ABC454 E — LRUD Moving

市松模様の議論と再帰的構成によるハミルトンパス

2026-04-18　Reo Kambayashi

Agenda

1. **問題概要** — 何を問われているか・制約
2. **必要条件の発見** — 市松模様（チェッカーボード）で考える
3. **十分条件の証明** — 再帰的にパスを構成する
4. **構成法の詳細** — 行・列の剥がし方とベースケース
5. **実装** — C++ コード解説
6. **まとめ**

問題概要

N×N グリッド上のハミルトンパス

問題概要

入力：T 個のテストケース。各テストケースで正整数 N, A, B

設定：$N \times N$ のグリッド上でコマを $(1, 1)$ から (N, N) まで移動

条件：

- (A, B) **以外**の全てのマスをちょうど1回ずつ通る
- 移動は上下左右の隣接マスのみ（L / R / U / D）
- $(1, 1)$ と (N, N) を途中で再訪問しない
- 移動回数は $N^2 - 2$ 回（$N^2 - 1$ マスを訪問）

出力：可能なら `Yes` と経路、不可能なら `No`

> 制約：T ≤ 5000、N ≤ 10^3、全テストケースの N^2 の総和 ≤ 10^6

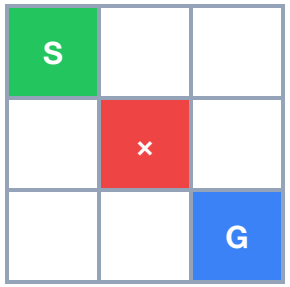
サンプルで理解する

N=2, A=1, B=2

経路：**D → R**（`DR`）

$(1,1) \to (2,1) \to (2,2)$　✓

N=3, A=2, B=2



答え：**No**（構成不可能）

> N=3 で No になる理由はまだ自明ではない。なぜ不可能なのか？→ 次のセクションで解明する。

必要条件の発見

市松模様（チェッカーボード）で考える

市松模様の着色

マス (i, j) を $i + j$ の偶奇で**白と黒**に塗り分ける。

白	黒	白	黒
黒	白	黒	白
白	黒	白	黒
黒	白	黒	白

N=4 の市松模様

重要な性質：

- $(1, 1) : 1 + 1 = 2$（偶数）→ **白**
- $(N, N) : N + N = 2N$（偶数）→ **白**
- **上下左右に1歩動くと必ず色が変わる**

つまりパス上では **白→黒→白→黒→…** と色が交互に並ぶ。

> 出発点・到着点がともに白 → パスの長さ（訪問マス数）が奇数でなければならない。

必要条件①：N は偶数

パスは $N^2 - 1$ マスを訪問する。出発・到着がともに**白**なので：

白→黒→白→…→白 と交互 → 訪問マス数 $N^2 - 1$ は**奇数**が必要。

N が奇数のとき

$N^2 - 1 = (\text{奇数})^2 - 1 = \text{偶数}$

→ **偶数個のマス**を訪問する
→ 白で始まり**黒で終わる**
→ (N, N) は白なので矛盾

N が偶数のとき

$N^2 - 1 = (\text{偶数})^2 - 1 = \text{奇数}$

→ **奇数個のマス**を訪問する
→ 白で始まり**白で終わる** ✓

> **必要条件①**：N は偶数でなければならない。N が奇数なら即 No。

必要条件②：A+B は奇数

N が偶数のとき、$N^2 - 1$ マスを訪問する。色の内訳を数える。

パス上の色の数：

- 白マス：$\frac{N^2-1+1}{2} = \frac{N^2}{2}$ 個（最初と最後が白 → 白が1つ多い）
- 黒マス：$\frac{N^2-1-1}{2} = \frac{N^2}{2} - 1$ 個

グリッド全体の色の数（N が偶数のとき）：

- 白マス：$\frac{N^2}{2}$ 個
- 黒マス：$\frac{N^2}{2}$ 個

> パスは白マスを全部（N²/2個）、黒マスを1つ少なく（N²/2 – 1個）訪問する。 → スキップされるマス (A,B) は**黒マス**でなければならない → **A+B は奇数**。

必要十分条件のまとめ

定理：答えが Yes となるための必要十分条件は

① **N が偶数**　かつ　② **A+B が奇数**

必要性は市松模様の議論で示した。次に十分性（実際にパスを構成できること）を示す。

判定例

N	A	B	A+B	判定
2	1	2	3（奇数）	Yes
3	2	2	4（偶数）	No
4	3	2	5（奇数）	Yes

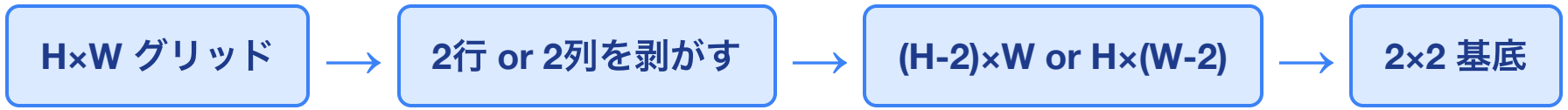
判定フロー

十分条件の証明

再帰的にパスを構成する

構成法のアイデア

条件を満たすとき、実際にパスを**再帰的に構成**する。



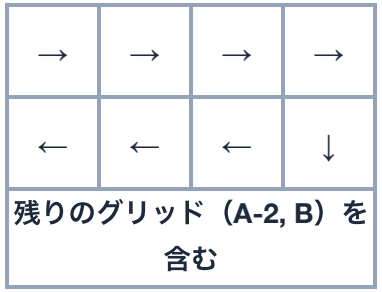
方針：

1. スキップマス (A, B) を**含まない**2行（または2列）を蛇行パターンで処理
2. グリッドを縮小して再帰
3. 2×2 まで縮小したらベースケースで解く

> 「スキップマスを含まない2行」を剥がすので、スキップマスは常に残りのグリッドに残る。

行の剥がし方（上から）

スキップマス (A, B) が上端でない（$A \geq 2$）場合、**上2行**を蛇行で処理できる。



生成される文字列：`R^(W−1)` `D` `L^(W−1)` `D`

1. 1行目を右端まで移動（R を W-1 回）
2. 2行目に降りる（D）
3. 2行目を左端まで移動（L を W-1 回）
4. 3行目に降りる（D）→ 残りのグリッドへ

帰着：$(H, W, A, B) \to (H-2, W, A-2, B)$

行の剥がし方（下から）

スキップマスが下端でない（$H - A + 1 > 2$、つまり $A \leq H - 2$）場合、**下2行**を蛇行で処理。

生成される文字列（パスの末尾に追加）：`D` `R^(W-1)` `D` `L^(W-1)`

イメージ：

- 残りグリッドの右下 $(H - 2, W)$ に到達
- 下2行を蛇行して (H, W) に到着

帰着：

$(H, W, A, B) \to (H - 2, W, A, B)$

スキップマスの座標は変わらない（上側に残るため）。

> 上から剥がすか下から剥がすかは、スキップマスの行位置 A で決まる。A ≥ 2 なら上から、H-A ≥ 2 なら下から。両方可能なときはどちらでもよい。

列の剥がし方

行と同様に、スキップマスを含まない**2列**を左右から剥がせる。

左から剥がす（B ≥ 2）

左2列を縦に蛇行で処理。

帰着：$(H, W, A, B) \to (H, W-2, A, B-2)$

右から剥がす（W-B ≥ 2）

右2列を縦に蛇行で処理。

帰着：$(H, W, A, B) \to (H, W-2, A, B)$

> 行方向の剥がしで H を 2 まで縮小 → 列方向の剥がしで W を 2 まで縮小 → **2×2** グリッドに帰着。

ベースケース：2×2 グリッド

2×2 グリッドで $A + B$ が奇数のとき、スキップマスは2通り。

(A,B) = (1,2)

パス：`DR`

$(1,1) \to (2,1) \to (2,2)$

(A,B) = (2,1)

パス：`RD`

$(1,1) \to (1,2) \to (2,2)$

> どちらのケースも1手ずつ確定で構成できる。これが再帰の終端条件。

構成の全体像

1. N が奇数 or A+B が偶数 → `No` を出力して終了
2. A ≥ 2 の間：上2行を剥がす（先頭に蛇行パターンを追加）、A -= 2, H -= 2
3. H-A ≥ 2 の間：下2行を剥がす（末尾に蛇行パターンを追加）、H -= 2
4. B ≥ 2 の間：左2列を剥がす（先頭に蛇行パターンを追加）、B -= 2, W -= 2
5. W-B ≥ 2 の間：右2列を剥がす（末尾に蛇行パターンを追加）、W -= 2
6. 2×2 ベースケース：(A,B) に応じて `DR` or `RD`

> 各ステップで H または W が 2 ずつ減る → N が偶数なので最終的にちょうど 2×2 になる。

実装

C++ コード解説

実装（C++）

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int T; cin >> T;
    while (T--) {
        int N, A, B; cin >> N >> A >> B;
        A--; B--;  // 0-indexed に変換
        if (N % 2 == 1 || (A + B) % 2 == 0) { cout << "No\n"; continue; }
        cout << "Yes\n";
        string s1, s2;  // s1: 先頭に追加, s2: 末尾に追加
        int H = N, W = N;
        // 行方向の剥がし
        while (A >= 2) {  // 上2行を剥がす
            s1 += string(W-1,'R') + 'D' + string(W-1,'L') + 'D';
            A -= 2; H -= 2;
        }
        while (H - A > 2) {  // 下2行を剥がす
            s2 = string(1,'D') + string(W-1,'R') + 'D' + string(W-1,'L') + s2;
            H -= 2;
        }
        // 列方向の剥がし（H=2 になっている）
        while (B >= 2) {  // 左2列を剥がす
            s1 += string(H-1,'D') + 'R' + string(H-1,'U') + 'R';
            B -= 2; W -= 2;
        }
        while (W - B > 2) {  // 右2列を剥がす
            s2 = string(1,'R') + string(H-1,'D') + 'R' + string(H-1,'U') + s2;
            W -= 2;
        }
        // ベースケース（2x2）
        string base = (A == 0 && B == 1) ? "DR" : "RD";
        cout << s1 + base + s2 << "\n";
    }
}
```

実装のポイント

0-indexed 化

`A--; B--;` で 0-indexed に変換。

- 条件判定が `A >= 2`（元の1-indexed で $A \geq 3$ ）
- ベースケースが `(0,1)` or `(1,0)`

s1 と s2 の役割

- `s1`：先頭に追加（上・左の剥がし）
- `s2`：末尾に追加（下・右の剥がし）
- 最終的に `s1 + base + s2` を連結

蛇行パターンの方向

剥がし方向	移動パターン
上2行	`R^(W-1) D L^(W-1) D`
下2行	`D R^(W-1) D L^(W-1)`
左2列	`D^(H-1) R U^(H-1) R`
右2列	`R D^(H-1) R U^(H-1)`

行の剥がしでは横移動（R/L）、列の剥がしでは縦移動（D/U）が主体。

計算量

処理	計算量
判定（偶奇チェック）	$O(1)$
パス文字列の構築	$O(N^2)$（全マス分の文字を生成）
1テストケース	$O(N^2)$
全テストケース	$O(\sum N^2) \leq O(10^6)$

制約「全テストケースの N^2 の総和 $\leq 10^6$」により、全体でも線形時間相当で収まる。

⚠ 文字列の先頭への追加（`s2 = ... + s2`）はコストが高い。実装では末尾追加で逆順に構築し、最後に反転する方が効率的。ただし制約が $N^2 \leq 10^6$ なので素朴な実装でも十分間に合う。

まとめ

Key Takeaways

- $N \times N$ グリッドで1マスをスキップするハミルトンパスの存在判定＋構成問題
- **市松模様**の着色で **N が偶数** かつ **A+B が奇数** が必要条件と分かる
- **再帰的構成法**：スキップマスを含まない2行/2列を蛇行パターンで剥がし、2×2 に帰着
- 必要条件がそのまま十分条件 → **判定は O(1)、構成は O(N²)**
- 全体の計算量は $O(\sum N^2) = O(10^6)$

出典：AtCoder ABC454 E 解説 https://atcoder.jp/contests/abc454/editorial/19007