L3: Searching programs in Java (AI programming - 1)

City connection problem.

- 探索アルゴリズムの復習
- 探索で問題解決アルゴリズムの実装
- 幅優先探索アルゴリズムの説明
- 課題は深さ優先探索アルゴリズムの実装

The algorithm follows:

1. Create a queue and add the first **node** to it.
2. Loop:
   - If the queue is empty, quit.
   - Remove the first **node** from the queue.
   - If the **node** contains the goal state, then exit with the node as the solution.
   - For each child of the current node: add the new state to the **back** of the queue.

どのように実装するか

1. 開始地点を空のキューに加える。

```
// これからチェックする Node を保存するリスト
ArrayList<Node> open = new ArrayList<Node>();
open.add(start);
```

2. もしキューが空ならば、グラフ内の全てのノードに対して処理が行われたので、探索をやめる。

```
// チェックする Node 数が 0 →探索失敗
if (open.size() == 0) {
    // 失敗
    success = false;
    // ループを抜ける
    break;
```

3. ノードをキューの先頭から取り出し、以下の処理を行う。
   - ノードがゴールであれば、探索をやめ結果を返す。
   - そうでない場合、ノードの子で未探索のものを全てキューに追加する。

```
if (node == goal) {
    // 成功
    success = true;
    // ループを抜ける
    break;
}// 目的地でない場合
```

4. 2に戻る。

どのように実装するか

1. 開始地点を空のキューに加える。

```
// これからチェックするNodeを保存するリスト
ArrayList<Node> open = new ArrayList<Node>();
open.add(start);
```

2. もしキューが空ならば、グラフ内の全てのノードに対して処理が行われたので、探索をやめる。

```
// チェックするNode数が0→探索失敗
if (open.size() == 0) {
        // 失敗
        success = false;
        // ループを抜ける
        break;
```

3. ノードをキューの先頭から取り出し、以下の処理を行う。
   - ノードがゴールであれば、探索をやめ結果を返す。
   - そうでない場合、ノードの子で未探索のものを全てキューに追加する。

```
if (node == goal) {
        // 成功
        success = true;
        // ループを抜ける
        break;
}// 目的地でない場合
```

4. 2に戻る。

The algorithm follows:

1. Create a queue and add the first **node** to it.
2. Loop:
   - If the queue is empty, quit.
   - Remove the first **node** from the queue.
   - If the **node** contains the goal state, then exit with the node as the solution.
   - For each child of the current node: add the new state to the **front** of the queue.

どのように実装するか

1. 開始地点を空のスタックに加える。
2. もしスタックが空ならば、グラフ内の全てのノードに対して処理が行われたので、探索をやめる。
3. ノードをスタックの先頭から取り出し、以下の処理を行う。
   - ノードがゴールであれば、探索をやめ結果を返す。
   - そうでない場合、ノードの子で未探索のものを全てスタックに追加する。
4. 2に戻る。

どのように実装するか

1. 開始地点を空のスタックに加える。

2. もしスタックが空ならば、グラフ内の全てのノードに対して処理が行われたので、探索をやめる。

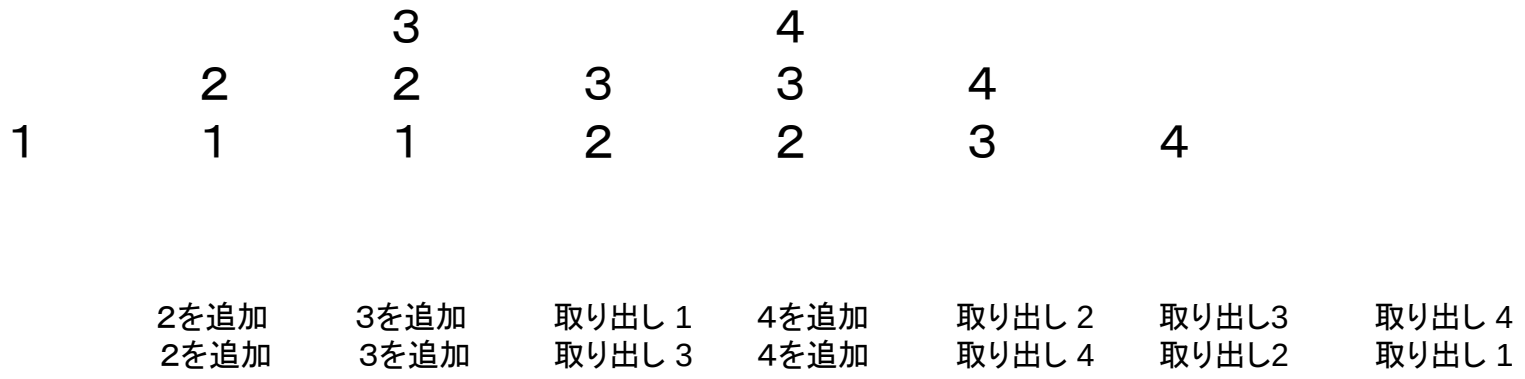
3. ノードをスタックの先頭から取り出し、以下の処理を行う。
   - ノードがゴールであれば、探索をやめ結果を返す。
   - そうでない場合、ノードの子で未探索のものを全てスタックに追加する。

4. 2に戻る。

キューとスタック

- キューとスタックはどちらもデータ構造であり、データの挿入と取り出しの順番が異なる

キュー：入れた順に取り出す(FIFO : First In First Out)

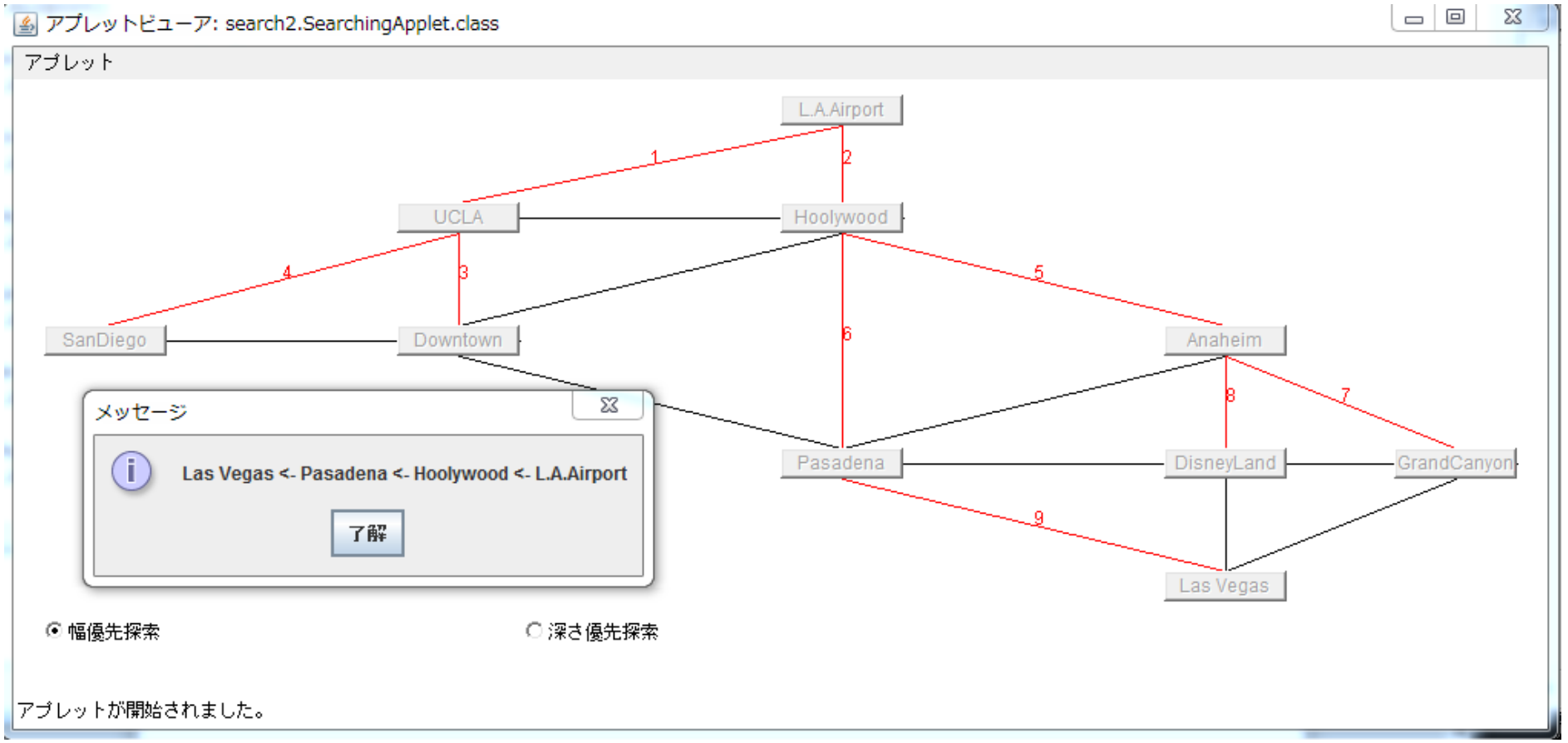


スタック：最後に入れたものから順に取り出す(FILO : First In Last Out)

```
public void breadthFirst() {
    ArrayList<Node> open = new ArrayList<Node>();
    open.add(start);
    ArrayList<Node> closed = new ArrayList<Node>();
    boolean success = false;
    int step = 0;
    g.setColor(Color.RED);
    while (true) {
        System.out.println("STEP:" + (step++));
        System.out.println("OPEN:" + open.toString());
        System.out.println("closed:" + closed.toString());
        if (open.size() == 0) {// no more node to be checked
            success = false;
            break;
        } else {
            Node node = open.get(0);
            open.remove(0);
            if (node == goal) {
                success = true;
                break;
            } else {
                ArrayList<Node> children = node.getChildren();
                closed.add(node);
                for (int i = 0; i < children.size(); i++) {
                    Node m = children.get(i);
                    if (!open.contains(m) && !closed.contains(m)) {
                        num++;
                        m.setPointer(node);
```

```
if (m == goal) {
    open.add(0, m);
    node.getHm().get(m).draw(g, num);
    sleep();
    break;
} else {
    open.add(m);
    node.getHm().get(m).draw(g, num);
    sleep();
    }
} // end-of-if
} // end-of-for
} // end-of-else
} // end-of-else
} // end-of-while

if (success) {
    reset.setVisible(true);
    String result = printSolution(goal, "");
    showDialog(result);
} // end-of-if
} // end-of -method
```

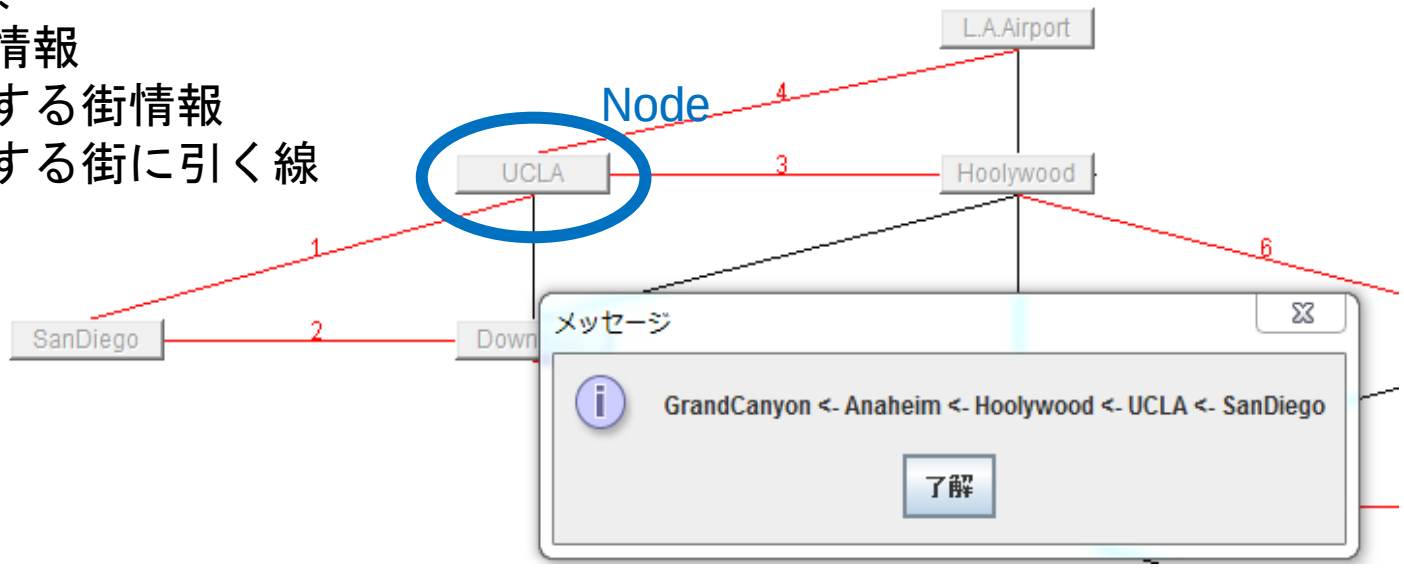
実行結果の例 in GUI (graphical User Interface)



Nodeの作成と表示

Nodeに必要なものを考えてみましょう

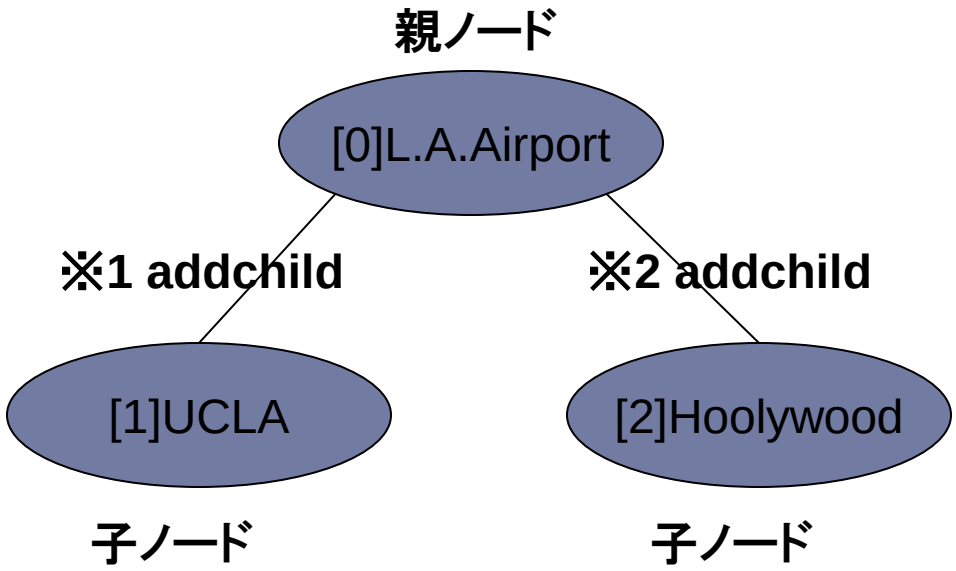
- (街の)名前
- ボタン
- X座標
- Y座標
- 経路情報
- 隣接する街情報
- 隣接する街に引く線



```
// この部分を参考に都市を１つ増やす
                node = new Node[10];
                node[0] = new Node("L.A.Airport", x / 2, 10);
                node[1] = new Node("UCLA", x / 4, y / 5);
                node[2] = new Node("Hoolywood", x / 2, y / 5);
                node[3] = new Node("Anaheim", 3 * x / 4, y * 2 / 5);
                node[4] = new Node("GrandCanyon", x - 100, y * 3 / 5);
                node[5] = new Node("SanDiego", 20, y * 2 / 5);
                node[6] = new Node("Downtown", x / 4, y * 2 / 5);
                node[7] = new Node("Pasadena", x / 2, y * 3 / 5);
                node[8] = new Node("DisneyLand", x * 3 / 4, y * 3 / 5);
                node[9] = new Node("Las Vegas", 3 * x / 4, y * 4 / 5);
```

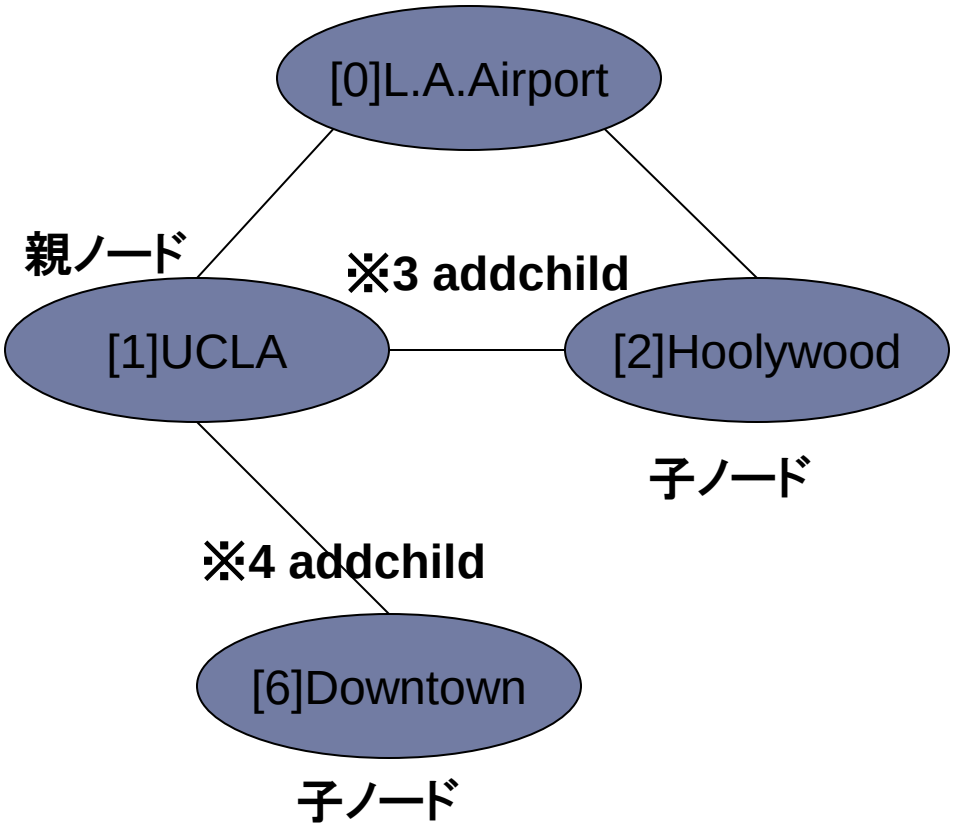
ノードの追加

```
node[0].addChild(node[1]);  //※1
node[0].addChild(node[2]);  //※2
node[1].addChild(node[2]);  //
node[1].addChild(node[6]);  //
node[2].addChild(node[3]);  //
node[2].addChild(node[6]);  //
node[2].addChild(node[7]);  //
node[3].addChild(node[4]);  //
node[3].addChild(node[7]);
node[3].addChild(node[8]);
node[4].addChild(node[8]);
node[4].addChild(node[9]);
node[5].addChild(node[1]);
node[6].addChild(node[5]);
node[6].addChild(node[7]);
node[7].addChild(node[8]);
node[7].addChild(node[9]);
node[8].addChild(node[9]);
```



ノードの追加

```
node[0].addChild(node[1]);  //
node[0].addChild(node[2]);  //
node[1].addChild(node[2]);  //※3
node[1].addChild(node[6]);  //※4
node[2].addChild(node[3]);  //
node[2].addChild(node[6]);  //
node[2].addChild(node[7]);  //
node[3].addChild(node[4]);  //
node[3].addChild(node[7]);
node[3].addChild(node[8]);
node[4].addChild(node[8]);
node[4].addChild(node[9]);
node[5].addChild(node[1]);
node[6].addChild(node[5]);
node[6].addChild(node[7]);
node[7].addChild(node[8]);
node[7].addChild(node[9]);
node[8].addChild(node[9]);
```



Nodeクラス

```
//都市クラス
public class Node extends Button {
String name; // 街の名前
private int x, y; // 座標
private int w = 80, h = 20; // 幅と高さ
ArrayList<Node> children; // 隣接都市
Node pointer; // 経路
HashMap<Node, Line> hm; // 隣接都市への線(隣接都市/それを結ぶ線)
// 街の名前と座標の設定
public Node(String theName, int x, int y) {
this.name = theName;
children = new ArrayList<Node>();
hm = new HashMap<Node, Line>();
this.x = x;
this.y = y;
this.setBounds(x, y, w, h);
this.setLabel(name);
} （略）
```

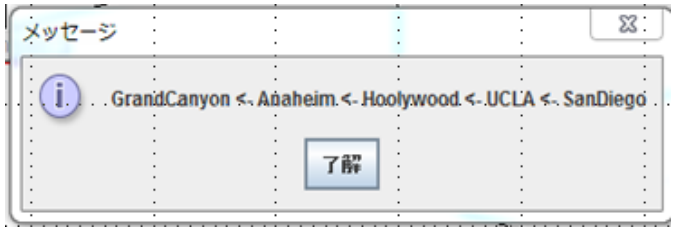
Buttonクラスを継承することでButtonの機能も使えるようになる

コンストラクタの引数で渡された都市名と座標を設定する

Nodeクラス

```
public int getX() {// ボタンのX座標を返す
return x;
}
public int getY() {// ボタンのY座標を返す
return y;
}
public Node getPointer() {// 前の都市への経路を返す
return this.pointer;
}
public void setPointer(Node theNode) {// 経路を保存する
this.pointer = theNode;
}
public void drawLine(Graphics g, Node m, int num) {// 線を引く(探索時)
hm.get(m).draw(g, num);
}
public void drawLine(Graphics g, Node m) {// 線を引く
hm.get(m).draw(g);
}
```

線を引く時に必要



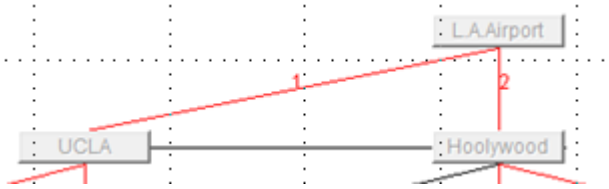
結果表示の時に必要

Lineクラスのdrawメソッドを呼び出す。探索時にはチェックした順番も引数に入れる

Nodeクラス

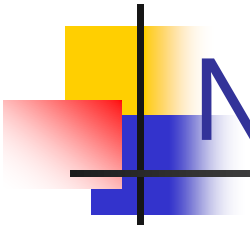
```
// 隣接都市の追加
public void addChild(Node theChild) {
children.add(theChild);
// 線も生成
hm.put(theChild, new Line(this, theChild));
}
```

引数に指定した都市と現在の都市をつなぎ線を作る




```
// 隣接都市への線データをまとめて返す
public HashMap<Node, Line> getHm() {
return hm;
}
// 隣接都市をまとめて返す
public ArrayList<Node> getChildren() {
return children;
}
public String toString() {
return name;
}
```

このメソッドを書くとコンソールに表示される形式を指定できる



Nodeの表示 – 練習３

- ShowNode.javaを書き加えていくつかのNodeをApplet上に表示し、Nodeをクリックすると都市名をダイアログで表示するようにしてみましょう

Nodeの実装

答案：ShowNode.java (+ Node.java, Line.java)

```
// 街の名前
String name;
// 座標
private int x, y;
// 幅と高さ
private int w = 80, h = 20;
// 隣接都市
ArrayList<Node> children;
// 経路
Node pointer;
// 隣接都市への線(隣接都市/それに付随する線)
HashMap<Node, Line> hm;
```

```
// 街の名前と座標の設定
public Node(String theName, int x, int y) {
        this.name = theName;
        children = new ArrayList<Node>();
        hm = new HashMap<Node, Line>();
        this.x = x;
        this.y = y;
        this.setBounds(x, y, w, h);
        this.setLabel(name);
}
```

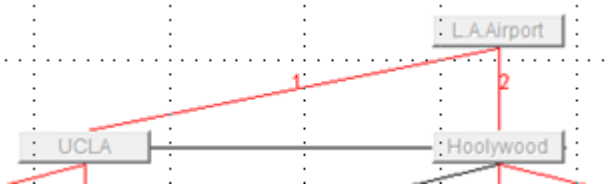
Lineクラス

```
private class Line {// 直線クラス
(略)
public Line(Node node, Node theChild) {
if (node.getY() == theChild.getY()) {// Y座標が同じ時は横に引く
        x0 = node.getX() + node.getWidth();
        y0 = node.getY() + node.getHeight() / 2;
        x1 = theChild.getX();
        y1 = theChild.getY() + theChild.getHeight() / 2;
}
else if (node.getY() < theChild.getY()) {// Y座標が異なるときはキレイになるように場合分け
        x0 = node.getX() + node.getWidth() / 2;
        y0 = node.getY() + node.getHeight();
        x1 = theChild.getX() + theChild.getWidth() / 2;
        y1 = theChild.getY();
} else {
        x0 = node.getX() + node.getWidth() / 2;
        y0 = node.getY();
        x1 = theChild.getX() + theChild.getWidth() / 2;
        y1 = theChild.getY() + theChild.getHeight();
}
node1 = node;
node2 = theChild;
}
```

Lineクラス

```
// 描画
public void draw(Graphics g) {
g.drawLine(x0, y0, x1, y1);
}

// 描画(探索時)
public void draw(Graphics g, int num) {
g.drawLine(x0, y0, x1, y1);
g.drawString("" + num, (x0 + x1) / 2, (y0 + y1) / 2);
}

public String toString() {
return node1 + " -> " + node2;
}
```

コンストラクタで指定した座標に線を引く
探索時には脇に番号を書く



ShowNode.java (1)

```
public class ShowNode extends Applet implements ActionListener {
        // 各都市
        Node node[];
        // 目標地点
        Node goal;
        // 開始地点
        Node start;
        // 画面の大きさ
        private int x = 1000, y = 400;
        // 都市のボタンの大きさ
        private int w = 150, h = 20;

        // 初期化メソッド
        public void init() {
                // 画面の生成
                super.setSize(x, y);
                // レイアウトの設定
                setLayout(null);
                // 各Nodeの作成
                nodeSetting();
                // 各都市の表示
                showNodes();
        }
```

ShowNode.java (2)

```
// 各都市の表示
private void showNodes() {
    // 各NodeをApplet上に表示しActionListernerに追加
}

// 各都市の定義とつながりの設定
private void nodeSetting() {
    // Nodeを作成するときは引数に都市名・x,y座標が必要
}

// ボタンや都市が押された時の処理
public void actionPerformed(ActionEvent e) {
    // Node情報をダイアログで表示
}
```

部分のコード

完全コードはSearchApplet.javaに参考する

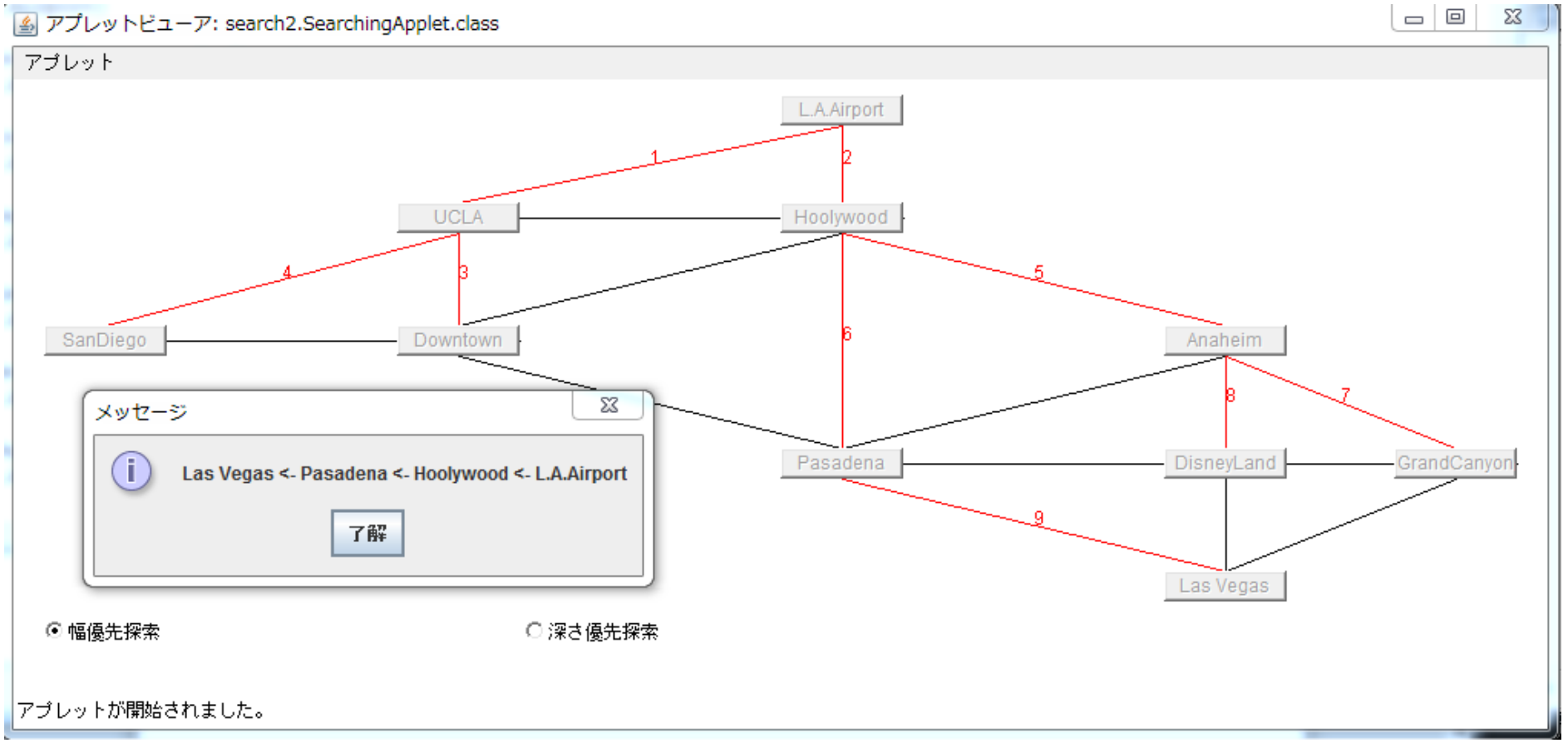
```
// 各都市の表示
private void showNodes() {
        for (int i = 0; i < node.length; i++) {
                // リスナーへの登録
                node[i].addActionListener(this);
                // 画面に追加
                add(node[i]);
        }
}
```

// 各都市の定義とつながりの設定

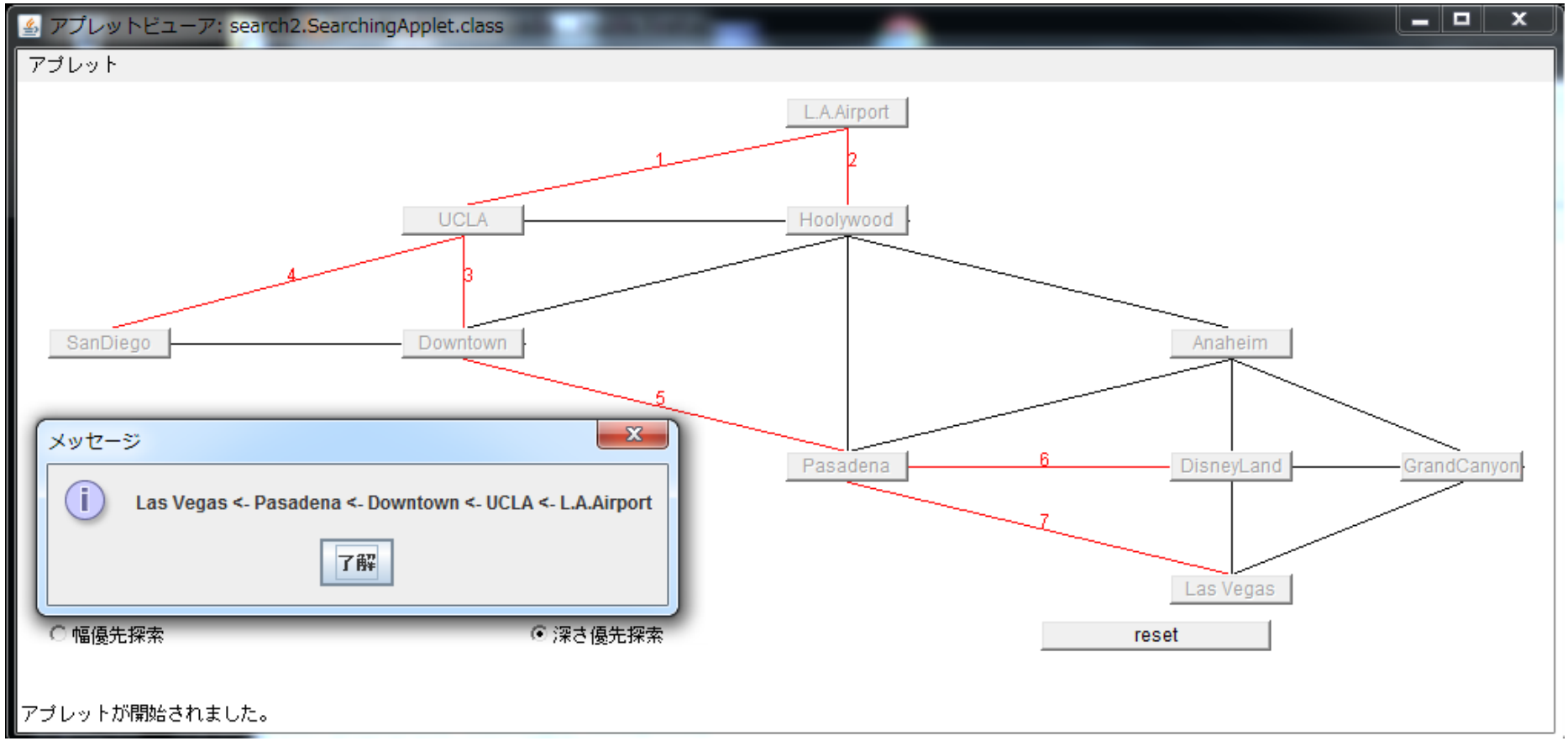
```
node[0] = new Node("L.A.Airport", x / 2, 10);
node[1] = new Node("UCLA", x / 4, y / 5);
node[2] = new Node("Hoolywood", x / 2, y / 5);

node[0].addChild(node[1]);
node[0].addChild(node[2]);
node[1].addChild(node[2]);
node[1].addChild(node[6]);
node[2].addChild(node[3]);
node[2].addChild(node[6]);
node[2].addChild(node[7]);
```

```
// ボタンや都市が押された時の処理
public void actionPerformed(ActionEvent e) {
        // リセットボタンが押された
        if (e.getSource() == reset) {
                // 初期化
                initialize();
        }
        // 到達地点が押されたとき
        else if (f1) {
                // 押された箇所を到達地点に設定
                goal = (Node) e.getSource();
                // 角都市を無効化する
                for (int i = 0; i < node.length; i++)
                        node[i].setEnabled(false);
                // 文字列を消去
                g.clearRect(2 * x / 3, y / 5 - 15, w, h);
                // 探索開始
                startSearch();
```

実行結果の例


アプレットビューア: search2.SearchingApplet.class
アプレット
L.A.Airport
1
2
UCLA
Hoolywood
4
3
5
SanDiego
Downtown
6
Anaheim
メッセージ
Las Vegas <- Pasadena <- Hoolywood <- L.A.Airport
了解
8
7
Pasadena
DisneyLand
GrandCanyon
9
Las Vegas
幅優先探索
深さ優先探索
アプレットが開始されました。

実行結果の例


アプレットビューア: search2.SearchingApplet.class
アプレット
L.A.Airport
1
2
UCLA
Hoolywood
4
3
SanDiego
Downtown
Anaheim
5
メッセージ
Las Vegas <- Pasadena <- Downtown <- UCLA <- L.A.Airport
了解
Pasadena
6
DisneyLand
GrandCanyon
7
Las Vegas
幅優先探索
深さ優先探索
reset
アプレットが開始されました。

SearchingAppletクラス

```
//メインクラス
public class SearchingApplet extends Applet implements ActionListener,
ItemListener {
(略)
// 初期化メソッド
public void init() {
// 画面の生成
setSize(x, y);
// レイアウトの設定
setLayout(null);
// グラフィックスの取得
g = getGraphics();
// 各都市のつながりの定義
makeStateSpace();
// 各都市の表示
showNodes();
// ボタンなどの表示
showButtons();
}
```

Appletを実行した時に最初に呼び出されるメソッド

Nullを指定しないと座標指定のレイアウトができない

SearchingAppletクラス

```
// 各都市の定義とつながりの設定
private void makeStateSpace() {
// この部分を参考に都市を1つ増やす
node = new Node[10];
node[0] = new Node("L.A.Airport", x / 2, 10);
:
:

// addChildメソッドは単方向に繋げる
node[0].addChild(node[1]);
:
:
// なので双方向にするには逆版も
node[1].addChild(node[0]);

}
```

配列のサイズを超えない程度に自分でいくつかの都市を作成する。座標にx,yを使うと画面サイズを変更した時に対応しやすい

これだけだと0→1は行けるが、1→0は行けないので双方向に繋げたい場合は逆版も

SearchingAppletクラス

```
// 各都市の表示
private void showNodes() {
for (int i = 0; i < node.length; i++) {
// リスナーへの登録
node[i].addActionListener(this);
// 画面に追加
add(node[i]);
}
}
```

NodeクラスはButtonクラスを継承しているので、Buttonクラスと同じようにリスナーの登録や配置ができる

SearchingAppletクラス

```
// 各都市の表示
private void showNodes() {
for (int i = 0; i < node.length; i++) {
// リスナーへの登録
node[i].addActionListener(this);
// 画面に追加
add(node[i]);
}
}
```

NodeクラスはButtonクラスを継承しているので、Buttonクラスと同じようにリスナーの登録や配置ができる

SearchingAppletクラス(幅優先探索部分)

```
// 幅優先探索
public void breadthFirst() {

// これからチェックするNodeを保存するリスト
ArrayList<Node> open = new ArrayList<Node>();
open.add(start);

// チェック済みのNodeを保存するリスト
ArrayList<Node> closed = new ArrayList<Node>();
// 目的地に到達できたかどうか
boolean success = false;
// ステップ数
int step = 0;
// 見やすく色変え
g.setColor(Color.RED);
```

openに、これから調べる都市を追加していく(キュー)

チェック済みの都市を格納する

SearchingAppletクラス(幅優先探索部分)

```
while (true) {
// チェックするNode数が0→探索失敗
if (open.size() == 0) {
success = false; // 失敗
break; // ループを抜ける
}
 else {
// 先頭を取り出してチェックを始める
Node node = open.get(0);
// 取り出したら、もうチェックしないので外す
open.remove(0);
// 取り出したのが目的地だったら
if (node == goal) {
success = true; // 成功
break; // ループを抜ける
}
// 目的地でない場合
```

Open=0ということは、全ての都市を調べ終えたか、今の都市が他の都市とつながっていないということ

探索開始

SearchingAppletクラス(幅優先探索部分)

```
while (true) {
// チェックするNode数が0→探索失敗
if (                                    ) {
success = false; // 失敗
break; // ループを抜ける
}
 else {
// 先頭を取り出してチェックを始める
Node node =                                    
// 取り出したら、もうチェックしないので外す
open.remove(0);
// 取り出したのが目的地だったら
if (                                    ) {
success = true; // 成功
break; // ループを抜ける
}
```

全ての都市を調べ終えたか、今の都市が他の都市とつながっていないということ

探索開始

探索終了

SearchingAppletクラス(幅優先探索部分)

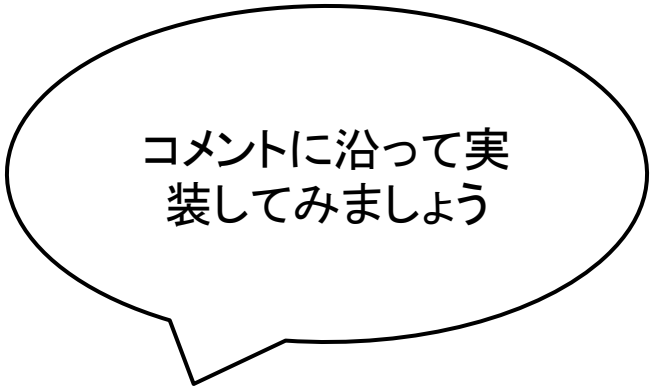
```
else {
// 取り出した都市に繋がっている都市のリストを取り出す
ArrayList<Node> children = node.getChildren();
// 取り出した都市はもう用済み
closed.add(node);
// 繋がっている都市を1つづつ取り出す
for (int i = 0; i < children.size(); i++) {
// リストから順番に取り出す
Node m = children.get(i);
// これからチェックするリスト・もうチェックしたリストにも入っていない
if (!open.contains(m) && !closed.contains(m)) {
num++;
// ポインターをセット
m.setPointer(node);
// 取り出したのが目的地なら
if (m == goal) {
```

こうすることで同じ都市を複数回チェックするミスをなくせる

子ノード→親ノードにポインターセット

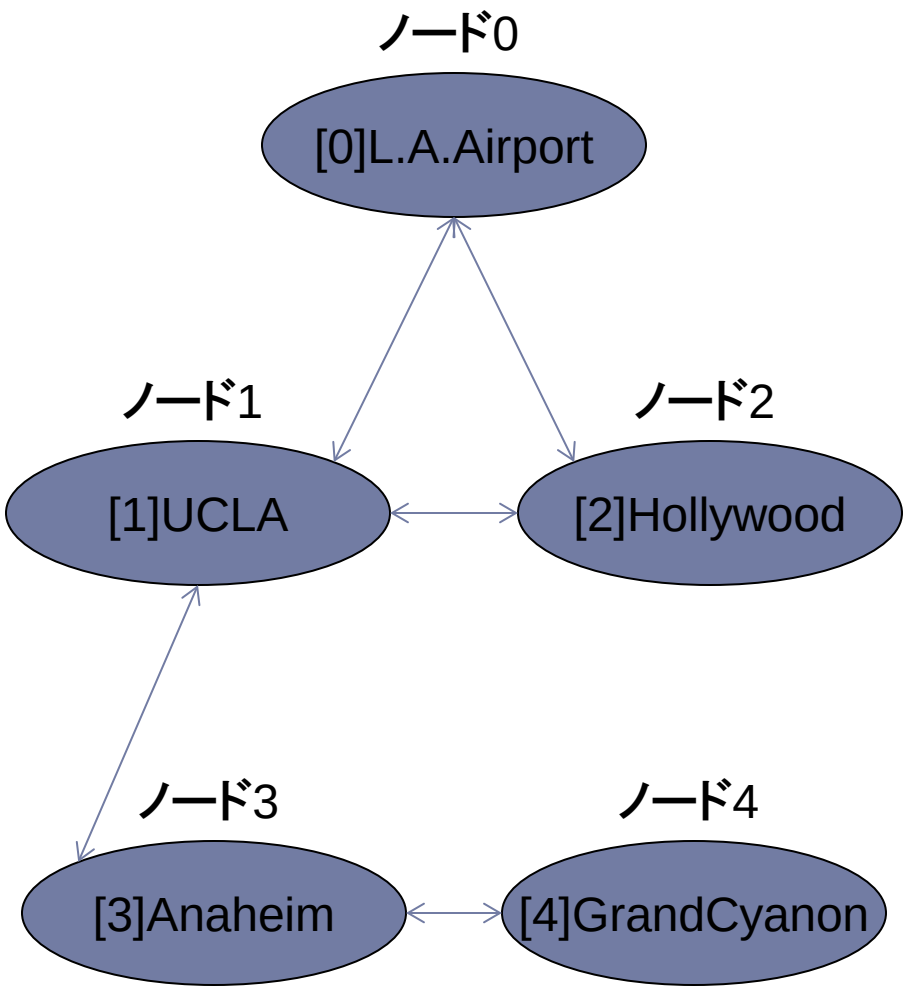
SearchingAppletクラス(幅優先探索部分)

```
// これからチェックするリストの先頭に入れる
// これにより次回は目的地がチェックされる

// 線を引く

//見やすいように小休止

// 目的地が見つかればこれ以上のチェックは不要
}
// そうでないとき
else {
// これからチェックするリストに入れる

// 線を引く

//見やすいように小休止
```



```
}
```

動作例(幅優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

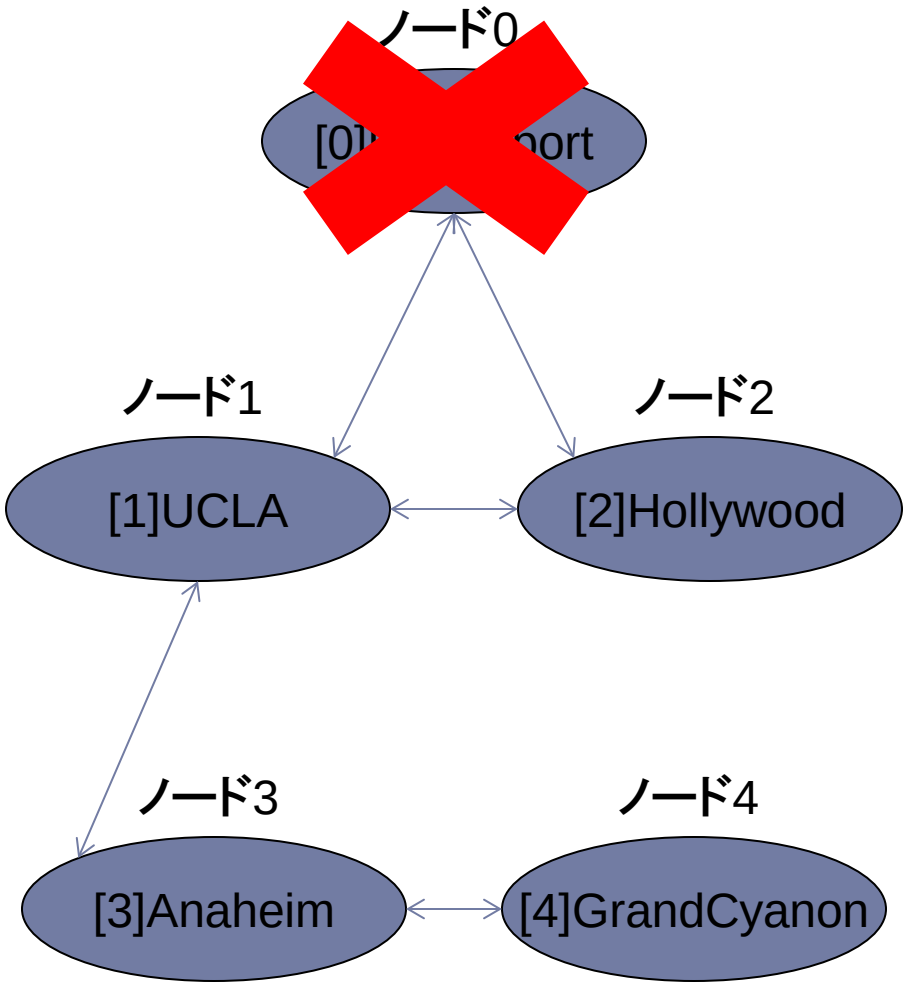
STEP:0

Open={L.A.Airport}

Closed={}

※Openはこれから調べるNode
※Closeは調べ終わったNode

動作例(幅優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

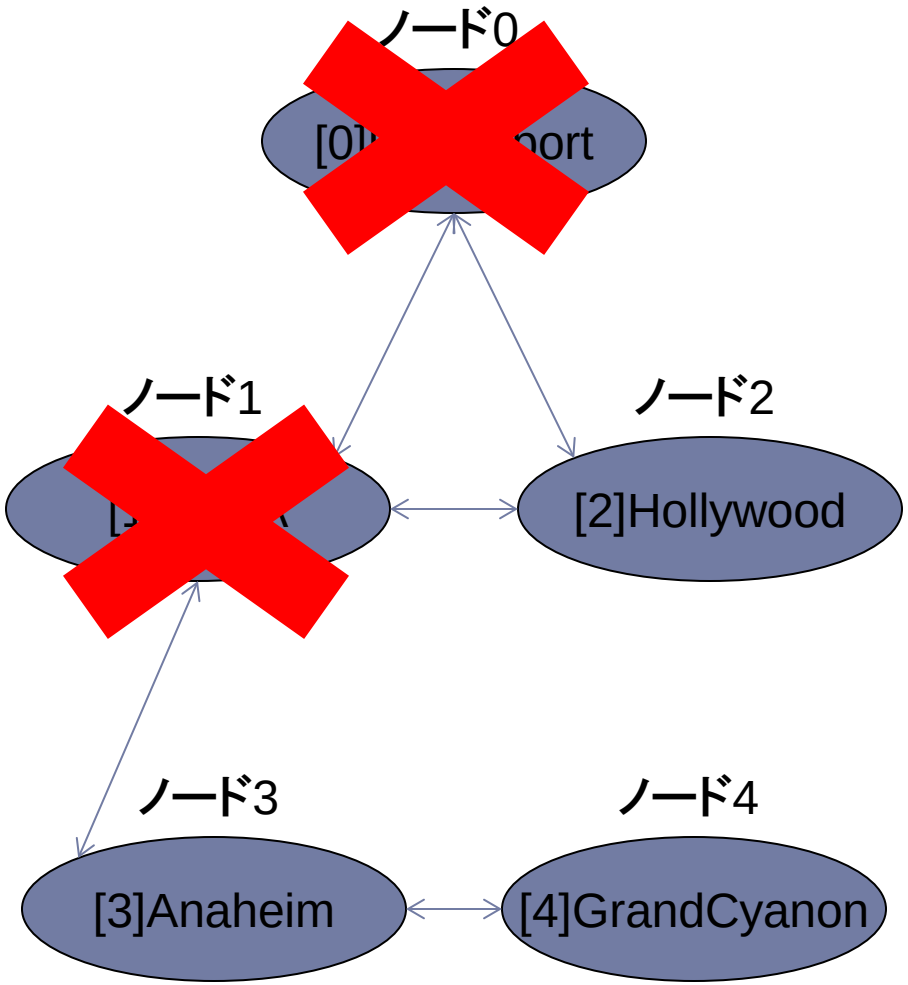
STEP:1

Open={UCLA,Hollywood}

Closed={L.A.Airport}

※L.A.Airportは調べ終わったのでcloseに追加し、L.A.Airportの子ノードであるUCLAとHollywoodをopenに追加

動作例(幅優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

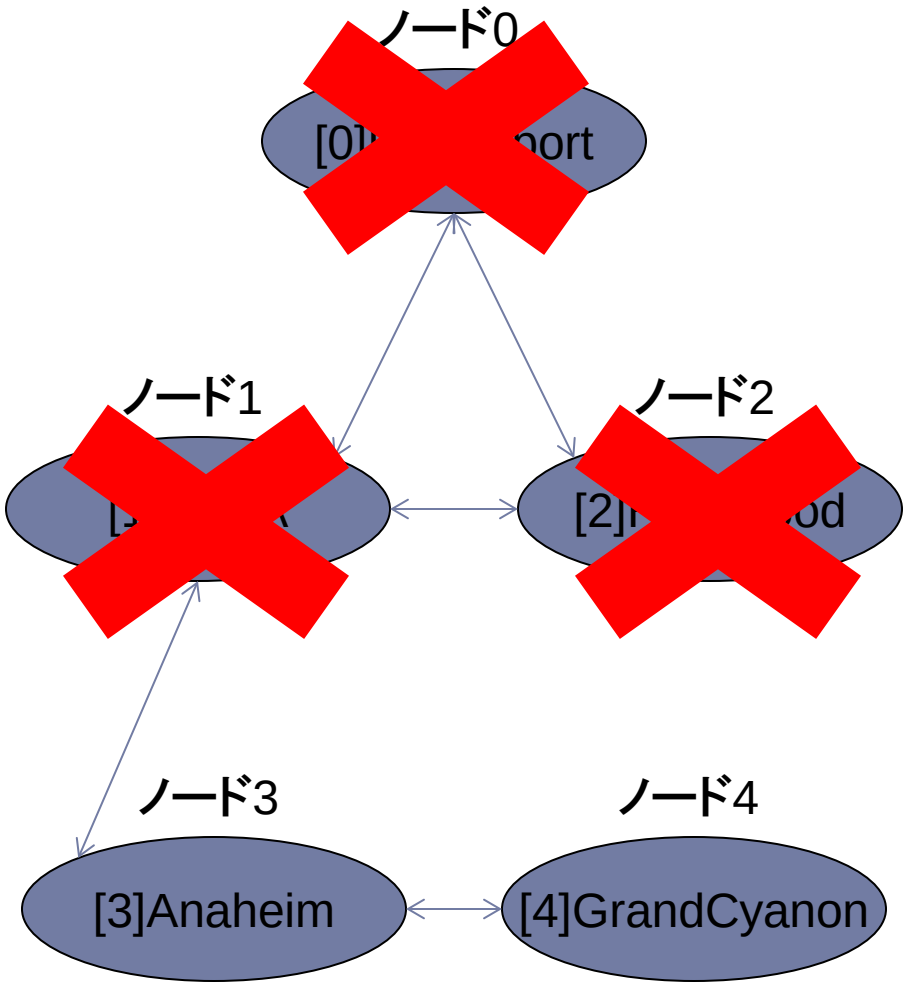
STEP:2

Open={Hollywood,Anaheim}

Closed={L.A.Airport,UCLA}

※UCLAは調べ終わったのでcloseに追加し、UCLAの子ノードであるAnaheimをopenに追加。Hollywodは既にopenに追加してあるので無視

動作例(幅優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

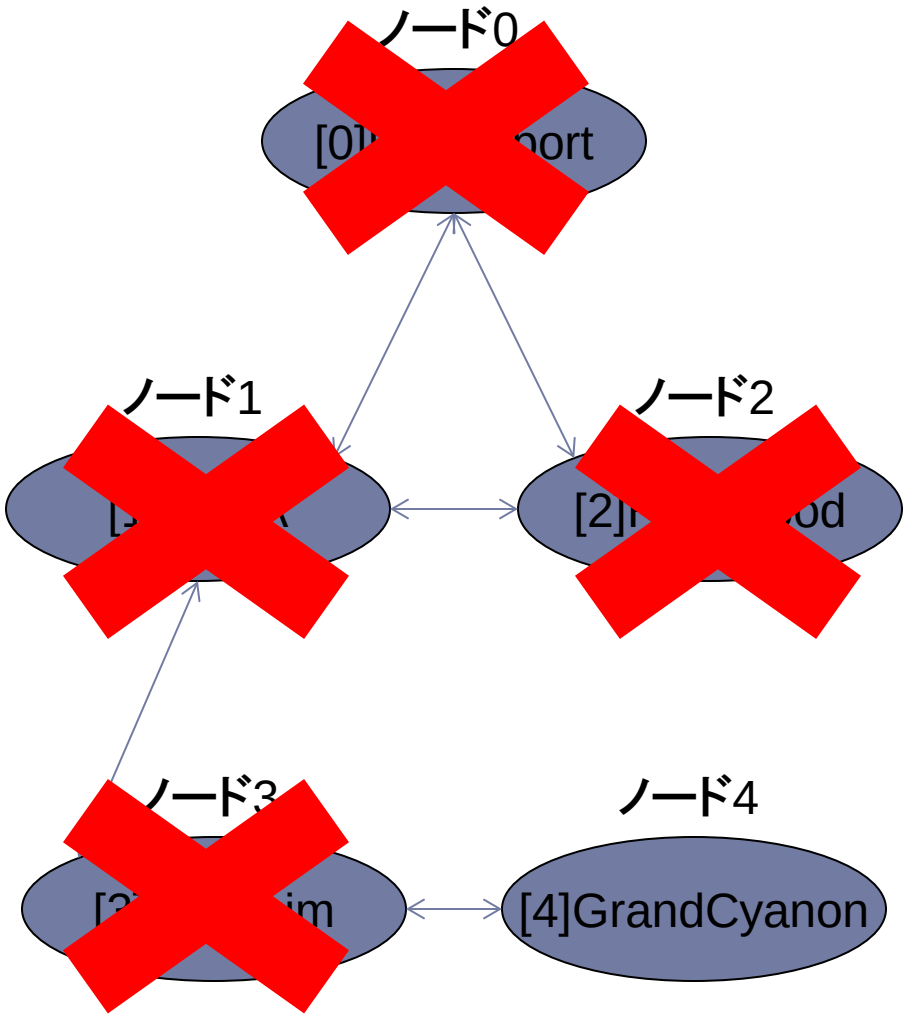
STEP:3

Open={Anaheim}

Closed={L.A.Airport,UCLA,Hollywood}

※Hollywoodは調べ終わったのでcloseに追加する。Hollywoodの子ノードであるUCLAは既にopenに追加してあるので無視

動作例(幅優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

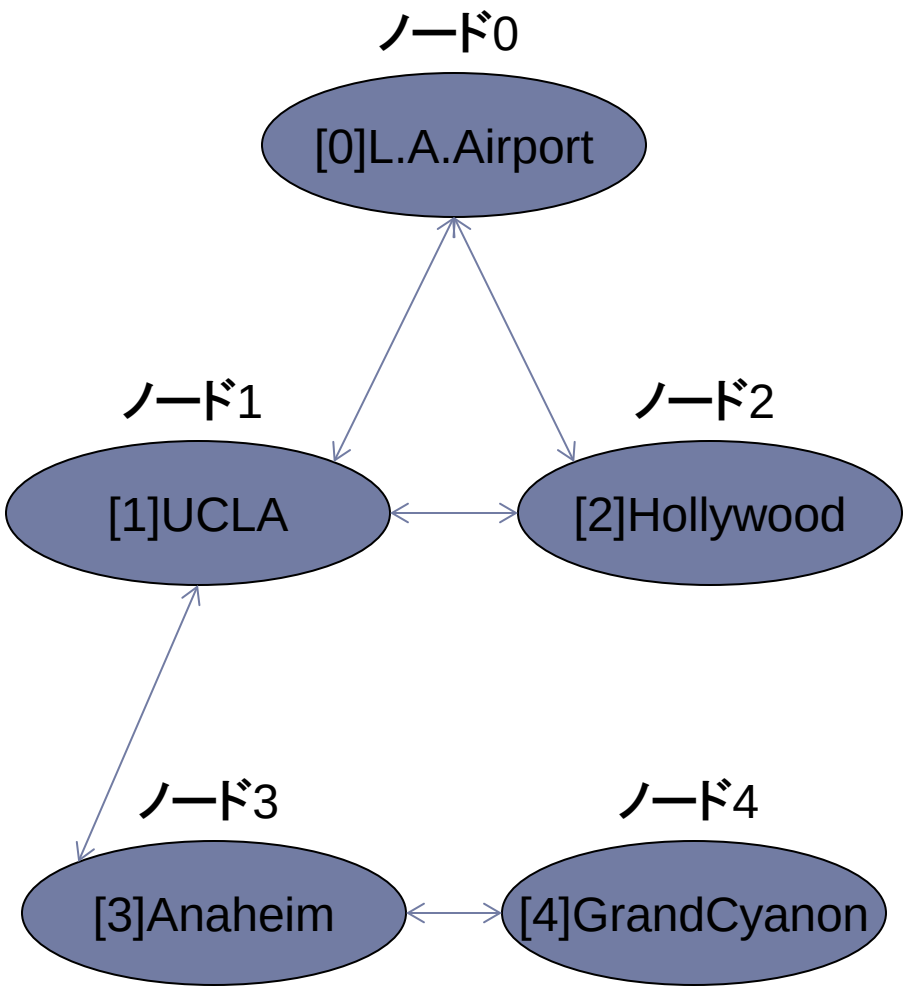
STEP:4

Open={GrandCyanon}

Closed={L.A.Airport,UCLA,Hollywood,Anaheim}

※Anaheimは調べ終わったのでcloseに追加し、子ノードであるGrandCyanonをopenに追加する
GOALが見つかったので終了

動作例(深さ優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

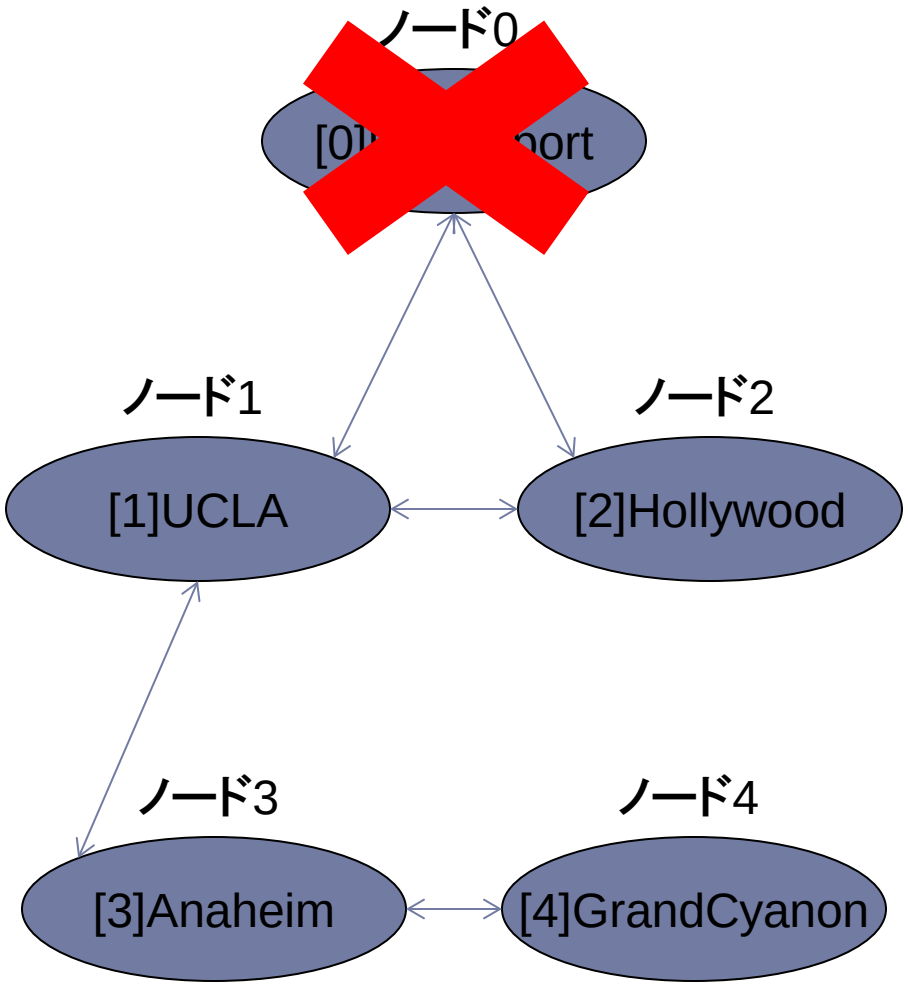
STEP:0

Open={L.A.Airport}

Closed={}

※Openはこれから調べるNode
※Closeは調べ終わったNode

動作例(深さ優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

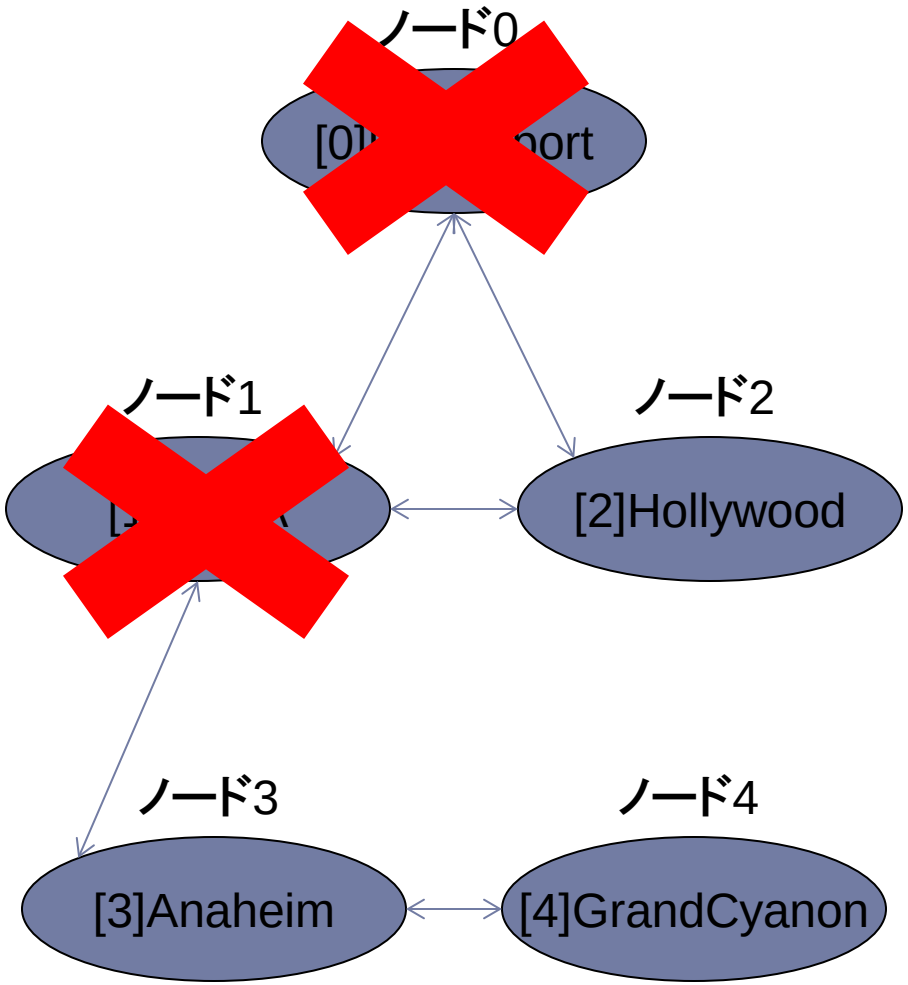
STEP:1

Open={UCLA,Hollywood}

Closed={L.A.Airport}

※L.A.Airportは調べ終わったのでcloseに追加し、L.A.Airportの子ノードであるUCLAとHollywoodをopenに追加

動作例(深さ優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

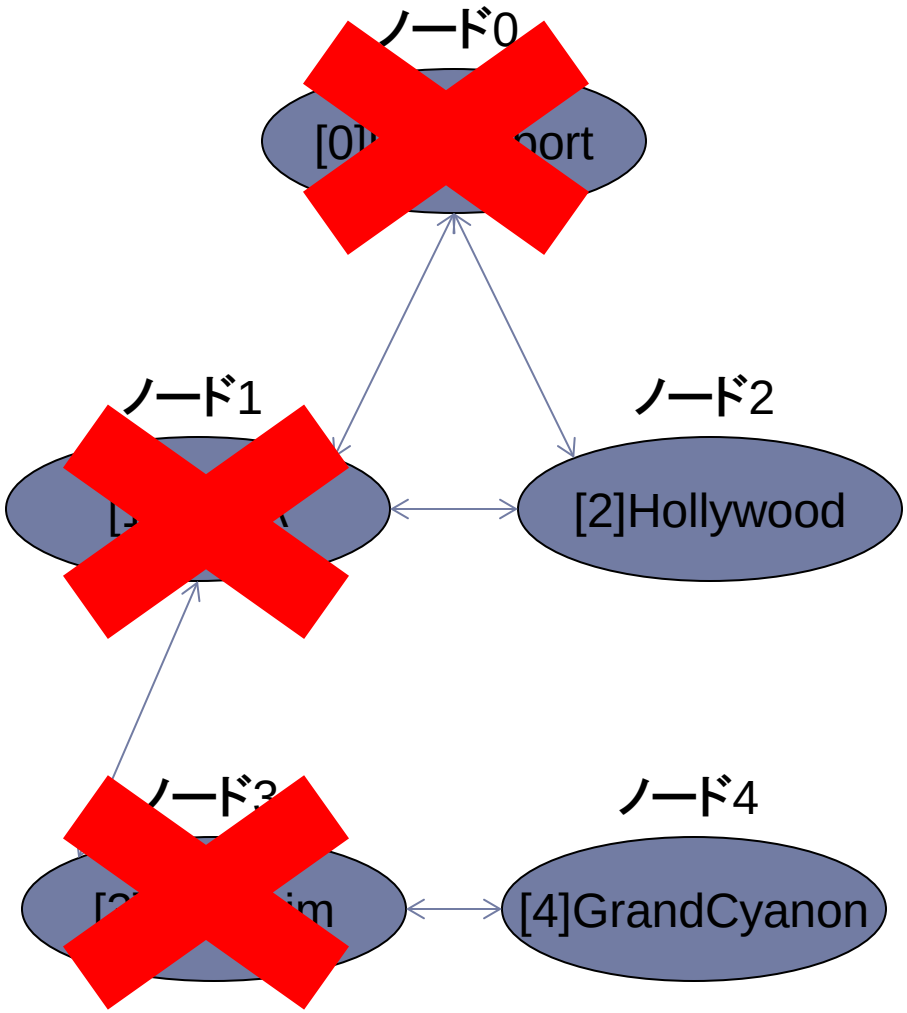
STEP:2

Open={Anaheim,Hollywood}

Closed={L.A.Airport,UCLA}

※UCLAは調べ終わったのでcloseに追加し、UCLAの子ノードであるAnaheimをopenの先頭に追加。Hollywodは既にopenに追加してあるので無視

動作例(深さ優先探索)



- START=L.A.Airport
- GOAL=GrandCyanon

STEP:3

Open={GrandCyanon,Hollywood}

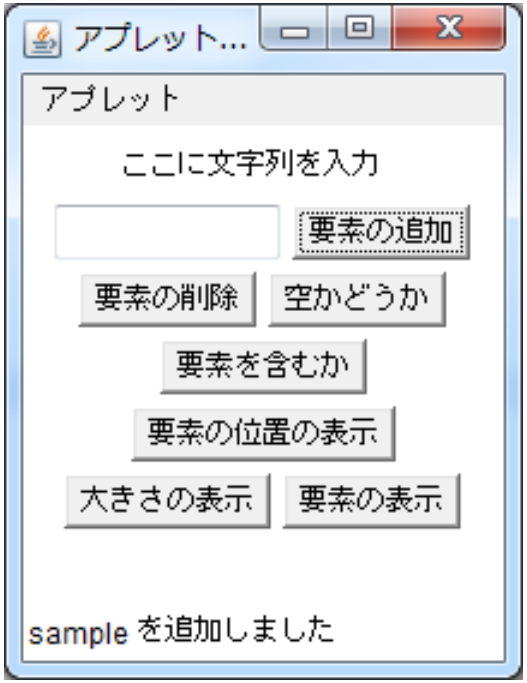
Closed={L.A.Airport,UCLA,Anaheim}

※Anaheimは調べ終わったのでcloseに追加する。Anaheimの子ノードであるGrandCyanonをopenの先頭に追加するGOALが見つかったので終了

練習 – 1

- ArrayTest.javaのコメントを見ながらコードを書き加え、正しく実行できるようにしてみましょう。

(add “全消去” button function)

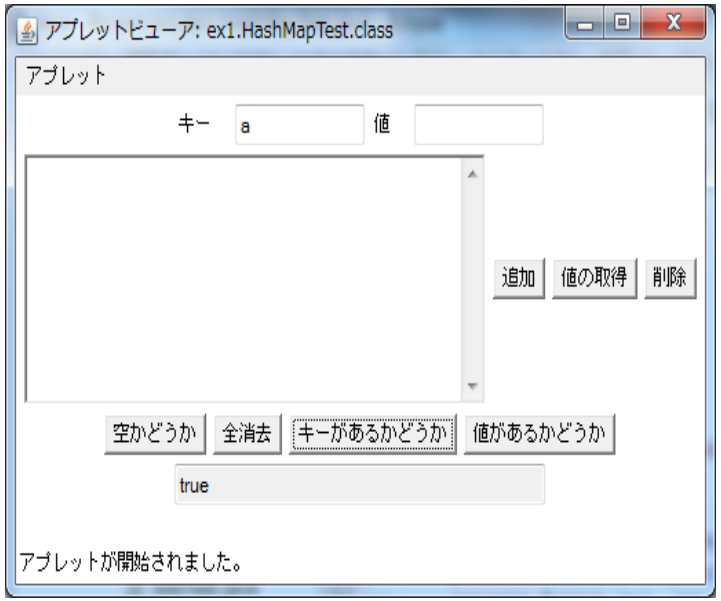


実行例

練習 – 2

- HashMapTest.javaのコメントを見ながらコードを書き加え、正しく実行できるようにしてみましょう。

(Add 要素の位置の表示)



宿題

- SearchingApplet.java (L3-toStudent.zip) をダウンロードして、やってみましょう

- 特に、以下関連のソースコードを理解してください。
  1. Applet上に複数の都市を表示する
  2. 幅優先探索を実装する

しなさい：
深さ優先探索を実装する。（都市名（node names）と都市数(node number) – can be changed to what you like）.
ほかの探索方法と応用例を考えてください。

参考サイト

http://el-tramo.be/jsearchdemo/