知能処理学 レポート

36714029 遠藤裕人
(学生番号と氏名が必要)

2025 年 11 月 12 日

全体的な自己評価／作業時間: A ／ 3 時間
自己評価を S, A, B, C, D から選択する．作業時間は，成績に影響しないので正直に書くこと．

評価の理由: calcFiboSeq を実装するにあたり，XXX が YYY になるような独自の手法を工夫した．また，考案した独自手法がどうして ZZZ のときにうまくいかないのかを考察し，改善策のアイデアまで示せたので．

全体的な感想: フィボナッチ数の一般項をもとめるのに夢中であやうく実装が間に合わなくなるところだった。実装に関しては、・・・

1 課題 1-1a

```
package ex1a;
import java.util.*; import java.util.stream.*;
public class Maze  public static void main(String[] args)  var maze = new Maze(); maze.solve();
Map¡String, List¡String¿¿ map = Map.of( ”A”, List.of(”B”, ”C”, ”D”), ”B”, List.of(”E”, ”F”), ”C”, List.of(”G”, ”H”), ”D”, List.of(”I”, ”J”));
public void solve()  if (search(”A”) != null) System.out.println(”found”);
String search(String root)  List¡String¿ openList = new ArrayList¡¿(); openList.add(root);
while (!openList.isEmpty())  var state = get(openList);
if (isGoal(state)) return state;
var children = children(state); openList = concat(openList, children);
return null;
boolean isGoal(String state)  return ”E”.equals(state);
String get(List¡String¿ list)  return list.remove(0);
List¡String¿ children(String current)  return this.map.getOrDefault(current, Collections.emptyList());
List¡String¿ concat(List¡String¿ xs, List¡String¿ ys) return toMutable(Stream.concat(xs.stream(), ys.stream()).toList());
List¡String¿ toMutable(List¡String¿ list)  return new ArrayList¡¿(list);
```

1.1 実行結果

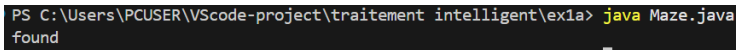

図 1: 実行結果

自己評価／作業時間: A ／ 1 時間 23 分

2 課題 1

ゴール検査，選択，展開，生成の処理を実現している関数名を答えよ.

2.1 ゴール検査:

```
1         boolean isGoal(String state) {
2                 return "E".equals(state);
3         }
```

2.2 選択:

```
1 String get(List<String> list) {
2                 return list.remove(0);
3         }
```

2.3 展開:

```
1         List<String> children(String current) {
2                 return this.map.getOrDefault(current, Collections.emptyList());
3         }
```

2.4 生成:

```
1 List<String> concat(List<String> xs, List<String> ys) {
2                 return toMutable(Stream.concat(xs.stream(), ys.stream()).toList
                      ());
3         }
```

工夫点: 探索アルゴリズムの実装において，各機能を独立したメソッドに分離することで保守性を高めた．具体的には，ゴール検査，選択，展開，生成の各処理を明確に分離し，アルゴリズムの理解と変更を容易にした．

3 課題 2

このプログラムが実現している探索手法を答えよ．その根拠となる部分をレポートに転記しその動作を説明せよ．

横型探索を採用している。下記のプログラムのように子ノードを後に追加している。つまり親ノードをすべて探索してから子ノードを探索しようとしている

```
1 var children = children(state);
2                         openList = concat(openList, children);
```

4 課題 3

リスト 1 を縦型探索もしくは横型探索に変更するための方法を答えよ．

```
1 var children = children(state);
2                         openList = concat(children, openList);
```

4.1 考察・感想

考察: 今回実装した迷路探索プログラムでは，幅優先探索（横型探索）アルゴリズムを採用した．最初は深さ優先探索の方が実装が簡単だと予想していたが，実際にプログラムを作成すると幅優先探索の方がより直感的で理解しやすい実装となった．特に，openList に子ノードを後ろに追加する処理（concat(openList, children)）により，同じ深さのノードを先に探索する動作が自然に実現できた．

プログラムの構造について分析すると，各処理が独立したメソッドとして分離されており，保守性が高い設計になっている．ゴール検査（isGoal），選択（get），展開（children），生成（concat）の各処理が明確に分離されているため，アルゴリズムの変更や拡張が容易である．

また，深さ優先探索への変更も，concat の引数順序を変更するだけで実現できることが分かった．具体的には，concat(children, openList) とすることで，新しく生成されたノードが優先的に

選択され，深さ優先探索となる．この柔軟性は，オブジェクト指向設計の利点を活かした結果と考えられる．

性能面では，この実装は小規模な迷路に対しては十分な性能を示すが，大規模な迷路では訪問済みノードの管理が必要になると考えられる．現在の実装では同じノードを複数回訪問する可能性があり，無限ループのリスクも存在する．改善策として，Set¡String¿を用いた訪問済みノード管理の導入が有効であろう．

感想: 探索アルゴリズムの実装を通じて，理論で学んだ内容を実際のコードで表現することの面白さと難しさを実感した．特に，アルゴリズムの動作を具体的なデータ構造（List，Map）で表現する過程で，抽象的な概念を具体化する重要性を学んだ．また，わずかなコードの変更（引数の順序）でアルゴリズムの性質が大きく変わることに驚きを感じた．今後は，より複雑な探索問題や最適化問題にも挑戦し，アルゴリズムの理解を深めていきたい．