Assignment 1 Report
-Minesweeper-

2014147550 컴퓨터과학과
강효림

1. 개요

지뢰찾기의 특정 state에서, covered cell에서 지뢰가 있는지 없는지, 아니면 주어진 정보만으로는 결정될 수 없는지를 판단하는 알고리즘을 본다. 기본적으로 backtracking을 이용한 완전탐색을 기본으로 하되, 적절한 prunning을 거쳐 성능을 향상한다.

2. 파일 구성

가. Main

main함수가 존재하는 class로, main 외에도 완전탐색의 기반이 되는 함수 f와 전체 게임판의 적법성을 점검하는 satisfyCheck함수가 있다.

나. Cell

게임판의 한 칸, 즉 Cell의 특성을 정의하는 클래스이다. 주요 함수로는 현재 게임판이 그 특정 Cell의 status(input에서 주어지는 -1~8사이의 정수)에 비추어 적법한지를 판별해주는 satisfy함수와 인접한 Cell의 배열을 반환해주는 getAdjacentCell함수가 있다.

3. 주요 함수

가. satisfyCheck

```
10    public static boolean satisfyCheck(int rowSize, int colSize, Cell[] cellList) {
11        boolean flag;
12        for (Cell c : cellList) {
13            flag = c.satisfy(rowSize, colSize, cellList);
14            if (!flag) {
15                return false;
16            }
17        }
18        return true;
19    }
20
```

게임판의 모든 cell에 대해 그 cell을 기준으로 판의 적법성을 검사하는 함수로, 반환되는 값은 당해 판의 (적어도 아직)위법하지 않음에 대한 결과를 반환한다.

나. satisfy

```
public boolean satisfy(int rowSize, int colSize, Cell[] cellList) {
    if (status < 0) {
        return true;
    }
    int mineNum = 0;
    int possible = 0;
    Cell[] adj = getAdjacentCell(rowSize, colSize, cellList);
    for (Cell c : adj) {
        if (c==null) {
            continue;
        }
        if (c.getMine() == -1) {
            possible++;
        } else if (c.getMine() == 1) {
            mineNum++;
        }
    }
    return mineNum <= status && mineNum + possible >= status;
}
```

특정 Cell의 status에 비추어 게임판의 적법성(정확하게는 아직 위법하지 않음)을 검사하는 함수로, possible이라는 'mine이 될 수 있는 가능성이 있는', 즉 아직 정해지지 않은 인접한 cell도 고려하여 결과를 반환한다.

다. f

```
23    public static void f(int rowSize, int colSize, Cell[] cellList) {
24        Cell c = rd.pollLast();
25        if (c==null) {
26            for (Cell tmp : dd) {
27                tmp.setAnswer();
28            }
29            return;
30        }
31        dd.offerLast(c);
32        c.setMine(0);
33        if(satisfyCheck(rowSize, colSize, cellList)) {
34            f(rowSize, colSize, cellList);
35        }
36        c.setMine(1);
37        if(satisfyCheck(rowSize, colSize, cellList)) {
38            f(rowSize, colSize, cellList);
39        }
40        c.setMine(-1);
41        dd.pollLast();
42        rd.offerLast(c);
43        return;
44    }
```

완전탐색 및 prunning을 구현하는 함수로, rd (readyDeque)에서 cell을 하나 꺼내 dd(doneQueue)에 넣고 그 cell의 mine을 0 ,1로 각각 바꿔본다. 그 때마다 satisfyCheck함수로 판의 적법성을 검사한다. **적법하다면 재귀적으로 같은 함수를 호출**하고, 만일 rq가 비어있다면 적법한 mine 배치 중 하나가 성공적으로 나온 것이고, 모든 cell은 dd에 있을 것이므로, dd를 for each 문을 통하여 순회하며 현재 mine 배치를 기반으로 거기의 Cell의 ans(mine의 유무, 결정할 수 없음을 표현하는 int형 field)를 설정한다. 마지막엔 Cell의 mine을 -1(결정되지 않음을 의미)로 바꾸고 dd에서 그 Cell을 제거하고 다시 rq에 넣음으로써 원상복구 시킨 뒤 리턴된다.

라. getAdjacentCell

이름처럼 특정 Cell에 인접한 Cell(최소 3개~최대 8개)의 배열을 반환한다. 코드는 케이스 처리로써 길고 무의미하므로 생략한다.

4. 주요 로직

```
1  package assignment01;
2
3⊕ import java.io.*;
5
6  public class Main {
7      static Deque<Cell> rd = new ArrayDeque<>();
8      static Deque<Cell> dd = new ArrayDeque<>();
```

앞서 살피었듯 rd는 재귀함수 f의 작업을 기다리는 덱, dd는 작업중인 덱을 나타낸다. 파일 입력을 받아. 게임판 내 모든 Cell들을 cellList에 저장하고(게임판 역할을 하게 된다), status가 -1인 Cell들 중에 인접한 Cell 중 단 하나라도 숫자가 있는 Cell이 있다면 그 Cell들은 targetCell로써, rd로 삽입되게 된다.(mine의 존재가 정해질 수 있는 가능성을 가진 예비집합) 그리고 그 덱을 기반으로 위 f함수가 돌게되고, 이에 따라 rd의 cell들은 모두 ans에 특정 값들을 갖게 된다.

```
List<Cell> mineExist = new ArrayList<>();
List<Cell> mineNotExist = new ArrayList<>();


for (Cell c : rd) {
    int ans = c.getAnswer();
    if (ans == 1) {
        mineExist.add(c);
    } else if (ans == 0) {
        mineNotExist.add(c);
    }
}
```

이후 rd를 순회하며 ans를 조회하고, 새로운 리스트 mineExist, mineNotExist를 생성한다. 이를 바탕으로 format에 맞추어 파일에 출력한다.

5. 결론

그러나 prunning이 들어갔다 하더라도 완전탐색의 특성상 지수적인 시간복잡도를 갖는다. targetCell이 지나치게 많아진다면 본 알고리즘으로는 현실적인 시간 내에 답을 낼 수 없다.