# 知能プログラミング演習II 課題1

グループ8
29114003 青山周平

2019年10月15日

**提出物**　rep1, search.java, searchGUI.java

**グループ**　グループ8

**メンバー**

| 学生番号 | 氏名 | 貢献度比率 |
|---|---|---|
| 29114003 | 青山周平 | NoData |
| 29114060 | 後藤拓也 | NoData |
| 29114116 | 増田大輝 | NoData |
| 29114142 | 湯浅範子 | NoData |
| 29119016 | 小中祐希 | NoData |

## 1　課題の説明

**必須課題 1-1** Search.javaの状態空間におけるパラメータ（コストや評価値）を様々に変化させて実行し，各探索手法の違いを説明せよ．　具体的には，変化させたパラメータと探索結果（最短パス探索の成否，解を返すまでのステップ数，etc.）の関係を，探索手法毎に表やグラフ等にまとめよ．それらの結果を参照して考察を行い，各探索手法の違いを説明せよ．

**必須課題 1-2** グループでの進捗管理や成果物共有などについて，工夫した点や使ったツールについて考察せよ．

**発展課題 1-3** Search.javaの探索過程や最終的に得られた順路をユーザに視覚的に示すGUIを作成せよ．

# 2　発展課題 1-3

Search.java の探索過程や最終的に得られた順路をユーザに視覚的に示す GUI を作成せよ.

私の担当箇所は，発展課題 1-3 の GUI 全般の Swing を用いた実装である.

## 2.1　手法

GUI を実装するにあたり，以下のような方針を立てた.

1. 探索空間における各要素（ノードや経路）を表示する.
2. 要素に付随する値を入力するための入力ボックスを表示する.
3. 探索を選択するためのボタンと，実行ボタンを表示する.
4. 入力ボックスやボタンで入力した値を Search.java に反映し，実行する.
5. 得られた経路を GUI に反映する.

1. に関して，ユーザーがなるべく直感的に理解できるようにすべく，表は用いずに 1 つの図の形式で完結するような仕様とした.

2．に関して，各要素とセットにして図中に入力ボックスを埋め込むことで，より直感的な入力を可能とした．また，入力ボックスには数値の入力に適した JSpinner クラスを用いた.

3．に関して，探索の選択には，選択肢の表示に適した JRadiButton クラスを用いた．値の反映には ActionListener インターフェースを用いて，Search.java の方も値を受け取って実行できるように改良した.

4．に関して，Search.java から探索結果を渡すように改良し，受け取った結果を経路に再描写することで，ユーザーが視覚的に経路を得ることができる仕様とした.

## 2.2 実装

実装にあたり，主に下記のサイトを参考にした．

TATSUO IKURA：『Swing を使ってみよう - Java GUI プログラミング』 https://www.javadrive.jp/tutorial/ (2019/10/15 アクセス)

SearchGUI.java には以下のクラスが含まれる．

- SearchGUI: メソッド main, searchButtons, actionPerformed を実装したクラス
- NodePanel: メソッド update と各種ゲッターを実装したノードに関するパネルを操作するためのクラス
- PathPanel: メソッド paintComponent, paintArrows, setShortestDistance, getMidPoints, execHor, execVer, update, forRepaint を実装した経路に関するパネルを操作するためのクラス

Search.java は以下のように改良した．

- Search クラスに，SearchGUI.java 用の実行メソッド exec を追加．
- Node クラスに，値の更新のためのメソッド setHValue, remakeChild を追加．
- Node クラスに値初期化のためのメソッド reset を追加．

### 2.2.1 探索空間における各要素（ノードや経路）を表示するまで

SearchGUI クラスではまず，フレームの中に mainPanel と subPanel の２つのパネルを生成する．

mainPanel には経路やノードを格納するためのパネルを挿入した．ノードのパネルは JButton クラスと JSpinner クラスを用いて簡単にできるが，経路のパネルは線の描写が必要であるため，やや複雑な paintComponent メソッドを拡張して用いる必要があった．

paintComponent の複雑な点は，このメソッドによって行われる描写は実行時に 1 度限りであるという点である．描写をパネルごとに分けて行いたかったので，その仕様は解決すべき課題となった．そこで経路のパネ

ル用のクラス PathPanel を用意し，その中で paintComponent メソッドを実装した．これによりインスタンスごとに描写呼び出しが行われて，経路を各パネルに分けることができた．

次に，表示についての課題が発生した．教科書通りのような経路図を表示するためには，各パネルを自由な位置に指定する必要があった．Swing には表を表示するのに適したレイアウトは数多くあるが，図を表示するためのレイアウトをあまり見つけることはできず，選択肢としては以下の 2 つの方法があった．

1. レイアウトマネージャーを無効にしてコンポーネントを座標指定で配置する．

2. SpringLayout クラスを用いてコンポーネントを他のコンポーネントとの相対位置で指定して配置する．

1．の方法のほうが簡単ではあるが，実行環境に依存するという問題があるため，2．の方法を用いて実装した．そうすると，経路の表示について，出発地と到着地のノードの座標から，その相対座標を計算する必要があった．そこで，getMidPoints メソッドではノードのパネル 4 面の各座標を計算し，setShortestDistance メソッドを用いてノード間の最短距離を計算することで実装した．

getMidPoints メソッドをソースコード 1 に，setShortestDistance メソッドをソースコード 2 に示す．

ソースコード 1: getMidPoints メソッド

```
1    Point[] getMidPoints(Rectangle r) {
2        Point[] midPoints = new Point[4];
3        for (int i = 0; i < midPoints.length; i++) {
4            midPoints[i] = new Point();
5        }
6        midPoints[0].setLocation(r.x + r.width / 2.0, r.y
             ); // 上の中点
7        midPoints[1].setLocation(r.x + r.width, r.y + r.
             height / 2.0); // 右の中点
8        midPoints[2].setLocation(r.x + r.width / 2.0, r.y
              + r.height / 2.0); // 下の中点
9        midPoints[3].setLocation(r.x, r.y + r.height /
             2.0); // 左の中点
10       return midPoints;
11   }
```

ソースコード 2: setShortestDistance メソッド

```
1    void setShortestDistance(Rectangle source, Rectangle
        distance) {
2        Point[] fromMidPoints = getMidPoints(source);
3        Point[] toMidPoints = getMidPoints(distance);
4
5        double min = Double.MAX_VALUE;
6        for (int i = 0; i < 4; i++) {
7            Point from = fromMidPoints[i].getLocation();
8            for (int j = 0; j < 4; j++) {
9                Point to = toMidPoints[j].getLocation();
10               double value = (from.getX() - to.getX())
                     * (from.getX() - to.getX())
11                       + (from.getY() - to.getY()) * (
                             from.getY() - to.getY());
12               if (value < min) {
13                   min = value;
14                   start = from;
15                   end = to;
16               }
17           }
18       }
19   }
```

次に，経路のパネルに，コストを格納するためのパネルを埋め込むためには，親となるパネルを大きめにする必要がある．そこで，int 型のフィールド grace を作り，相対座標にこの値を考慮することで，パネルの大きさに余裕を持たせることができた．

### 2.2.2 要素に付随する値を入力するための入力ボックスを表示するまで

まず，経路の初期値を得るために Search.java を実行し，得られたノードを以下のような Map で管理した．

ソースコード 3: main メソッドの一部

```
1        for (int i = 0; i < 10; i++) {
2            map.put(node[i], new NodePanel(node[i]));
3        }
```

NodePanel クラスにおいて，これのインスタンスを親のパネルとし，ノードのラベルと値を格納するための入力ボックスを配置した．レイア

ウトには 2 行 1 列の GridLayout クラスを利用した．NodePanel のコンストラクタをソースコード 4 に示す．

ソースコード 4: NodePanel のコンストラクタ

```
1    NodePanel(Node node) {
2        id = counter++;
3        this.node = node;
4        setLayout(new GridLayout(2, 1));
5        setBackground(Color.ORANGE);
6        setBorder(new BevelBorder(BevelBorder.RAISED));
7
8        JLabel label = new JLabel(id + ": " + node.
             getName());
9        model = new SpinnerNumberModel(node.getHValue(),
             0, 9999, 1);
10       JSpinner spinner = new JSpinner(model);
11       spinner.setPreferredSize(new Dimension(50, 25));
12
13       add(label);
14       add(spinner);
15   }
```

### 2.2.3 探索を選択するためのボタンと，実行ボタンを表示するまで

探索を選択するためのボタンには JRadioButton クラスを用いた．複数選択を許可しないために，ButtonGroup クラスを用いた．これらと実行ボタンを含んだパネルを生成する searchButtons メソッドをソースコード 5 に示す．

ソースコード 5: searchButtons メソッド

```
1    JPanel searchButtons() {
2        JPanel p = new JPanel();
3        p.setLayout(new BoxLayout(p, BoxLayout.PAGE_AXIS
             ));
4
5        radio = new JRadioButton[6];
6        radio[0] = new JRadioButton("幅優先探索");
7        radio[1] = new JRadioButton("深さ優先探索");
8        radio[2] = new JRadioButton("分岐限定法");
9        radio[3] = new JRadioButton("山登り法");
```

```
10          radio[4] = new JRadioButton("最良優先探索");
11          radio[5] = new JRadioButton("A*アルゴリズム");
12          ButtonGroup group = new ButtonGroup();
13          for (int i = 0; i < radio.length; i++) {
14              group.add(radio[i]);
15              p.add(radio[i]);
16          }
17
18          JButton button = new JButton("実行");
19          button.addActionListener(this);
20          p.add(button);
21
22          return p;
23      }
```

### 2.2.4 入力ボックスやボタンで入力した値を Search.java に反映し，実行するまで

まず，search.java を反映した値を用いて再実行するために，再実行用のメソッド exec を実装した．

次に，実行ボタンを押したときに入力した値を反映するためには，ActionListener インターフェースの actionPerformed メソッドを実装する必要がある．ノードの値，探索の種類，コストの値を全て反映し実行する，actionPerformed メソッドの前半をソースコード 6 に示す．

ソースコード 6: actionPerformed メソッドの前半

```
1       public void actionPerformed(ActionEvent e) {
2           for (NodePanel p : map.values()) {
3               p.getNode().reset();
4               p.update((Integer)p.getModel().getValue());
                    // ヒューリスティック値の変更を反映
5           }
6
7           for (int i = 0 ; i < radio.length; i++){ // 探索
                の選択
8               if (radio[i].isSelected()){
9                 which = i + 1;
10              }
11          }
12
```

```
13          for(int i = 0; i < paths.size(); i++) {
14              PathPanel p = paths.get(i);
15              p.update((Integer)p.getModel().getValue());
                    // コストの変更を反映
16          }
17
18          ArrayList<Node> route = search.exec(which); // 再
                実行
```

### 2.2.5 得られた経路を GUI に反映するまで

再描写には repaint メソッドを用いる必要がある．これにより，paint-Component メソッドの再度呼び出しが行われるため，描写の条件による分岐を paintComponent 側で用意する必要がある．そこで PathPanel クラス内に boolean 型のフィールド pass を用意し，forRepaint メソッド内で状態を確認，更新することで必要な経路のみ色を変えるような仕様にすることができた．また，Search.java の exec メソッドの戻り値を，通るノードのみ格納したリストし，それを用いることで以上の操作を容易にした．

得られた経路を反映する，actionPerformed メソッドの後半をソースコード 7 に示す．

ソースコード 7: actionPerformed メソッドの後半

```
1           for(int i = 0; i < paths.size(); i++) {
2               PathPanel p = paths.get(i);
3               for(int j = 0; j < route.size() - 1; j++) {
4                   if(p.forRepaint(route.get(j), route.get(j
                        + 1))) {
5                       break;
6                   }
7               }
8               p.repaint();
9           }
10      }
```

## 2.3 実行例

SearchGUI を実行したところ，以下のような画面が得られる．

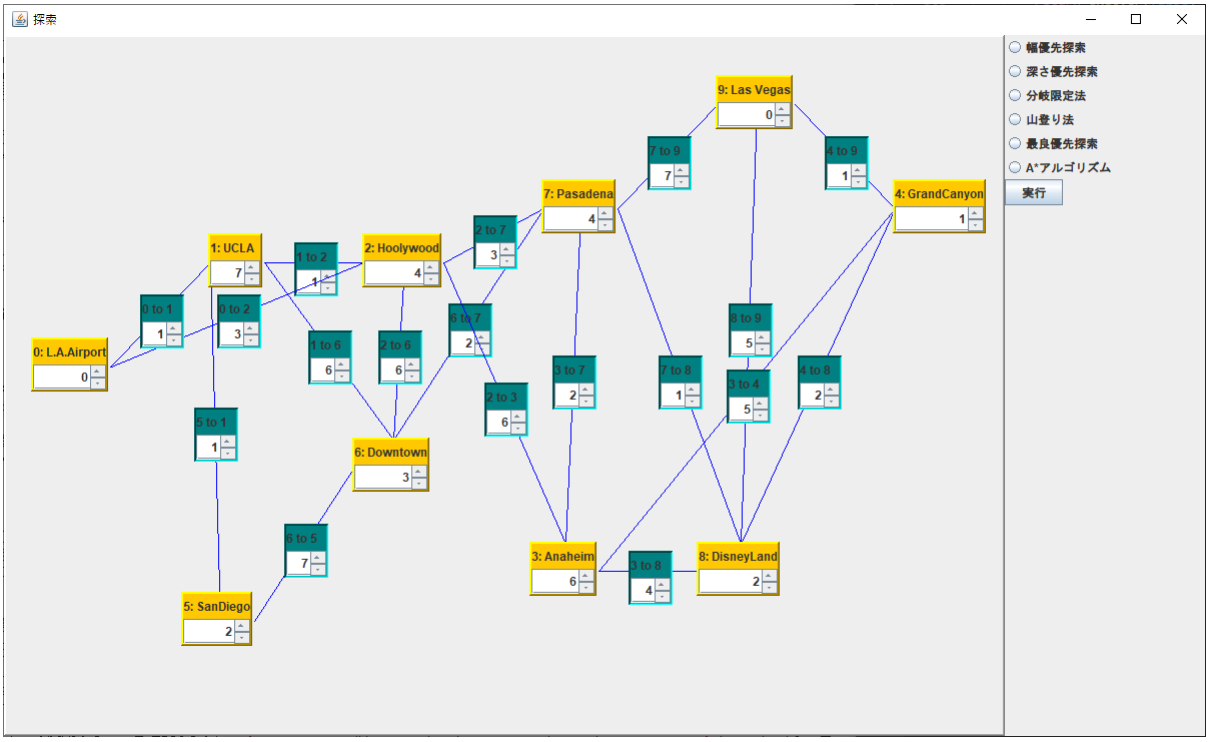


図 1: 初期状態

幅優先探索を選択し，実行したところ，以下のような画面が得られる．

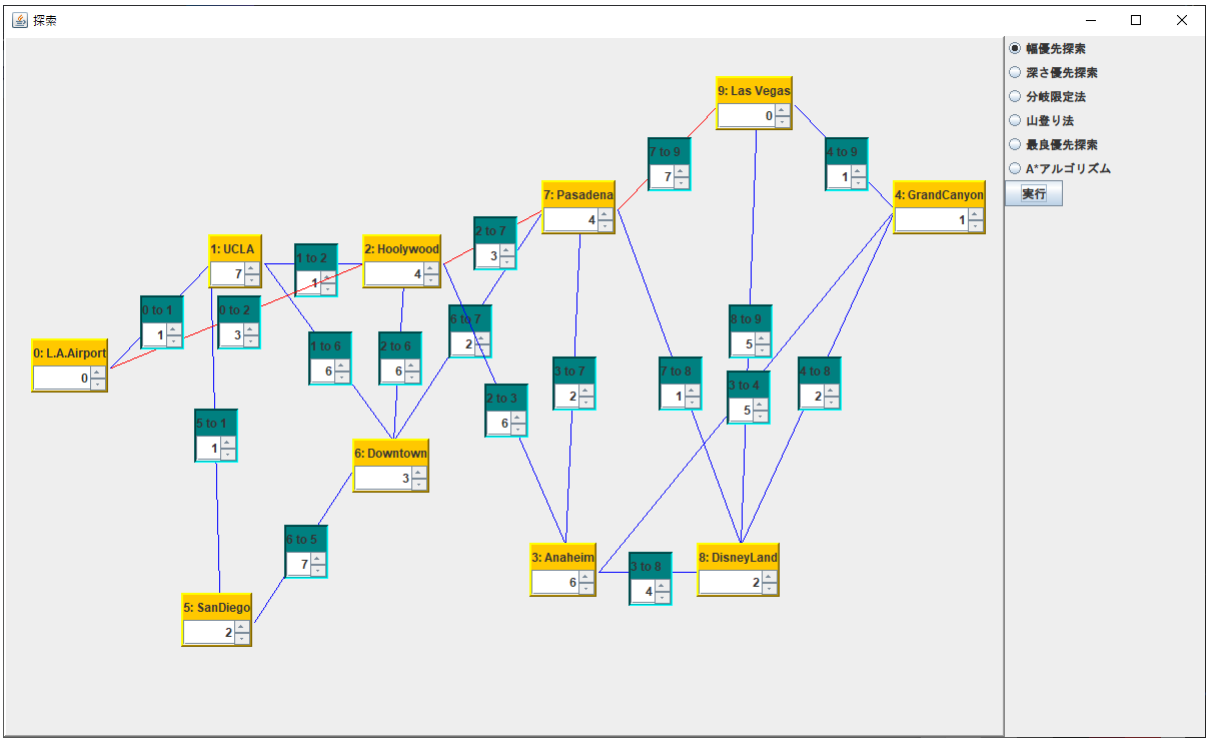


図 2: 幅優先探索の実行

同様にして A*アルゴリズムを実行したところ，以下のような画面が得られる．

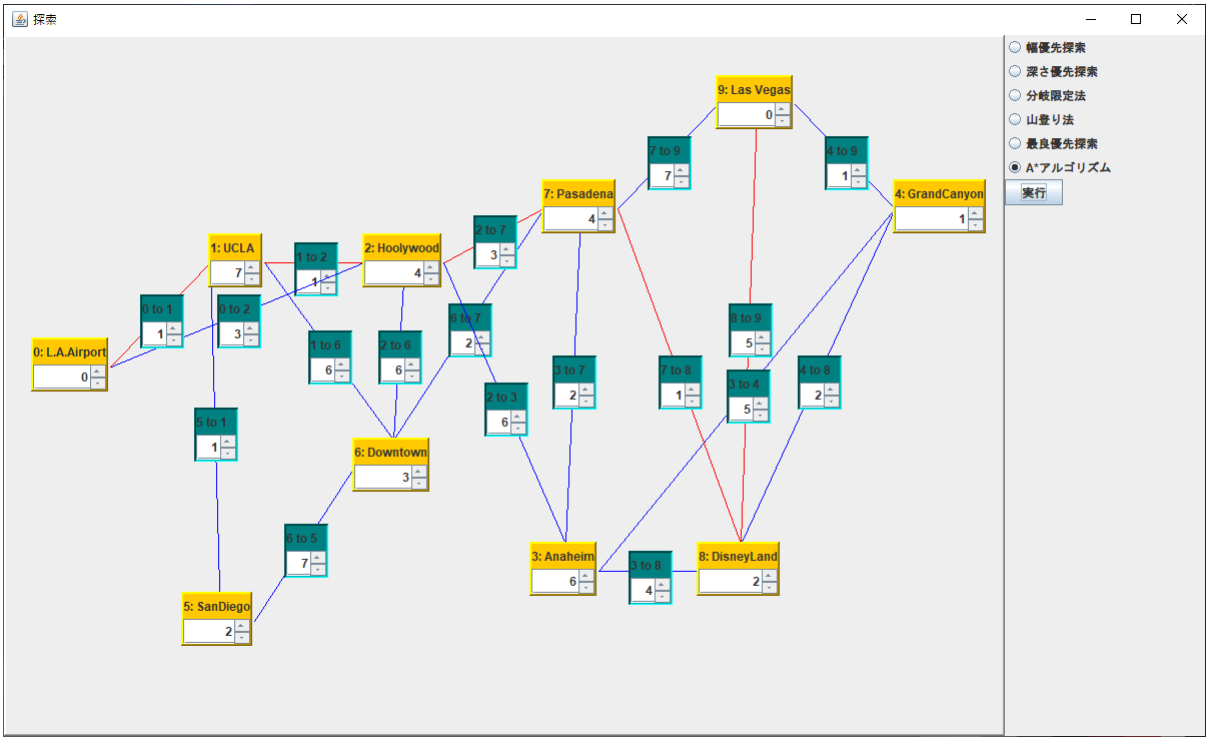


図 3: A*アルゴリズムの実行

## 2.4 考察

初めてSwingを使ってみて，ボタンや入力ボックスを用いた表形式のGUIは作りやすそうだと感じた一方で，今回実装したような図形式のGUIは大変であると感じた．そもそも，SpringLayoutクラスによる相対座標の指定となると，実行と表示を繰り返しながら試行的に作っていかねばならず，また，相対座標というもの自体，パネルを階層的に作っている中で，正しく相対すべき相手との座標になっているかなどの点で間違いやすいと感じた．

一方で，マウスを使ったパネルの操作から，座標の取得もできるようなので，あらかじめそういったプログラムを作り，それによる座標の取得と保持を上手に実装できれば，Swingにおける自由な配置でのGUI制作も，より簡単なものになると考えられる．

また，表示面では，paintComponentメソッドの扱いも難しかった．関連する情報の載っているサイトがあまり見つからず，どのように扱えばよいのか理解するのに時間を要した．このメソッドは実行時に自動的に1度だけ呼び出されることが難点だと感じたが，インスタンスごとにこのメソッドを持たせることで，より自由で部分的な実行を可能にすることができた．このように，制約のあるものは，大元から変えてあげることや，javaであるならばオブジェクト指向を用いた分解ができないかを第一に考えることこそが，より迅速な問題解決につながると考えられる．

また，経路の表示には矢印を用いるよう試みたが，先述した相対座標の難しさ等もあり，上手に表示できなかったため断念した．また，矢印では先端部分の重なり等の問題等もあるため，今回のように”0 to 2”のようなラベルで表現した方が，誤解を減らす上でも役立ったと考えられる．しかし，経路同士の重なりを防ぐことはできないため，より誤解を減らすための方法として，線を曲線にすることや，線を縁取ることでより誤解の少ない表示ができるのではないかと考えられる．

また，今回NodePanelやPathPanelといったクラスに分解したことで，SearchGUIクラスのコードの複雑さを軽減することができた．これらのクラスは，実装中に分解できるんじゃないかと思い，後で分解したものであったので，今後はコードを書く前からクラスやメソッドとして取り出せないかを考えるようにすれば，より効率的なオブジェクト指向におけるプログラミングが見込めると考えられる．

Swingにおけるパネルについても，どこを一纏めにするかを予め考慮しておくことで，混乱せずにGUIの実装が見込めると考えられる．

# 3　感想

GUI のについては，警戒していたよりも簡単に実装することができた．ただ，線の描写や相対座標という概念には本当に苦しめられた．

最近，Unity を触っているが，あれを使えば今回のようなプログラムなどより簡単に，更に綺麗に作れるんじゃないかと終わった今感じている．そもそも，Swing 自体こういった自由な配置の GUI 作成には向いていないんじゃないかと感じ，実装しづらいもので頑張って実装しても見返りはあまりは大きくないな，と感じた．

それよりも，実装しやすいような形式に問題自体を変えられないか，目的の実装により向いているライブラリやソフトは無いか，ということを今後は軸に考えるようにしてゆきたい．これからは，いかに手を抜いて高品質なものを作れるかということを軸に取り組んでゆきたい．