1 그래프 이론

1 그래프

- 점과 선을 연결한 것
- 점을 꼭지점, 정점(Vertex), 노드(Node)
- 선을 변, 간선(Edge)

1 그래프

연결 그래프

- 모든 정점 사이를 연결한 그래프
- 모든 정점의 경로가 존재하는 그래프

✓ **고립 정점 : 어떤 정점도 연결되지 않은 정점**

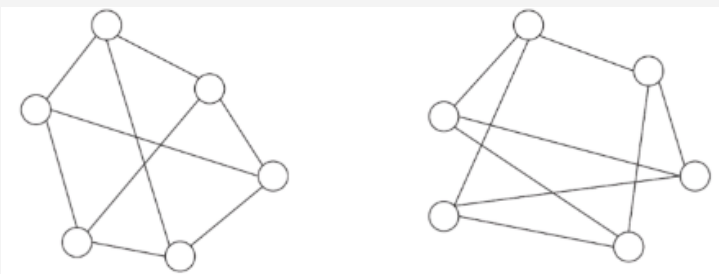
[그래프의 구성 요소]

※ 출처 : 처음 배우는 인공지능, 다다 사토시

1 그래프

동형 그래프

- 연결되어 있는 정점들은 어느 위치에 있는지 관계가 없음
- 정점을 이동해도 같은 그래프가 될 수 있음



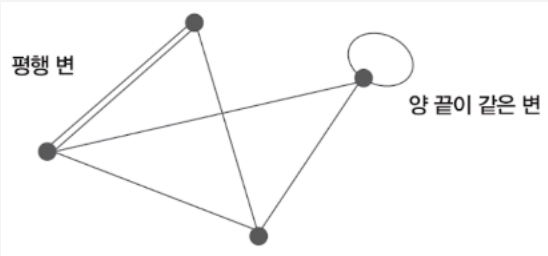
※ 출처 : 처음 배우는 인공지능, 다다 사토시

1 그래프

복잡한 그래프

- **평행 변(Parallel Edge)**
  : 그래프에서 정점 2개가 2개 이상의 변으로 연결되는 변

- **양끝이 같은 변(Self-loop)**
  : 1개의 정점에 시작과 끝이 연결된 변



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 무향 그래프와 유향 그래프

유향 그래프(Directed Graph)

- 그래프의 변에 방향이 존재하는 그래프
- 유향 비순회 그래프(Direct Acyclic Graph; DAG)
  - 임의의 정점에서 출발한 후 해당 정점에 돌아오는 경로가 하나인 그래프

2 무향 그래프와 유향 그래프

유향 그래프(Directed Graph)

- 가중 그래프(Weighted Graph)
  - 유향 그래프 중 가중치 정보가 추가된 그래프
  - 간선 가중 그래프
    - 변에 가중치를 나타낸 그래프
  - 정점 가중그래프
    - 정점에 가중치를 나타낸 그래프

2 무향 그래프와 유향 그래프

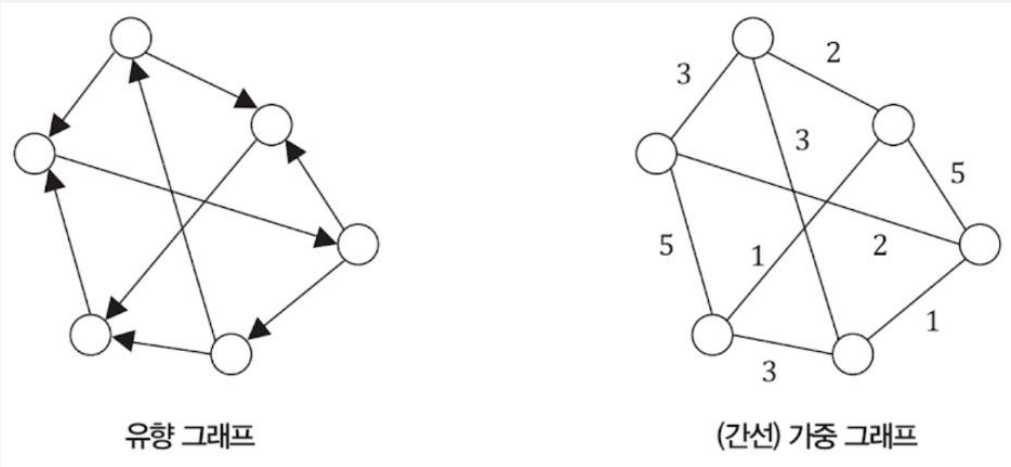
유향 그래프(Directed Graph)

- 가중 그래프(Weighted Graph)
  - 네트워크라고도 함
    - 신경망, 베이즈 네트워크 등
    - 상태 전이 다이어그램도 네트워크의 한 종류

2 무향 그래프와 유향 그래프

무향 그래프(Undirected Graph)

- 변에 방향이 존재하지 않는 그래프



유향 그래프 (간선) 가중 그래프

※ 출처 : 처음 배우는 인공지능, 다다 사토시

3 그래프의 행렬 표현

인접 행렬(Adjacency matrix)

- 정점 사이의 관계를 나타내는 행렬
- 정점 수가 n일 때 nxn 행렬로 표현
- 정점이 연결되어 있으면 1,
  연결되어 있지 않으면 0으로 표현

3 그래프의 행렬 표현

근접 행렬(Incidence matrix)

- 정점과 변의 관계를 나타내는 행렬
- 정점 수가 n, 변의 수가 m일 때 n x m 행렬로 표현
- 정점과 변이 연결되어 있으면 1,
  연결되어 있지 않으면 0으로 표현

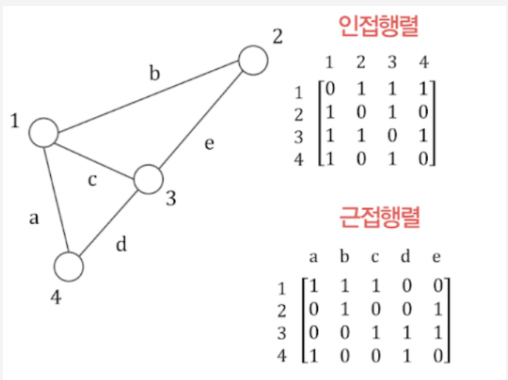
3 그래프의 행렬 표현

- 변에 가중치가 있는 그래프일 경우 인접 행렬 요소를 0과 1이 아닌 가중치 값으로 구성

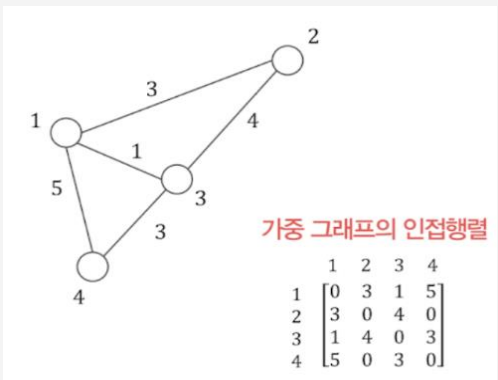
3 그래프의 행렬 표현

- 그래프를 행렬로 표현하면 그래프를 표 형식으로 변경이 쉬움
- 표 형식의 값을 가중 그래프로 변환해서 히트 맵처럼 표현 가능
- 네트워크 분석
  : 가중 그래프를 이용한 데이터 분석 방법

[인접 행렬과 근접 행렬]



[가중 그래프의 인접 행렬]

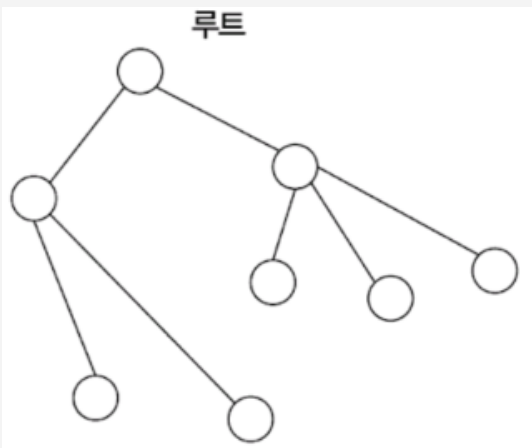


※ 출처 : 처음 배우는 인공지능, 다다 사토시

4 트리 구조 그래프

- 그래프에 있는 여러 개의 정점에서 출발점이 되는 정점으로 돌아가는 경로가 유일한 경우
- 출발점이 되는 정점이 막다른 정점인 그래프
- 루트(Root) : 트리 구조 그래프의 출발점이 되는 정점
- 의사 결정 트리
  - 통계 모델 예측을 위한 조건 분기에 사용되는 규칙으로 트리 구조 사용
- 탐색 트리
  - 상태를 나누는 수단으로 트리 구조 사용

[트리 구조 그래프 예]



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 그래프 탐색

1 그래프 탐색

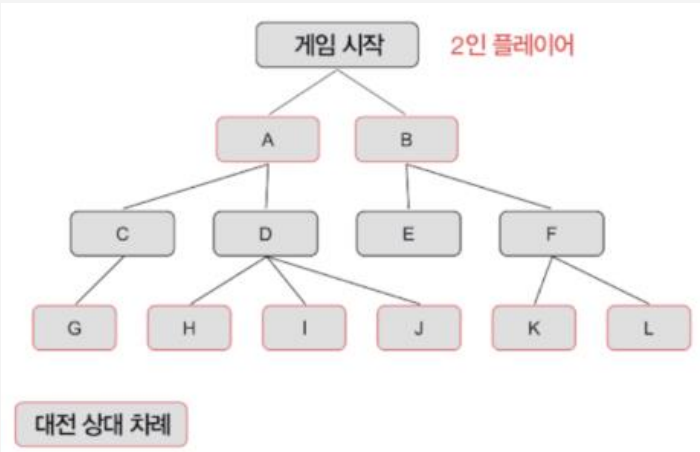
- 트리 구조와 이진 탐색 트리
- 너비 우선 탐색
- 깊이 우선 탐색
- 에이스타(A*) 알고리즘
- 동적 계획법

2 탐색 트리 구축

- 트리 구조 그래프를 출발점이 되는 정점(루트, 초기 상태)에서 다른 정점을 향해 나아가는 모습을 통해 여러 가지를 선택할 수 있는 상태 표현
- 출발점에서 종착점으로 가는 경로를 탐색하는 경우 유용
- 출발점에서 목적지를 노드로 구성한 후 분기를 이용해 상태 선택
- 예
  - 제로섬 유한 확정 완전 정보 게임 : 체스, 오셀로 등
  - 경로 탐색 : 미로 찾기, 대중교통 환승 등

2 탐색 트리 구축

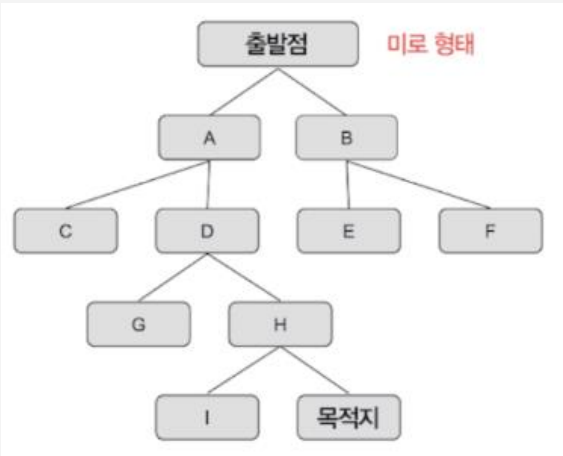
- 2인 플레이 게임 시 어느 플레이어를 선택하느냐에 따라 대전하는 상대가 달라짐



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 탐색 트리 구축

- 미로에서 출발점으로부터 어느 경로를 선택하느냐에 따라 최종 목적지에 도달 경로를 결정



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 탐색 트리 구축

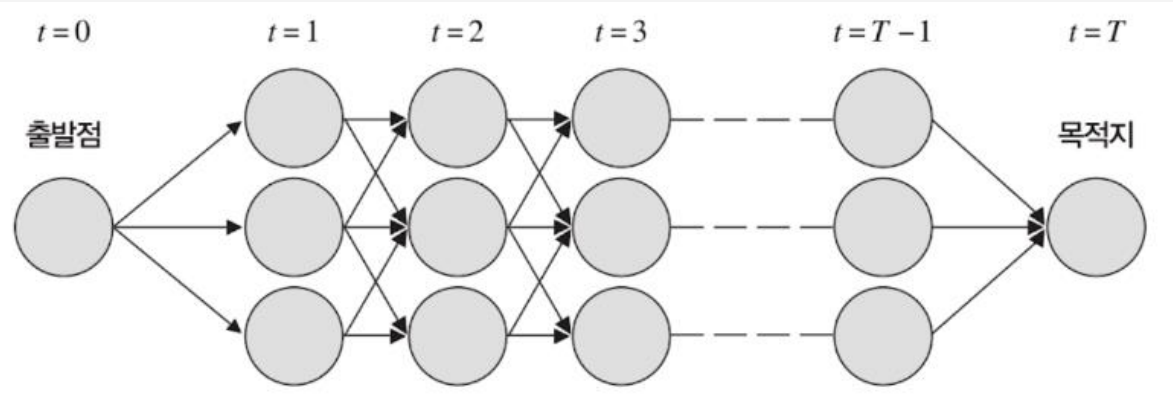
- 탐색 트리는 노드에 이익과 비용 같은 평가 값을 저장해 두고 목적지에 도달하는 최적 경로를 검색
- 평가 값 : 최적 경로 선택 시 필요한 **'상태'**
- 예(대중교통 환승)
  - 평가 값 : 최소 이동 시간, 최소 환승 횟수, 필수 경유지, 환승 요금 등
  - 비용을 평가 값으로 환산하여 최적 경로 탐색 시 활용

2 탐색 트리 구축

- 다단계 (의사) 결정 문제
  - 최적 경로 탐색 시 현재 시간 t 에 해당하는 상태에 어떤 행동을 했을 때 얻을 수 있는 이익이나 지불 비용을 계산하여 다음 시각 $t+1$의 상태를 결정하는 것
  - 위 과정을 반복 실행하여 목적지에 도달했을 때 시간 T의 이익을 최대화 또는 비용을 최소화 하는 계획 문제 해결

2 탐색 트리 구축

- 다단계 (의사) 결정 문제



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 탐색 트리 구축

- 데이터베이스 시스템에 활용
  - 이진 탐색
    - 정렬된 데이터를 반으로 나눠 저장하는 작업 반복 수행
    - 트리 구조를 만든 후 원하는 데이터를 빠르고 효율적으로 검색
    - 이러한 트리를 이진 탐색 트리라고 함

2 탐색 트리 구축

- 데이터베이스 시스템에 활용
  - 실제 데이터베이스 시스템은
    이진 탐색 트리보다 유연한 B 트리 등을 사용
  - **온톨로지 트리**
    : 데이터 간의 관계를 나타내는 요소를 추가
  - **시멘틱 네트워크**
    : 네트워크 기반의 지식 표현 방법

2 탐색 트리 구축

이진 탐색 트리

- 이진 탐색과 연결 리스트를 결합한 자료구조의 일종
- 이진 탐색의 효율적인 탐색 능력은 유지
- 빈번한 자료 입력과 삭제를 가능하게 고안됨

2 탐색 트리 구축

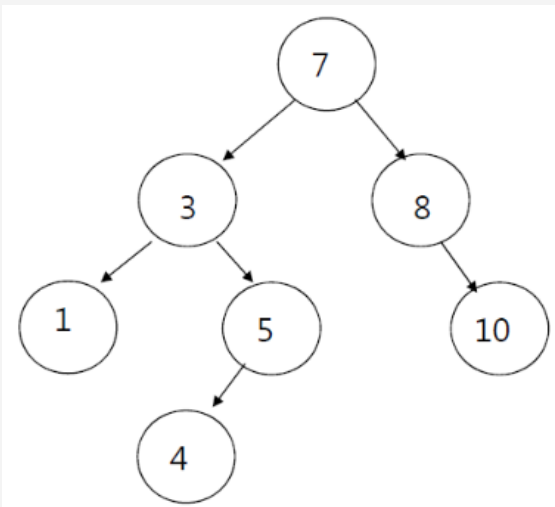
이진 탐색 트리

- 이진 탐색 트리 특징
  - 완전 이진 트리
  - 각 노드의 왼쪽 서브 트리에는 해당 노드보다 작은 값들로 구성된 노드
  - 각 노드의 오른쪽 서브 트리에는 해당 노드보다 큰 값들로 구성된 노드
  - 중복된 노드가 허용되지 않음
  - 왼쪽/오른쪽 서브 트리 또한 이진 탐색 트리
- 중위 순회 방식을 사용

2 탐색 트리 구축

이진 탐색 트리

- 이진 탐색 트리 예
  - 탐색
    - 10을 탐색한다고 가정
    - 10은 7보다 크므로 오른쪽 노드로 이동
    - 10은 8보다 크므로 오른쪽 노드로 이동
    - 10을 탐색함



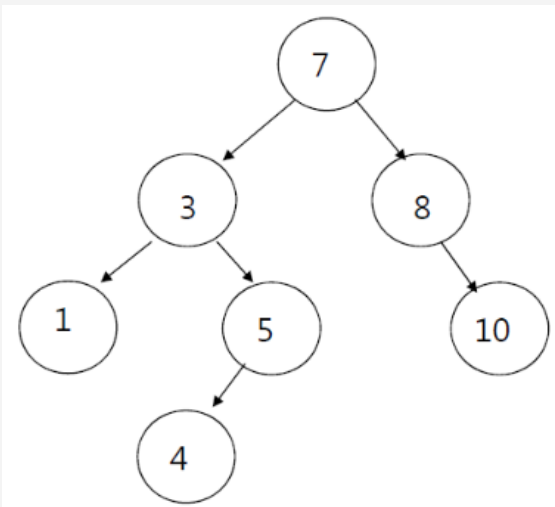
[이진 탐색 트리 예]

※ 출처 : https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst

2 탐색 트리 구축

이진 탐색 트리

- 이진 탐색 트리 예
  - 삽입
    - 4를 삽입한다고 가정
      (그림은 삽입 후 결과임)
    - 4는 7보다 작으므로 왼쪽 노드로 이동
    - 4는 3보다 크므로 오른쪽 노드로 이동
    - 4는 5보다 작으므로 왼쪽 노드로
      이동하여 삽입



[이진 탐색 트리 예]

※ 출처 : https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst

2 탐색 트리 구축

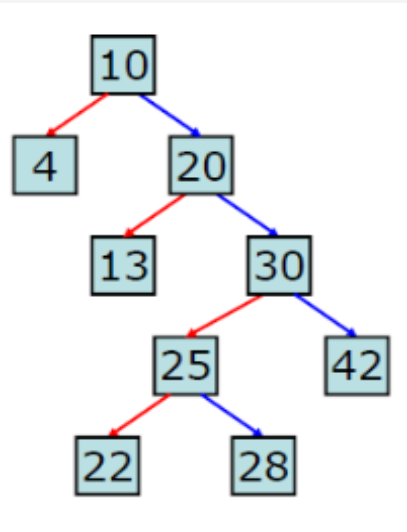
이진 탐색 트리

- 이진 탐색 트리 예
  - 삭제
    - 탐색과 마찬 가지로 과정으로 삭제할 노드를 검색하여 삭제
    - 말단 노드인 경우는 쉽게 삭제 가능
    - 하나의 자식 노드를 갖는 경우는 노드 삭제 후 자식 노드를 부모 노드에 연결
    - 두 개의 자식 노드를 갖는 경우는 두 자식 노드 중 어느 노드를 부모 노드와 연결할 것인지에 따라 다른 모양이 나타날 수 있음

2 탐색 트리 구축

이진 탐색 트리

- 이진 탐색 트리 예
  - 삭제
    - 그림에서 30을 삭제할 경우 노드 25를 노드 20에 연결할 것인지 노드 42를 노드 20에 연결할 것인지에 따라 모양이 다르게 나타남



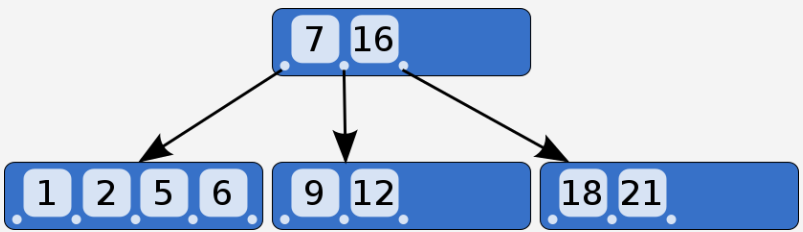
[이진 탐색 트리 삭제 예]

※ 출처 : https://ratsgo.github.io/data%20structure&algorithm/2017/10/22/bst/

2 탐색 트리 구축

B 트리

- 데이터베이스와 파일 시스템에서 널리 사용되는 트리 자료구조의 일종
- 이진 트리를 확장해 하나의 노드가 가질 수 있는 자식 노드의 최대 숫자가 2보다 큰 트리 구조
- 방대한 양의 저장된 자료 검색 시 검색어와 자료의 일대 일 비교는 비효율적
- 자료를 정렬된 상태로 보관



※ 출처 : https://ko.wikipedia.org/wiki/

2 탐색 트리 구축

온톨로지 트리

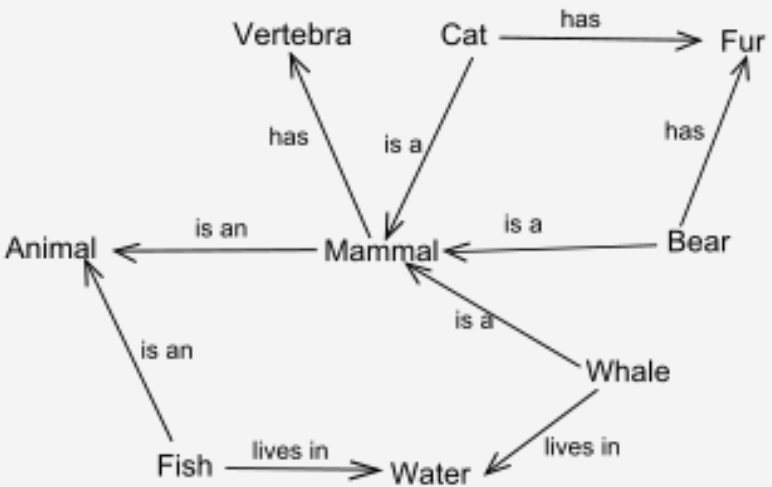
- 의미 네트워크에 메타 데이터를 추가한 데이터 작성법
- 지식 표현과 활용을 위한 인공지능 분야에 처음 적용
- 에이전트의 상호작용을 통해 의미 있는
  문제 해결을 위해 지식 기반 필요성이 대두하여
  개념 계층도 이용

2 탐색 트리 구축

시멘틱 네트워크

- 네트워크 기반의 지식 표현 방법
- 객체, 개념, 사건들을 표현하는 노드의 집합과 노드 사이의 관계를 표현하며 연결하는 아크의 집합으로 구성
- 장점
  - 복잡한 분류나 인과관계를 갖는 추론에 자연스러운 표현이 가능
- 단점
  - 너무 복잡하여 다루기 힘들 수 있음

[시멘틱 네트워크 예]



※ 출처 : https://ko.wikipedia.org/wiki/

3 탐색 트리 추적 방법

- 막힌 지점이 있는 미로나 탐색 트리의 경로 탐색 시
  최소 단계(회수)에 목적지에 도달하는 것이 목표인 경우

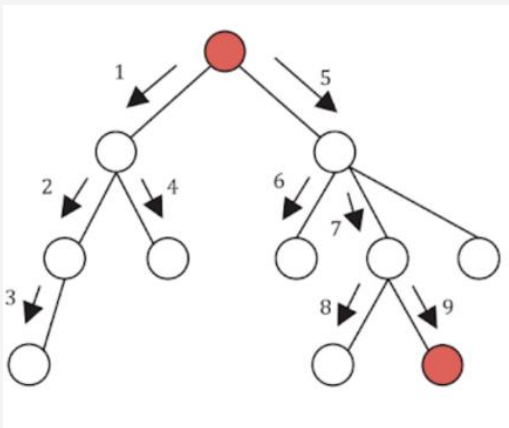
3 탐색 트리 추적 방법

- 탐색 순서에 따른 구분
  - 깊이 우선 탐색(Depth First Search, DFS)
  - 너비 우선 탐색(Breath First Search, BFS)
  - 도달하는 목표 지점까지의 경로를 미리 파악할 수 있으면 어느 방법이 더 효율적인지 명확하지만 그런 경우는 없으므로 두 방법 모두 장단점을 가질 수 있음
  - 목적지는 해당 노드의 상태 평가 값과 이를 계산하는 평가 함수로 결정
  - 상태 평가 값의 예
    : 승부의 포석, 막다른 노드나 목적지, 점수 등

3 탐색 트리 추적 방법

깊이 우선 탐색(Depth First Search, DFS)

- 루트 노드에 연결된 경로 중 하나를 선택해 막다른 노드에 도착할 때까지 일단 탐색
- 다시 바로 앞 노드로 이동하여 다음 막다른 노드에 도착할 때까지 탐색 반복

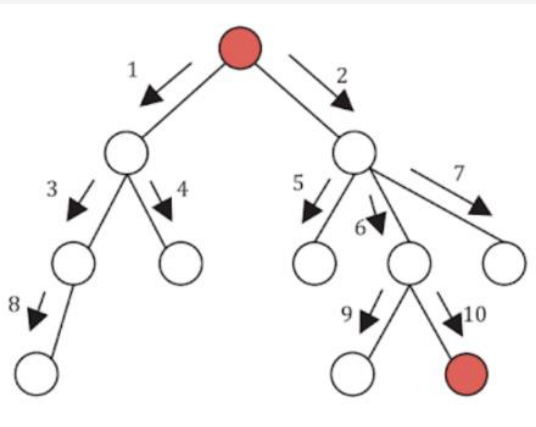


※ 출처 : 처음 배우는 인공지능, 다다 사토시

3 탐색 트리 추적 방법

너비 우선 탐색(Breath First Search, BFS)

- 루트 노드와 연결된 노드를 모두 탐색 후 바로 다음 깊이의 노드들을 전부 탐색하는 과정을 반복



※ 출처 : 처음 배우는 인공지능, 다다 사토시

3 탐색 트리 추적 방법

- 탐색 트리의 탐색에 필요한 목록
  ① 오픈 리스트(Open List)
  : 탐색 대상 노드와 연결된 주변 노드를 포함하는 탐색 노드의 목록

  ② 클로즈드 리스트(Closed List)
  : 탐색을 종료한 노드의 목록

2 탐색 트리 추적 방법

LIFO(깊이 우선 탐색 시)

- 탐색할 노드를 오픈 리스트의 맨 위에 추가해 첫 번째 노드부터 차례대로 탐색
- 마지막으로 목록에 넣은 노드의 탐색 결과가 먼저 나옴
- 스택(Stack) 구조 사용

3 탐색 트리 추적 방법

FIFO(너비 우선 탐색 시)

- 탐색할 노드를 오픈 리스트의 마지막에 추가
- 먼저 목록에 넣은 노드의 탐색 결과가 먼저 나옴
- 큐(Queue) 구조 사용

3 탐색 방법

1 효율적인 탐색 방법

- 깊이 우선 탐색 및 너비 우선 탐색은 탐색할 노드 순서를 결정하는 것
- 효율적인 탐색 방법을 고려하여 처리 시간 단축 필요

1 효율적인 탐색 방법

비용에 따른 탐색 방법

- 사전 지식이나 경험(휴리스틱) 을 이용하여
  처리 시간 단축

- 비용의 종류
  - 초기 상태
    : 상태 s 의 최적 경로 이동에 드는 비용의 총합 $\boldsymbol{g(s)}$
  - 상태 s
    : 목표하는 최적 경로 이동에 드는 비용의 총합 $\boldsymbol{h(s)}$
  - 상태 s를 거치는 초기 상태
    : 목표의 최적 경로 이동에 드는 비용의 총합
    $\boldsymbol{f(s) = g(s) + h(s)}$

1 효율적인 탐색 방법

- 최적 탐색
  - $\hat{g}(s)$: 오픈 리스트에 있는 노드 탐색 전 누적 비용의 예측값
  - $\hat{g}(s)$를 최소화 하는 노드를 탐색하는 탐색 방법

- 최선 우선 탐색
  - $\hat{h}(s)$: 비용의 예측 평가 값
  - $\hat{h}(s)$를 최소화하도록 노드를 탐색하는 방법

- 스택이나 큐에 포함되는 노드의 순서를 변경하는 원리

1 효율적인 탐색 방법

- 최적 탐색은 탐색량이 많아지는 단점을 포함
- 최선 우선 탐색은 잘못된 결과가 나올 수 있는 단점 포함

1 효율적인 탐색 방법

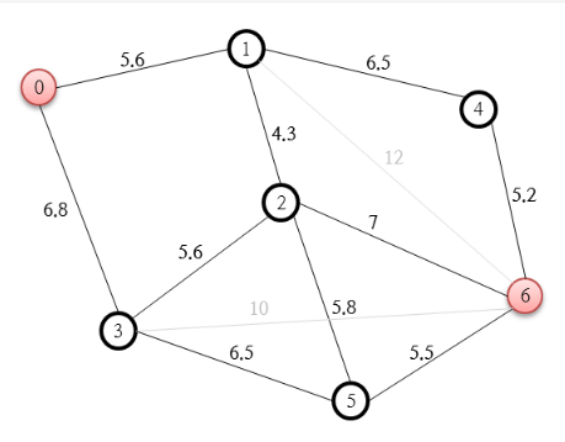
에이스타(A*) 알고리즘

- $\hat{f}(s)$: $\hat{g}(s)$과 $\hat{h}(s)$ 모두를 이용한 예측 값
- $\hat{f}(s)$를 최소화 하는 탐색 방법
- 상태 s 의 노드에 연결된 상태인 s' 의 노드 예측값 $\hat{f}(s')$
- 클로즈드 리스트에 포함되어 있을 때,
  $\hat{f}(s)$ 쪽이 작으면, s' 쪽을 클로즈드 리스트에서
  오픈 리스트로 되돌림

1 효율적인 탐색 방법

에이스타(A*) 알고리즘

예



O =

Node ID	1	3
F Score	17.6	16.8
G Score	5.6	6.8
H Score	12	10
Parent Node	0	0

C =

Node ID	0
F Score	0
G Score	0
H Score	0
Parent Node	-

※ 출처 : http://www.gisdeveloper.co.kr/?p=3897

1 효율적인 탐색 방법

게임 트리를 전략에 이용하는 방법

- 게임 트리
  : 턴마다 자신과 상대를 통해 노드를 구성
- 게임 트리의 끝 지점
  : 해당 시점의 상태(자신의 유리, 불리를 나타내는 점수) 보유
- 자신의 턴에서는 자신이 유리하다고 가정하고
  상대 턴에서는 자신이 불리하다고 가정하고
  전략을 세움

1 효율적인 탐색 방법

게임 트리를 전략에 이용하는 방법

- 탐색 노드를 줄여 시간을 줄임
  - 미니맥스(Mini-max) 원리
  - 알파베타 가지치기(Alpha-beta Pruning)

1 효율적인 탐색 방법

미니 맥스 원리(최소극대화 원리)

- 자신의 턴에 점수를 최대화하고,
  상대의 턴에 점수를 최소화 하도록 노드 선택
- 다음 선택 변을 줄여 탐색 시간을 줄이는
  최적 탐색 실행 고려
- 두 명의 참가자가 존재하는 제로섬 게임 이론으로부터
  형성
- 복잡한 게임 및 불확실성이 존재할 때의
  일반적 의사 결정에까지 적용 확장

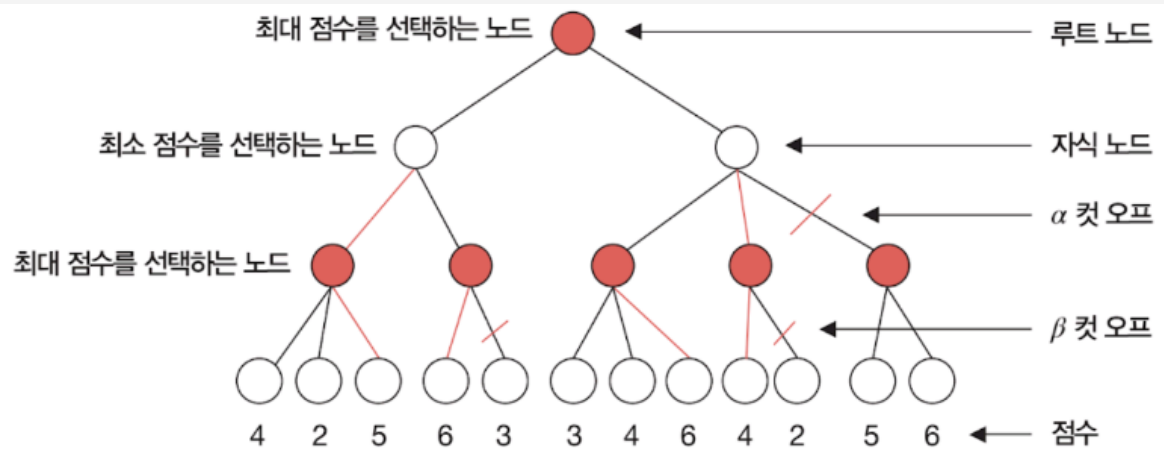
1 효율적인 탐색 방법

알파베타 가지치기

- 노드를 왼쪽부터 탐색해 변을 잘라 내는 작업 수행
- β컷 오프
  - 최대 점수를 선택하면서 이미 저장한 점수보다 작은 점수의 노드가 나오면 해당 노드를 탐색 대상에서 제외
- α컷 오프
  - 점수가 최소인 것을 선택하면서 이미 저장한 점수보다 큰 점수의 노드가 나오면 앞에 연결된 노드를 검색 대상에서 제외

1 효율적인 탐색 방법

알파베타 가지치기



※ 출처 : 처음 배우는 인공지능, 다다 사토시

1 효율적인 탐색 방법

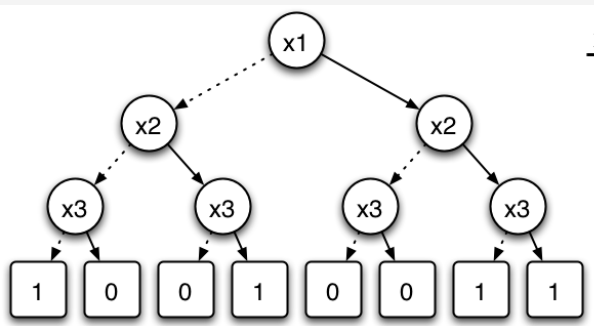
- 바둑이나 장기 등 탐색 공간, 즉 탐색할 노드 수가 방대한 경우 메모리 용량과 탐색 시간이 부족

1 효율적인 탐색 방법

- 효율적인 탐색 대상 선택
  - 몬테카를로 트리 탐색
    (Monte Carlo Tree Search, MCTS)
    - 의사 결정을 위한 체험적 탐색 알고리즘
    - 게임 주로 적용

1 효율적인 탐색 방법

- 효율적인 탐색 대상 선택
  - 이진 의사 결정 다이어그램 (Binary Decision Diagram, BDD)
    - 불리안 함수를 표현하기 위해 사용되는 자료구조
    - 관계 집합의 압축 표현



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

※ 출처 : https://en.wikipedia.org/wiki/

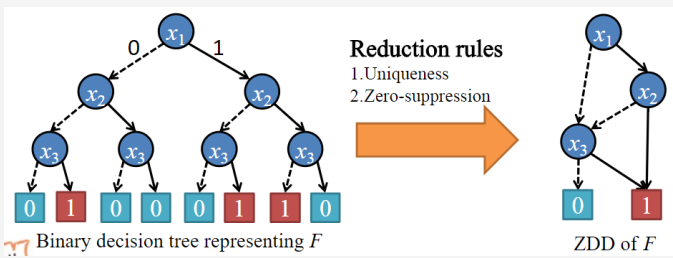
1 효율적인 탐색 방법

- 효율적인 탐색 대상 선택
  - ZDD(Zero-suppressed Decision Diagram)
    - 이진 의사 결정 다이어그램의 한 종류
    - 불리안 함수의 함축적 표현

[불리안 함수의 함축적 표현 예]

$$F = (x_1 \wedge x_2 \wedge \overline{x_3} \vee (\overline{x_2} \wedge x_3) \leftrightarrow \{\{x_1, x_2\}, \{x_1, x_3\}\ \{x_3\}\}$$

[ZDD의 적용 예]



※ 출처 : http://slideplayer.com/slide/7896999/

2 동적 계획법

- 경로 탐색에서 체크포인트(경유지)를 통과할 필요가 있거나 탐색 필요
- 비용이 있을 때 노드에서 노드를 이동하는 모습을 '시계열 기준 상태 변화'로 가정
- 어떤 상태에서 다음 상태로 전환하는 것은 비용을 고려하면서 결정을 수 차례 반복하는 다단계 결정 문제로 취급
- 다단계 결정 문제로 다뤄서 얻는 경로의 평가 함수 J 의 계산 결과를 극대화 하여 경로 탐색 목표를 결정

2 동적 계획법

- 다단계 결정 문제의 평가 함수 식
  $J = (s_1, s_2, s_3, \cdots, s_n)$
- 시간 $t = 1, \ldots, T$ 일 때, s_t라는 상태 패턴을
  N 으로 설정하면 얻을 수 있는 상태가 N^T
- N=3이고 단계 수가 T = 10인 경우
  N^T는 약 6만 여개, T = 20일 경우 약 35억개
- 모든 경로를 열거해 평가 시 기하급수적인
  계산량($O(NT)$)이 발생하여 현실적으로 불가능

2 동적 계획법

- 다단계 결정 문제의 평가 함수 식 J를 2개의 변수 함수의 합으로 변경

[다단계 결정 문제의 평가 함수를 두 가지 상태로 나타냈을 때의 식]

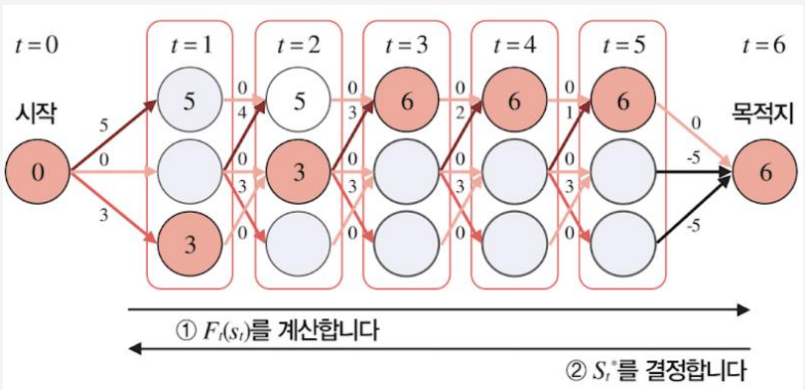
$$J = (s_1, s_2, s_3, \cdots, s_n) = \sum_{t=2}^{T} h_t(s_{t-1}, s_t)$$

- 계산량이 $O(N^2T)$까지 감소
- 이런 처리 방식을 동적 계획법, 또는 동적 프로그래밍 (Dynamic Programming)이라 함

2 동적 계획법

- 경로 선택에 따라 t = 1에서 t= T까지 순서대로 $F_t(s_t)$를 계산
- 마지막 단계 T에서는 $F_T(s_T)$까지 구한 시점의 최대값을 J^*로 설정한 다음 반대 순서로$F_T(s^*_T)$를 얻는 s^*_T 를 계산
- 이를 통해 최상 경로 (s^*_1 , s^*_2, .. s^*_T)와 최고점수 J^*를 구함

[경로 선택 예]



※ 출처 : 처음 배우는 인공지능, 다다 사토시

2 동적 계획법

- $F_t(s_t)$의 최고 점수를 구하는 식
  $F_t(s_t) = \max_{s_{t-1}}[F_{t-1}(s_{t-1}) + h_t(s_{t-1}, s_t)]$
- 메모리화
  : s_t에는 3개의 상태를 저장하고 이를 메모리에 저장하는 작업

2 동적 계획법

- 원본 비교에 응용
  - 생물정보학
    - 2개의 염기 서열과 아미노산 서열을 효율적으로 비교하여 상동성을 계산하는데 동적 계획법 사용
    - 레벤슈타인 거리를 이용하여 점수나 페널티 계산
    - 종과 종 사이의 유사성이 높은 아미노산의 상대 빈도나 치환 확률에서 구한 확률 비율의 대수행렬을 이용
- 경로가 증가하면 메모리 양이 증가하므로 대규모 병렬 처리를 이용하여 효율적 계산을 할 수 있도록 발전하고 있음