제 9 장 해 탐색 알고리즘

창원대학교

박동규

9.1 백트래킹 기법

- 백트래킹 (Backtracking) 기법은 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법이다.
- 백트래킹 기법은 최적화 (optimization) 문제와 결정 (decision) 문제를 해결할 수 있다.
- 결정 문제: 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'가 답하는 문제
  - 미로 찾기
  - 해밀토니안 사이클 (Hamiltonian Cycle) 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 나타낸다.
- tour는 점의 순서 (sequence)
- bestSolution의 tour의 거리는 'bestSolution의 거리'로 표현
- tour = [시작점] // tour는 점의 순서 (sequence)
- bestSolution = (tour, ∞) // tour는 시작점만 가지므로 그 거리는 가장 큰 상수로 초기화시킨다.

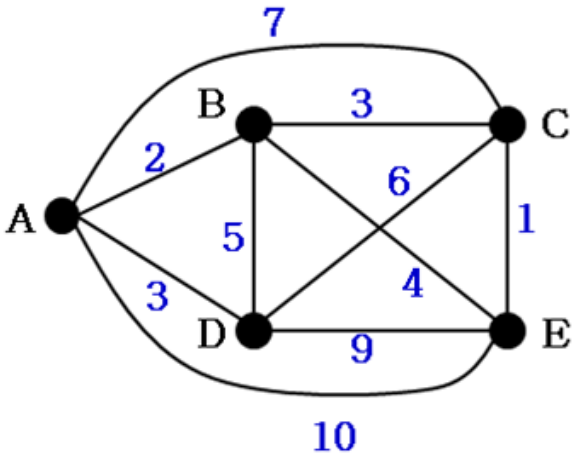
BacktrackTSP(tour)

```
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)
          }
    }
```

- Line 1~3: 현재 tour가 완전한 해이면서 현재까지 찾은 가장 우수한 해인 bestSolution의 거리보다 짧으면 현재 tour로 bestSolution이 갱신 bestSolution의 거리보다 같거나 긴 경우에는 갱신 없이 이전 호출했었던 곳으로 돌아감
- Line 4~8: tour가 아직 완전한 해가 아닐 때 수행
- Line 5~8의 for-루프: 현재 tour에서 확장 가능한 각 점에 대해서 루프의 내부가 수행
  - 확장 가능한 점이란 현재 tour에 없는 점으로서 현재 tour의 가장 마지막 점과 선분으로 연결된 점을 말함
- Line 6: 현재 tour를 확장 (즉, 확장 가능한 점을 기존 tour에 추가)하여 newTour를 얻음

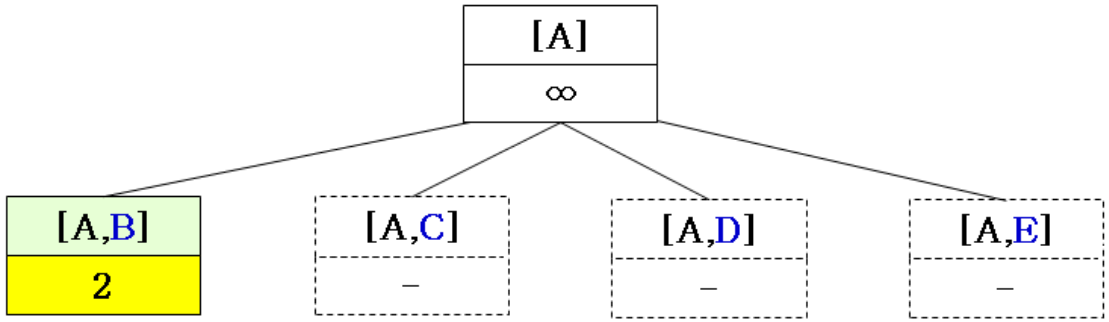
- Line 7~8: 확장된 newTour의 거리가 bestSolution의 거리보다 짧으면, newTour를 가지고 알고리즘을 재귀 호출한다.
- 만일 newTour의 거리가 bestSolution의 거리보다 같거나 길면, newTour를 확장하여 보아도 현재까지의 bestSolution의 거리보다 짧은 tour를 얻을 수 없기 때문에 가지치기 (pruning)함
- 가지를 친 경우에는 다음의 확장 가능한 점에 대해서 루프가 수행

- 다음의 그래프에 대한 BacktrackTSP 알고리즘의 수행과정 (점 A=시작점)



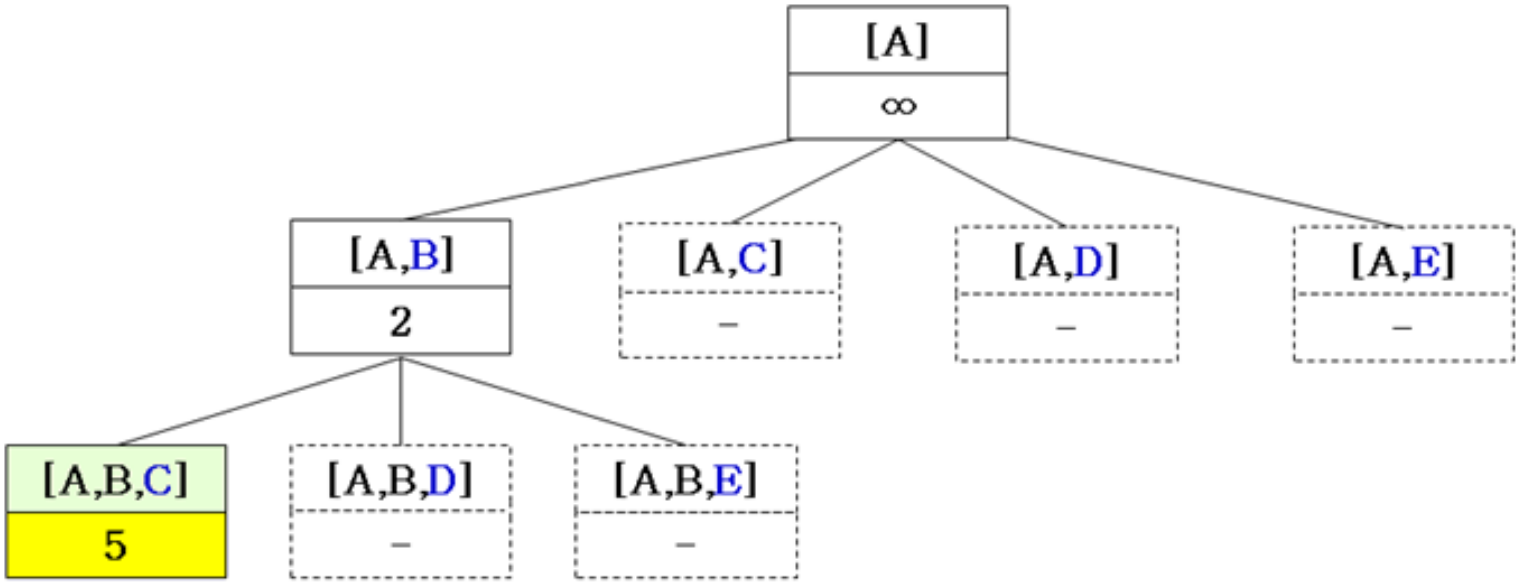
- 시작점이 A이므로, tour=[A]이고, bestSolution=([A],∞)이다.
- BacktrackTSP(tour)를 호출하여 해 탐색 시작

- Line 1: [A]가 완전한 해가 아니므로 line 5의 for-루프 수행
- Line 5의 for-루프에서 현재 tour [A]를 확장할 수 있는 점을 살펴보면, 점 B, C, D, E가 있다. 따라서 각 점에 대해 루프가 수행됨.
- 먼저 점 B에 대해서 line 6~8이 수행된다고 가정
- Line 6: newTour=[A,B]가 되고, newTour의 거리는 2. 왜냐하면 선분 (A,B)의 가중치가 2이므로.



- Line 7~8: newTour의 거리 2가 bestSolution의 거리 ∞보다 짧으므로, BacktrackTSP([A,B])를 재귀 호출. 확장 가능한 점 C, D, E에 대해서는 BacktrackTSP([A,B]) 호출을 다 마친 후에 각각 수행
- BacktrackTSP([A,B])가 호출되면, line 1에서 [A,B]가 완전한 해가 아니므로 line 5의 for-루프가 수행
- Line 5의 for-루프에서 현재 tour [A,B]를 확장할 수 있는 점을 살펴보면, 점 C, D, E가 있다. 따라서 각 점에 대해 루프가 수행.
- 먼저 점 C에 대해서 line 6~8이 수행된다고 가정

- Line 6에서 newTour=[A,B,C]가 되고, newTour의 거리는 5.
- 왜냐하면 선분 (B,C)의 가중치가 3이므로

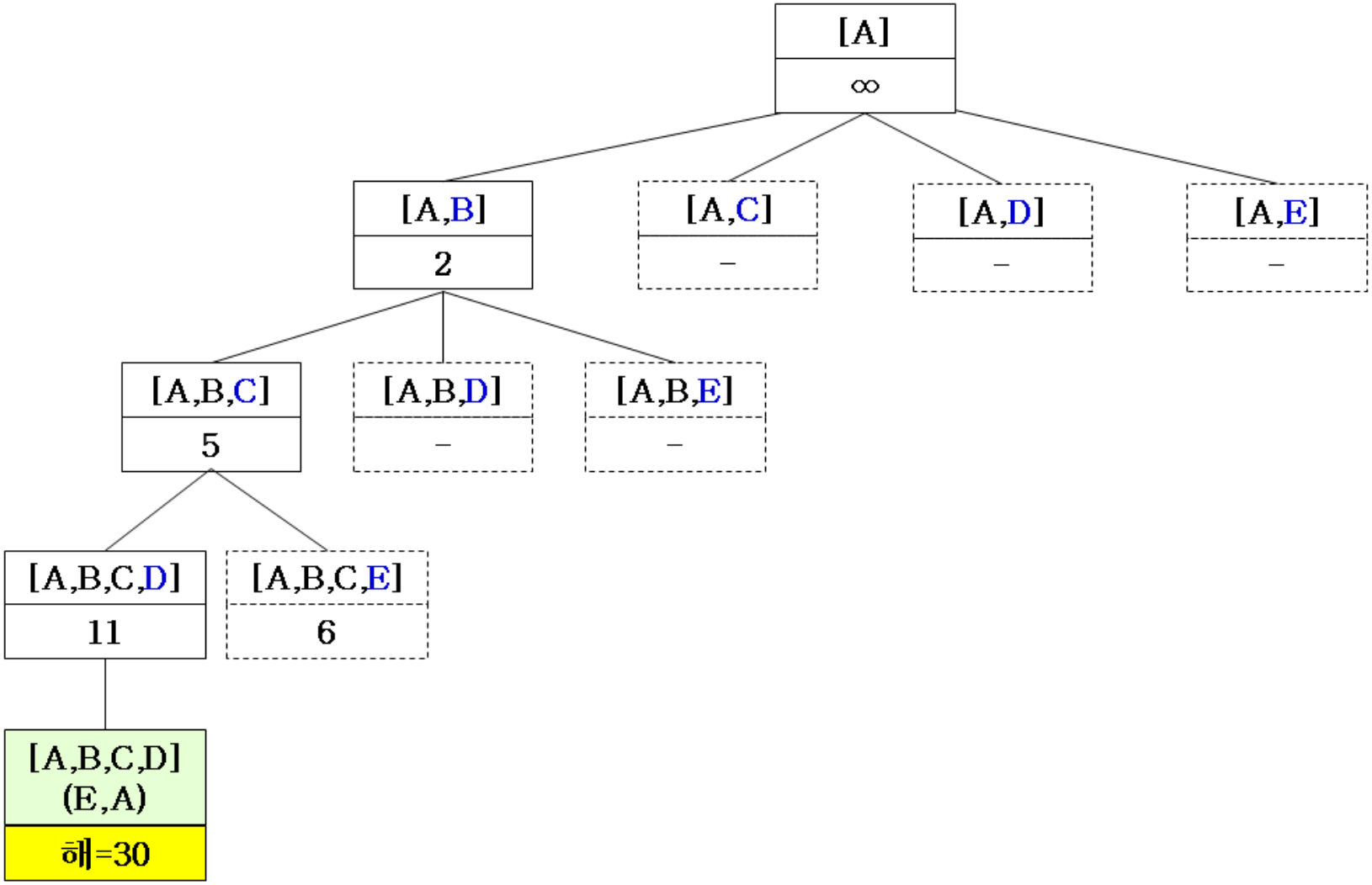


- Line 7~8: newTour의 거리 5가 bestSolution의 거리 ∞보다 짧으므로, BacktrackTSP([A,B,C])를 재귀 호출. 확장 가능한 점 D, E에 대해서는 BacktrackTSP([A,B,C]) 호출을 다 마친 후에 각각 수행

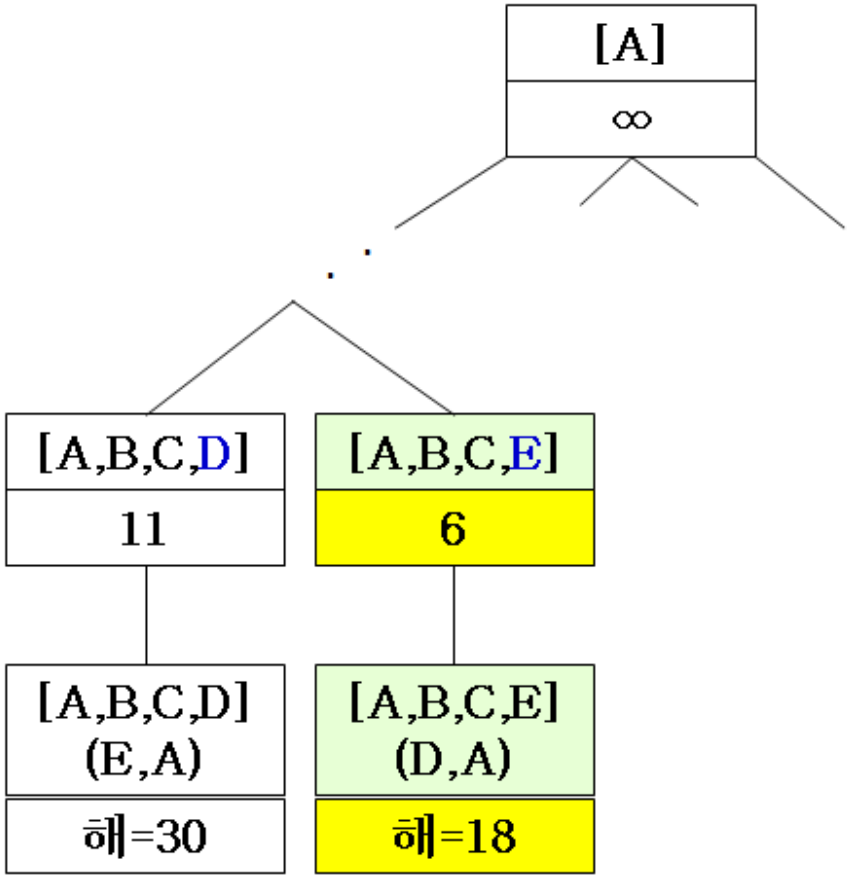
  …

- 이와 같이 계속 탐색이 진행되면 다음과 같이 첫 번째 완전한 해를 찾는다.

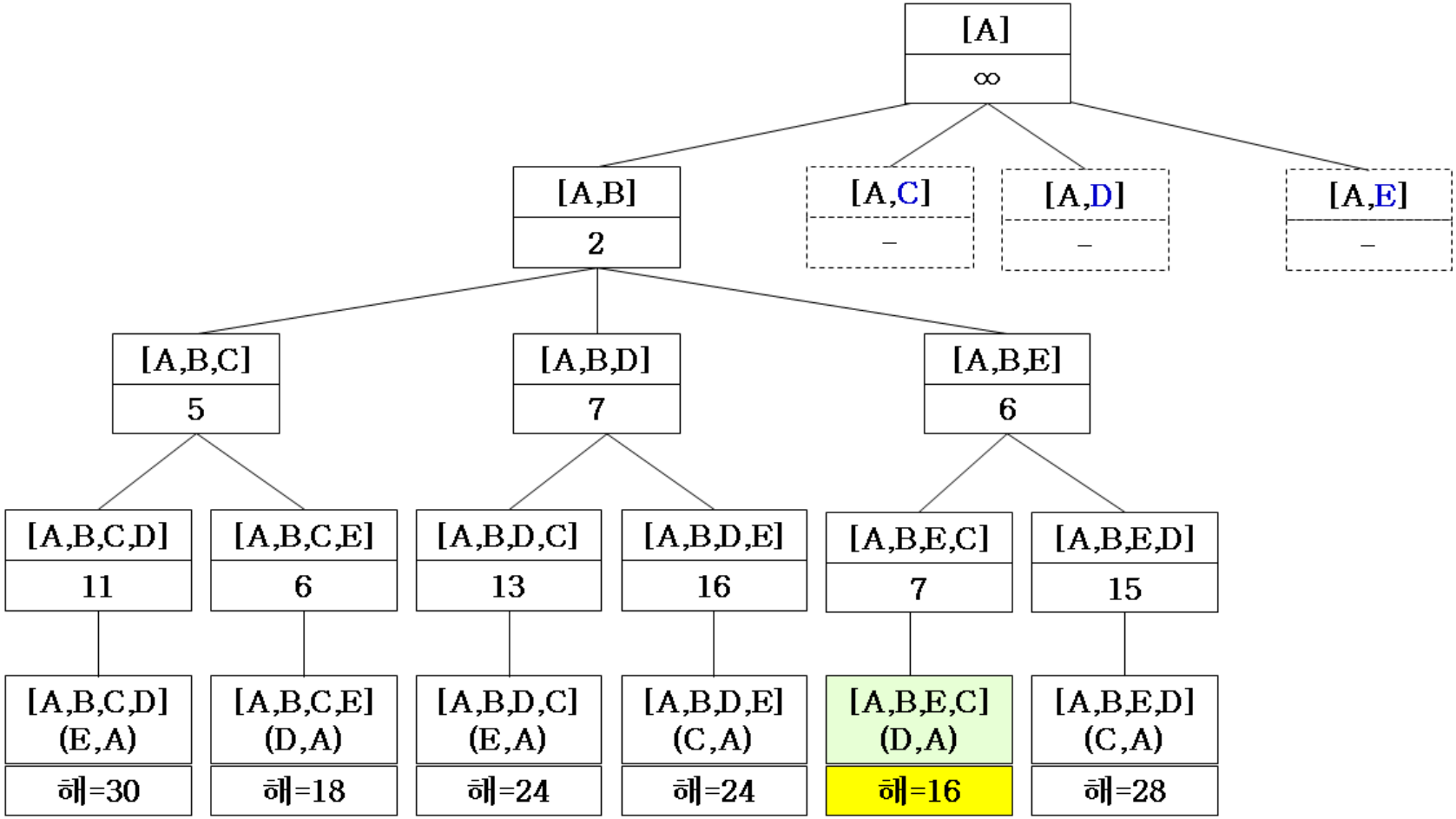
- 이때 bestSolution=([A,B,C,D,E,A], 30)이 된다.


[A]
∞
[A,B]
2
[A,C]
–
[A,D]
–
[A,E]
–
[A,B,C]
5
[A,B,D]
–
[A,B,E]
–
[A,B,C,D]
11
[A,B,C,E]
6
[A,B,C,D]
(E,A)
해=30

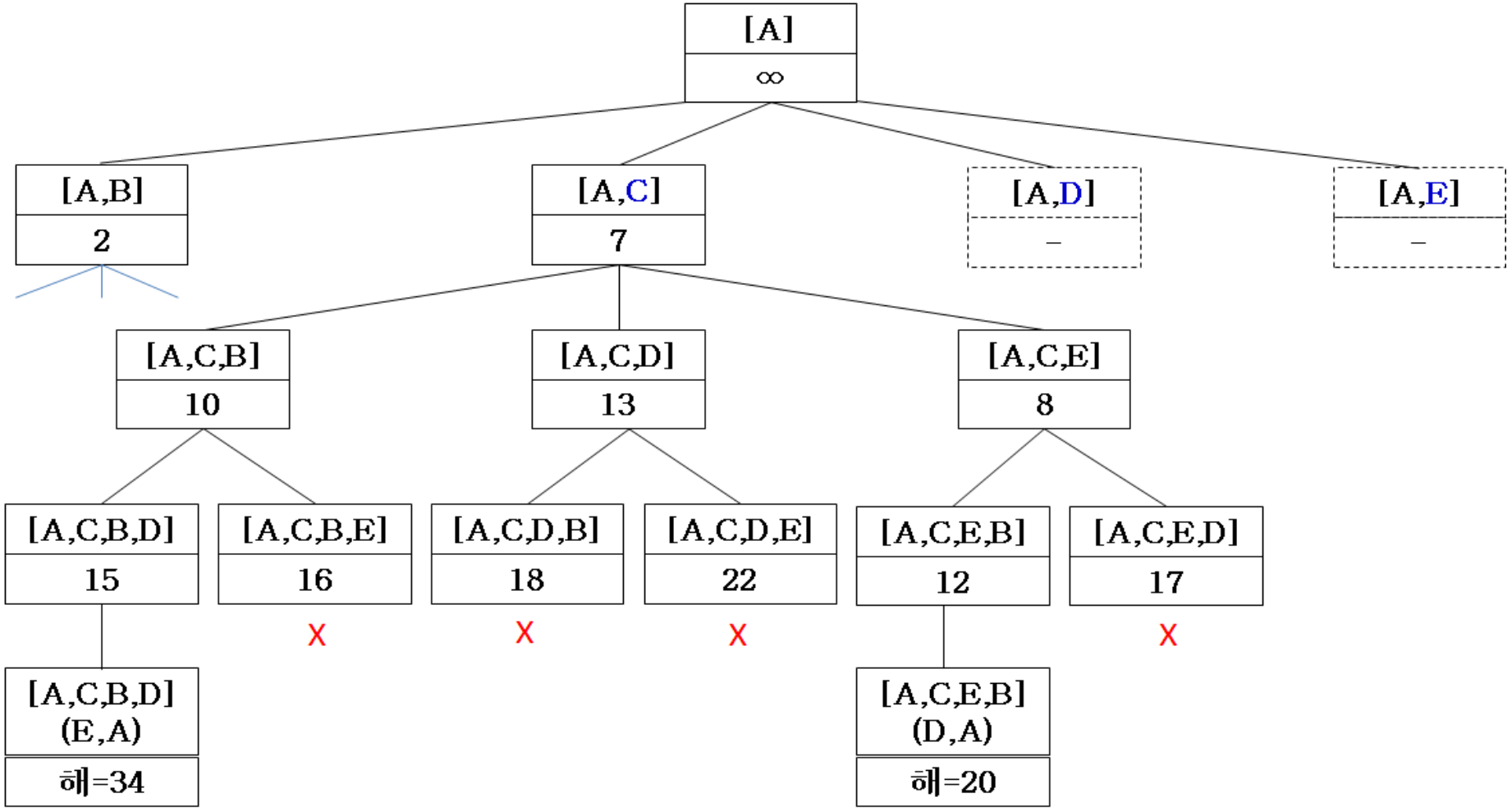
- 첫 번째 완전한 해를 찾은 후에는 다음과 같이 수행되며, 이때 더 짧은 해를 찾으므로 bestSolution=([A,B,C,E,D,A], 18)이 된다.



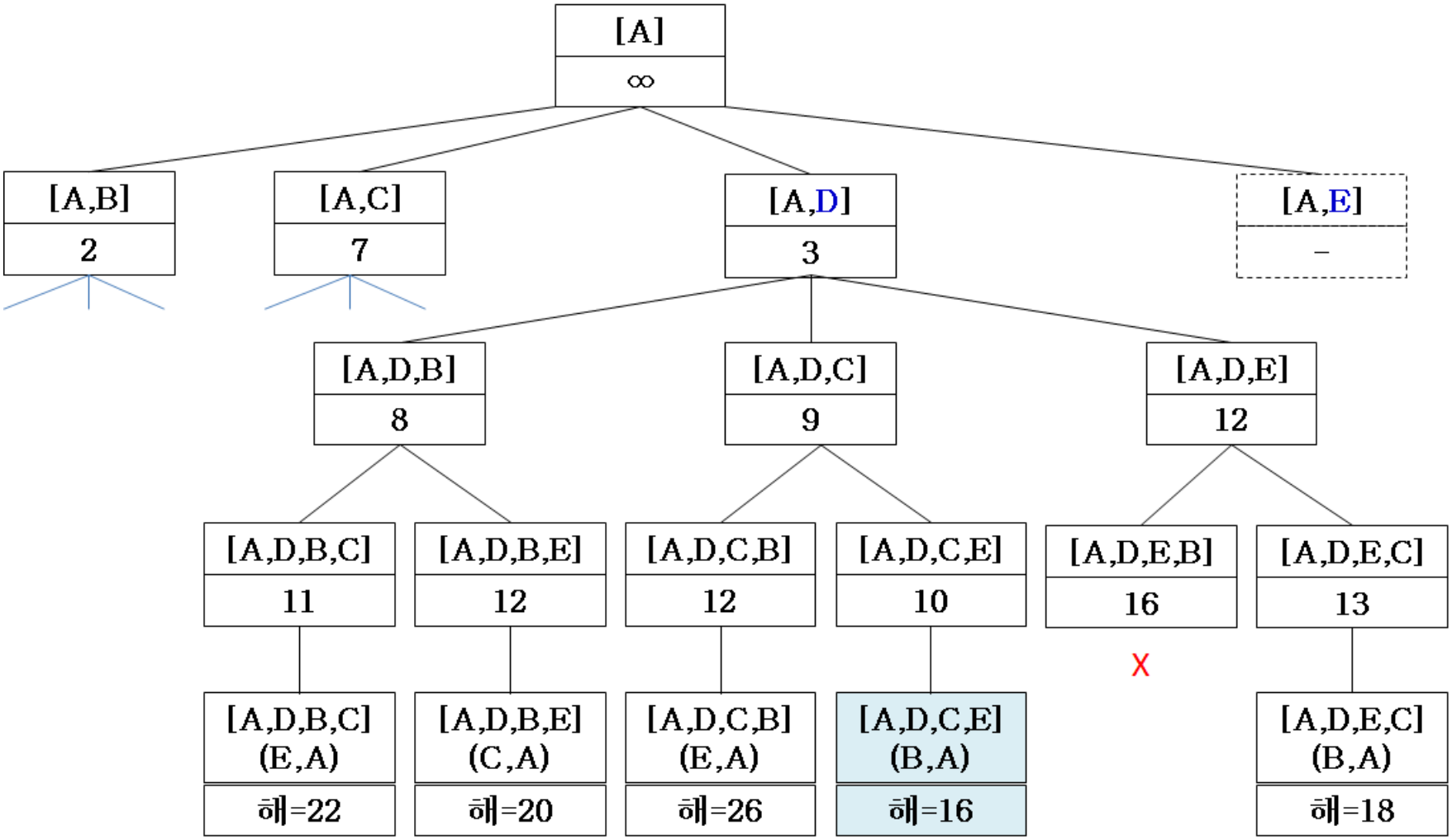
- 다음은 tour=[A,B]에 대해서 모든 수행을 마친 결과이다. bestSolution=([A,B,E,C,D,A], 16)이다.



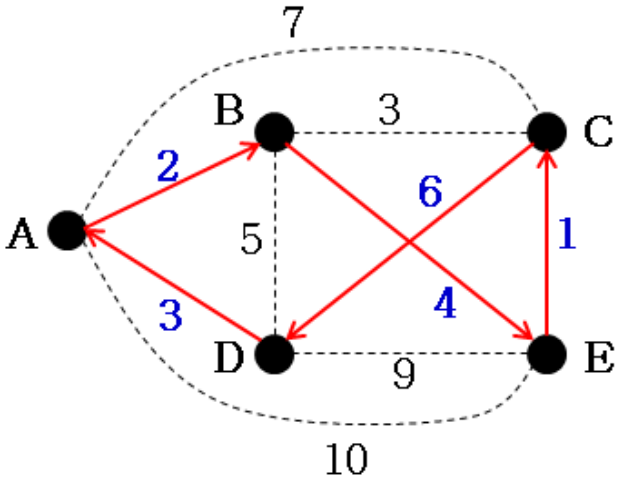
- 다음은 tour=[A,C]에 대해서 모든 수행을 마친 결과를 보여줌
- 그러나 bestSolution보다 더 우수한 해는 탐색되지 않았고, **X**로 표시된 4개의 상태 각각은 bestSolution의 거리보다 짧지 않으므로 가지치기된 것임


[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
[A,C,D,B]
18
[A,C,D,E]
22
[A,C,E,B]
12
[A,C,E,D]
17
X
X
X
X
[A,C,B,D]
(E,A)
해=34
[A,C,E,B]
(D,A)
해=20

- 다음은 tour=[A,D]에 대해서 모든 수행을 마친 결과를 보여줌
- 이때 bestSolution보다 더 우수한 해는 탐색되지 않았으나 같은 거리의 해를 찾는데 이 해는 bestSolution tour의 역순
- **X**로 표시된 1개의 상태는 bestSolution의 거리와 같으므로 가지치기된 것임

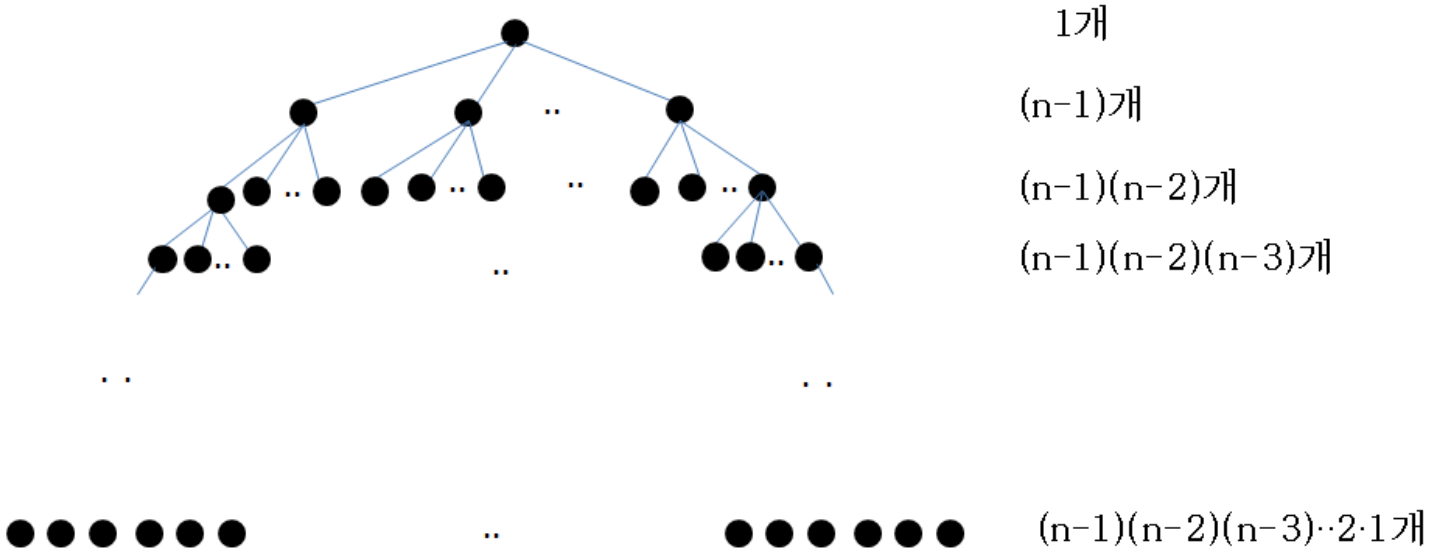

[A]
∞
[A,B]
2
[A,C]
7
[A,D]
3
[A,E]
–
[A,D,B]
8
[A,D,C]
9
[A,D,E]
12
[A,D,B,C]
11
[A,D,B,E]
12
[A,D,C,B]
12
[A,D,C,E]
10
[A,D,E,B]
16
[A,D,E,C]
13
X
[A,D,B,C]
(E,A)
해=22
[A,D,B,E]
(C,A)
해=20
[A,D,C,B]
(E,A)
해=26
[A,D,C,E]
(B,A)
해=16
[A,D,E,C]
(B,A)
해=18

- 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 비례한다.
- n개의 점이 있는 입력 그래프에 대해서 BacktrackTSP 알고리즘이 탐색하는 최대 크기의 상태 공간 트리



- 위의 트리의 이파리 노드 수만 계산해도 (n-1)!
- 문제에 따라서 이진트리 형태의 상태 공간 트리가 형성되기도 하는데 이때에도 최악의 경우에 2^n개의 노드를 모두 탐색해야 하므로 지수 시간이 걸림
- 이는 모든 경우를 다 검사하여 해를 찾는 완결 탐색 (Exhaustive Search)의 시간복잡도와 같음
- 그러나 일반적으로 백트래킹 기법은 '가지치기'를 하므로 완결 탐색보다 훨씬 효율적임

9.2 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

- 분기 한정 기법은 상태 공간 트리의 각 노드 (상태)에 특정한 값 (한정값)을 부여
- 노드의 한정값을 활용하여 가지치기를 함으로서 백트래킹 기법보다 빠르게 해를 찾음
- 분기 한정 기법에서는 가장 우수한 한정값을 가진 노드를 먼저 탐색하는 최선 우선 탐색 (Best First Search)으로 해를 찾음

- **분기 한정 기법의 효율적인 탐색 원리**

  1. 최적해를 찾은 후에, 탐색하여야 할 나머지 노드의 한정값이 최적해의 값과 같거나 나쁘면 더 이상 탐색하지 않는다.

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

  3. 최적해가 있을만한 영역을 먼저 탐색한다.

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'_i로부
                                                터 탐색을 수행
          }
    }
```

- 각 상태에서는 한정값을 계산하는 방법은 문제에 따라 다르다.
- 하나의 상태에 대해 탐색을 마친 후에는 acitveNodes에서 가장 작은 한정값을 가진 상태를 탐색한다. 즉, 최선 우선 탐색을 한다.
- 여기서 activeNodes는 탐색할 상태의 집합이다.
- Line 1~3: 문제의 초기 상태의 한정값을 계산한 후, 초기 상태만을 원소로 갖는 activeNodes로서 탐색이 시작
- bestValue는 현재까지 탐색된 해 중의 가장 작은 값을 가지는데, bestValue를 가장 큰 수로 초기화

- Line 4~15의 while-루프: activeNodes가 공집합이 되면, 즉, 더 이상 탐색할 상태가 없으므로 탐색을 중단
- activeNodes가 공집합이 아니면, line 5에서는 activeNodes에서 한정값이 가장 작은 상태를 선택하여 이를 S_{min}이라고 하자.
- Line 6: S_{min}을 activeNodes에서 제거
- Line 7: S_{min}으로부터 확장 가능한 상태 (자식 노드)를 생성하고 각각의 상태에 대한 한정값을 계산
- Line 8~15의 for-루프: line 7에서 S_{min}으로부터 생성된 각각의 S'_i에 대하여 루프가 수행

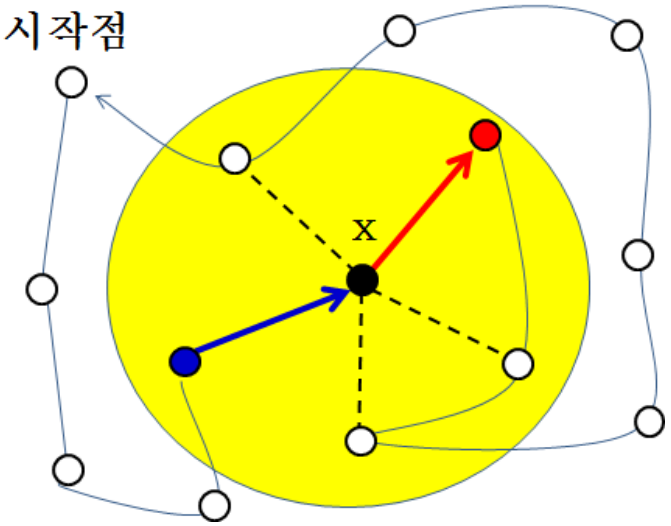
- Line 9~10: S'_i의 한정값이 bestValue보다 같거나 크면 가지치기: S'_i로부터 탐색되지 않도록 한다.
- Line 11~13: 만일 S'_i가 완전한 해이고 동시에 S'_i의 값이 bestValue보다 작으면, 즉, 더 '우수한' 해이면, bestValue를 S'_i의 값으로 갱신하고, S'_i가 bestSolution이 됨
- Line 15: line 9와 11의 if-조건이 모두 '거짓'이면 S'_i를 나중에 탐색하기 위해서 activeNodes에 추가함

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.

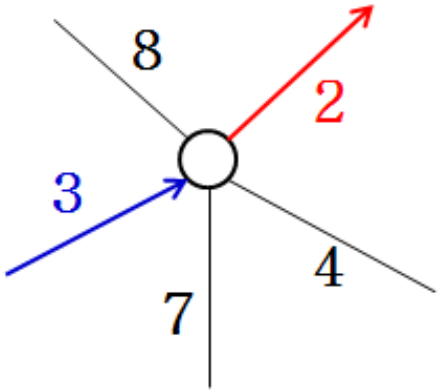
한정값의 계산 방법:

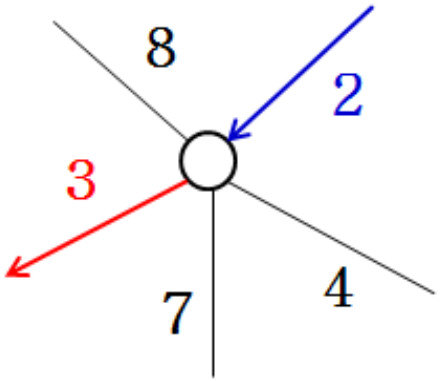
- 점 x로 들어올 때와 나갈 때 사용되는 선분의 가중치를 한정값 계산에 활용
- 점 x에 연결된 선분 중에서 가중치가 가장 작은 두 선분의 가중치의 합의 1/2을 한정값으로 이용



- 이 방법은 최적의 여행자 경로를 ‘근시안적’으로 임의의 점 하나에 대해서만 고려한 것임
- 가중치의 합을 1/2로 곱하는 이유: 한 점에서 나가는 선분은 인접한 (다른) 점으로부터 들어오는 선분과 동일하기 때문
- 단, 소수점 이하의 숫자는 올림을 한다.

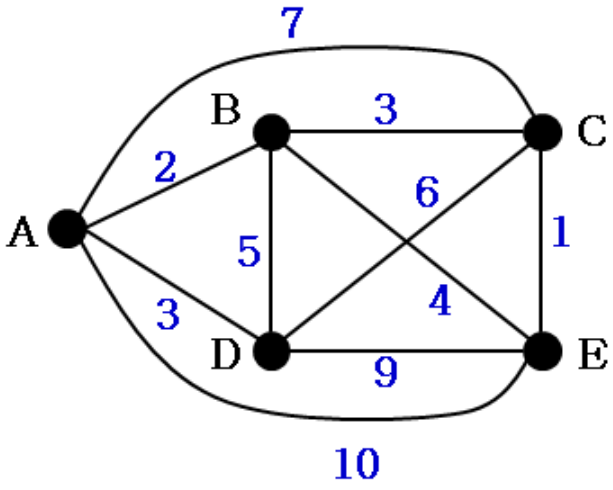
- 아래의 그림은 점에 인접한 선분의 가중치 중에서 2개의 가장 작은 가중치는 3과 2이다.
- 가중치 3인 선분으로 들어와서 가중치 2인 선분으로 나가든지 (왼쪽 그림) 반대로 가중치 2인 선분으로 들어와서 가중치 3인 선분으로 나가든지 (오른쪽 그림), 두 경우 모두 최소의 비용으로 이 점을 방문하는 것이다.





Branch-and-Bound 알고리즘 수행과정

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

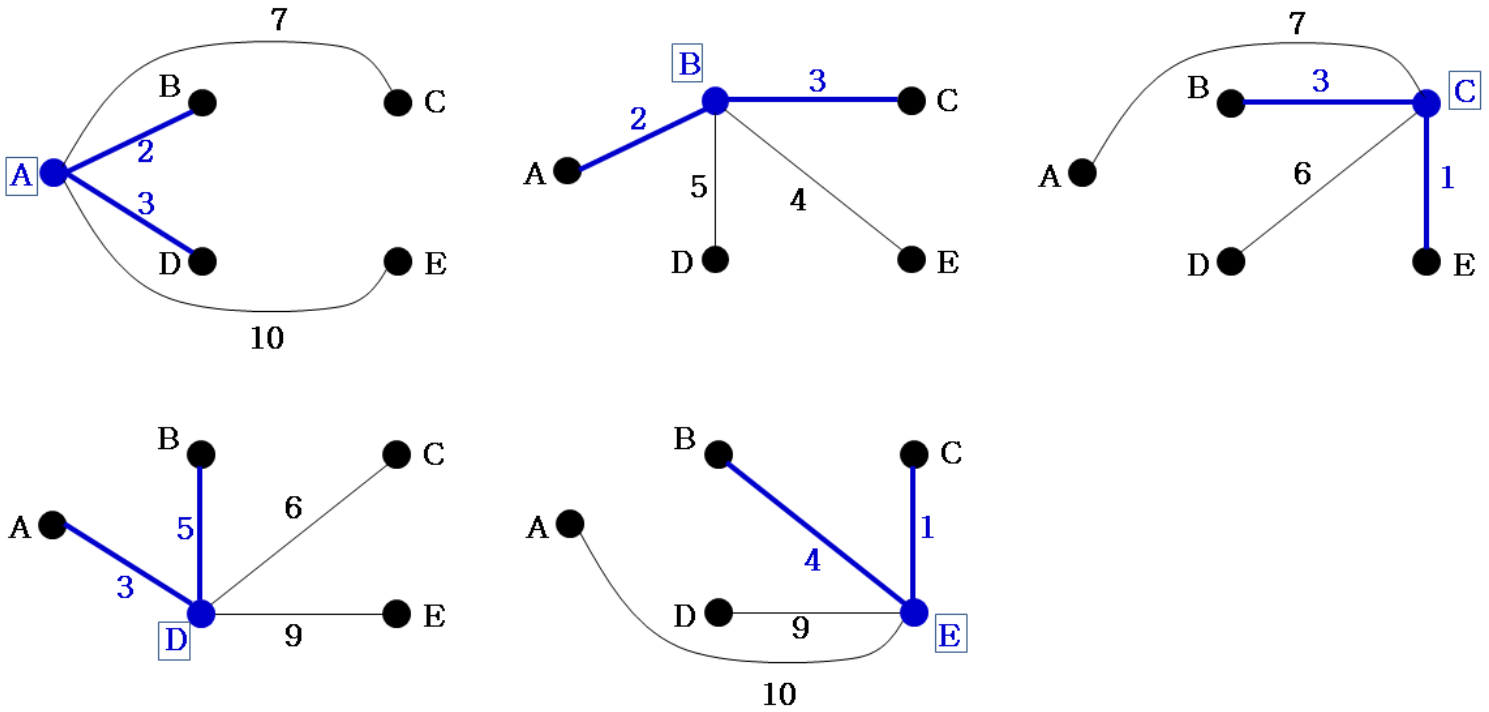


- Line 1: 초기 상태 [A]의 한정값 계산
- 초기 상태는 경로를 시작하기 전이므로, 각 점에 인접한 선분의 가중치 중에서 가장 작은 2개의 가중치의 합을 구한 다음에, 모든 점의 합의 1/2을 한정값으로 정한다.
    - 점 A: 2, 3
    - 점 B: 2, 3
    - 점 C: 1, 3
    - 점 D: 3, 5
    - 점 E: 1, 4

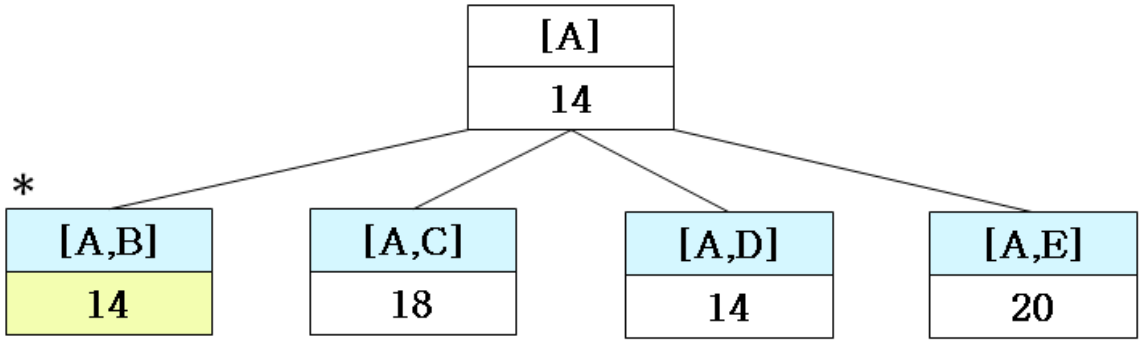
- 따라서 초기 상태의 한정값은 다음과 같이 계산된다.

A B C D E

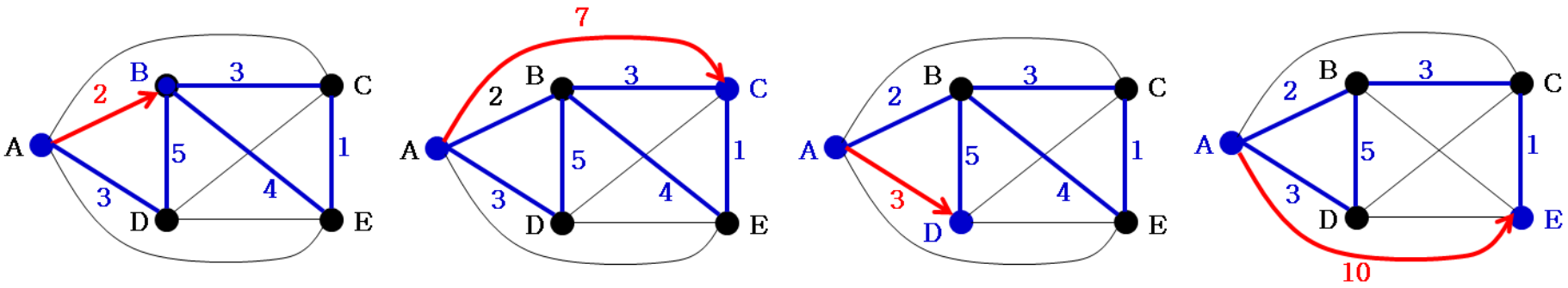
- [(2+3) + (2+3) + (1+3) + (3+5) + (1+4)] x 1/2 = 27/2 = 14



- Line 2~3: activeNodes={S}, bestValue=∞로 각각 초기화
- Line 4의 while-루프가 activeNodes 집합이 공집합이 될 때까지 수행
- Line 5: activeNodes 집합에 초기 상태 [A]만 있으므로, S_{min}=[A]
- Line 6: [A]가 activeNodes 집합으로부터 제거되어 일시적으로 activeNodes 집합은 공집합.
- Line 7: S_{min} (즉, 상태 [A])의 자식 상태 노드를 아래와 같이 생성하고, 각각 한정값을 구한다.
  - 여기서 자식 노드는 두 번째 방문하는 점이 B인 상태 [A,B], C인 상태 [A,C], D인 상태 [A,D], E인 상태 [A,E]이다.

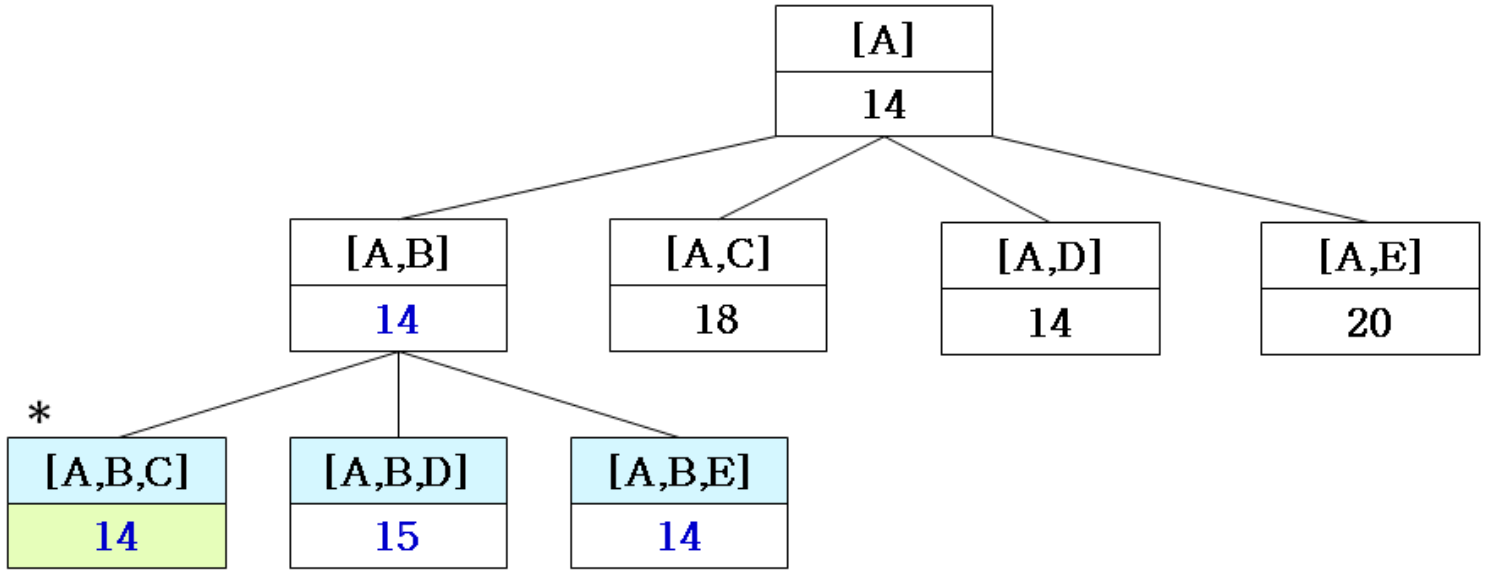


- 상태 [A,B], [A,C], [A,D], [A,E]의 한정값은 다음과 같이 각각 계산됨

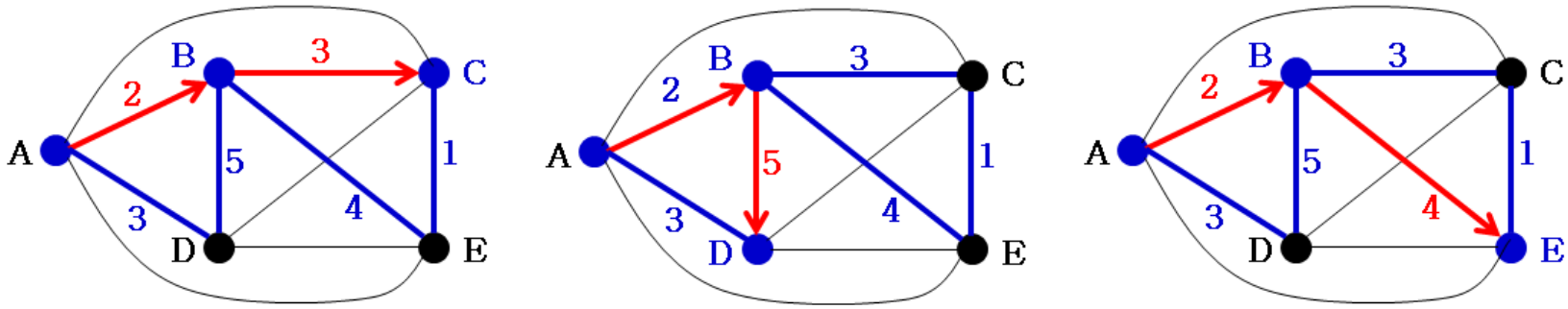


- Line 8의 for-루프: 위와 같이 생성된 4개 (k=4)의 상태 각각에 대하여, (즉, S'_1=[A,B], S'_2=[A,C], S'_3=[A,D], S'_4=[A,E]) line 9~15를 수행한다.
- Line 9: i=1: S'_1의 한정값인 14와 현재의 bestValue인 ∞를 비교하여서 if-조건이 '거짓'이고, line 11에서 상태 [A,B]가 완전한 해가 아니므로, line 14~15에서 S'_1을 activeNodes에 추가
- 이와 유사하게 i=2, 3, 4일 때에도 각각 S'_2, S'_3, S'_4가 activeNodes에 추가
- activeNodes = {[A,B], [A,C], [A,D], [A,E]}

- 다음으로 line 4 while-루프의 조건 검사에서 activeNodes가 공집합이 아니므로, line 5에서 한정값이 가장 작은 상태를 찾는다. 상태 [A,B]와 [A,E]가 동일한 최소의 한정값을 가지므로 이 중에서 임의로 S_{min} = [A,B]라고 하자.
- Line 6: activeNodes로부터 [A,B]를 제거하여, activeNodes = { [A,C], [A,D], [A,E]}가 된다.
- Line 7: [A,B]의 자식 상태를 아래와 같이 생성하고, 각각의 한정값을 계산한다.
  - 여기서 자식 노드는 세 번째 방문하는 점이 C인 상태 [A,B,C], D인 상태 [A,B,D], E인 상태 [A,B,E]이다.



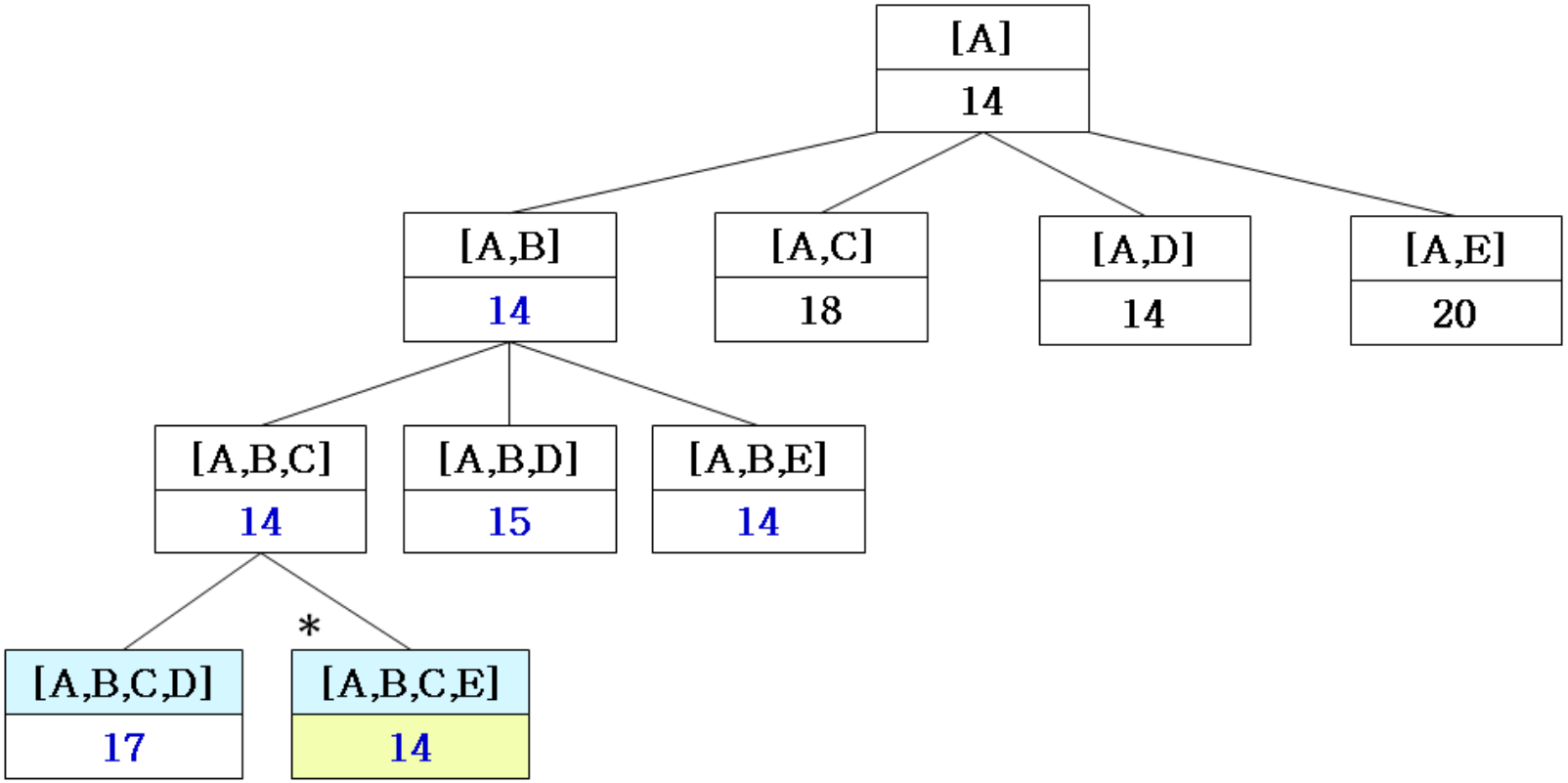
- 상태 [A,B,C], [A,B,D], [A,B,E]의 한정값은 다음과 같이 각각 계산된다.



- [A,B,C]의 한정값: ([**2**+3]+[**2**+**3**]+[1+**3**]+[3+5]+[1+4])/2 = 27/2 = 14
- [A,B,D]의 한정값: ([**2**+3]+[**2**+**5**]+[1+3]+[3+**5**]+[1+4])/2 = 29/2 = 15
- [A,B,E]의 한정값: ([**2**+3]+[**2**+**4**]+[1+3]+[3+5]+[1+**4**])/2 = 28/2 = 14
- Line 8의 for-루프: 위와 같이 생성된 3개 (k=3)의 상태 각각에 대하여, (즉, S'_1=[A,B,C], S'_2=[A,B,D], S'_3=[A,B,E]) line 9~15를 수행한다.

- Line 9: i=1: S'_1의 한정값인 14와 현재의 bestValue인 ∞를 비교하여서 if-조건이 '거짓'이고,
- Line 11: 상태 [A,B,C]가 완전한 해가 아니므로
- Line 14~15에서 S'_1을 activeNodes에 추가한다.
- 이와 유사하게 i=2, 3일 때에도 각각 S'_2, S'_3이 activeNodes에 추가된다. 따라서activeNodes = {[A,C], [A,D], [A,E], [A,B,C], [A,B,D], [A,B,E]}이다.
- 다음엔 line 4 while-루프의 조건 검사에서 activeNodes가 공집합이 아니므로, line 5에서 한정값이 가장 작은 상태를 찾는다.
  - 상태 [A,B,C], [A,B,E], [A,D]가 동일한 최소의 한정값을 가지므로 이 중에서 임의로 S_{min} = [A,B,C]라고 하자.

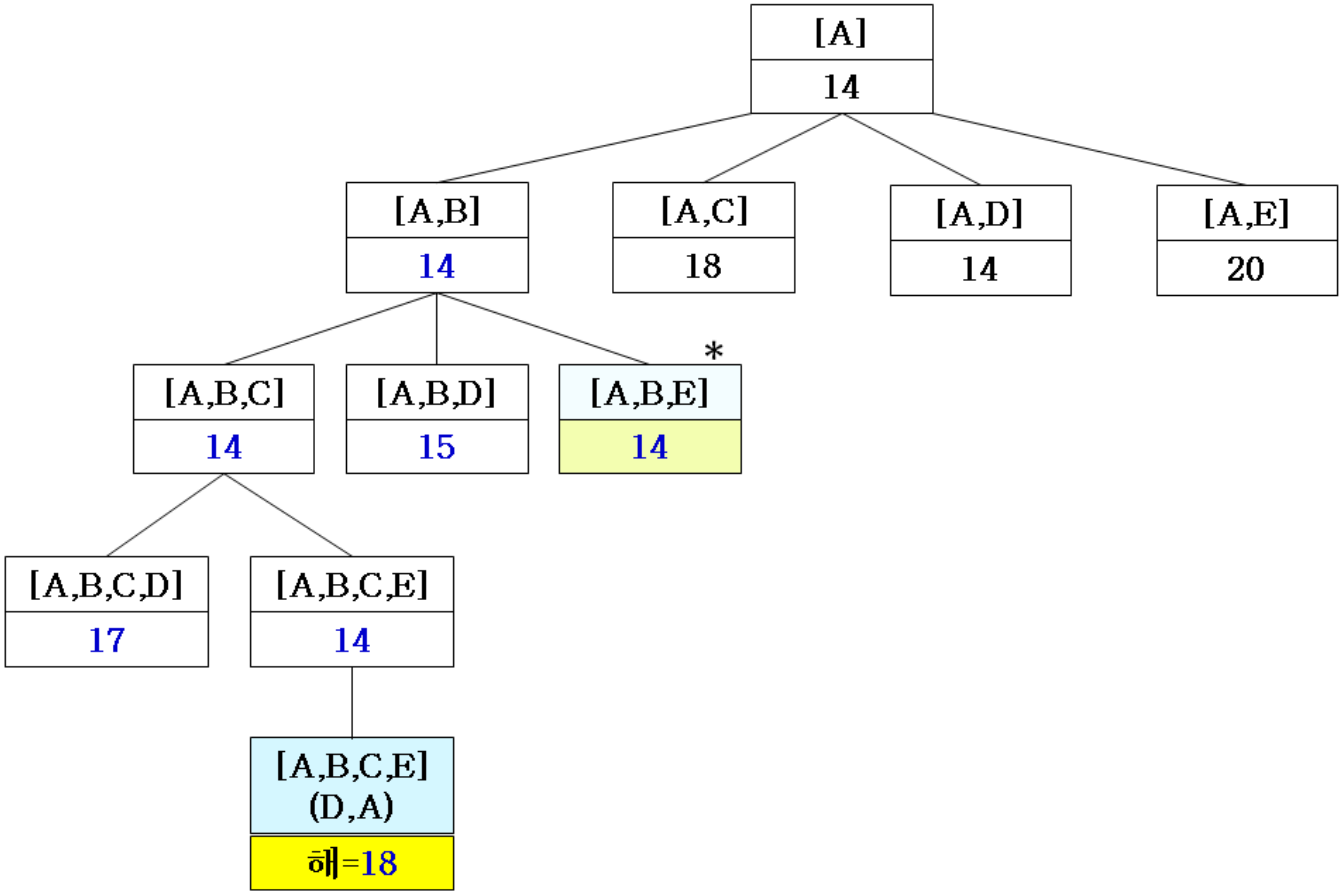
- Line 6: activeNodes로부터 [A,B,C]를 제거하여, activeNodes = {[A,C], [A,D], [A,E], [A,B,D], [A,B,E]}가 된다.
- Line 7: [A,B,C]의 자식 상태를 아래와 같이 생성하고, 각각의 한정값을 구한다.
  - 여기서 자식 노드들은 네 번째 방문하는 점이 D인 상태 [A,B,C,D]와 E인 상태 [A,B,C,E]이다.


[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

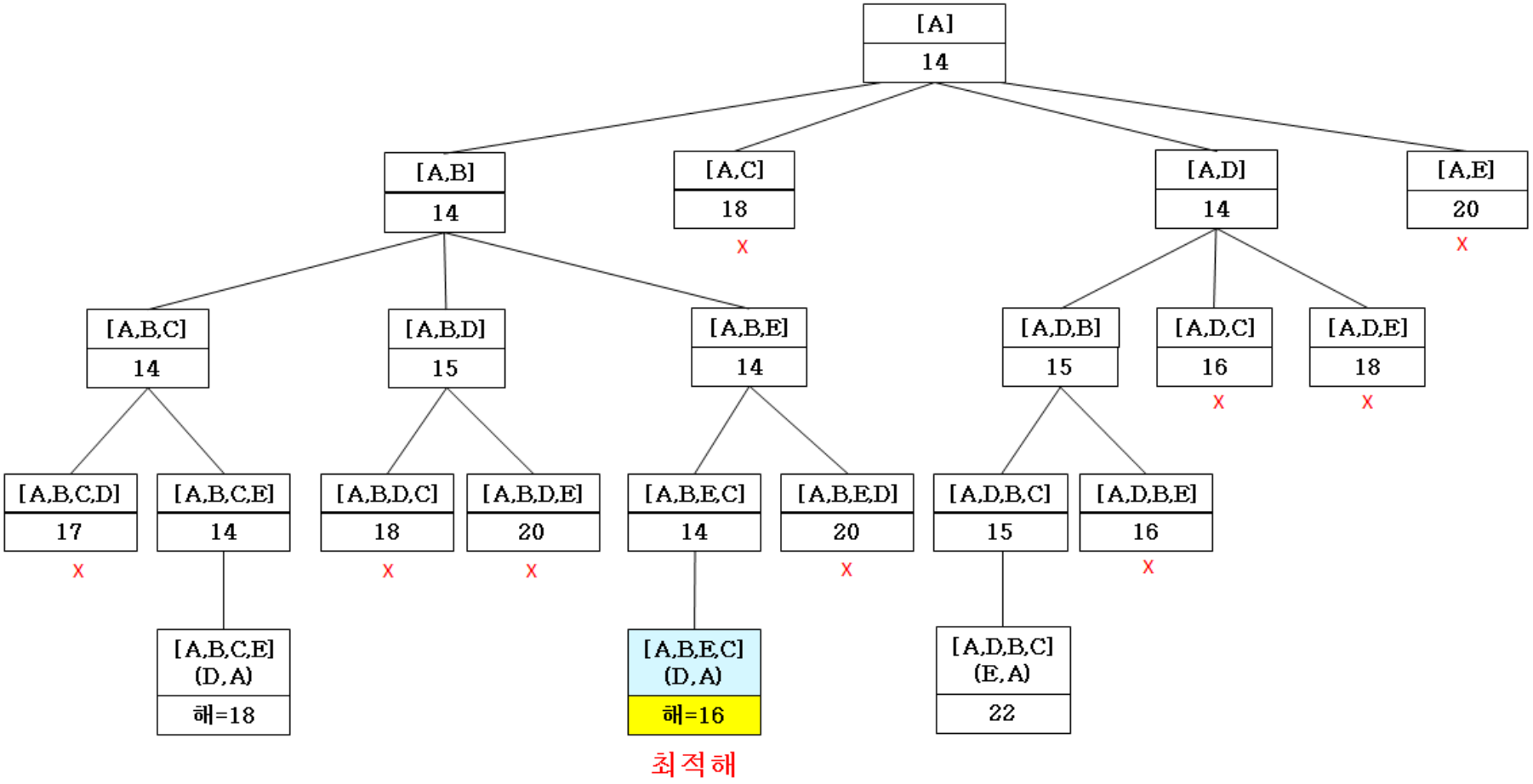
- 상태 [A,B,C,D], [A,B,C,E]의 한정값은 다음과 같이 각각 계산된다.
- [A,B,C,D]의 한정값: ([**2**+3]+[**2**+**3**]+[**6**+**3**]+[3+**6**]+[1+4])/2 = 33/2 = 17
- [A,B,C,E]의 한정값: ([**2**+3]+[**2**+**3**]+[**1**+**3**]+[3+5]+[**1**+4])/2 = 27/2 = 14
- Line 8의 for-루프에서는 위와 같이 생성된 2개 (k=2)의 상태 각각에 대하여, (즉, S'_1=[A,B,C,D], S'_2=[A,B,C,E]) line 9~15를 수행한다.

- Line 9에서 i=1: C_1의 한정값인 17과 현재의 bestValue인 ∞를 비교하여서 if-조건이 '거짓'이고
- Line 11에서 상태 [A,B,C,D]가 완전한 해가 아니므로
- Line 14~15에서 S'_1을 activeNodes에 추가시킨다.
- 이와 유사하게 i=2일 때에도 C_2가 activeNodes에 추가된다. 따라서 activeNodes = {[A,C], [A,D], [A,E], [A,B,D], [A,B,E], [A,B,C,D], [A,B,C,E]}이다.
- 다음엔 line 4 while-루프의 조건 검사에서 activeNodes가 공집합이 아니므로, line 5에서 한정값이 가장 작은 상태를 찾는다.
- 상태 [A,B,C,E], [A,B,E], [A,D]가 동일한 최소 한정값을 가지므로 이 중에서 임의로 S_{min} = [A,B,C,E]라고 하자.

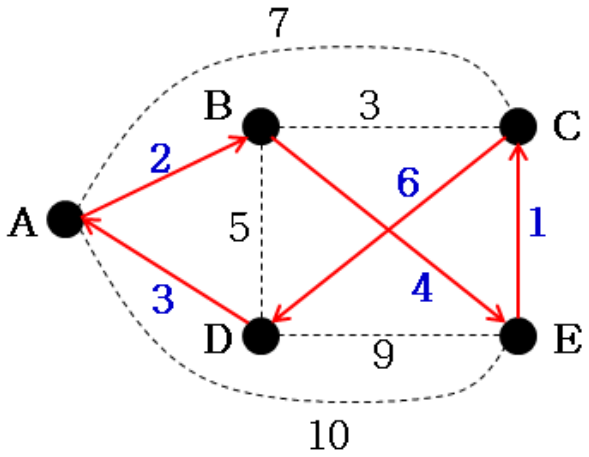
- Line 6: activeNodes로부터 [A,B,C,E]를 제거하여, activeNodes = {[A,C], [A,D], [A,E], [A,B,D], [A,B,E], [A,B,C,D]}가 된다.
- Line 7: [A,B,C,E]의 자식 상태가 1개이므로, 즉, E 다음에 방문할 점인 점 D 하나만 남아 있으므로 D를 방문하는 상태 [A,B,C,E,D]이다. 그런데 D에서 시작점 A로 돌아가야 하므로 하나의 해가 완성된 셈이다. 이 해의 경로 A-B-C-E-D-A의 거리는 2+3+1+9+3 = 18이다.
- Line 11: 해가 발견되었고, 경로 거리가 bestValue =∞보다 작으므로 if-조건이 '참'이 되어서, bestValue=18, bestSolution=[A,B,C,E,D,A]가 된다.


[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
[A,B,C,E]
(D,A)
해=18

- 다음엔 상태 [A,B,E]로부터 탐색이 시작되며, 그 최종 결과는 다음과 같다.



- 이 예제에서 상태 [A,B,E,C,D,A]가 최적해이고, 경로의 길이는 16이다. 다음 그림은 최적해에 대한 경로를 보이고 있다.



- 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

9.3 유전자 알고리즘

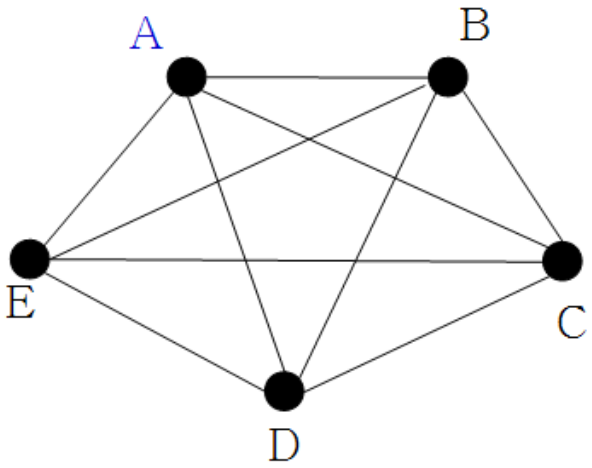
- 유전자 알고리즘 (Genetic Algorithm, GA)은 다윈의 진화론으로부터 창안된 해 탐색 알고리즘이다.
- 즉, ‘적자생존’의 개념을 최적화 문제를 해결하는데 적용한 것이다.
- 유전자 알고리즘은 다음과 같은 형태를 가진다.

GeneticAlgorithm

1. 초기 후보해 집합 G_0을 생성한다.
2. G_0의 각 후보해를 평가한다.
3. t ← 0
4. repeat
5. G_t로부터 G_{t+1}을 생성한다.
6. G_{t+1}의 각 후보해를 평가한다.
7. t ← t + 1
8. until (종료 조건이 만족될 때까지)
9. return G_t의 후보해 중에서 가장 우수한 해

- 유전자 알고리즘은 여러 개의 해를 임의로 생성하여 이들을 초기 세대 (generation) G_0로 놓고, repeat-루프에서 현재 세대의 해로부터 다음 세대의 해를 생성해가며, 루프가 끝났을 때의 마지막 세대에서 가장 우수한 해를 리턴한다.
- 이 해들은 repeat-루프의 반복적인 수행을 통해서 최적해 또는 최적해에 근접한 해가 될 수 있으므로 후보해 (candidate solution)라고 일컫는다.

- 후보해에 대한 이해: 5개의 도시 (A, B, C, D, E)에 대한 여행자 문제 (Traveling Salesman Problem)를 예로 들어보자.
- 단, 시작 도시는 A이다. 여행자 문제의 조건은 시작 도시에서 출발하여 모든 다른 도시를 1번씩만 방문하고 시작 도시로 돌아와야 하므로, ABCDEA, ACDEBA, AECDBA 등이 후보해이다.



- 이 문제의 후보해의 수는 시작 도시를 제외한 5개의 도시를 일렬로 나열하는 방법의 수와 같으므로 5! = 120이다.
- 만일 n개의 도시가 있다면, 후보해의 수는 (n-1)!이다.
- 후보해의 평가: 후보해를 평가한다는 것은 후보해의 값을 계산하는 것이다. 여행자 문제의 후보해의 값은 도시 간의 거리가 입력으로 주어지므로, 다음과 같이 계산한다.

- 후보해 ABCDEA의 값 =

  (A와 B 사이의 거리)

  + (B와 C 사이의 거리)

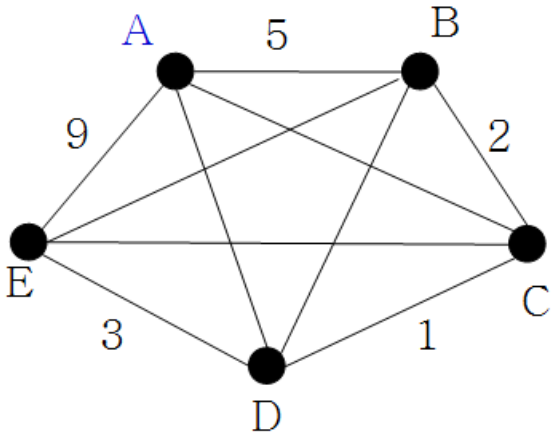
  + (C와 D 사이의 거리)

  + (D와 E 사이의 거리)

  + (E와 A 사이의 거리)

  = 5 + 2 + 1 + 3 + 9

  = 20



- 후보해의 값을 후보해의 적합도(fitness value)라고 한다.
- 후보해 중에서 최적해의 값에 근접한 적합도를 가진 후보해를 '우수한' 해라고 부른다.
- GeneticAlgorithm에서 가장 핵심적인 부분은 현재 세대의 후보해에 대해서 다음과 같은 3개의 연산을 통해서 다음 세대의 후보해를 생성하는 것이다.

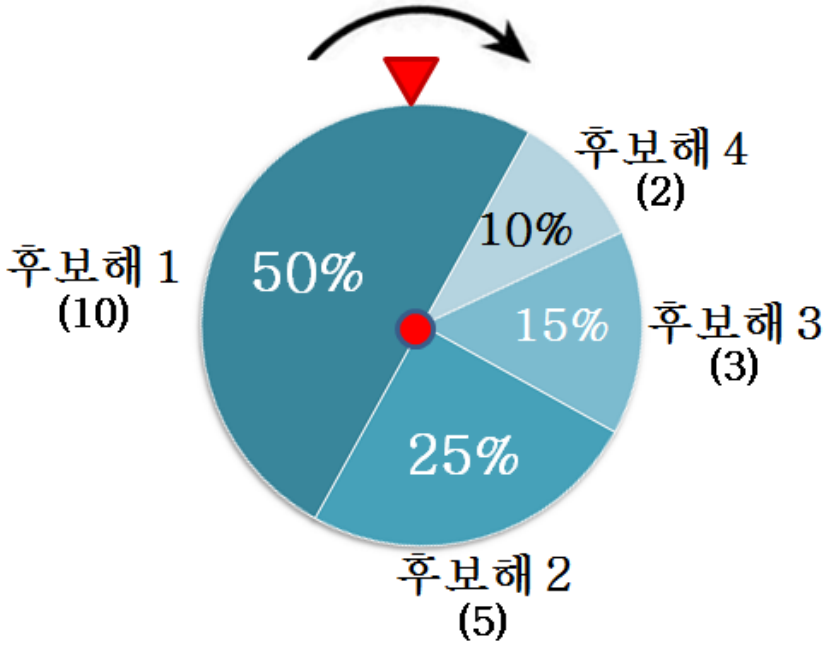
    1. 선택 (selection) 연산
    2. 교차 (crossover) 연산
    3. 돌연변이 (mutation) 연산

1. 선택 연산

- 선택 연산은 현재 세대의 후보해 중에서 우수한 후보해를 선택하는 연산이다.
- 현재 세대에 n개의 후보해가 있으면, 이들 중에서 우수한 후보해는 중복되어 선택될 수 있고, 적합도가 상대적으로 낮은 후보해 선택되지 않을 수도 있다.
- 이렇게 선택된 후보해의 수는 n개로 유지된다. 이러한 선택은 '적자생존' 개념을 모방한 것이다.

- 선택 연산을 가장 간단히 구현하는 방법은 룰렛 휠 (roulette wheel) 방법이다.
  - 각 후보해의 적합도에 비례하여 원반의 면적을 할당하고, 원반을 회전시켜서 원반이 멈추었을 때 핀이 가리키는 후보해를 선택한다.
  - 면적이 넓은 후보해가 선택될 확률이 높다.

- 후보해 1의 적합도: 10
- 후보해 2의 적합도: 5
- 후보해 3의 적합도: 3
- 후보해 4의 적합도: 2



- 각 후보해의 원반 면적은 (후보해의 적합도 / 모든 후보해의 적합도의 합)에 비례한다.
- 앞의 예제에서 모든 적합도의 합이 10 + 5 + 3 + 2 = 20이므로,
  - 후보해 1의 면적은 10/20 = 50%
  - 후보해 2의 면적은 5/20 = 25%
  - 후보해 3의 면적은 3/20 = 15%
  - 후보해 4의 면적은 2/20 = 10%
- 현재 4개의 후보해가 있으므로, 4번 원반을 돌리고 회전이 멈추었을 때 핀이 가리키는 후보해를 각각 선택한다.

2. 교차 연산

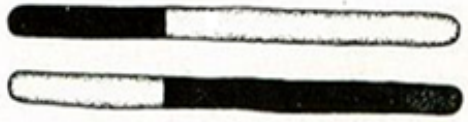
- 교차 연산은 선택 연산을 수행한 후의 후보해 사이에 수행되는데, 이는 염색체가 교차하는 것을 모방한 것이다.
- 예를 들어, 2개의 후보해가 각각 2진수로 아래와 같이 표현된다면, 교차점 이후의 부분을 서로 교환하여 교차 연산이 수행되며, 그 결과 각각 새로운 후보해가 만들어진다.
- 이와 같은 교차 연산을

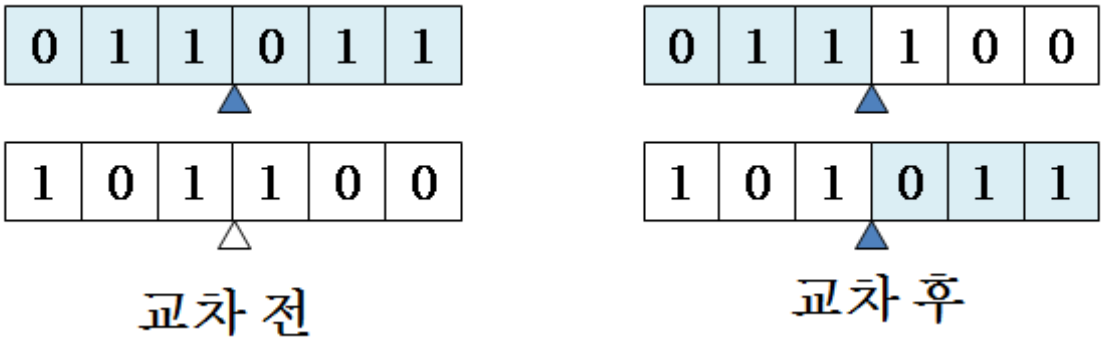
  1-점 (point) 교차 연산이

  라고 한다.

염색체
교차 전

염색체
교차 후

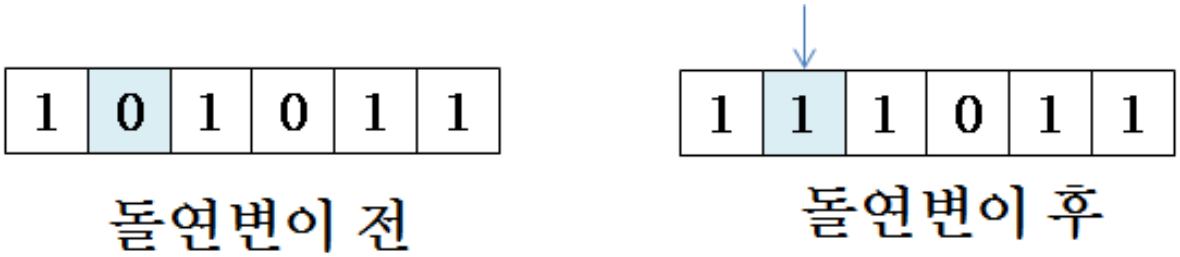




- 후보해가 길게 표현되면, 여러 개의 교차점을 임의로 정하여 교차 연산을 할 수도 있다.
- 교차 연산의 목적: 선택 연산을 통해서 얻은 우수한 후보해보다 우수한 후보해가 생성하는 것
- 문제에 따라서 교차 연산을 수행할 후보해의 수를 조절하는데, 이를 교차율 (crossover rate)이라고 한다. 일반적으로 교차율은 0.2~1.0 범위에서 정한다.

3. 돌연변이 연산

- 교차 연산이 수행된 후에 돌연변이 연산을 수행한다.
- 돌연변이 연산은 아주 작은 확률로 후보해의 일부분을 임의로 변형시키는 것이다.
- 이 확률을 돌연변이율 (Mutation Rate)이라고 하며, 일반적으로 (1/PopSize)~(1/Length)의 범위에서 사용된다.
  - 여기서 PopSize란 모집단 크기 (Population Size)로서 한 세대의 후보해의 수이고,
  - Length란 후보해를 이진 표현으로 했을 경우의 bit 수이다.
- 다음의 예는 두 번째 bit가 0에서 1로 돌연변이된 것을 보여주고 있다.

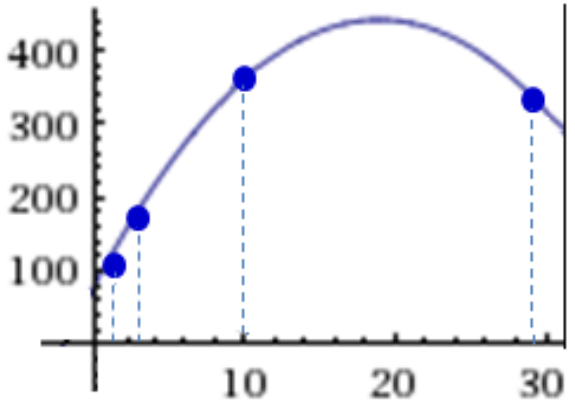


- 돌연변이가 수행된 후에 후보해의 적합도가 오히려 나빠질 수도 있다.
- 돌연변이 연산의 목적은 다음 세대에 돌연변이가 이루어진 후보해와 다른 후보해를 교차 연산함으로서 이후 세대에서 매우 우수한 후보해를 생성하기 위한 것이다.
- GeneticAlgorithm의 종료 조건은 일정하지 않다. 왜냐하면 유전자 알고리즘이 항상 최적해를 찾는다는 보장이 없기 때문이다.
- 따라서 일반적으로 알고리즘을 수행시키면서 더 이상 우수한 해가 출현하지 않으면 알고리즘을 종료시킨다.

- 다음의 2차 함수에 대해 유전자 알고리즘으로 $0 \le x \le 31$ 구간에서 최댓값을 찾아보자.

$$f(x) = -x^2 + 38x + 80$$

- 먼저 한 세대의 후보해 수를 4로 정하고, 0~31에서 랜덤하게 4개의 후보해인 1, 29, 3, 10을 선택하였다고 가정하자.
- 이들이 초기 세대를 구성하는 후보해들이다.
- 각 후보해의 적합도:
  - $f(1) = -(1)^2 + 38(1) + 80 = 117$,
  - $f(29) = 341$,
  - $f(3) = 185$,
  - $f(10) = 360$

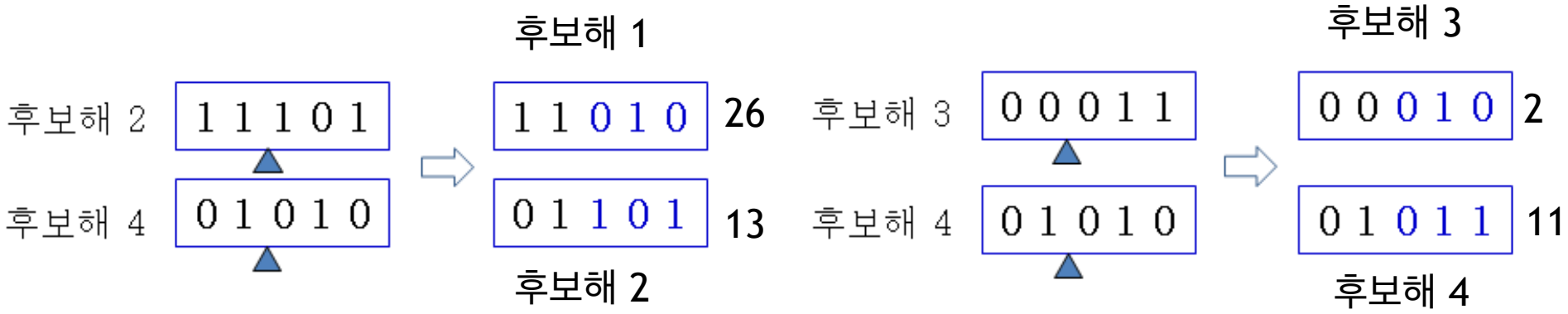


후보해	2진 표현	x	적합도 f(x)	원반 면적 (%)
1	0 0 0 0 1	1	117	12
2	1 1 1 0 1	29	341	34
3	0 0 0 1 1	3	185	18
4	0 1 0 1 0	10	360	36
계			1,003	100
평균			250.75	

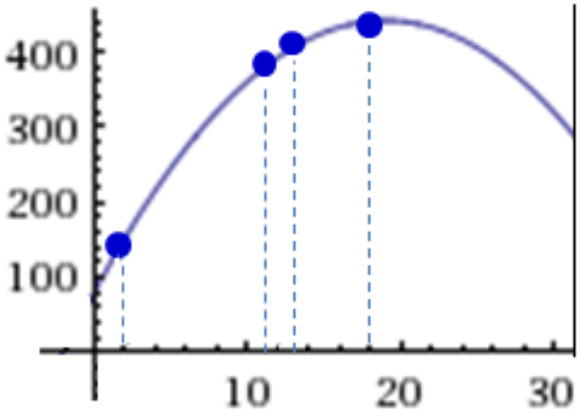
- 위의 표는 각 후보해의 2진 표현, 적합도, 룰렛 휠 선택을 위한 원반 면적을 보이고 있다.
- 또한 초기 세대의 평균 적합도는 250.75이다.

- 선택 연산: 룰렛 휠 선택 방법을 이용하여, 후보해 4는 2번 선택되었고, 후보해 2와 3은 각각 1번 선택되었으며, 후보해 1은 선택 안되었다고 가정하자.

- 교차 연산: 후보해 4가 2개이므로, 후보해 2와 4를 짝짓고, 후보해 3과 4를 짝지어 아래와 같이 교차 연산을 수행한다. 단, 1점-교차 연산을 위해 아래와 같이 임의의 교차점이 선택되었다고 가정하자.



- 돌연변이 연산: 교차 연산 후에 후보해 1의 왼쪽에서 두 번째 bit가 돌연변이가 되어서 ‘1’에서 ‘0’으로 바뀌었다고 가정하자. 다른 후보해는 교차 연산 후와 동일하다.



후보해	2진 표현	x	적합도 f(x)	원반 면적 (%)
1	1 0 0 1 0	18	440	32
2	0 1 1 0 1	13	405	29
3	0 0 0 1 0	2	152	11
4	0 1 0 1 1	11	377	27
계			1,374	100
평균			343.5	

- 두 번째 세대의 평균 적합도가 343.5로 많이 향상되었다.
- repeat-루프를 더 수행하여 후보해의 적합도가 변하지 않으면, 알고리즘을 종료하고, 후보해 중에서 가장 적합도가 높은 후보해를 리턴한다.

- 다음은 여행자 문제를 해결할 때 GeneticAlgorithm을 적용하기 위해 사용되는 2가지의 교차 연산을 소개한다.

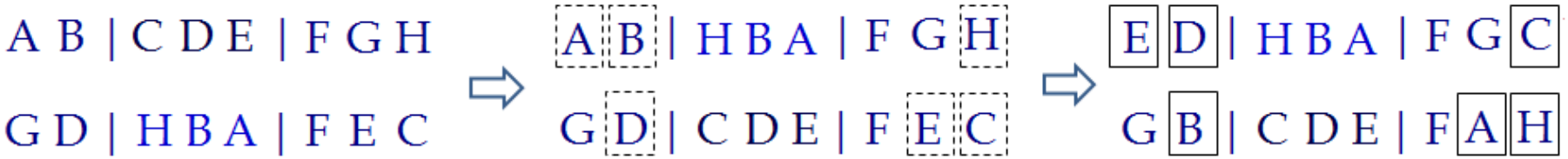
    1. 2점 교차 연산

    2. 사이클 교차 연산

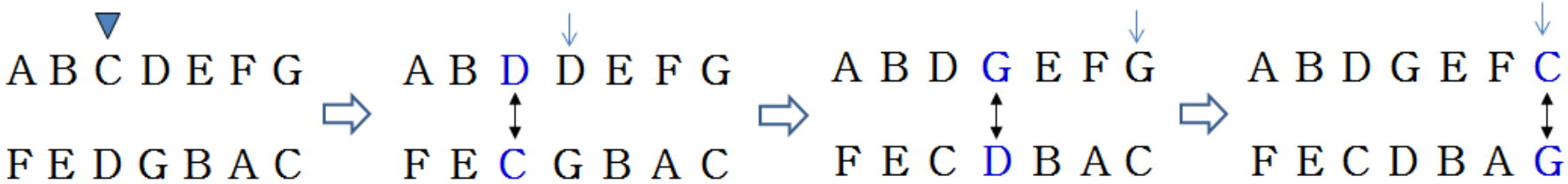
- 여행자 문제의 후보해는 시작 도시부터 각 도시를 중복 없이 나열하여 만들어진다.

- 2점-교차 연산: 임의의 2점을 정한 후, 가운데 부분을 서로 바꾼다. 이후 중복되는 도시 (점선 박스 내의 도시)를 현재 후보해에 없는 도시로 차례로 바꾼다.

- 예를 들어, ABCDEFGH가 ABHBAFGH이 되면, C, D, E가 없어졌고, H, B, A는 각각 2개씩 있다. 따라서 가운데 부분의 좌우에 있는 H, B, A를 각각 C, D, E로 바꾸어 주면 EDHBAFGC가 된다.



- 사이클 교차 연산: 후보해 1에서 임의 도시 A를 선택한 후, A와 같은 위치에 있는 후보해 2의 도시 B와 바꾼다.
- 바꾼 후에는 후보해 1에는 A가 없고 B가 2개 존재하게 된다.
- 이를 해결하기 위해 후보해 1에 원래부터 있었던 B를 후보해 2에 B와 같은 위치에 있는 도시와 바꾼다.
- 이렇게 반복하여 A가 후보해 2로부터 후보해 1로 바뀌게 되면 교차 연산을 마친다.



- 위의 예를 보면, 처음에 후보해 1에서 임의로 C가 선택된 후, C와 같은 위치의 후보해 2의 도시 D와 서로 바꾼다.
- 그러면 후보해 1에는 2개의 D가 있다.
- 후보해 1에 원래부터 있었던 D (화살표로 가리키고 있는)를 후보해 2의 같은 위치의 G와 서로 바꾼다.
- 그러면 후보해 1에는 2개의 G가 있다. 후보해 1에 원래부터 있었던 G (화살표로 가리키고 있는)를 후보해 2의 같은 위치의 C와 서로 바꾼다. 이때 처음에 후보해 1에서 선택했던 C가 후보해 1로 바뀌어 올라오게 되어서, 교차 연산을 마친다.
- 즉, C→D→G→C의 사이클을 따라서 상하로 도시를 바꾼 것이다.

- 유전자 알고리즘은 실제로 적지 않은 실험을 요구한다. 다음과 같은 파라미터의 값들과 적절한 연산을 선택해야 한다.

    1. 모집단 크기
    2. 선택 연산
    3. 교차 연산과 교차율
    4. 돌연변이율
    5. repeat-루프의 종료 조건

응 용

- 유전자 알고리즘은 문제의 최적해를 알 수 없고, 기존의 어느 알고리즘으로도 해결하기 어려운 경우에, 최적해에 가까운 해를 찾는데 매우 적절한 알고리즘이다.
- 유전자 알고리즘이 최적해를 반드시 찾는다는 보장은 없으나 대부분의 경우 매우 우수한 해를 찾는다.
- 통 채우기
- 작업 스케줄링
- 차량 경로
- 배낭 문제

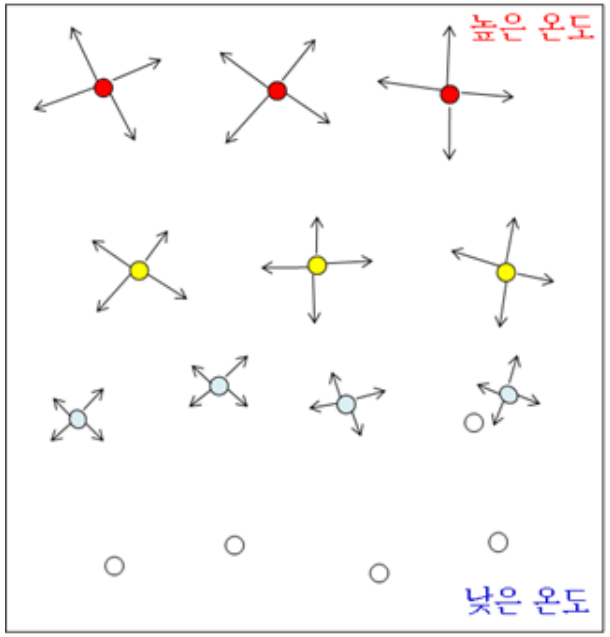
- 로봇 공학
- 기계 학습 (Machine Learning)
- 신호 처리 (Signal Processing)
- 반도체 설계
- 항공기 디자인
- 통신 네트워크
- 패턴 인식

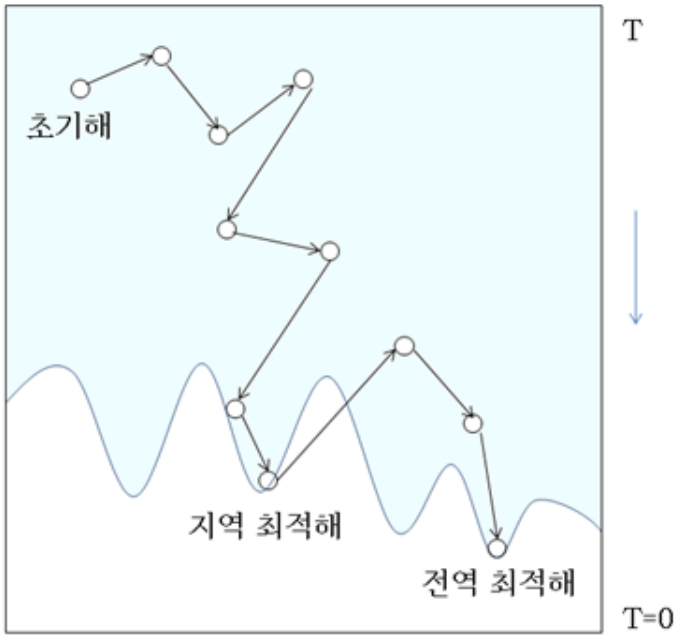
- 그 외에도 경제, 경영, 환경, 의학, 음악, 군사 등과 같은 다양한 분야에서 최적화 문제를 해결하는데 활용된다.

9.4 모의 담금질 기법

- 모의 담금질 (Simulated Annealing) 기법은 높은 온도에서 액체 상태인 물질이 온도가 점차 낮아지면서 결정체로 변하는 과정을 모방한 해 탐색 알고리즘이다.
- 용융 상태에서는 물질의 분자가 자유로이 움직이는데 이를 모방하여, 해를 탐색하는 과정도 특정한 패턴 없이 이루어진다.
- 그러나 온도가 점점 낮아지면 분자의 움직임이 점점 줄어들어 결정체가 되는데, 해 탐색 과정도 이와 유사하게 점점 더 규칙적인 방식으로 이루어진다.

- 이러한 방식으로 해를 탐색하려면, 후보해에 대해 이웃하는 해 (이웃해)를 정의하여야 한다.
- 아래의 오른쪽 그림에서 각 점은 후보해이고 아래쪽에 위치한 해가 위쪽에 있는 해보다 우수한 해이다. 또한 2개의 후보해 사이의 화살표는 이 후보해들이 서로 이웃하는 관계임을 나타낸다.





- 앞의 그림은 모의 담금질 기법이 최솟값을 탐색하는 과정을 보이고 있다.
- 높은 T에서의 초기 탐색은 최솟값을 찾는데도 불구하고 확률 개념을 도입하여 현재 해의 이웃해 중에서 현재 해보다 '나쁜' 해로 (위 방향으로) 이동하는 자유로움을 보인다.
- 그러나 T가 낮아지면서 점차 탐색은 아래 방향으로 향한다.
- 즉, T가 낮아질수록 위 방향으로 이동하는 확률이 점차 작아진다.
- 앞의 그림에서 처음 도착한 골짜기 (지역 최적해, local optimum)에서 더 이상 아래로 탐색할 수 없는 상태에 이르렀을 때 '운 좋게' 위 방향으로 탐색하다가 전역 최적해 (global optimum)를 찾은 것을 보여준다.

- 그러나 유전자 알고리즘과 마찬가지로 모의 담금질 기법도 항상 전역 최적해를 찾아준다는 보장은 없다.
- 모의 담금질 기법의 또 하나의 특징은 하나의 초기 해로부터 탐색이 진행된다는 것이다. 반면에 유전자 알고리즘은 여러 개의 후보해를 한 세대로 하여 탐색을 진행한다.
- 다음은 모의 담금질 기법의 기본적인 알고리즘이다.

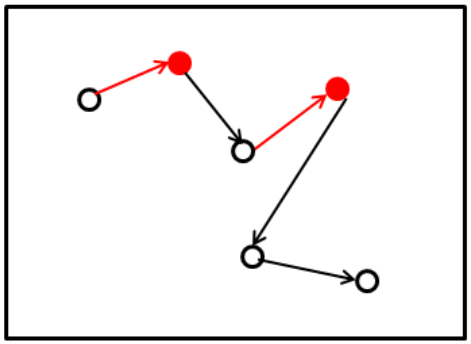
SimulatedAnnealing

1. 임의의 후보해 s를 선택한다.
2. 초기 T를 정한다.
3. repeat
4. for i = 1 to k_T { // k_T는 T에서의 for-루프 반복 횟수이다.
5. s의 이웃해 중에서 랜덤하게 하나의 해 s'를 선택한다.
6. d = (s'의 값) - (s의 값)
7. if (d < 0) // 이웃해인 s'가 더 우수한 경우
8. s ← s'
9. else // s'가 s보다 우수하지 않은 경우
10. q ← (0,1) 사이에서 랜덤하게 선택한 수
11. if (q < p) s ← s' // p는 자유롭게 탐색할 확률이다.

    }
12. T ← αT // 1보다 작은 상수 α 를 T에 곱하여 새로운 T를 계산
13. until (종료 조건이 만족될 때까지)
14. return s

- Line 1: 임의의 해 s를 선택하여 탐색을 시작한다.
- Line 2: 충분히 높은 값의 T를 실험을 통해 정한다.
- Line 3~12의 repeat-루프는 종료 조건이 만족될 때까지 수행된다. repeat-루프 내에서는 for-루프가 k_T만큼 반복 수행된다.
  - k_T는 현재 T에서 for-루프가 수행되는 횟수인데, 일반적으로 T가 작아질수록 k_T가 커지도록 조절한다.
  - k_T는 입력의 크기와 이웃하는 해의 수 등에 따라 실험하여 정한다.
- Line 5: 현재 해인 s에 이웃하는 해 중에서 임의로 s'를 선택
- Line 6: s'와 s의 값의 차이인 d를 계산한다.
  - 해의 값은 유전자 알고리즘에서의 적합도와 같다.

- Line 7의 if-조건에서는 s'가 s보다 우수한 해이면 다음 탐색을 위해 s'가 현재 해인 s가 된다.
- Line 9~11: s'가 s보다 우수하지 않더라도 0~1사이에서 랜덤하게 선택한 수 q가 확률 p보다 작으면, s'가 현재 해인 s가 될 기회를 준다.
  - 즉, 이 기회가 그림에서 최소값을 찾는데도 불구하고 위쪽에 위치한 이웃해로 탐색을 진행하는 것이다.
  - 여기서 p는 자유롭게 탐색할 확률이다.



- Line 12: T를 일정 비율 α로 감소시킨다. 실제로 $0.8 \le \alpha \le 0.99$ 범위에서 미리 정한 냉각율 α (cooling ratio)를 T에 곱하여 새로운 T를 계산한다.
  - 일반적으로 α 를 0.99에 가까운 수로 선택하여, T가 천천히 감소되도록 조절한다.
- Line 13의 종료 조건은 더 이상 우수한 해를 찾지 못하거나, 미리 정한 repeat-루프의 최대 반복 횟수의 초과 여부로 정한다.
- 마지막으로 repeat-루프가 끝나면 line 14에서 현재 해인 s를 리턴한다.

- 모의 담금질 기법은 T가 높을 때부터 점점 낮아지는 것을 확률 p에 반영시켜서 초기에는 탐색이 자유롭다가 점점 규칙적이 되도록 한다.
- 그러므로 확률 p는 T에 따라서 변해야 한다.
  - T가 높을 땐, p도 크게 하여 자유롭게 탐색할 수 있도록 하고,
  - T가 0이 되면, p를 0으로 만들어서 line 11에서 나쁜 이웃해 s'가 s가 되지 못하도록 한다.
- p에 반영시켜야 할 또 하나의 요소는 s'와 s의 값의 차이 d이다.
  - d 값이 크면, p를 작게 하고,
  - d 값이 작으면, p를 크게 한다.
- 이렇게 하는 이유는 값의 차이가 큼에도 불구하고 p를 크게 하면 그 동안 탐색한 결과가 무시되어 랜덤하게 탐색하는 결과를 낳기 때문이다.

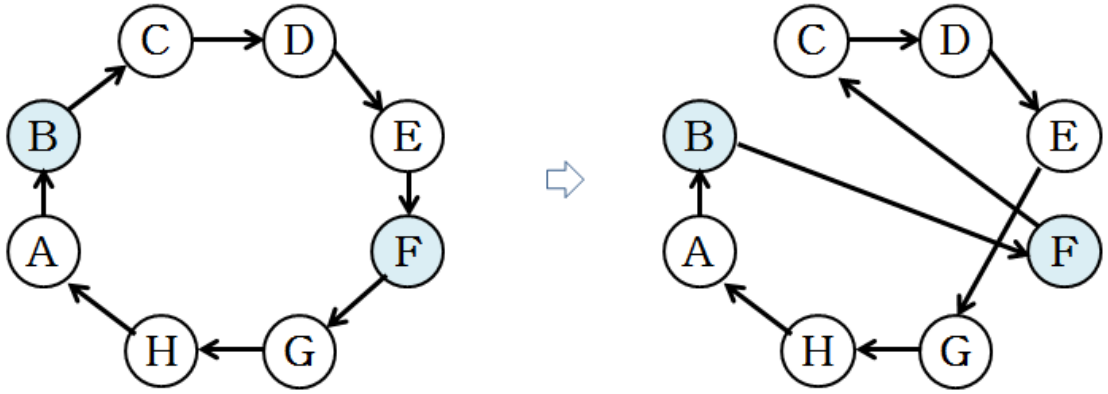
- 이 2가지 요소를 종합하여 확률 p는 다음과 같이 정의된다.

$$p = 1 / e^{d/T} = e^{-d/T}$$

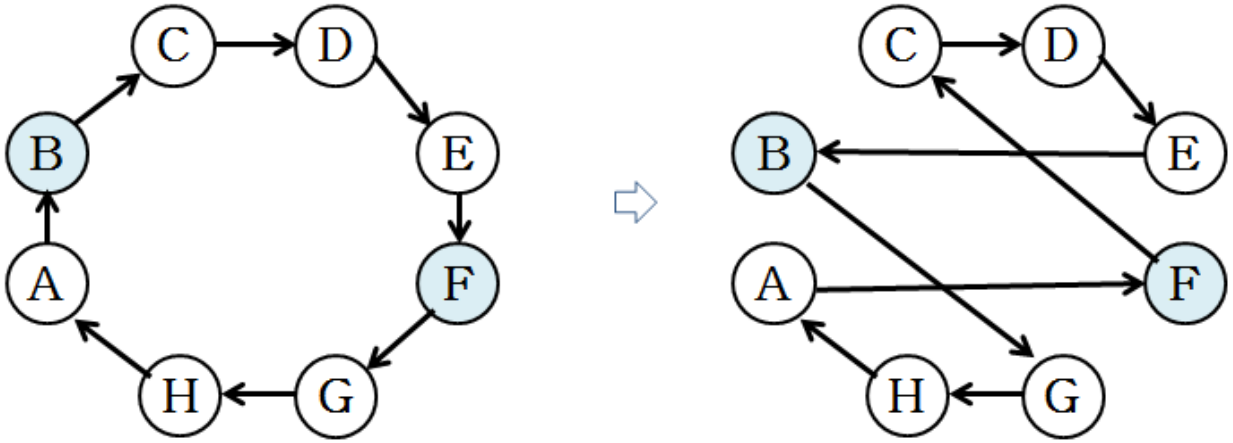
- 위의 식에서 T는 큰 값으로부터 0까지 변하고, d는 s'와 s의 값의 차이이다.
- 모의 담금질 기법으로 해를 탐색하려면, 먼저 후보해에 대한 이웃 해를 정의하여야 한다.
- 여행자 문제의 이웃 해 정의 3가지 예
  1. 삽입 (Insertion)
  2. 교환 (Switching)
  3. 반전 (Inversion)

- 삽입 (Insertion): 2개의 도시를 랜덤하게 선택한 후에, 두 번째 도시를 첫 번째 도시 옆으로 옮기고, 두 도시 사이의 도시들은 오른쪽으로 1칸씩 이동한다.

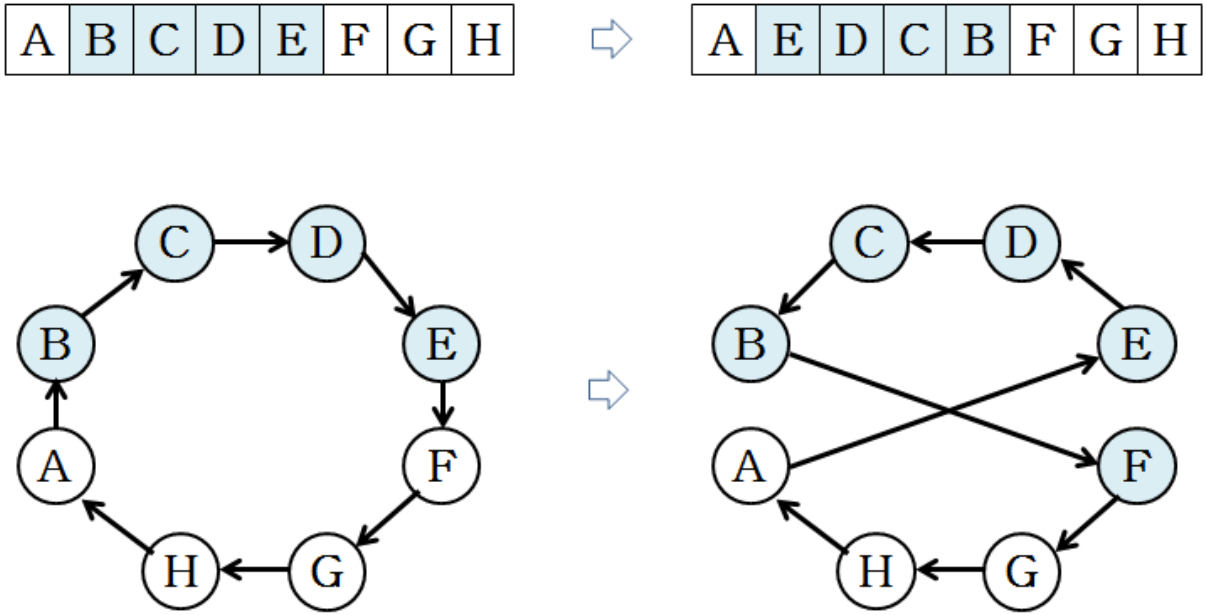
- 아래의 예제에서 도시 B와 F가 랜덤하게 선택되었다고 하면, F가 B의 바로 오른쪽으로 이동한 후, B와 F 사이의 C, D, E를 각각 오른쪽으로 1칸씩 이동한다.



- 교환 (Switching): 2개의 도시를 랜덤하게 선택한 후에, 그 도시들의 위치를 서로 바꾼다.
- 아래의 예제에서 도시 B와 F가 랜덤하게 선택되었다면, B와 F의 자리를 서로 바꾼다.



- 반전 (Inversion): 2개의 도시를 랜덤하게 선택한 후에, 그 두 도시 사이의 도시를 역순으로 만든다. 단, 선택된 두 도시도 반전에 포함시킨다.
- 아래의 예제에서는 도시 B와 E가 랜덤하게 선택되었다면, [B C D E]가 역순으로 [E D C B]로 바뀐다.



- SimulatedAnnealing 알고리즘은 line 5에서 현재 해 s에 이웃하는 해 중 하나를 랜덤하게 하나를 선택하는데, 이때 위에 소개된 이웃해 정의 중의 하나를 이용하여 여행자 문제를 해결한다.

응 용

- 반도체 회로 설계
- 유전자 배열
- 단백질 구조 연구
- 경영 분야의 재고 계획
- 원자재 조달
- 상품의 생산 및 유통
- 운송 분야의 스케줄링
- 건축 분야의 빌딩 구획 및 배치 (Building Layout)
- 항공기 디자인
- 복합 물질 모델링
- 금융 분야의 은행의 재무 분석 등 매우 광범위하게 활용된다.

요 약

- 백트래킹 (Backtracking) 기법은 해를 찾는 도중에 '막히면' 되돌아가서 다시 해를 찾아 가는 기법으로 상태 공간 트리에서 깊이 우선 탐색 (Depth First Search) 방법으로 해를 찾는 알고리즘이다.
- 백트래킹 기법의 시간복잡도는 상태 공간 트리의 노드 수에 비례하고, 이는 모든 경우를 다 검사하여 해를 찾는 완결 탐색 (Exhaustive Search)의 시간복잡도와 같다. 그러나 일반적으로 백트래킹 기법은 '가지치기'를 하므로 완결 탐색보다 훨씬 효율적이다.
- 분기 한정 기법은 상태 공간 트리의 각 노드(상태)에 특정한 값(한정값)을 부여하고, 노드의 한정값을 활용하여 가지치기를 함으로서 백트래킹 기법보다 빠르게 해를 찾는다.

- 분기 한정 기법에서는 가장 우수한 한정값을 가진 노드를 먼저 탐색하는 최선 우선 탐색 (Best First Search)으로 해를 찾는다. 또한 분기 한정 기법은 최적화 문제를 해결하는데 적절하다.

- 유전자 알고리즘 (Genetic Algorithm, GA)은 다윈의 진화론으로부터 창안된 해 탐색 알고리즘이다. 즉, '적자생존'의 개념을 최적화 문제를 해결하는데 적용한 것이다.

- 유전자 알고리즘은 여러 개의 해를 임의로 생성하여 이들에 대해 선택, 교차 돌연변이 연산을 반복 수행하여 마지막에 가장 우수한 해를 리턴한다.

- 유전자 알고리즘은 문제의 최적해를 알 수 없고, 기존의 어느 알고리즘으로도 해결하기 어려운 경우에, 최적해에 가까운 해를 찾는 데 매우 적절한 알고리즘이다.
- 모의 담금질 (Simulated Annealing) 기법은 높은 온도에서 액체 상태인 물질이 온도가 점차 낮아지면서 결정체로 변하는 과정을 모방한 해 탐색 알고리즘이다.
- 유전자 알고리즘과 마찬가지로 모의 담금질 기법도 항상 전역 최적해를 찾아준다는 보장은 없다.