13주차 1차시 해 탐색 알고리즘의 이해 I

【학습목표】

1. 해 탐색 알고리즘을 이해할 수 있다.
2. 백 트래킹 기법을 이해할 수 있다.

지난 강의 정리

1. 동적 계획알고리즘 이란
동적 계획 (Dynamic Programming) 알고리즘은 그리디 알고리즘과 같이 최적화 문제를 해결하는 알고리즘이다. 동적 계획 알고리즘은 먼저 입력 크기가 작은 부분 문제들을 모두 해결한 후에 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여, 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘이다.
1) 모든 쌍 최단경로
2) 연속 행렬의 곱셈
3) 편집 거리 문제
4) 배낭 문제
5) 동전 거스름돈

2. 그리디 알고리즘은 욕심쟁이 알고리즘, 탐욕적 방법, 탐욕 알고리즘 등으로 불리기도 한다.
그리디 알고리즘은 (입력) 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심내어' 최소값 또는 최대값을 가진 데이터를 선택하여 최적해를 구하는 알고리즘이다.
1) 최소 신장 트리
2) 최단경로,
3) 부분 배낭문제
4) 집합 커버,
5) 동전 거스름돈,
6) 작업 스케줄링

3. 동전 거스름돈
- 동전 거스름돈 (Coin Change) 문제를 해결하는 가장 간단하고 효율적인 방법은 남은 액수를 초과하지 않는 조건하에 '욕심내어' 가장 큰 액면의 동전을 취하는 것이다.
- 다음은 동전 거스름돈 문제의 최소 동전 수를 찾는 그리디 알고리즘이다. 단, 동전의 액면은 500원, 100원, 50원, 10원, 1원이다.

* 알고리즘

CoinChange

입력: 거스름돈 액수 W

출력: 거스름돈 액수에 대한 최소 동전 수

1. change=W, n500=n100=n50=n10=n1=0
   // n500, n100, n50, n10, n1은 각각의 동전 카운트
2. while (change ≥ 500)
   change = change-500, n500++ // 500원짜리 동전 수를 1 증가
3. while (change ≥ 100)
   change = change-100, n100++ // 100원짜리 동전 수를 1 증가
4. while (change ≥ 50)
   change = change-50, n50++ // 50원짜리 동전 수를 1 증가
5. while (change ≥ 10)
   change = change-10, n10++ // 10원짜리 동전 수를 1 증가
6. while (change ≥ 1)
   change = change-1, n1++ // 1원짜리 동전 수를 1 증가
7. return (n500+n100+n50+n10+n1) // 총 동전 수를 리턴 한다.

- 거스름돈이 200원이라면, CoinChange 알고리즘은 160원짜리 동전 1개와 10원짜리 동전 4개로서 총 5개를 리턴 한다.

CoinChange 알고리즘의 결과

최소 동전의 거스름돈

- CoinChange 알고리즘은 항상 최적의 답을 주지 못한다.

4. 작업 스케줄링

- 작업 스케줄링 (Task Scheduling) 문제는 작업의 수행 시간이 중복되지 않도록 모든 작업을 가장 적은 수의 기계에 배정하는 문제이다.
- 작업 스케줄링 문제는 학술대회에서 발표자들을 강의실에 배정하는 문제와 같다.
 발표= '작업', 강의실= '기계'

- 작업 스케줄링 문제에 주어진 문제 요소
→ 작업의 수
→ 각 작업의 시작시간과 종료시간
→ 작업의 시작시간과 종료시간은 정해져 있으므로 작업의 길이도 주어진 것이다.
- 여기서 작업의 수는 입력의 크기이므로 알고리즘을 고안하기 위해 고려되어야 하는 직접적인 요소는 아니다.
- 그렇다면, 시작시간, 종료시간, 작업 길이에 대해 다음과 같은 그리디 알고리즘들을 생각해볼 수 있다.

- 빠른 시작시간 작업 우선 (Earliest start time first) 배정
- 빠른 종료시간 작업 우선 (Earliest finish time first) 배정
- 짧은 작업 우선 (Shortest job first) 배정
- 긴 작업 우선 (Longest job first) 배정

위의 4가지 중 첫 번째 알고리즘을 제외하고 나머지 3가지는 항상 최적해를 찾지 못한다.

학습내용1 : 해 탐색 알고리즘 이란

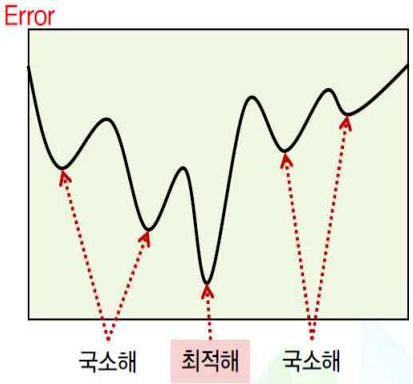
1. 해 탐색 알고리즘의 개요

문제해결은 대부분의 AI 응용에 기본이 된다. 사실상, 문제를 해결하는 능력은 종종 사람과 기계에 대한 지능의 측정 기준으로 사용된다. 주로 두 가지 유형의 문제가 있다. 첫 번째 유형은 성공이 보장된 결정적 프로시저 (deterministic procedure)를 사용함으로써 해결될 수 있다. 이 과정은 계산 (computation) 이라고 불린다. 계산에 의해 해결하는 것은 보통, 수학에서 처럼, 프로시저가 문제를 위하여 존재하는 그런 유형의 문제들에만 적용된다. 종종 이 문제들을 해결하기 위하여 사용되는 방법들을 컴퓨터가 수행할 수 있는 알고리즘으로 쉽게 바꿀 수 있다. 그러나, 계산적인 해결에 도움이 되는 실세계 문제는 거의 없기 때문에 그런 문제들은 해를 탐색함으로써 (searching for a solution) 해결하는 문제들로 구성된 두 번째 부류에 놓여야 한다. 우리는 이런 것을 NP-완전 문제(다항식 시간 알고리즘이 아직 만들어지지도 않았고, 그렇다고 해서 그러한 알고리즘이 존재하지 않는다고 증명되지도 않은 문제) 라고 앞에서 배웠다.

NP-완전 문제의 해를 탐색하기 위한 방법 중에서 가장 대표적인 알고리즘으로 백트래킹 (Backtracking)기법, 분기한정 기법, 유전자 알고리즘 , 모의 담금질 알고리즘을 통해 적절한 해를 탐색하는 것을 해 탐색 알고리즘 이라 한다.

➢탐색 공간

▪ 모든 가능한 해의 공간



학습내용2 : 백 트래킹 기법

1. 백 트래킹 기법의 개요

- 백트래킹(Backtracking) 기법은 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법으로 상태 공간 트리에서 깊이 우선 탐색 (Depth First Search) 방법으로 해를 찾는 알고리즘이다.
- 백트래킹 기법은 최적화 (optimization) 문제와 결정 (decision) 문제를 해결할 수 있다.
- 결정 문제: 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'가 답하는 문제이다.

☞ 미로 찾기
☞ 해밀토니안 사이클 (Hamiltonian Cycle) 문제
☞ 서양 장기 여왕 말(n-Queens)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.

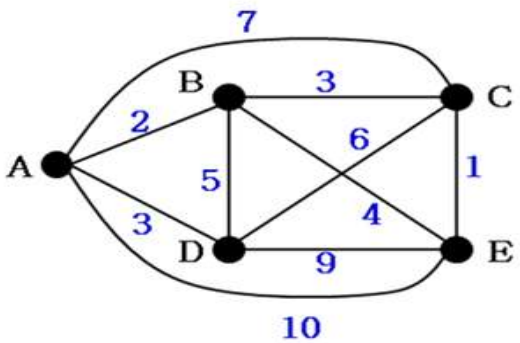
* BacktrackTSP(tour)

```
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) {
              newTour = tour + v          // 기존 tour의 뒤에  v 를추가

7.              if (newTour의 거리 < bestSolution의 거리)
8.                              BacktrackTSP(newTour)
                }
      }
```

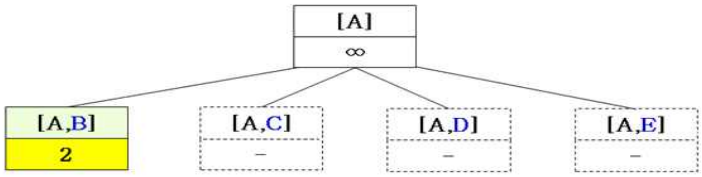
- Line 1~3: 현재 tour가 완전한 해이면서 현재까지 찾은 가장 우수한 해인 bestSolution의 거리보다 짧으면 현재 tour로 bestSolution이 갱신 bestSolution의 거리보다 같거나 긴 경우에는 갱신 없이 이전 호출했었던 곳으로 돌아감
- Line 4~8: tour가 아직 완전한 해가 아닐 때 수행
- Line 5~8의 for-루프: 현재 tour에서 확장 가능한 각 점에 대해서 루프의 내부가 수행
확장 가능한 점이란 현재 tour에 없는 점으로서 현재 tour의 가장 마지막 점과 선분으로 연결된 점을 말함
- Line 6: 현재tour를 확장 (즉, 확장 가능한 점을 기존 tour에 추가)하여 newTour를 얻음
- Line 7~8: 확장된newTour의 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)



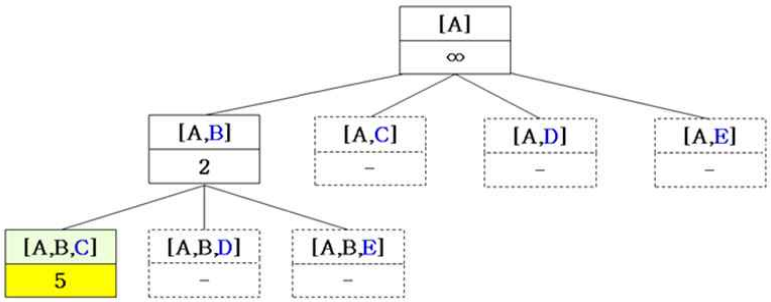
- 시작점이 A이므로, tour=[A]이고, bestSolution=([A],∞)이다.
- BacktrackTSP(tour)를 호출하여 해 탐색 시작

- 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 2이다.



- 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

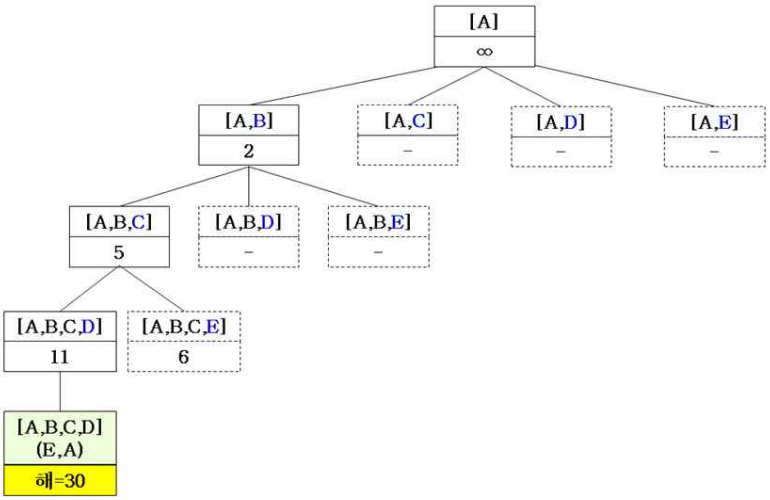
- Line 6에서 newTour=[A,B,C]가 되고, newTour의 거리는 5.
- 왜냐하면 선분 (B,C)의 가중치가 3이므로



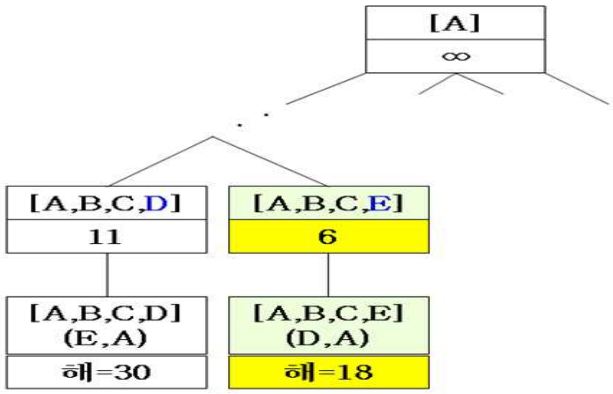
- Line 7~8: newTour의 거리 5가 bestSolution의 거리 ∞보다 짧으므로, BacktrackTSP([A,B,C])를 재귀 호출. 확장 가능한 점 D, E에 대해서는 BacktrackTSP([A,B,C]) 호출을 다 마친 후에 각각 수행

…

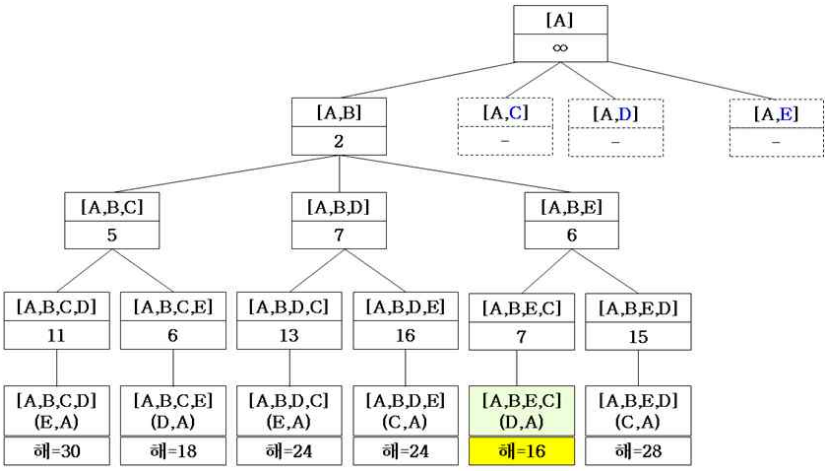
- 이와 같이 계속 탐색이 진행되면 다음과 같이 첫 번째 완전한 해를 찾는다.
이때 bestSolution=([A,B,C,D,E,A], 30)이 된다.



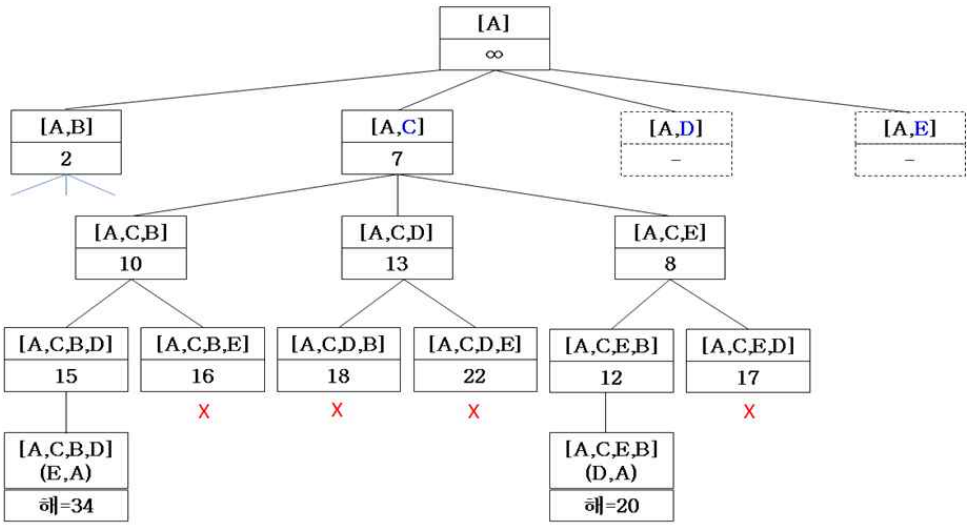
- 첫 번째 완전한 해를 찾은 후에는 다음과 같이 수행되며, 이때 더 짧은 해를 찾으므로 bestSolution=([A,B,C,E,D,A], 18)이 된다.



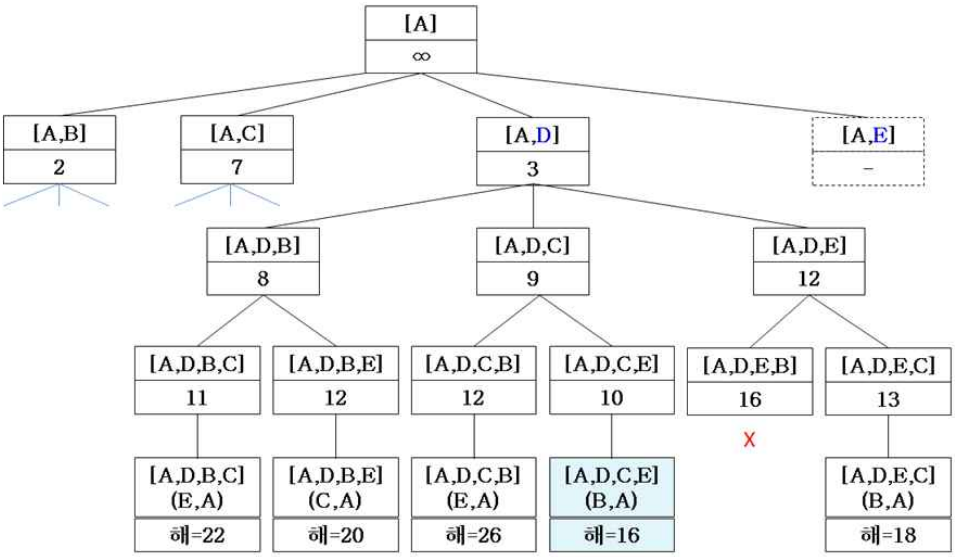
- 다음은 tour=[A,B]에 대해서 모든 수행을 마친 결과이다. bestSolution=([A,B,E,C,D,A], 16)이다



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



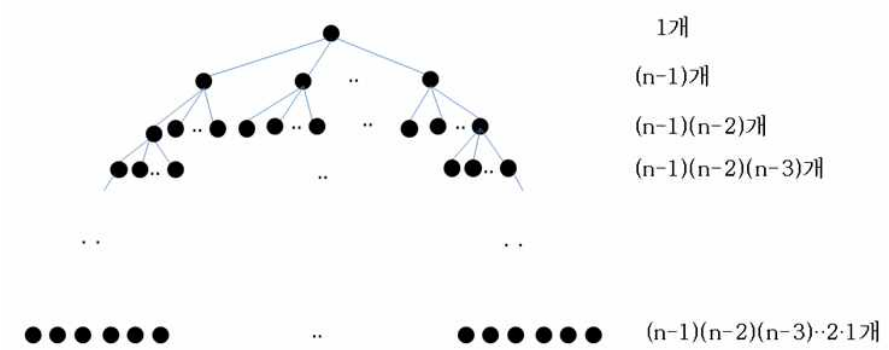
- 다음은 tour=[A,D]에 대해서 모든 수행을 마친 결과를 보여줌
- 이때 bestSolution보다 더 우수한 해는 탐색되지 않았으나 같은 거리의 해를 찾는데 이 해는 bestSolution tour의 역순
- x로 표시된 1개의 상태는 bestSolution의 거리와 같으므로 가지치기된 것임



- 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.

3. Backtracking 알고리즘의 시간 복잡도

- 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 이파리노드 수만계산해도(n-1)!
- 문제에 따라서 이진트리 형태의 상태 공간트리가 형성되기도 하는데 이때에도 최악의 경우에 2^n개의 노드를 모두 탐색해야 하므로 지수시간이 걸림
- 이는 모든 경우를 다 검사하여 해를 찾는 완결 탐색 (Exhaustive Search)의 시간복잡도와 같음
- 그러나 일반적으로 백트래킹 기법은 '가지치기'를 하므로 완결탐색 보다 훨씬 효율적임

【학습정리】

1. 해 탐색 알고리즘 이란

문제해결은 대부분의 AI 응용에 기본이 된다. 사실상, 문제를 해결하는 능력은 종종 사람과 기계에 대한 지능의 측정 기준으로 사용된다. 주로 두 가지 유형의 문제가 있다. 첫 번째 유형은 성공이 보장된 결정적 프로시저 (deterministic procedure)를 사용함으로써 해결될 수 있다. 이 과정은 계산 (computation) 이라고 불린다. 계산에 의해 해결하는 것은 보통, 수학에서 처럼, 프로시저가 문제를 위하여 존재하는 그런 유형의 문제들에만 적용된다. 종종 이 문제들을 해결하기 위하여 사용되는 방법들을 컴퓨터가 수행할 수 있는 알고리즘으로 쉽게 바꿀 수 있다. 그러나, 계산적인 해결에 도움이 되는 실세계 문제는 거의 없기 때문에 그런 문제들은 해를 탐색함으로써 (searching for a solution) 해결하는 문제들로 구성된 두 번째 부류에 놓여야 한다. 우리는 이런 것을 NP-완전 문제(다항식 시간 알고리즘이 아직 만들어지지도 않았고, 그렇다고 해서 그러한 알고리즘이 존재하지 않는다고 증명되지도 않은 문제) 라고 앞에서 배웠다.

NP-완전 문제의 해를 탐색하기 위한 방법 중에서 가장 대표적인 알고리즘으로 백트래킹 (Backtracking)기법, 분기한정 기법, 유전자 알고리즘 , 모의 담금질 알고리즘을 통해 적절한 해를 탐색하는 것을 해 탐색 알고리즘 이라 한다.

2. 백 트래킹 기법

- 백트래킹(Backtracking) 기법은 해를 찾는 도중에 '막히면' (즉, 해가 아니면) 되돌아가서 다시 해를 찾아 가는 기법으로 상태 공간 트리에서 깊이 우선 탐색 (Depth First Search) 방법으로 해를 찾는 알고리즘이다.

- 백트래킹 기법은 최적화 (optimization) 문제와 결정 (decision) 문제를 해결할 수 있다.
- 결정 문제: 문제의 조건을 만족하는 해가 존재하는지의 여부를 'yes' 또는 'no'가 답하는 문제이다.

☞ 미로 찾기
☞ 해밀토니안 사이클 (Hamiltonian Cycle) 문제
☞ 서양 장기 여왕 말(n-Queens)문제
☞ 부분 집합의 합 (Subset Sum) 문제 등

- 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.