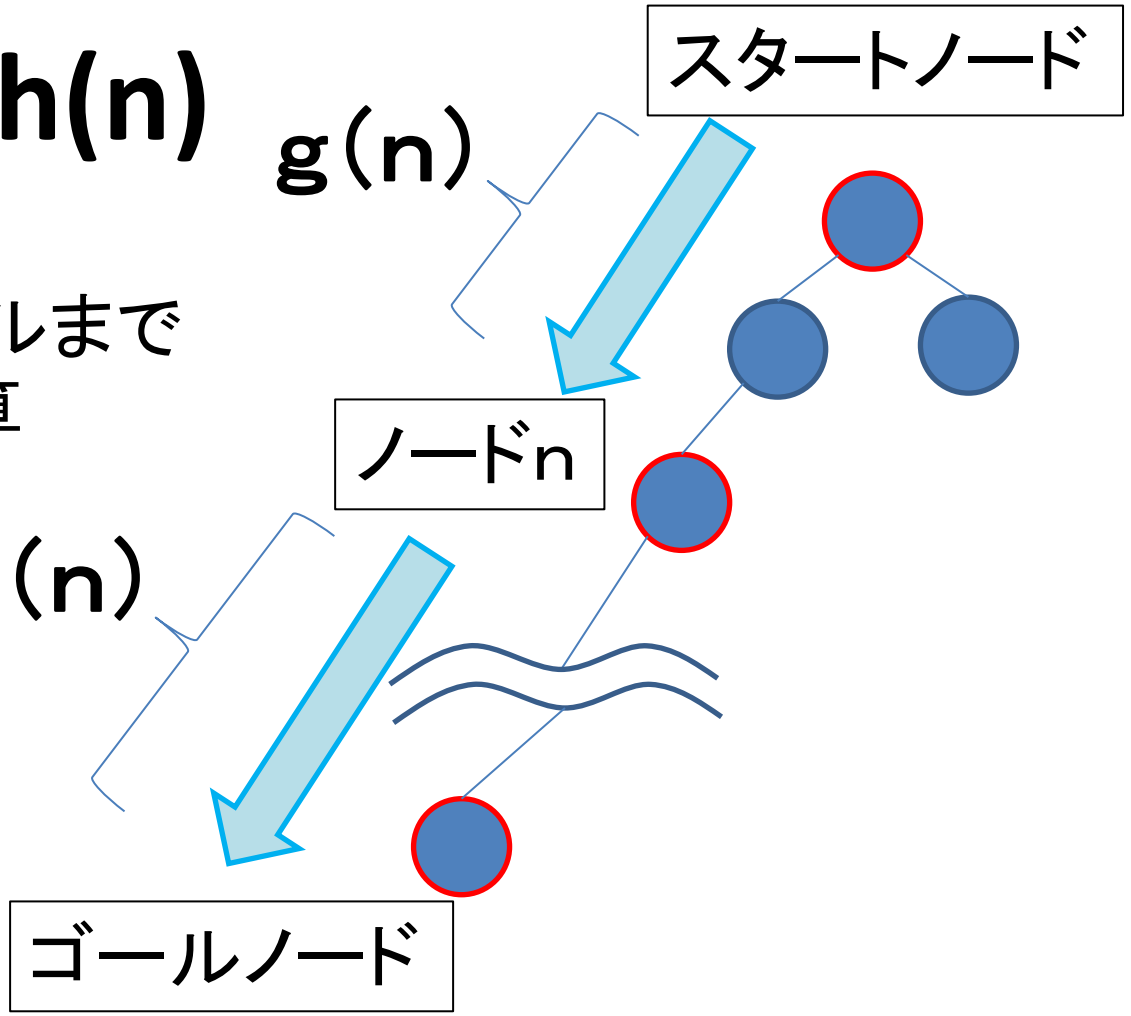
A*探索

A*の考え方①

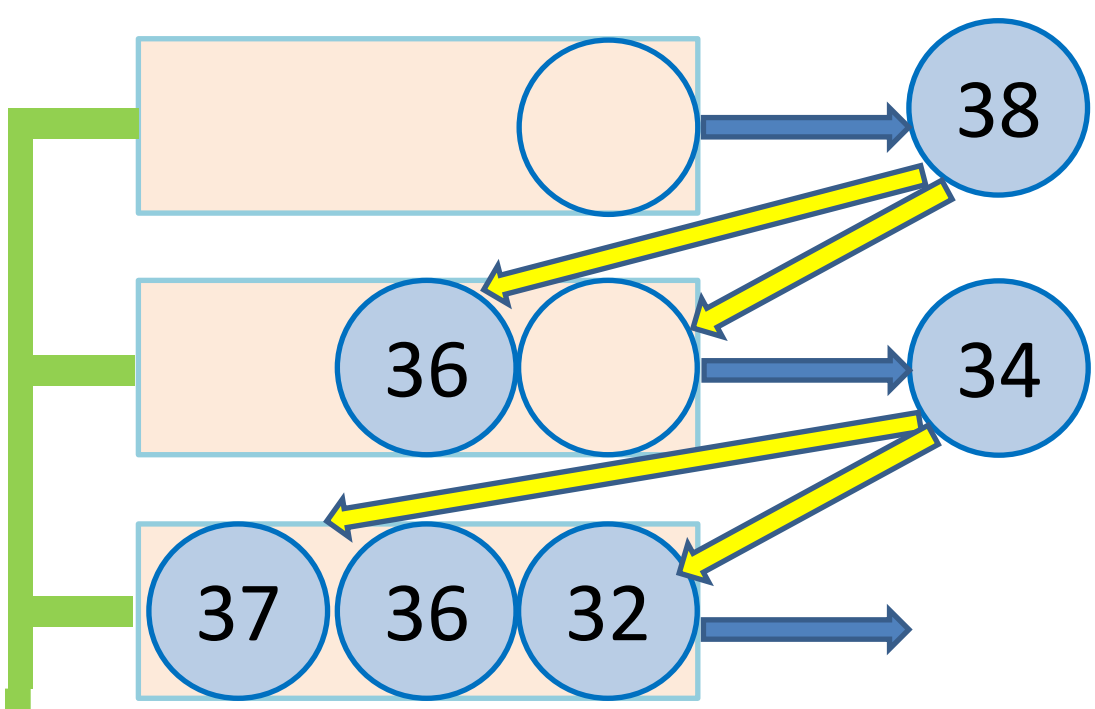
- **f(n)=g(n)+h(n)**

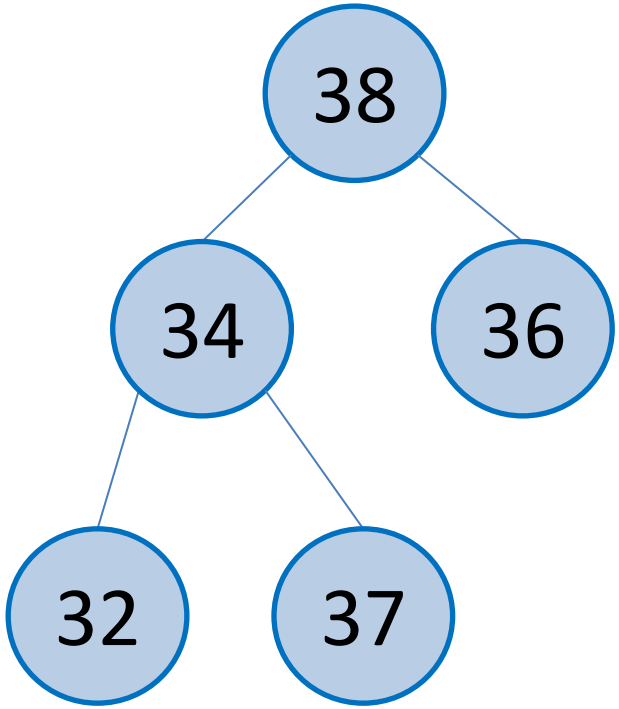
h(n)はnからゴールまでの推定コストを計算



A*の考え方②

●より良いコストを持っているノードを見ていく





PriorityQueue(優先度付きキュー)

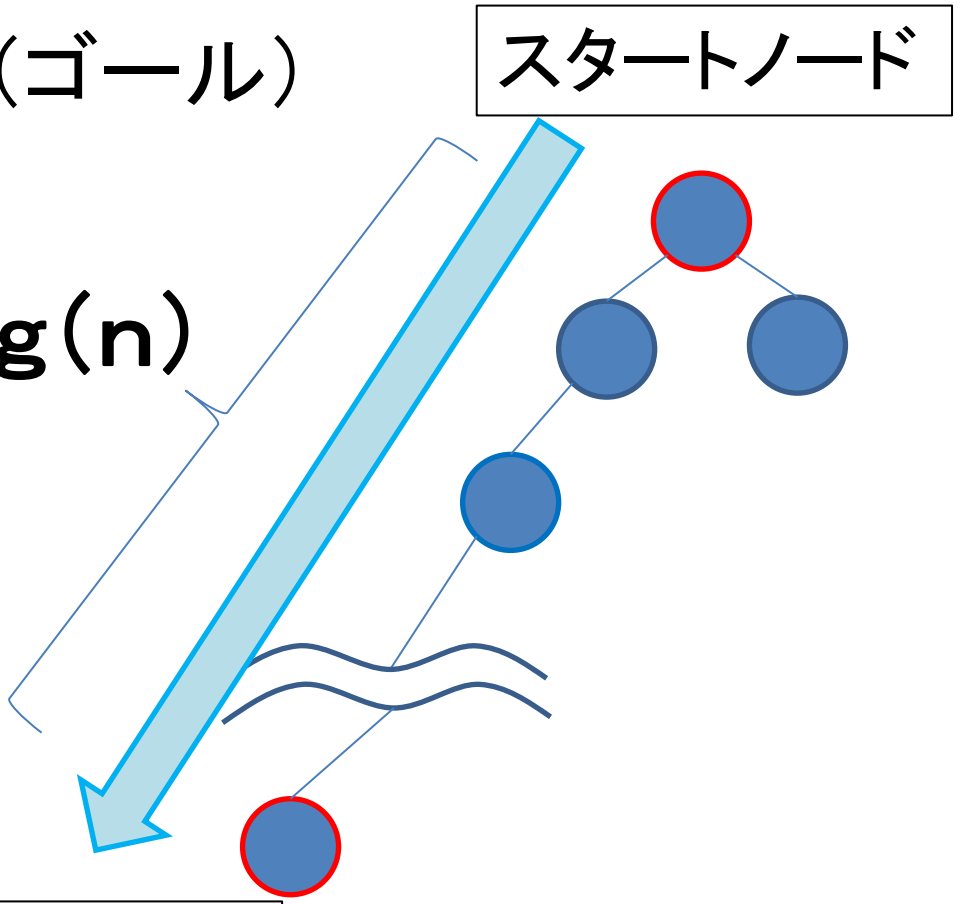
A*の考え方③

●h＝0なら探索終了（ゴール）

f(n)＝g(n)

g(n)

h(n)＝0

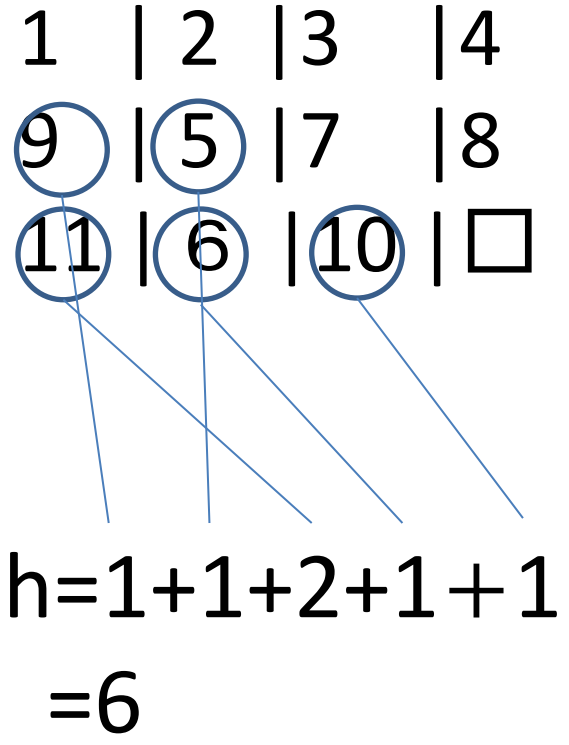


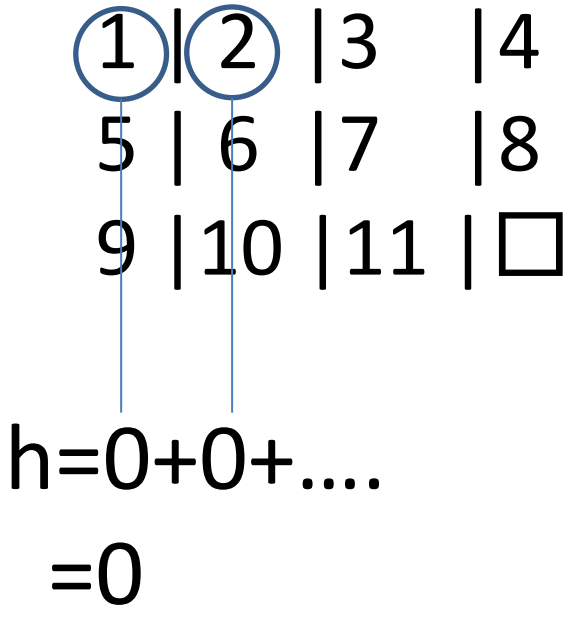
ノードn ＝ ゴールノード

１１パズルの例

完成形

1 | 2 |3 |4
5 | 6 |7 |8
9 |10 |11 |□





実装(11パズル)

- Puzzle11(main)クラス
- Comparableインターフェースを実装したNodeクラス

Comparableインターフェース

- 自然順序付けをする

- int compareTo(T o)

このオブジェクトと指定されたオブジェクトの順序を比較

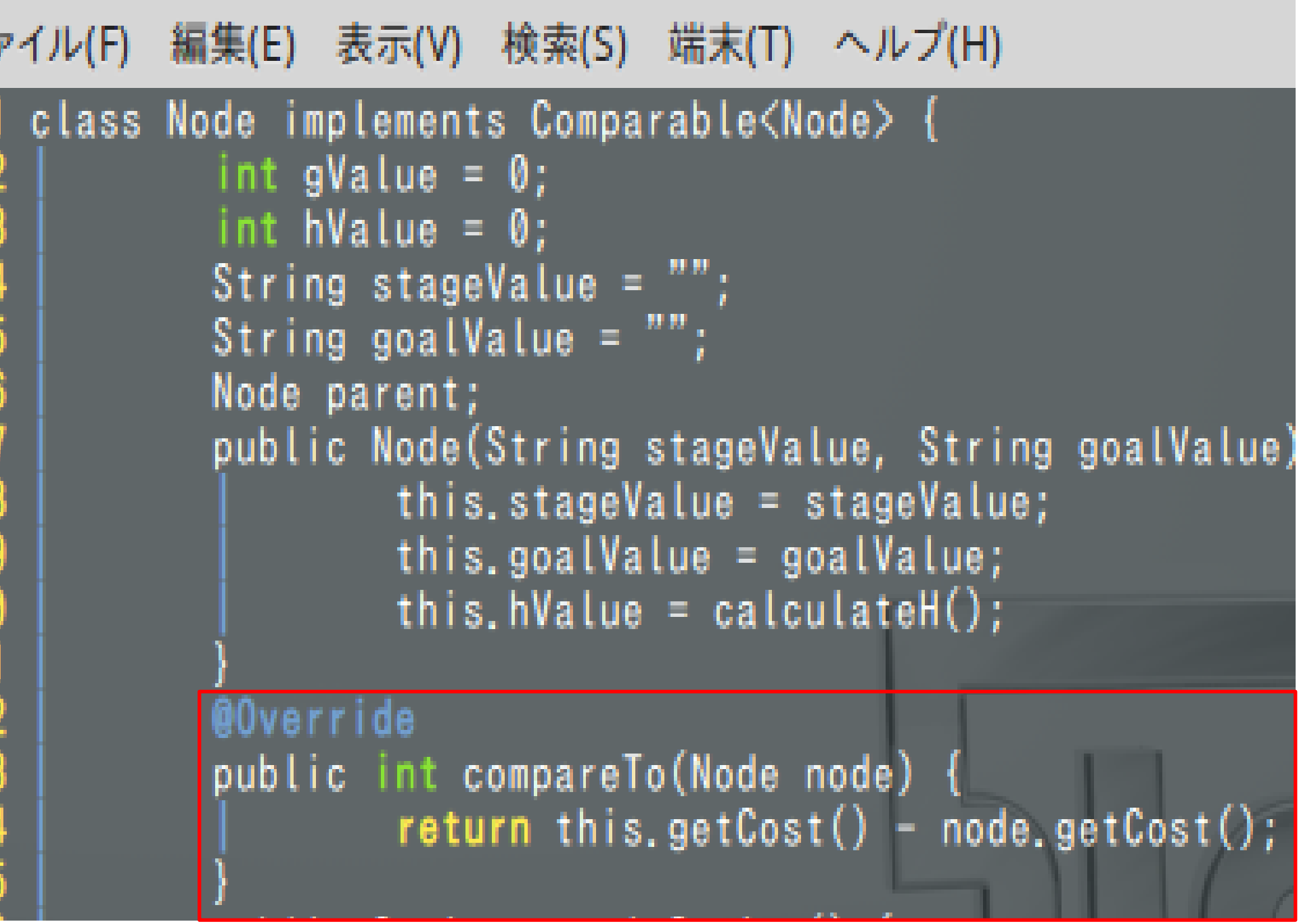
このオブジェクトの方が小さい場合　→ 戻り値 -1
等しい場合　→ 戻り値 0
大きい場合　→ 戻り値 1

PriorityQueueのコンストラクタ

引数なしPriorityQueue()

…自然順序付けに従って要素を順序付け

```
PriorityQueue<Node> pq = new PriorityQueue<>();
```


ァイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
class Node implements Comparable<Node> {
int gValue = 0;
int hValue = 0;
String stageValue = "";
String goalValue = "";
Node parent;
public Node(String stageValue, String goalValue)
this.stageValue = stageValue;
this.goalValue = goalValue;
this.hValue = calculateH();
}
@Override
public int compareTo(Node node) {
return this.getCost() - node.getCost();
}

```
class Puzzle11 {
        public static void main(String[] args) {
                PriorityQueue<Node> open = new PriorityQueue<>();
                boolean answer = false;
                String goalValue = "abcdefghijkl";
                String startValue = "";
                String fileName = "";
                Scanner sc = new Scanner(System.in);
                System.out.print("ファイル名：");
                try {
                        fileName = sc.next();
                } catch(Exception e) {}
                try {
                        sc = new Scanner(new File(fileName));
                } catch(Exception e) {}
                while(sc.hasNext()) startValue += sc.next();
```

a	b	c	d
e	f	g	h
i	j	k	l

1	2	3	4
5	6	7	8
9	10	11	□

```
Node first = new Node(startValue, goalValue);
System.out.println("ステージ初期状態");
System.out.println(first.toString());
open.add(first);
Node node;

while(open.size() > 0) {
        node = open.poll();
        if(node.hValue == 0) {
                System.out.println(node.resultString());
                answer = true;
                break;
        }
        for(Node child : node.openChild()) {
                open.add(child);
        }
}
if(!answer) System.out.println("解なし");
```

- open（優先度付きキュー）にfirstノードを追加
- hが0になるかopen内のノードがなくなるまで以下ループ

1. openからノードを取ってくる
2. 取ってきたノードの子ノードを展開し、openに加える

```
41 class Node implements Comparable<Node> {
42         int gValue = 0;
43         int hValue = 0;
44         String stageValue = "";
45         String goalValue = "";
46         Node parent;
47         public Node(String stageValue, String goalValue) {
48                 this.stageValue = stageValue;
49                 this.goalValue = goalValue;
50                 this.hValue = calculateH();
51         }
```

gValue…g(n)の値
hValue…h(n)の値
parent…親ノード
calculateH()…h(n)の値を求めるメソッド

```
public int calculateH() {
        int h = 0;
        for(int i = 0; i < goalValue.length(); i++) {
                int j = goalValue.indexOf(stageValue.charAt(i)+"");
                h += Math.abs(i-j)/4 + Math.abs(i-j)%4;
        }
        return h;
}
```

stageValue

10	2	3	4
5	6	7	8
9	1	11	□

goalValue

1	2	3	4
5	6	7	8
9	10	11	□

9 ----- 1のindex ----- 0

0 ----- 10のindex ----- 9

h=3+3

=6

```
public List<Node> openChild() {
        List<Node> child = new ArrayList<>(4);
        int from = this.stageValue.indexOf("l");
        int[] d = {from-4, from-1, from+1, from+4};
        for(int to : d) {
                if(
                        to >= 0 && to < 12 &&
                        !(to==3 && from==4) &&
                        !(to==4 && from==3) &&
                        !(to==7 && from==8) &&
                        !(to==8 && from==7)
                ) {
                        Node n = new Node(swap(from, to, this.stageValue), goalValue);
                        n.parent = this;
                        if(this.parent != null) {
                                if(n.stageValue == this.parent.stageValue) continue;
                        } else {
                                if(n.stageValue == this.stageValue) continue;
                        }
                        n.gValue = (this.parent!=null ? this.gValue : 0) + 2;
                        child.add(n);
                }
        }
        return child;
}
```

1	2	10	4
5		7	8
9	11	3	6

1.空き場所のindexをfromに格納
2.交換出来る4方向をto[]に
3.交換出来たら子ノード作成
4.子ノードのg(n)とparent設定