

対スライドパズル幅型波動砲

住友 孝郎
twitter : @cattaka_net



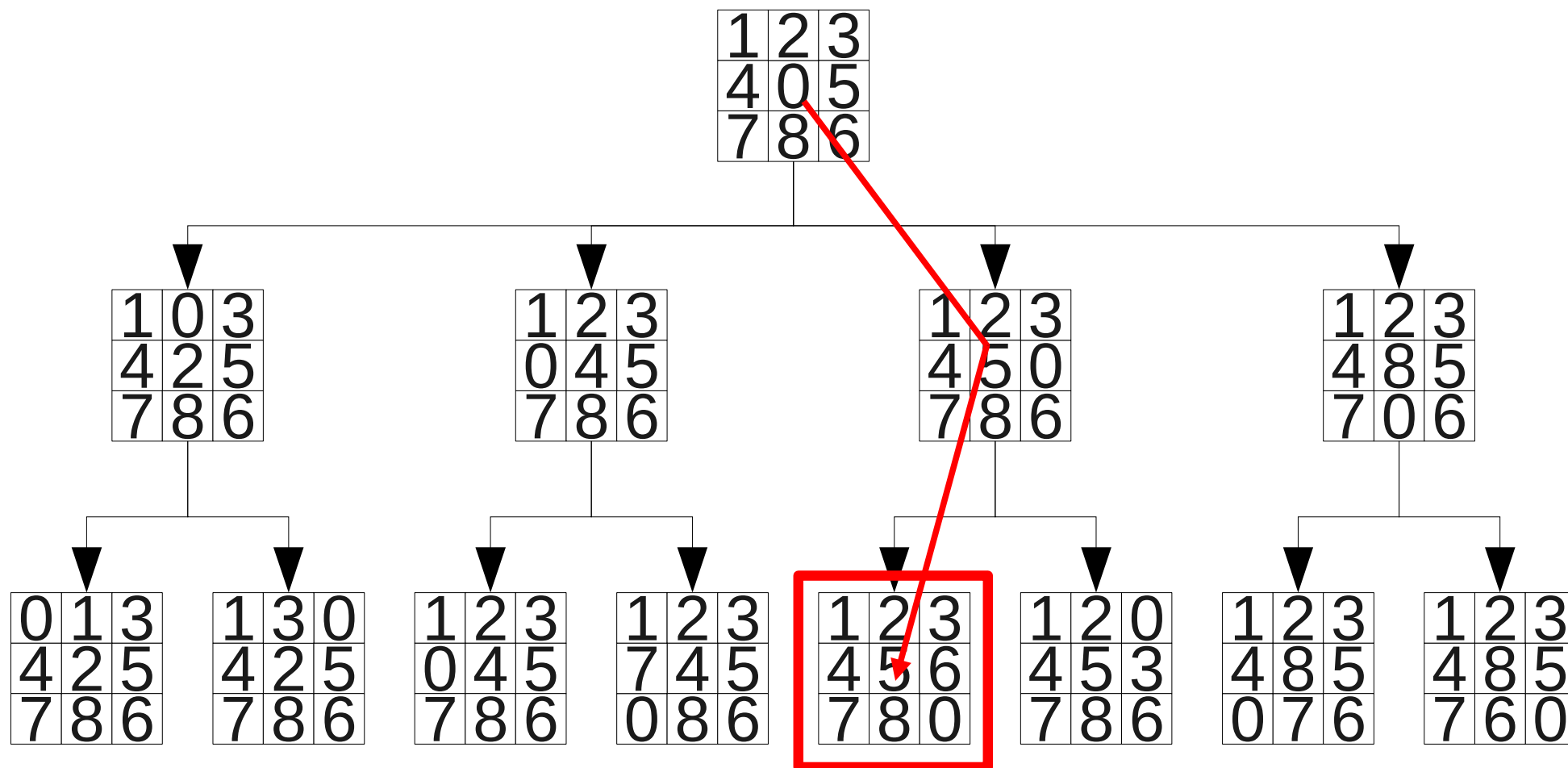
スライドパズル解きました

1	2	3
4	0	5
7	8	6

RD →

1	2	3
4	5	6
7	8	0

探索木にするとこんな感じ



探索問題ですね

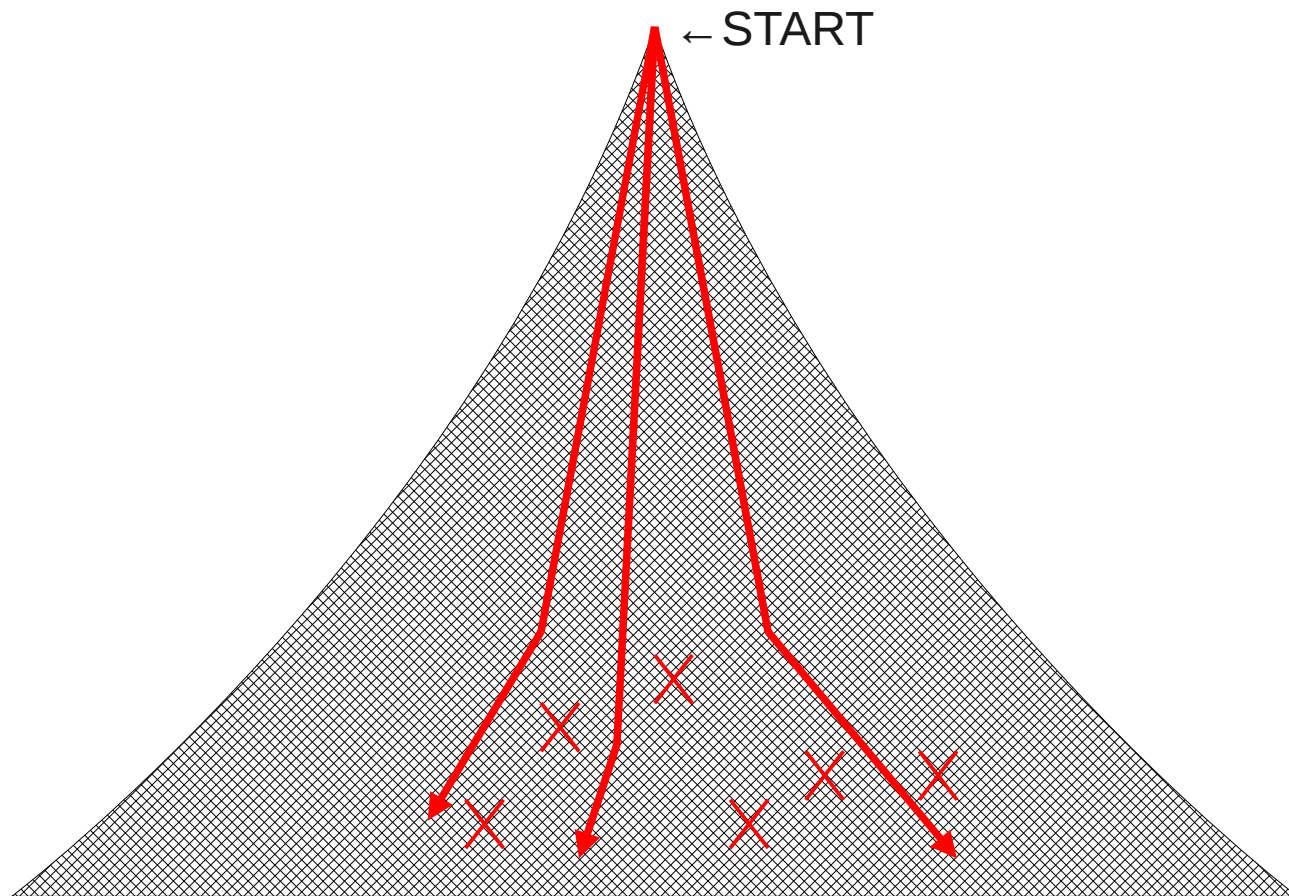
探索空間はかなり広い

- 選べる操作はL,R,U,D(上下左右)の4つ
- すぐに戻ることは(Lの直ぐ後にRなど)無いので実質3つ
- 単純計算で50手先は

$$3^{50} = 2.06 \times 10^{14}$$

100手先になると?...

木にするとこんな感じ



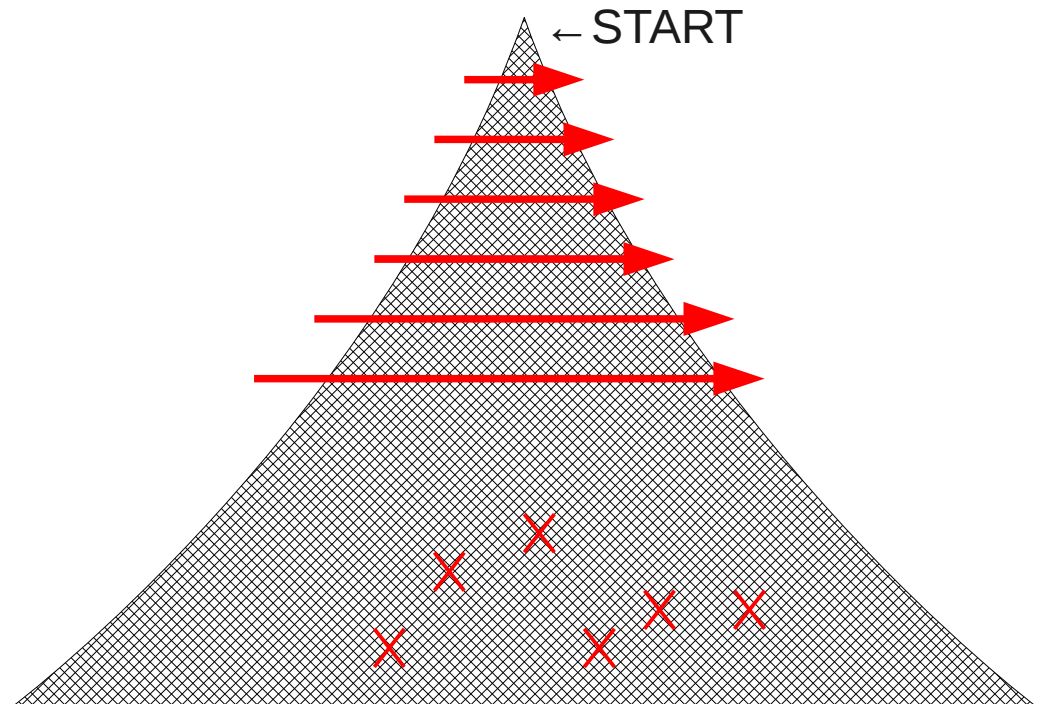
正解には早々当たらない

さあどうしよう・・・

横型探索

- 1手ずつ進めながら探索する
- 進めた手は覚えておく必要がある
- 覚えようとするメモリがオーバーする

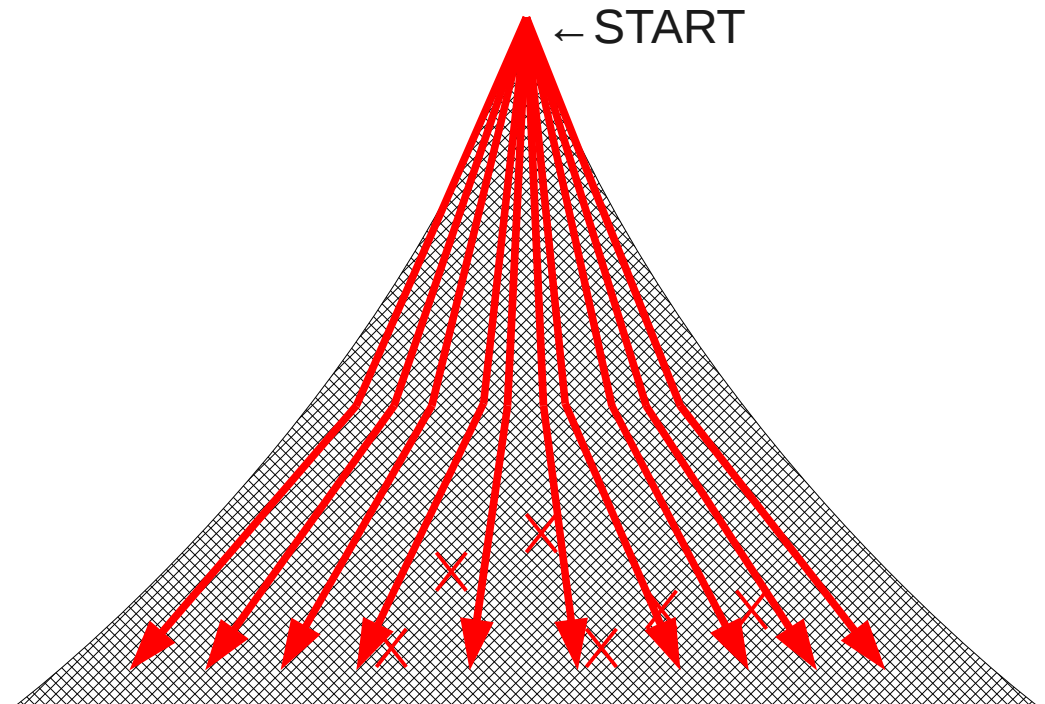
断念



縦型探索

- メモリは使わない
- 深さがわからないと見当も付かない

断念



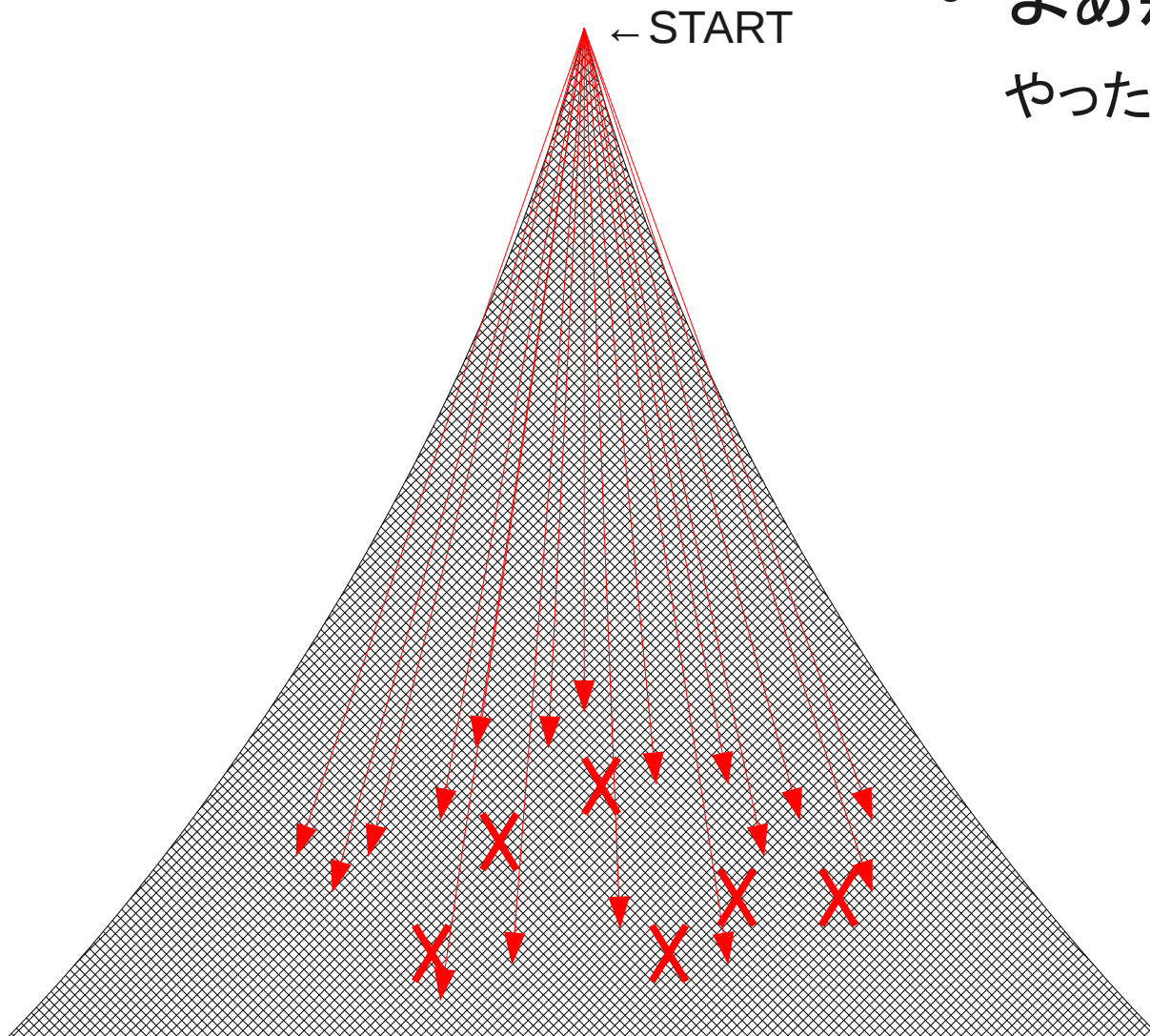
A*探索

- ヒューリスティック関数の定義が難しいよなあ・・・
- 最短でなくてもいいしなあ・・・

断念

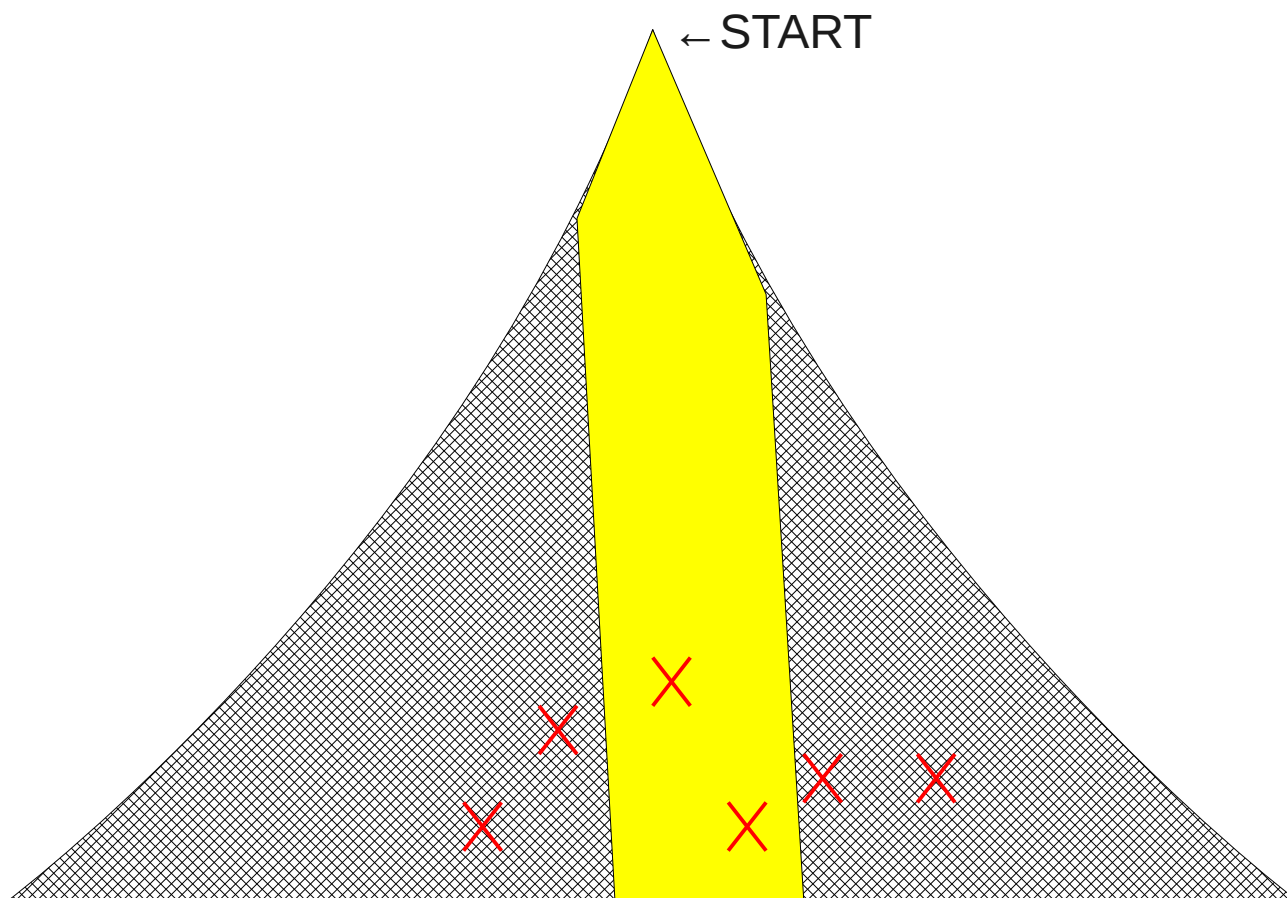
ランダムガトリング砲

- まあ無理だよね
やった人いたけど、、、

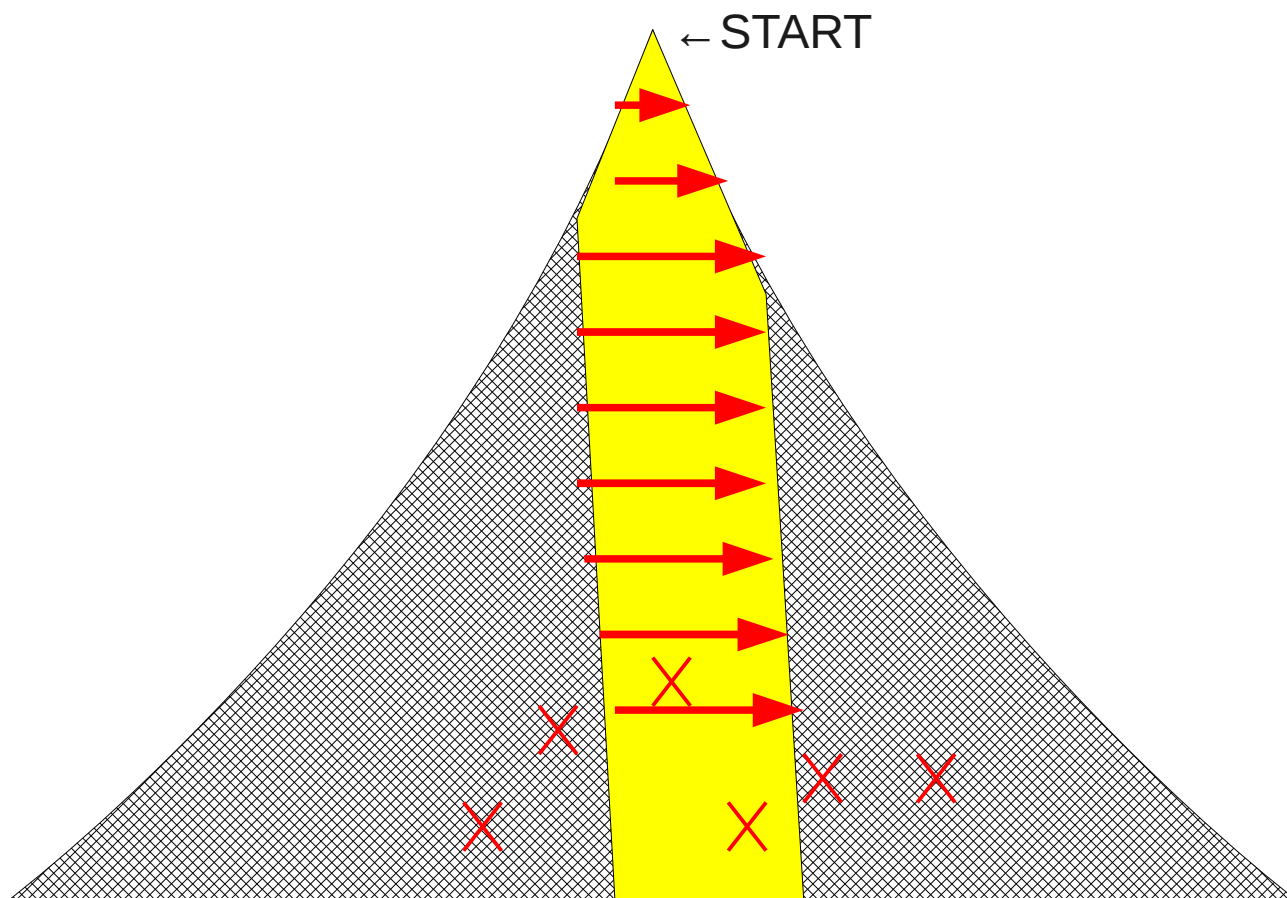


マイアプローチ

こんな風に探索したい



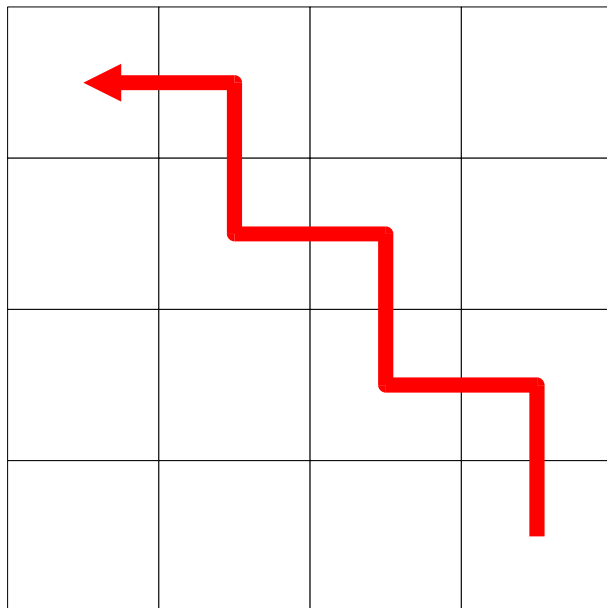
横型の幅を絞り込めばいいんじゃない？



探索空間を吹き飛ばす
幅型波動砲！

これなら評価関数が いい加減でも行ける？

- マンハッタン距離の総和とか



マンハッタン距離はこんなん ↑

評価関数の例

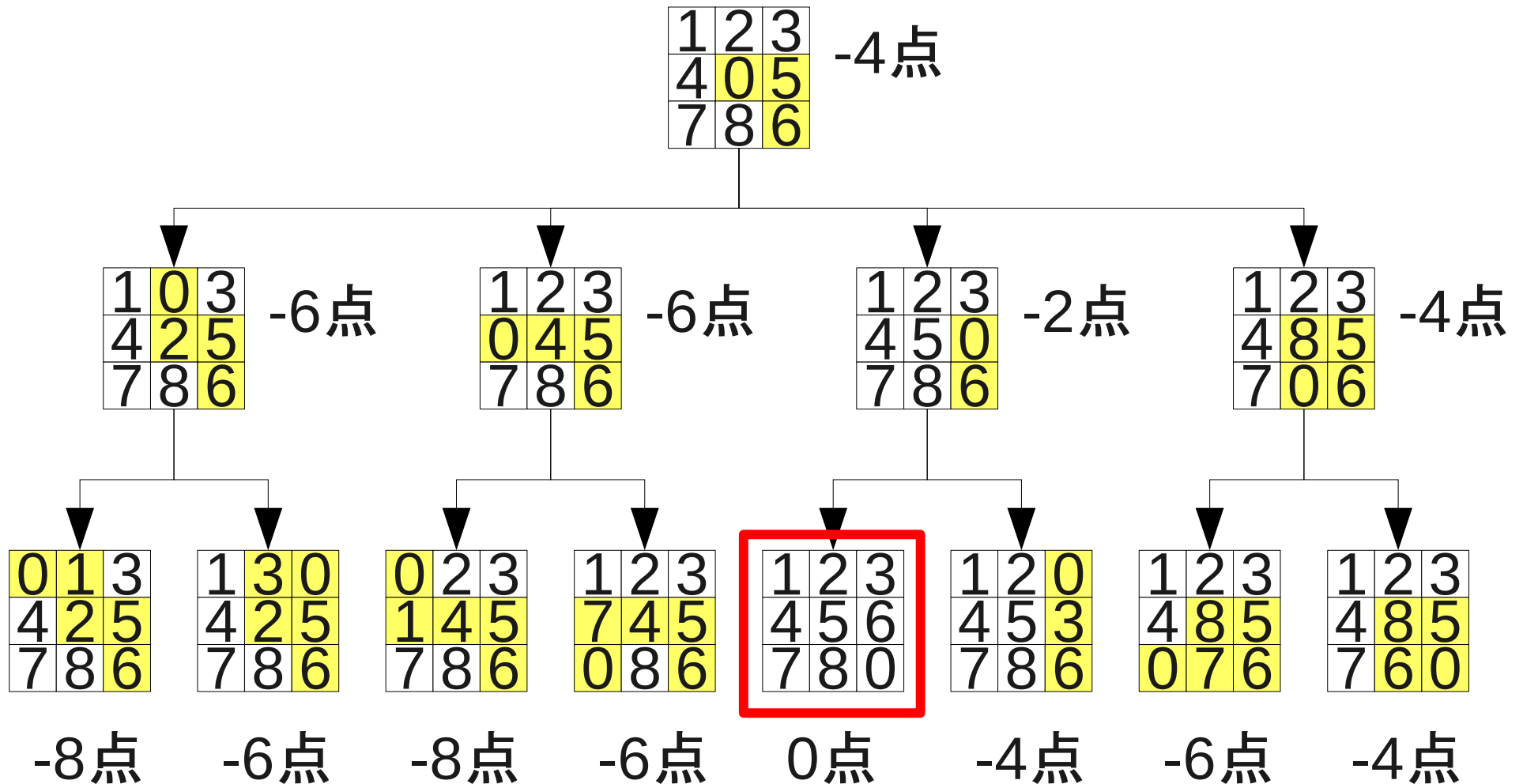
1	2	3
4	5	6
7	8	0

- 全部そろっている
→ 0点(最大)

3	1	2
4	5	6
7	8	0

- そろってない
 - 1が本来の位置から1ズレてる
 - 2が本来の位置から1ズレてる
 - 3が本来の位置から2ズレてる
- -4点

評価関数の例



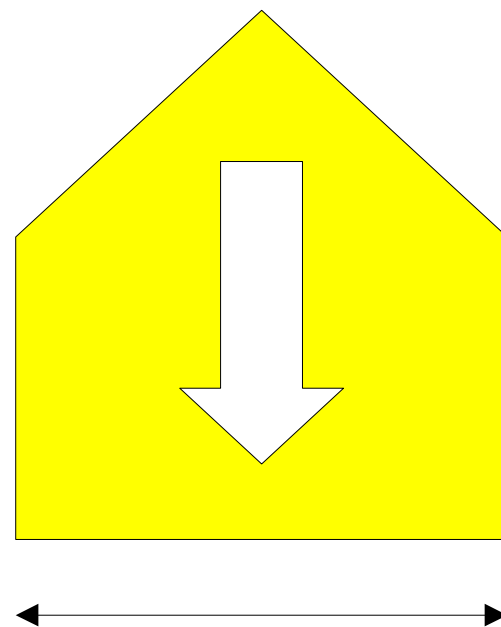
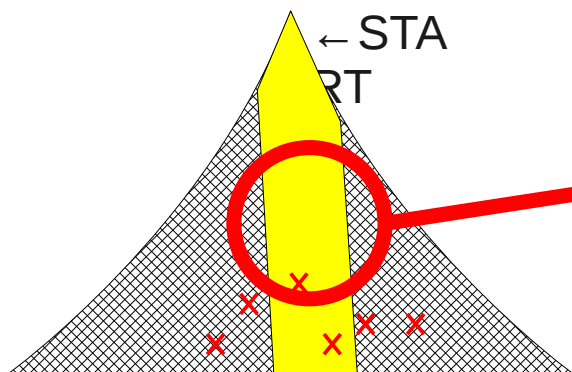
実装について

- Javaで実装しました

試し撃ち

- 結果
 - 1時間で100発
 - そのうち命中は90発超え

- 履歴は100万
- 幅は10万



意外に行けるんじゃない？

でも1時間で100発だと

5000問に50時間掛かる・・・

AWS使いました

- エクストララージ インスタンス
 - CPU : 8 EC2 Compute Unit
 - メモリ : 15GB
- 1プロセスに2GBを割り当てて7プロセスを実行
 - $100[\text{発/時間}] \times 7 = 700[\text{発/時間}]$
 - 7時間強で5000発
 - $\$0.68/\text{時間} \times 7\text{時間} = \$4.76 \div 366 \text{ 円}$
 - 日本だと\$0.80/時間だけど、米国だと少し安い

お昼休みにAWSの インスタンスを作って実行

- 7プロセスを一斉発射

```
# initialize database
```

```
java -classpath slidepuzzle.jar net.cattaka.slidepuzzle.InDataLoader
```

```
# run Solver Type1
```

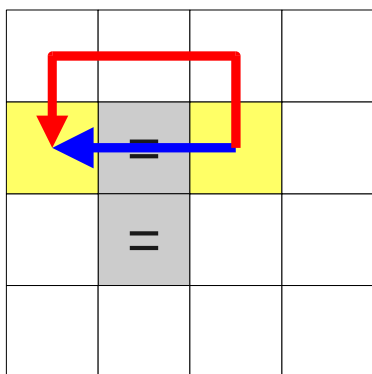
```
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 1 -e 714 > type1_1.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 715 -e 1428 > type1_2.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 1429 -e 2142 > type1_3.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 2143 -e 2857 > type1_4.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 2858 -e 3571 > type1_5.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 3572 -e 4285 > type1_6.log &  
nohup java -XX:-UseGCOverheadLimit -Xms2048m -Xmx2048m -classpath slidepuzzle.jar  
net.cattaka.slidepuzzle.SolverRunner -s 1 -c 100000 -h 1000000 -b 4286 -e 5000 > type1_7.log &
```

ソルバー1号

- 評価関数
 - 壁無視マンハッタン距離の総和
- 幅の大きさ(メモリ2GB)
 - 履歴 : 100万
 - 幅 : 10万
- 結果
 - 命中 : 4948発
 - 使用手数 : 276783手 (308017)
 - 平均 : 55.94手

ちょっと改造

- 壁を無視していたのを考慮するようにした
 - 端っこに重みをつけた
-
- 壁考慮マンハッタン距離 × 重みの総和



壁無視だと距離2

壁考慮だと距離4

4	4	3	3
4	=	2	2
4	=	1	1
4	4	1	1

端っこに重みをつけた

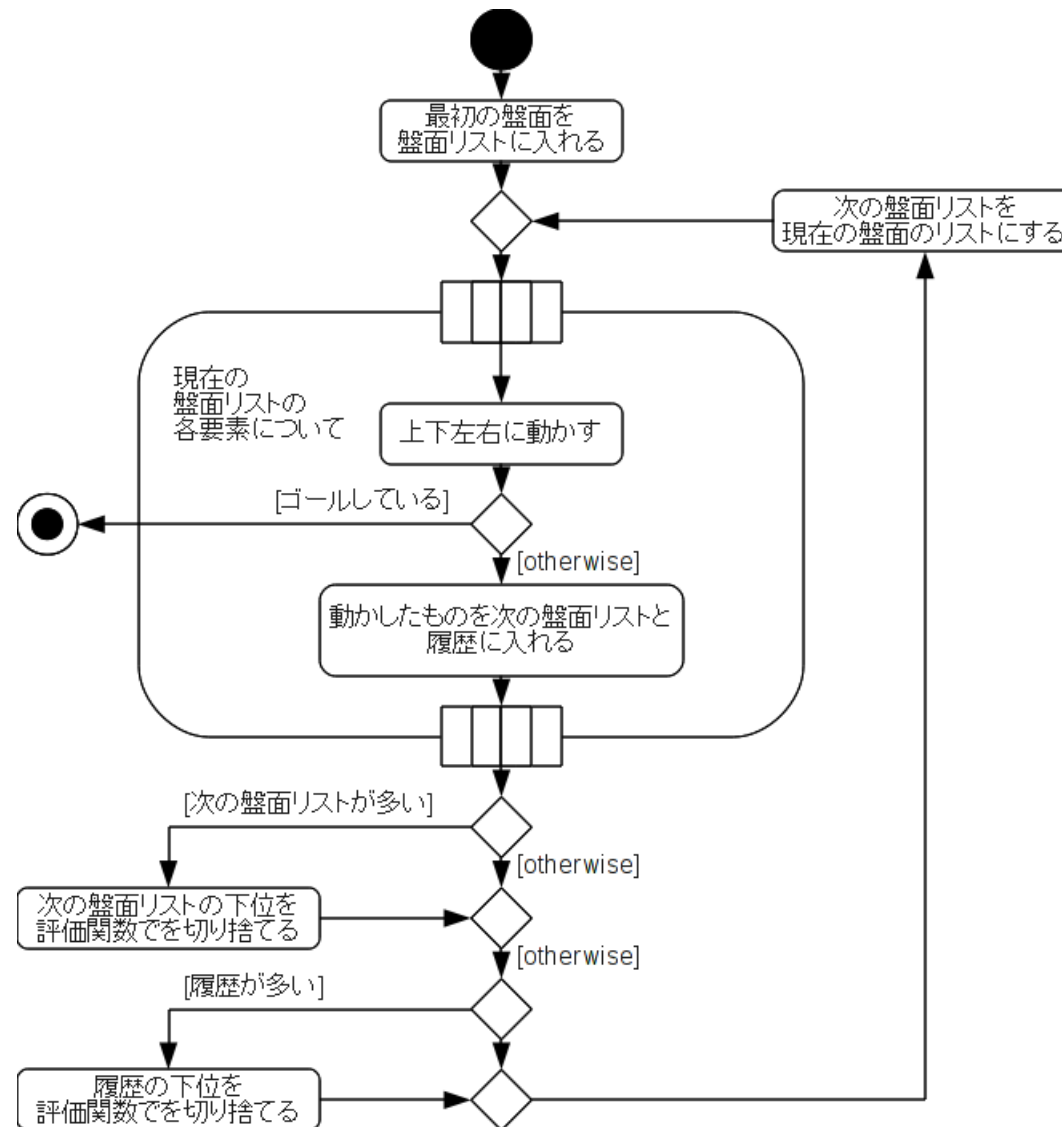
ソルバー2号

- 評価関数
 - 壁考慮マンハッタン距離×重みの総和
- 幅の大きさ(メモリ2GB)
 - 履歴 : 100万
 - 幅 : 10万
- 結果
 - 命中 : 4892発
 - 使用手数 : 272069手 (308017)
 - 平均 : 55.62手

ソルバー1号と2号の結果

- 1号の結果 U 2号の結果
 - 4974問
 - 275366手(308017手)
- 残り
 - 26問
 - 32651手

ちなみにアルゴリズムは単純



雑魚は片付けた

残った問題(一部)

7	2	1	0	9	5
D	=	3	H	=	6
J	K	=	S	I	C
X	E	G	A	4	=
L	P	Q	N	M	T
V	W	R	=	Z	U

330

D	8	2	3	6	B
K	E	=	=	5	4
1	0	P	V	G	J
7	W	=	F	=	C
X	=	Y	H	U	I
R	S	M	T	Z	O

1052

1	2	3	=	=	=
7	=	H	B	C	S
D	=	A	G	N	I
J	=	P	F	Z	=
Q	W	M	9	Y	T
=	R	L	X	0	U

3318

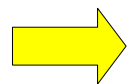
=	7	4	6	3	8
E	F	9	=	=	C
2	K	D	G	H	I
5	0	=	=	=	=
J	=	R	S	T	U
P	V	W	X	Y	Z

4790

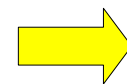
ソルバー3号

- 正攻法で解く

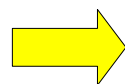
		=
↑ 2		



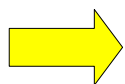
2		=
↑ 1		



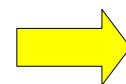
1	2	=
	←4	



1	2	=
4	←7	



1	2	=
4		
7		

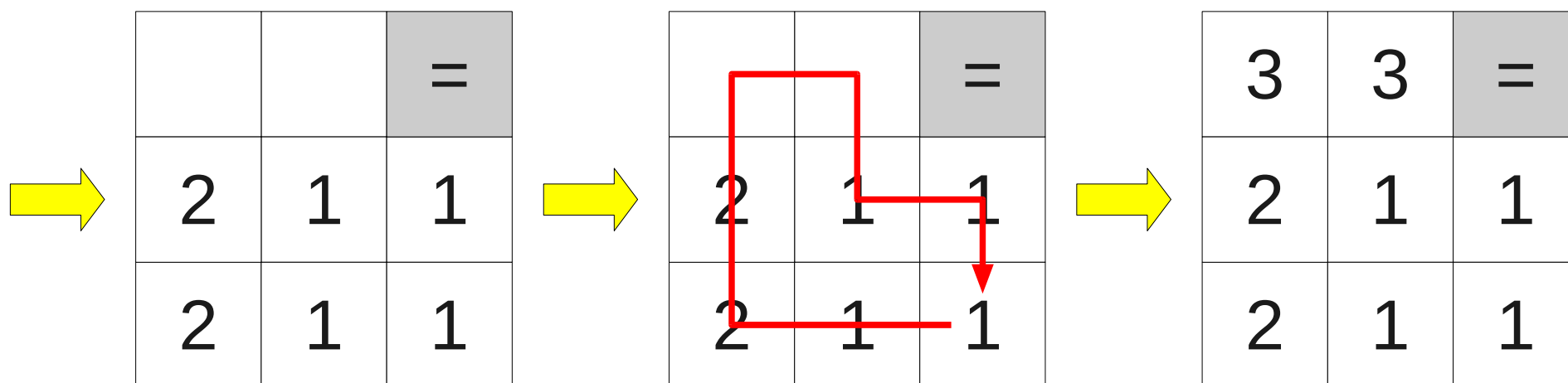
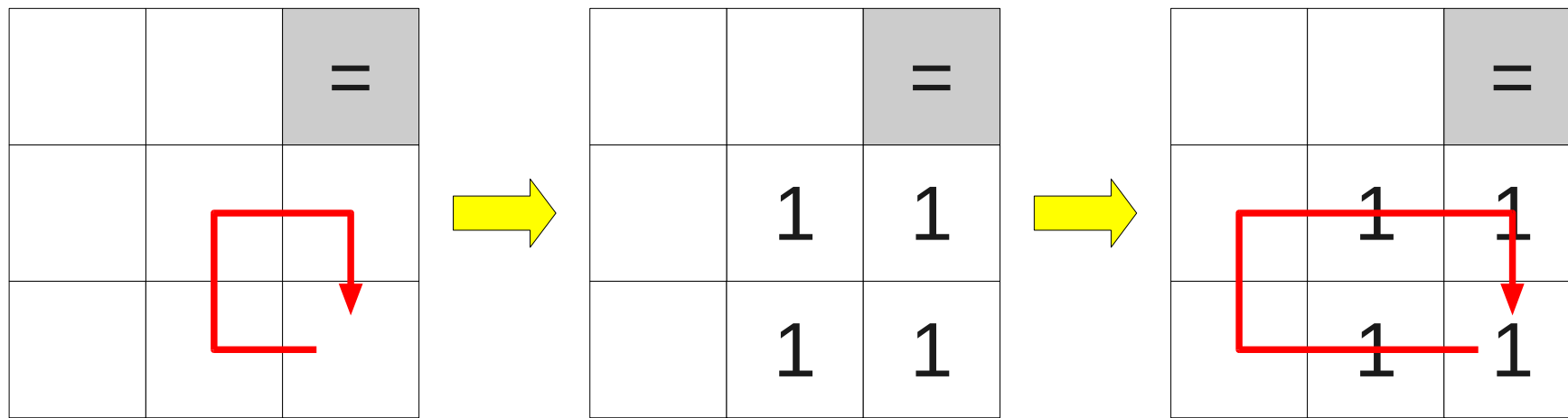


1	2	=
4	5	6
7	8	0

考え方

- 部分を揃えるときは壁から埋める
- 埋める順位を決める
- そのためには回せるスペースが必要

順位を決める



ややこしいものでも解ける

8	7	7	7	7	9
8	7	=	=	6	6
8	5	5	4	4	4
5	5	=	4	=	4
5	=	3	2	1	1
5	5	3	2	1	1

1052

ソルバー3号

- 残り26問全問正解
 - 8855手(残っていたのは32651手)
 - 平均：340.58手
 - チューニング前は1000手超えはザラだった、、
- 5000問を2分くらいで解ける
 - ただし100万手ほど使うので
3号単独だと余裕で足りない

というわけで

- ソルバー1号と2号の幅型波動砲で粗方を撃破
- ソルバー3号の正攻法で掃討
- 5000問解きました

プログラマ的な工夫

メモリ使用量の削減

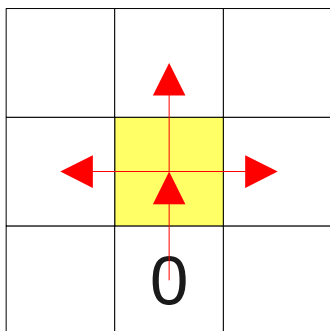
- 盤面をbyte型配列で表現

$$6 \times 6 = 36 \text{byte}$$

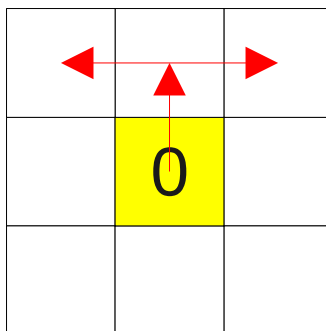
- int型だと144byte必要

無駄な履歴は記録しない

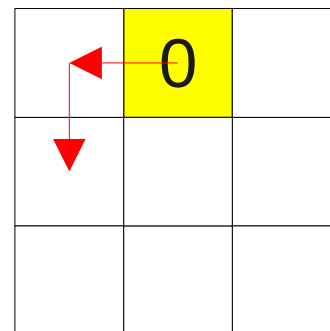
- 2方向しか進めない場所は除外する
- 初回に分岐点をチェックしておく



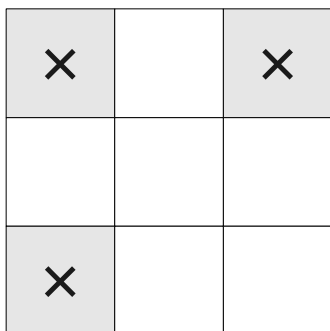
3手に分岐する



2手に分岐する



分岐しない



×のところは分岐しないから履歴不要
(右下は最後なので例外)

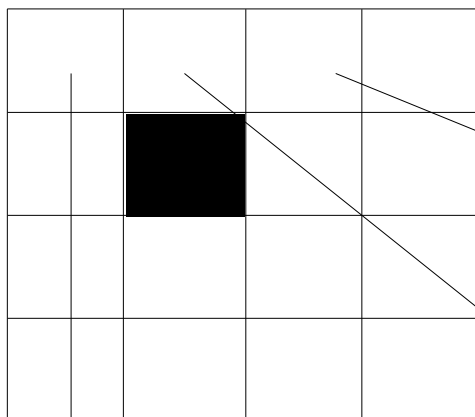
×		×	×		×
		=	=		
		×			
	×	=	×	=	×
×	=	×			
×	×				

1052

×のところは分岐しないから履歴不要
(右下は最後なので例外)

壁考慮マンハッタン距離をキャッシュ

- 6x6 → 6x6の距離を
予め計算しておく
- 多くても1296byte



0	1	2	3
1		3	4
2	3	4	5
3	4	5	6

1	0	1	2
2		2	3
3	4	3	4
4	5	4	5

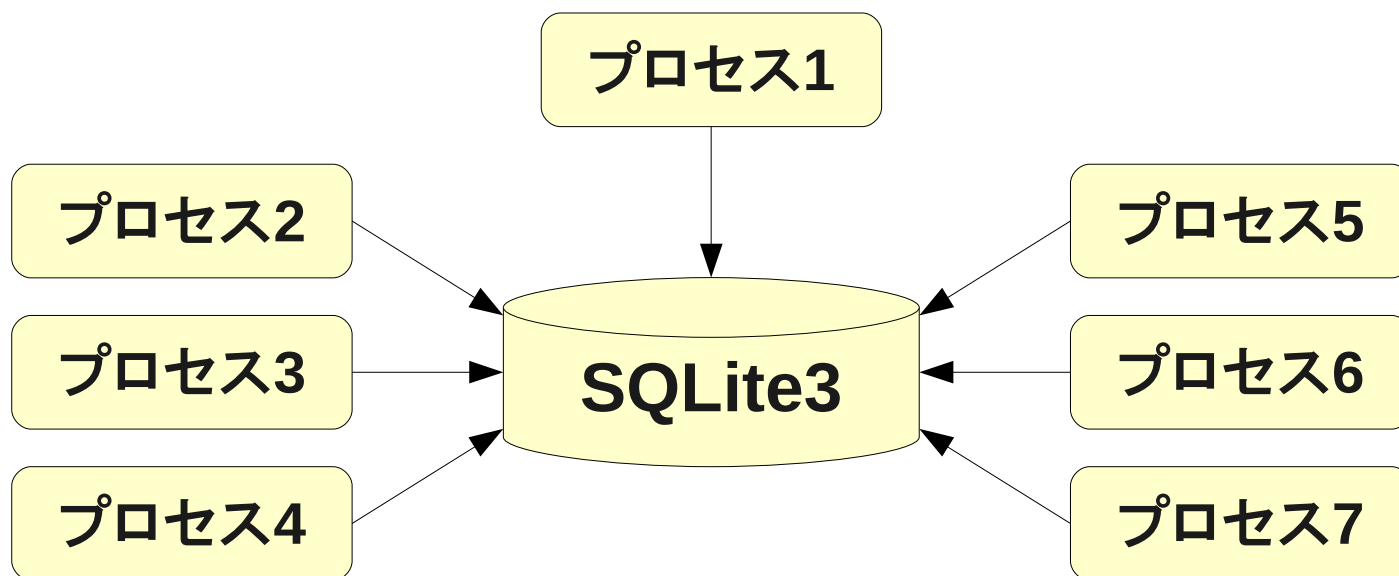
2	1	0	1
3		1	2
4	3	2	3
5	4	3	4

AWS上での実行を簡略化

- デプロイ用のantファイルを作成
- 起動用シェルスクリプトの準備

結果データをSQLiteに出力

- 基本的に処理は1プロセスシングルスレッド
- 結果は1つのSQLiteのDBファイル
- 1プロセスに715問ずつ割り振って実行



ご清聴ありがとうございました。