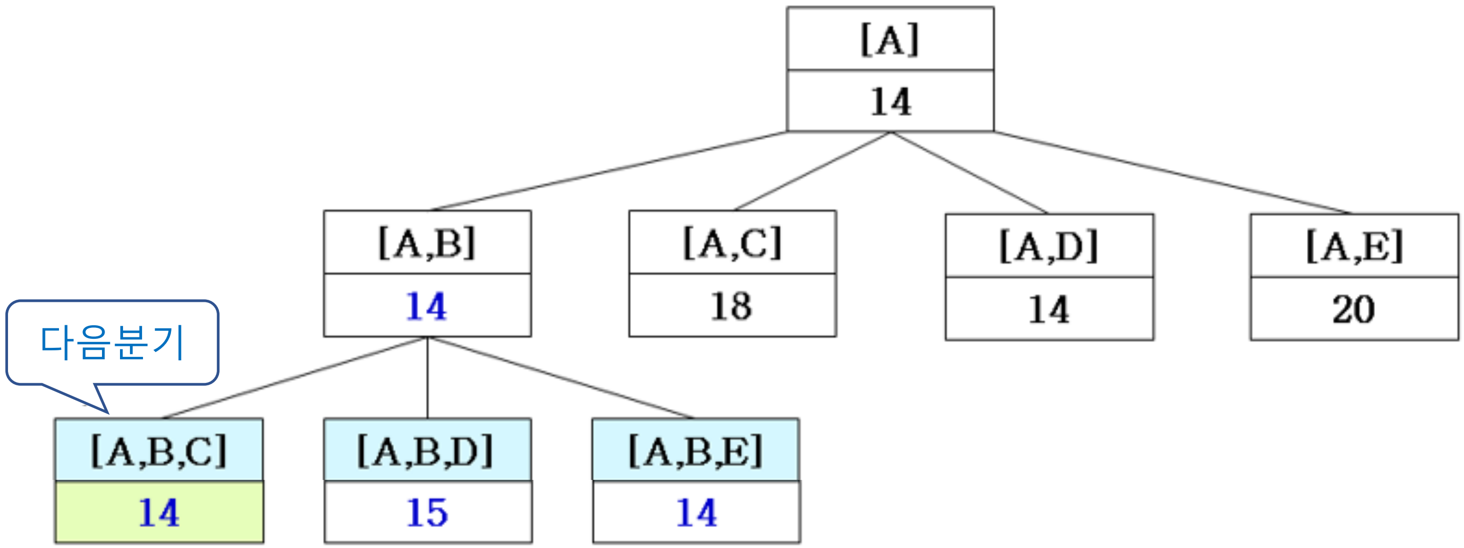
백트래킹과 분지한정법과의 차이

- 백트래킹
  - 한정값이 없음

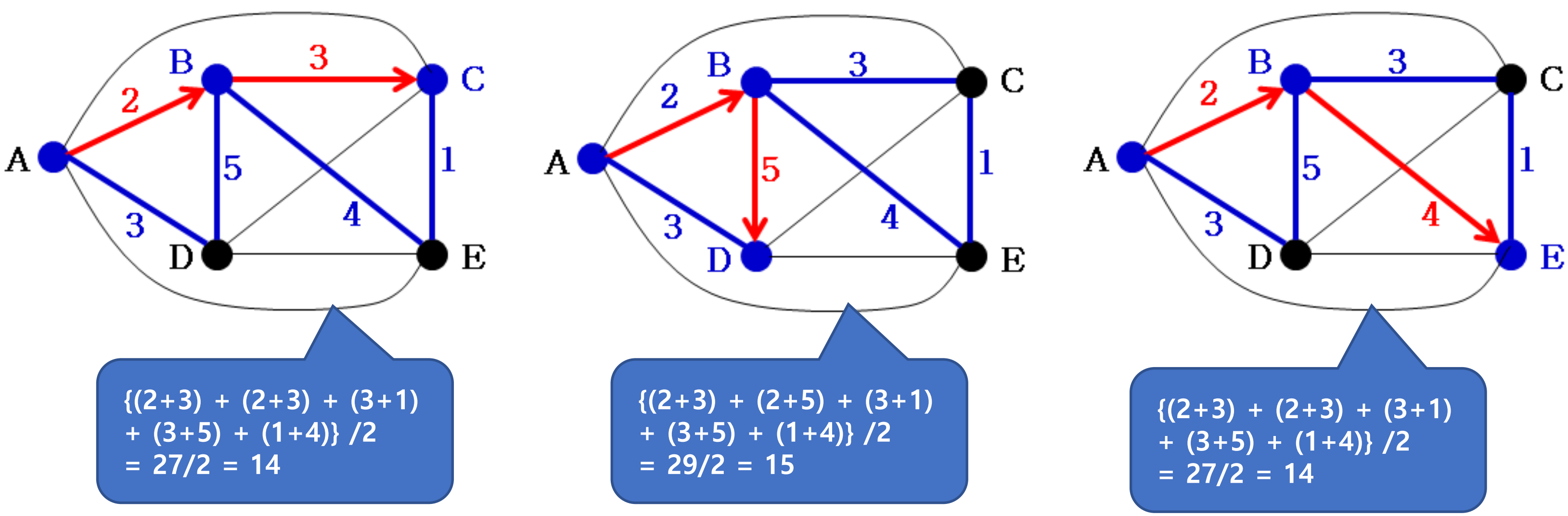


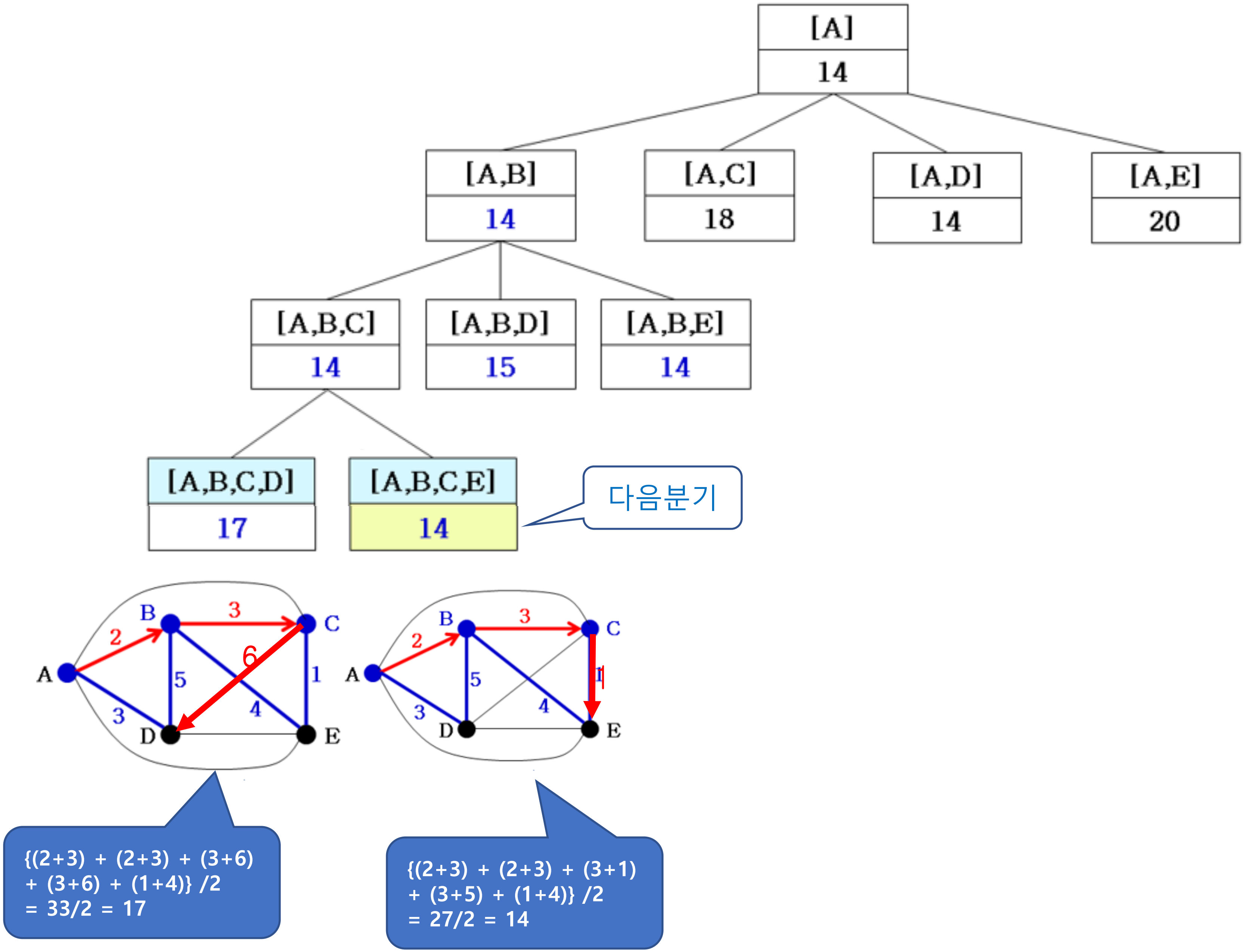
- 분지한정법
  - 한정값이 있음



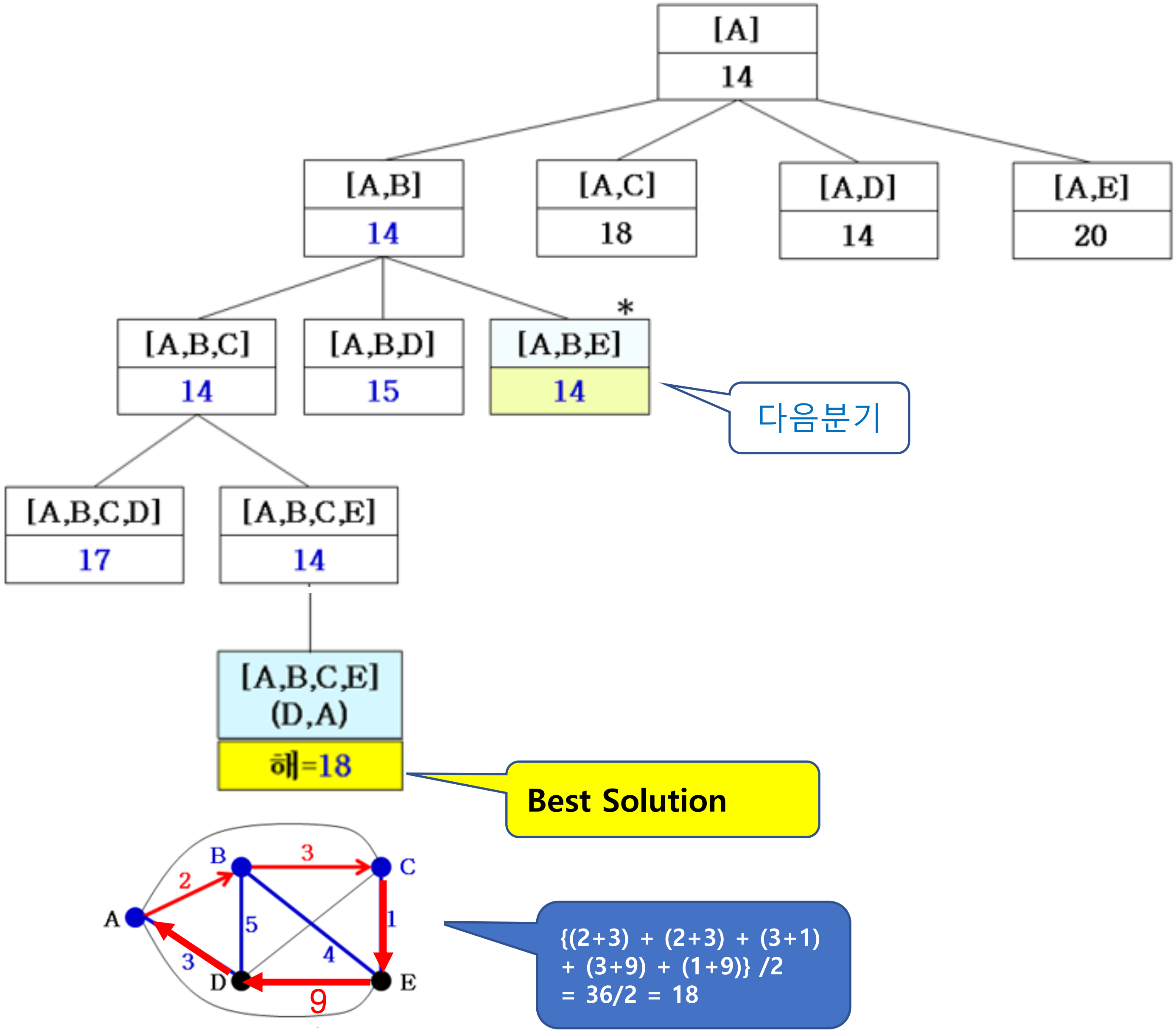


- 상태 [A,B,C], [A,B,D], [A,B,E]의 한정 값은 다음과 같이 각각 계산된다.



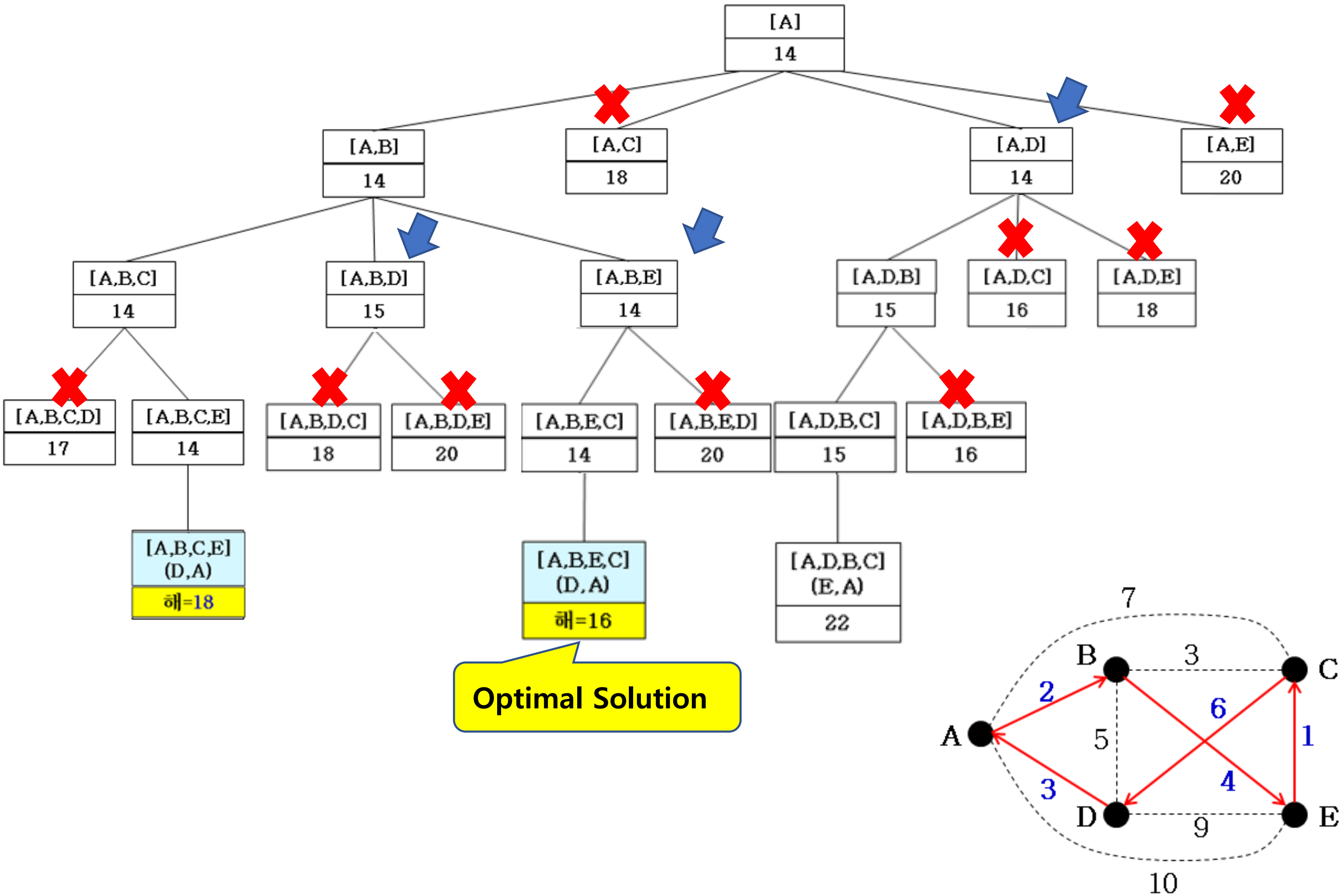


[A]
14
[A,B]
14
[A,C]
18
[A,D]
14
[A,E]
20
[A,B,C]
14
[A,B,D]
15
[A,B,E]
14
[A,B,C,D]
17
[A,B,C,E]
14
다음분기
{(2+3) + (2+3) + (3+6)
+ (3+6) + (1+4)} /2
= 33/2 = 17
{(2+3) + (2+3) + (3+1)
+ (3+5) + (1+4)} /2
= 27/2 = 14



[A]
14
[A,B]
14
[A,C]
18
[A,D]
14
[A,E]
20
[A,B,C]
14
[A,B,D]
15
*
[A,B,E]
14
다음분기
[A,B,C,D]
17
[A,B,C,E]
14
[A,B,C,E]
(D,A)
해=18
Best Solution
A
B
C
D
E
2
3
5
1
3
4
9
{(2+3) + (2+3) + (3+1)
+ (3+9) + (1+9)} /2
= 36/2 = 18

이후의 전체 알고리즘 과정



Assignment Problem

- 최소비용할당문제
  - 각 기계는 한 작업만 할 수 있음
  - 어떻게 할당해야 최소의 비용으로 모든 작업을 마칠 수 있을까?

기계 \ 작업	1	2	3	4
A	10	33	41	20
B	24	17	50	60
C	39	32	62	29
D	22	27	39	37

- 경우의 수를 따져보자
  - A 기계에 작업을 할당하는 방법➔4가지
  - 그 후에, B 기계에 작업을 할당하는 방법➔3가지
  - 그 후에, C 기계에 작업을 할당하는 방법➔2가지
  - 그 후에, D 기계에 작업을 할당하는 방법➔1가지
  - 총 4! = 24가지의 Feasible solution이 존재
- 최적해(Optimal Solution)은?

알고리즘

- Step1: Branch
  - 두 가지로 분리
    - A 기계에 작업 1을 할당하는 방법과
    - A 기계에 작업 1을 할당하지 않는 방법
- Step2: Bound
  - lower bound와 best solution을 구함
    - 기계입장에서 최소값의 조합/작업입장에서 최소값의 조합
      - 둘 중 큰 수를 lower bound으로
    - 만약 feasible 값이면 best solution으로
- Step3: Cut
  - lower bound보다 큰 값의 서브 카테고리를 버림

최소비용문제 : solution

$(A,1)$

M \ T	1	2	3	4	Min.
A	10	33	41	20	10
B	24	**17**	**50**	**60**	17
C	39	**32**	**62**	**29**	29
D	22	**27**	**39**	**37**	27
Min.	10	17	39	29	83 / 95

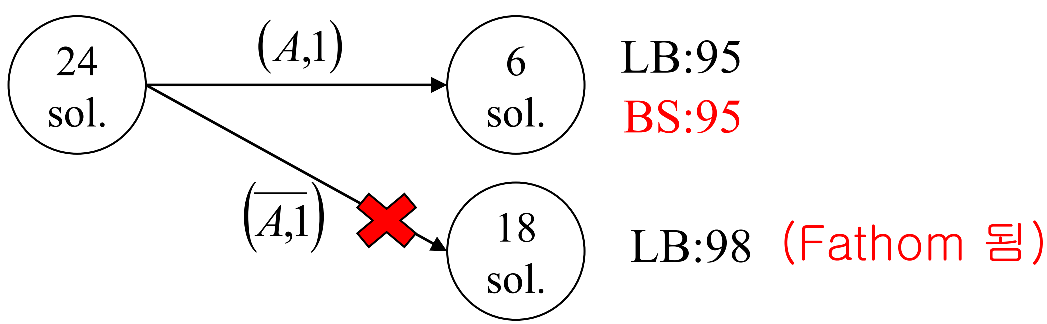
(A,1)에 대한 lower bound

Feasible함. 따라서best solution

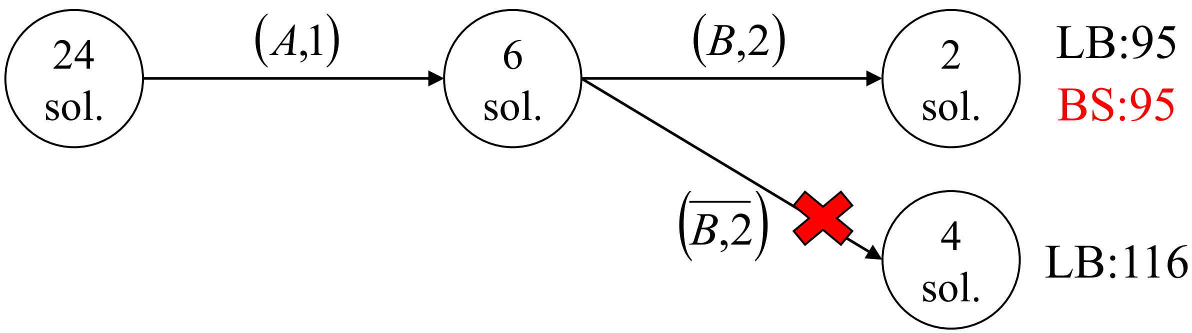
$(\overline{A},1)$

M \ T	1	2	3	4	Min. Sum
A	10	33	41	20	20
B	24	17	50	60	17
C	39	32	62	29	29
D	22	27	39	37	22
Min. Sum	22	17	39	20	88 / 98

$(\overline{A},1)$에 대한 lower bound



최소비용문제 : solution



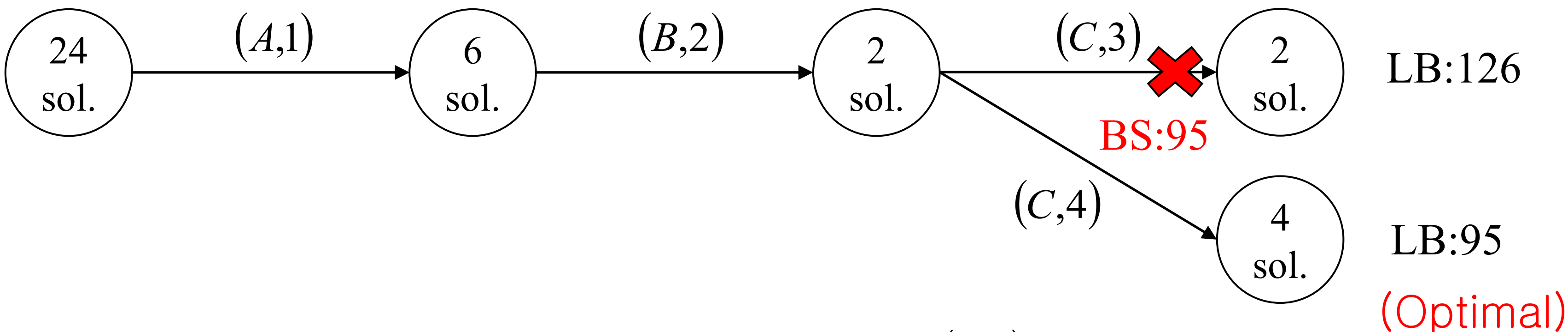
$(B,2)$

M \ T	1	2	3	4	Min. Sum
A	10	33	41	20	10
B	24	**17**	**50**	**60**	17
C	39	**32**	**62**	**29**	29
D	22	**27**	**39**	**37**	27
Min. Sum	10	17	39	29	83 **95**

$(\overline{B,2})$

M \ T	1	2	3	4	Min. Sum
A	10	33	41	20	10
B	24	17	50	60	50
C	39	32	62	29	29
D	22	27	39	37	27
Min. Sum	10	27	39	29	116 105

최소비용문제 : solution



$(C,3)$

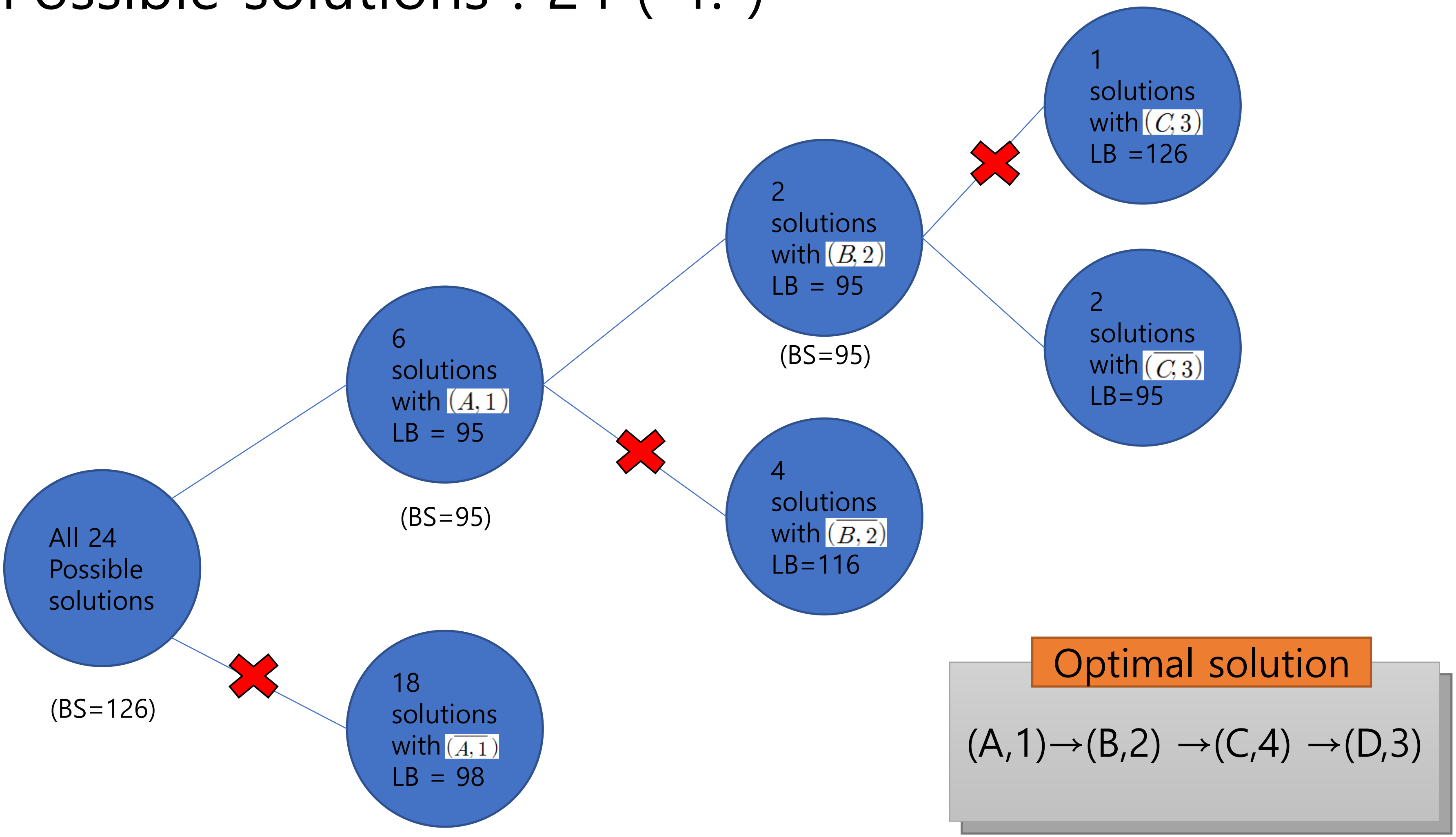
T / M	1	2	3	4	Min. Sum
A	10	33	41	20	10
B	24	**17**	**50**	**60**	17
C	39	**32**	**62**	**29**	62
D	22	**27**	**39**	**37**	37
Min. Sum	10	17	62	37	**126**

$(\overline{C,3}) \Rightarrow (C,4)$

T / M	1	2	3	4	Min. Sum
A	10	33	41	20	10
B	24	17	50	60	17
C	39	32	62	29	29
D	22	27	39	37	39
Min. Sum	10	17	39	29	95

최소비용문제 : 요약

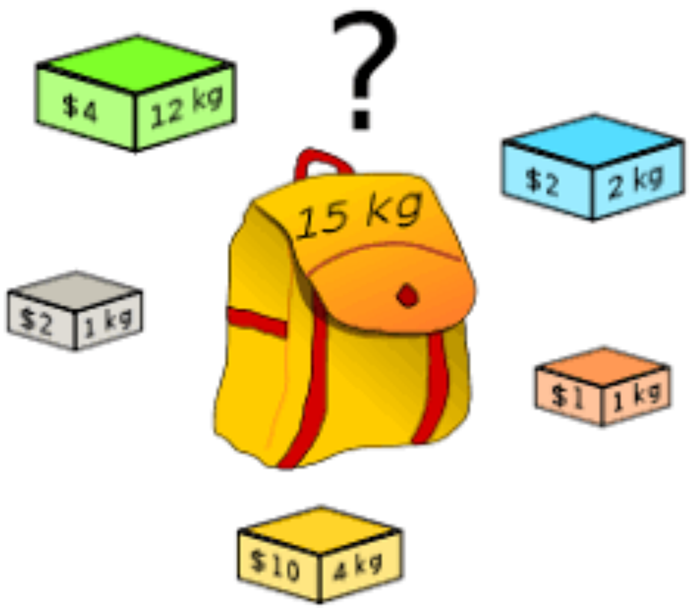
- Possible solutions : 24 (4!)



0-1 Knapsack 문제

n개의 물건이 있고, 각각의 물건을 배낭에 넣을 것인가 말 것인가를 결정하는 문제

- 각 물건의 가치를 p_i, 무게를 w_i
- 배낭 안에 넣을 수 있는 총 무게를 W
- 배낭에 담긴 가치의 합을 최대화



0-1 Knapsack : Branch & Bound Approach

Branch

- 각각의 물건을 배낭에 넣을 것인가 말 것인가에 대하여 분지
- 가급적 물건의 무게당 가치가 큰 것부터 가지치기를 하여 검토
  - 왜냐하면 그 물건의 포함 여부에 따라 해의 차이가 커서 fathom의 확률이 높아짐

Upper Bound (최소화 문제와 달리 최대화 문제에서는 상한치를 결정)

- Feasibility check
  - 담긴 물건의 무게의 합이 W를 초과하면 탈락
- 각 물건을 무게 당 가치가 큰 것부터 담되, 마지막에는 건을 쪼개서 담을 수 있다고 가정
  - 즉, 물건의 개수를 소수점까지 가능하다고 하면 이것이 Upper bound

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

배낭의 용량 : 16

- Upper Bound의 계산 예
  - 아무 것도 담기지 않았을 때
    - p_i/w_i 값이이 큰 것부터 1개씩 담아 나가다 용량을 초과할 때는 그 물건 무게를 소수점으로 계산
    - 물건 1, 2를 담으면 용량이 16-(2+5)=9가 남고 물건 3을 9/10 만큼 담는다고 하면 40+30+ 50*(9/10) = 105
  - 물건 1이 담기지 않을 때
    - 물건 2, 3를 담으면 용량이 16-(5+10)=1이 남고 물건 4을 1/5 만큼 담는다고 하면 30+50+ 10*(1/5) = 82

0-1 Knapsack : Branch & Bound 풀이과정

- Upper Bound의 계산

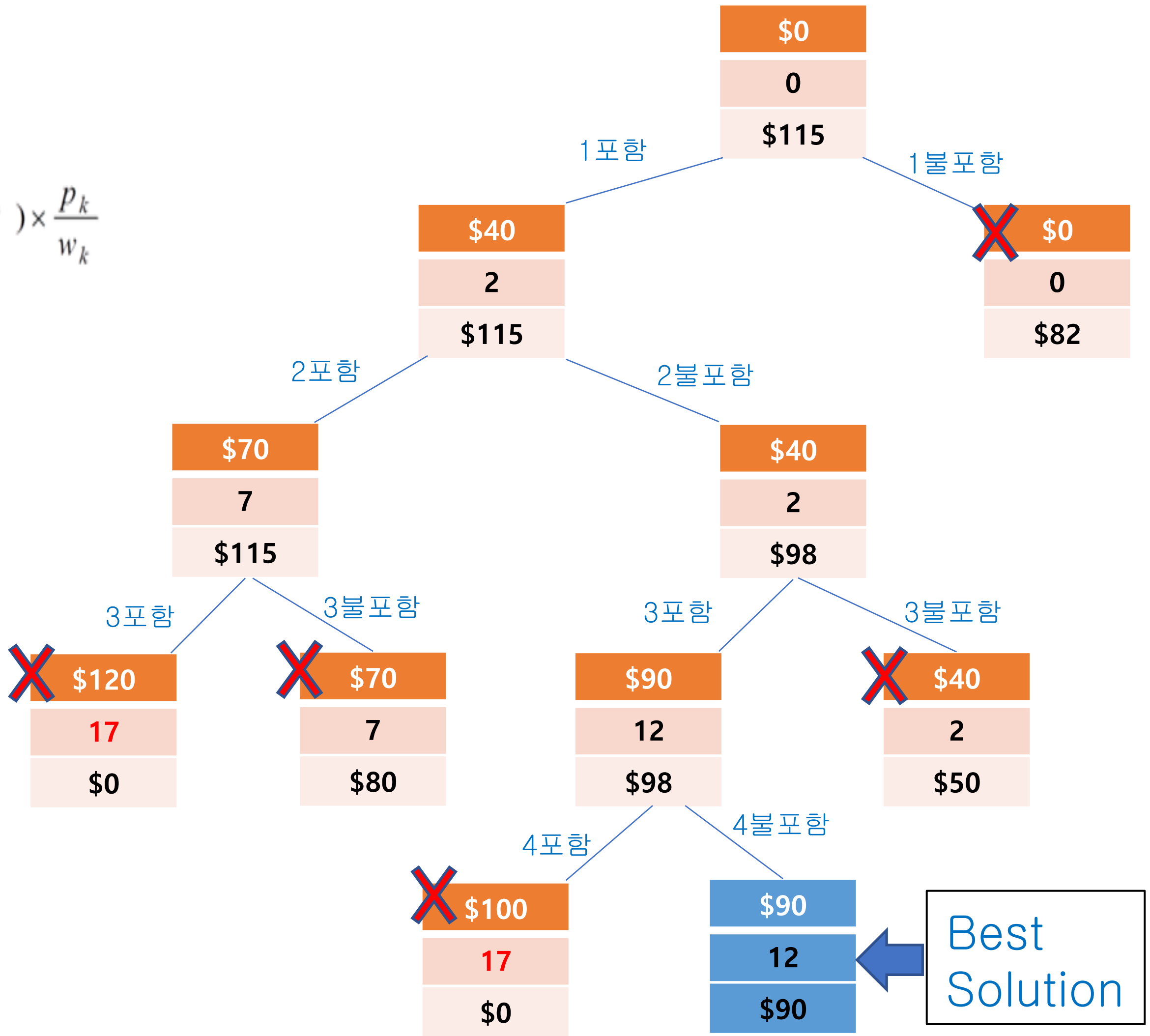
$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

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

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

총가치
담긴 무게
Upper Bound

노드 표시



0-1 Knapsack : Genetic Algorithm

염색체 구성

- 4비트의 0,1 (물건이 포함되면 1, 포함되니 않으면 0 (예 : 1010)
- Population은 5로 하며 최세대는 랜덤하게 구성
- 단, 총 무게가 16을 초과하면 실격 시킴

적합도 함수

- 배낭에 담긴 가치의 합계

교배 및 자손 생성

- 부모의 선택은 룰렛을 이용하여 랜덤하게 두개를 선택
  - 적합도에 비례하여 선택확률 높아짐
- 두 마디씩 끊어 교차 (예: 1111 + 1010 → 1110, 1011)
  - 30%의 확률로 임의의 위치에 돌연변이 발생(반전)
- 가장 우수한 적합도를 가진 염색체는 자손으로 남김

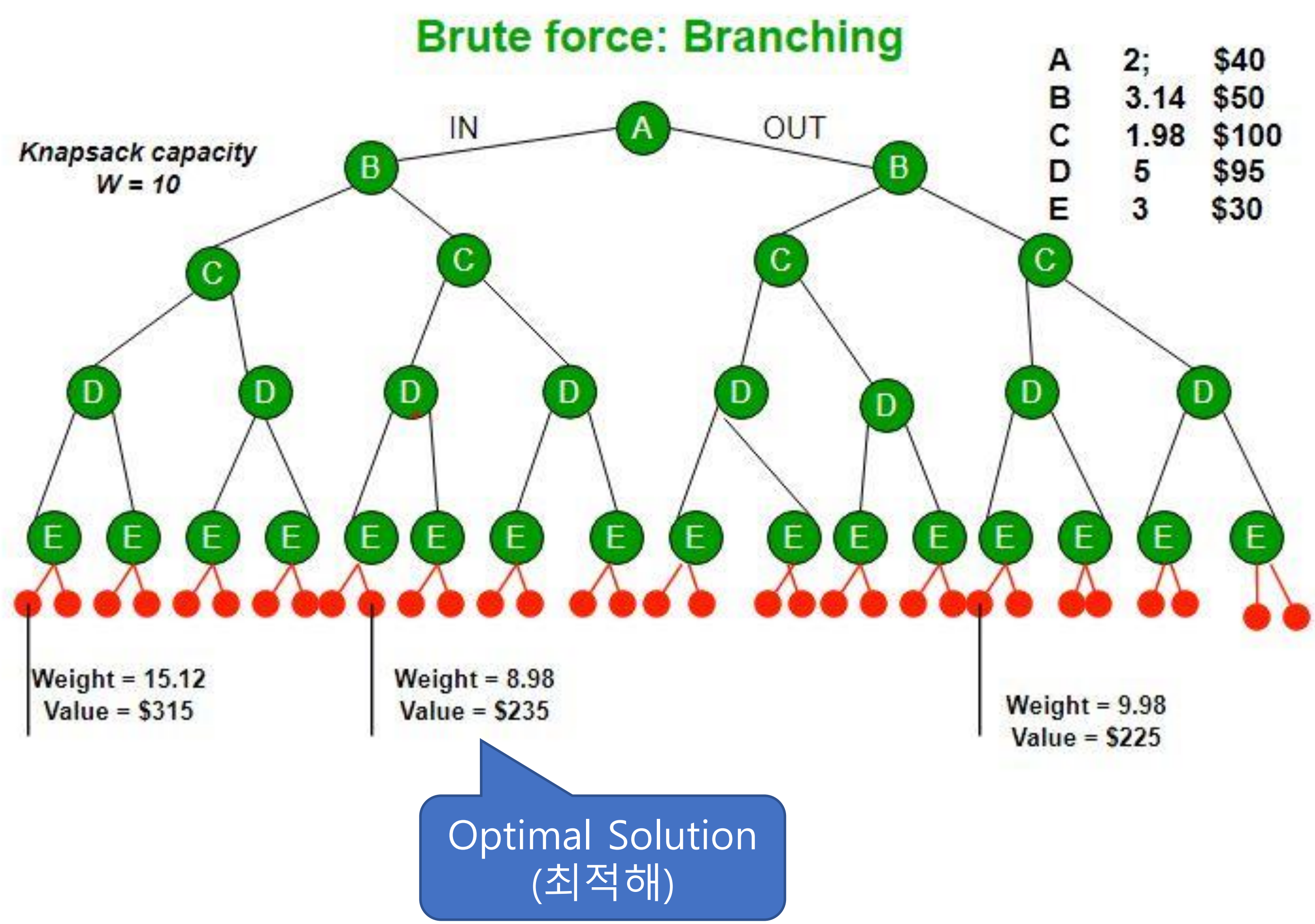
실행 결과

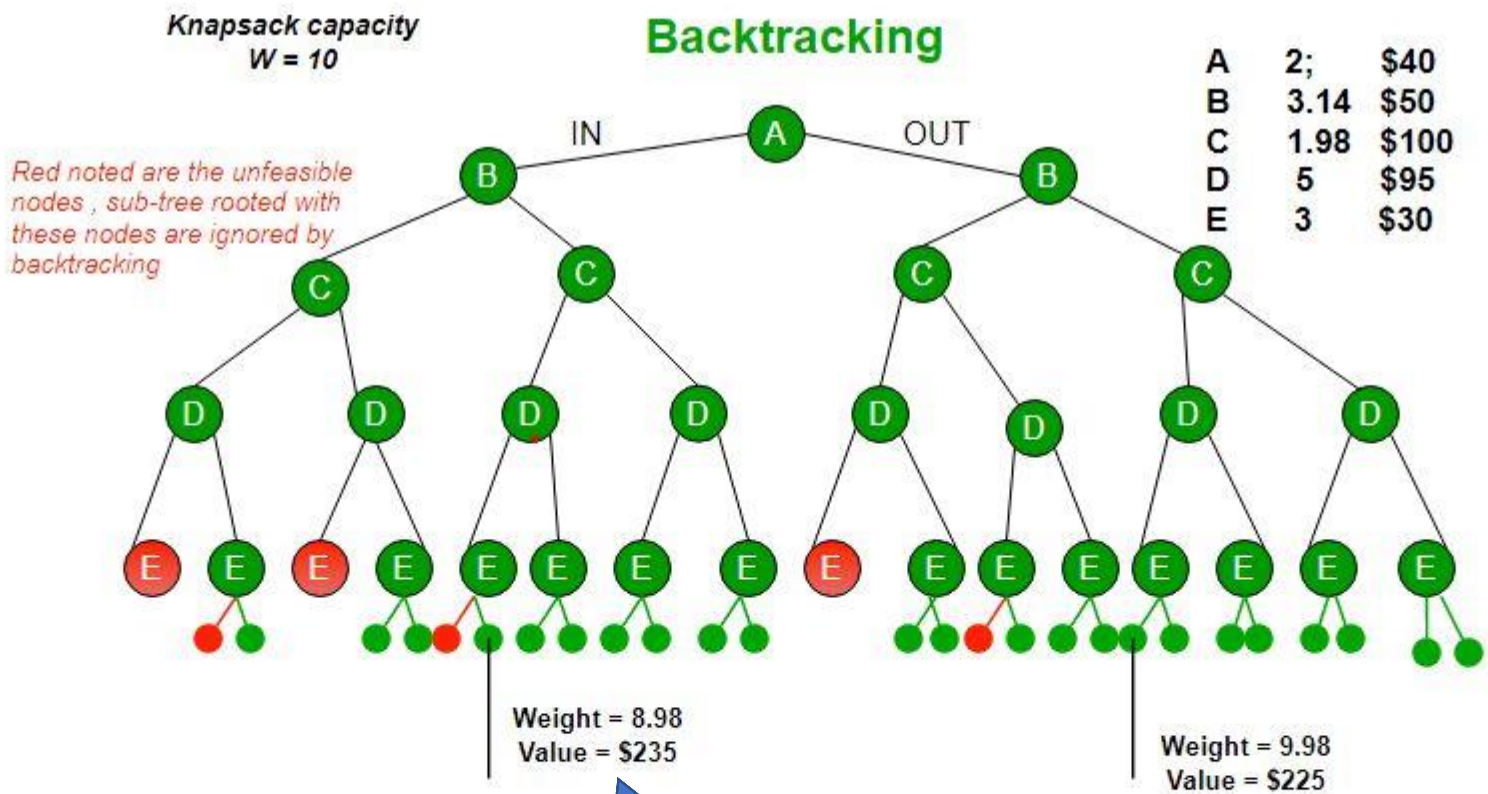
```
<<<<0세대>>>>
염색체 [0] 1001,  Weight =    7, Fitness =   50
염색체 [1] 1111   Weight =   22 (실격)
염색체 [1] 0010,  Weight =   10, Fitness =   50
염색체 [2] 1000,  Weight =    2, Fitness =   40
염색체 [3] 1000,  Weight =    2, Fitness =   40
염색체 [4] 1110   Weight =   17 (실격)
염색체 [4] 0010,  Weight =   10, Fitness =   50
--------------------------------------------------
적합도 합계 =  230
최적염색체 [0] 1001,  Fitness =  50
--------------------------------------------------

<<<<1세대>>>>
     보존 염색체 : 1001,  Fitness =  50
부모염색체[0] = 4번 : 0010
부모염색체[1] = 0번 : 1001
    자손염색체[1] : 0001,  Weight =    5, Fitness =   10
    자손염색체[2] : 1010(2)
   --->변이결과[2] : 1000,  Weight =    2, Fitness =   40
부모염색체[0] = 3번 : 1000
부모염색체[1] = 2번 : 1000
    자손염색체[3] : 1000,  Weight =    2, Fitness =   40
    자손염색체[4] : 1000(1)
   --->변이결과[4] : 1100,  Weight =    7, Fitness =   70
--------------------------------------------------
 적합도 합계 =  210
최적염색체 [4] 1100,  Fitness =  70
--------------------------------------------------
<<<<3세대>>>>
    보존 염색체 : 1001,  Fitness =  70
부모염색체[0] = 4번 : 1100
부모염색체[1] = 0번 : 1001
   자손염색체[1] : 1101,  Weight =   12, Fitness =   80
   자손염색체[2] : 1000,  Weight =    2, Fitness =   40
부모염색체[0] = 2번 : 1000
부모염색체[1] = 4번 : 1100
   자손염색체[3] : 1000,  Weight =    2, Fitness =   40
   자손염색체[4] : 1100(0)
  --->변이결과[4] : 0100,  Weight =    5, Fitness =   30
--------------------------------------------------
 적합도 합계 =  260
최적염색체 [1] 1101,  Fitness =  80
--------------------------------------------------

<<<<11세대>>>>
    보존 염색체 : 1001,  Fitness =  80
부모염색체[0] = 3번 : 1001
부모염색체[1] = 0번 : 1001
   자손염색체[1] : 1001,  Weight =    7, Fitness =   50
   자손염색체[2] : 1001,  Weight =    7, Fitness =   50
부모염색체[0] = 2번 : 1001
부모염색체[1] = 0번 : 1001
   자손염색체[3] : 1001,  Weight =    7, Fitness =   50
   자손염색체[4] : 1001,  Weight =    7, Fitness =   50
--------------------------------------------------
 적합도 합계 =  280
최적염색체 [0] 1001,  Fitness =  80
--------------------------------------------------

<<<<12세대>>>>
    보존 염색체 : 1001,  Fitness =  80
부모염색체[0] = 3번 : 1001
부모염색체[1] = 1번 : 1001
   자손염색체[1] : 1001,  Weight =    7, Fitness =   50
   자손염색체[2] : 1001,  Weight =    7, Fitness =   50
부모염색체[0] = 3번 : 1001
부모염색체[1] = 0번 : 1001
   자손염색체[3] : 1001,  Weight =    7, Fitness =   50
   자손염색체[4] : 1001,  Weight =    7, Fitness =   50
--------------------------------------------------
 적합도 합계 =  280
최적염색체 [0] 1001,  Fitness =  80
--------------------------------------------------

<<<<13세대>>>>
    보존 염색체 : 1001,  Fitness =  80
부모염색체[0] = 1번 : 1001
부모염색체[1] = -1번 : 0000
   자손염색체[1] : 1000,  Weight =    2, Fitness =   40
   자손염색체[2] : 0000,  Weight =    0, Fitness =    0
부모염색체[0] = 3번 : 1001
부모염색체[1] = 2번 : 0000
   자손염색체[3] : 1000(2)
  --->변이결과[3] : 1010,  Weight =   12, Fitness =   90
   자손염색체[4] : 0010,  Weight =   10, Fitness =   50
--------------------------------------------------
 적합도 합계 =  260
최적염색체 [3] 1010,  Fitness =  90
--------------------------------------------------
```

백트래킹 vs. 분기한정

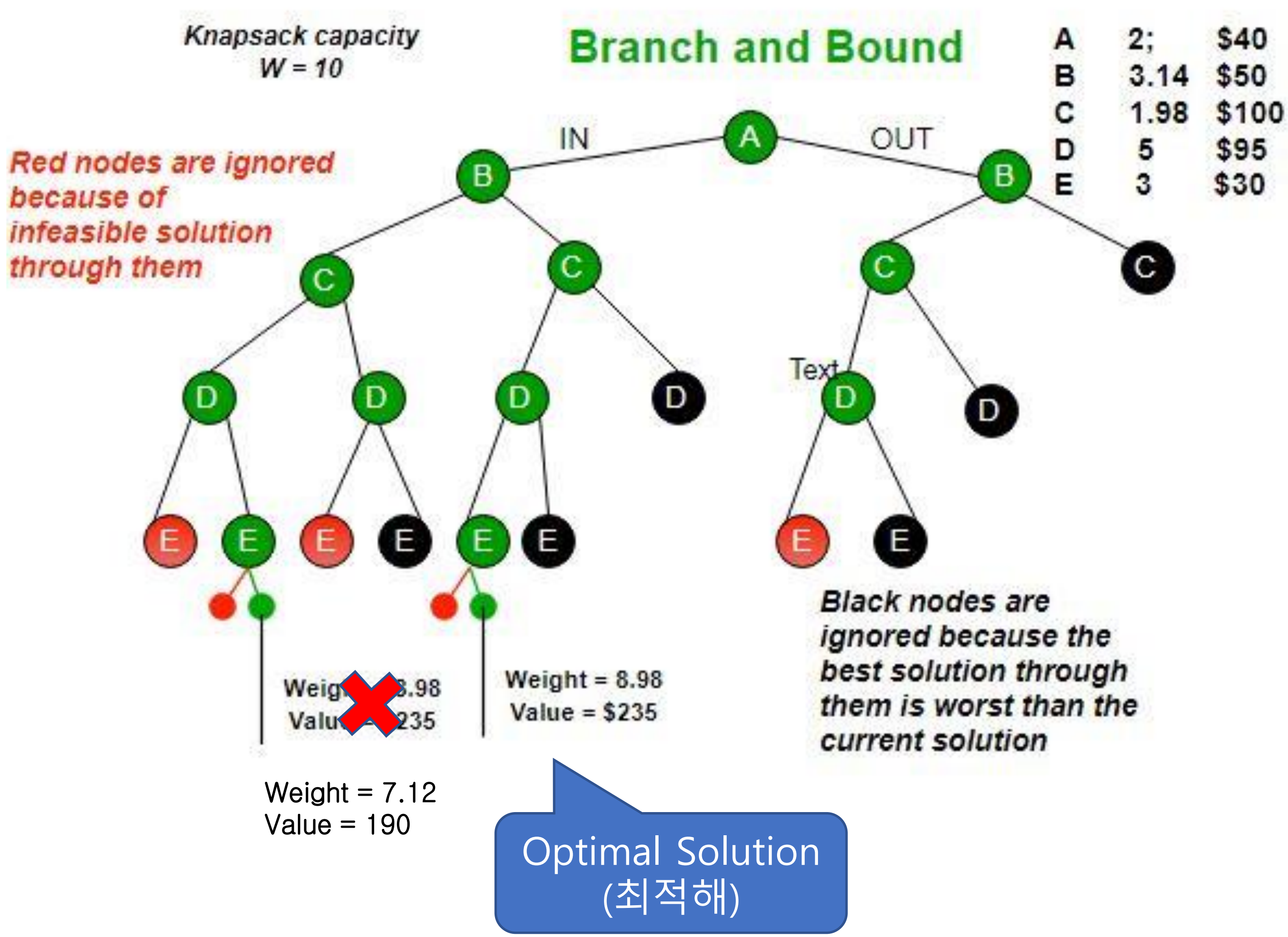
- 백트래킹 알고리즘이 방문한 상태 공간 트리의 노드 수는 총 51개
- 분기 한정 알고리즘은 22개
- 이처럼 최적화 문제의 해를 탐색하는 데는 분기 한정 기법이 백트래킹 기법보다 훨씬 우수한 성능을 보임
- 분기 한정 알고리즘은 한정값을 사용하여 최적해가 없다고 판단되는 부분은 탐색을 하지 않고 최선 우선 탐색을 하기 때문임

각 알고리즘의 비교(0-1 Knap Sack)





Optimal Solution
(최적해)


Knapsack capacity
W = 10
Branch and Bound
A 2; $40
B 3.14 $50
C 1.98 $100
D 5 $95
E 3 $30
IN
OUT
Red nodes are ignored because of infeasible solution through them
Text
Weight = 8.98
Value = $235
Black nodes are ignored because the best solution through them is worst than the current solution
Weight = 7.12
Value = 190
Optimal Solution
(최적해)