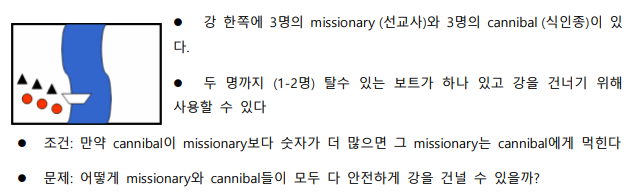
**탐색 문제 정의, 상태와 행동, 탐색 문제 해결 알고리즘의 이해**

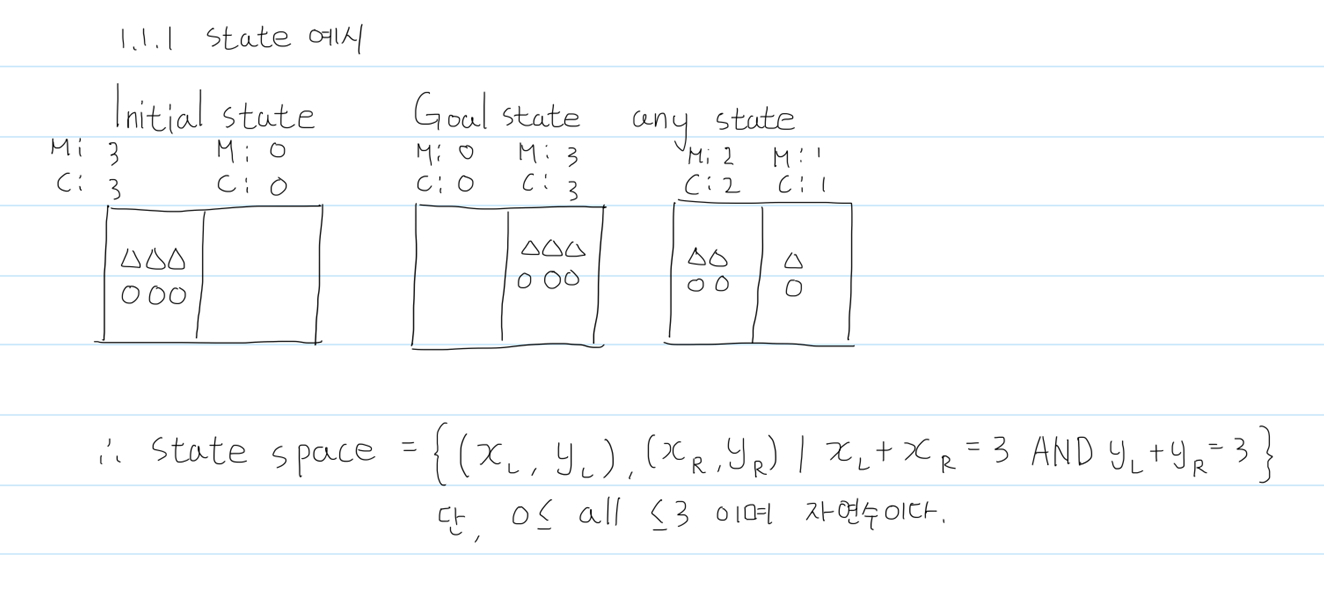
|  |  |  |  |
| --- | --- | --- | --- |
| 과목명 | ISE413\_인공지능시스템(B098-1) | 학생 이름 | 안 상현 |
| 학번 | 201820651 | 지도 교수 | 민 현정 |

1. **Missionaries-Cannibals problem**



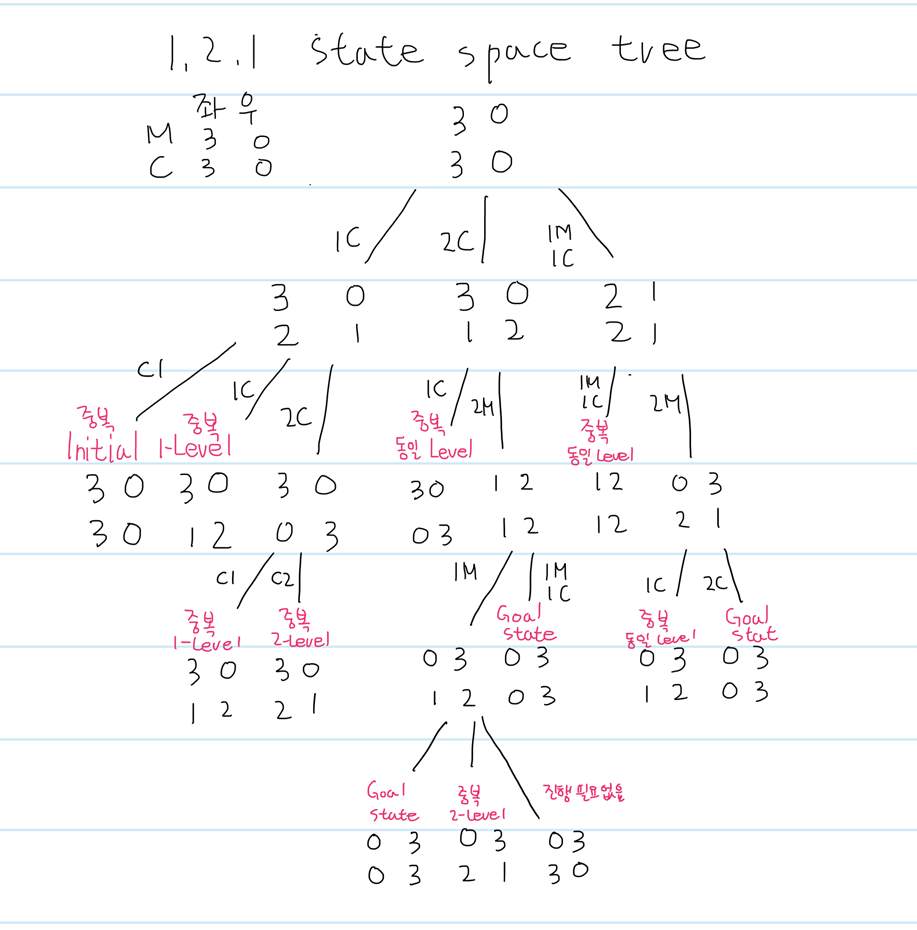
* 1. 이 문제의 state와 state space는 무엇인가?

State는 특정 시간에 주어진 문제가 당면해 있는 상황으로서 이러한 State를 다 모아놓은 집합을 State space라고 한다. State는 초기 상태(Initial state), 문제 해결상태(Goal state)를 포함한 가능한 모든 상태를 나타내기도 하며 예시로 하나씩 들고 state space를 정의한다면 아래(그림 1.1.1)와 같다.



* 1. 이 문제의 state space를 tree 구조로 표현하세요.

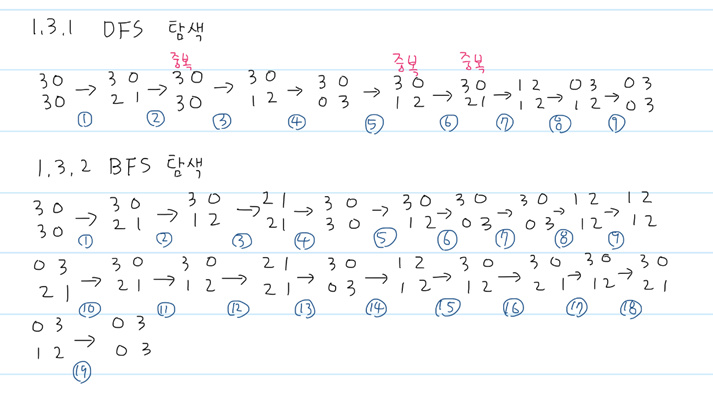
initial state를 root node로 가능한 모든 상태(가능해)를 자식노드로 이어나간 tree는 아래(그림1.2.1)와 같다. 또한 강을 기준으로 Missionary, Cannibal 2행 2열 행렬 (좌, 우)로 표현한다. (중복이 되면 더 이상 진행안함)



* 1. 문제를 해결하기 위한 해 (solution)가 무엇이 있는지 그 과정과 함께 쓰시오. 위의 트리 구조로 표현된 space에서 DFS/BFS는 각각 몇 번 만에 해를 찾는가?

좌측 노드부터 방문하여 우측으로 백트래킹하는 DFS와 좌측노드부터 다음 Level로 넘어가도 좌측으로 진행하는 BFS 기준으로 봤을 때, 결과는 아래와 같다.

DFS는 위 구축된 트리라면 방문한 노드에 한하여 중복에 대한 중지 조건이 없다면 무한 루프에 빠지고 방문한 노드 중복에 대한 중지 조건이 있다면 9번, BFS는 무한루프에 빠지지는 않으나 19번에 해를 찾는다. 결론적으로 DFS가 빨라보이나 중복 탐색에 대한 루틴이 stack이 쌓일 때마다 돌아야 하므로 이 트리구조를 통해 BFS가 Optimal하며 Complete함을 이해할 수 있다.



* 1. 위의 해를 찾기 위한 알고리즘은 무엇인가? 선택한 알고리즘은 complete인가? Optimal 인가?

DFS, BFS 두 탐색을 모두 한 결과 DFS는 무한루프에 빠질 수 있음을 확인할 수 있었다. 그러므로 DFS는 Complete하지도 Optimal하지도 않다. 그러나 DFS 특성상 트리구조가 유한한 노드로 구성되어 있다면, Complete하다고 볼 수 있다. 반대로 BFS는 Complete하면서 Optimal 함을 1.3 문제 풀이로 확인하였다.

* 1. 만약 문제가 3명씩 missionary와 cannibal이 아니라 5명 또는 9명 등 숫자가 늘어날 경우, 어떻게 문제를 해결할 수 있는가? 3명일경우와 달라지는 것이 무엇인가?

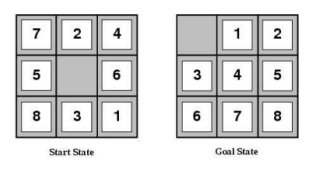
이는 트리의 규모가 달라진다. 보트 탑승인원에 대한 변경이 없으므로 가능해상관없이 나올 수 있는 자식 노드는 최대 5개이며 트리의 규모는 깊이만 증가할 것이다. 그러니 트리의 깊이가 얼마나 증가하는 지 모른다면 세가지 방법을 고려할 수 있다. 첫번째는 기존처럼 BFS를 이용하거나 두번째로는 반복적 깊이 탐색을 통해 DFS의 단점을 보완하여 일정 깊이까지는 DFS, 나머지는 BFS로 탐색하고 마지막으로 양방향 탐색을 통하여 Initial state와 goal state 모두 BFS로 탐색할 수 있다.

* 1. 이 문제를 파이썬으로 프로그램한다면 무엇(어떤 변수, 함수, 정보 등)이 필요한가?

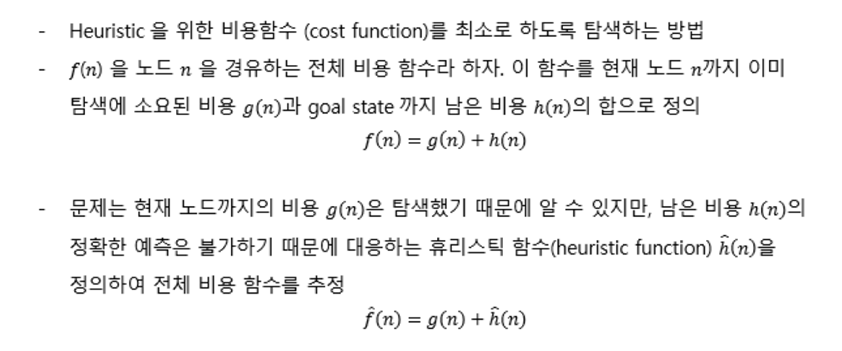
2x2 형태의 행렬 변수(정수형 배열)와 행렬 변수를 담는 스택(트리) BFS 탐색 함수, 가능해 점검 함수, 가능해 생성 함수, 시각화 함수 마지막으로 문제정의, 제약조건에 대한 정보 입력이 필요하다.

* 1. (option) 이 문제를 해결하는 파이썬 프로그램으로 해를 찾도록 구현하시오.

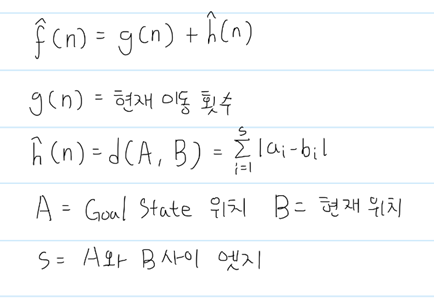
1. **8-puzzle problem**



* 1. 이 8-puzzle 문제 해결을 위한 문제를 정의(formulate) 하세요.
* 빈 칸에 숫자를 넣어 Goal State 모양으로 만들어야 한다.
* 빈 칸을 기준으로 상, 하, 좌, 우에 해당하는 숫자를 빈칸으로 넣을 수 있다.
* 빈 칸에 넣은 숫자의 원래 자리는 빈칸으로 남는다
  1. A\* 알고리즘으로 탐색 문제를 해결한다면 이 문제를 위한 비용 함수를 어떻게 정의할 것인가?



강의자료에 있는 내용을 토대로 g(n)은 탐색에 소요된 비용으로 노드 하나를 지날 때 마다 1씩 증가하고 h\_hat(n)은 휴리스틱 함수를 정의하여 전체 비용 함수를 추정한다. 여기서 현재 숫자의 위치와 Goal state 위치를 맨하탄 거리를 이용하여 추정할 수 있다. 맨하탄 거리를 쓴 이유는 상, 하, 좌, 우로만 이동 가능한 문제 특성 때문이다. 결론적으로 아래와 같이 비용함수를 정의 한다.



* 1. A\* 알고리즘과 위의 DFS 또는 BFS 알고리즘의 차이는 무엇인가?

BFS와 DFS는 Complete하다. A\* 알고리즘 또한 Complete하다. 더 나아가 BFS와 같이 Optimal하다. 그러나 A\* 알고리즘에는 비용함수라는 것이 있다. 비용함수에서 현재 소모한 비용과 앞으로 예상되는 값을 두고 최저의 비용을 목적을 두고 있어 전체 비용이 기하학적으로 증가하는 예측 비용에 대해서는 탐색 후보에 제거되기에 불 필요한 탐색이 줄어든다. 결국 BFS와 DFS는 단편적이고 수동적인 노드탐색을 한다면 A\* 예측비용에 따른 능동적인 노드탐색을 하기에 기하학적으로 본다면 BFS, DFS는 선형함수적 성향이 강하고 A\*는 단조함수적 성향이 강하다. 더 나아가 Greedy algorithm과 비교하자면 동일한 evaluation function을 정의하지만 A\*와 비교하면 미래 예측 비용이 없는 Greedy는 Complete하지 않으며 Optimal하지도 않다.