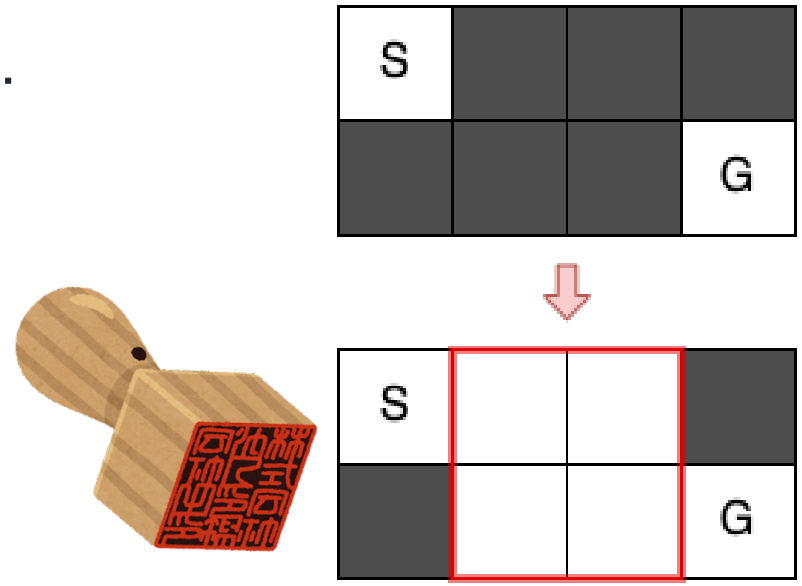
JOI 2022/2023 本選 問題 3
迷路 (Maze) 解説

解説担当 : tatyam

問題概要

- $R \times C$ マスのマス目があり，通れるマスと通れないマスがある.
- $N \times N$ マスの領域を選び，通れるようにする操作を何回でも行える.
- マス (S_r, S_c) とマス (G_r, G_c) を連結にしたい.
- 操作回数の最小値は？
- $R \times C \leq 6 \times 10^6$



小課題

1. (8 点) $N = 1$, $R \times C \leq 1.5 \times 10^6$.
2. (19 点) $R \times C \leq 10^3$.
3. (16 点) 答えは 10 以下である, $R \times C \leq 1.5 \times 10^6$.
4. (19 点) $R \times C \leq 6 \times 10^4$.
5. (5 点) $R \times C \leq 1.5 \times 10^5$.
6. (19 点) $R \times C \leq 1.5 \times 10^6$.
7. (8 点) $R \times C \leq 3 \times 10^6$.
8. (6 点) $R \times C \leq 6 \times 10^6$.

小課題 1 (8 点)

- $N = 1$
- 1×1 マスの領域を選び，通れるようにする操作を何回でも行える.

これは…？

小課題 1 (8 点)

C - 器物損壊！高橋君 解説

実行時間制限: 2 sec / メモリ制限: 64 MB

問題文

　良く見てみるとカードの有効期限が切れていたので、高橋君は諦めて魚屋に直接うなぎを買いに行くことにしました。
　彼の住む街は長方形の形をしており、格子状の区画に区切られています。区画は道または塀のどちらかであり、高橋君は道を東西南北に移動できますが斜めには移動できません。また、塀の区画は通ることができません。高橋君の家から魚屋までの道のりは非常に複雑なため、単純に歩くだけでは辿り着くことは困難です。
　しかし、高橋君は腕力には自信があるので道に上下左右で面している塀を 2 回までなら壊して道にすることができます。

　高橋君が魚屋に辿り着くことができるかどうか答えてください。

入力

入力は以下の形式で標準入力から与えられる。

$$
\begin{array}{l}
H \quad W \\
c_{(0,0)}c_{(0,1)} \quad \cdots \quad c_{(0,W-1)} \\
c_{(1,0)}c_{(1,1)} \quad \cdots \quad c_{(1,W-1)} \\
\vdots \\
\vdots \\
c_{(H-1,0)}c_{(H-1,1)} \quad \cdots \quad c_{(H-1,W-1)}
\end{array}
$$

- 入力は $H+1$ 行ある。

考察① 重みつきグラフへの変換

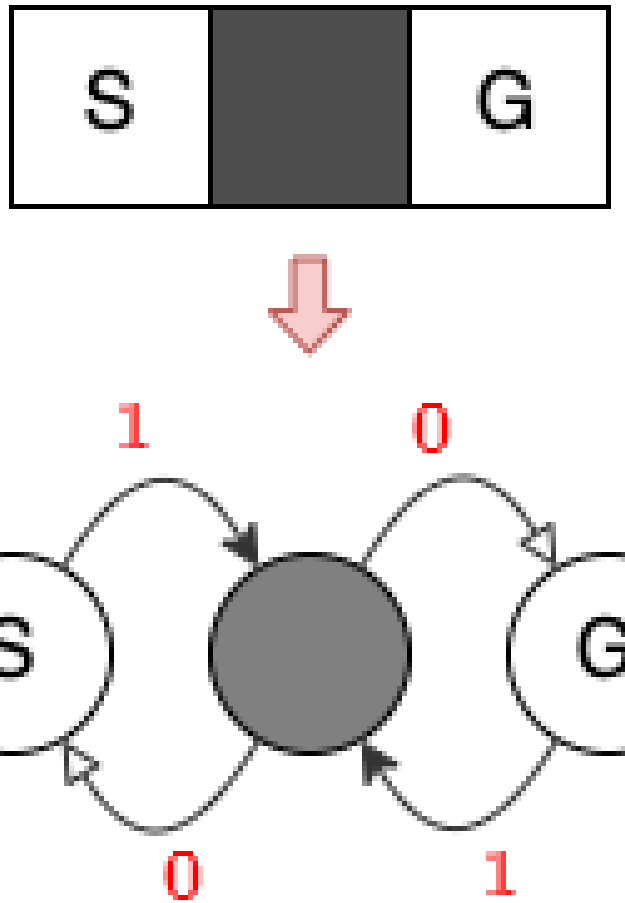
- 黒いマスを 1 個通るのに 1 回操作が必要
- **同じ黒マスを複数回通る必要はない**

考察① 重みつきグラフへの変換

- 黒いマスを 1 個通るのに 1 回操作が必要
- **同じ黒マスを複数回通る必要はない**

→ 黒マスを通る回数が最小となる S から G への経路を見つける問題

考察① 重みつきグラフへの変換



- 黒いマスに入る辺のコストを 1
- 白いマスに入る辺のコストを 0

とすると，頂点 S から頂点 G への最短経路問題になる

小課題 1 の解法

- 重み付きグラフの最短路問題

→ ダイクストラ法で $O(RC \log(RC))$ だ！

それでもいいが…？

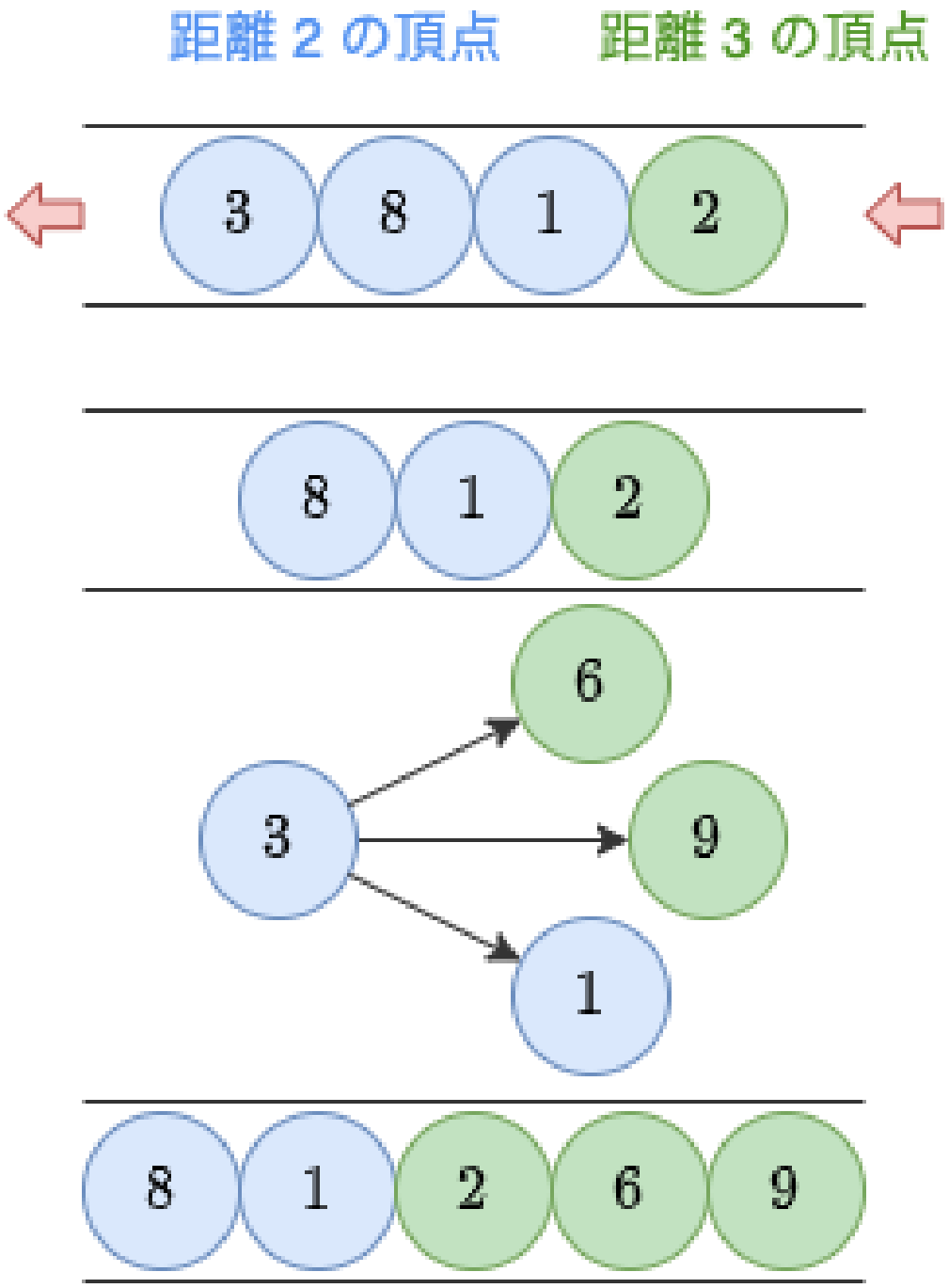
小課題 1 の解法

01 BFS

- 辺の重みが $0, 1$ のみである場合に使える BFS の亜種
- BFS では探索する頂点をキュー (queue) に入れていた
- 01-BFS では探索する頂点を両終端キュー (deque) に入れる

BFS の場合

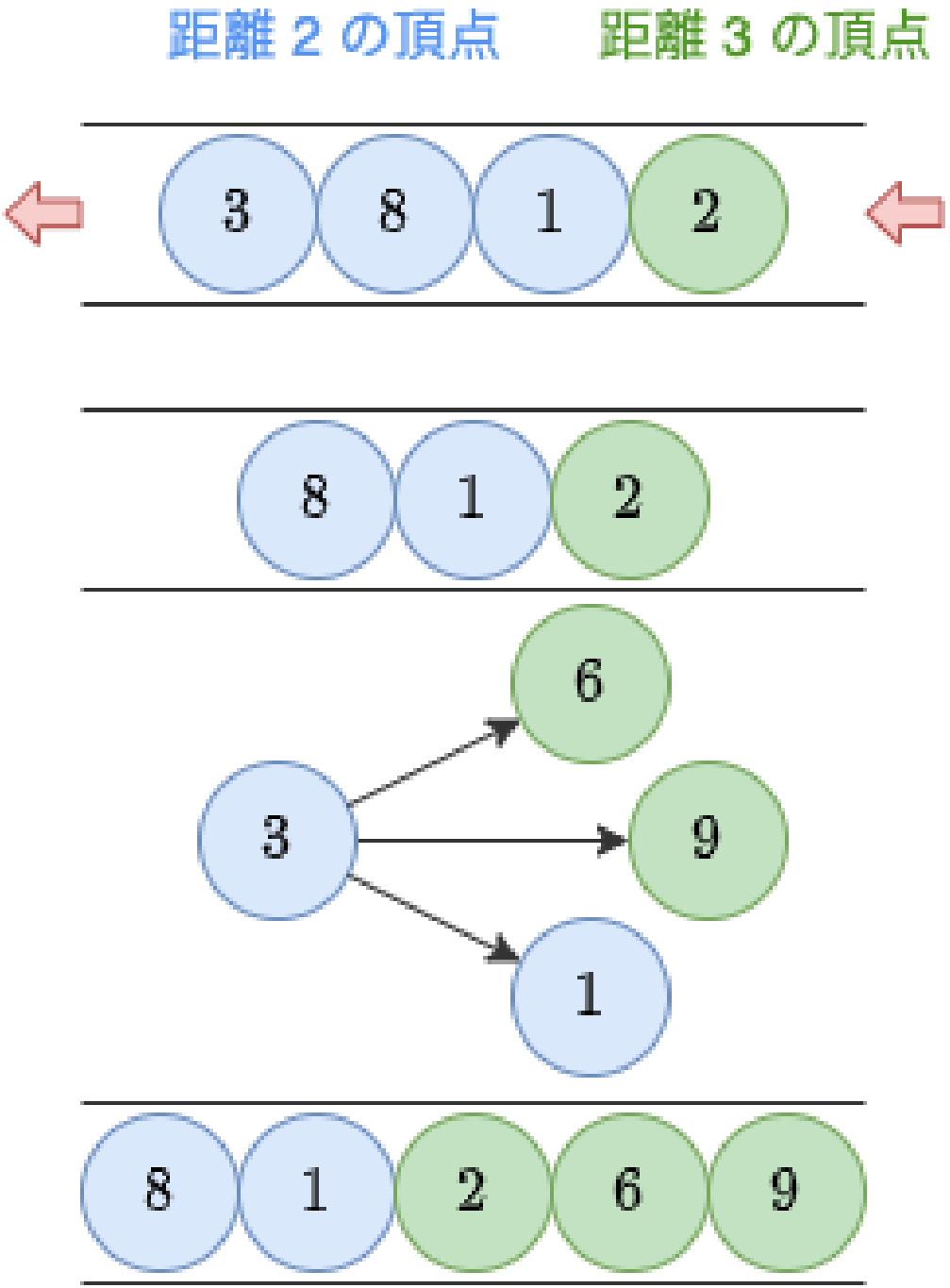
- 探索する頂点を **queue** に入れる
- queue の内部に**境界がある**．境界の前には始点から距離 x の頂点が，境界の後ろには始点から距離 $x+1$ の頂点が入っている $\cdots$ $(\bigstar)$



BFS の場合

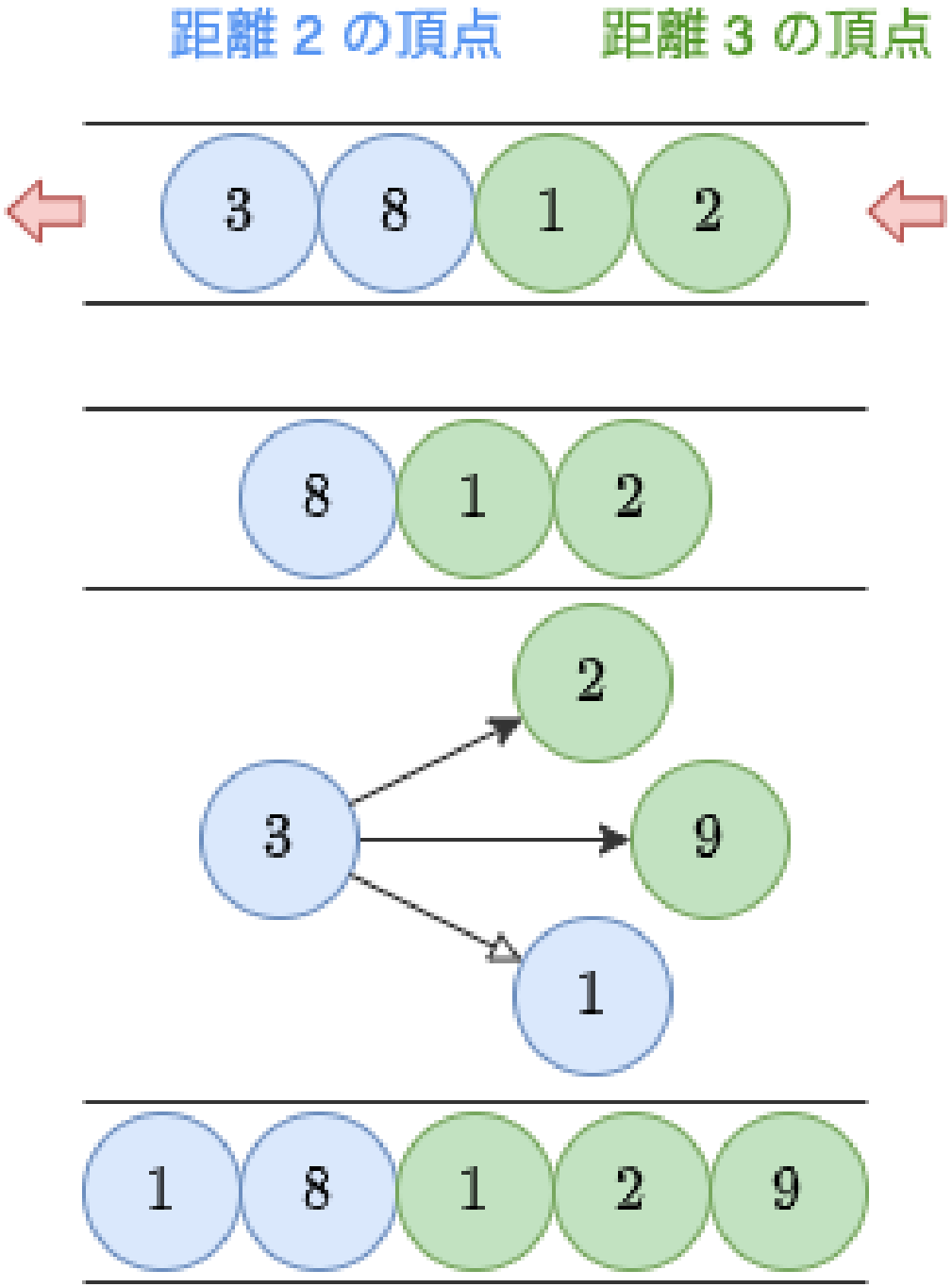
- 探索する頂点を **queue** に入れる
- queue の内部に**境界がある**・・・(★)
- 前から頂点を取り出して，隣接する頂点を距離 $x+1$ で更新する
- 更新された頂点を queue の後ろに入れる
- これらの操作を行なっても (★) は満たされたまま

→ 距離の小さい順に探索できる！



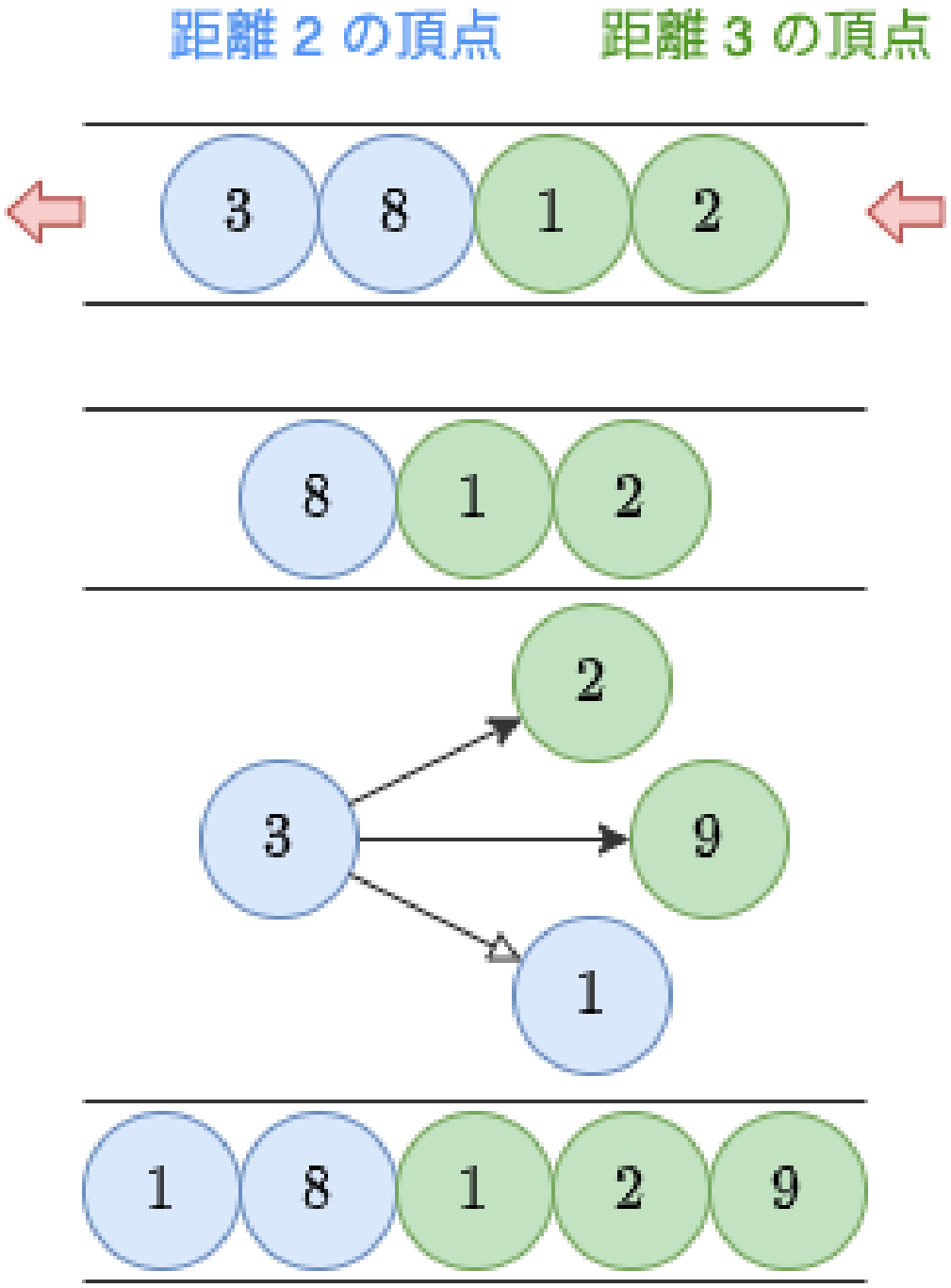
01-BFS の場合

- 探索する頂点を **deque** に入れる
- deque の内部に境界がある・・・(★)
- 前から頂点を取り出して，隣接する頂点を距離 x または距離 $x+1$ で更新する



01-BFS の場合

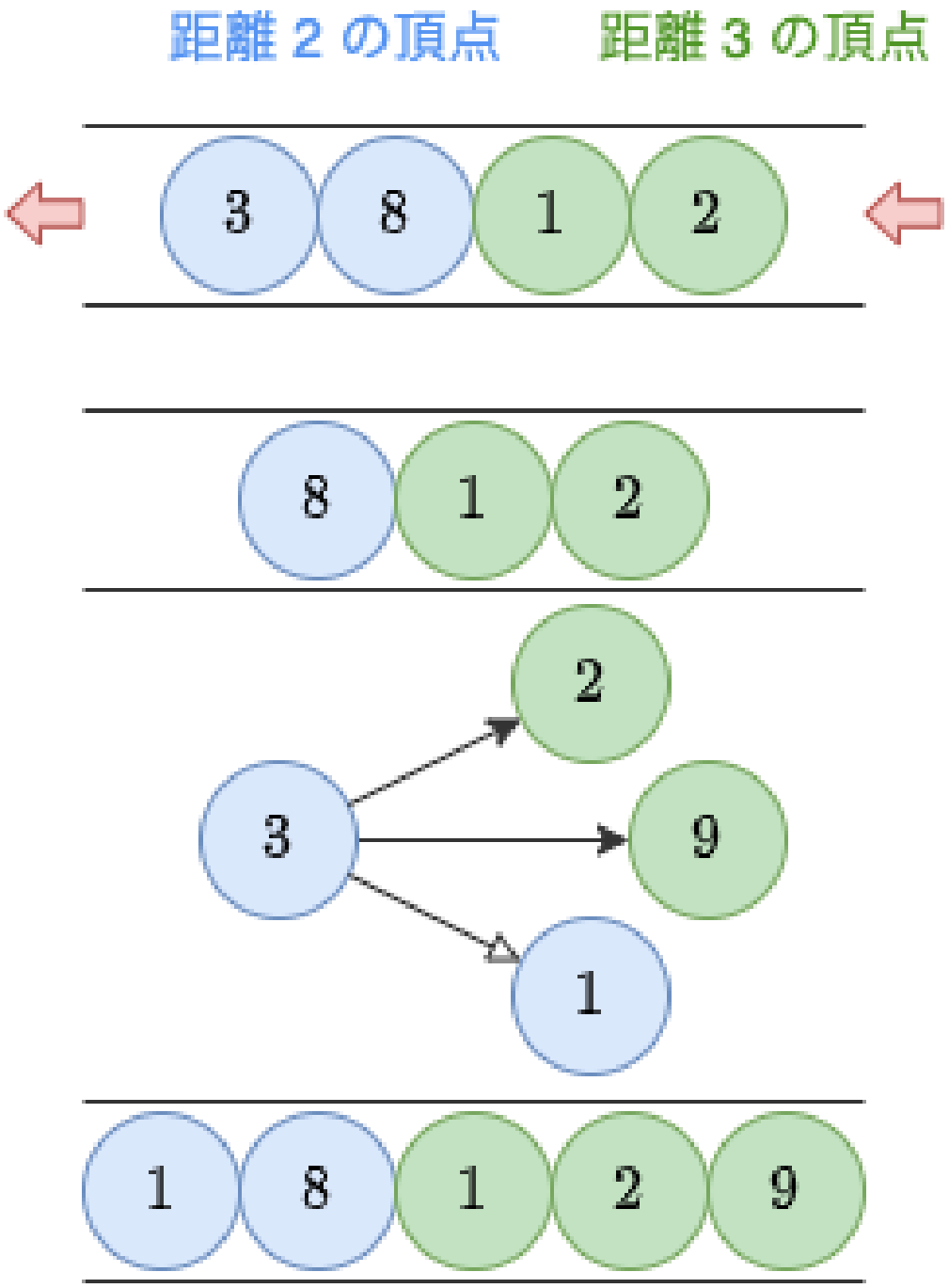
- 探索する頂点を **deque** に入れる
- deque の内部に境界がある・・・(★)
- 前から頂点を取り出して，隣接する頂点を距離 x または距離 $x+1$ で更新する
- 距離 x で更新された頂点を deque の前に入れる
- 距離 $x+1$ で更新された頂点を deque の後ろに入れる



01-BFS の場合

- 探索する頂点を deque に入れる
- deque の内部に境界がある・・・(★)
- 前から頂点を取り出して，隣接する頂点を距離 x または距離 $x+1$ で更新する
- 距離 x で更新された頂点を deque の前に入れる
- 距離 $x+1$ で更新された頂点を deque の後ろに入れる
  - これらの操作を行なっても (★) は満たされたまま

→ 距離の小さい順に探索できる！



小課題 1 の解法 まとめ

- マス目を辺の重みが 0 または 1 である重み付き有向グラフに変換する
- RC 頂点，$4RC - 2(R + C)$ 辺のグラフができる
- 01-BFS で S から G への最短経路を $O(RC)$ 時間で求める

豆知識 : vector のメモリ再確保

`vector<vector<pair<int, int>>>` で RC 頂点の重み付きグラフを作って持つとかなり遅い

- `vector` は **連続したメモリ領域** を使用することが保証されている．確保したメモリ領域が足りなくなった場合は，長さを 2 倍程度にして **メモリを再確保** する．このメモリ再確保はそれなりの時間がかかる．
- `vector` を RC 個作って `push_back` していくと，メモリ再確保が RC 回以上起こり，かなり遅くなる．

豆知識 : vector のメモリ再確保

`vector<vector<pair<int, int>>>` で RC 頂点の重み付きグラフを作って持つとかなり遅い

- `vector` は **連続したメモリ領域** を使用することが保証されている．確保したメモリ領域が足りなくなった場合は，長さを 2 倍程度にして **メモリを再確保** する．このメモリ再確保はそれなりの時間がかかる．
- `vector` を RC 個作って `push_back` していくと，メモリ再確保が RC 回以上起こり，かなり遅くなる．

→ グラフを作らずにその都度 4 方向を調べるようにすると，速い

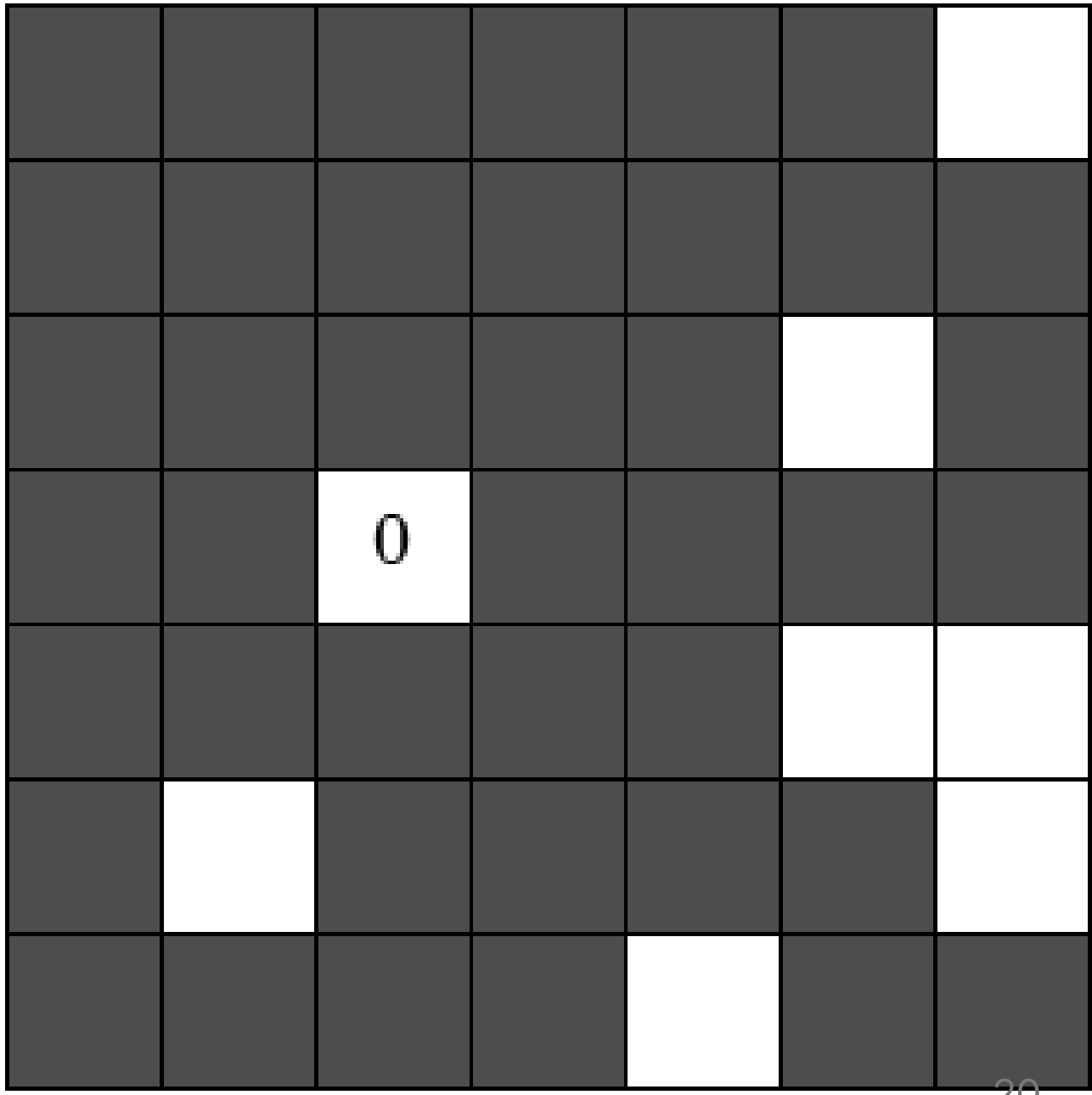
より一般的な方法 : 全部まとめて 1 つの vector に入れる (**CSR format**)

小課題 2 (19 点)

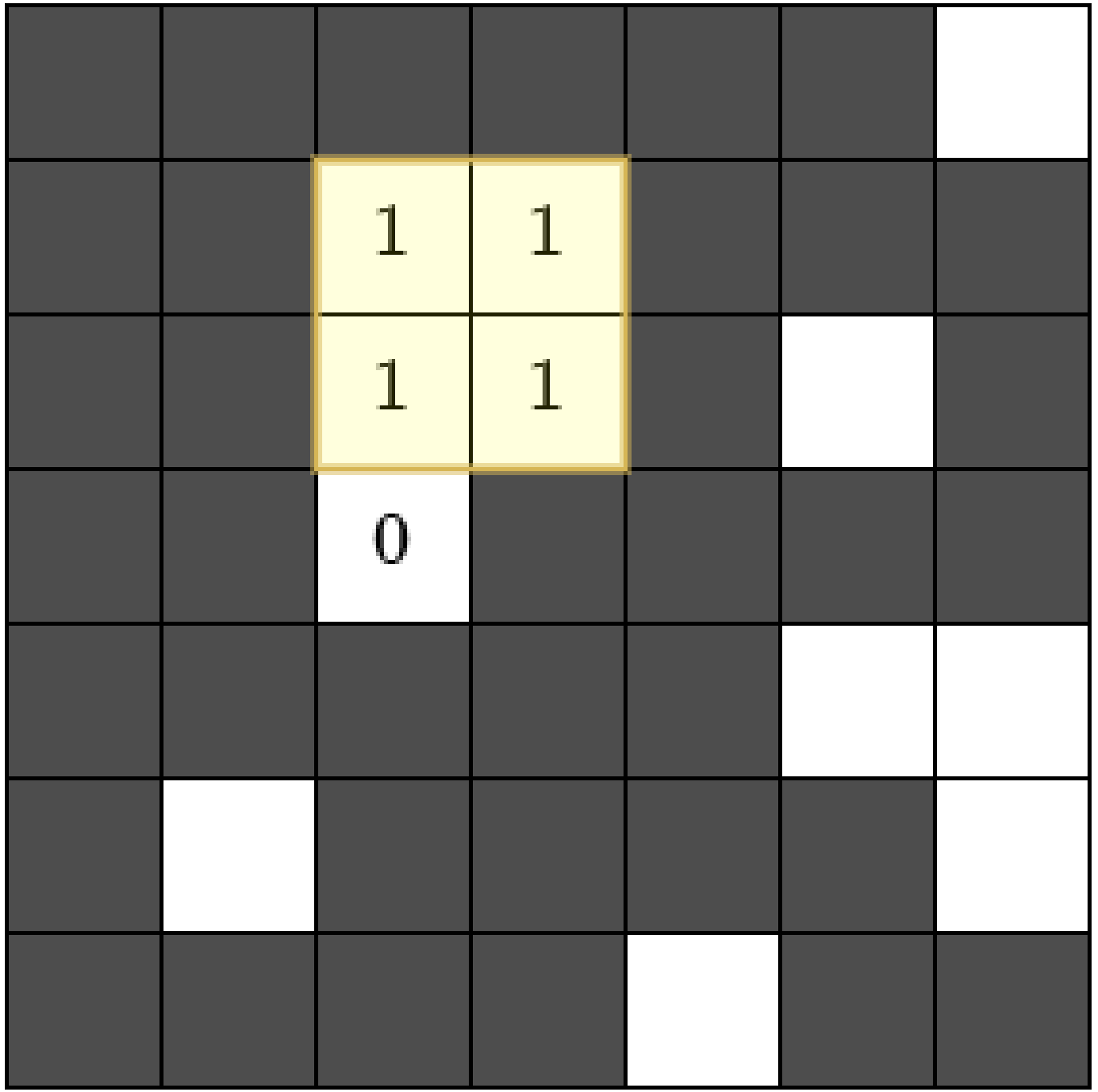
- $R \times C \leqq 10^3$
- $O((RC)^2)$ などの計算量で解けば良い

考察② 1 手で行ける範囲

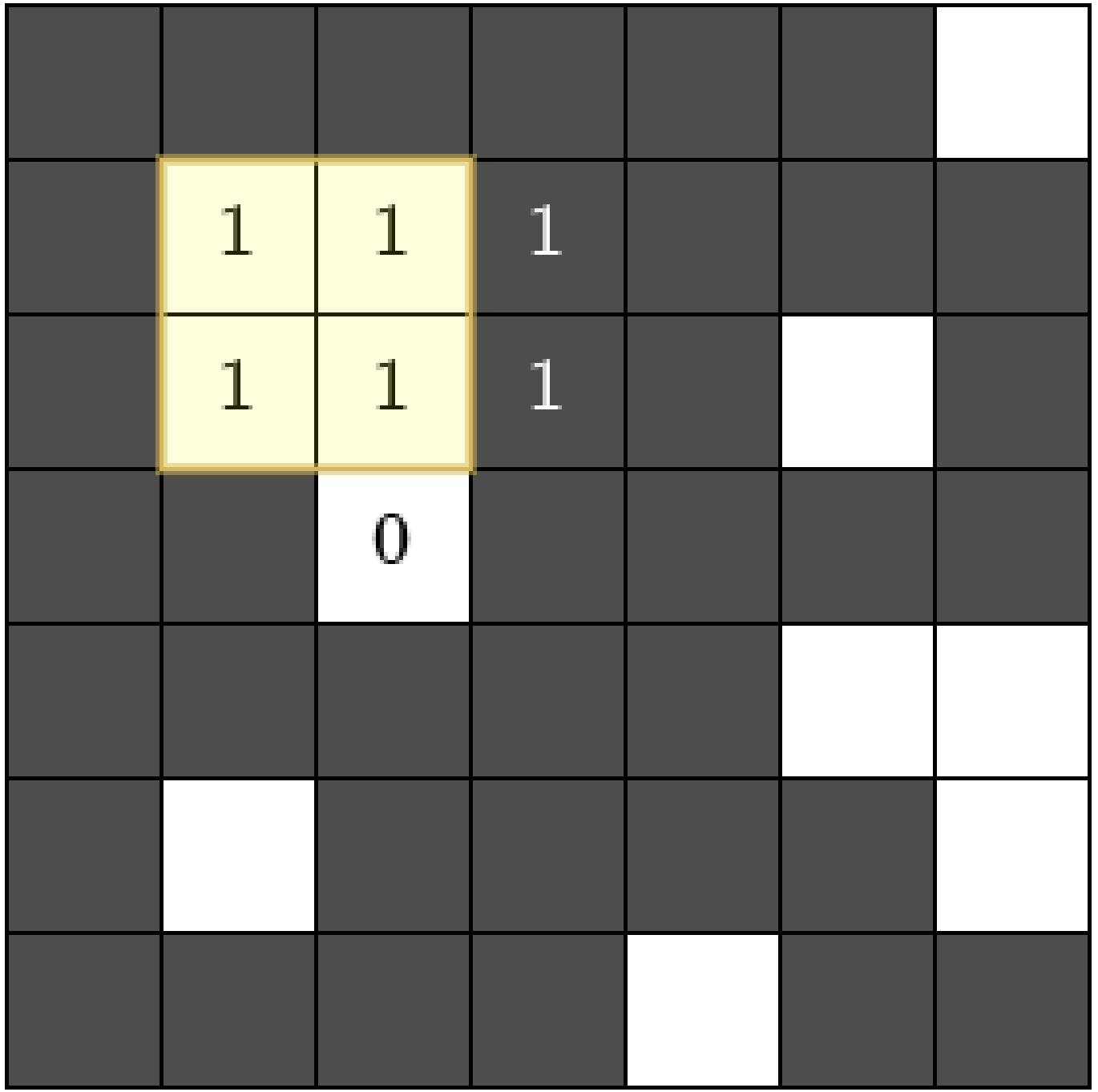
$N = 2$ とする.
4 方向を黒マスで塞がれているとき，1 手でどこまで行けるか？



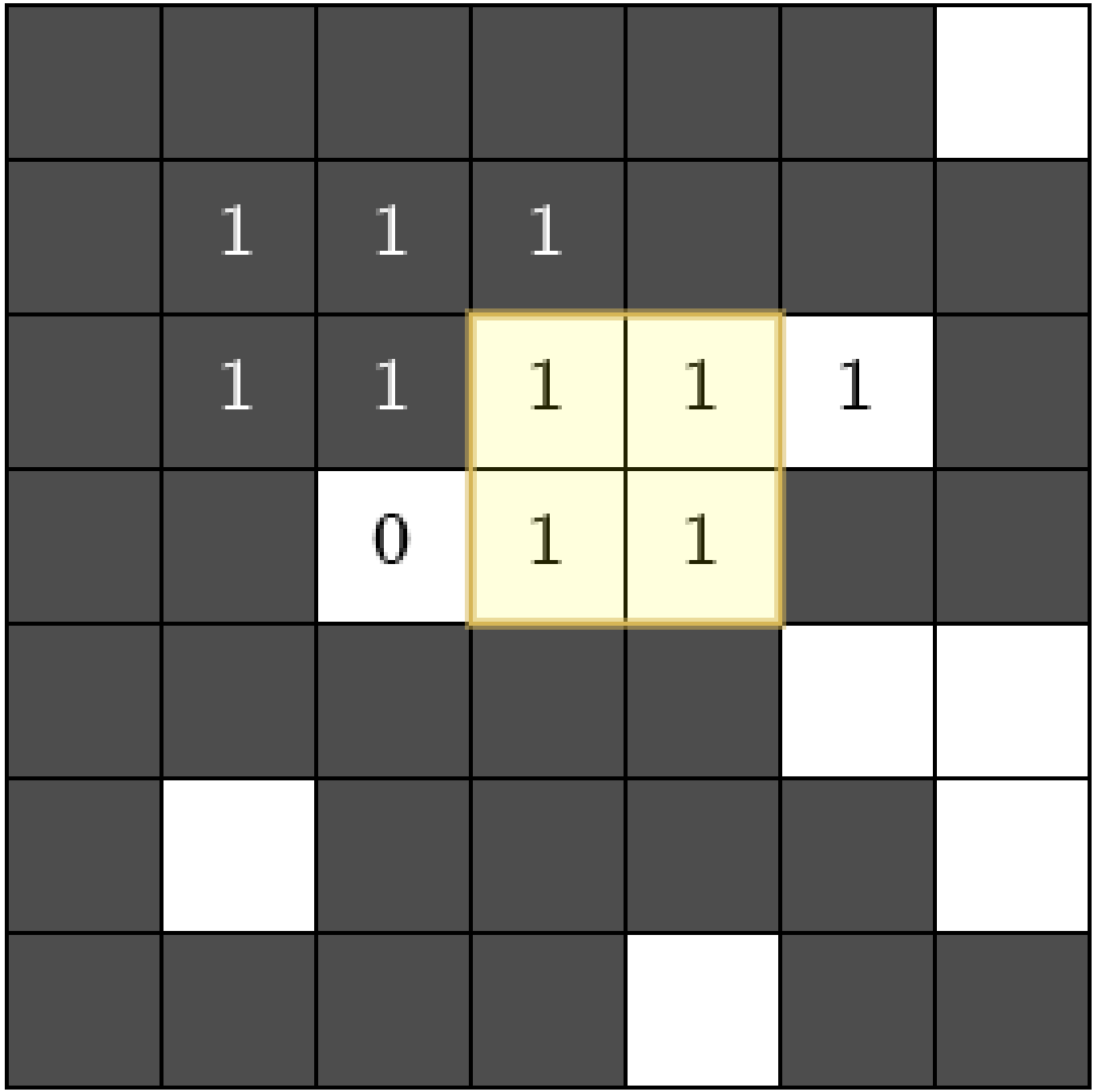
小課題 2 (19 点)



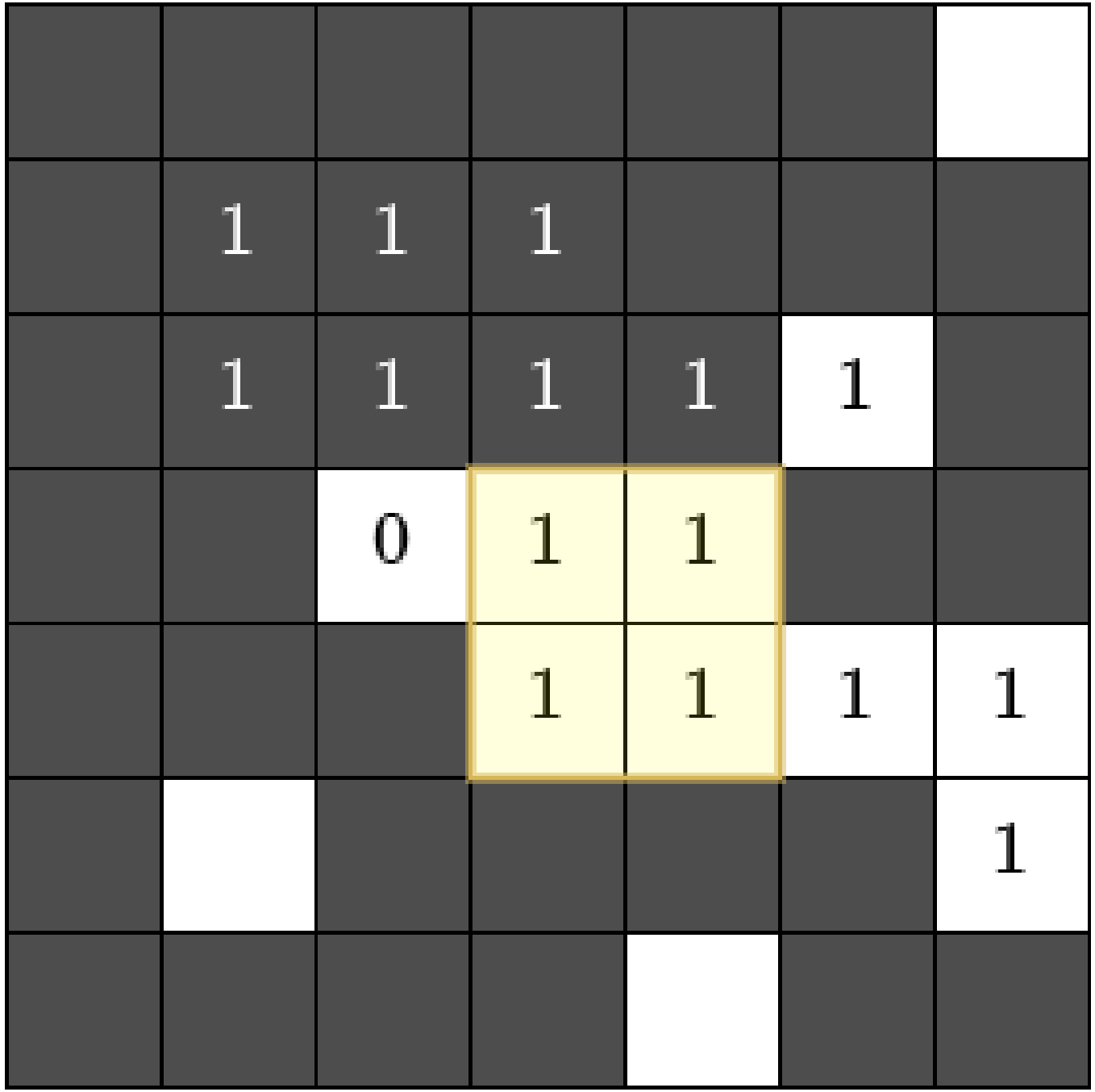
小課題 2 (19 点)



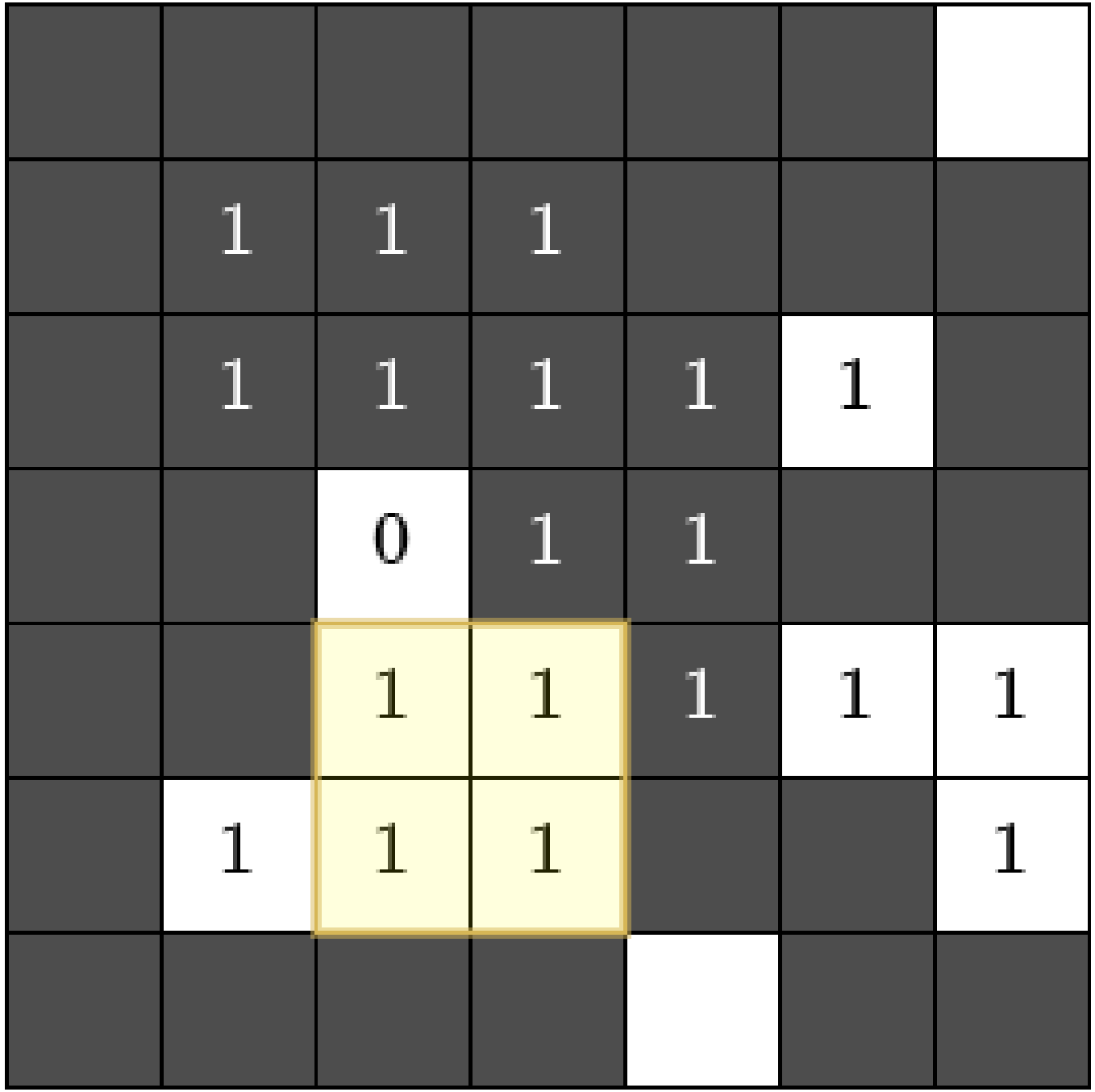
小課題 2 (19 点)



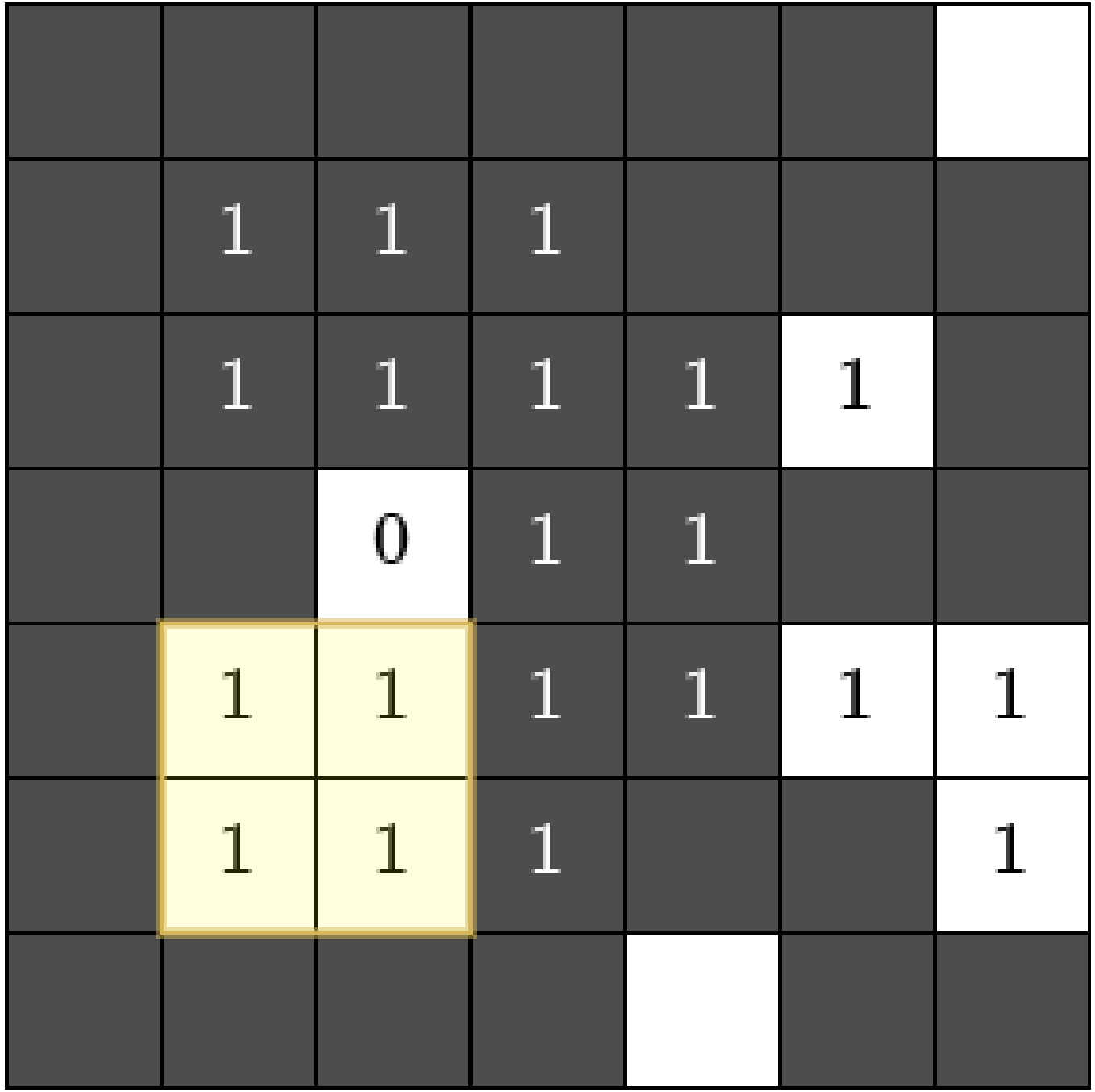
小課題 2 (19 点)



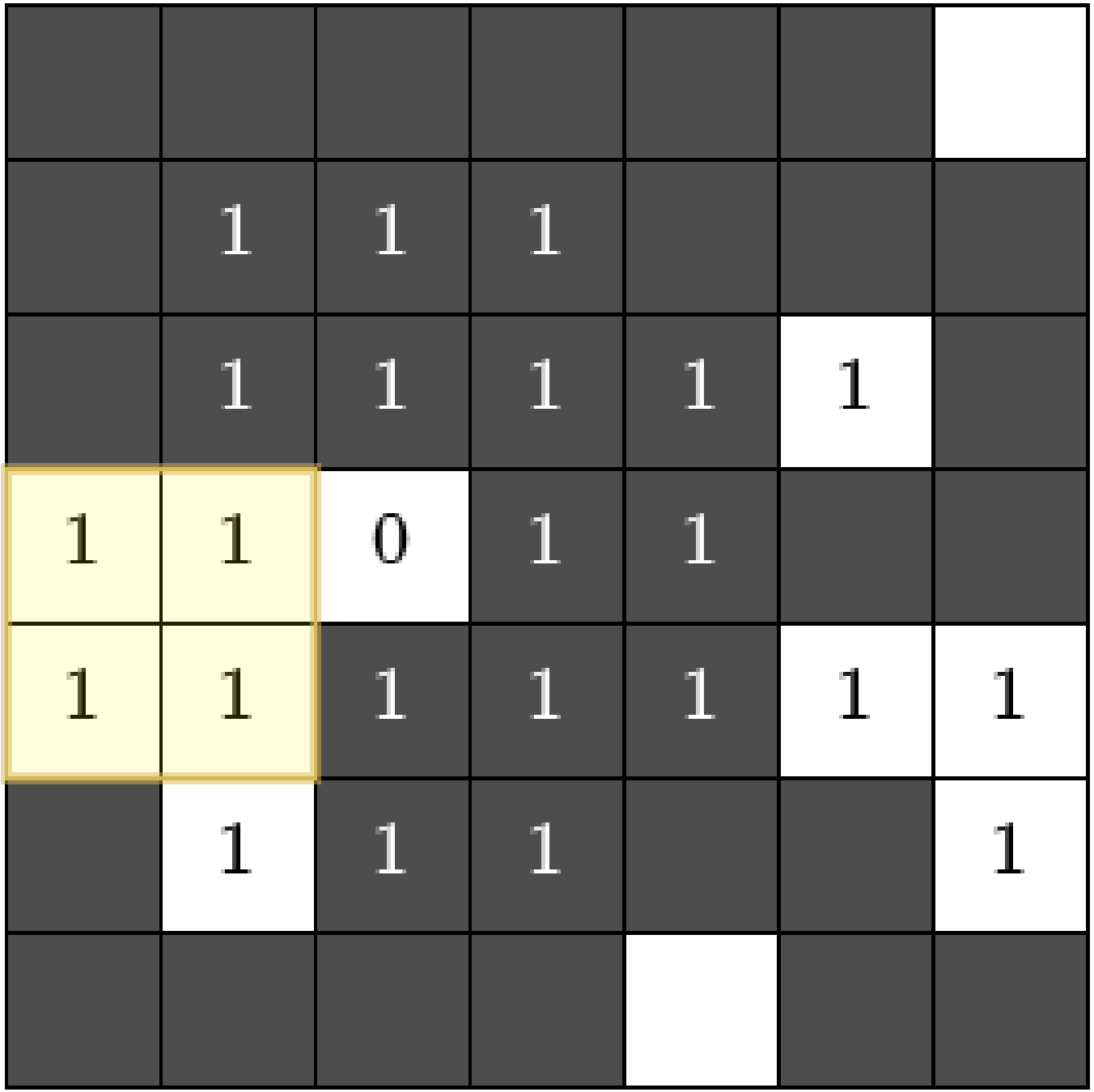
小課題 2 (19 点)



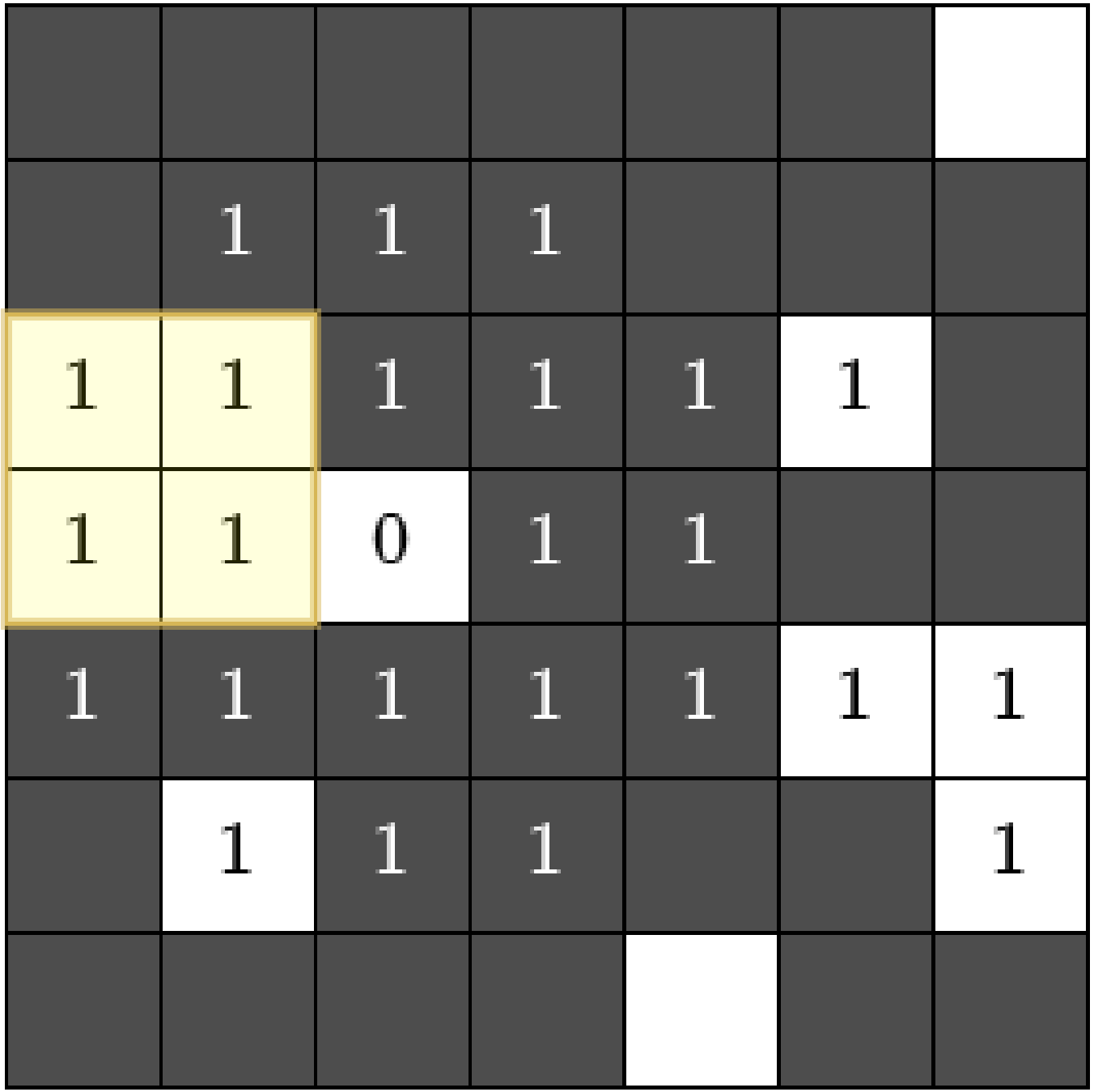
小課題 2 (19 点)



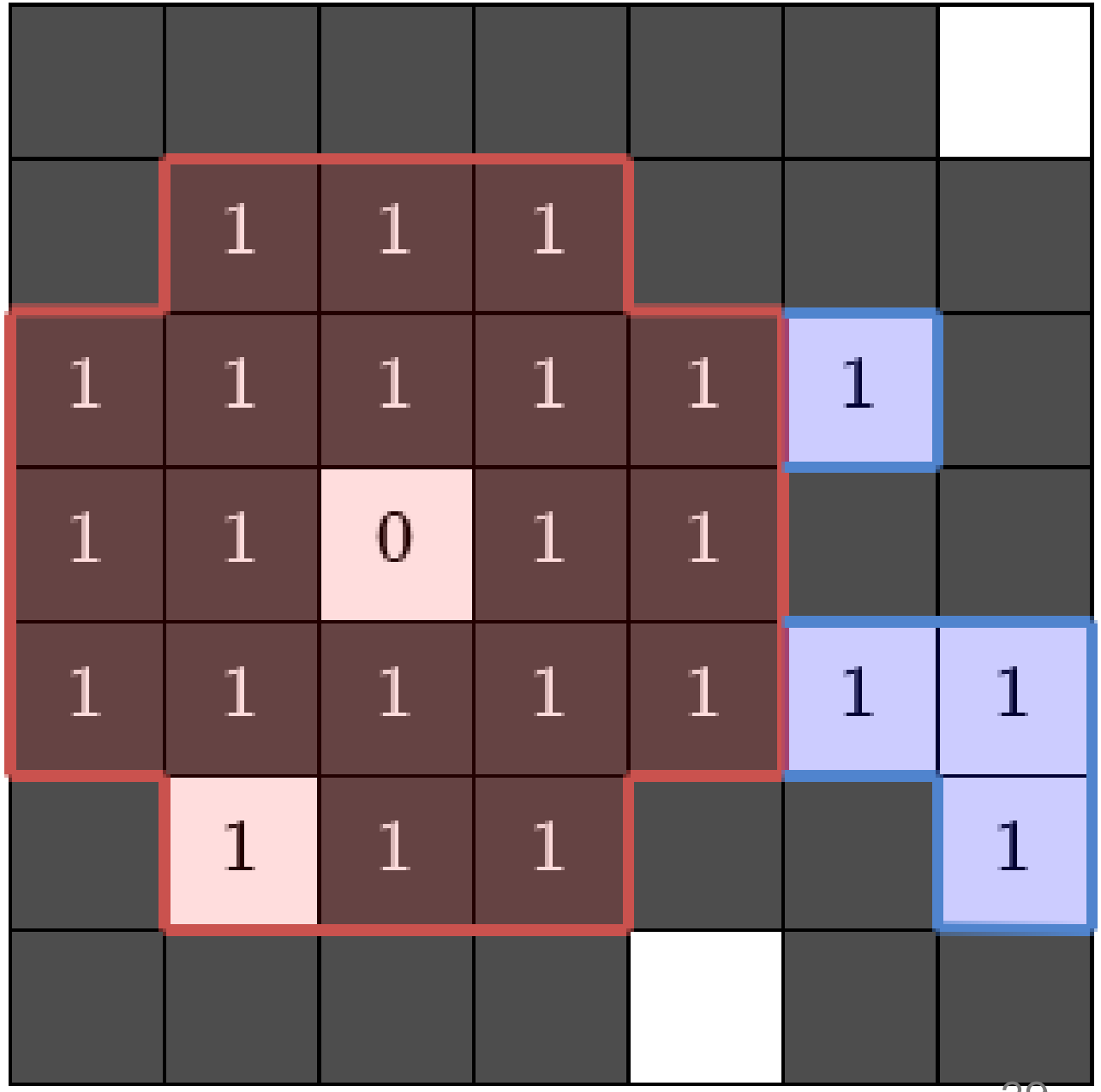
小課題 2 (19 点)



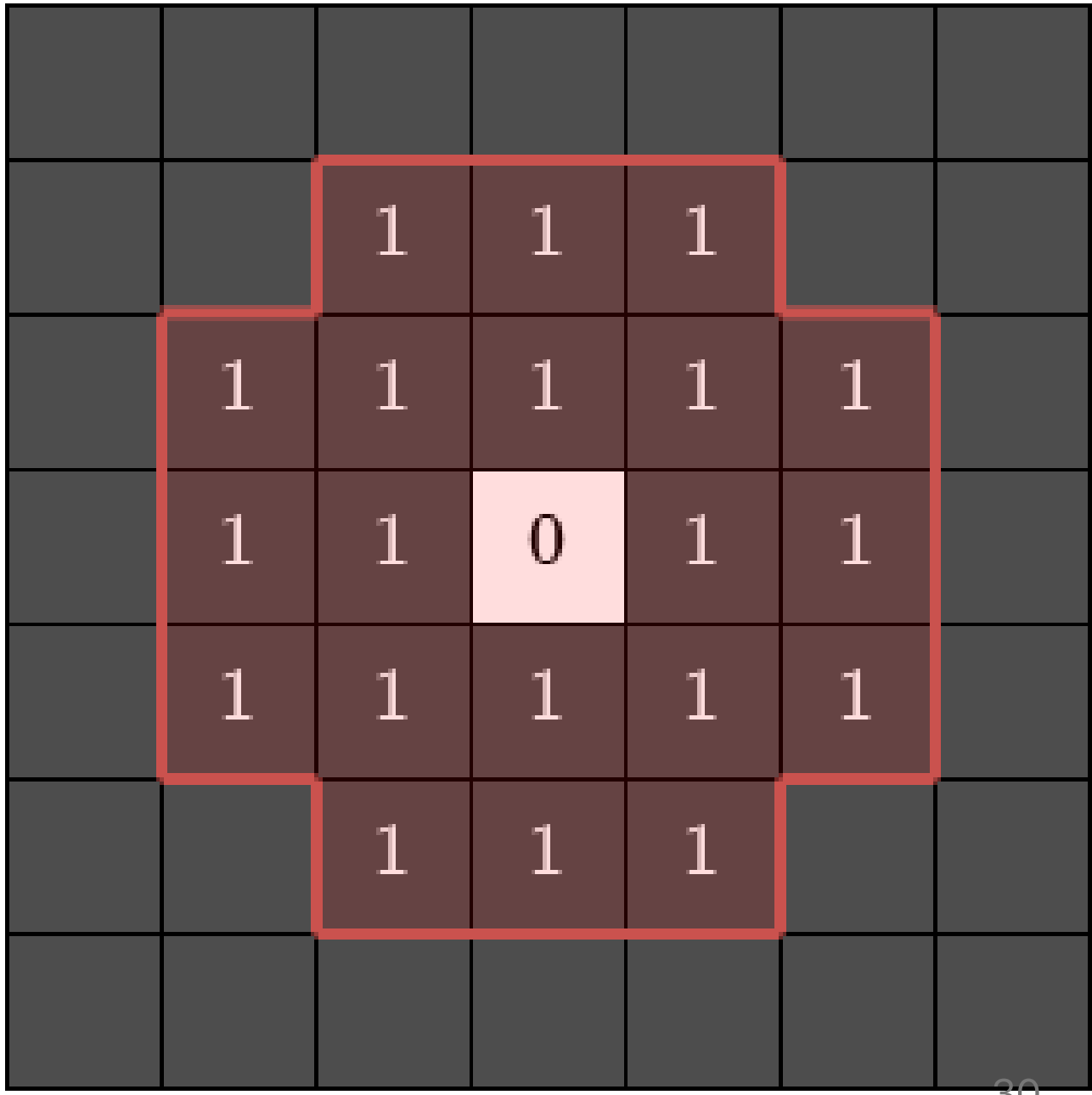
小課題 2 (19 点)



(1 手で行ける範囲)
＝ (ハンコで塗れる範囲) ＋
(そこから白いマスで移動できる範囲)

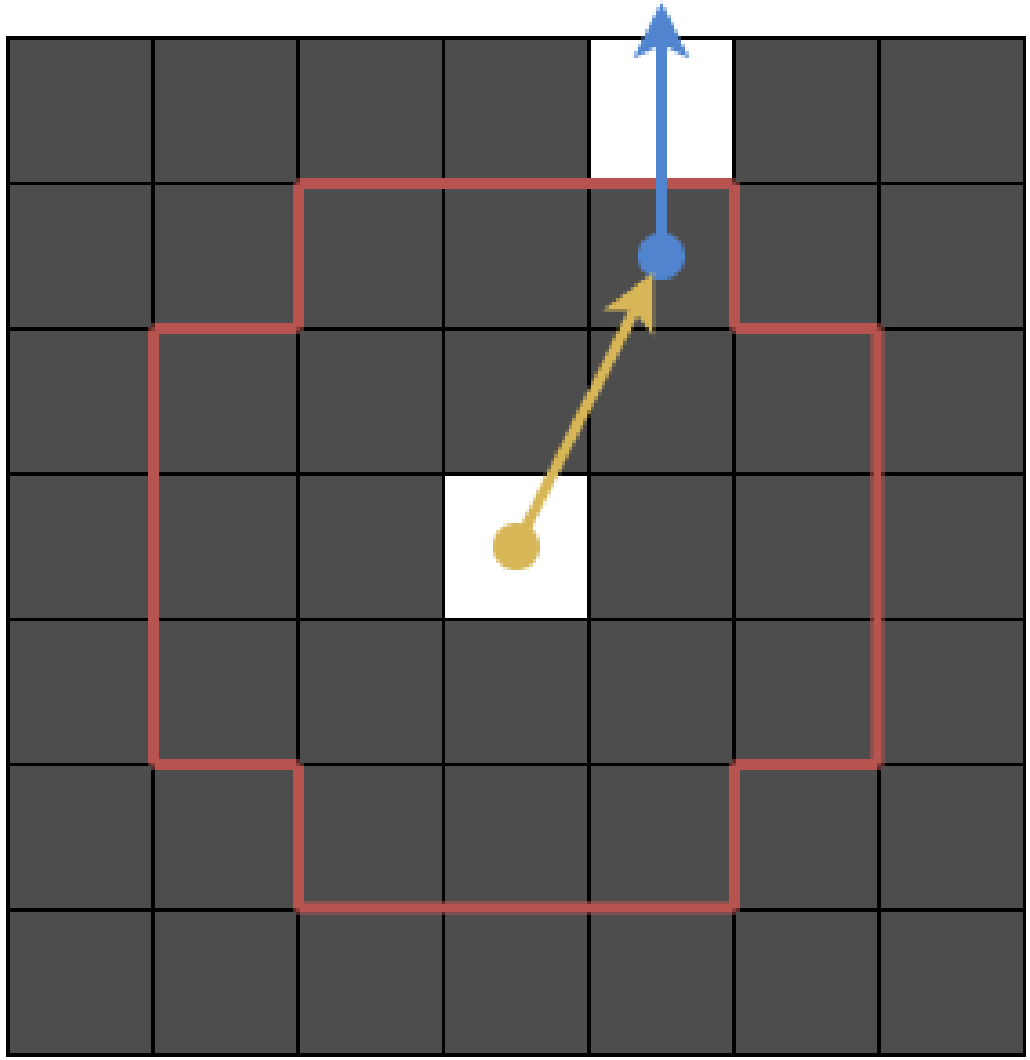


ハンコで塗れる範囲 は
$(2N+1) \times (2N+1)$ マスの正方形の領域から四隅を取り除いた形



考察③ ワープへの言い換え

「$N \times N$ のハンコを押して移動する」操作は，「ハンコで塗れる範囲 のマスに色を無視して**ワープする**」操作に言い換えることができる.

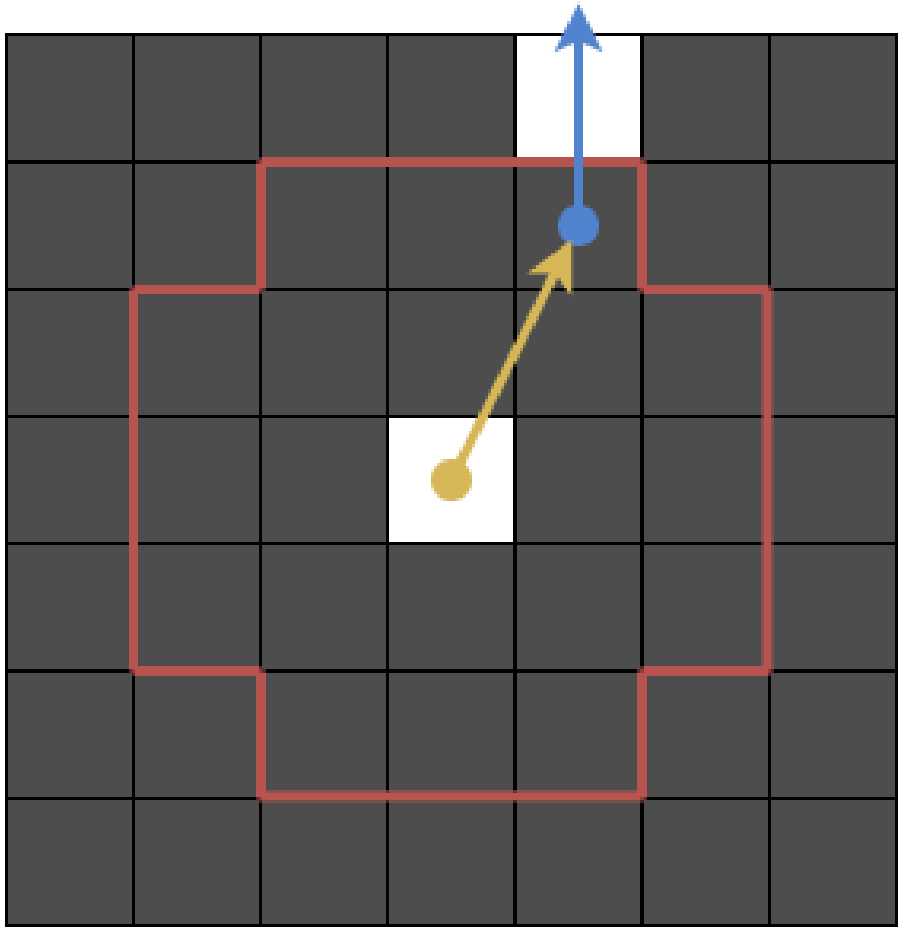


考察③ ワープへの言い換え

- ワープする操作は白マスが残らないため,
  ハンコを押す操作より弱い操作

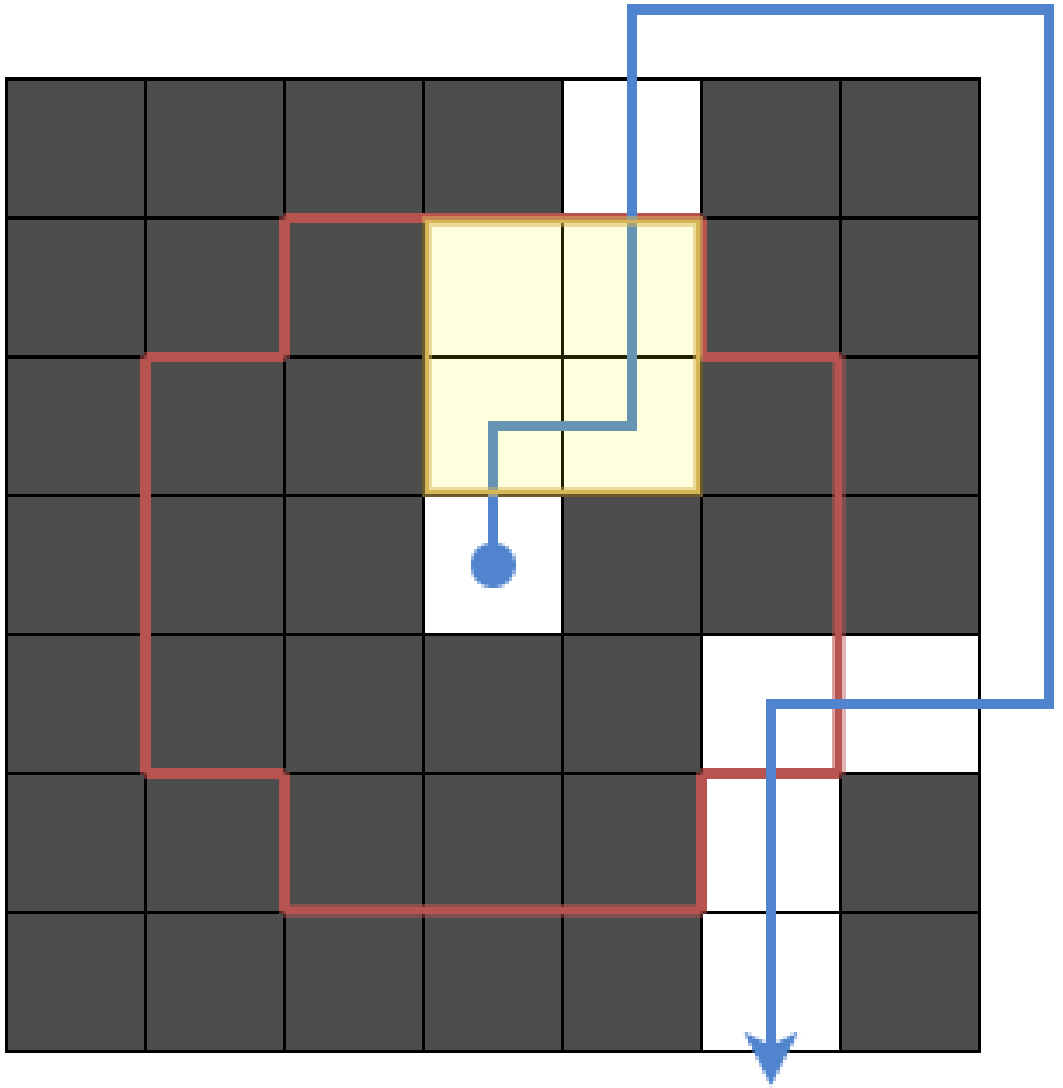
→ (ハンコの場合の最小の操作回数) $\leq$ (ワープの場合の最小の操作回数) は明らか.

(ハンコの場合の最小の操作回数) $\geq$ (ワープの場合の最小の操作回数) を示す.



補題

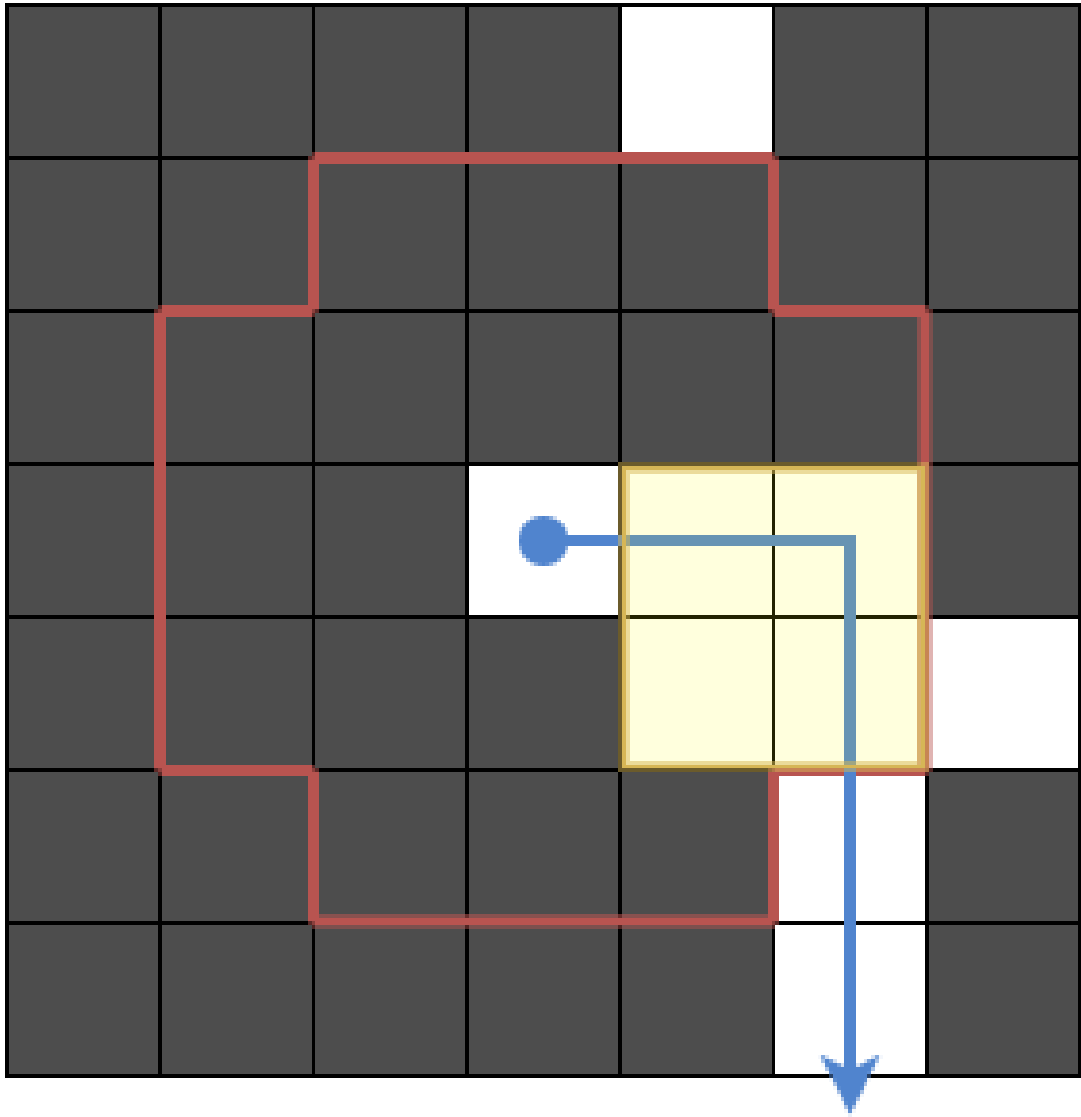
ハンコを押した後 ハンコで塗れる範囲 を出たら，ハンコで塗れる範囲 には戻ってこないとして良い (最小の操作回数は変わらない)



補題

ハンコを押した後 ハンコで塗れる範囲 を出たら，ハンコで塗れる範囲 には戻ってこないとして良い (最小の操作回数は変わらない)

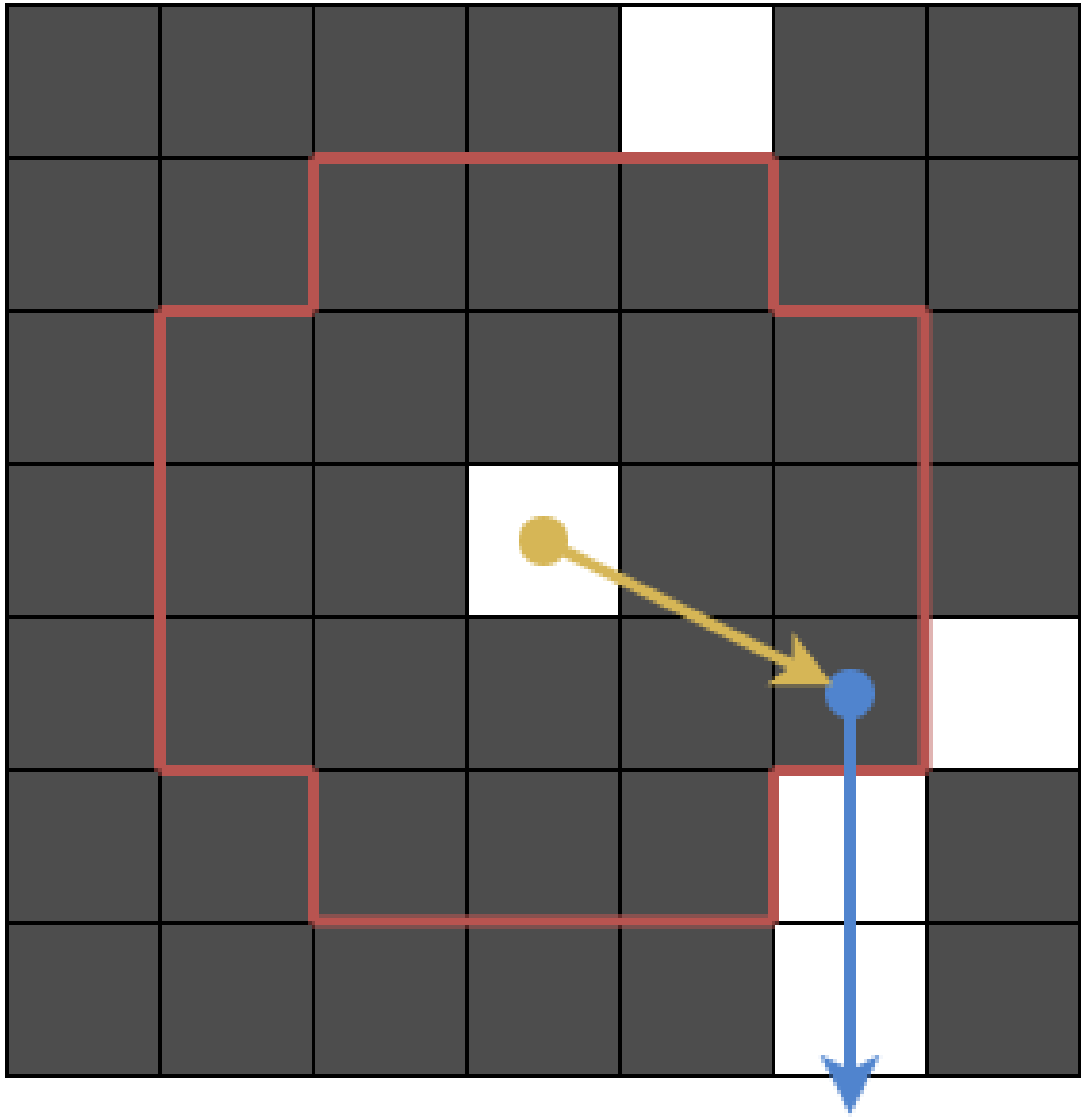
- 最初から 2 回目の方へ向かえば良い



考察③ ワープへの言い換え

ハンコで塗れる範囲 を 1 回しか出ないのであれば，ハンコを**ワープで代用**できる

→ ワープなら 01-BFS で扱える！



小課題 2 の解法 まとめ

- 小課題 1 と同様の 01-BFS
- 各マスから ハンコで塗れる範囲 へコスト 1 の辺を張る
- 各マスから隣接する白マスへコスト 0 の辺を張る
- 辺の数は $O(RCN^2)$ 本なので，01-BFS で $O(RCN^2)$

豆知識 : push 関数を作ろう

グリッド BFS を書くとき…

```
while(q.size()) {
    auto [r, c] = q.front();
    q.pop();
    if(r > 0 && chmin(cost[r - 1][c], cost[r][c] + 1))
        q.emplace(r - 1, c);
    if(r + 1 < R && chmin(cost[r + 1][c], cost[r][c] + 1))
        q.emplace(r + 1, c);
    if(c > 0 && chmin(cost[r][c - 1], cost[r][c] + 1))
        q.emplace(r, c - 1);
    if(c + 1 < C && chmin(cost[r][c + 1], cost[r][c] + 1))
        q.emplace(r, c + 1);
}
```

同じことを何度も書きすぎ！！

```
auto push = [&](int r, int c, int x) {
    if(cost[r][c] <= x) return;
    cost[r][c] = x;
    q.emplace(r, c);
};
while(q.size()) {
    auto [r, c] = q.front();
    q.pop();
    const int x = cost[r][c] + 1;
    if(r > 0) push(r - 1, c, x);
    if(r + 1 < R) push(r + 1, c, x);
    if(c > 0) push(r, c - 1, x);
    if(c + 1 < C) push(r, c + 1, x);
}
```

豆知識 : push 関数を作ろう

関数を使ってまとめよう

```
auto push = [&](int r, int c, int x) {
    if(r < 0 || r >= R || c < 0 || c >= C) return;
    if(cost[r][c] <= x) return;
    cost[r][c] = x;
    q.emplace(r, c);
};
while(q.size()) {
    auto [r, c] = q.front();
    q.pop();
    const int x = cost[r][c] + 1;
    push(r - 1, c, x);
    push(r + 1, c, x);
    push(r, c - 1, x);
    push(r, c + 1, x);
}
```

push 関数の中で境界判定をやっても良いですね

小課題 3 (16 点)

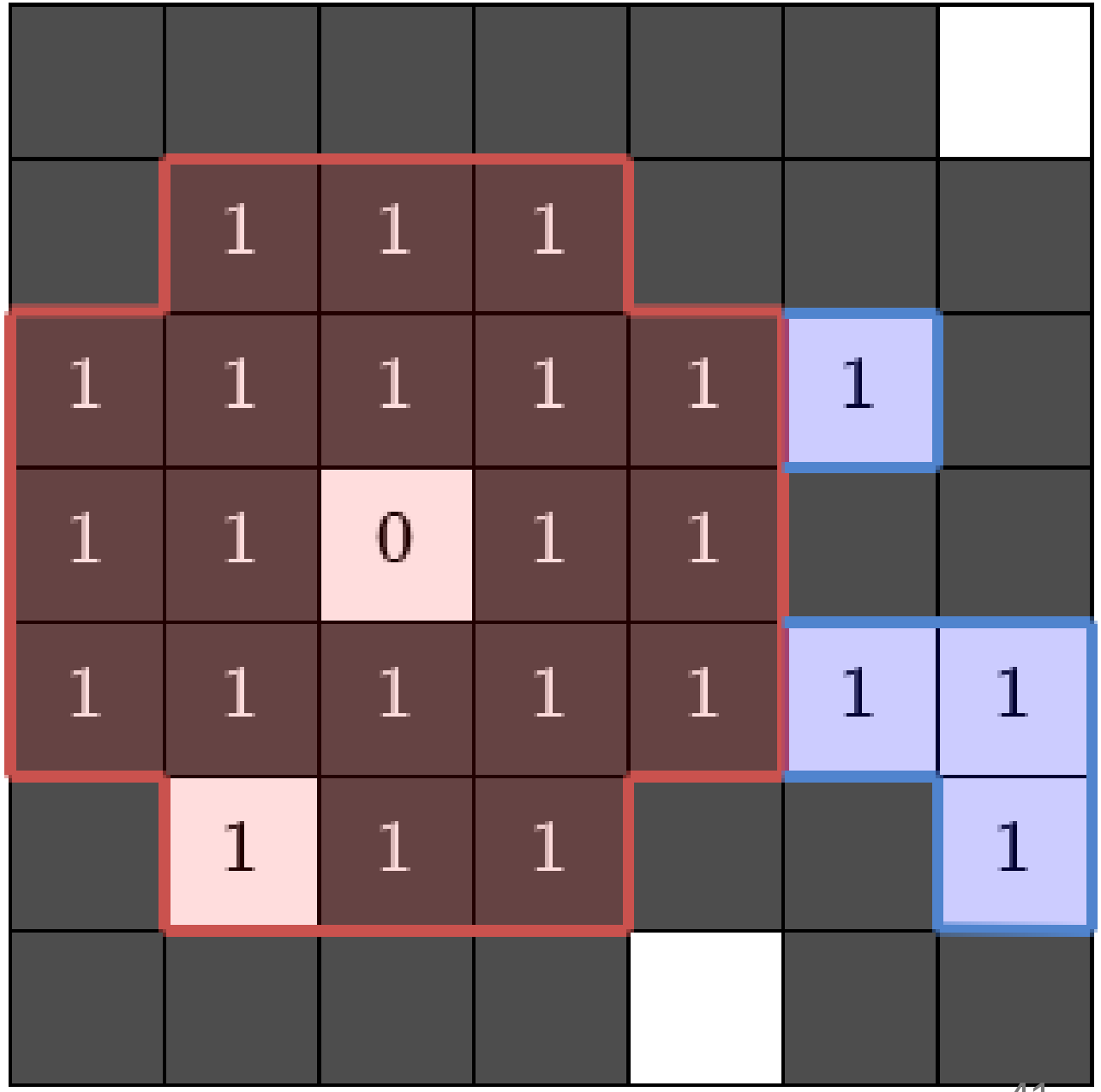
- 答えは 10 以下である.
- 答えを ANS として，$O(RCANS)$ くらいの計算量で解けば良い

考察④ x 手で行ける範囲

周囲を黒マスで囲まれているとき,

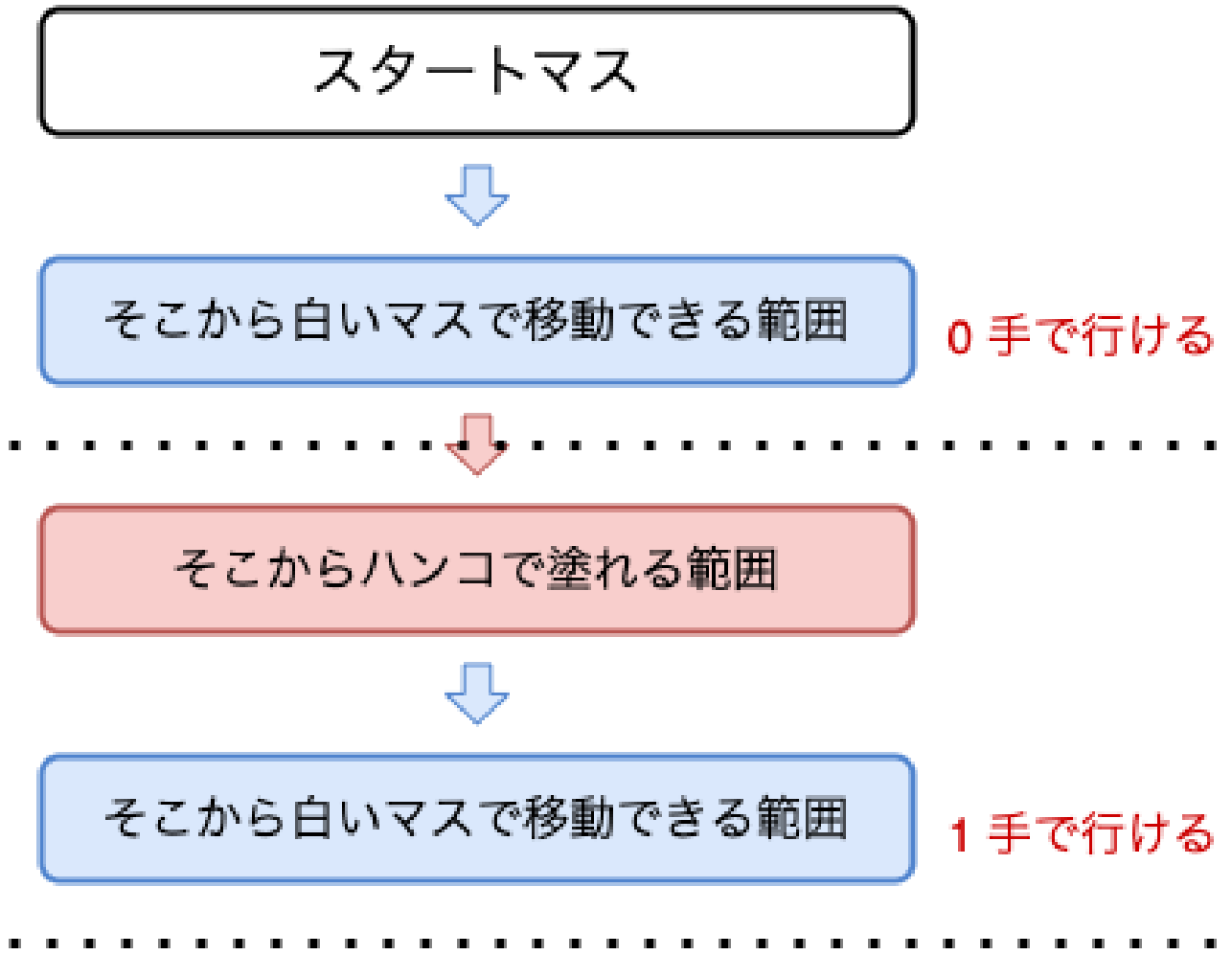
(1 手で行ける範囲)
= (ハンコで塗れる範囲) +
(そこから白いマスで移動できる範囲)

だった.



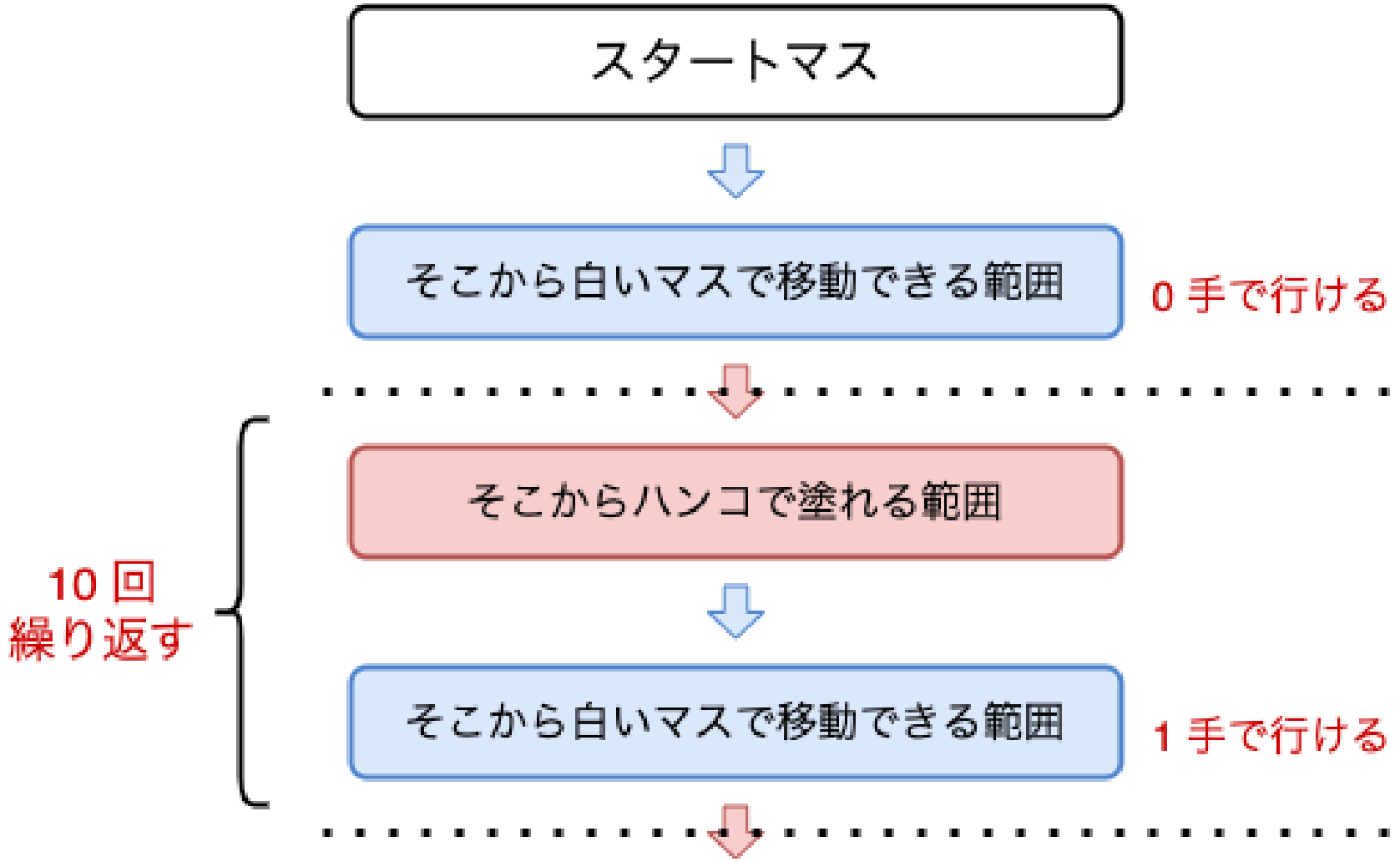
考察④ x 手で行ける範囲

図で整理するとこう



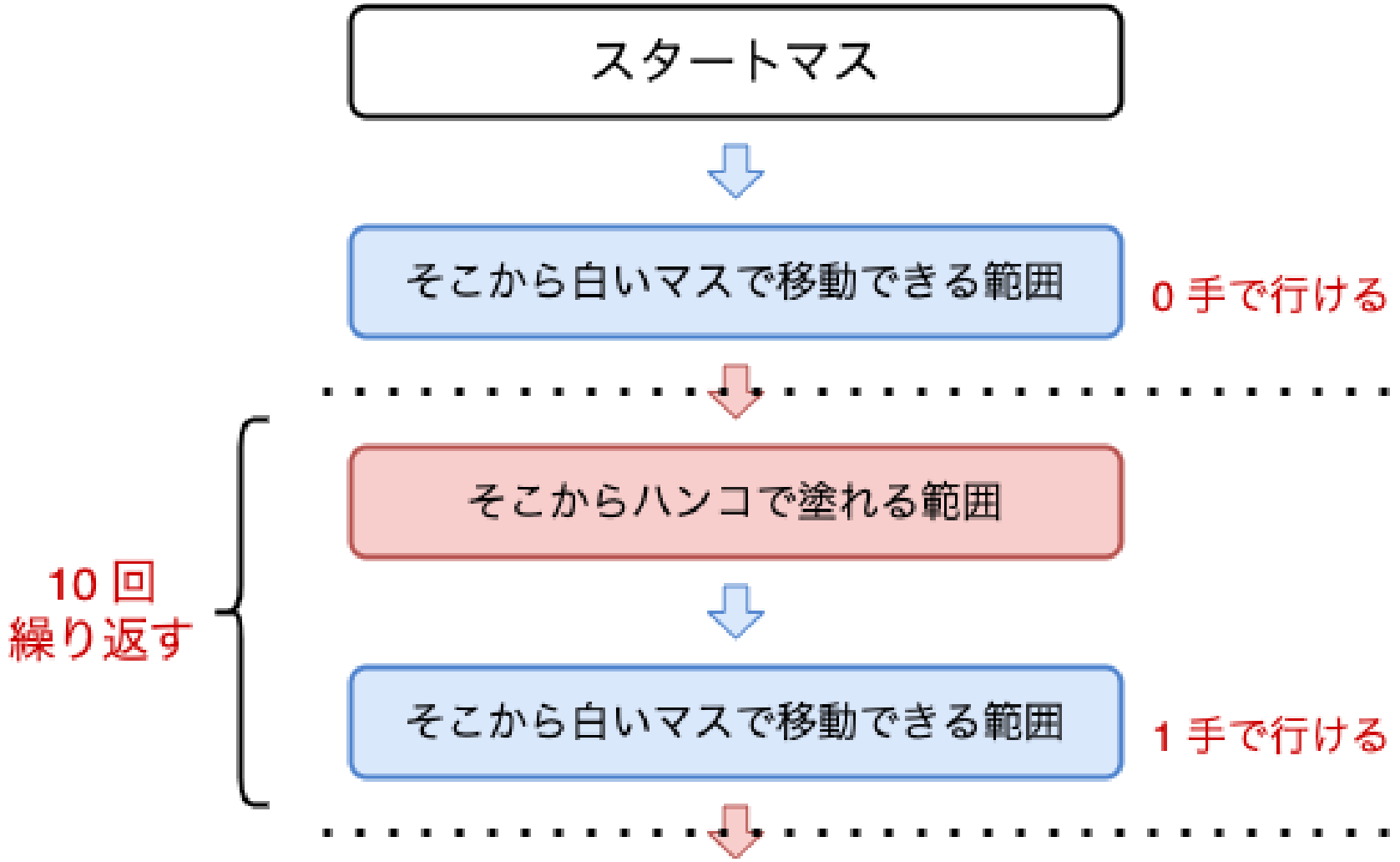
考察④ x 手で行ける範囲

$ANS \leq 10$ なら，
10 回繰り返せば答えがわかる



考察④ x 手で行ける範囲

$ANS \leq 10$ なら，
10 回繰り返せば答えがわかる



考察④ x 手で行ける範囲

そこから白いマスで移動できる範囲 は **BFS** で求められる.

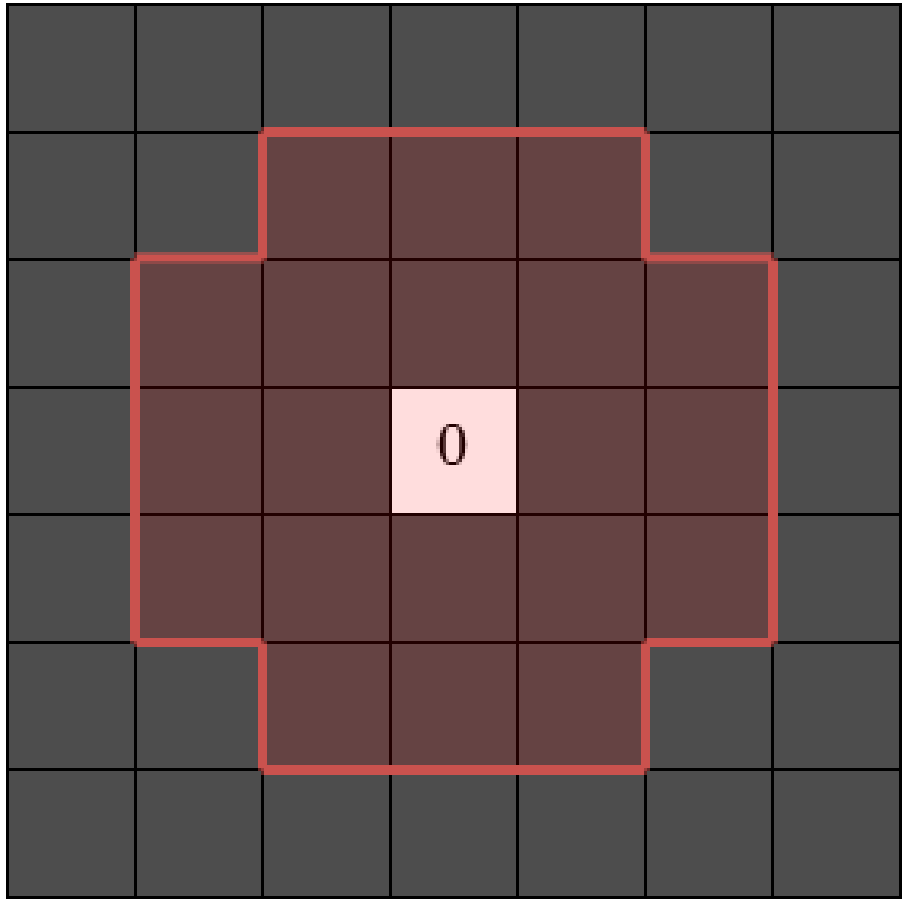
そこからハンコで塗れる範囲 は全探索だと遅いが，どうやって求める？

そこからハンコで塗れる範囲 を求める

- 1 つ前の範囲に入っている各マスから右図の形を伸ばして，それらの和集合を取りたい．

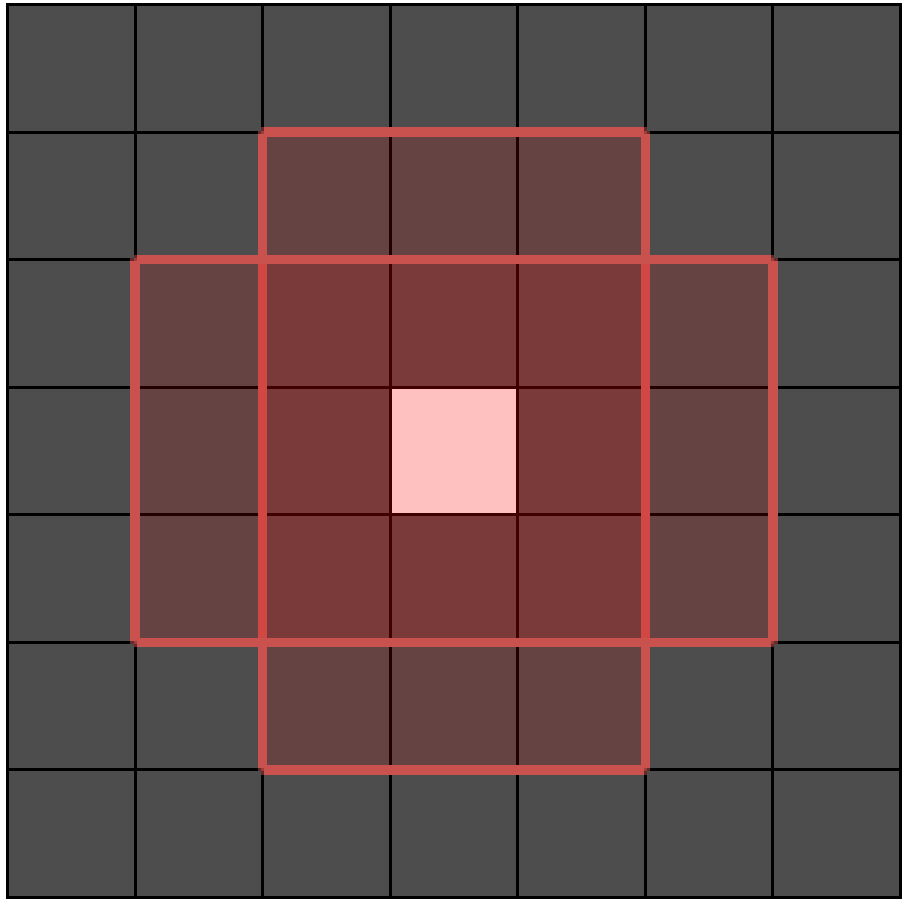
範囲加算・1 点取得　これは…

いわゆる **いもす法！**



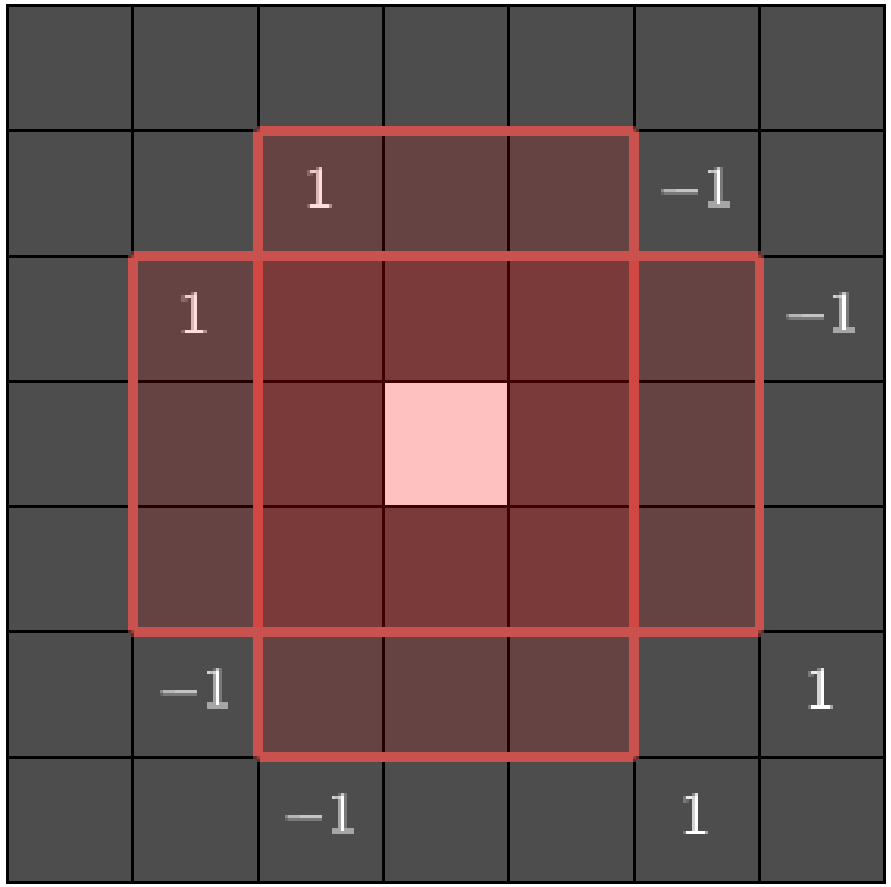
いもす法

- 正方形から四隅を抜いた形 を
  $(2N+1) \times (2N-1)$ マスの長方形と
  $(2N-1) \times (2N+1)$ マスの長方形に**分解**
- 和集合を取るので，多く加算しても良い



いもす法

- 正方形から四隅を抜いた形 を
  $(2N+1) \times (2N-1)$ マスの長方形と
  $(2N-1) \times (2N+1)$ マスの長方形に**分解**
- 和集合を取るので，多く加算しても良い
- ± 1 を配置し，後から 2 次元累積和
- 値が正になった部分が，**そこからハンコで塗れる範囲** である．



小課題 3 の解法まとめ

$O(RCANS)$ で解ける.

豆知識 : queue は空でも 512 Byte のメモリを食う

- `std::queue` や `std::stack` は別のコンテナ型を用いて動く型である．デフォルトでは `std::deque` を用いる.
- GCC では，`std::deque` は 512 Byte ごとのブロックでメモリ確保を行う．空でも 1 ブロック確保するため，`std::deque` は空でも 512 Byte のメモリを食う.

→ queue は `std::vector` で代用できる.
先頭要素を削除する代わりに **先頭の index を管理** する変数 `s` を持ち，`s++` で pop を行う.

注意

- ここから先は**考察要素の組み合わせ**によって様々な計算量が考えられます.
- **実装方針による定数倍**も様々あり，計算量が同じでも様々な点数になることが予想されます.
  - 実際にグラフを構築して遅くなる
  - 01-BFS の代わりにダイクストラ法を使って $\log$ をつける
- したがって，考察要素のみを紹介します.

小課題 4 (19 点)

- $R \times C \leq 6 \times 10^4$
- $N \leq R \leq C$ に注意すると，$N \leq \sqrt{RC} \leq \sqrt{6 \times 10^4} < 245$
- $O(RCN \log N)$ などの計算量で解けば良い.

小課題 5 (5 点)

- $R \times C \leq 1.5 \times 10^5$
- $O(RCN)$ や $O((RC)^{3/2})$ などの計算量で解けば良い.

小課題 6 (19 点)

- $R \times C \leq 1.5 \times 10^6$
- $O(RC(\log N)^2)$ や $O((RC)^{4/3})$ などの計算量で解けば良い.

小課題 7 (8 点)

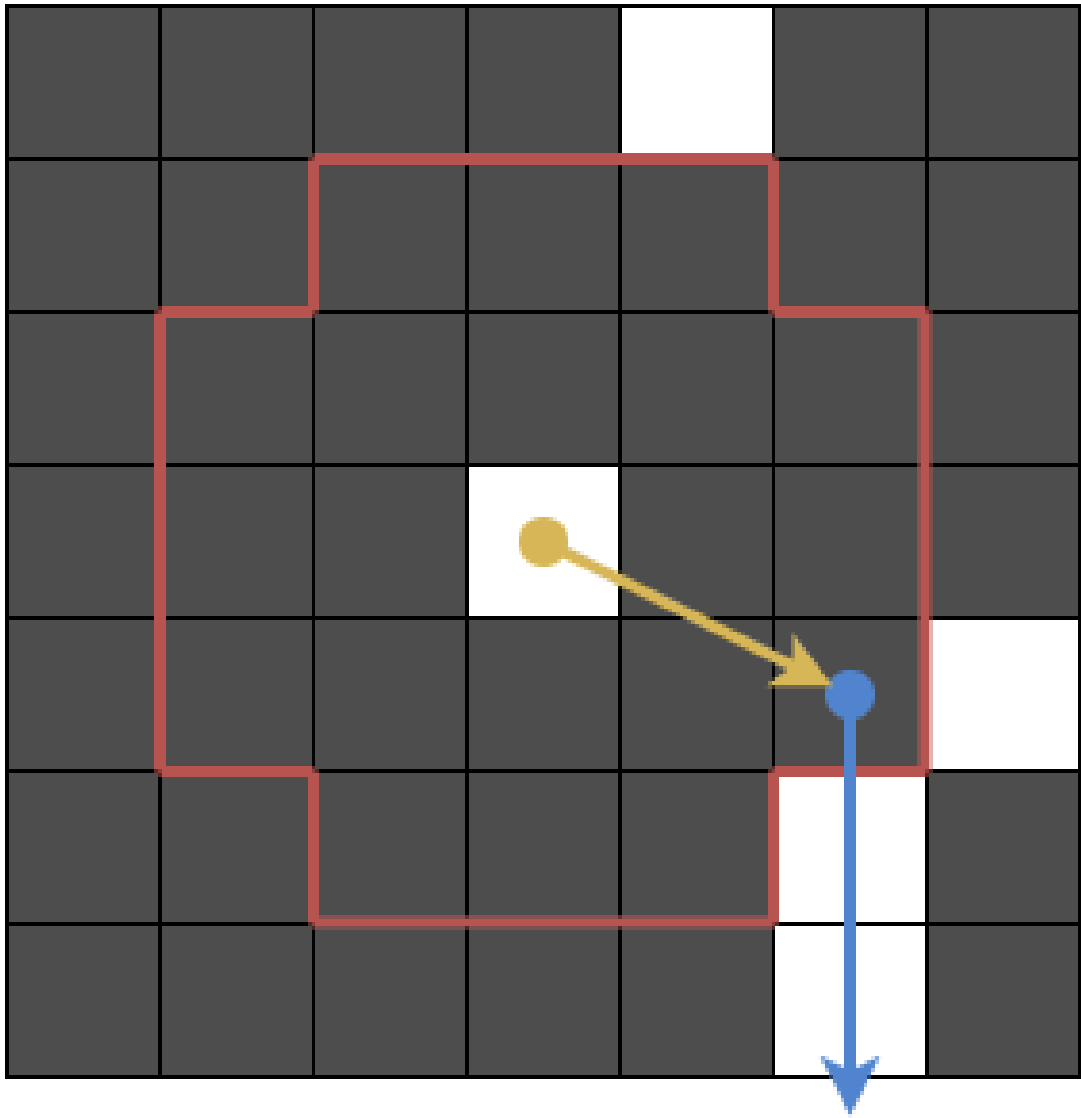
- $R \times C \leq 3 \times 10^6$
- $O(RC \log N)$ などの計算量で解けば良い.

小課題 8 (6 点)

- $R \times C \leq 6 \times 10^6$
- $O(RC)$ などの計算量で解けば良い.

考察⑤ 外側しか遷移しなくて良い

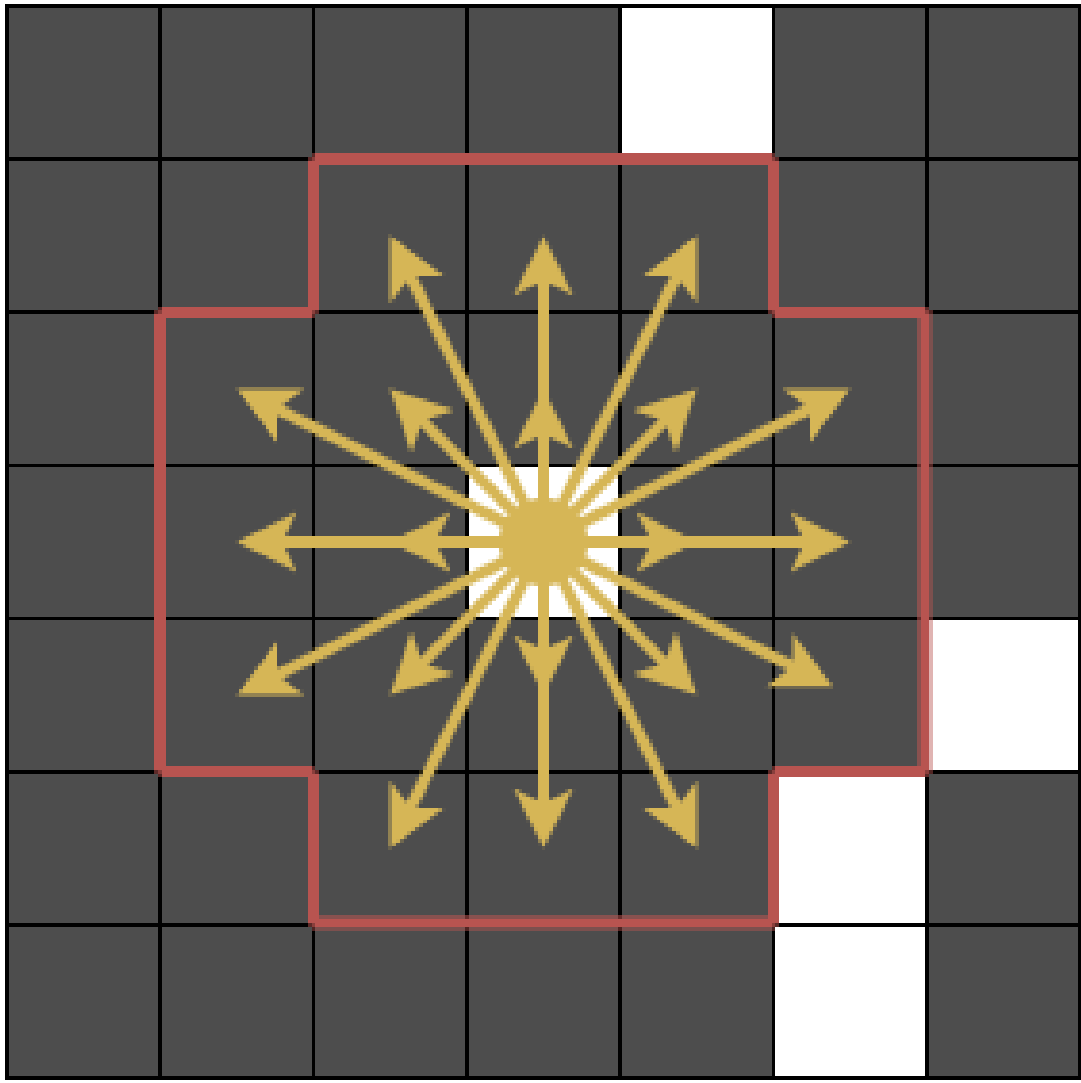
小課題 2 では，ハンコで塗れる範囲 を 1 回までしか出ないので，ハンコをワープに言い換えていた.



考察⑤ 外側しか遷移しなくて良い

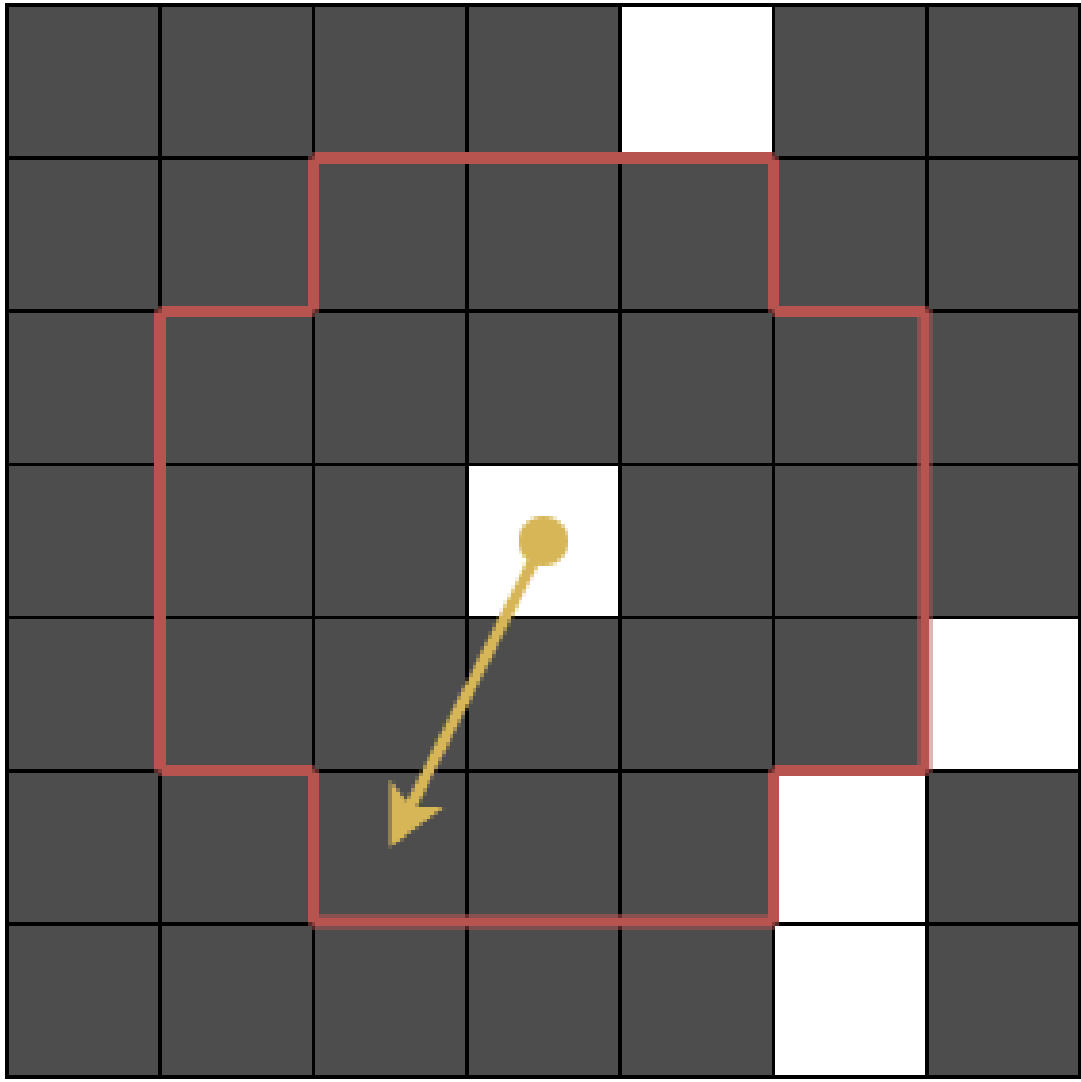
小課題 2 では，ハンコで塗れる範囲 を 1 回までしか出ないので，ハンコをワープに言い換えていた.

また，各マスから ハンコで塗れる範囲 全てにコスト 1 の辺を張っていた.



考察⑤ 外側しか遷移しなくて良い

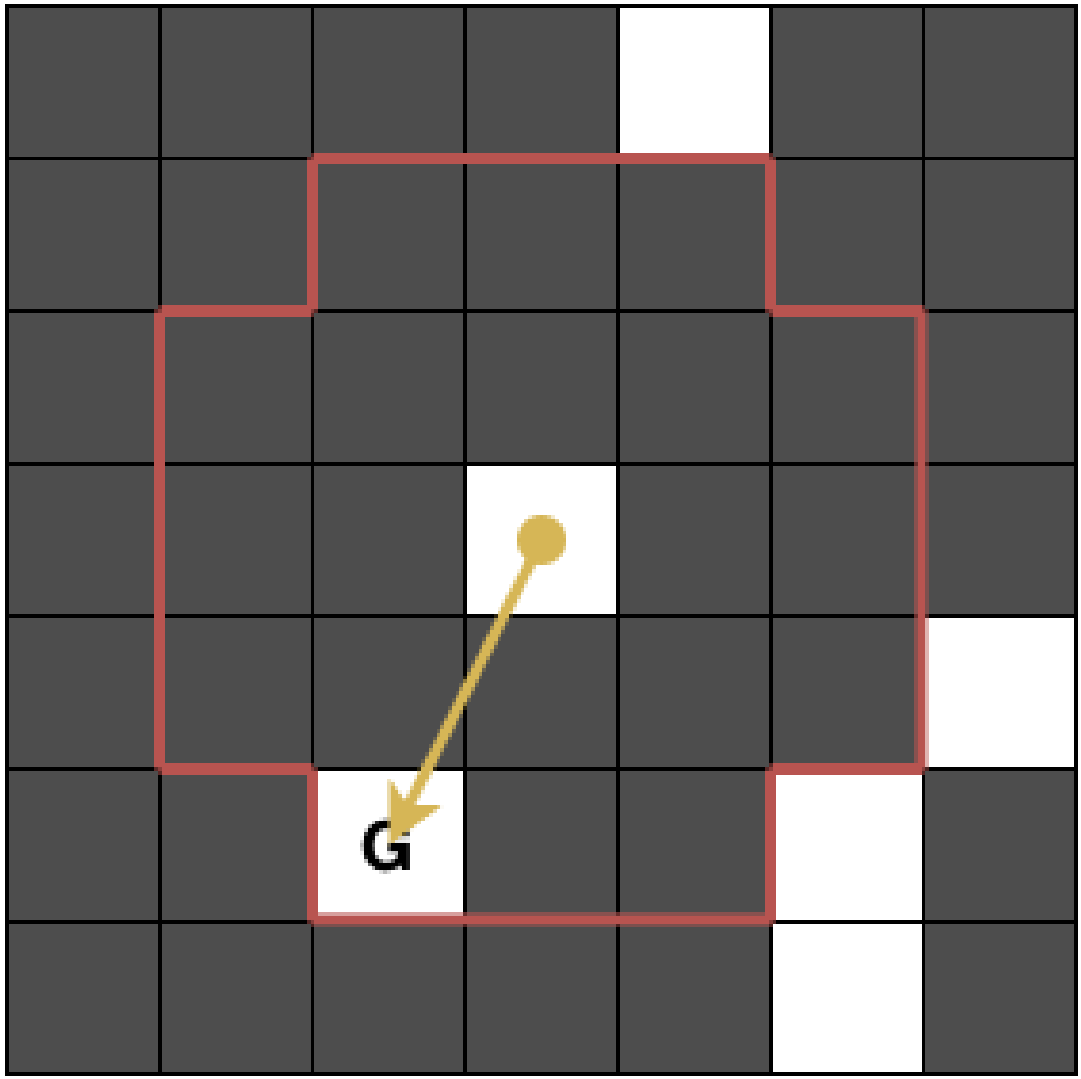
Q. ハンコで塗れる範囲 を 1 回も出ないときってどんなとき？



考察⑤ 外側しか遷移しなくて良い

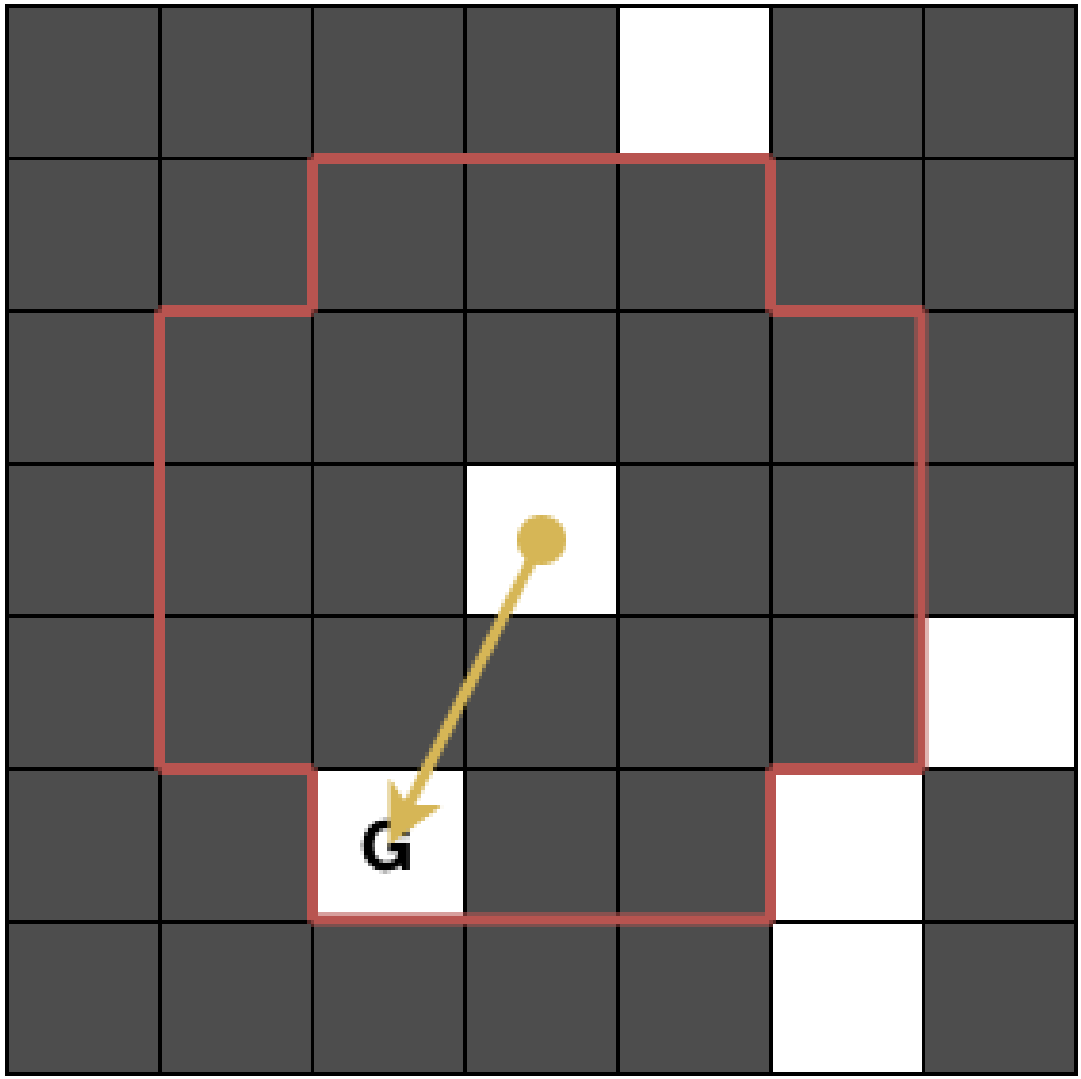
Q. ハンコで塗れる範囲 を 1 回も出ないときってどんなとき？

A. ハンコで塗れる範囲 の中にゴールがあるとき



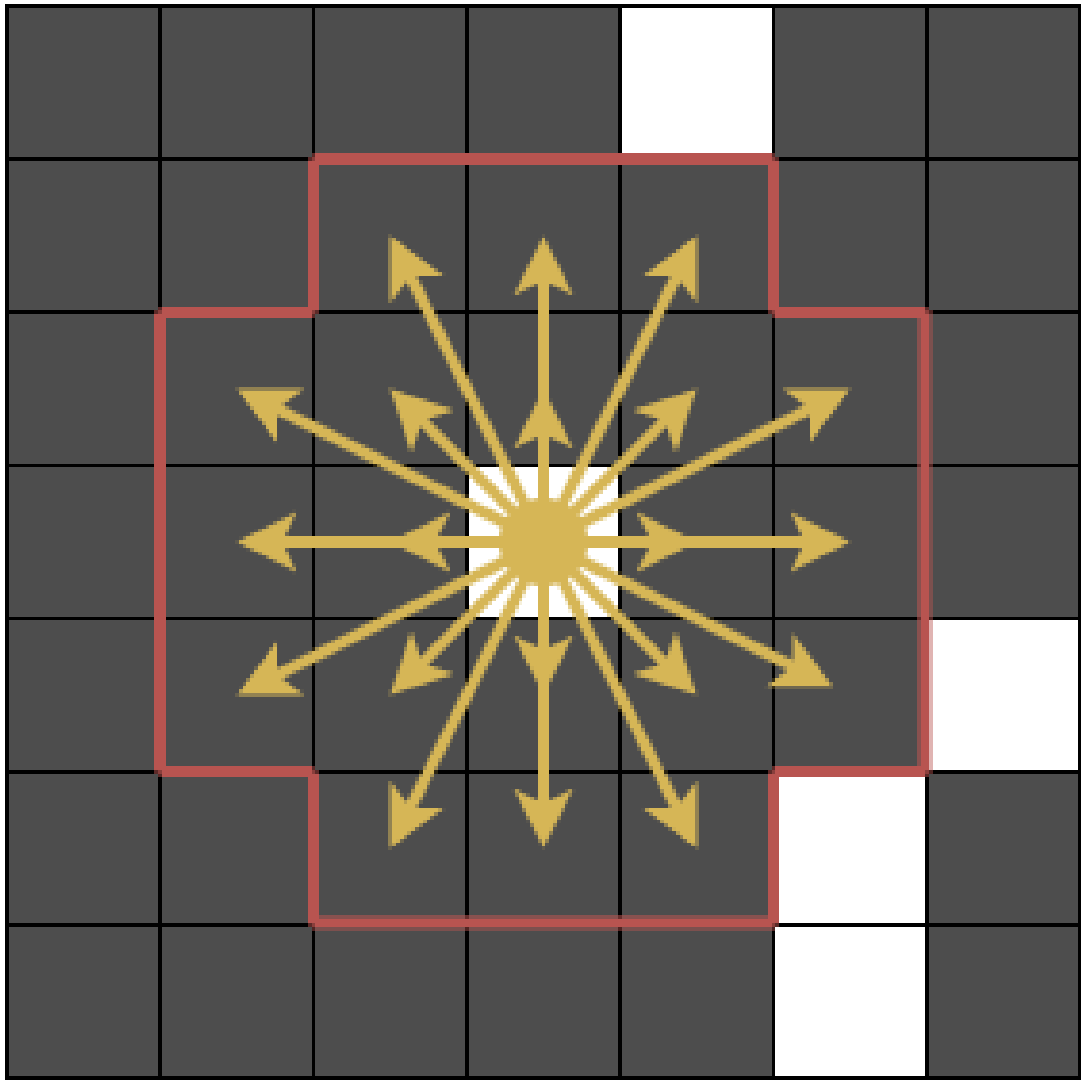
考察⑤ 外側しか遷移しなくて良い

- ハンコで塗れる範囲 の中にゴールがあるとき，ゴールにワープできて，終了
- ハンコで塗れる範囲 の外にゴールがあるとき，絶対に ハンコで塗れる範囲 を出なければならない



考察⑤ 外側しか遷移しなくて良い

ということは，ハンコで塗れる範囲 全部に辺を張る必要はなくて…

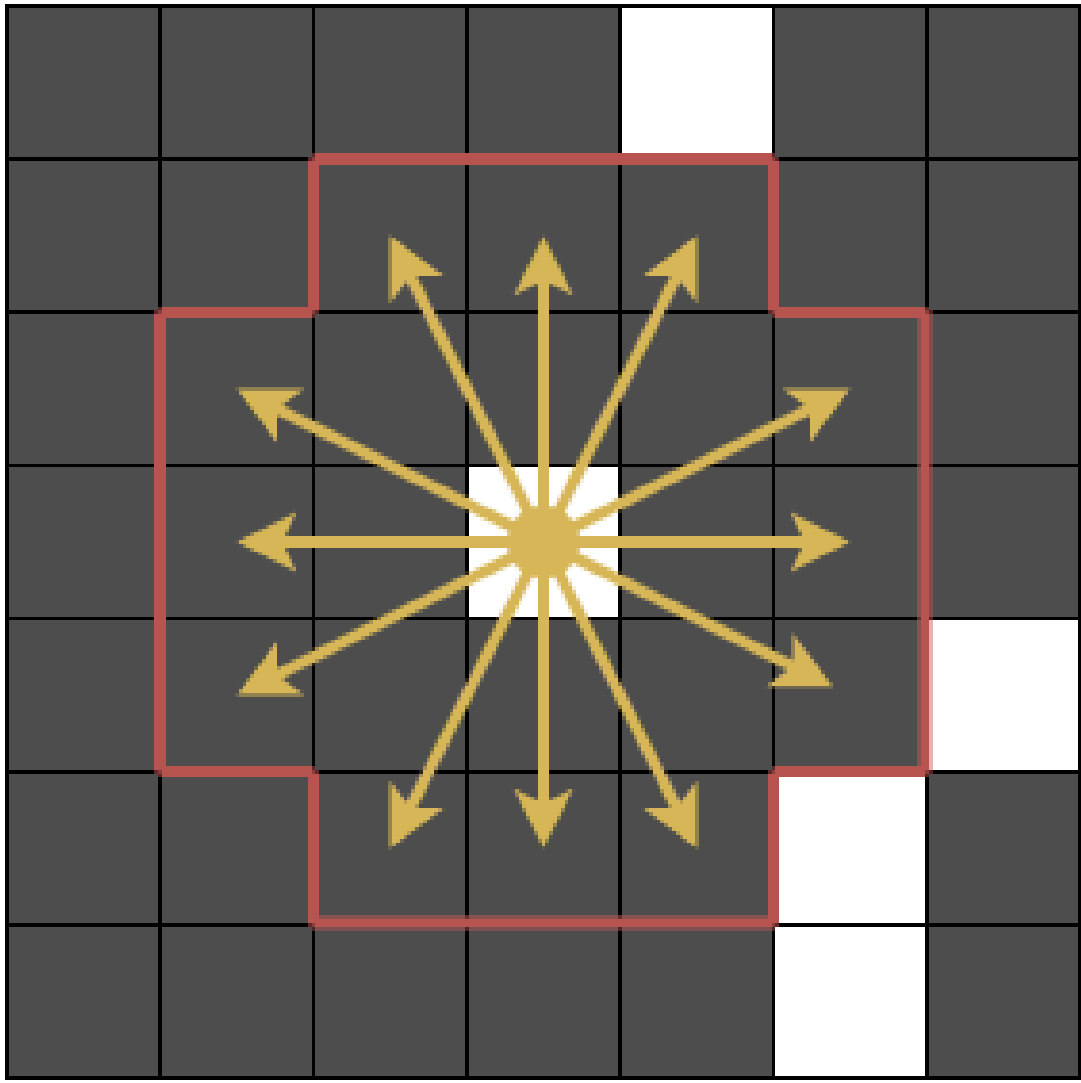


考察⑤ 外側しか遷移しなくて良い

ということは，ハンコで塗れる範囲 全部に辺を張る必要はなくて，

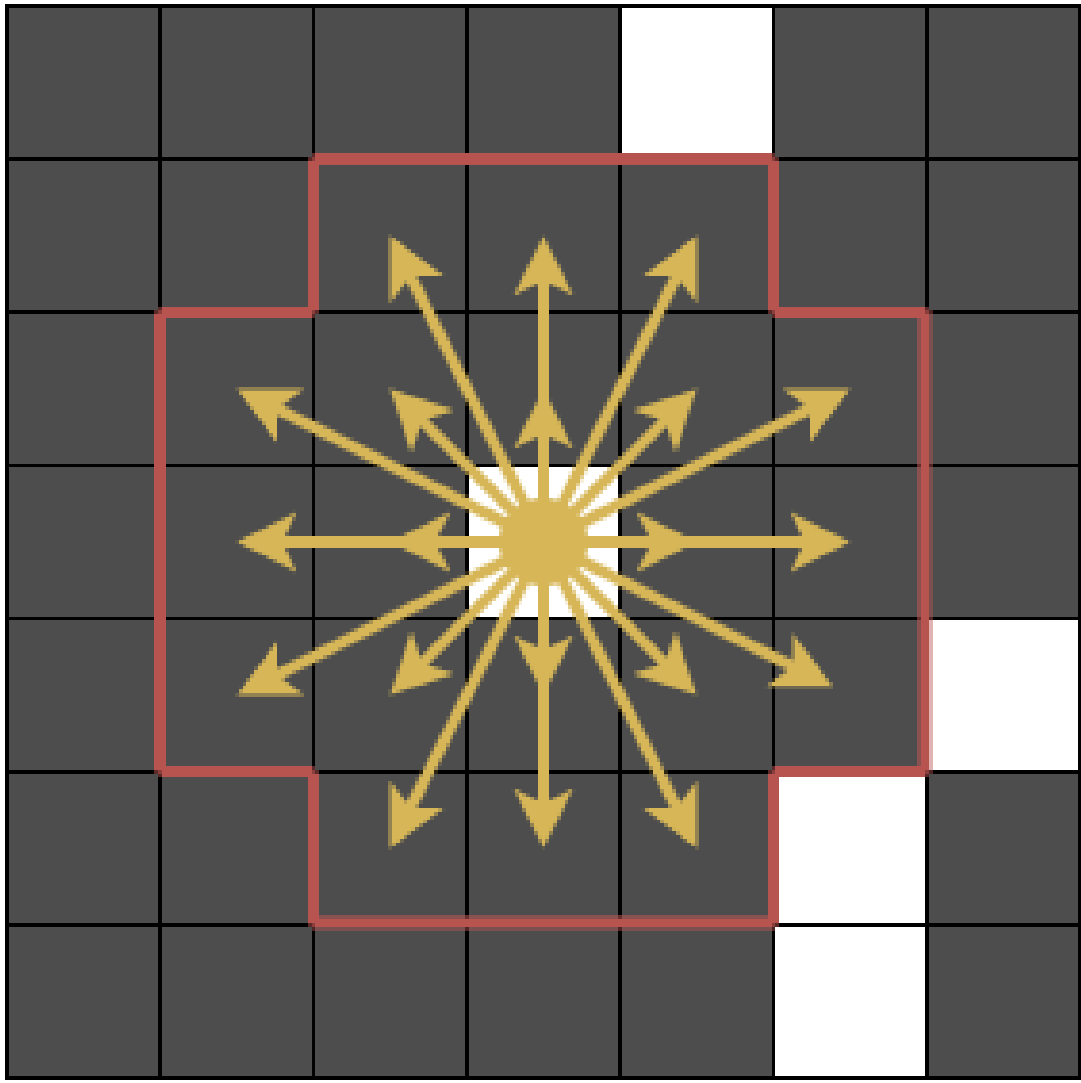
最も外側の $4 \times (2N - 1)$ マスにのみ辺を張れば良い！

※ 範囲内 にゴールがあればゴールへの辺も必要



考察⑤ 外側しか遷移しなくて良い

- 辺の個数が $O(RCN)$ 本になったので，01-BFS で $O(RCN)$ 時間



考察⑥ 解法を組み合わせる

- 小課題 3 の計算量は $O(RCANS)$ だった.
- N が大きくなるほど，1 度に広い範囲を塗れるので，ANS は小さくなる
- 具体的には,
  $ANS \leq \lceil \max(\frac{R+C}{2N-1}, \frac{C}{N}) \rceil \leq \frac{2C}{N} + 1$

考察⑥ 解法を組み合わせる

- ANS は N に反比例して小さくなる
- 一方，$O(RCN^2)$ や $O(RCN)$ などの解法は N に比例して時間がかかる

→ 解法を組み合わせよう！

考察⑥ 解法を組み合わせる

- $O(RCN^2)$ が間に合うときは小課題 2 の解法 $O(RCN^2)$ で，そうでないときは小課題 3 の解法 $O(RCANS)$ で解く
- $O(RC\min(N^2, ANS))$ 時間
- $ANS \leq \frac{2C}{N} + 1$ に注意すると，$O((RC)^{3/2})$ 時間であることが分かる

考察⑥ 解法を組み合わせる

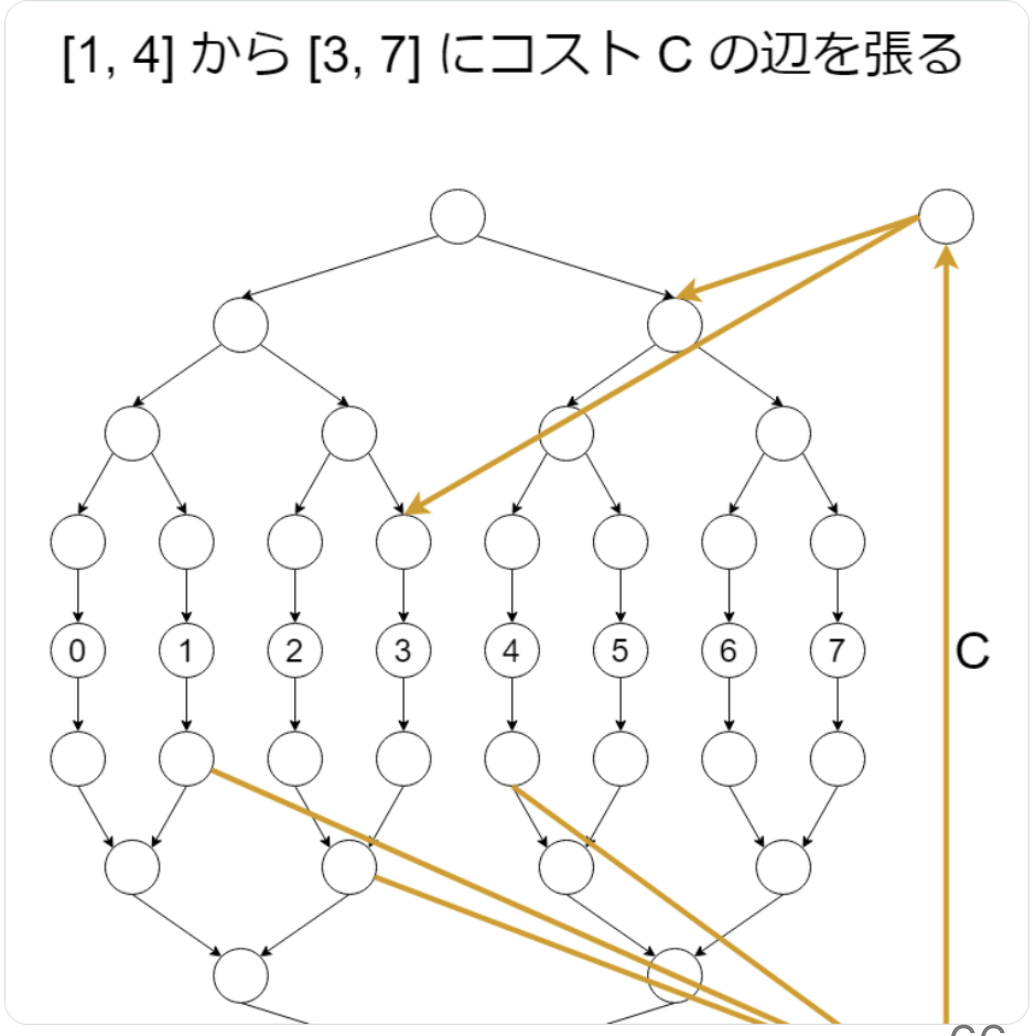
- $O(RCN)$ が間に合うときは外側しか遷移しない解法 $O(RCN)$ で，そうでないときは小課題 3 の解法 $O(RCANS)$ で解く
- $O(RC\min(N, ANS))$ 時間
- $ANS \leq \frac{2C}{N} + 1$ に注意すると，$O((RC)^{4/3})$ 時間であることが分かる

区間に辺を張るテクニック

セグメント木状に頂点を増やしておくと，長さ $2N+1$ の区間に辺を張るのに $2N+1$ 本必要だったのが，$2\log_2 N$ 本くらいで張れるようになる

熨斗袋
@noshi91 · Follow

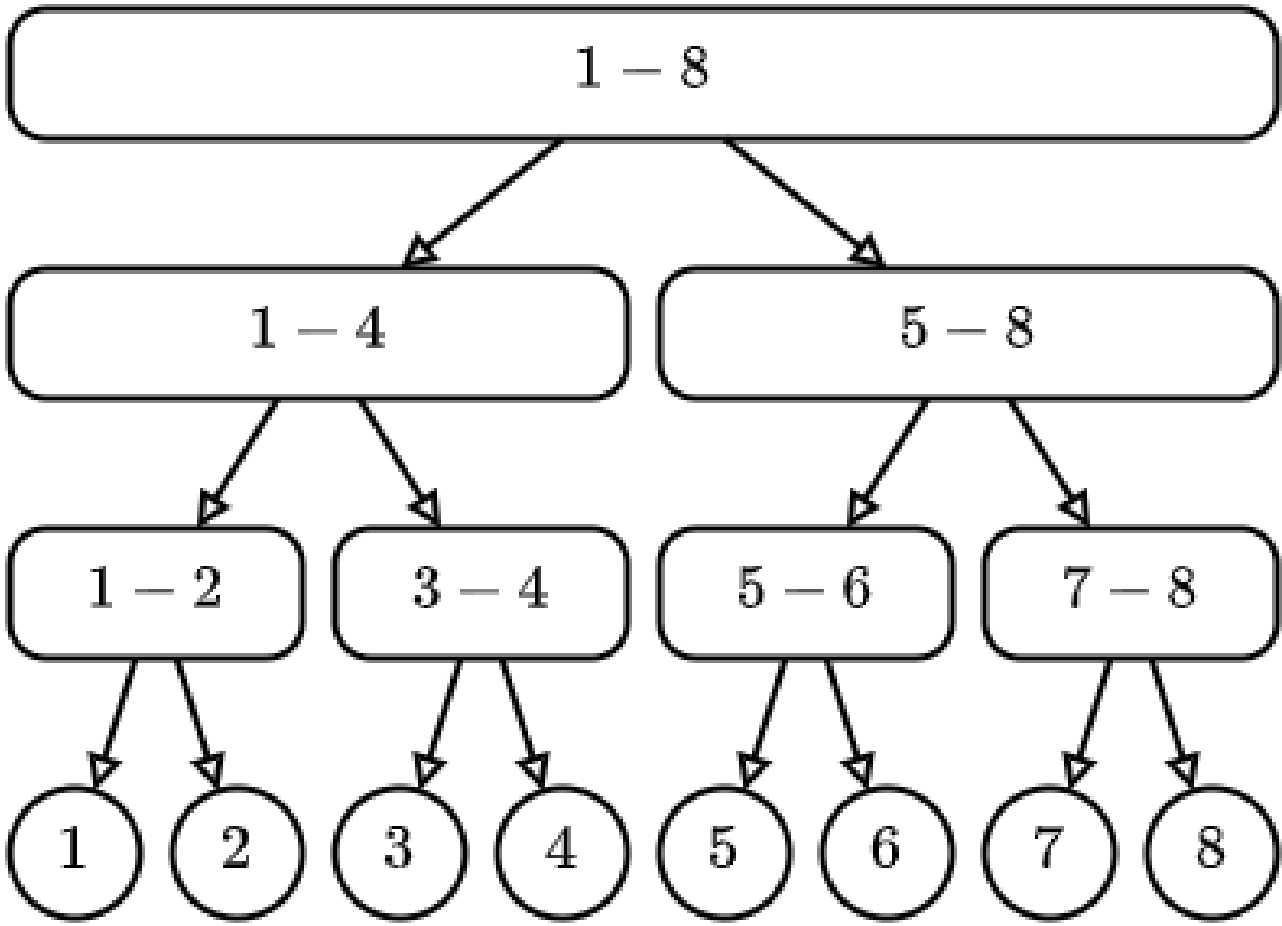
セグ木の形にして区間に辺を張るテク
頂点 +N 個
辺 +N+ElogN 個



11:43 PM · Nov 9, 2019

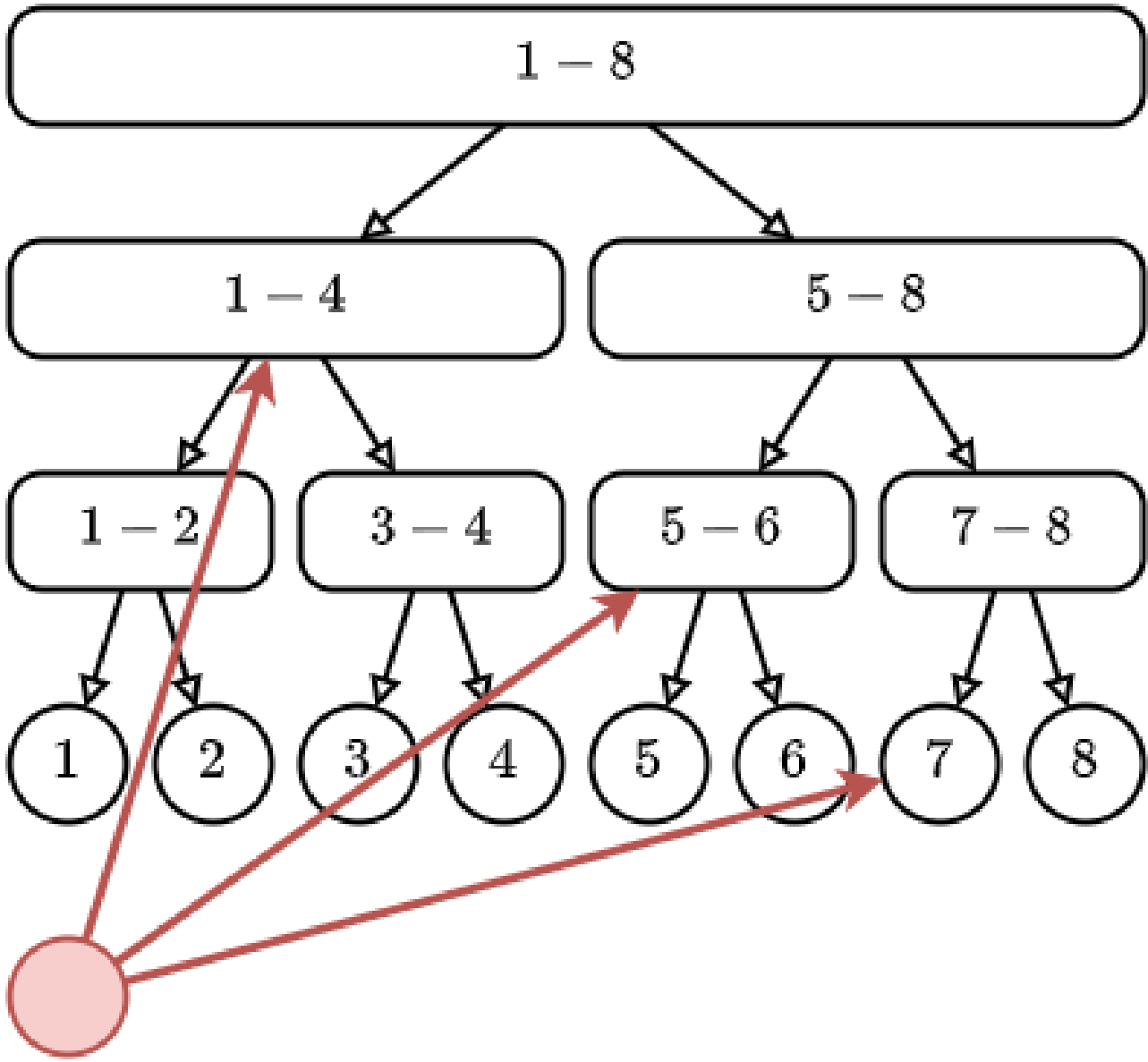
区間に辺を張るテクニック

- セグメント木状に頂点を用意し，下方向にコスト 0 の辺を張る



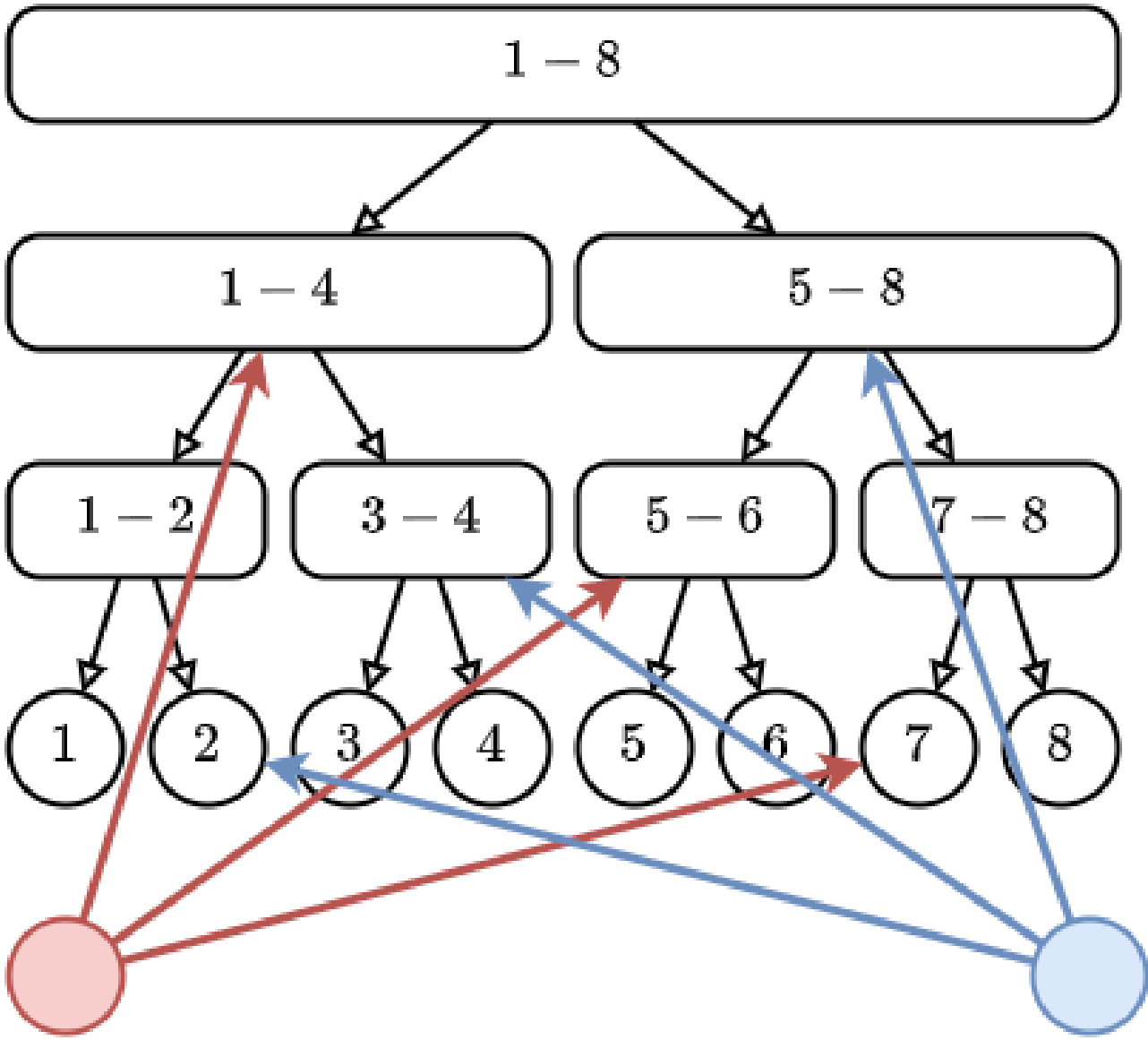
区間に辺を張るテクニック

- セグメント木状に頂点を用意し，下方向にコスト 0 の辺を張る
- 区間 $[1, 7]$ にコスト 1 の辺を張る



区間に辺を張るテクニック

- セグメント木状に頂点を用意し，下方向にコスト 0 の辺を張る
- 区間 $[1, 7]$ にコスト 1 の辺を張る
- 区間 $[2, 8]$ にコスト 1 の辺を張る



区間に辺を張るテクニック

- 小課題 2 の $O(RCN^2)$ 時間
  → $O(RCN \log N)$ 時間に
- 外側にしか辺を張らなくていい $O(RCN)$ 時間
  → $O(RC \log N)$ 時間に (定数倍によっては満点を取れる)

注意 : グラフを実際に構築すると遅い

豆知識 : 簡単なセグ木の作り方

長さ $2n$ のセグ木にするとよい

> 出典 : 非再帰セグ木サイコー！ 一番すきなセグ木です p.35

驚愕の事実？

上のコードは要素数 n を二ベキにしなくても動く.

O(log n) 個の完全二分木のセグ木が存在していると見なせる.

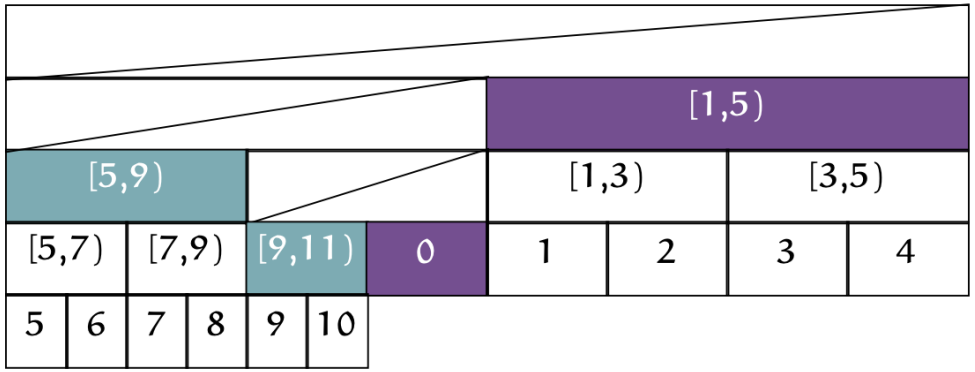
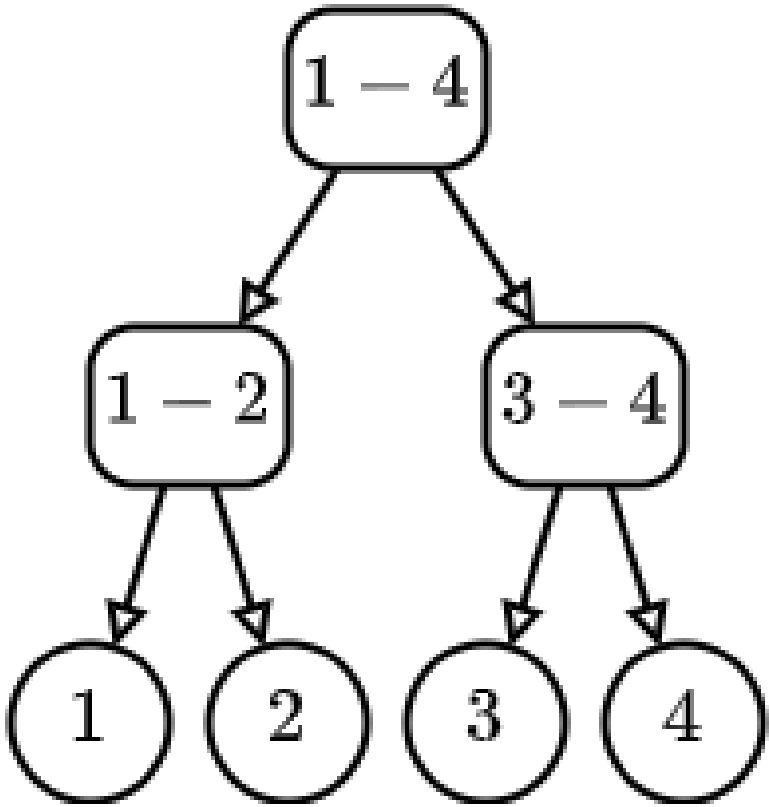


Figure: 11 要素の配列に対応するセグ木

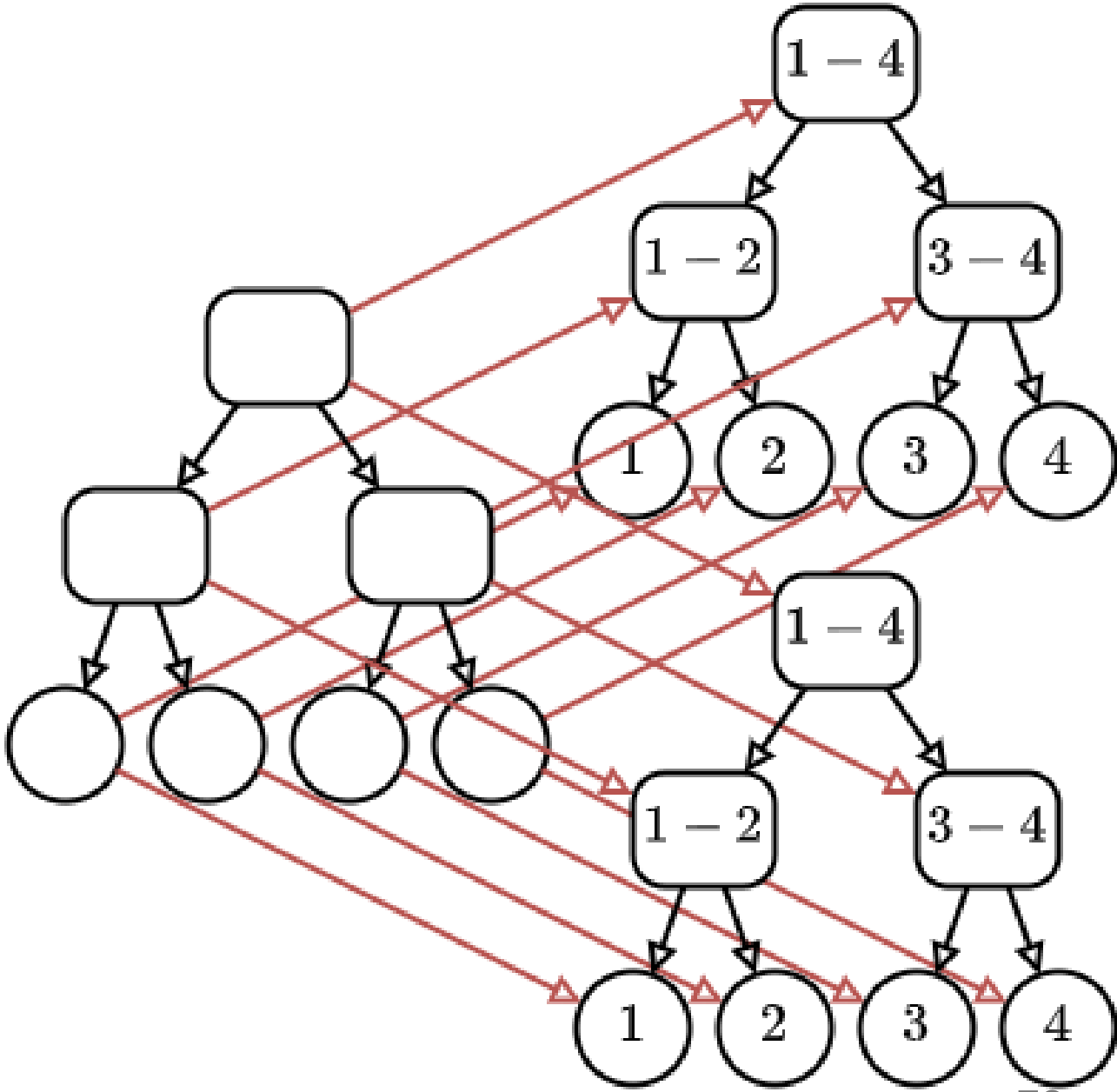
区間に辺を張るテクニック②

- 1 次元セグ木でできるなら，2 次元セグ木でもできるよね



区間に辺を張るテクニック②

- 1 次元セグ木でできるなら，2 次元セグ木でもできるよね



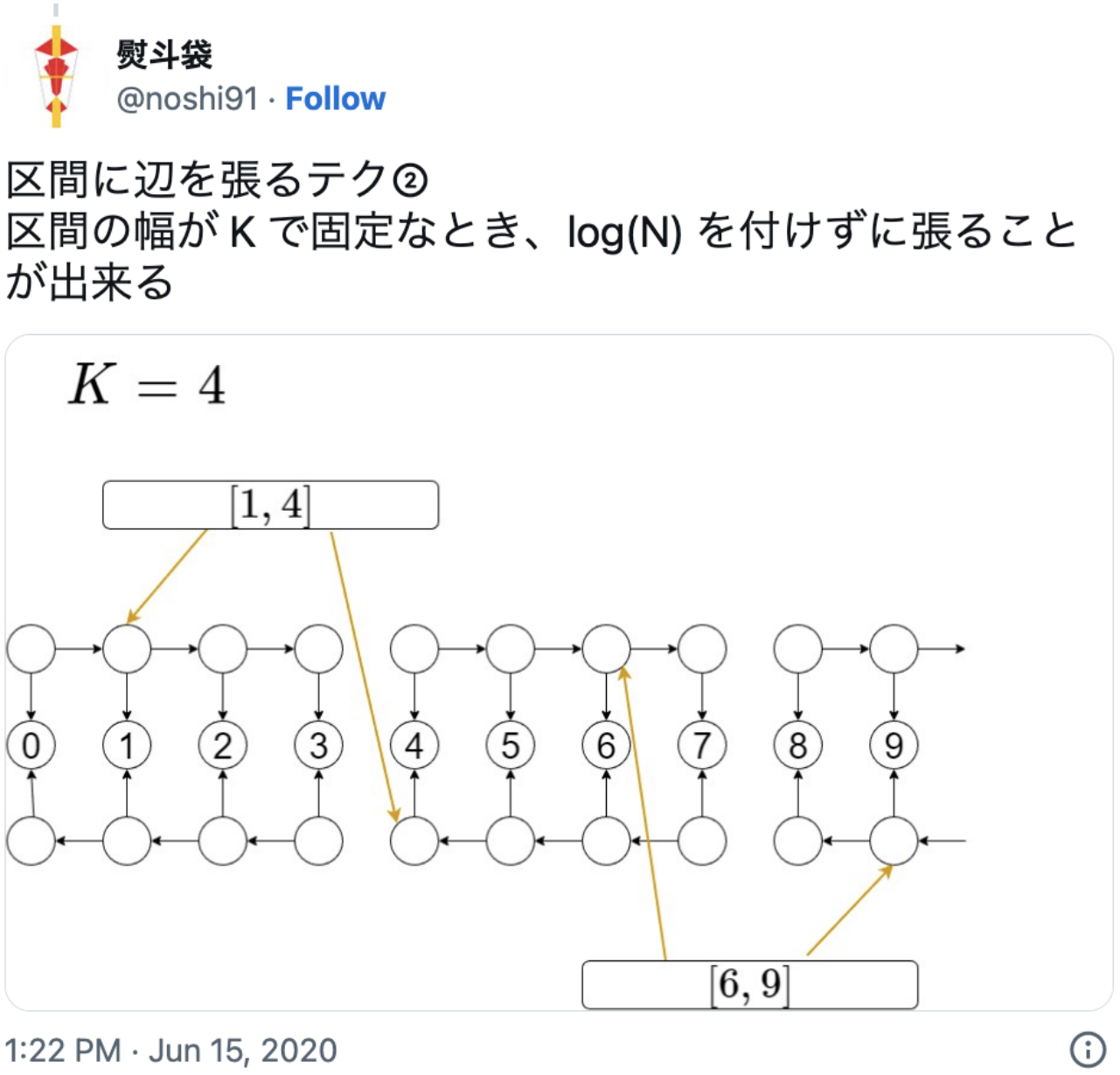
区間に辺を張るテクニック②

- 2 次元セグ木を使って長方形に辺を張る
- 小課題 2 の $O(RCN^2)$ 時間
  → $O(RC(\log N)^2)$ 時間に (小課題 6 まで取れる)

注意 : グラフを実際に構築すると遅い

区間に辺を張るテクニック③

- 列を N 個くらいのブロックに区切って，各ブロック内で右側方向に繋げる頂点と，左側方向に繋げる頂点を作る.

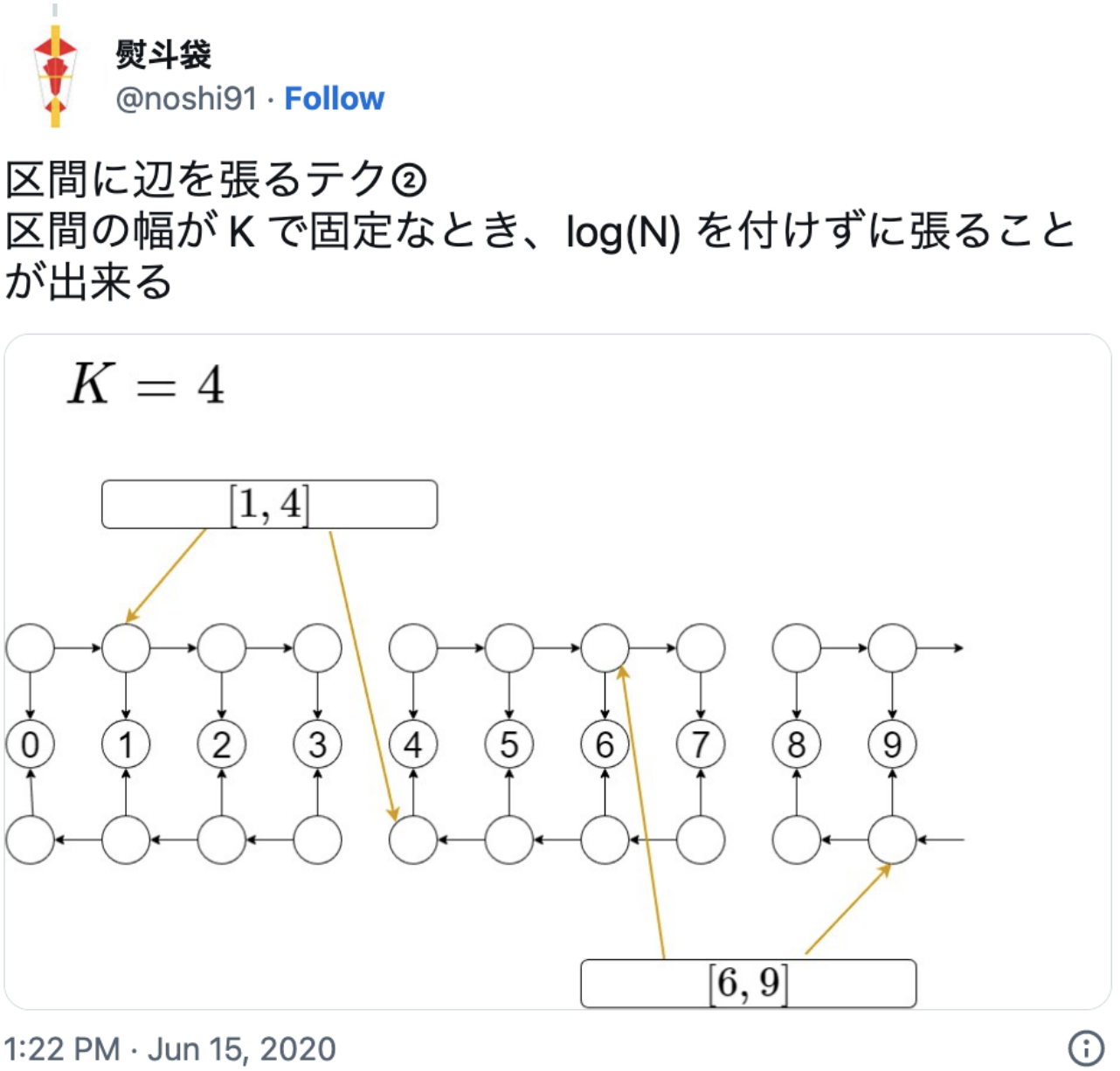


区間に辺を張るテクニック③

- 列を N 個くらいのブロックに区切って，各ブロック内で右側方向に繋げる頂点と，左側方向に繋げる頂点を作る.
- 小課題 2 の $O(RCN^2)$ 時間
  → $O(RCN)$ 時間に
- 外側にしか辺を張らなくていい $O(RCN)$ 時間
  → $O(RC)$ 時間に

区間に辺を張るテクニック④

- こっちも同様に 2 次元にできる
- $N \times N$ マスくらいのブロックに区切って，各ブロック内で右上方向，右下方向，左上方向，左下方向に繋げる頂点を作る



区間に辺を張るテクニック④

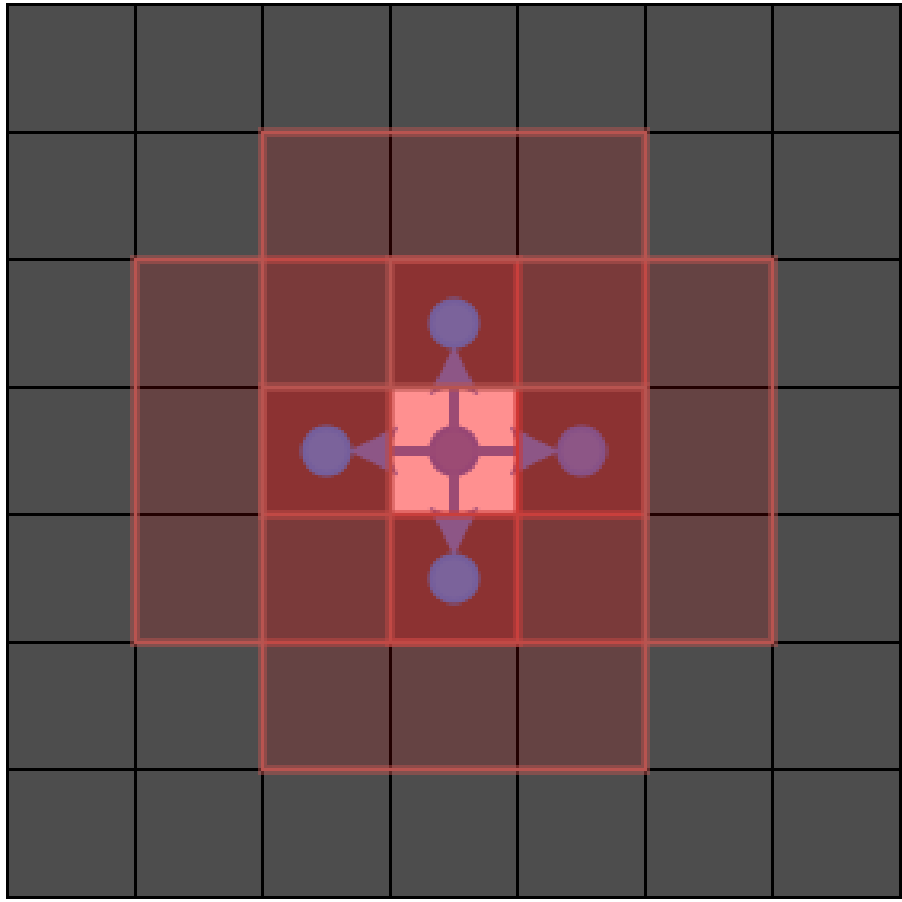
- $N \times N$ マスくらいのブロックに区切って，各ブロック内で右上方向，右下方向，左上方向，左下方向に繋げる頂点を作る
- 小課題 2 の $O(RCN^2)$ 時間
  → $O(RC)$ 時間に (定数倍が重いので，工夫をしないと小課題 7 までかも)

考察⑦ 3 段階に分ける

- 小課題 3 の解法は，$\Theta(RC)$ 時間の 2 次元いもす法を ANS 回繰り返すのがボトルネック
- これを全体で $\Theta(RC)$ にしたい．

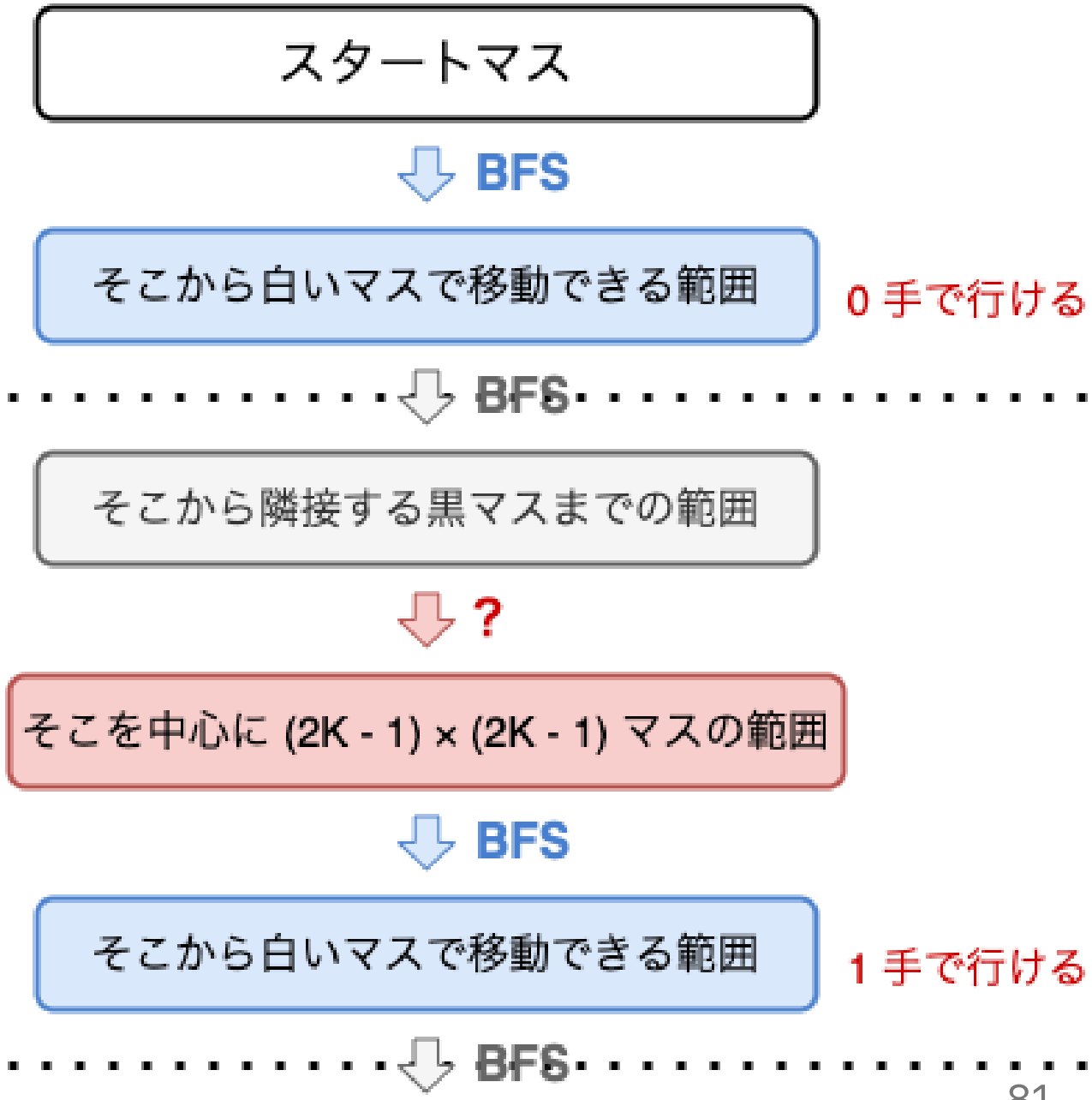
考察⑦ 3 段階に分ける

(ハンコで塗れる範囲) = (隣接する黒マスを中心に $(2N-1) \times (2N-1)$ マスの範囲)



考察⑦ 3 段階に分ける

周囲 $(2N-1) \times (2N-1)$ マス
に広げた範囲を計算するには？

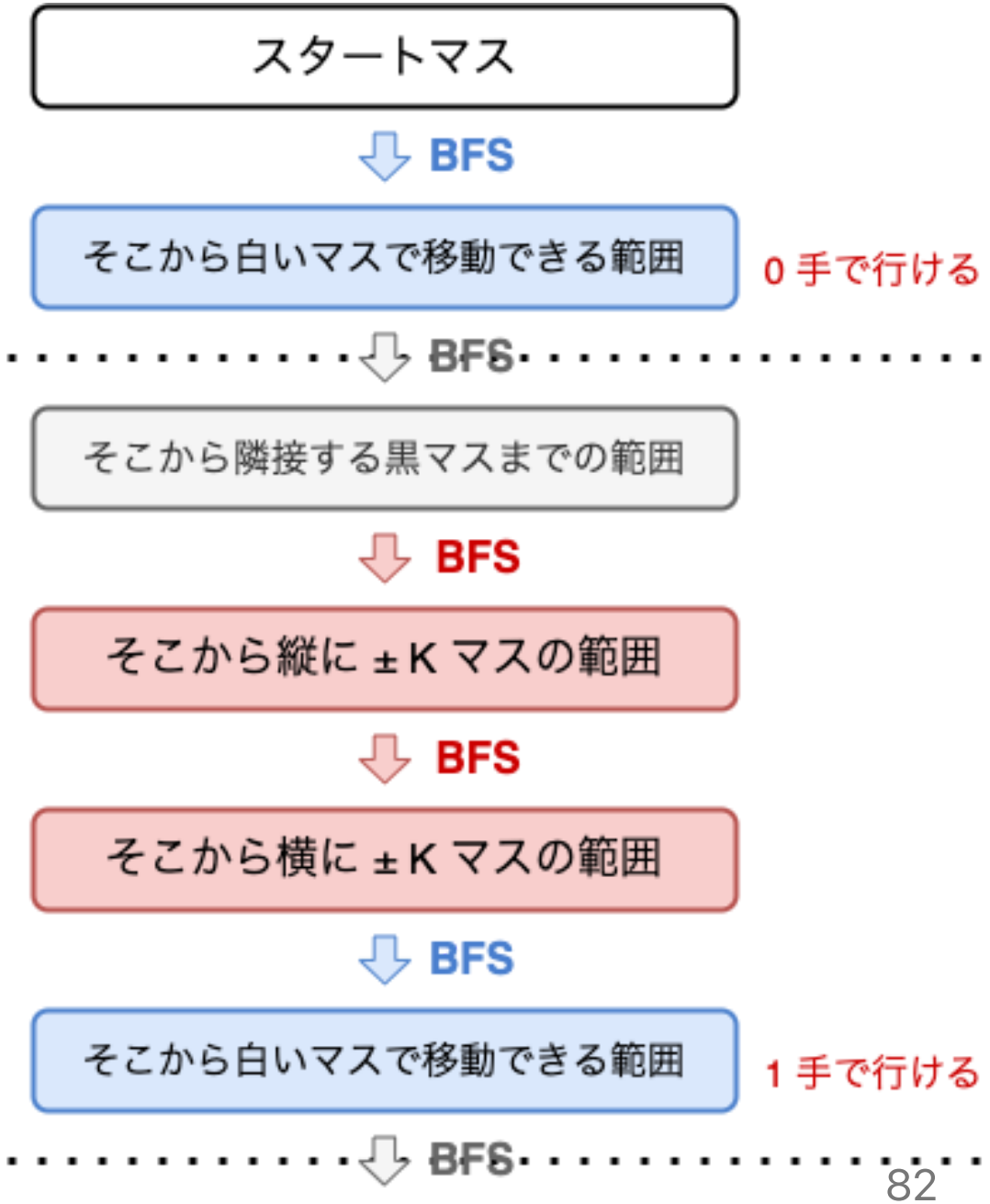


考察⑦ 3 段階に分ける

周囲 $(2N-1) \times (2N-1)$ マスに広げた範囲を計算するには？

→ 縦と横が独立なので，分けてやる

全部 BFS でできるので，$O(RC)$ 時間で解けた！ 💯



得点情報

得点	小課題	人数	累積人数
💯	12345678	8	8
94	1234567	4	12
86	123456	3	15
67	12345	2	17
62	1234	1	18
51	12 45	7	25
46	12 4	3	28
43	123	1	29
27	12	25	54
8	1	32	86