知能処理学 演習指示書*

演習 1-2 探索

大囿忠親

2023/11/7

1　概要

演習 1-2 は、問題固有の情報を利用する探索プログラムに関する演習である。前回に比べると格段に難易度が高いため、計画的に取り組むこと。試行錯誤の繰り返しにより成長が期待できる。

1.1　例題: 宣教師と人食い人

リスト 1〜リスト 8 は、次の問題を表すプログラムである。特に、リスト 8 はこの問題を定義している。

宣教師と人食い人

3 人の宣教師と 3 人の人食い人が川岸にいる。2 人乗りの舟を使って、向こう岸まで渡ろうとしている。ところが、それぞれの岸にいる人食い人の人数が宣教師の人数を超えると、人食い人は宣教師を食べてしまう。全員が生きたまま向こう岸に渡るための手順を答えよ。

2　課題

以下のすべての課題に取り組むこと。**写経**が指示されている場合は、次のいずれかの作業を実施せよ。プログラミングが得意でない場合は、2 の作業を行うことを強く推奨する。今回のプログラムは、前回のプログラムに基づいており、大きな違いはない。前回との違いに注意するとよい。写経において、前回のコードを再利用してもよい。

1. 対象のプログラムを演習指示書からコピペして、重要箇所をコメントせよ。
2. プログラムの意味を考えながらキーボードで一文字ずつプログラムを打ち込むこと。理解した内容をプログラム中にコメントとして記載すること。作業時間を必ず計測し、レポートにて報告せよ。写経の作業時間とは、入力を開始してから、プログラムを正常に実行するまでの時間を意味する。

課題 1-2a　（基礎・必須 ／ ex12a）

リスト 1〜リスト 8 を写経せよ。パッケージが 2 つに分けられている点に注意すること。

演習 1-1 のプログラムにコストを導入するための拡張点を説明せよ。特に、経路コスト g やヒューリスティック関数 h が、どのように実装されているのかを答えよ。また、どのように探索プログラムに評価関数を与えているのかを説明せよ。提出先ディレクトリを `ex12a` とする。

課題 1-2b　（応用・必須 ／ ex12b）

これまで作成したプログラムを改良して、8 パズルを解くプログラムを作成よ。最小コスト優先探索、最良優先探索、および A*アルゴリズムを用いて解け。各手法の性能を調べよ。性能として、訪問ノード数、オープンリスト最大長、実行時間（ミリ秒）を用いること。プログラムが 3 分間で停止しない場合は、測定不能とせよ。提出先ディレクトリを `ex12b` とする。

8 パズルの例

```
初期状態      ゴール状態
 2 3 5         1 2 3
 7 1 6         4 5 6
 4 8 0         7 8 0   （0 は空白マスを表す）
```

（ヒント）

- プログラムに自信がなければ、3×3 の 2 次元配列により盤面を表現せよ。そうでなければ、1 次元配列で表現せよ。
- アクションを、空白とそれに隣接するパネルとの交換と考えるとよい。空白に隣接するパネルは上下左右の 4 通りであるから、それらを表す 4 つのアクションを作成すればよい。ただし、状態によっては実行不可能なアクションがあることを考慮する必要がある。
- まずは、ヒューリスティック関数として、目標状態と異なるパネルの枚数を試すとよい。

課題 1-2c　（応用・必須 ／ ex12c）

課題 1-2b で作成したプログラムを改良して、繰り返し回避を実現せよ。繰り返し回避の導入に伴う各手法の性能改善について報告せよ。最小コスト優先探索、最良優先探索、および A*アルゴリズムを対象とする。性能として、訪問ノード数、オープンリスト最大長、**クローズドリスト最大長**、実行時間（ミリ秒）を用いること。プログラムが 3 分間で停止しない場合は、測定不能とせよ。提出先ディレクトリを ex12c とする。

課題 1-2d　（発展・必須 ／ ex12d）

これまで作成したプログラムを改良し、「難しい 8 パズルの例」を解け。「難しい 8 パズルの例」を解くために、異なる 3 種類のヒューリスティック関数 h'_1, h'_2, h'_3 を比較し、ヒューリスティック関数の違いが性能に与える影響を調べよ。h'_2 を目標状態と異なるパネルの枚数とし、$h'_1 \leq h'_2 \leq h < h'_3$ とすること。任意の状態 n について、$h'_1(n) \leq h'_2(n)$ が期待されるが、必ずしもこれを保証する必要はなく、不等式を経験的に満たしていればよい。すなわち、稀に $h'_1(n) > h'_2(n)$ となってもよい。

レポートには、h'_1, h'_2, h'_3 を数式等で簡潔に表現した上で、これらの大小関係について説明すること。例外なく不等式を満たしているか否かを示すこと。性能として、訪問ノード数、オープンリスト最大長、（もし使っていれば）クローズドリスト最大長、実行時間（ミリ秒）を比較し報告すること。提出先ディレクトリを ex12d とする。

難しい 8 パズルの例

初期状態

```
8 6 7
5 0 4
2 3 1
```

難しすぎてプログラムを完成できなくとも、試行錯誤の過程を課題毎に詳細に説明すれば、それも認める。何をやろうとして何が達成できなかったのかを記載しつつ、振り返ること。

リスト 1: search/World.java

```
package search;

import java.util.*;

public interface World extends Cloneable {
  boolean isValid();
  boolean isGoal();
  List<Action> actions();
  World successor(Action action);
}
```

リスト 2: search/Action.java

```
package search;

public interface Action {
  float cost();
}
```

リスト 3: search/Heuristic.java

```
package search;

public interface Heuristic {
  float eval(State s);
}
```

リスト 4: search/State.java

```
package search;

import java.util.*;

public class State {
  State parent;
  World world;
  Action action;
  float cost;

  State(State parent, Action action, World child) {
    this.parent = parent;
    this.action = action;
    this.world = child;

    if (parent != null && action != null) {
      this.cost = parent.cost + action.cost();
    } else {
      this.cost = 0;
    }
  }

  public State parent() { return this.parent; }
  public World world() { return this.world; }
  public Action action() { return this.action; }
  public float cost() { return this.cost; }

  public boolean isGoal() {
    return this.world.isGoal();
  }

  List<State> children() {
    return this.world.actions().stream()
    .map(a -> new State(this, a, this.world.successor(a)))
    .filter(s -> s.world.isValid())
    .toList();
  }

```

```
39   public String toString() {
40     var str = this.world.toString() + "@" + this.cost;
41     if (this.action != null) {
42       str = String.format("%s (%s)", str, this.action);
43     }
44     return str;
45   }
46
47 }
```

リスト 5: search/Evaluator.java

```
1  package search;
2
3  public interface Evaluator {
4    public static Evaluator minCost() { return new MinCostEvaluator(); }
5    public static Evaluator bestFirst(Heuristic h) { return new BestFirstEvaluator(h); }
6    public static Evaluator aStar(Heuristic h) { return new AStarEvaluator(h); }
7
8    float f(State s);
9
10   default float g(State s) {
11     return s.cost;
12   }
13 }
14
15 class MinCostEvaluator implements Evaluator {
16   public float f(State s) {
17     return g(s);
18   }
19 }
20
21 class BestFirstEvaluator implements Evaluator {
22   Heuristic h;
23
24   BestFirstEvaluator(Heuristic h) {
25     this.h = h;
26   }
27
28   public float f(State s) {
29     return h.eval(s);
30   }
31 }
32
33 class AStarEvaluator implements Evaluator {
34   Heuristic h;
35
36   AStarEvaluator(Heuristic h) {
37     this.h = h;
38   }
39
40   public float f(State s) {
41     return g(s) + h.eval(s);
42   }
43 }
```

リスト 6: search/InformedSolver.java

```
1  package search;
2
3  import java.util.*;
4  import java.util.stream.*;
5
6  public class InformedSolver {
7    Evaluator eval;
8    long visited = 0;
9    long maxLen = 0;
10
11   public InformedSolver(Evaluator eval) {
12     this.eval = eval;
13   }
14
15   public void solve(World world) {
16     var root = new State(null, null, world);
17     var goal = search(root);
18
19     if (goal != null)
20       printSolution(goal);
21
22     System.out.printf("visited: %d, max length: %d\n", this.visited, this.maxLen);
23   }
24
25   State search(State root) {
26     var openList = toMutable(List.of(root));
27
28     while (openList.isEmpty() == false) {
29       var state = get(openList);
30       this.visited += 1;
31
32       if (state.isGoal())
33         return state;
34
35       var children = state.children();
36       openList = concat(openList, children);
37
38       sort(openList);
39
40       this.maxLen = Math.max(this.maxLen, openList.size());
41     }
42
43     return null;
44   }
45
46   State get(List<State> list) {
47     return list.remove(0);
48   }
49
50   List<State> concat(List<State> xs, List<State> ys) {
51     return toMutable(Stream.concat(xs.stream(), ys.stream()).toList());
52   }
53
54   void sort(List<State> list) {
55     list.sort(Comparator.comparing(s -> this.eval.f(s)));
56   }
57
58   List<State> toMutable(List<State> list) {
59     return new ArrayList<State>(list);
60   }
61
62   void printSolution(State goal) {
63     var list = new ArrayList<State>();
64
65     while (goal != null) {
66       list.add(0, goal);
67       goal = goal.parent;
68     }
```

```
69
70      list.forEach(System.out::println);
71    }
72  }
```

リスト 7: ex2a/Main.java

```java
package ex2a;

import search.*;

public class Main {
  public static void main(String[] args) {
    var h = new MisCanHeuristic();

    var minCostSearch = new InformedSolver(Evaluator.minCost());
    minCostSearch.solve(new MisCanWorld(3, 3, 1));

    var bestFirstSearch = new InformedSolver(Evaluator.bestFirst(h));
    bestFirstSearch.solve(new MisCanWorld(3, 3, 1));

    var aStarSearch = new InformedSolver(Evaluator.aStar(h));
    aStarSearch.solve(new MisCanWorld(3, 3, 1));
  }
}
```

リスト 8: ex2a/MisCanProblem.java

```java
package ex2a;

import java.util.*;

import search.*;

class MisCanAction implements Action {
  int missionary;
  int cannibal;
  int boat;

  static List<Action> all = List.of(
      new MisCanAction(-2, 0, -1),
      new MisCanAction(-1, -1, -1),
      new MisCanAction(0, -2, -1),
      new MisCanAction(-1, 0, -1),
      new MisCanAction(0, -1, -1),
      new MisCanAction(+2, 0, +1),
      new MisCanAction(+1, +1, +1),
      new MisCanAction(0, +2, +1),
      new MisCanAction(+1, 0, +1),
      new MisCanAction(0, +1, +1));

  MisCanAction(int missionary, int cannibal, int boat) {
    this.missionary = missionary;
    this.cannibal = cannibal;
    this.boat = boat;
  }

  public float cost() {
    return 1;
  }

  public String toString() {
    var dir = this.boat < 0 ? "right" : "left ";
    var m = Math.abs(this.missionary);
    var c = Math.abs(this.cannibal);
    return String.format("move (%d, %d) to %s", m, c, dir);
  }
}

class MisCanWorld implements World {
  int missionary;
  int cannibal;
  int boat;

  public MisCanWorld(int missionary, int cannibal, int boat) {
```

```
48      this.missionary = missionary;
49      this.cannibal = cannibal;
50      this.boat = boat;
51    }
52
53    public MisCanWorld clone() {
54      return new MisCanWorld(this.missionary, this.cannibal, this.boat);
55    }
56
57    public boolean isValid() {
58      if (this.missionary < 0 || this.missionary > 3) return false;
59      if (this.cannibal < 0 || this.cannibal > 3) return false;
60      if (this.boat < 0 || this.boat > 1) return false;
61      if (this.missionary > 0 && this.missionary < this.cannibal) return false;
62      if (3 - this.missionary > 0 && 3 - this.missionary < 3 - this.cannibal) return false;
63      return true;
64    }
65
66    public boolean isGoal() {
67      return this.missionary == 0 && this.cannibal == 0;
68    }
69
70    public List<Action> actions() {
71      return MisCanAction.all;
72    }
73
74    public World successor(Action action) {
75      var a = (MisCanAction) action;
76      var next = clone();
77      next.missionary += a.missionary;
78      next.cannibal += a.cannibal;
79      next.boat += a.boat;
80      return next;
81    }
82
83    public String toString() {
84      return String.format("(%d, %d, %d)", this.missionary, this.cannibal, this.boat);
85    }
86  }
87
88  class MisCanHeuristic implements Heuristic {
89    public float eval(State s) {
90      var w = (MisCanWorld) s.world();
91      return w.missionary;
92    }
93  }
94
95  class BetterMisCanHeuristic implements Heuristic {
96    public float eval(State s) {
97      var w = (MisCanWorld) s.world();
98      return w.missionary + w.cannibal;
99    }
100 }
```