파이썬을 이용한
루미큐브 문제 해결

-알고리즘 연구반 프로젝트-

Solving Rummikub Problem by Python

2020년 02월 05일

선린인터넷고등학교
알고리즘 연구반
채하늘, 최보민

목차

제 1장 서론

1. 루미큐브 보드게임 규칙

루미큐브(Rummikub)는 각각 검정, 빨강, 파랑, 주황 4가지의 색깔을 가진 1~13까지의 숫자카드 한 쌍의 카드가 존재하고 조커 2장이 존재한다 (이번 프로젝트에서 조커는 사용하지 않음, 총 104장의 카드)

참여자가 각각의 턴을 가지고 진행한다. 참여자는 손에 있는 카드와 테이블에 있는 카드를 조합하여 모든 카드를 손을 비우면 이기는 게임이다.

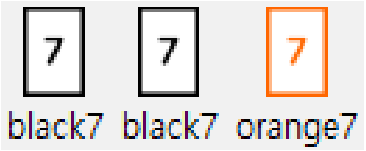
조합하여 낼 수 있는 방법은 두 가지이다. 첫 번째는 같은 색깔 연속된 숫자 3개 이상, 두 번째는 같은 숫자 다른 색깔 3개로 조합할 수 있다.

다음은 조합할 수 있고, 없는 예를 들겠다.

<같은 색깔 연속된 숫자 3개 이상>

<잘못된 예>

<다른 색깔 같은 숫자 3개 이상>



<잘못된 예>

2. 프로젝트 기획 동기

남녀노소 모두에게 인기가 있는 루미큐브가 최근에 들어 더욱더 인기를 끌고 있다. 또한, 각자 평소에 루미큐브에 대해 관심이 많았기 때문에 수학적 계산을 활용하여 루미큐브의 최적해를 컴퓨터의 연산을 통해 알아낼 수 있는지 궁금했다. 이러한 이유로 "완전 탐색으로 루미큐브 문제의 최적해를 구할 수 있다"라는 가설을 세우고 프로젝트를 진행하였다. 그리고 프로젝트를 진행할 때 막힘을 최대한 줄일 수 있을까 고민하였다. 그 결과 코드가 직관적이고, GUI 구현이 다른 언어에 비해 비교적 간단한 파이썬을 이용하여 프로젝트를 진행하면 좋을 것 같다는 의견에 파이썬을 통해 구현하기로 하였다.

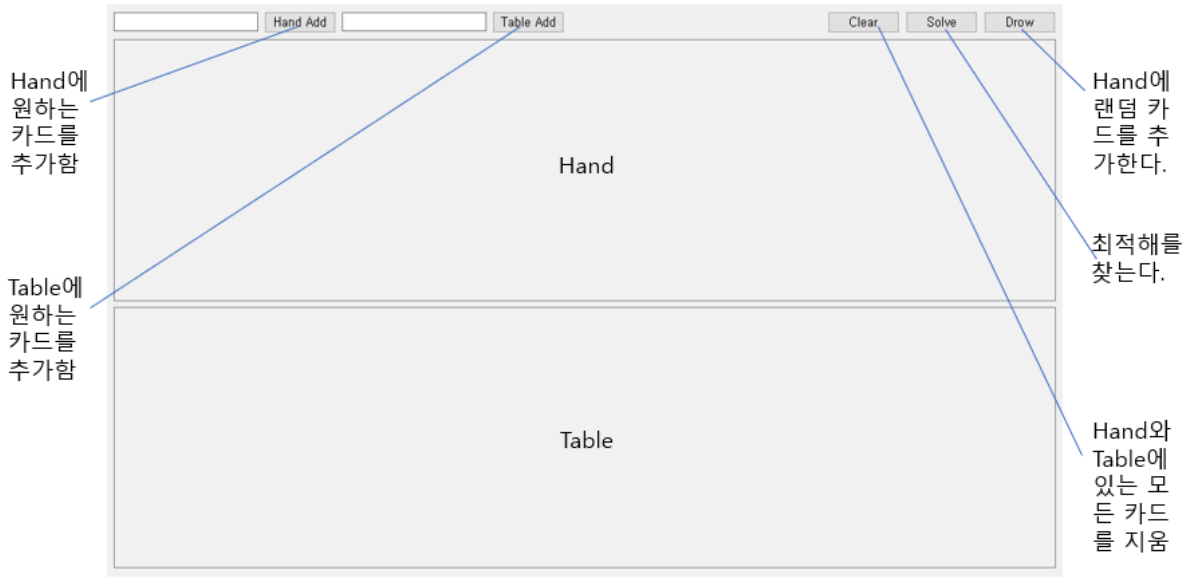
3. 루미큐브 문제

루미큐브 문제(Rummikub problem)는 NP-Hard 문제이다. 따라서 이미 알려진 방식으로는 다항 시간에 풀 수 없어, 최적해를 구하기 위해선 모든 경우의 수를 확인해보아야 한다. 정수 선형 계획법(Integer Linear Programming)을 통해 모델링 하여 문제를 푸는 데에 사용할 수 있으나, 이 역시 NP-Hard에 속하여 다항 시간 내에 풀 수 있진 않다. 이번 프로젝트에서는 모든 경우의 수를 탐색하는 완전 탐색 방식으로 구현하였다.

제 2장 본론

1. User Interface

다음은 프로젝트를 시각적으로 보여주는 UI이다.



<User Interface>

카드는 각각의 추가 버튼 왼쪽에 있는 box에 카드들의 색+숫자(ex red13) 형태를 쉼표(,)로 구분하여 입력하여 추가할 수 있다. 추가된 카드들은 Hand와 Table 중 알맞은 영역 추가된다.

다음은 Solve 버튼을 눌렀을 때 나타나는 최적해이다.

2. 프로그램 동작 원리

최적해를 찾는 과정은 다음과 같은 식에 만족하는 i의 조합을 찾아내도록 구현하였다.

$$max \sum \sum v_{i_j}$$

i *현재 타일로 만들 수 있는 모든 조합*

i_j *i의 속한 타일의 j번째 원소*

v_x *타일 x의 점수*

문제를 해결하는 함수는 다음과 같은 형태로 되어있다.

1. 현재 손패와 테이블에 있는 카드들을 받아온다.

2-1. 현재 손패가 비어 있다면 4로 간다.

2-2. 현재 손패가 비어 있지 않다면 현재 손패와 테이블 카드로 만들 수 있는 모든 조합을 구한다.

2-2-1. 구한 조합이 없다면 4로 간다.

2-2-2. 구한 조합이 있다면 3으로 간다.

3. 조합 리스트에서 각각의 조합에 대해 다음 동작을 실행한다.

a. 해당 조합에 해당하는 카드를 제외한 손패와 테이블을 구한다.

b. 구한 손패와 테이블을 통해 1을 실행시키고 점수와 정답을 구한다.

c. 해당 점수가 현재 탐색한 리스트에서 가장 크다면, 정답을 갱신한다.

c-1. 조합이 갱신되었다면, 갱신된 정답에 현재 조합을 추가한다.

4. 점수와 정답을 반환한다.

위 함수에서 조합을 구하는 방식은 아래와 같다.

1. 손패와 테이블에 있는 카드에서 중복을 제외한다.

2. 같은 색깔의 연속된 카드들의 조합들을 구하기 위해 전체 카드를 같은

색깔끼리 묶는다.

3. 각각의 묶음에 대해 다음과 같은 동작을 실행한다.
    a. 묶음을 정렬하고, index-value가 같은 카드끼리 배열로 묶는다.
    b. 새로운 각각의 배열에 대해 다음과 같은 동작을 실행한다.
    c. 각각의 배열의 길이가 3 이상 5이하인 모든 하위배열을 구한다.
4. 같은 숫자의 다른 색 카드들의 조합들을 구하기 위해 전체 카드를 같은 숫자끼리 묶는다.
5. 각각의 묶음에 대해 다음과 같은 동작을 실행한다.
    a. 길이가 3이상 묶음의 모든 하위 시퀀스를 구한다.
6. (3)과 (5)에서 구한 조합 합친다.

위에서 같은 색, 연속된 카드 조합을 구할 때 길이가 5 이하인 조합들만 구하는 이유는 길이가 6 이상인 조합들은 다시 길이가 3,4,5인 조합들로 분할할 수 있기 때문이다.

제 3장 결론

1. 느낀 점

평소에 가볍게 즐기던 보드게임을 수학적으로 접근하여, 최적해를 갖는 공식을 생각하여 계산해낸다는 점이 굉장히 흥미로웠다. 특히 이 과정에서 선형 계획법(Linear Programming)이라는 모델링 방식을 알게 되었다. 이는 선형 조건을 만족시키면서 목적 함수의 최적해를 찾는 것인데, 실제 루미큐브 문제는 선형 계획법으로 모델링 하여 접근한다는 점이 놀라웠다. 또한, 시뮬레이터를 GUI로 구현하는 점에서 Qt 라이브러리에 대해 공부하고, GUI에 대해 조금 더 자세히 이해할 수 있는 계기가 되었다. 그리고 개인 프로젝트가 아닌, 팀 프로젝트로 진행함으로써 깊진 않지만 협업을 경험할 수 있었고, 서로 무엇을 추가하면 좋을지, 어떤 것을 수정해야 할지 의논하는 것은 매우 재미있게 다가왔다.