

問 3 スライドパズルを解くプログラムに関する次の記述を読んで、設問に答えよ。

表 1 のルールで定義されるスライドパズルについて考える。

表 1 スライドパズルのルール

用語	説明
盤面	正方形の面に、1 から順に 1 ずつ大きくなる数字の書かれた駒が配置されている。1 か所だけ駒の置かれていない空白のマス（以下、空白マスという）がある。
開始時点の盤面	開始時点では、駒と空白マスはランダムに配置されている。
ゴールの盤面	盤面左上に数字が 1 の駒があり、右の駒の数字は 1 ずつ増えていき、その行の右端の駒に書かれた数字より 1 大きい数字が、次の行の左端の駒の数字となる。盤面右下のマスが空白マスとなる。
駒の移動	駒の上下左右のいずれかに空白マスがある場合、駒を空白マスに移動することができる。駒が移動した場合、駒が置かれていたマスが空白マスになる。駒を空白マスに移動させて盤面を変化させ、ゴールの盤面と同じにすることを目指す。

一辺が 3 マスのスライドパズルの例を図 1 に示す。

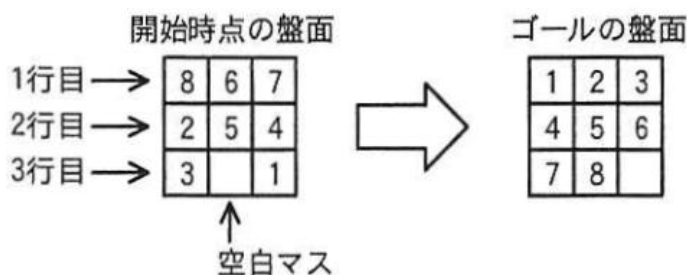


図 1 一辺が 3 マスのスライドパズルの例

本問では、一辺のマス目の個数が任意のスライドパズルにおいて、ゴールの盤面になるまでの駒の移動回数が最小となる移動方法（以下、最小解という）を一つ求めるプログラムを作成する。

〔一辺が N マスのスライドパズルの最小解を幅優先探索を用いて求める方法〕

幅優先探索を行ったときの、スライドパズルの盤面の遷移を、グラフで表現する。開始時点の盤面をルートノード、ある時点の盤面をノード、駒の移動に伴う盤面の遷移をエッジで表現する。また、ゴールの盤面をゴールノードとして定義する。

幅優先探索で最小解を求める方法を次のように考える。ここで、 N は 2 以上とする。探索対象のノードに対して、移動できる駒ごとにその駒を移動した後のノードを作成して、探索対象のノードの子ノードとし、探索対象のノードは探索済みとなる。このとき、子ノードが表す盤面が既に探索したノード（以下、探索済みノードという）と同じであれば、この子ノードは終端ノードとして、以降の探索は行わない。また、子ノードがゴールノードと同じであれば、最小解が見つかったと判断して探索を終了する。

〔 N が 3 の場合の例〕

一辺が 3 マスのスライドパズルの最小解を求める過程の例を図 2 に示す。

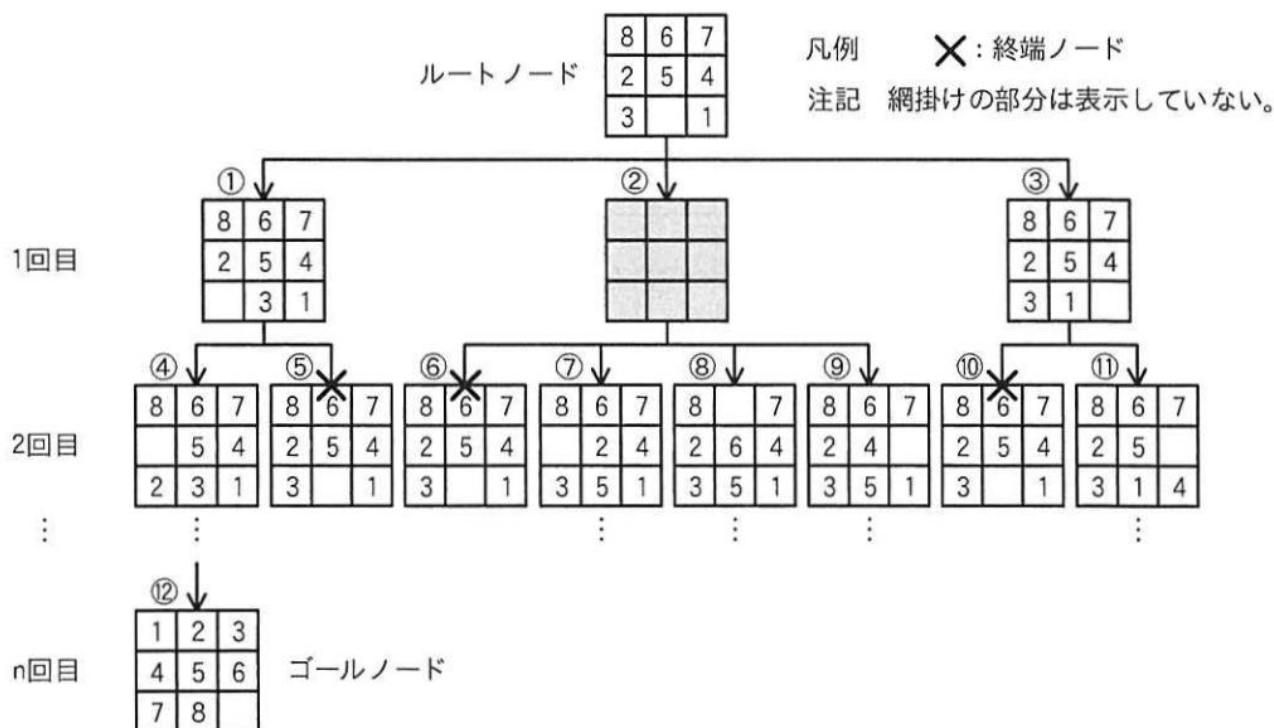


図 2 一辺が 3 マスのスライドパズルの最小解を求める過程の例

(1) 1 回目の駒の移動では、ルートノードを探索対象のノードとする。ここでは、移動できる駒が三つあるので、ルートノードから深さが 1 となる①～③の子ノードを作成し、ルートノードは探索済みノードとなる。ここで、①～③の子ノードに対して、ゴールノードの判定及び終端ノードの判定を行う。

(2) 次に、作成した子ノードで終端ノード以外のノードをそれぞれ探索対象のノードとし、駒の移動にあわせて子ノードを作成する。図 2 では、①～③のノードから④～⑪の子ノードを作成し、①～③は探索済みノードとなる。ここで、④～⑪の子ノードに対して、ゴールノードの判定及び終端ノードの判定を行う。図 2 では、盤面が探索済みノードと一致する⑤、⑥及び⑩は終端ノードとなり、以降の探索は行わない。

(3) 以降、(2)で作成したノードの子ノードの作成と、ゴールノードの判定及び終端ノードの判定を繰り返す。図 2 の⑫は、n 回目の移動でゴールノードに至ったことを示している。なお、全てのリーフノードが終端ノードと判定された場合、ゴールの盤面に至る駒の移動方法がないことを意味しているので、その旨のメッセージを出力して探索を終了する。

次に、配列を用いた盤面の表現方法を図 3 に示す。ここで、空白マスを表す値は最も大きい駒の数字である 8 に 1 を加えた 9 とする。なお、配列の要素番号は 1 から始まるものとする。

盤面を次の例に示すように配列を用いて表現する。

(1) 盤面の行ごとに並ぶ駒の数字を整数型の配列で表す。

3列目

2列目

1列目

8	6	7
2	5	4
3	9	1

→ 1行目を表す配列 {8, 6, 7}

→ 2行目を表す配列 {2, 5, 4}

→ 3行目を表す配列 {3, 9, 1}

(2) (1)で行ごとに定義した配列を整数型配列の配列（以下、盤面配列という）で表す。
{ {8, 6, 7}, {2, 5, 4}, {3, 9, 1} }

(3) 1行目の3列目の駒の数字は、盤面配列名をboardとするとboard[1][3]として参照する。

図 3 配列を用いた盤面の表現方法

〔一辺が N マスのスライドパズルの最小解を求めるプログラム〕

一辺が N マスのスライドパズルにおいて、開始時点の盤面をランダムに作成し、最小解を求め、開始時点の盤面からの遷移及び駒の移動回数を出力するプログラムを作成する。開始時点からの盤面の遷移を保持する単方向連結リストの要素となるクラス BoardState の説明を図 4 に、キューを実現するクラス Queue の説明を図 5 に、リストを実現するクラス List の説明を図 6 に、プログラムで使用する主な関数を表 2 に、最小解を求めるプログラムを図 7 に示す。

メンバ変数	型	説明
board	整数型配列の配列	盤面配列。初期状態は未定義。
space	整数型の配列	空白マスの場所を示す行番号と列番号の二つの要素から成る配列。初期状態は未定義。
prev	BoardState	駒が 1 回移動する前の BoardState のインスタンスへの参照。開始時点の盤面からの遷移を出力する際に用いる。初期状態は未定義。

コンストラクタ	説明
BoardState()	インスタンスを初期化する。
BoardState(BoardState: st)	インスタンスを初期化し、メンバ変数に次の処理を行う。 (1) メンバ変数 board に引数で渡された BoardState のインスタンスの board の値を格納する。 (2) メンバ変数 space に引数で渡された BoardState のインスタンスの space の値を格納する。 (3) メンバ変数 prev に引数で渡された BoardState のインスタンスへの参照 st を格納する。

図 4 クラス BoardState の説明

コンストラクタ	説明
Queue()	可変長のキューを生成する。

メソッド	戻り値	説明
add(BoardState: st)	なし	キューに BoardState のインスタンスへの参照 st を追加する。
isEmpty()	論理型	キューが空の場合は true を返し、空でない場合は false を返す。
poll()	BoardState	先頭の BoardState のインスタンスへの参照を取り出して返す。
peek()	BoardState	先頭の BoardState のインスタンスへの参照を返す。

図 5 クラス Queue の説明

コンストラクタ	説明
List()	可変長のリストを生成する。

メソッド	戻り値	説明
add(整数型配列の配列: b)	なし	リストに盤面配列 b を追加する。
isEmpty()	論理型	リストが空の場合は true を返し、空でない場合は false を返す。
peek()	整数型配列の配列	リストの先頭に存在する盤面配列を返す。

図 6 クラス List の説明

表2 プログラムで使用する主な関数

名称	戻り値	説明
createGoal(整数型: board_size)	整数型配列の配列	引数 board_size の値を一辺のマスの数としたスライドパズルのゴールの盤面を表す配列（以下、ゴールの盤面配列という）を作成して返す。
createStart(整数型: board_size)	整数型配列の配列	引数 board_size の値を一辺のマスの数としたスライドパズルの開始時点の盤面配列をランダムに作成し、出力して返す。なお、ゴールの盤面配列と同じものが作成されることはない。
getSpace(整数型配列の配列: b)	整数型の配列	盤面配列 b を用いて、空白マスの場所を行番号と列番号から成る配列で返す。
checkGoal(整数型配列の配列: b, 整数型配列の配列: g)	論理型	第一引数の盤面配列 b と第二引数のゴールの盤面配列 g を用いて両者が一致しているかどうかを判定して結果を返す。一致する場合は true を返し、一致しない場合は false を返す。
printResult(BoardState: st)	なし	引数として与えられた BoardState のインスタンスへの参照 st を用いて、開始時点の盤面からの遷移及び駒の移動回数を出力する。
checkSameBoard(整数型配列の配列: b, List: l)	論理型	引数で渡される探索済みの盤面配列のリストへの参照 l を用いて、盤面配列 b が探索済みの盤面配列のリストに存在するかどうかを判定して結果を返す。存在する場合は true を返し、存在しない場合は false を返す。

1:	○solveNPuzzle(整数型: board_size)	
2:	BoardState: start_state	/* 開始時点の盤面を保持するBoardStateのインスタンスへの参照 */
3:	整数型配列の配列: goal_board	/* ゴールの盤面配列 */
4:	整数型配列の配列: direction	/* 駒の移動に伴う空白マスの移動方向を行番号の増減を1番目の要素、列番号の増減を2番目の要素で表現する配列 */
5:	Queue: explore_queue	/* 探索対象の盤面を保持するキューのインスタンスへの参照 */
6:	List: check_list	/* 探索済みの盤面配列を保持するリストのインスタンスへの参照 */
7:	BoardState: state	/* 駒が移動する前の盤面を保持するBoardStateのインスタンスへの参照 */
8:	BoardState: new_state	/* 駒が移動した後の盤面を保持するBoardStateのインスタンスへの参照 */
9:	整数型: change_nun	/* 移動する駒の数字 */
10:	整数型: i	/* forループ内で使用するカウンタ変数 */

図7 最小解を求めるプログラム

```
11: start_state ← BoardState()
12: goal_board ← createGoal(board_size)
13: start_state.board ← createStart(board_size)
14: start_state.space ← getSpace(start_state.board)
15: direction ← {{1, 0}, {0, -1}, {-1, 0}, {0, 1}}
16: explore_queue ← Queue()
17: explore_queue.add(start_state)
18: check_list ← List()
19: check_list.add(start_state, board)
20: while ( [ ア ] がfalseと等しい)
21:     state ← explore_queue. [ イ ]
22:     for (iを1からdirectionの要素数まで1ずつ増やす)
23:         if ((state.space[1] + [ ウ ] が1以上) かつ
24:             (state.space[1] + [ ウ ] が [ エ ] 以下) かつ
25:             (state.space[2] + [ オ ] が1以上) かつ
26:             (state.space[2] + [ オ ] が [ エ ] 以下))
27:             new_state ← BoardState(state)
28:             new_state.space[1] ← state.space[1] + [ ウ ]
29:             new_state.space[2] ← state.space[2] + [ オ ]
30:             change_num ← new_state.board[new_state.space[1]][new_state.space[2]]
31:             new_state.board[new_state.space[1]][new_state.space[2]] ← board_size × board_size
32:             new_state.board[state.space[1]][state.space[2]] ← change_num
33:             if ( [ カ ] がtrueと等しい)
34:                 printResult(new_state)
35:                 return
36:             endif
37:             if (checkSaneBoard(new_state.board, check_list)がfalseと等しい)
38:                 explore_queue. [ キ ]
39:                 check_list. [ ク ]
40:             endif
41:         endif
42:     endfor
43: endwhile
44: ゴールの盤面に至る駒の移動方法がない旨のメッセージを出力
45: return
```

図7 最小解を求めるプログラム（続き）

設問1 図2中の②の盤面を図3に倣って、配列で答えよ。

設問2 図7中の[ア]～[ク]に入れる適切な字句を答えよ。