과제 0919

2315028 김성현

▼ 탐색

- 탐색 : 원하는 것을 찾거나 목표도달에 도움되는 결과를 추구하는 과정. 주어진 데이터 집합에서 원하는 데이터를 찾는 과정.

  특징) 자료구조에 의존함. 목표와 탐색은 밀접한관계가 있음.

  복잡한 문제) 그래프, 트리구조(그래프의 일종으로 방향성 있으며 비순환하는 특징 지님) 사용

▼ 단순한 탐색종류

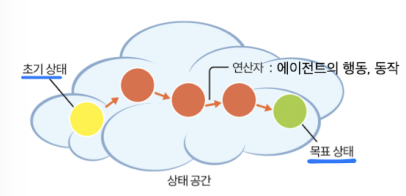
- 무작위탐색(random search) : 랜덤하게 찾는 방법. 체계가 없음. 단순하지만 운이 좋으면 빠르게 찾을 수 있음
- 순차탐색(sequential, linear search) : 순차적으로 탐색하는 방법. 정렬한 뒤에 탐색하면 더 효율적임
- 이진탐색(binary search) : 정렬되어있는 것을 반으로 나누며 중간의 자료를 중심으로 비교하며 찾아가는 방법

▼ 복잡한 문제풀이

- 탐색 : 에이전트가 목표상태의 경로를 형성하는 동작열의 산출하는 계산과정. 에이전트가 어떤 동작을 하였을때의 최선의 경로로 갈 수 있는지 산출하는 계산과정

  활용분야) 지도, 네비, 게임, 인공지능기술

▼ 탐색문제 정의



- 상태공간(state space) : 환경이 가능한 모든 상태들로 이루어진 집합
- 전이모형(transition model) : 각 동작과 동작의 효과들을 서술하는 모형. 규칙, 함수
- 초기상태(initial state) : 에이전트의 최초상태

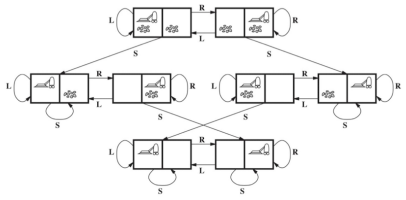
- 목표상태(goal state) : 추구하는 목표상태 (단일, 여러목표집합일 수 있음)

▼ 탐색문제 풀이

1. 목표 형식화(goal formulation) - 목표를 분명히하면 불필요한것들을 감소시킴
2. 문제 형식화(problem formulation) : 목표도달에 필요한 상태와 동작열, 추상화된 모형 구축. 어떻게 도달할지, 어떠한 상태일지
3. 탐색(search) : 목표도달에 필요한 동작열을 발견할때까지 실행해 해를 찾음. (찾을 수없는 경우, 해가 없다는 결론)
4. 실행(execution) : 해답에 필요한 동작들을 실행

▼ 예제

▼ 진공청소기



상태 : 에이전트의 위치(왼/오) x 먼지 존재(유/무) x 먼지위치(왼/오) = 2x2×2=8가지

초기상태 : 어떤 상태든 초기상태로 지정가능

동작 : 왼쪽, 오른쪽, 흡입

전이모형 : ex_가장 왼쪽칸에서 왼쪽으로 이동. 깨끗한 칸에서 흡입하면 아무로 효과없음

목표상태 : 모든 칸이 깨끗한가

▼ 8-puzzle

칸들을 움직여서 목표상태로 만드는 것

상태 : 8개의 타일, 빈칸의 위치

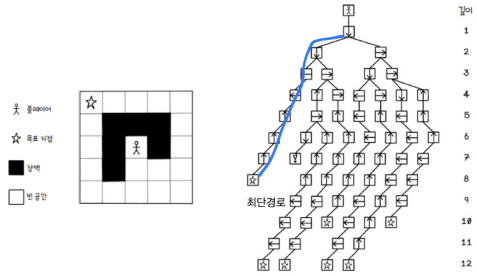
초기상태 : 어떤 상태든 초기상태로 지정가능

동작 : 왼쪽, 오른쪽, 위, 아래

전이모형 : ex_초기상태에서 왼쪽적용시 결과상태는 5번타일과 빈칸의 위치가 바뀜

목표상태 : 현재상태가 목표구성과 일치한가

▼ 미로찾기



상태 : 미로 내 에이전트의 위치, 미로의 구조

초기상태 : 에이전트가 미로 시작점에 있는 상태

동작 : 왼쪽, 오른쪽, 위, 아래(벽이있는 방향으로 이동불가)

전이모형 : ex_에이전트가 왼쪽으로 이동하면 해당위치로 이동. 벽이있는 경우에는 해당동작 무효

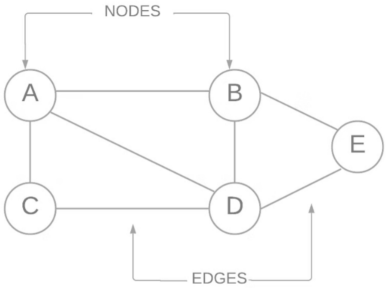
목표상태 : 미로 내의 특정위치에 도달하는 것

▼ 문제해결 성능측정

- 완결성(completeness) : 문제에 해답이 있는경우에 반드시 해답을 찾을 수 있는가
- 최적성(optimality) : 최적의 경로로 목표에 도달할 수 있는가
- 복잡도(complexity) - 빅오 표기법 : 연산횟수 모델링
  - 시간복잡도(time complexity) : 해답 찾는데 걸린 시간
  - 공간복잡도(space complexity) : 탐색 수행시 필요한 메모리의 양

    탐색트리의 최대분기계수(너비), 목표노드의 높이, 트리의 최대깊이

▼ 그래프

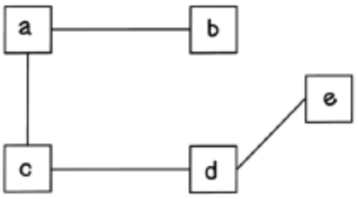


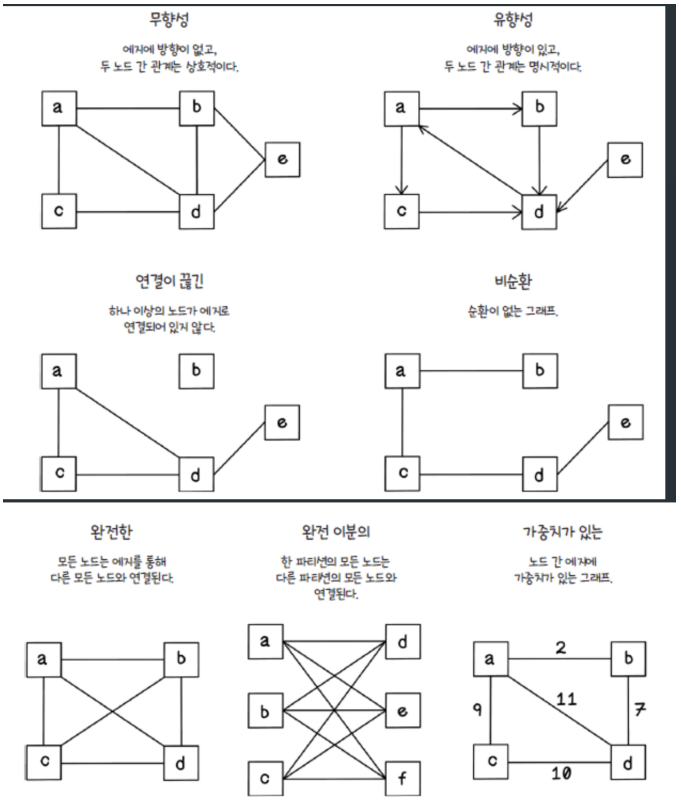
그래프 : 노드와 엣지로 구성된 관계를 나타내는 자료구조

- 표현) G = (V, E)

  V - 노드의 집합, E - 엣지의 집합

- 유형)





	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	0	0
c	1	0	0	1	0
d	0	0	1	0	1
e	0	0	0	1	0

인접행렬

인접리스트

```python
#인접행렬

nodes = ['A', 'B', 'C', 'D', 'E']

adj_matrix = [
    [0, 1, 1, 0, 0], #A
    [1, 0, 0, 0, 0], #B
    [1, 0, 0, 1 ,0], #C
    [0, 0, 1, 0, 1], #D
    [0, 0, 0, 1, 0] #E
]

for i in range(len(adj_matrix)):
    connected_nodes = [nodes[j] for j in range(len(adj_matrix[i])) if adj_matrix[i][j] == 1]
    print(f"{nodes[i]}는 {', '.join(connected_nodes)}와 연결")
```

```
A는 B, C와 연결
B는 A와 연결
C는 A, D와 연결
D는 C, E와 연결
E는 D와 연결
```

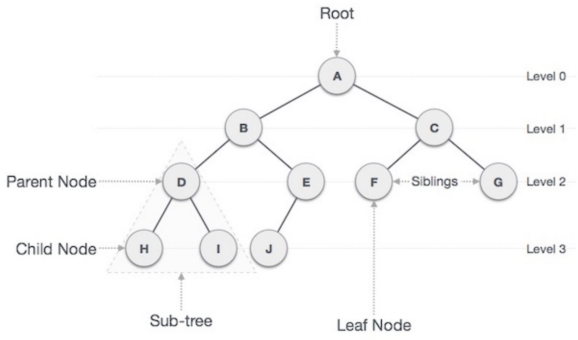
인접행렬

```python
#인접리스트

graph = {
    'A':['B', 'C'],
    'B':['A'],
    'C':['A','D'],
    'D':['C','E'],
    'E':['D']
}

for node in graph:
    print(f"{node}는 {', '.join(graph[node])}와 연결")
```

```
A는 B, C와 연결
B는 A와 연결
C는 A, D와 연결
D는 C, E와 연결
E는 D와 연결
```

인접리스트

▼ 트리



트리 : 노드들이 나무가지처럼 연결된 비선형적이고 계층적인 자료구조

특징) 그래프의 방향이 존재. 순환이 없는 연결그래프의 일종.

트리는 보통 클래스 형태를 사용

```python
#tree구조
#     1
#   /   \
#  2     3
# / \
#4  5

#tree생성을 위한 node
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.right.right = Node(5)

print(root.data, root.left.data, root.left.left.data)
```

```
1 2 4
```

▼ 탐색

▼ 8puzzle만들기

```python
#클래스 설계 (상태, 전이모형, 그외 함수 포함)

class Puzzle:
    def __init__(self, board):
        self.board = board

    def get_new_board(self, i1, i2): #i1, i2인덱스에 위치한 값을 서로 바꿈
        new_board = self.board[:]
        new_board[i1], new_board[i2] = new_board[i2], new_board[i1]
        return Puzzle(new_board)

    #보드의 인덱스
    # 0, 1, 2
    # 3, 4, 5
    # 6, 7, 8

    def expand(self):
        result = []
        i = self.board.index(0) #빈칸의 위치를 파악
        if i not in [0,3,6]: #빈칸이 왼쪽끝 쪽에 위치하지 않으면
            result.append(self.get_new_board(i, i-1)) #왼쪽으로 이동
        if i not in [0,1,2]: #빈칸이 위쪽끝 쪽에 위치하지 않으면
            result.append(self.get_new_board(i, i-3)) #위쪽으로 이동
        if i not in [6,7,8]: #빈칸이 아래끝 쪽에 위치하지 않으면
            result.append(self.get_new_board(i, i+3)) #아래쪽으로 이동
        if i not in [2,5,8]: #빈칸이 오른쪽끝 쪽에 위치하지 않으면
            result.append(self.get_new_board(i, i+1)) #오른쪽으로 이동
        return result

    def __str__(self): #상태를 보기좋게 출력해주는 함수
        return str(self.board[:3])+ "\n" + str(self.board[3:6])+ "\n" + str(self.board[6:9])

    def __eq__(self, other): #보드와 다른보드간에 비교
        return self.board == other.board

    def __ne__(self, other):
        return self.board != other.board
```

트리구조를 이용해 경로탐색 용이하도록 상태공간을 class로 구현

- board : 상태 정보를 담기위한 속성
- get_new_board : 값의 위치를 바꾸는 동작하는 전이모형에 해당하는 함수
- expand : 확장 가능 상태를 생성하는 전이모형에 해당하는 함수
- _str_ : 객체를 출력해 현재 상태를 확인하는 함수
- _eq_ : 두 객체 (현재상태 vs 목표상태)를 비교하기위한 함수

  같은지
- _ne_ : 두 객체 (현재상태 vs 목표상태)를 비교하기위한 함수

  다른지

퍼즐 게임 동작

```
initial_state = [0,1,3,
                 4,2,5,
                 6,8,7]
```

```
initial_puzzle = Puzzle(initial_state)
```

```
print(initial_puzzle)
```

```
[0, 1, 3]
[4, 2, 5]
[6, 8, 7]
```

```
next_puzzle_list = initial_puzzle.expand()

for i, v in enumerate(next_puzzle_list):
    print(f"#{i}\n{v}")
```

```
#0
[4, 1, 3]
[0, 2, 5]
[6, 8, 7]
#1
[1, 0, 3]
[4, 2, 5]
[6, 8, 7]
```