2.주차

교재 3장: 탐색을 통한 문제 해결

학습 목표:

* 상태 공간 표현을 이해하고, 상태 공간 탐색 문제를 형식화한 탐색 방법들을 학습한다.
* 탐색 알고리즘을 이해하고 적용할 수 있으며 각 알고리즘의 코딩을 할 수 있다.
  + 탐욕적 검색
  + DFS
  + BFS
  + 균일 비용 검색
  + 반복 심화 검색
  + 양방향 검색
* 알고리즘들의 특성을 이해하고 분석할 수 있다.
  + 시간 복잡성
  + 공간 복잡성
  + 종료
  + 최적
* 주어진 문제에 탐색 알고리즘을 적용할 수 있다.
* 탐색 중에 암시적 상태공간이 전개되는 방법을 이해한다.
* 상태가 특징에 의하여 표현되는 방법을 이해한다.
* 탐색을 위한 전략을 학습한다.
  + 너비 우선 탐색
  + 깊이 우선 탐색
  + 반복 심화 탐색
  + 양방향 탐색
* 또한, 이장의 학습은 다음을 포함한다.
  + 알고리즘
  + 시간과 공간의 복잡성
  + 특정한 전략의 선택 능력
* 학습 효과
  + 주어진 문제를 분석하고 가장 적절한 검색 전략을 선택할 수 있다.
  + 문제를 해결하기 위하여 선택된 전략을 적용하여 문제를 해결할 수 있다.

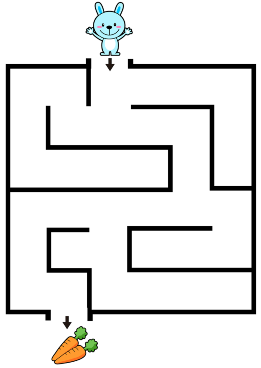
2.1 상태공간 탐색

* 허용된 문제 공간, 연산자, 초기상태와 목표 상태로 문제의 형식화한다.
* 상태는 세계에서 관심 대상의 모든 속성들의 값을 지정함으로 정의된다.
* 운영자는 다른 하나의 상태로 변화; 이는 오퍼레이터에 의해 변경되어 특정 속성의 속성 값이 운전자의 적용 이전 인 전제 및 효과의 세트를 갖는다
* 시작 위치초기 상태입니다
* 목표 상태는 해의 부분의 설명이다.

2.1.1 목표 지향 에이전트(Goal Directed Agent)

우리는 이전에 지능형 에이전트에 관하여 학습하였으며, 오늘 지능형 에이전트의 한 유형인 목표 지향 에이전트에 관하여 학습한다.

* 사전 지식 및 용어 정리
* 목표를 기반으로 하는 에이전트이다.
* 현실세계의 많은 문제들이 상태들의 집합과 상태들의 변환하는 규칙들의 집합으로 설명이 가능하다.
* 각각의 상태는 에이전트가 속한 환경의 추상적 표현이며 에이전트의 설정을 의미하는 추상화이다.
* 초기 상태: 에이전트의 초기 설정을 나타낸다.
* 에이전트는 행동/연산자에 의하여 한 상태로부터 다른 상태로 옮겨 간다.
* 상태는 다수의 후속 상태로 구성될 수 있다.
* 계획(plan)은 행동들의 순차열(행동렬)이다.
* 목표는 문제의 세계에서 요구되는 상태들의 집합으로 표현된다.
* 경우에 따라서 목표 상태는 목표가 만족하여야만 하는 목표 시험의 통과 여부에 따라 정의되기도 한다.
* 목표 지향 에이전트의 예
  + 15 퍼즐: 이 퍼즐의 목표는 좌측 상단의 행부터 1부터 15까지 순차적으로 배열하는 것이다.
  + 미로 찾기: 에이전트의 목표는 초기 위치에서 최종 위치로 이동하는 경로를 찾는 게임이며, 목표 지점에 도달할 때까지 에이전트는 행동열을 선택하여야만 한다.

2.1.2 상태 공간 탐색의 표현

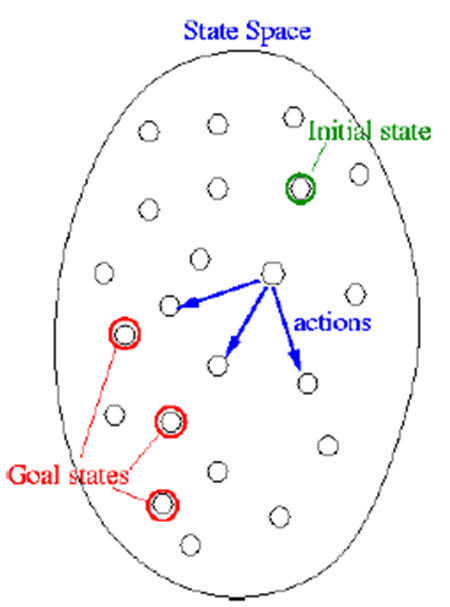
* 초기 상태는 에이전트의 시작 상태에 대한 설정이다.
* 행동 또는 연산자는 에이전트가 한 상태에서 다른 상태(후속 상태)로 이동시키는 주체이다.
* 상태는 다수의 후속 상태들을 갖을 수도 있다.
* 계획은 순차적 행동열이다
* 계획의 비용은 경로 비용이며 경로 비용은 항상 양(+)의 값을 갖는다.
* 일반적으로 경로 비용은 경로 상의 각 상태 변환에 소요되는 비용의 합이다.
* 탐색의 개념

**문제의 형식화**는 초기 상태에서 목표 상태로 이동하는 경로인 상태열의 집합 중에서 문제와 관련된 상태열을 선택하는 것을 의미한다.

**탐색**은 초기 상태부터 가능성 있는 연사자들을 적용하는 과정으로 최종 목표에 도달하는 상태열을 찾는 것이다.

2.1.3 탐색문제

* S: 상태들의 전체 집합
* s0: 초기 상태
* A: S→S는 연사자즐의 집합이다.
* G는 최종(목표) 상태의 집합이다.(단, G⊆S)



* 탐색 문제는 초기 상태에서 목표 g∈G로 에이전트를 변환시키는 행동열을 찾는 것으로 정의한다.
* 탐색 문제는 4-변수의 집합 {S, s0, A, G}로 표현된다.

여기서 S는 상태들의 집합, s0∈S는 초기 상태, A : S→ S는 상태 변환 유발 연산자열 또는 행동열, G는 목표(상태들의 집합)이며 G⊆S이다.

* 행동의 순차열은 초기 상태에서 목표 상태까지의 경로이며 솔루션 계획 이라고도 한다.
* 계획 P는 행동들의 순차열이다. 즉,

P={a0, a1,∙∙∙, aN}

* 계획 P는 상태 변환을 유발한다. 즉,

{s0, s1,∙∙∙, sN+1∈G}

* 상태 s의 순차열은 경로이며 경로 비용은 항상 양의 수이다. 일반적으로 경로비용은 각 행동의 비용의 합이다 .
  + 탐색 문제의 표현

탐색 문제는 방향성 그래프로 표현된다.

* 상태들은 절점(node)들로 표시 된다.
* 허용되는 행동은 원호로 표시된다.
* 탐색과정

일반 검색 과정은 다음과 같다.

Do until a solution is found or the state space is exhausted.

1. Check the current state

2. Execute allowable actions to find the successor states.

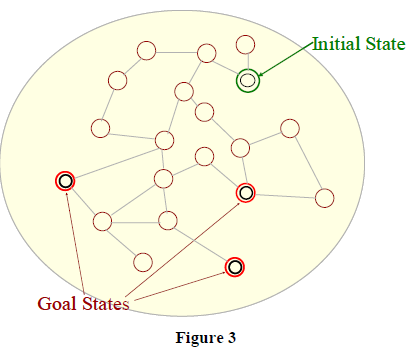
3. Pick one of the new states.

4. Check if the new state is a solution state if it is not, the new state becomes the current state and the process is

* 허용되는 행동은 원호로 표시된다.

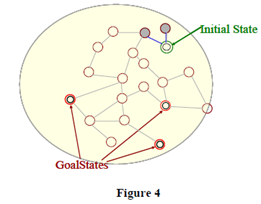
2.1.4 탐색 문제 예제

2.1.4.1 검색 과정

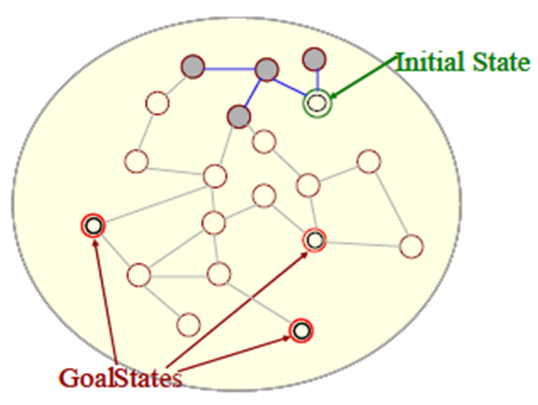
검색 과정을 설명하기 위하여 아래와 같은 예를 사용한다.

s0는 초기상태(녹색으로 표시)는 2개의 후보 후속 상태가 있다.

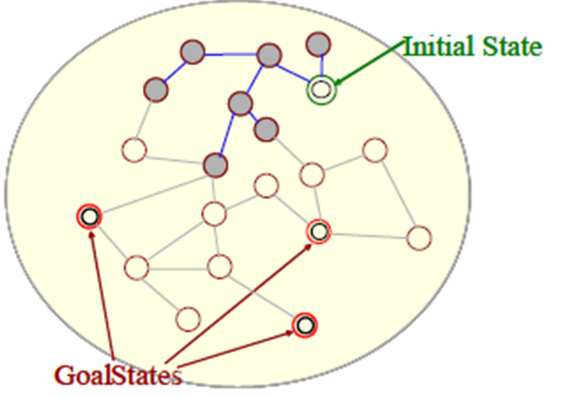
목표 상태들은 3개(적색으로 표시)



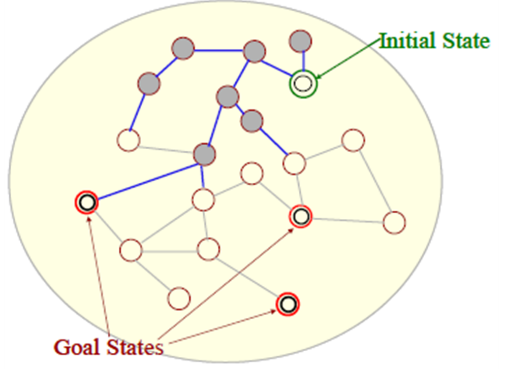
초기 상태에서 2개의 후계자가 생성된다.



각각의 후계자에서 다시 후계자를 생성한다.



모든 후계자(상태)의 후계자를 생성한다.



목표 상태를 찾았다.

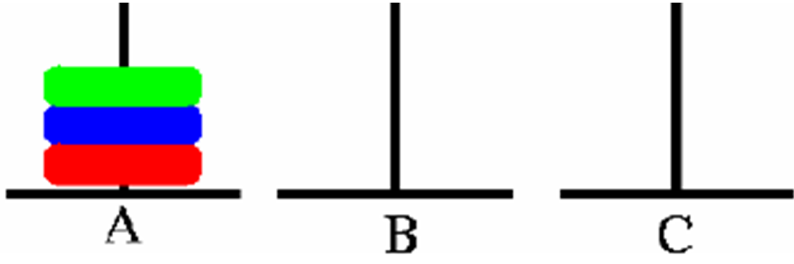
위의 예는 주어진 초기 상태에서 시작하여 후계자의 생성을 수행하여 목표 상태에 도달하는 경로를 탐색하는 과정을 보여 준다. 여기서 회색 노드는 탐색 트리를 나타내며 탐색 트리는 한번에 하나의 노드로 확장되며, 탐색 트리가 확장되는 순서는 탐색 전략에 의해 정해진다.

2.1.4.2 하노이 타워(Pegs and Disks)

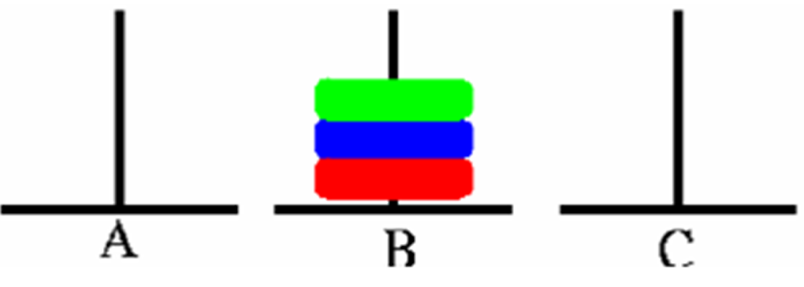
상태 공간 탐색의 또 하나의 예인 하노이 타워이다. 이 게임은 3개의 기둥과 3개의 원반으로 구성되어 있다.

게임 방법(연산자): 각 기둥의 맨 위에 있는 원반 하나만 임의의 기둥으로 이동이 가능하다.

초기상태: 원반이 아래로부터 적, 청, 녹 색 순으로 A기둥에 쌓여 있다.

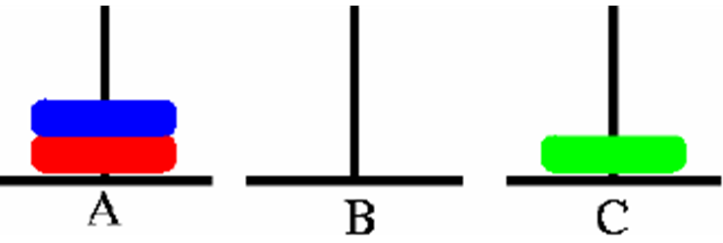


목표상태: 원반이 아래로부터 적, 청, 녹 색 순으로 B기둥에 쌓여 있다.

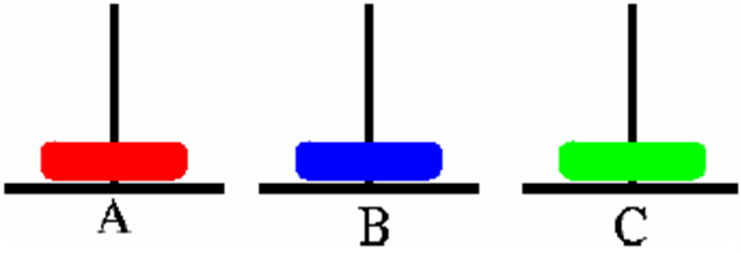


* 탐색 과정
* 초기 상태에서 연산자 법칙을 적용한 후보 상태를 탐색한다.

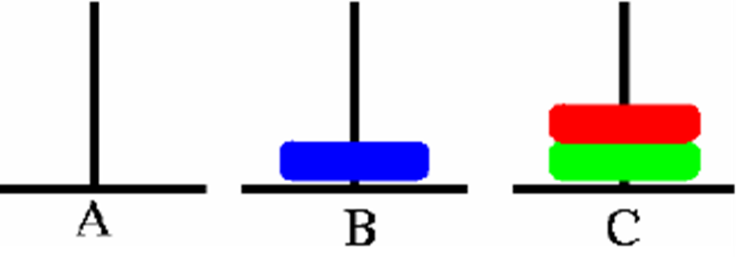
1 단계: A → C로 이동



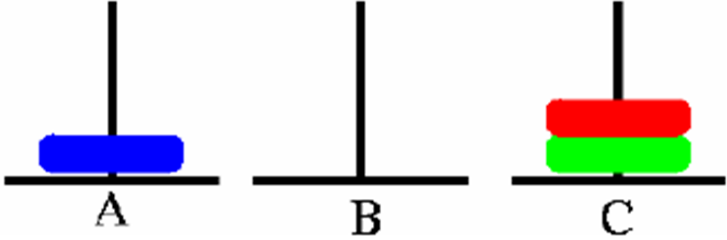
2 단계: A → B를 이동



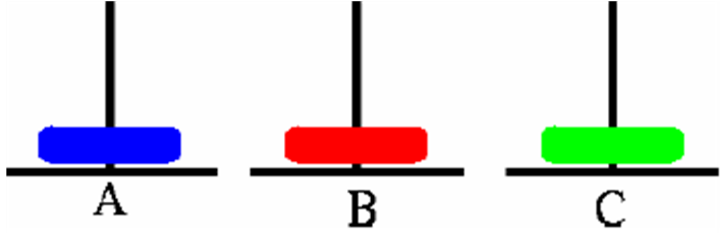
3 단계: A → C로 이동



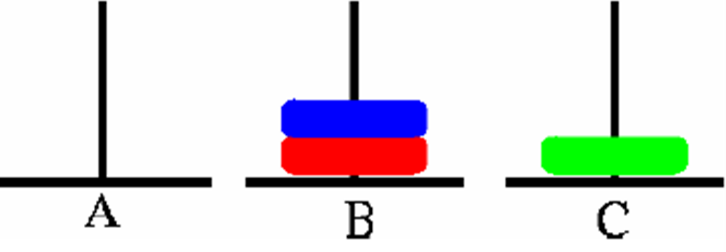
4 단계: 이동 B → A



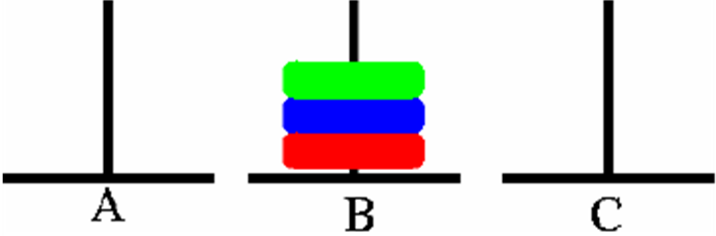
5 단계: C → B로 이동



6 단계: A → B를 이동

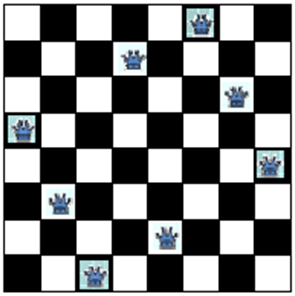


7 단계: C → B로 이동

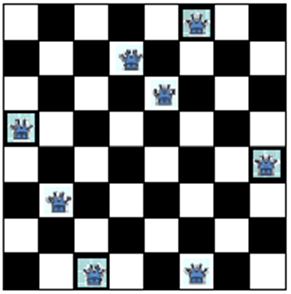


2.1.4.3 8-여왕(Queen)

8-여왕 문제는 N-여왕의 문제로 일반화 할 수 있다. 문제는 두 개의 왕비가 동일한 행, 열 또는 대각선에 있으면 공격을 받을 수 있기 때문에 서로 공격이 불가능 하도록 체스 판에 8개의 여왕을 배치하는 것입니다. 문제의 한 해결책을 다음과 같다.



다음은 여왕의 일부가 서로를 공격하기 때문에 올바른 해결책이 아니다.



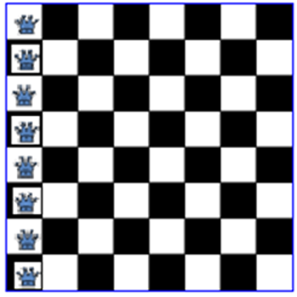
이를 공간 탐색 문제로 형식화하기로 한다. 문제의 형식화는 상태의 표현, 초기 상태표현의 선택, 행동의 서술, 후속 상태 등을 결정하는 것들이 포함된다. 탐색 문제의 형식화의 방법들을 N-여왕 문제로 설명하기로 한다.

* N-여왕 문제 형식화 1
* 상태: 서양 장기판에 여왕의 0에서 8까지 임의 배치의 모든 구성
* 초기 상태: 여왕이 배치 안된 서양 장기판 (8X8)
* 후속 함수: 한 개의 여왕을 64칸 중 임의의 칸에 배치
* 목표 시험: 8여왕이 배치되었으나, 어느 여왕도 공격할 수 없는 형세

초기 단계(상태)에는 64칸의 후보가 있다. 다음 단계는 각 상태마다 63개의 후보 칸이 있다. 여왕의 추가 배치할 때마다 후보 칸이 감소한다. 또한, 제약 조건은 후보 칸들 중에서 여왕들이 서로 공격을 할 수 있는 후보 칸을 배제하는 것이다. 따라서 새로운 여왕을 추가할 때마다 추가된 여왕이 공격 받지 않도록 이전에 배치된 모든 여왕들의 위치를 점검해야만 한다. 해결책은 8단계(depth of 8)에서 구할 수 있다.

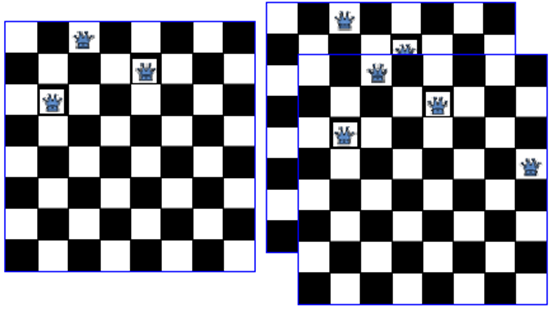


* N-여왕 문제 형식화 2
* 상태: 서양 장기판에 8-여왕의 모든 구성
* 초기 상태: 모든 여왕은 1열에 배치되어 있다.
* 후속 함수: 임의의 여왕을 이동
* 목표 시험: 8여왕이 배치되었으나, 어느 여왕도 공격할 수 없는 형세



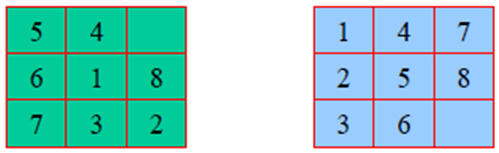
1열의 여왕을 이동한다면, 일곱 나머지 열 중 하나에 이동할 수 있다.

* N-여왕 문제 형식화 3
* 상태: 서로 공격하지 못하도록 첫 행부터 k개의 행에 k여왕의 가능한 모든 임의 배치
* 초기 상태: 여왕이 배치 안된 서양 장기판
* 후속 함수: 여왕들이 서로 공격하지 않도록 k+1번째 여왕 추가 배치
* 목표 시험: 8여왕이 배치되었으나, 어느 여왕도 공격할 수 없는 형세



2.1.4.4 8-퍼즐

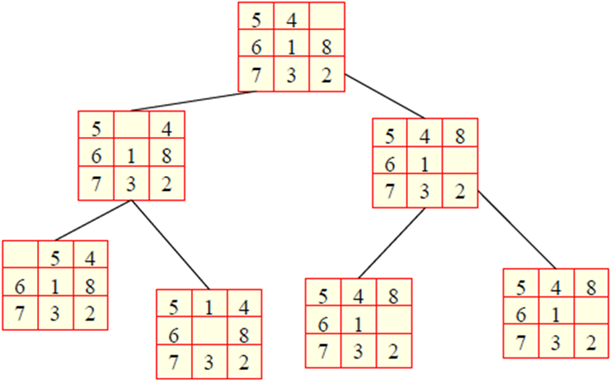
8-퍼즐 문제는 3X3 보드의 9칸에 8개의 타일로 채워 있으며 한 칸은 비어 있다. 타일은 빈칸으로 이동할 수 있습니다. 즉, 빈칸을 이용하여 초기 상태에서 정해진 목표상태로 도달하는 것이다. 이 역시 15-퍼즐(4X4)과 유사하다.



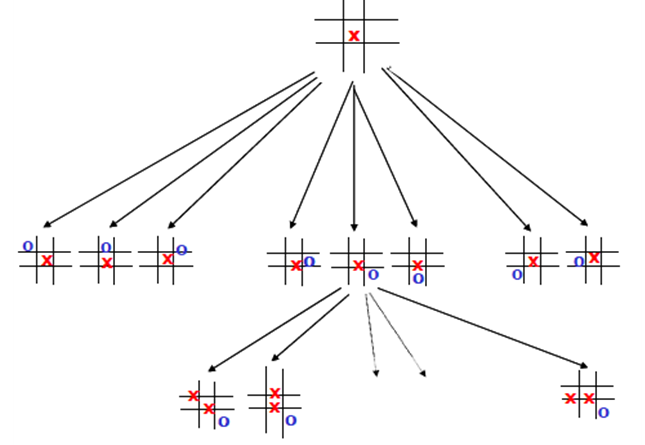
초기 상태 목표상태

* 상태: 한 상태는 8개의 타일의 점유한 위치로 표현된다.
* 연산자/행동: 빈칸이 상, 하, 좌, 우로 이동한다.
* 목표 시험: 주어진 최종 목표에 도달 여부
* 경로비용: 빈칸이 이동할 때 비용은 1이다.

8퍼즐의 상태 공간의 일부는 다음과 같다. 탐색이 시작되기 전의 모든 상태를 생성할 필요는 없으며, 상태는 필요할 때 생성할 수 있다.



2.1.4.5 틱택토(Tic-Tac-Toe)

틱택토 게임은 2인 선수와 후수로 교대로 하는 게임이다. 선수가 빈칸에 X를 두면 후수는 빈칸에 O를 둔다. 먼저 자신들의 심볼 셋을 열과 행 또는 대각선으로 두는 사람이 승자이다.

2.1.4.6 물 측정 용기 예제

12, 8, 3리터 측정용 용기가 있다. 1리터를 정확히 측정하는 방법을 제시하시오.

* 초기 상태: 3종류의 용기가 비어 있다.
* 목표 시험: 3종류의 용기 중 한 용기에 정확히 1리터의 물이 들어 있다.
* 후속 함수:
  + 물 따르기 행동: 용기 i와 j라 가정하면 그의 용량은 각각Ci, Cj 리터라 하고 그 용기들에 들어 있는 물을 Gi, Gj 리터라 하자. i용기의 물을 j용기에 따르면 max(0, Gi-(Cj –Gj) 리터의 물이 남아 있고, j용기에는 min(Cj, Gi+Gj)리터의 물이 들어 있게 된다.
  + 물 채우기 행동: i 용기에 Ci리터의 물을 채운다.
* 비용 함수: 매 1리터의 물을 따르고 채울 때 마다 1점씩 부여한다.
* 암묵적 상태 공간과 명시적 상태 공간

상태 공간은 그래프에 의해 명시적으로 표현이 가능하나, 더 전형적인 상태 공간은 암묵적으로 표현이 되고 필요할 때 생성할 수 있다.상태 공간을 암묵적으로 표현하기 위하여 에이전트는 초기 상태와 연산자 및 연산자의 효과의 서술에 관한 지식이 필요하다. 여기서 연산자는 노드(절점)을 확장하고 후속 노드(들)을 계산하는 함수 이다. 예를 들어N-여왕 문제에서 몇가지 형식화들에서 상당히 커질 수 있는 모든 가능한 상태의 명시적 표현을 유지할 필요가 없다.

교재 3.장4절(2주-2회차)

2.2. 탐색

상태 공간에서의 탐색은 다음을 포함한다.

* + 상태들의 집합
  + 연산자와 비용
  + 시작 상태
  + 목표 상태 달성 점검 시험

이제부터 기본 탐색 알고리즘을 설명하고 이 알고리즘의 변형들을 다루기로 한다.

2.2.1 기본 탐색 알고리즘

노드라고 불리는 생성된 상태의 조짐을 볼 필요가 있다. 노드의 데이터 구조는 상태뿐 만 아니라 부모 상태 또는 이 상태에 적용된 연산자로 구성된다. 더 나아가서 탐색 알고리즘은 프린지(가장자리)라고 불리는 노드들의 목록을 유지하고 있다. 프린지는 생성된 노드들 중에 아직 탐색되지 않는 노드들의 명칭이다. 즉, 프린지는 생성된 탐색 트리에서 최말단을 의미한다. 기본 탐색 알고리즘은 다음과 같다.

Let L be a list containing the initial stat (L = the fringe)

Loop

if L is empty return failure

Node ← select (L)

if Node is a goal

then return Node

(the path from initial state to Node)

else generate all successors of Node, and

merge the newly generated states into L

End Loop

탐색 초기의 프린지는 시작 상태에 대응하는 단일 노드이다. 여기서는 개방 목록 또는 개방 프린지만을 고려하기로 한다. 알고리즘은 항상 프린지의 첫 노드를 선택하여 확장을 시작한다. 만약 확장된 노드가 목표 상태이면 목표에 도달하는 경로를 반환한다. 목표 노드에 도달하는 경로는 부모 노드를 추적하면 얻을 수 있으며, 그렇지 않으면 모든 후속 노드를 생성하고 프린지에 추가된다.

현재 확장되는 노드의 후속 노드들은 프린지가 된다. 프린지에 넣는 후속 노드의 순서에 따라 탐색 알고리즘의 특성이 결정됨을 알 수 있을 것이다.

2.2.2 탐색 알고리즘: 중요한 사항

탐색 알고리즘은 생성되고 탐색되는 노드들을 포함한 트리를 탐색하는 것이다. 트리 탐색이 무한한 경우는 상태 공간이 무한대일 경우와 탐색 공간에 루프가 있을 경우에 발생할 수 있다.

탐색 알고리즘에서 패스와 노드를 반환해야만 하는가? 그 답은 문제에 따라 달라진다. N-여왕 문제는 단지 목표 상태에만 관심이 있고 15-퍼즈릉 해결책 경로만 관심을 갖는다.

기본 탐색 알고리즘에서 확장할 노드의 선택을 하는 방법, 새로 생성된 노드를 프린지에 배치하는 방법을 설명하기 위하여 탐색 전략과 속성에 관해 학습할 것이다.

탐색 문제에서 탐색 그래프는 가중치가 있을 수도 있고 없을 수도 있다. 어떤 경우에는 중간 상태의 품질에 관한 지식을 갖고 있고 그 지식은 탐색 알고리즘에 의해 이용될 수도 있다. 또한, 문제에 따라 최소 비용 경로를 탐색하거나 아니면 가능한 한 빨리 경로 탐색을 목표로 할 수도 있다.

* 경로를 찾는 방법은?

탐색 문제의 목적은 초기 상태에서 목표 상태까지의 경로를 찾는 것이다. 선택 가능한 경로의 수가 여러 개인 경우 어느 경로를 선택할지는 경우에 따라 달라진다. 즉, 목적에 따라 어떤 경로도 선택이 가능하나, 최단 경로 또는 최소 비용 경로 등을 선택하는 것이 일반적이다.

2.2.3 탐색 전략의 평가

다양한 탐색 전략들은 자신들의 특성이 다르므로 전략들을 문제해결능력들이 서로 다를 뿐만 아니라 효율성도 문제에 따라 다를 수 있다. 따라서 탐색 전략들의 평가를 위하여 다음과 같이 세가지 요소를 제시한다.

1. 완전성: 해결책이 존재한다면 전략이 반드시 해결책을 찾는 것을 보장하는가?
2. 최적성: 해결책을 찾는 비용이 최소 또는 저비용인가?
3. 해결책을 탐색하는데 소요 시간 및 메모리의 관점에서 탐색 비용은 얼마인가?
   1. 시간 복잡성: 해결책을 찾는데 소요시간(확장 노드의 수, 최악 또는 평균의 경우)
   2. 공간 복잡성: 프린지의 최대 크기로 측정되는 알고리즘이 사용한 공간

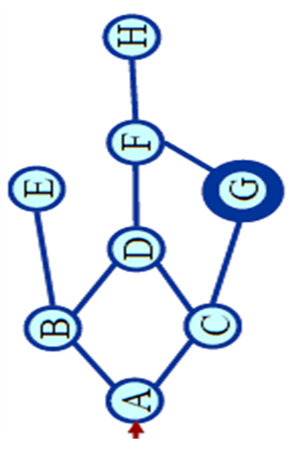
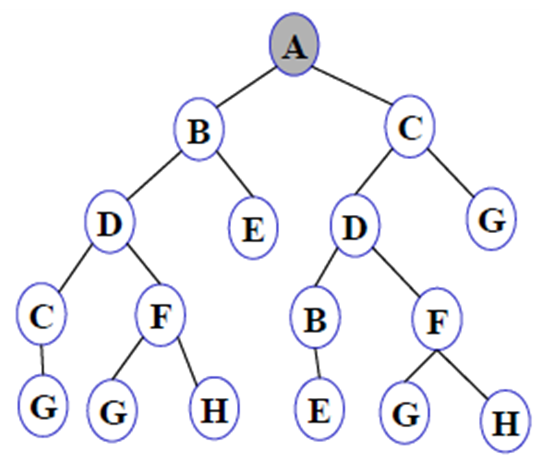
* 탐색 전략

탐색 전략은 다음과 같다.

1. 블라인드 탐색 또는 미지 탐색
   1. 깊이 우선 탐색(DFS: Depth First Search)
   2. 너비 우선 탐색(Breadth First Search)
   3. 반복 심화 탐색
   4. 반복 증폭 탐색
2. 경험적 탐색(또는 정보에 근거한 탐색: Informed Search)
3. 제약 만족 탐색
4. 적대적 탐색

2.2.4 탐색 트리(탐색에 필요한 정보로, 나중에 복습 반드시 할 것)

경로에서 순환(cycles) 루프를 제거하고 모든 가능한 경로를 나열하여 상태 공간 그래프에서 전체 탐색 트리를 얻을 수 있다. 탐색 트리와 관련된 용어를 먼저 알아보자. 탐색 트리는 탐색이 시작되는 뿌리 노드를 포함하는 데이터 구조이다. 모든 노드는 0부터 1개 이상의 자식을 가질 수 있다. 만약 X는 Y의 자식 노드라 하고 Y는 X노드의 부모 노드이다. 그림과 같은 명시적 상태공간 그래프를 이용하여 설명하기로 한다.

좌측의 상태 공간 그래프에서 우측의 탐색 트리를 얻을 수 있다. A는 탐색이 시작되는 곳이며 G는 목표이다.

* 탐색 트리-용어 정리
  + 뿌 리노드: 탐색이 시작되는 노드
  + 리프노드: 자식 노드가 없는 노드(변곡점 역할)
  + 조상(부모)/후손(자식): X는 Y의 조상이라함은 X는 Y의 부모 또는 X는 Y의 부모의 조상을 의미한다. 만약 X가 Y의 조상이라면 Y는 X의 후손이라 한다.
  + 분기 계수: 탐색 트리에서 자식 노드들 중 리프 노드가 아닌 노드의 최대 자식 노드 수 (1이 됨)
  + 경로: 탐색 트리에서 만약 시작 노드에서 목표 노드까지의 경로를 모두 포함하면 완전 경로라 하고 그렇지 않으면 부분 경로이다.
* 노드의 데이터 구조

탐색 알고리즘에서 노드의 데이터 구조는 다음을 포함한다.

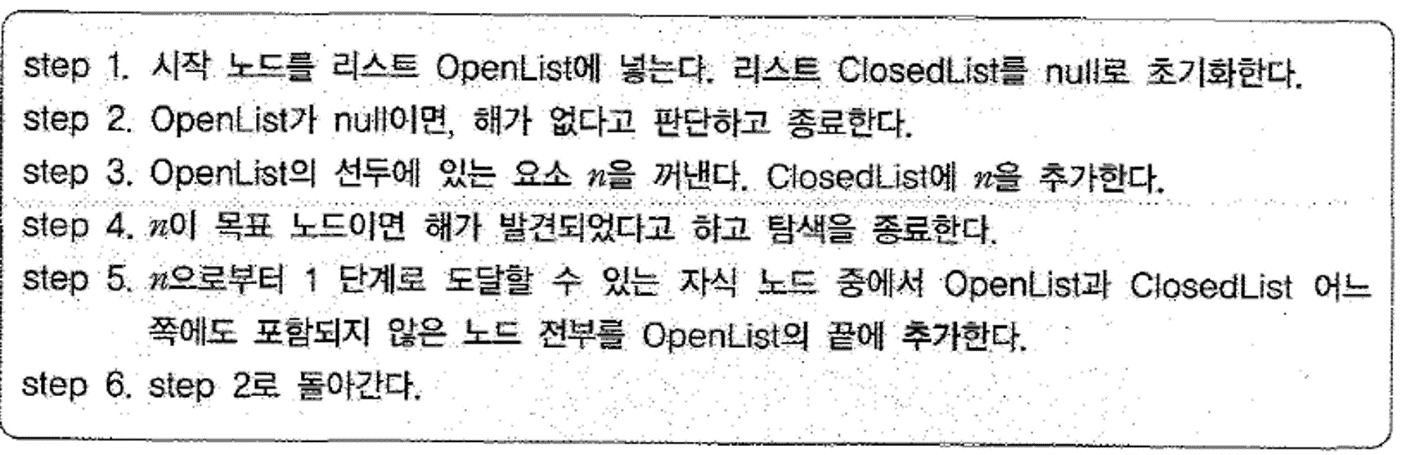
* 상태 설명
* 노드의 부모 포인터
* 노드의 깊이
* 노드의 생성 연산자
* 경로 비용(시작 상태부터 현재 노드 까지 소요된 연산자 비용의 합)
* 탐색 알고리즘이 데이터 구조를 유지하면서 생성된 노드는 OPEN(개방) 또는 프린지(fringe) 라고 한다.
* 탐색은 뿌리 노드에서 시작하여 알고리즘이 OPEN에서 확장할 노드를 선택하고 모든 자식 노드들을 생성한다. OPEN에서 확장하는 노드는 폐쇄 노드를 생성하기도 한다. 어떤 탐색 알고리즘은 데이터의 구조에 폐쇄 노드들을 포함하며 이를CLOSED라고 부른다.
* 탐색 문제의 해결책이란 시작 노드에서 목표 노드까지 연관된 연산자의 순차적 연산 결과이다. 해결책의 비용은 각 노드의 확장 시 발생되는 비용의 합이다.
* 상태 공간이 거대한 경우, 모든 공간을 표시하는 것은 현실적이지 못하다.
* 상태 공간의 탐색은 목표 노드를 발견하기 위하여 함축적인 상태 공간 그래프의 필요한 부분을 명시적으로 보여준다.
* 각 노드는 시작 노드에서 주어진 노드까지 부분 해결책 경로를 보여준다. 일반적으로 부분 해결책 경로에는 다양한 경로들이 존재할 수 있다.
* 탐색과정이란 탐색 트리를 만드는 것이라 볼 수 있으며 탐색 트리는 다음의 정보가 포함되어 있다.
  + - 뿌리 노드가 초기상태 또는 시작노드이다.
    - 리프 노드는 프린지에서 아직 확장되지 않았거나 자식이 없는 노드들이다.
* 상태 공간이 작을 지라도 루프가 있다면 탐색 트리는 무한 루프에 빠질 수도 있다.
* 탐색 문제에서 해결책 경로는 목표 노드에 반환하며 경로의 탐색은 15-퍼즐과 같은 문제에서 경로의 발견처럼 중요한 주요한 사항이다. 또한, N-여왕 문제의 해결책처럼 목표 상태 만을 반환하고 경로는 중요하지 않는 문제도 있다.

2.2.5 너비 우선 탐색(Breadth First Search)

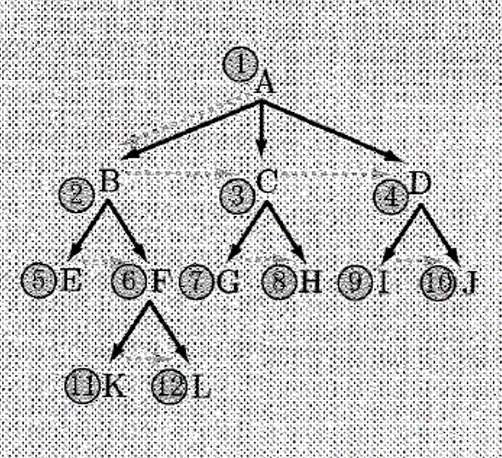
“횡형 탐색”이라고도 하며, 탐색 트리에서 같은 깊이의 노드를 모두 탐색한 후에 다음 깊이의 노드를 탐색하는 방법(즉, 탐색 트리의 너비를 우선하여 탐색을 수행하는 방법)이다.

2.2.5.1BFS 알고리즘

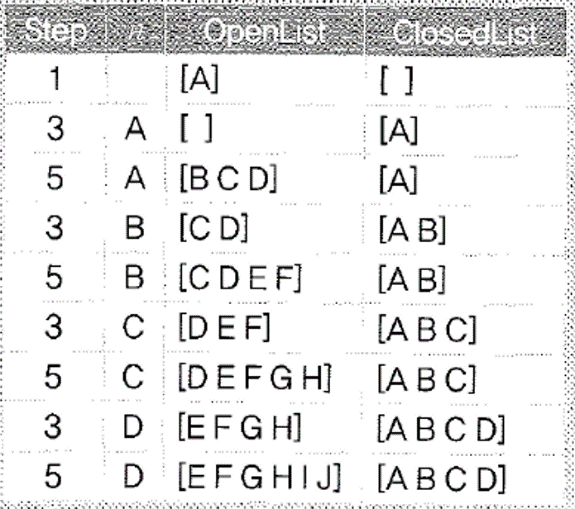
|  |
| --- |
| Breadth first search  Let fringe be a list containing the initial state  Loop  if fringe is empty return failure  Node ← remove-first (fringe)  if Node is a goal  then return the path from initial state to Node  else generate all successors of Node, and  (merge the newly generated nodes into fringe)  add generated nodes to the back of fringe  End Loop |



* 탐색 트리와 탐색 순서

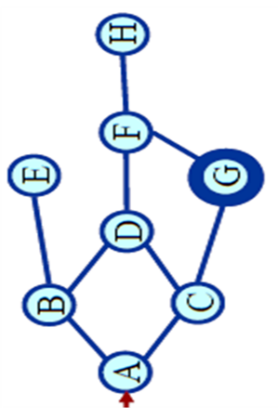


* 탐색과정



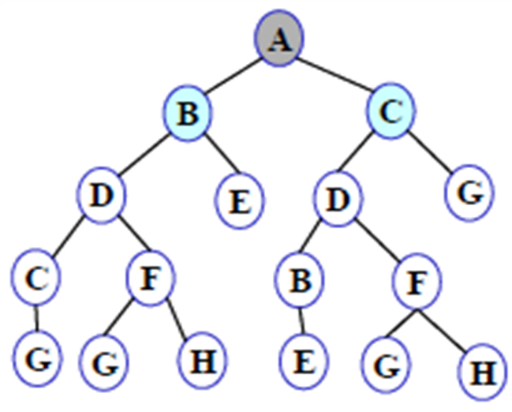
2.2.5.2 너비 우선 탐색의 설명

아래 그림과 같은 탐색 공간에서 도출된 탐색 트리에서 너비 우선 탐색을 설명하기로 한다.

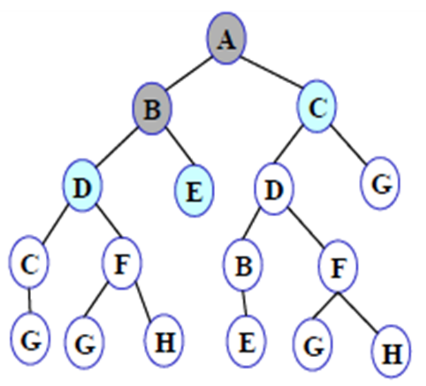
 

단계1: 시작 상태 A를 프린지에 넣는다.

프린지:A

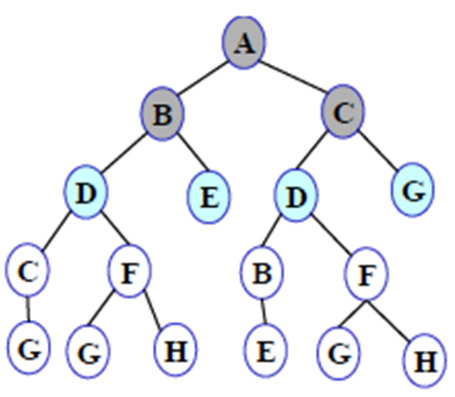
단계 2: A를 프린지에서 제거하고 노드가 확장한다. 자식 노드 B와 C가 생성되면 프린지에 넣는다. 

프린지:B C

단계 3: B를 프린지에서 제거하고 노드가 확장한다. 자식 노드 D와 E가 생성되면 프린지의 C뒤에 넣는다.

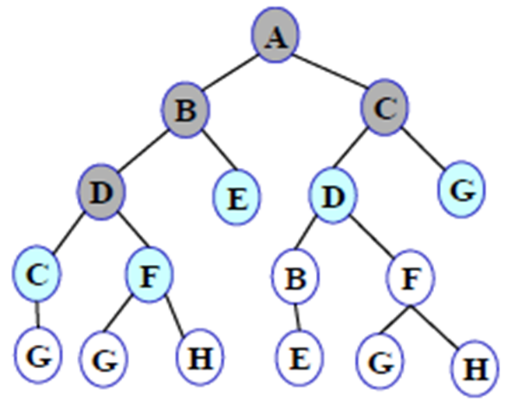
프린지: C D E

단계 4: C를 프린지에서 제거하고 노드가 확장한다. 자식 노드 D와 G가 생성되면 프린지의 C뒤에 넣는다.



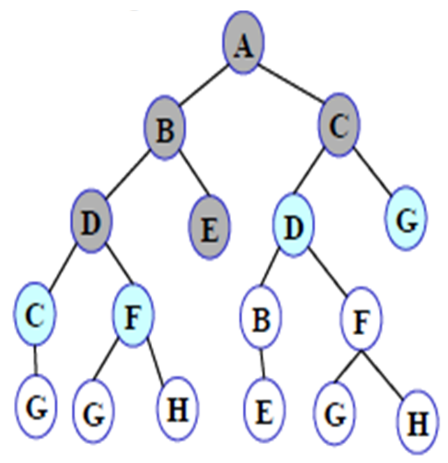
프린지: D E D G

단계 5: D를 프린지에서 제거하고 노드가 확장한다. 자식 노드 C와 F가 생성되면 프린지의 G뒤에 넣는다.



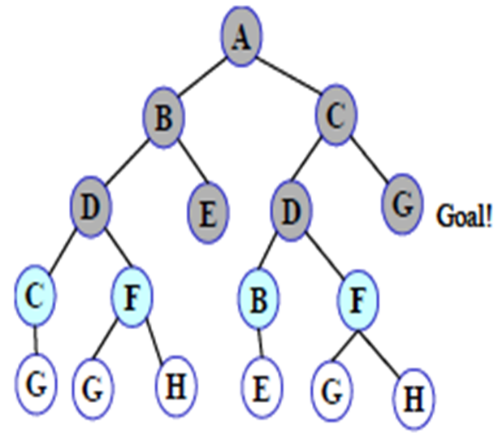
프린지: E D G C F

단계 6: E는 자식이 없으므로 프린지에서 삭제한다.



프린지: D G C F

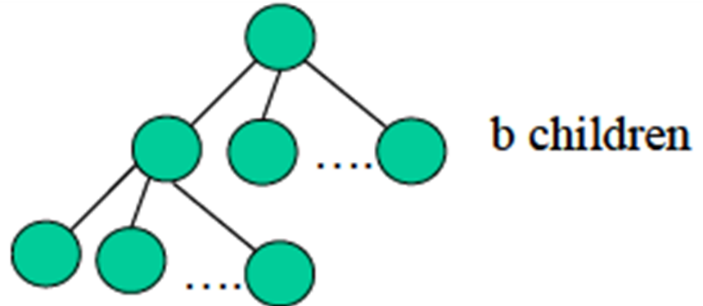
단계 7: D를 확장하고 B와 F를 OPEN에 배치한다.



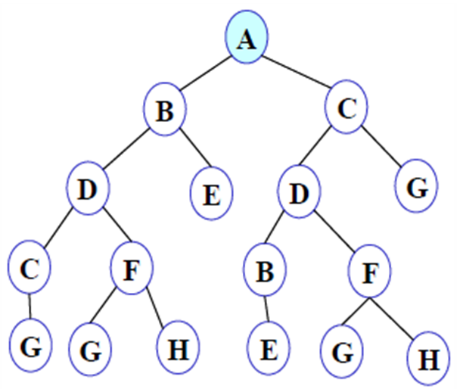
프린지: G C F B F

단계 8: G를 확장하기 위하여 선택 한다. G가 목표 노드임이 확인 된다. 알고리즘은 G의 부모 포인터를 따라 경로 A, C, G를 반환하고 종료된다.

2.2.5.3 너비 우선 탐색의 특성

너비 우선 탐색의 특징을 설명하기 위하여 아래의 그림과 같은 트리에서 리프 노드를 제외한 모든 노드는 b의 자식 노드를 갖는다 가정한다. 또 d는 가장 얕은 곳에 있는 목표 노드의 깊이로 가정하고 m은 현재 조사 중인 노드의 깊이로 가정한다.

* 너비 우선 탐색의 특성
  + - 완전성
    - 알고리즘은 모든 연산자들의 비용이 동일하다면 최적화된 또는 적어도 용인할 만 것이 해결책을 찾을 수 있으며 또한 알고리즘은 최단 경로의 해결책을 찾는다.
    - 알고리즘은 시간과 공간의 복잡성에 지수적으로 변화된다. 아래에 보인 탐색 트리에서 시간과 공간 복잡성은 O(bd)이다. 여기서 d는 해결책의 깊이이고 b는 가지 인수(branching factor), 각 노드에서 자식의 수이다.



리프 노드가 아니며 깊이가 d인 각 노드가 b개의 자식 노드를 갖는 완전한 트리 탐색은 총 1+b+b2+ ... + bd = (b(d+1) - 1)/(b-1)노드를 갖는다.

깊이 0에서 14까지는 자식 노드의 수가 10이고 깊이 15의 모든 노드는 리프 노드인 완전한 탐색 트리는 O(1015)노드를 갖는다. 만약 너비 우선 탐색으로 초당10,000 노드의 속도로 연산하고 메모리는 각 노드 당 100바이트가 필요하다면 이 탐색은 최악의 경우 3,500년이 소요되고 메모리는 11,100 테라바이트가 필요하다. 따라서 너비 우선 탐색은 탐색 공간이 작지 않으면 효율적이지 못하다. 또한 메모리의 부족으로 탐색 알고리즘이 실행되지 않을 수도 있다.

* 너비 우선 탐색의 장점: 최단 길이로 목표를 찾을 수 있다.
* 너비 우선 탐색의 단점: 목표 노드의 최단 깊이의 지구적인 크기를 갖는 트리의 생성과 저장이 필요하다

2.2.6 균일 비용 탐색

이 알고리즘은 디즈크스트라(Dijkstra, 1959)가 제안한 것으로 트리에서 그들의 비용의 정도에 따라 노드 확장의 순서가 정하여지는 알고리즘이다. 즉, 연산자의 비용에 따라 확장 순서가 정해진다. 경로 비용은 일반적으로 단계별 비용의 합이다.

미지 탐색에서 새로 생성된 노드들을 그들의 경로 비용에 따라 OPEN에 넣는다. 이는 한 노드가 확장을 위해 선택될 경우 그 노드는 OPEN안의 노드 중에 가장 저비용의 노드임을 보증한다.

g(n)은 시작 노드에서 현재 노드 n까지 경로의 비용이다. 노드의 분류는 g값이 증가에 따라 노드가 분류된다. 균일 비용 검색은 가장 얕은 노드를 검색하는 것이 아니라 경로 비용 g(n)이 가장 낮은 노드 n을 확장한다.

* 균일 비용 탐색의 특성

- 완전성

- 최적화/ 수용 가능

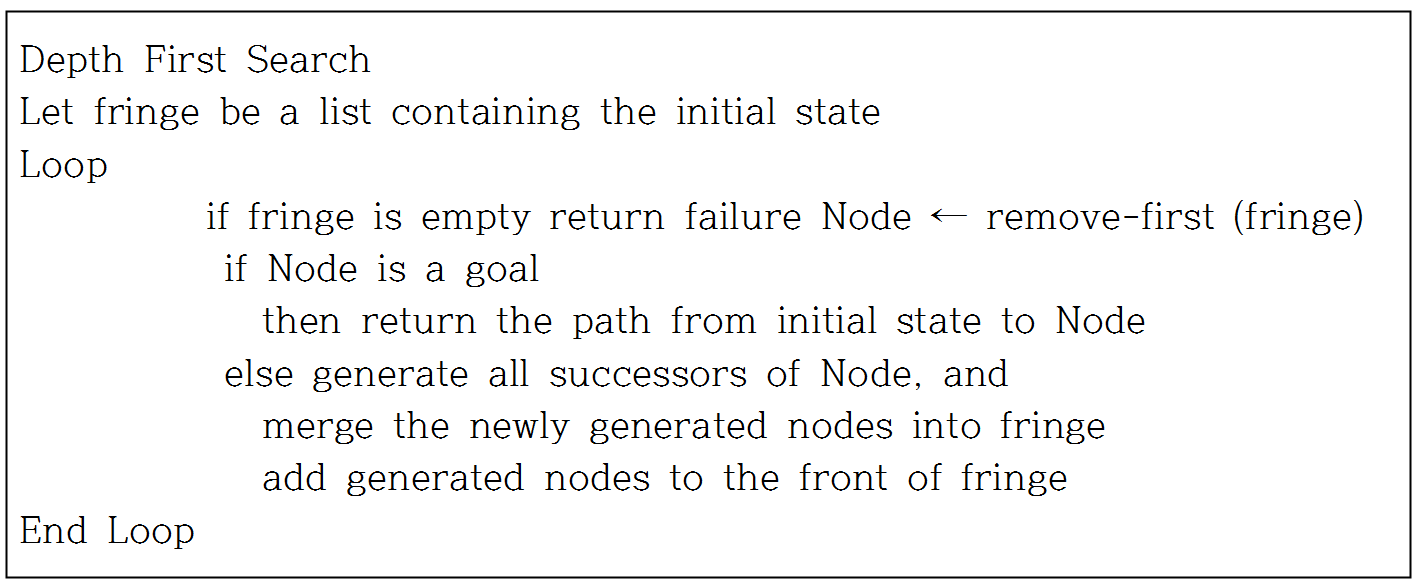
- 시간과 공간 복잡성은 지수적이다, O(bd).

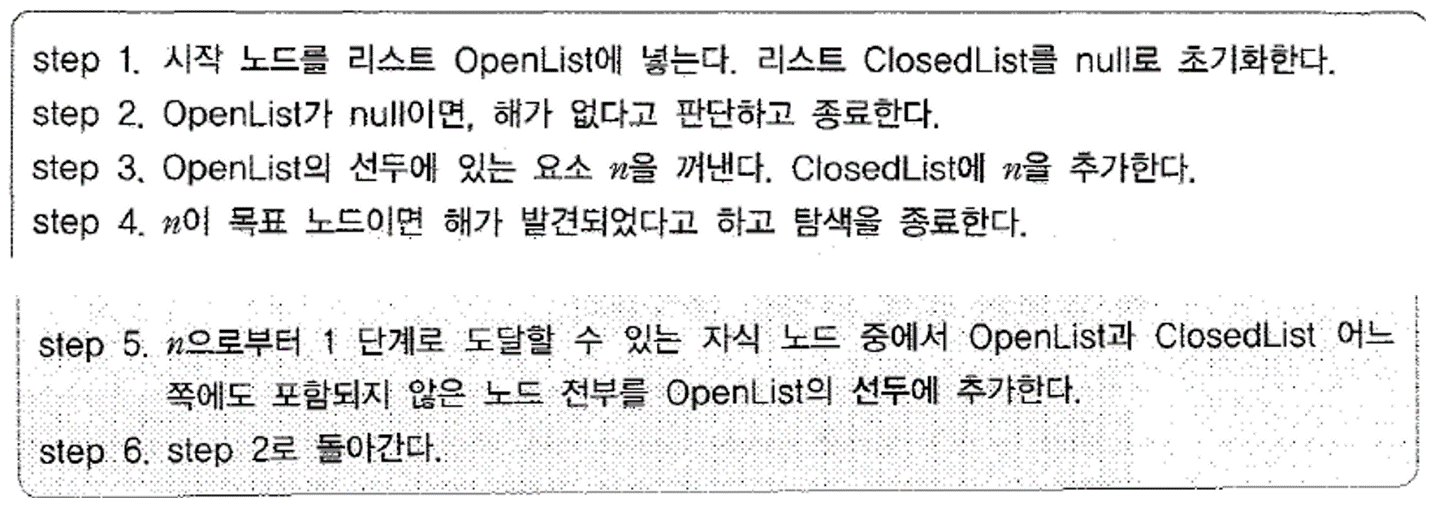
2주차 3차시

2.3.1. 깊이 우선 탐색

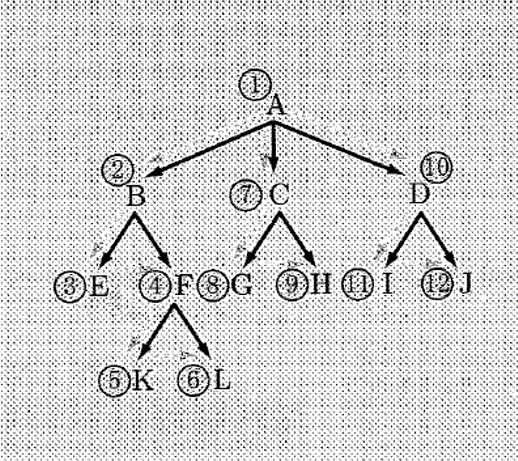
* 알고리즘

최대 깊이의 확장을 우선하기 위하여 깊이 우선 탐색 알고리즘은 OPEN의 앞에 새로 생성된 노드를 넣는다. 따라서 OPEN에서 노드는 최종 입력이 우선출력(LIFO:Last-in First-out)순서를 갖으므로 OPEN은 스택 데이터 구조이다.

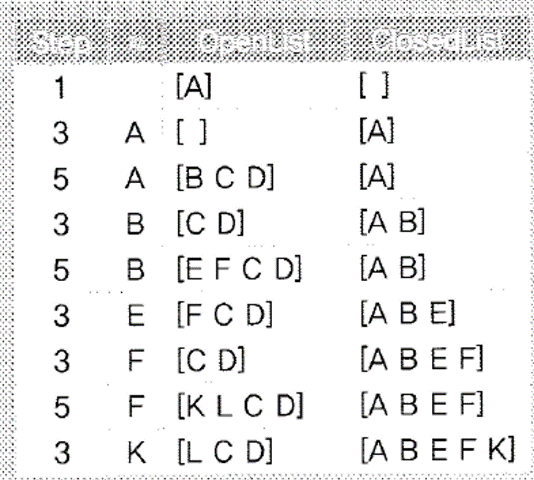




* 탐색 트리와 탐색 순서

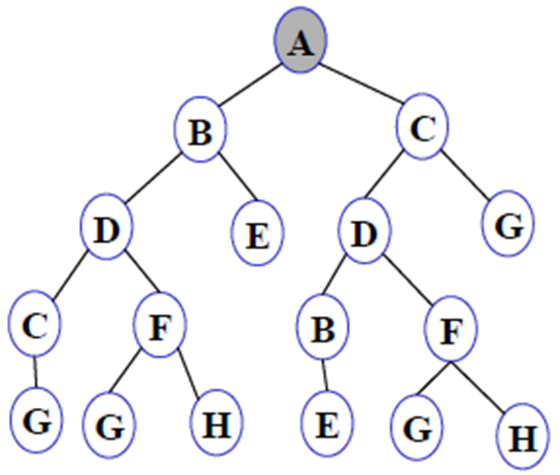
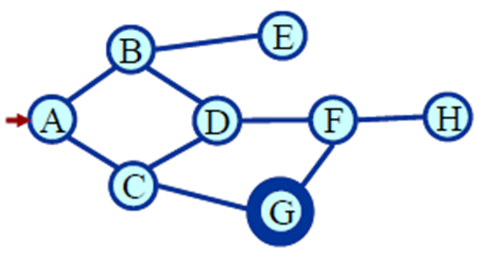


* 탐색 과정

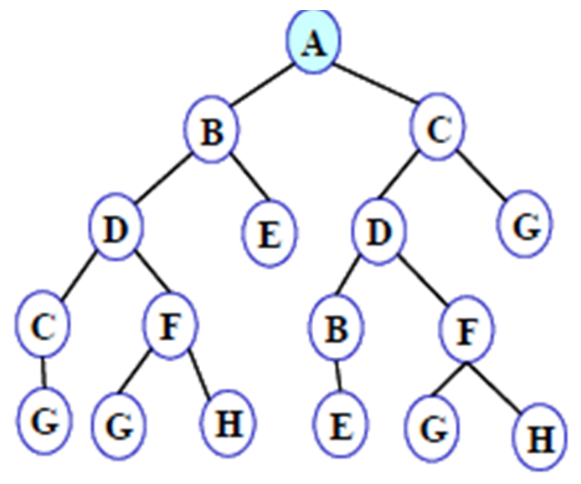


2.3.1.1 깊이 우선 탐색 설명

상태 공간이 좌측과 같이 주어졌다면 탐색 트리는 우측과 같다. 깊이 우선 검색을 단계별로 수행하여 설명하기로 한다.

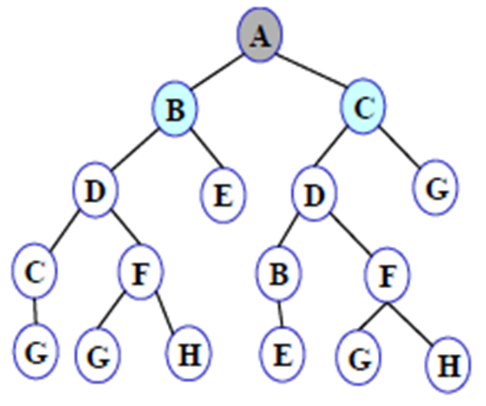


단계 1: 초기 상태의 프린지에 시작 노드인 A를 넣는다.



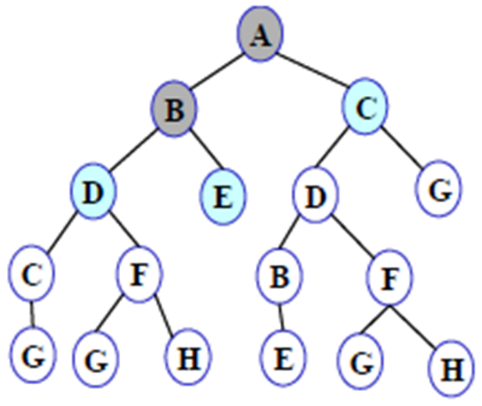
|  |
| --- |
| 프린지: A |

단계 2: 프린지에서 확장을 위해 노드 A는 제거되고, 자식 노드인 B와C는 프린지의 앞에 넣는다.



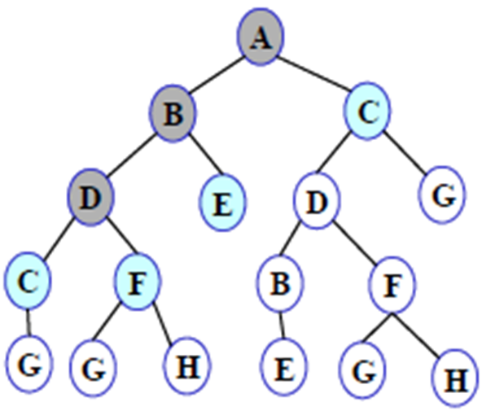
|  |
| --- |
| 프린지: B C |

단계 3: 프린지의 맨 앞의 노드 B를 제거하고, B의 자식 노드 D와 E를 프린지의 맨 앞에 넣는다.

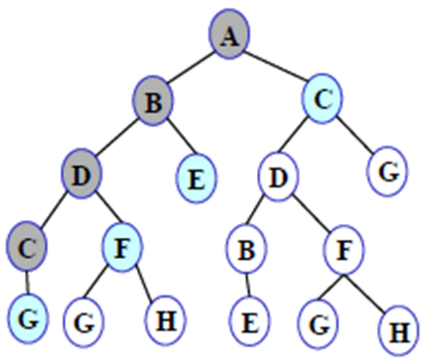


|  |
| --- |
| 프린지: D E C |

단계 4: 프린지의 맨 앞의 노드 D를 제거하고, D의 자식 노드 C와 F를 프린지의 맨 앞에 넣는다.

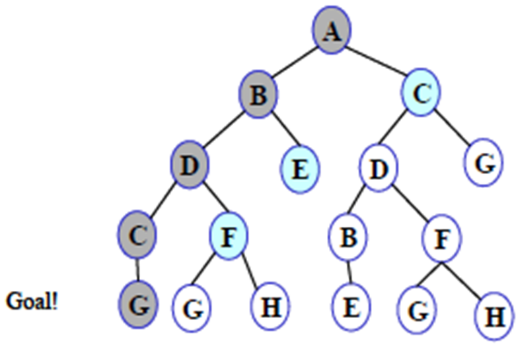


|  |
| --- |
| 프린지: C F E C |

단계 5: 프린지의 맨 앞의 노드 C를 제거하고, D의 자식 노드 G를 프린지의 맨 앞에 넣는다.

|  |
| --- |
| 프린지: G F E C |

단계 6: 노드 G가 확장되고 목표 노드임을 발견한다. 그림과 같이 경로 A→B→D→C→G를 반환되고 알고리즘은 종료된다.



|  |
| --- |
| 프린지: G F E C |

2.3.1.2 깊이 우선 탐색의 특성

DFS 알고리즘의 몇 가지 특성을 살펴 보기로 한다.

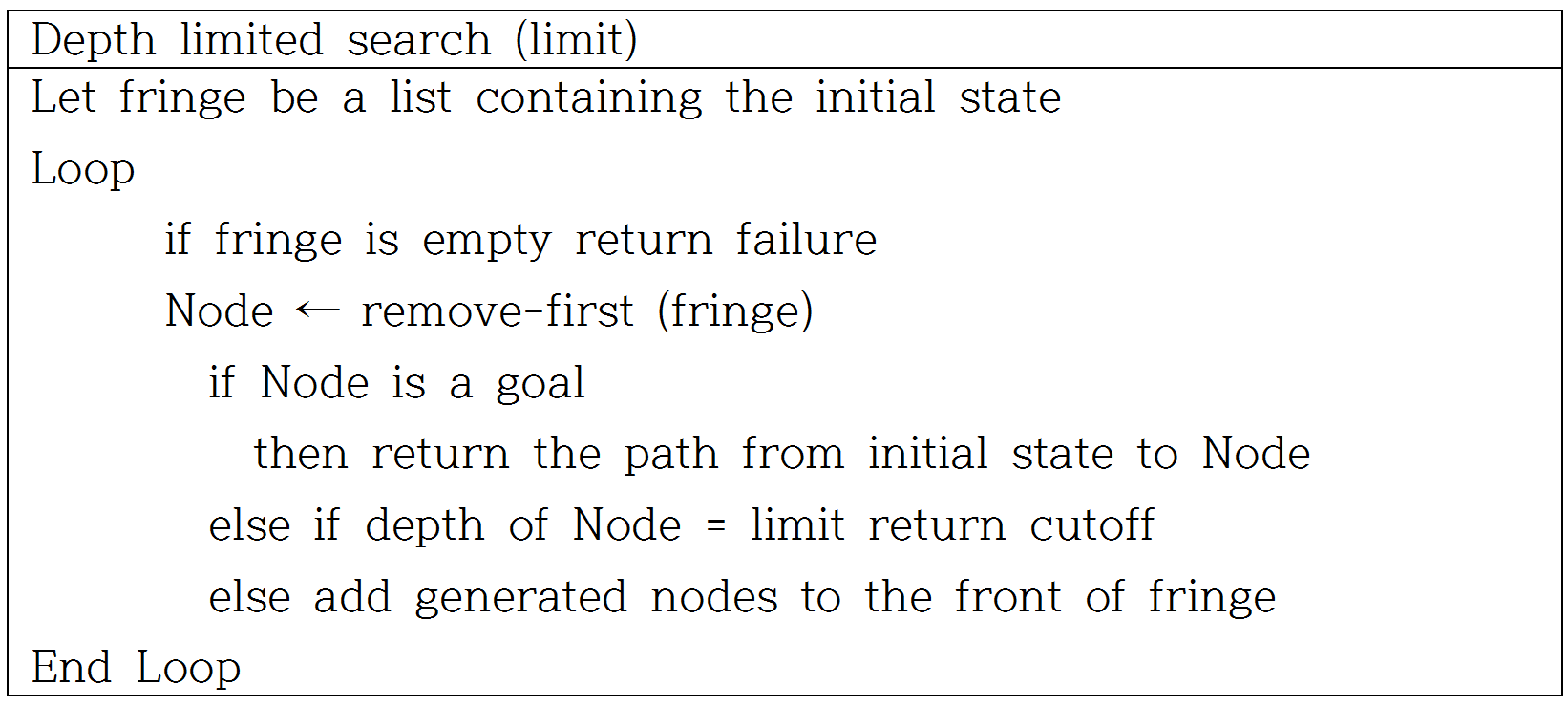
이 알고리즘의 수행시간은 지수적으로 증가된다. 만약 N이 탐색 공간 내의 노드의 최대 깊이 인 경우, 최악의 경우, 알고리즘은 시간 O (BN)이 소요된다. 그러나 탐색 공간은 탐색 트리의 깊이에 선형적이다, O (bN).

알고리즘의 수행 시간은 탐색 트리의 최대 깊이에 종속 된다. 상태 공간이 무한대일 경우 탐색 트리가 무한 깊이이므로 알고리즘은 종료될 수 없다. 또한, 검색 공간에 순환 경로(사이클)를 포함하는 경우에도 발생할 수 있다. 후자의 경우는 알고리즘의 주기를 확인하여 처리 될 수도 있지만, 깊이 우선 탐색은 완전성을 유지하지는 않는다.

2.3.2 깊이 제한 탐색

무한 상태 공간에서 탐색 트리의 깊이가 무한한 경우 깊이 우선 탐색의 적용이 곤란하므로 깊이 우선 검색에서 깊이를 제한하는 것도 방법이다. 즉, 제한을 초과하는 깊이 의 노드들은 마치 자식 노드가 없는 것처럼 취급하면 무한 경로 문제를 해결할 수 있지만 목표가 깊이 제한 보다 깊다면 알고리즘은 완결되지 않는다.

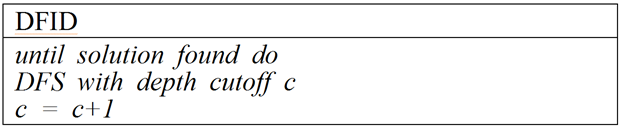
* 알고리즘

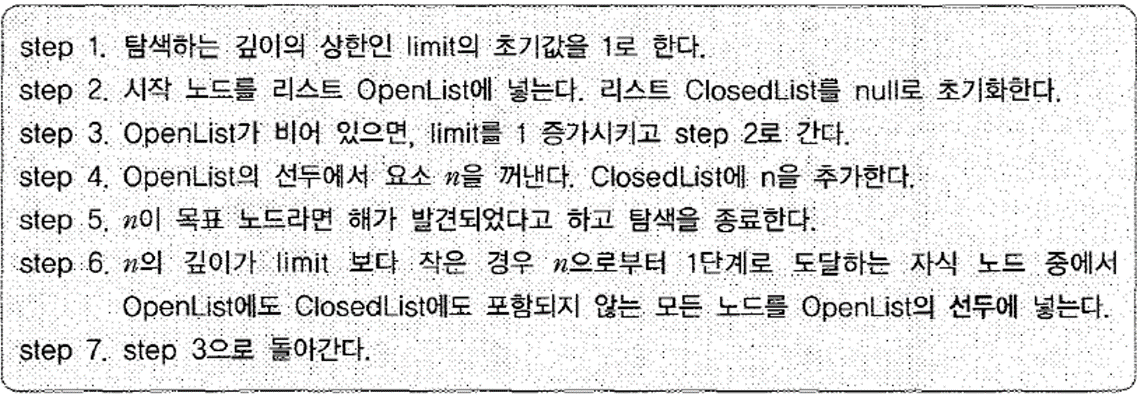


2.3.3 깊이 우선 반복 심화 탐색(Depth First Iterative Deepening Search)

이 탐색 방법은, 깊이를 우선시하는 탐색 방법을 취하되, 탐색의 깊이를 서서히 깊게 하면서 깊이 우선 탐색을 수행하는 방법이다. 너비 우선 탐색과 깊이 우선 탐색에서 각각의 장점을 도입한 탐색이다.

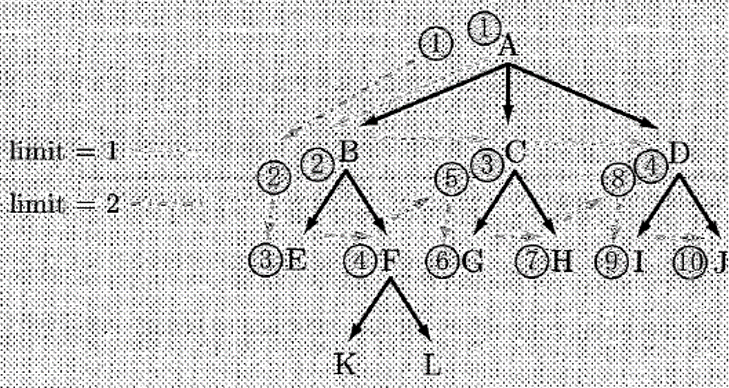
* 알고리즘

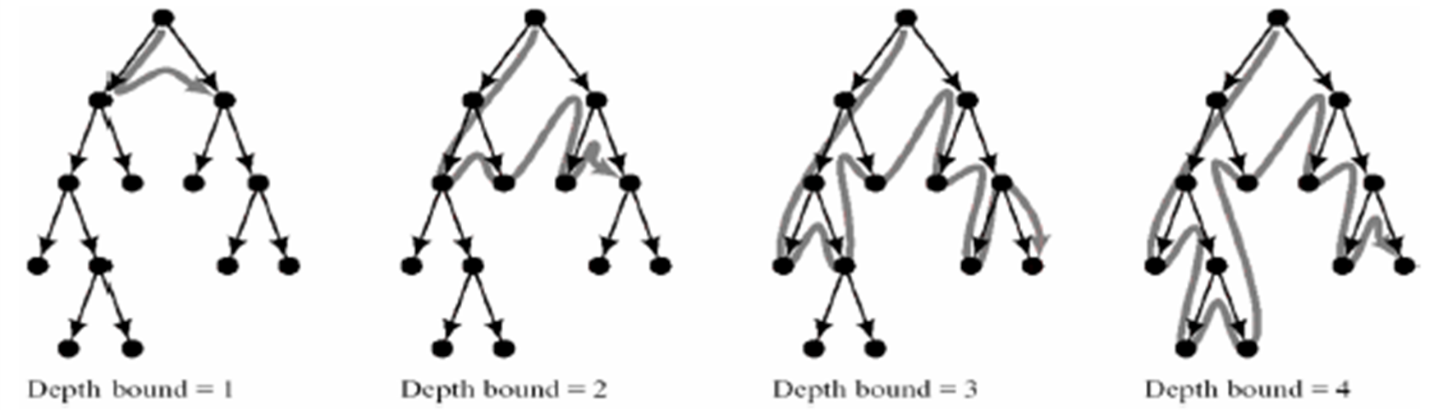




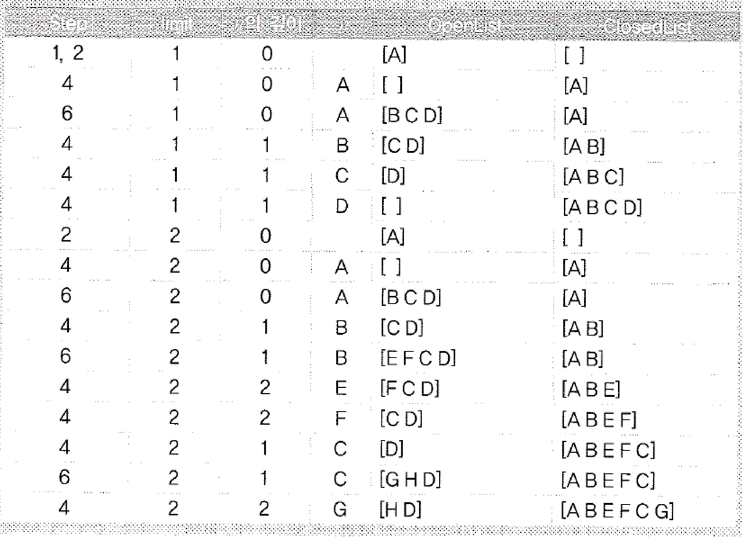
* 탐색 트리와 탐색 순서

반복적인 깊이 우선 탐색의 수행 순서(깊이의 제한이 +1인 경우)





* 탐색과정



* 반복 심화 탐색의 특징

반복 심화 탐색은 너비 우선 탐색과 깊이 우선 탐색의 좋은 특정을 포함하고 있지만, 위 표에 나타난 탐색 과정을 보면 알 수 있듯이 limit 변수의 값을 증가시킬 때마다 다시 시작 노드에서부터 탐색을 시작하기 때문에 처리 대상이 되는 노드가 늘어나게 되는 것과 한 번 탐색한 노드들을 다시 탐색하므로 노드의 처리량이 크게 되는 단점이 있다.

깊이(d)가 깊은 경우DFID 의 확장 된 노드의 수를 비교하면 DIFD와 DFS와 비는 b/(b-1)이다. 목표 상태가 깊고 가지의 인수(branching factor)가 10이라 하면 DIFD에서 확장 노드의 수는 DFS보다약 11% 정도 증가하지만 다음과 같은 속성이 있다.

* 완전성(Complete)
* 모든 연산자가 균일 비용이라면 최적화 또는 허용할 만 하지만 BFS처럼 최단 해결책의 발견을 보장하지 않는다.
* 탐색 트리의 상단 노드는 반복적으로 생성되므로 BFS나 DFS보다 시간의 복잡성은 다소 커지지만, 대부분의 노드들이 탐색 트리의 하단부에 있기 때문에 최악의 시간 복잡도는 O(bd)이다.
* 공간 복잡도는 DFS처럼 선형적으로 O(bd)이다.
* DFIS는 BFS의 장점인 완전성과 제안된 공간에서 경로를 빨리 찾는 DFS의 장점이 결합되었음으로 해결책의 깊이를 모르는 거대 상태 공간에서의 탐색에 유리하다.
* 장점
  + - 깊이 우선 탐색과 같이 선형 메모리가 소요된다.
    - 목표 노드의 최소 깊이 탐색을 보장한다.

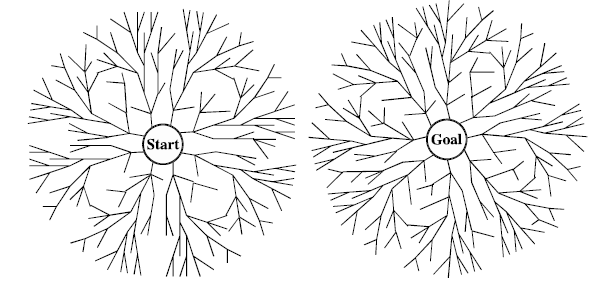
2.3.4 양방향 탐색

앞에서 다룬 물 용기 문제와 같이 상태 공간 형식화가 다르며 양방향성을 갖자 않는 문제도 있기는 하지만 탐색 문제에서 호가 양방향성을 갖고 있어 가역적 특성을 갖는다고 성립할 때, 상태 A에서B로 사상하는 연산자(행동)있다면 상태 B에서 A로 사상하는 또다른 연산자가 있다. 8-퍼즐, 15-퍼즐, 경로 계획 등등 많은 탐색 문제에서 양방향성 호를 갖는다. 그러한 조건이 만족되면 초기 상태부터 시작해 목표를 탐색하는 대신에 목표에서부터 해결책을 찾을 수도 있다.

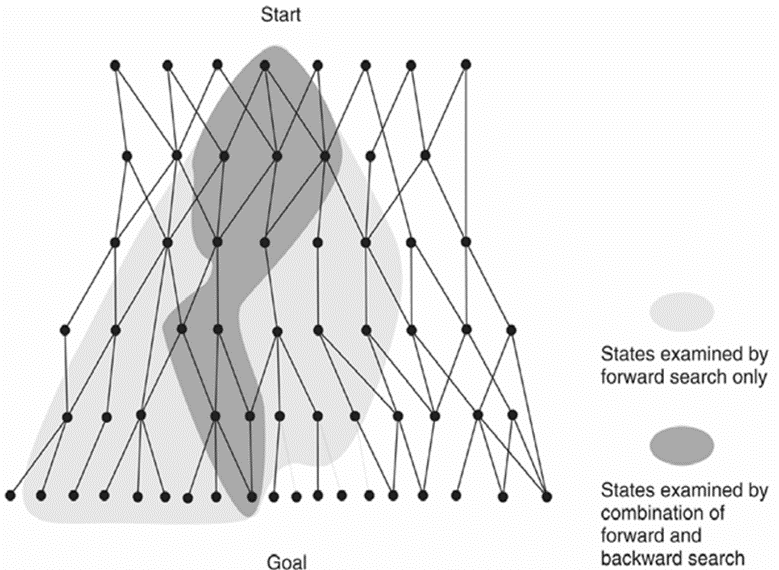
양방향탐색(bidirectional search)은 두 가지탐색, 즉 초기 상태에서 시작하는 전방(순방향) 탐색과 목표에서 시작하는 후방(역방향) 탐색을 동시에 진행한다는 것이다. 두 탐색이 중간에서 만난다면 성공이다. 아래에 양방향 탐색의 개념도를 보였으며 시작 노드에서 나온 가지 하나가 목표 노드에서 나온 가지 하나와 만나서 솔루션을 성공적으로 찾아내기 직전의 모습이다.

* 알고리즘

양방향 탐색은 시작 상태에서목표를 향하고 목표에서 시작 상태를 향해 동시에 탐색하는 알고리즘으로 두 탐색이 교차되면 알고리즘은 중지된다.



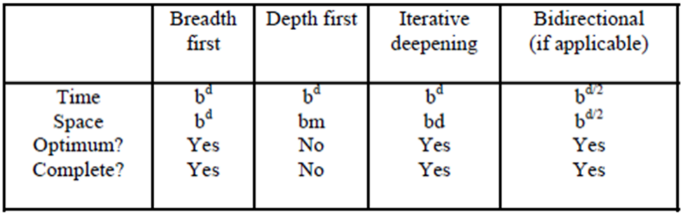
* 양방향 탐색은 경우에 따라서 해결책을 더 빨라 찾을 수 있다. 그 이유는 아래 그림 보면 알 수 있다. 또한, 시작과 목표 상태가 있는 경우에 적용이 가능함은 당연하다.



* 시간과 공간 복잡성

분기 인수 B를 갖는 탐색 공간에서 목표는 깊이가 d 라면 너비 우선 탐색은 O(bd) 노드를 확장한다. 만약 양방향 검색을 수행 할 경우 순방향 및 역방향 탐색 트리의 깊이가 d / 2에서 양방향 탐색이 교차된다. 프린지의 노드를 확인하는 좋은 해시 함수가 있다면 쌍방향 탐색 소요 시간은 O(bd / 2)이며, 공간 복잡도도 간단한 경우라면 O(bd / 2)이다.

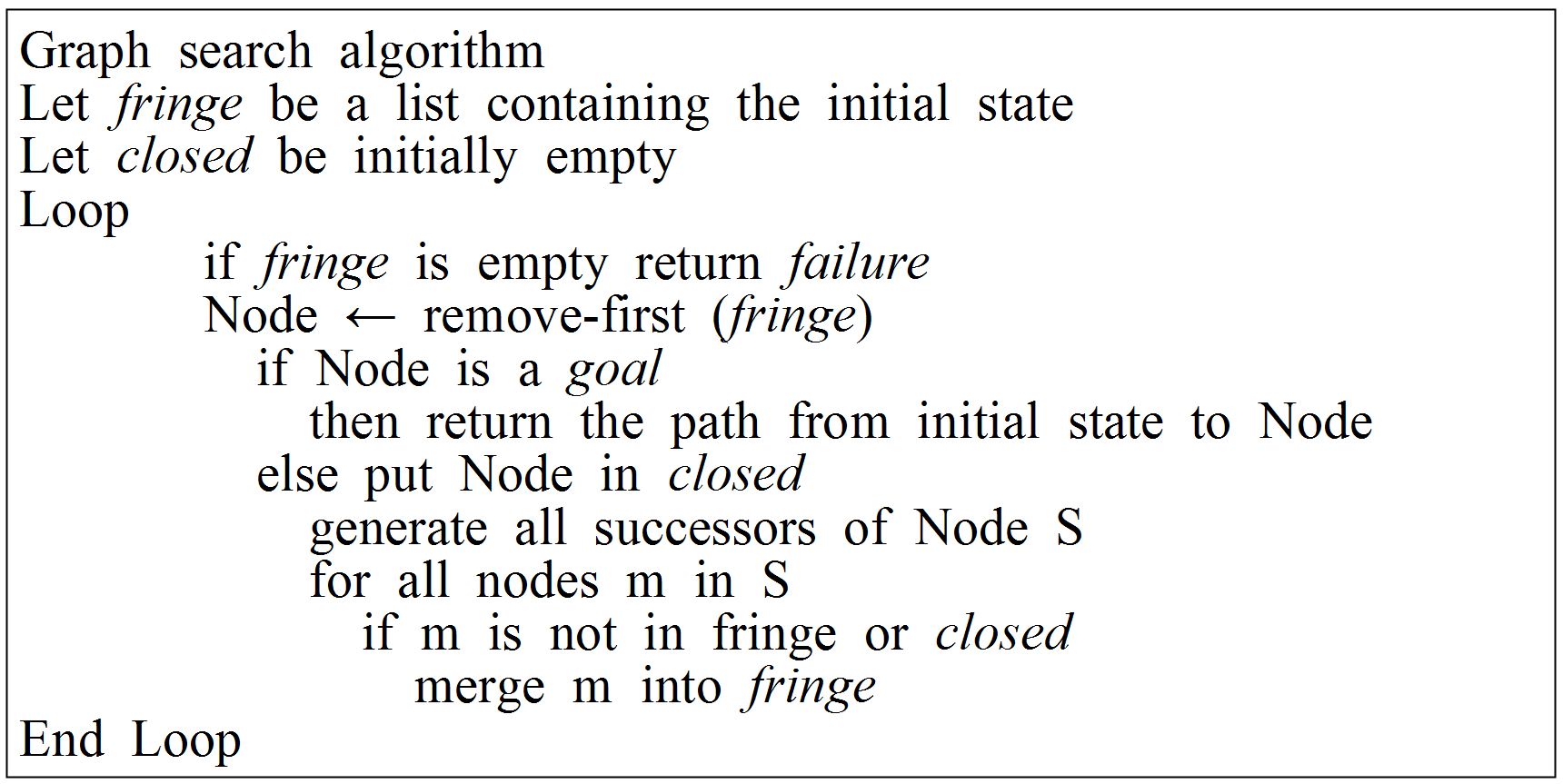
2.3.5 탐색 전략 비교



2.3.6 그래프 탐색

탐색 공간이 트리가 아니고 그래프로 표현된다면 탐색 트리는 동일한 상태에 대응하는 다른 노드를 포함 할 수 있다. 이는 병리학 예와 같이 탐색 공간의 크기는 총 상태의 수의 지수적이다.

많은 경우 반복 된 상태 확장을 방지하기 위해 탐색 알고리즘을 수정할 수 있다. 필요하지 않은 경우에, 같은 상태를 다시 생성하지 않도록 탐색 알고리즘이 상태를 확장하기 전에 점검하도록 수정하는 것이다. 만약 그 상태에 해당하는 다른 노드가 OPEN에 이미 들어있다면. 새로운 노드는 폐기한다. 문제는 어떤 노드가 OPEN에 들어 있고 버린 노드는 무엇인지를 추적할 필요가 있다. 이를 위해 확장된 모든 노드들이 기록된 CLOSED라는 목록을 사용한다. 새롭게 생성된 노드는 CLOSD목록과 점검하고 없는 OPEN에 넣는다. 이 알고리즘은 다음과 같다.

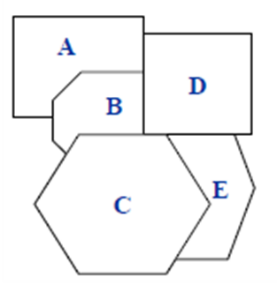


* 그래프 탐색의 특성
* 알고리즘은 CLOSED 목록을 유지하여야 할 뿐만 아니라 모든 노드에서 노드를 색인 하기 위한 매우 효율적인 방법이 없다면 추가 비용이 상당히 소요된다.
* 많은 탐색 문제에서 연산량이 적은 전략을 선택하고 상태들의 중복 생성을 방지하는 방법을 강구하지만 효과는 기대에 미흡하다.
* 가장 간단한 전략은 알고리즘이 진행된 곳으로는 되돌아 가지 않는 것이다. 이 간단한 전략은 15-퍼즐문제에서 많은 노드들의 재확장의 방지가 가능하다.
* 알고리즘이 순환 경로를 생성 못하도록 감시하고 알고리즘이 전체 검사 보다는 현재 경로상의 노드들을 검사하도록 하고 깊이 우선 탐색을 적용하고 추가적인 메모리의 소요를 방지하는 것이다.
* 마지막으로 앞에서 언급한 바와 같이 이전 생성된 모든 상태는 생성하지 못하도록 하고 상태 공간 내에서 루프의 빈도를 고려하여 전략을 채택하여야 한다.

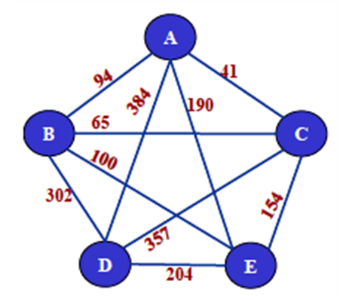
2.3. 질문

1. 각각의 주어진 탐색 문제에서 초기 상태, 목표 시험, 후속 함수(연산자), 비용 함수를 구하여라. 충분히 구현 가능하도록 자세한 형식화하여라.

a) 주어진 지도를 4가지 색으로 색칠하라. 단 인접한 지역은 다른 색을 사용하여야 한다.



b) 아래와 같이 길로 연결된 N(=5)개의 도시가 있다. 이는 판매원 순회 문제로 판매원의 목표는 현재 도시에서 모든 도시를1번씩만 방문하고 돌아오는 최단거리를 구하는 것이다.



c) 선교사와 식인종이 각각 3명씩 강변에 있다. 보트를 이용하여 강을 건너야 한다. 어떤 경우에도 선교사들의 수가 식인종들의 수보다 적으면 공격받는다. 모두 강을 건너는 방법을 제시하시오

2. 5리터의 용기에 물이 가득 차 있고 2리터의 용기는 비어 있다. 목표는 2리터의 용기에 정확히 1리터만 담는 것이다. 상태 공간 형식화는 다음과 같다.

State = (x,y), 여기서 x는 5리터 용기의 물의 양이고 y는 2리터 용기의 물의 양이다.

초기 상태는 (5, 0), 목표 상태는 (\*, 1) 여기서 \*는 임의의 물의 양, 탐색 트리를 그리고 이 문제에 적절한 탐색 전략을 기술하라.

3. 다음과 같은 그래프에서 상태 A에서 시작하여 목표 상태인 G상태에 도달하는 경로를 깊이 우선 탐색으로 찾고자 한다. 노드가 확장되는 과정을 보여라. 확장되는 노드가 하나 이상 일 경우 알파벳 순으로 하라.



4. 다음과 같은 탐색 공간이 있다.

a) 이 문제의 상태 공간을 그려라

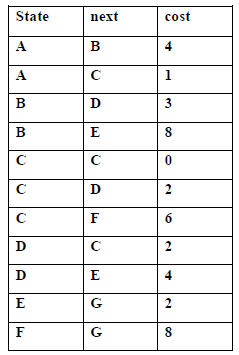
b) 초기 상태는 A이고 목표 상태는 G이다. 다음과 같은 전략들을 적용하여 초기 상태에서 목표 상태에 도달하는 경로를 발견하기 위한 탐색 트리를 그려라. 또 탐색 알고리즘의 각 단계에서 어떤 노드가 확장되는지 프린지에 들어있는 노드들을 보이고, 각 탐색 전략 별 해결책과 총 경로비용을 구하시오.

I. 너비 우선 탐색

II. 깊이 우선 탐색

III. 균일 비용 탐색

IV. 반복 심화 학습



5. 너비 우선 탐색에서 주어진 그래프에서 N노드들이 확장된다. 반복 심화 탐색에서 확장되는 노드들의 최대 수는 몇인가?

2.3 풀이

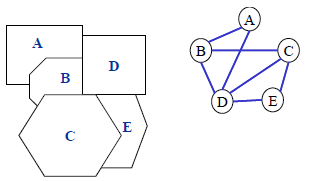
1. a) 네 가지 색으로 주어진 지도를 채색한다. 단 인접한 지역은 같은 색이 아니어야 한다. 지도를 그래프로 표현하자면 각 지역은 그래프의 꼭지점으로 표현하고 인접 지역은 해당 꼭지점들을 연결한다. 주어진 문제는 N-지역에 c개의 색으로 채색하는 문제로 일반화가 가능하다.

꼭지점들의 이름은 (v1, v2, … , vn)

4가지 색들을 c1, c2, c3, c4라 하기로 한다.

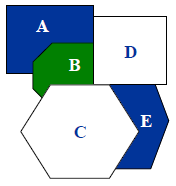
상태는 꼭지점들과 색들로 표현된 N-튜플(속성값들의 모음)이다.

꼭지점의 색이 정해지지 않았다면 x로 표기하면 각 상태의 표시는

{ci, x, cj, x, x, x....}가 되며 여기서 i와 j는 1에서 4까지의 임의의 수이다.

이문제(위 좌측)의 상태는 다음과 같다

상태 S는 {A의 색깔, B의 색깔, C의 색깔, D의 색깔, E의 색깔}.

이를 아래 그림의 경우, 상태는 {청, 녹, x, x, 청}이 된다.

이 문제의 형식화는 다음과 같다.

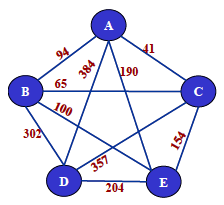
초기 상태는 {x, x, x, x, x}로 주어짐을 알 수 있다.

목표 상태는 모든 지역의 쌍이 달라야 하므로 임의 지역 si에 sj가 인접한 지역이라면 si의 색 ci는 sj의 색 cj는 반드시 달라야만 한다.

후속 함수는 지역의 현재 색을 다른 색으로 바꾸는 함수로

Change( vi,cj) 여기서 i≠j

1. b) 영업사원이 N개의 도시를 한번씩만 방문하고 돌아오는 최단 거리를 찾는 문제이다. 이를 해밀턴 문제 또는 TSP(Traveling Sales Person)라고도 한다.



Y : N 도시의 집합

d(x, y) : 도시 x와 y 사이의 거리. x,y∈Y

상태는 해밀턴 경로 (방문한 도시는 다시 방문하지 않습니다)

X : 상태들의 집합:

X ={(x1, x2, ..., xN) 여기서 n = 1, ..., N + 1이며 xi ∈Y for every I,

xi ≠ xj unless i = 1, j = N + 1}

후속 상태 (x1, x2, ..., xn):

δ (x1, x2, ..., xn) = {(x1, x2, ..., xn, xn+1) | xn+1∈ Y

여기서 모든 1≤ i≤ n에 대하여 xn+1 ≠ xi}

목표 상태의 집합은 길이 N + 1의 모든 상태를 포함

1. c) 선교사 및 식인종 문제 : 3 선교사 및 3 식인종이 강 한 편(왼쪽이라 가정)에 있다. 1개의 보트는 2명을 태울 수 있으며 한 순간이라도 식인종의 수가 선교사의 수보다 많으면 안된다.

모두 안전하게 강을 건너도록 하는 방안은 다음과 같이 형식화 된다.

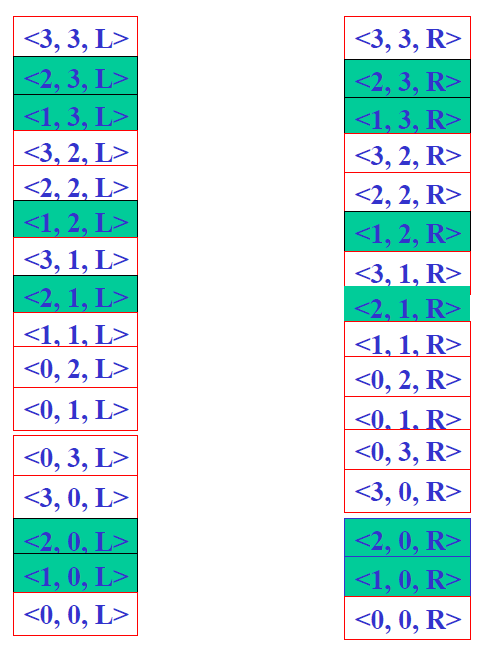
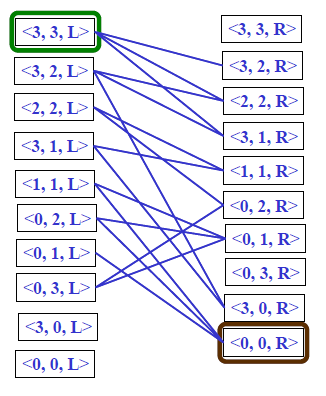
상xo: <M, C, B> 여기서 M은 강 왼편의 선교사의 수, C는 강 왼편의 식인종 수, B는 보트의 위치(강변 왼쪽,강변 오른쪽)

초기 상태: <3,3,L>

목표 상태: <0,0,R>

연산자: <M,C> 여기서 M은 보트 안의 선교사 수이며 C는 보트 안의 식인종 수

유효 연산자: <1,0>, <2,0>,<1,1>, <0, 1>, <0,2>

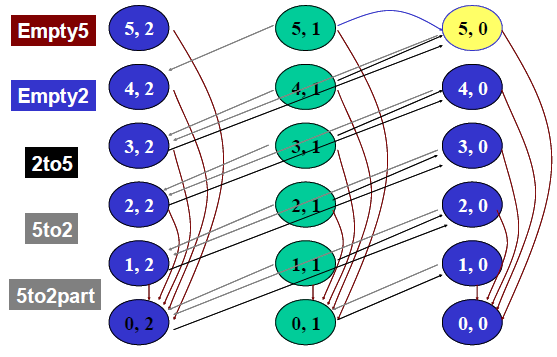
 

1. 채워진 5 리터 용기와 빈 2 리터 용기가 있을 때 목표는 2리터의 용기에 1리터만 채우는 문제이며 상태 공간은 다음과 같다.
   * 형식화

* 상태 = (x, y) 여기서 x는 5 리터 용기에 채워진 양이고 y는 2리터 용기에 채워진 양
* 초기 상태 = (5,0)
* 목표 상태 = (\* 1) 여기서 \*는 0부터 5 까지 어느 수라도 상관 없다.
* 연산자:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Cond.** | **Transition** | **Effect** |
| Empty5 | – | (x,y)*→*(0,y) | 빈5리터 용기 |
| Empty2 | – | (x,y)→(x,0) | 빈 2리터 용기 |
| 2to5 | x *≤* 3 | (x,2)→(x+2,0) | Pour 2리터. into 5리터 |
| 5to2 | x ≥ 2 | (x,0)→(x-2,2) | Pour 5리터 into 2리터 |
| 5to2part | y < 2 | (1,y)→(0,y+1) | Pour partial 5리터 into 2리터 |

* + 상태 전이: 아래 그림은 서로 다른 상태에서 위의 연산자를 사용하여 상태 간의 전환을 보여준 다. Empty2에 해당하는 변환은 단순화를 위해 표시하지 않았다.



검색 트리를 만들고 이 문제에 대한 적절한 검색 전략에 대해 기술한다.

(5,0)는 초기 상태이다.

(0,1) 목표 상태이다.

이 문제에 대한 해결책은 다음의 경로이다

(5,0) - (3,2) - (3,0) - (1,2) - (1,0) - (0,1).

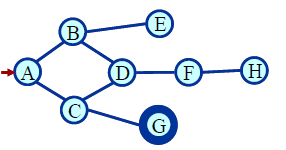
연산자를 사용하면 {5to2, Empty2, 5to2, Empty2, 5to2.}

깊이 탐색은 적절하지 않다

탐색 공간은 그래프이다.

탐색 트리는 상태가 많이 반복된다.

너비 우선 탐색이 적합

1. 다음 그래프를 생각해 보자.

상태 A로부터 시작하여 DFS를 수행한다. 목표 노드 G일 경우 노드의 확장 순서를 보여라. 단 동 순위의 확장 노드가 발생할 경우 알파벳 순으로 한다.

