# 2日目演習③解説

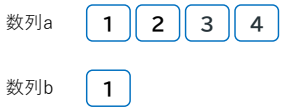
## pushpush 解説 1/9

まず素朴な実装を考えてやってみる

- 数列なのでvectorが使えそう
- vectorにはreverseという反転させる機能があった

**目標**

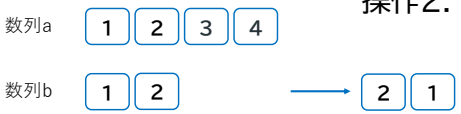

**操作1回目**




操作1: $a_1$(今回は1)をbの末尾に追加
操作2: bを反転する（今回は数字がひとつなので変化無し）

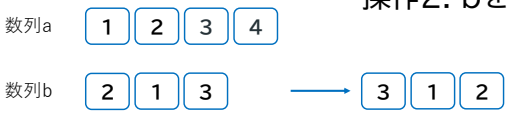
## pushpush 解説 2/9

**操作2回目**




操作1: $a_2$(今回は2)をbの末尾に追加
操作2: bを反転する

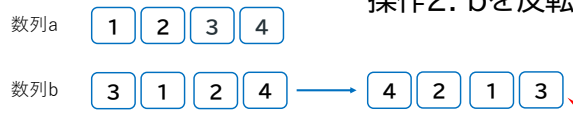
**操作3回目**




操作1: $a_3$(今回は3)をbの末尾に追加
操作2: bを反転する

## pushpush 解説 3/9

**操作4回目**

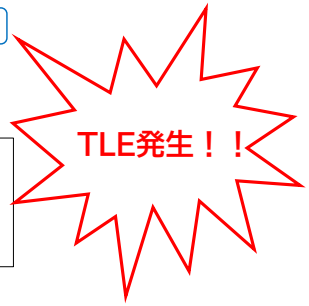



操作1: $a_4$(今回は4)をbの末尾に追加
操作2: bを反転する

この操作に該当する部分のコードはこんな感じ…

```
for(int i = 0; i<a; i++){
    data.push_back(num.at(i));
    reverse(data.begin(), data.end());
}
```

http://atcoder.jp/contests/abc066/submissions/56911519より



# pushpush 解説 4/9

TLE発生はきっとreverseに時間かかっているに違いない！
Reverseを使わない方法を考えてみよう！

| | | |
|---|---|---|
| 入力データ | 数列a | 1 2 3 4 |
| 操作1回目 | 数列b | 1 |
| 操作2回目 | 数列b | 2 1 |
| 操作3回目 | 数列b | 3 1 2 |
| 操作4回目 | 数列b | 4 2 1 3 |

Reverseを使わなくてもつじつまが
合うような挿入のパターンはないか？

# pushpush 解説 5/9

TLE発生はきっとreverseに時間かかっているに違いない！
Reverseを使わない方法を考えてみよう！

| | | |
|---|---|---|
| 入力データ | 数列a | 1 2 3 4 |
| 1回目 | 数列b | 1 |
| 2回目 | 数列b | 2 1 |
| 3回目 | 数列b | 2 1 3 |
| 4回目 | 数列b | 4 2 1 3 |

方針その1:
例えば, 奇数番目は後ろ,
偶数番目は前, という法則で
データを入れていると見てみる

# pushpush 解説 6/9

奇数個の入力例でも成り立つかチェック！

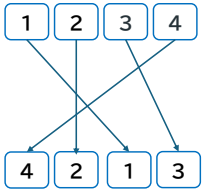
| | | | | |
|---|---|---|---|---|
| 入力データ | 数列a | 1 2 3 | 目標 | 3 1 2 |
| 操作1回目 | 数列b | 1 | | |
| 操作2回目 | 数列b | 2 1 | | |
| 操作3回目 | 数列b | 2 1 3 | | |

操作3回目を逆順にすれば目標の出力になる
→ 最後の結果だけ逆順で出力する！

# pushpush 解説 7/9

方針その2:
入力データと目標となる出力でデータの位置を比較してみる

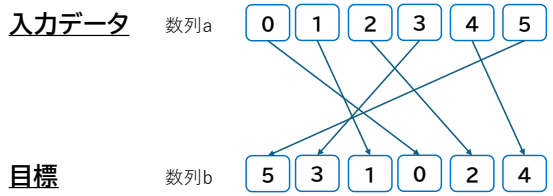
| | | |
|---|---|---|
| 入力データ | 数列a | 1 2 3 4 |
| 目標 | 数列b | 4 2 1 3 |



矢印を整理すると…
- $a_1$が$b_3$へ
- $a_2$が$b_2$へ
- $a_3$が$b_4$へ
- $a_4$が$b_1$へ

# pushpush 解説 8/9

方針その2:
入力データと目標となる出力でデータの位置を比較してみる




矢印を整理すると…
- $a_1$が$b_4$へ
- $a_2$が$b_3$へ
- $a_3$が$b_5$へ
- $a_4$が$b_2$へ
- $a_5$が$b_6$へ
- $a_6$が$b_1$へ

一般化してみると,
偶数番目→逆順で前から追加, 奇数番目→偶数番目の後に前から追加

# pushpush 解説 9/9

奇数個の数列で確認:
偶数番目→逆順で前から追加, 奇数番目→偶数番目の後に前から追加をしてみると…

入力データ 数列a 1 2 3　　目標 3 1 2

操作後 数列b 2 1 3

今回も奇数個の数列では操作後が目標と逆なので, 最後に逆順へ変換してから出力！

# Unification 解説 1/4

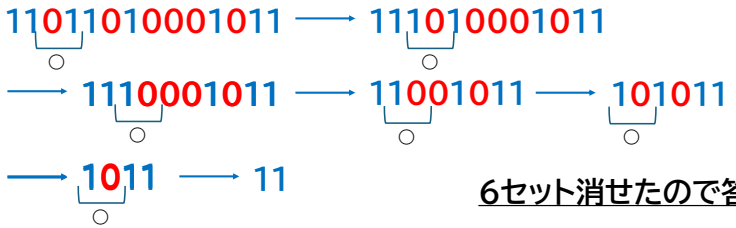
これもまずは素朴な方法を考えてみる
下から順に見ていき, 0と1のペアが出来たら消し, 消えたらまた下からチェックし直せばいけそう
(※ 縦書きは大変なので問題と同様に左が下, 右が上とする)


**4つ全て消せたので答えは4個**

# Unification 解説 2/4

これもまずは素朴な方法を考えてみる
下から順に見ていき, 0と1のペアが出来たら消し, 消えたらまた下からチェックし直せばいけそう
(※ 縦書きは大変なので問題と同様に左が下, 右が上とする)

長いので×の部分は省略




**6セット消せたので答えは12個**

# Unification 解説 3/4

実装方針その1:

・配列に入れた0/1を右から捜査する

・0と1が隣合う部分が出たら…

- 該当部分を配列から削除する（上から下に落ちる動作に該当）
- 消した数のカウントを2増やす

・これが消せなくなるまで繰り返す

（配列の最後に到達したときペアがこれ以上ないはずである）

# Unification 解説 4/4

実装方針その2:

手で書いてみると気づくと思うが, この問題の場合, **0/1ペアがあった場合は消えて, 上から下に詰まっていくため, ペアの個数が消えるセットの個数**であることがわかる.

そのため, 0と1の数を数え, その数の合計から大きい方-小さい方の差を引けばよい.（大きい方-小さい方の差は余る個数）

0011なら0が2個, 1が2個, 差は0:2+2-0 = **4**

11011010001011なら0が6個, 1が8個, 差が2個:
6+8-2 = 14-2 = **12**

（よくよく考えたら, 小さい方の数 × 2でいけそうですね）

# 最高のピザ 解説 1/2

このピザ屋のトッピングの特徴:

- トッピングの**値段はどれも同じ**
- トッピングに**よってカロリーが違う**
- **同種のトッピングはできない**

→ **トッピング無しから一つずつフルトッピングに向かって追加していく**

追加方針:

✓カロリーが低い順トッピングをつけると1ドルあたりのカロリーが下がるので, 高い順に載せる

# 最高のピザ 解説 2/2

実装方針:

① トッピングの値段を高い順でソートする

② トッピングを一つずつ加え, カロリーあたりの単価を求める

③ もし, これまでの単価より大きければカロリーあたりの単価の最大値を更新する

④ ②, ③を乗せるトッピングがなくなるまで繰り返す