知能システム論第 8 回 問 2

05-231001 阿部 桃大

2024 年 1 月 22 日

1　概要

本課題では、反復深化探索法を用いて、$N \times M - 1$ パズルを解くプログラムを作成し、その性能を評価する。

2　実装

プログラムコードは、puzzle.cpp に記載した。以下では、その概要を説明する。

2.1　探索アルゴリズム

探索は反復深化探索法を用いた。深さ優先探索を行う関数 solve を定義し、それを深さを 1 から順に増やしながら呼び出すことで、解を探索する。

解をみつけた際は、移動をキューに追加しながら、親ノードに戻っていく。これを、初期状態に戻るまで繰り返すことで、解を得る。

また、探索中に、同じ状態が出現した場合、すでに探索した深さよりも浅い深さで同じ状態が出現した場合は、探索を打ち切る。

同じ状態の出現を検知するため、状態を int 型の変数に変換する関数 board_to_num を定義した。この関数では、任意の状態を、0 から $(N \times M)! - 1$ までの整数に変換する。この値を、すでに出現した状態の最大深さを記録する map のキーとして用いることで、同じ状態の出現を検知する。

2.2　解なしの場合

探索関数 solve は、解が見つかった場合は true を返し、見つからなかった場合は false を返す。しかし、十分な深さがない場合、解が見つからないことがある。

解なしを検出するにはこの十分な深さをどのように定義するかが問題となる。今回の実装では、前の深さの探索と比較して、探索した状態数が増えていない場合、すべての状態を探索したとみなせるので、解なしと判定している。

3　実行結果

実行結果を以下に示す。

入力

```
4 4 1 2 3 4 0 5 6 7 9 10 11 8 13 14 15 12
```

出力

```
depth 3 failed
checked: 10
5steps

 1  2  3  4
 0  5  6  7
 9 10 11  8
13 14 15 12

 1  2  3  4
 5  0  6  7
 9 10 11  8
13 14 15 12

 1  2  3  4
 5  6  0  7
 9 10 11  8
13 14 15 12

 1  2  3  4
 5  6  7  0
 9 10 11  8
13 14 15 12

 1  2  3  4
 5  6  7  8
 9 10 11  0
13 14 15 12

 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

これは解がある場合の入力で、解の手順が出力されている。

入力

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

出力

```
depth 3 failed
checked: 7
depth 6 failed
checked: 51
depth 9 failed
checked: 268
depth 12 failed
checked: 1102
depth 15 failed
checked: 4767
depth 18 failed
checked: 17402
depth 21 failed
checked: 54802
depth 24 failed
checked: 116088
depth 27 failed
checked: 170273
depth 30 failed
checked: 181217
depth 33 failed
checked: 181440
depth 36 failed
checked: 181440
no answer
```

これは解が無い場合の入力で、最後に no answer と出力されている。

4 考察

4.1 探索アルゴリズム

今回の実装では、反復深化探索法を用いた。反復深化探索のメリットとしては、深さ優先探索のメリットであるメモリ使用量の少なさを保ちつつ、最短経路に近い解を得ることができることが挙げられる。

しかし、本課題では解なしの場合に、解なしであることを出力する必要があるため、探索済みノードを記録する必要がある。このため、メモリ使用量の節約には至っていない。

したがって、メモリ使用量をデメリットとしている幅優先探索を用いたほうが、より良い実装となると考えられる。しかし、幅優先探索の実装においても、すべての状態を調べ尽くしたことをどう検出するかが問題となる。

4.2 計算量

本プログラムの計算量は、以下の通り計算される。

$$(\text{探索空間の広さ}) \times (1\text{ノードの探索にかかる計算量})$$

探索空間の広さは、$0, 1, 2, \cdots, N \times M - 1$ の順列の数で近似される。これは、$(N \times M)!$ である。

1 ノードの探索にかかる計算量は、map のインデックス計算が支配的である。これは、$O(N \times M)$ である。

したがって、本プログラムの計算量は、

$$O((N \times M)! \times N \times M) \sim O((N \times M + 1)!)$$

と計算される。