PUSH_SWAP で考える
データ構造

push_swap
ってどんな課題？

push_swap を 理解するためのスライド

各種命令

今回考えたいデータ構造

1. 配列
2. 単方向リスト
3. 双方向リスト

結論

どれでも実装できます 😎

この課題においては
処理の速さ　＜　命令の少なさ

影響度

	処理速度	命令数
データ構造	◎	△
アルゴリズム	○	◎

高得点を目指す(命令数を少なくする)上で
データ構造はあまり影響しません...

今回は
「処理速度を意識したベストなデータ構造は何か」
という観点で考えたい

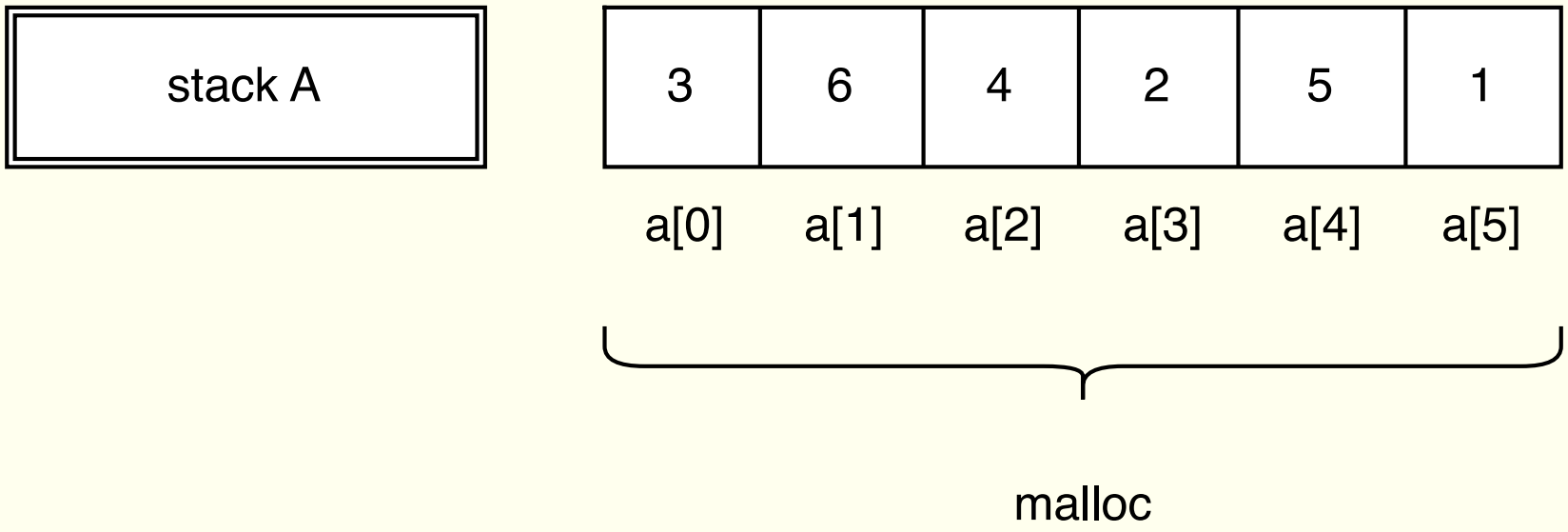
前提

処理が速い　＝　LOOP が少ない

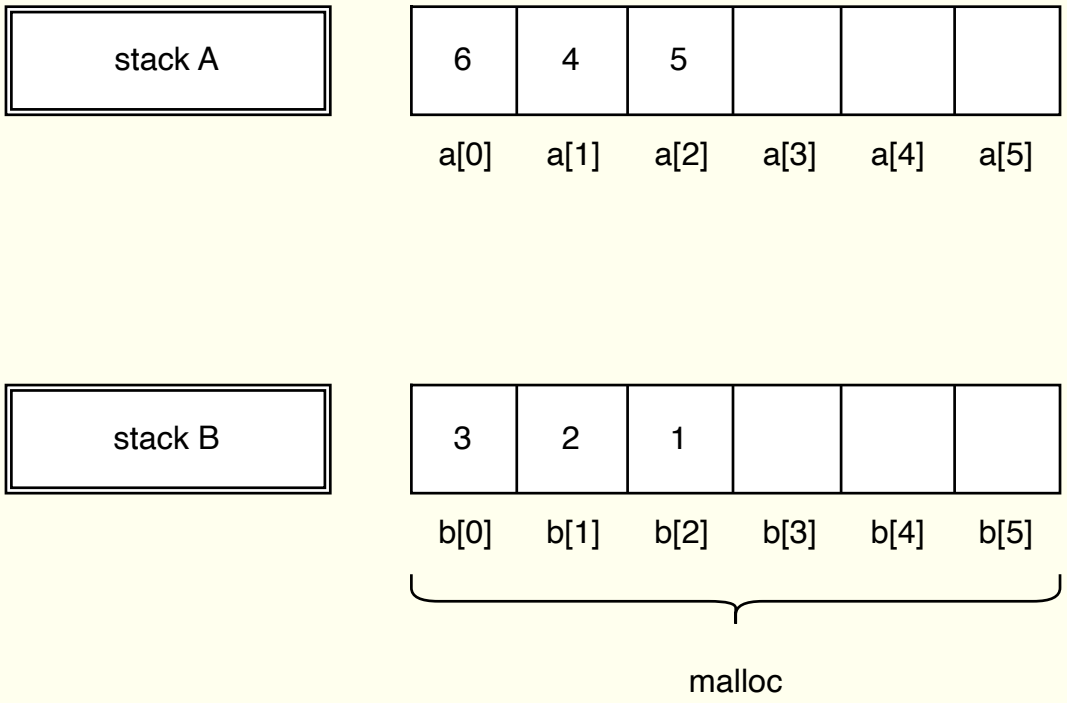
1️⃣ 配列

配列の実装イメージ

./push_swap 3 6 4 2 5 1



配列の実装イメージ

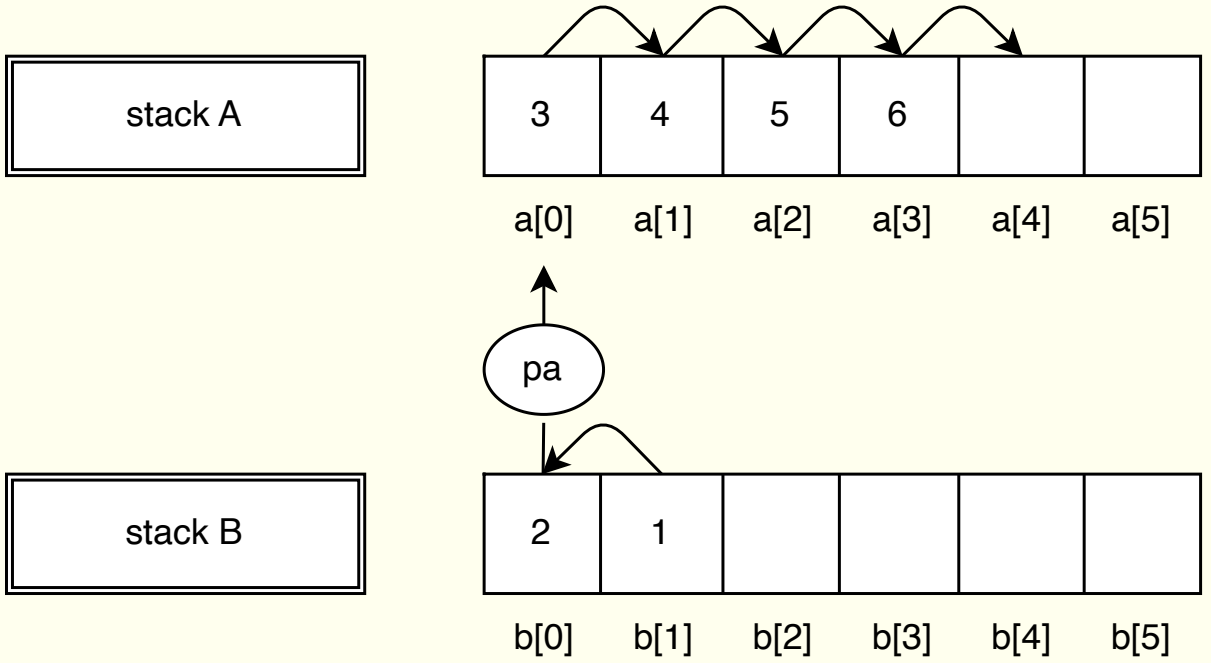


必要に応じてスタックBの領域を確保

配列のマイナス点

push / rotate 操作で
要素を全移動させる必要がある

push 操作の例



push前に
stack A内の要素を
一つ後ろに移動させる

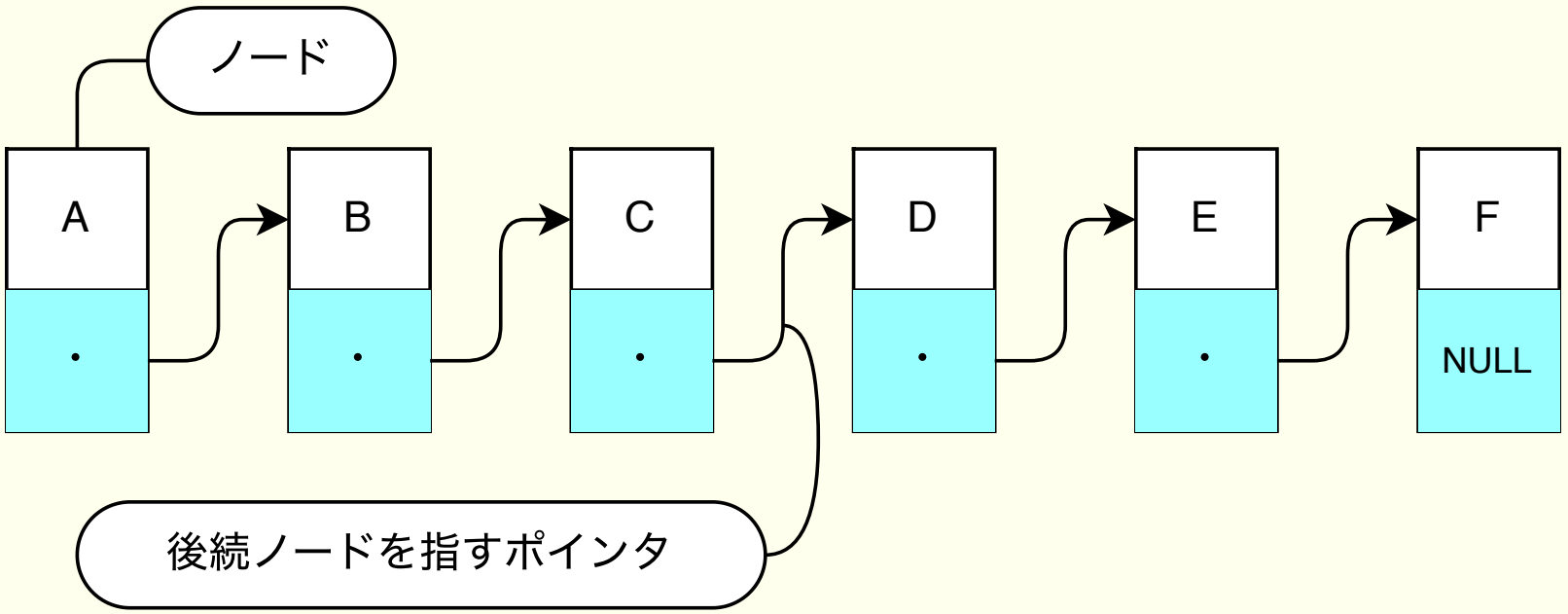
push後に
stack B内の要素を
一つ前に移動させる

配列の場合
push / rotate 操作で
LOOP が発生する！

2️⃣ 単方向リスト

単方向リストとは

先頭から末尾まで要素が数珠繋ぎに並んでいる



実装イメージ

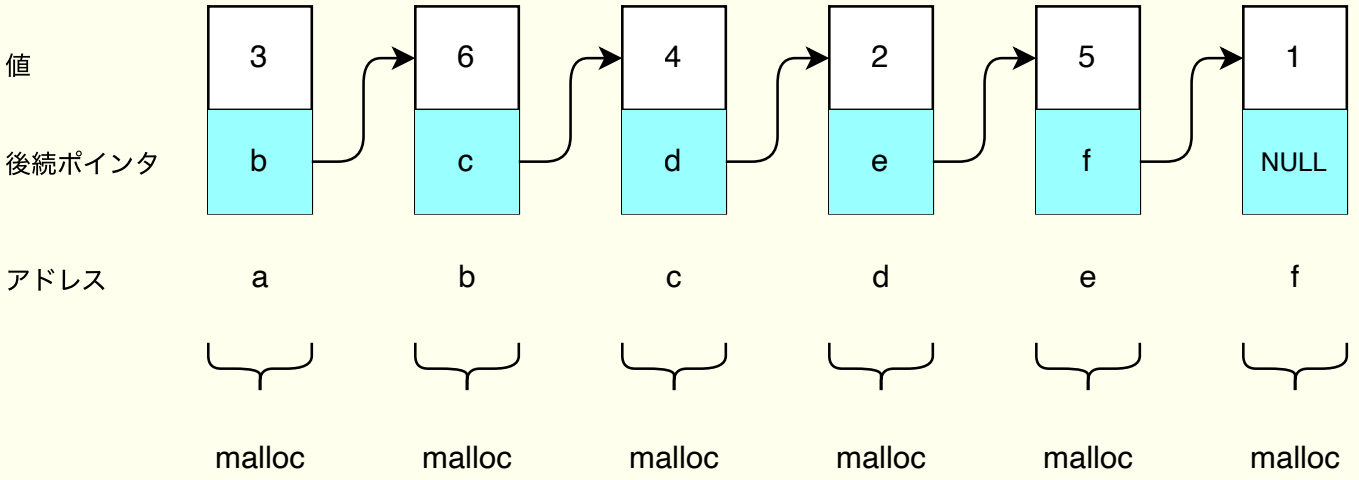
t_list

head	先頭ノードのポインタ

t_node

num	ソート対象の値
next	後続ノードのポインタ

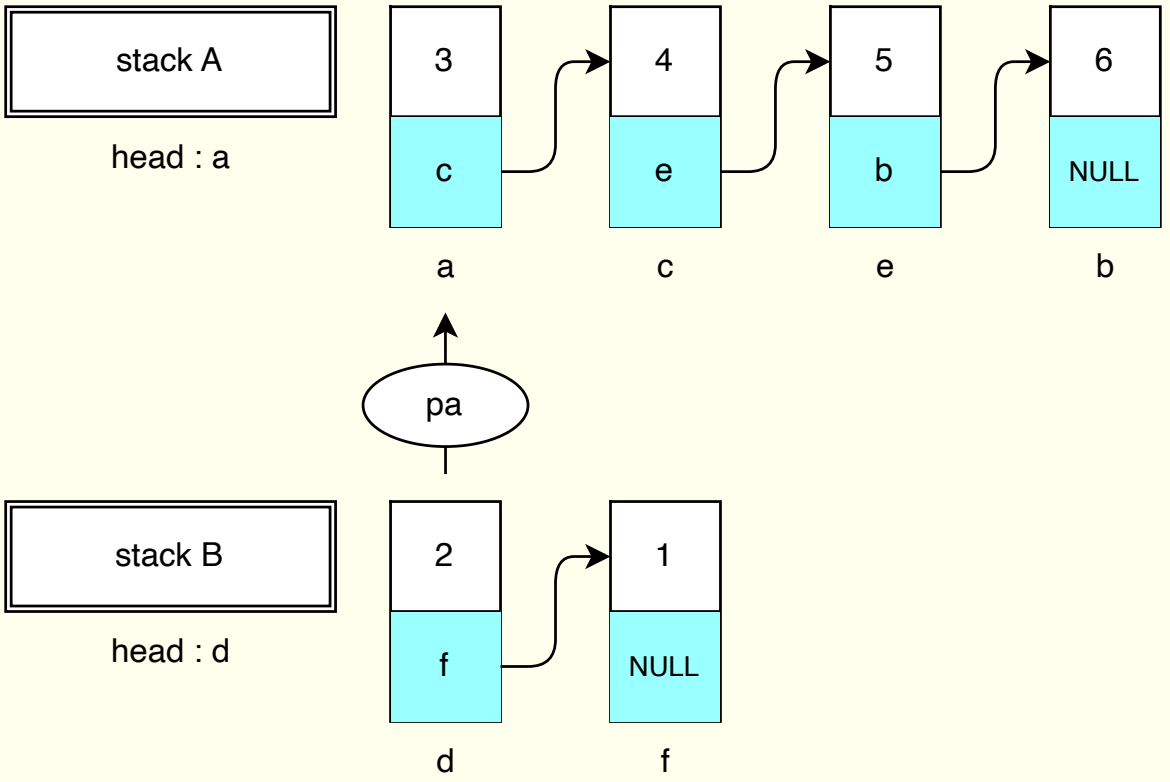
./push_swap 3 6 4 2 5 1



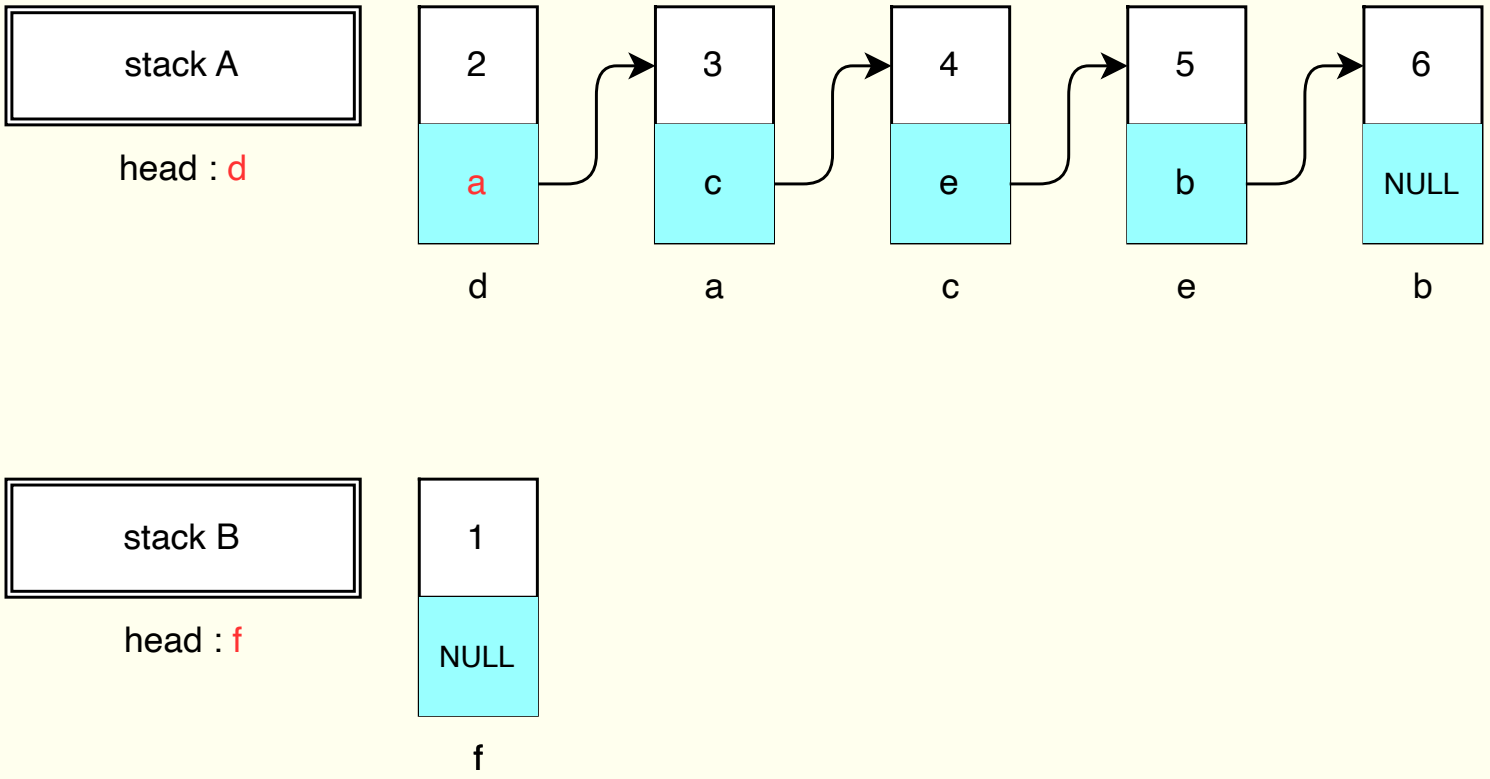
配列と比較して

リストでは値の移動は発生しない

push 前



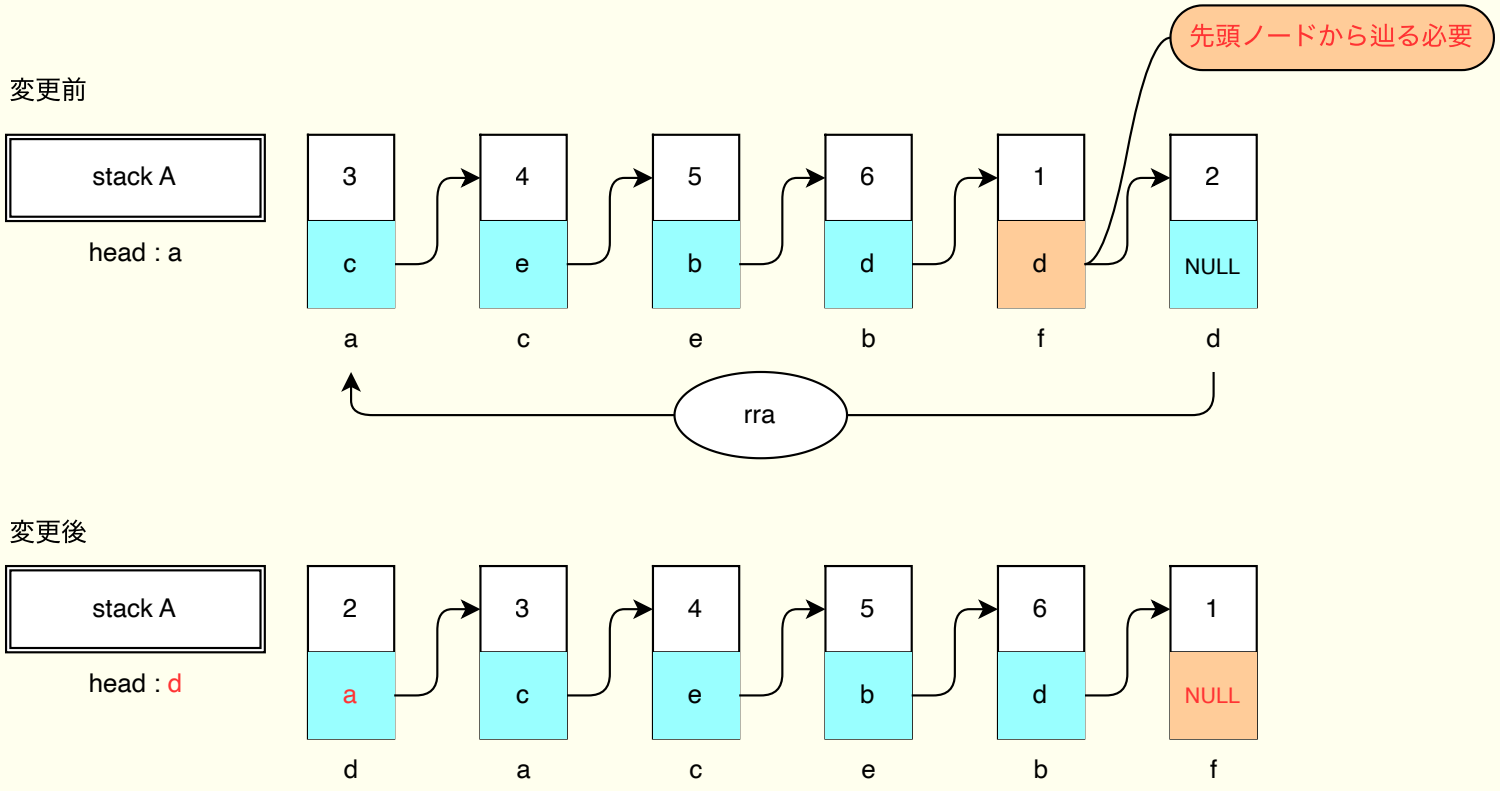
push 後



単方向リストの問題点

rotate 操作で
最後のノード取得するのに next で辿る必要がある

rotate 操作


変更前
stack A
head : a
3
c
a
4
e
c
5
b
e
6
d
b
1
d
f
2
NULL
d
先頭ノードから辿る必要
rra
変更後
stack A
head : d
2
a
d
3
c
a
4
e
c
5
b
e
6
d
b
1
NULL
f

単方向リストの場合
rotate 操作で
LOOP が発生する！

配列とリスト
それぞれのマイナスポイント

配列

挿入時に全移動が発生

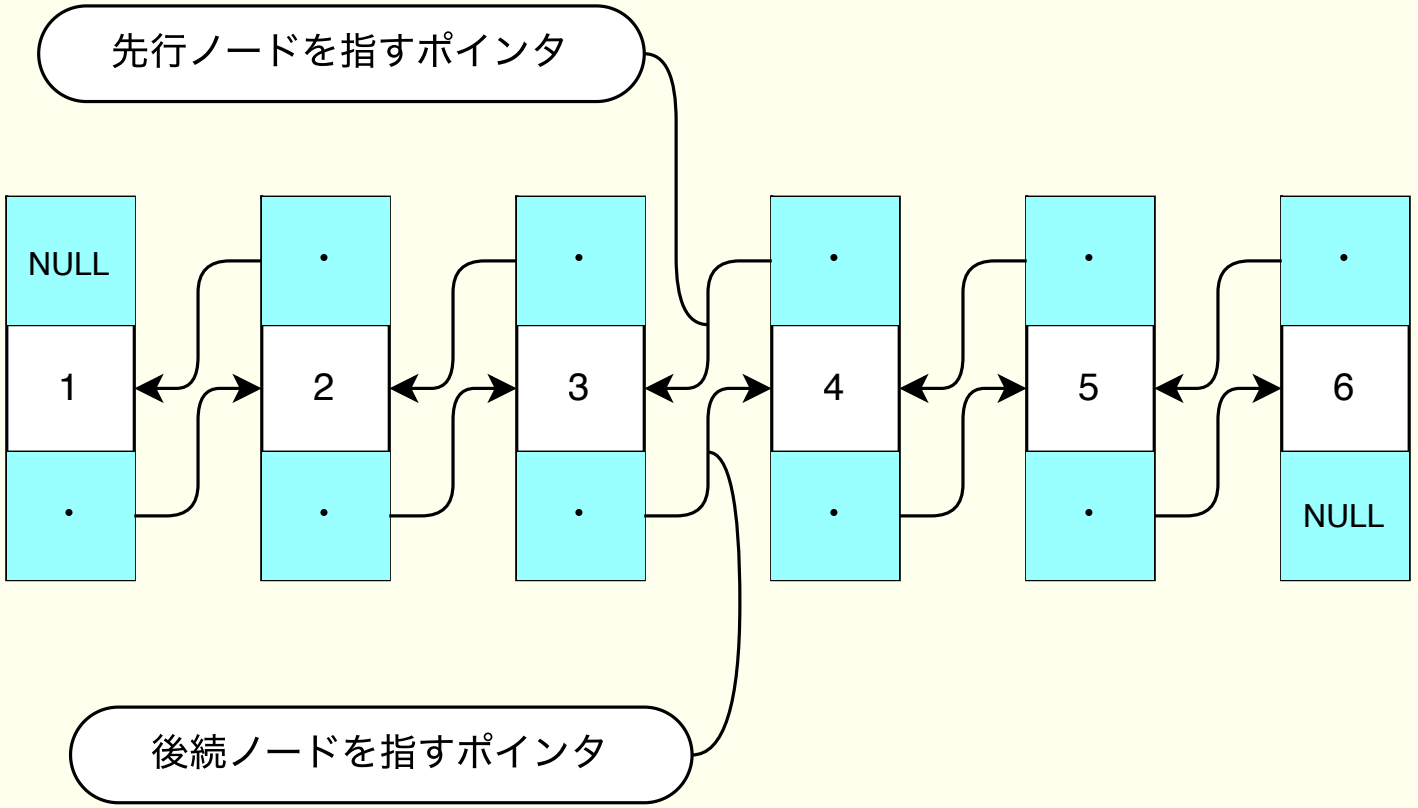
リスト

要素へのアクセスが遅い

3️⃣ 双方向リスト

双方向リストとは

単方向リストに先行ノードのポインタを加えたもの



実装イメージ

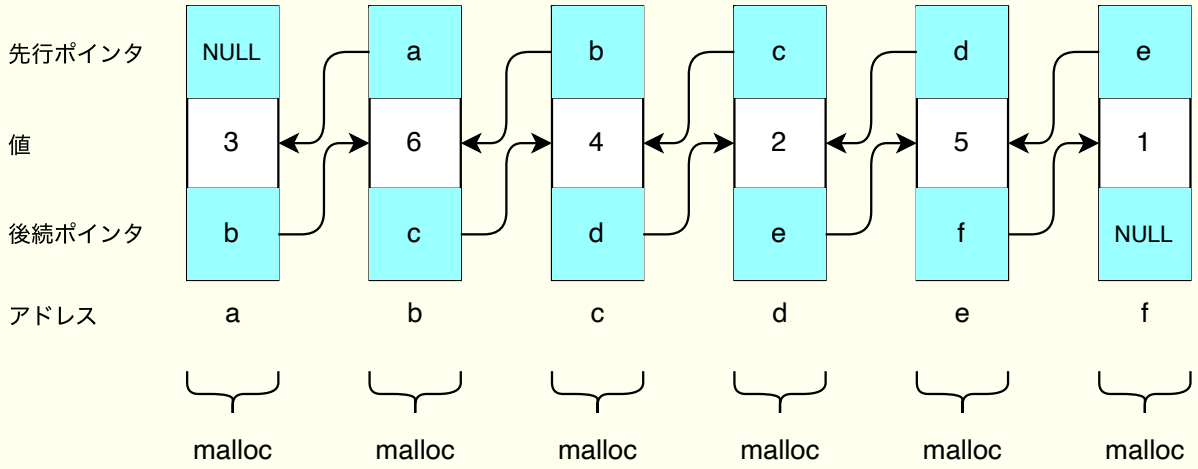
t_list

head	先頭ノードのポインタ
tail	末尾ノードのポインタ

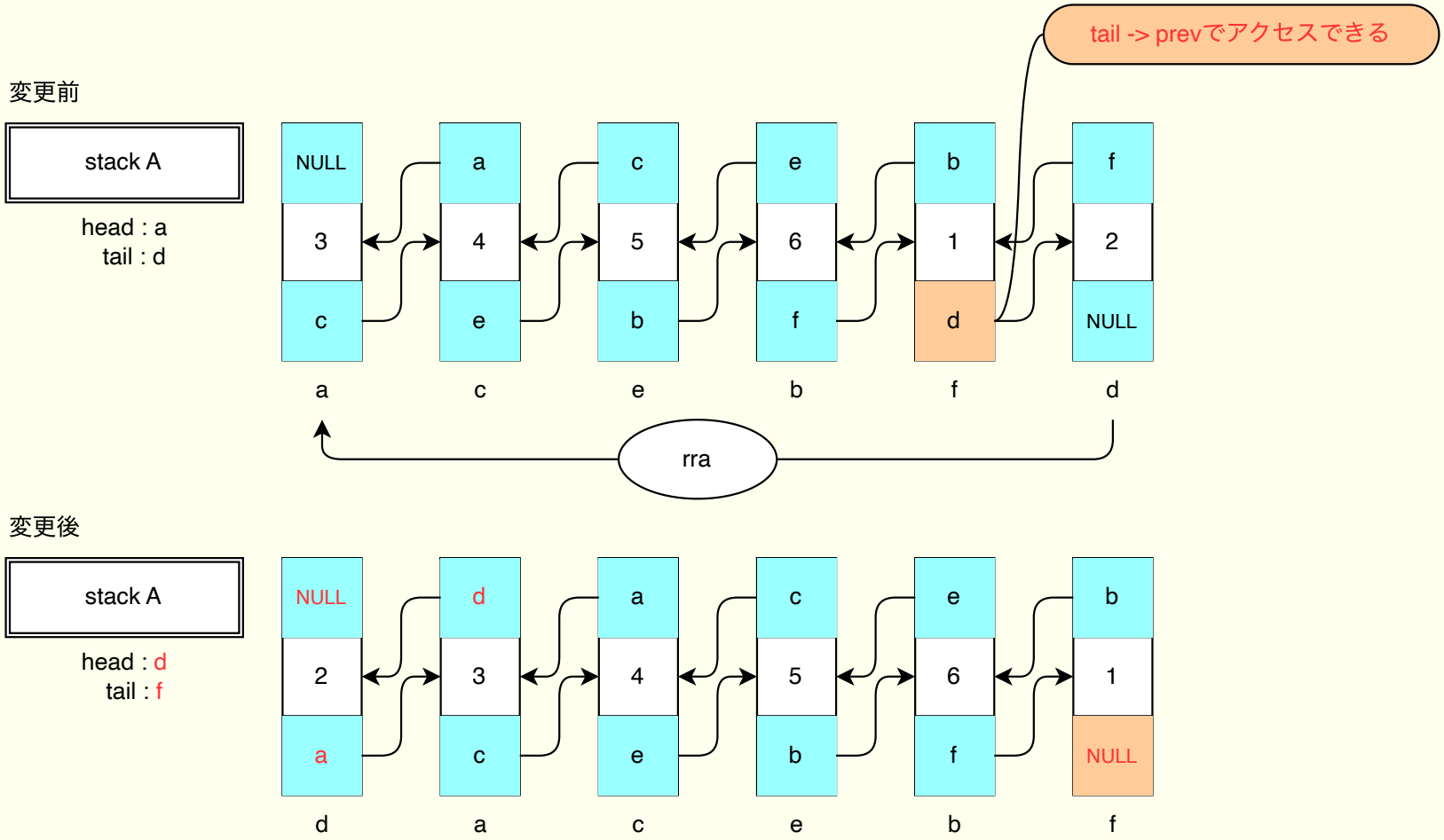
t_node

num	ソート対象の値
next	後続ノードのポインタ
prev	先行ノードのポインタ

./push_swap 3 6 4 2 5 1



rotate 操作


変更前
stack A
head : a
tail : d
tail -> prevでアクセスできる
NULL
a
c
e
b
f
3
4
5
6
1
2
c
e
b
f
d
NULL
a
c
e
b
f
d
rra
変更後
stack A
head : d
tail : f
NULL
d
a
c
e
b
2
3
4
5
6
1
a
c
e
b
f
NULL
d
a
c
e
b
f

双方向リストだと命令処理で
LOOP が発生しない！

時間があればコードをお見せします

結論

(この課題においてはデータ構造は
あまり意識しなくても良いけど...)
"回転するスタック領域"では双方向リストがベスト

補足・ご指摘あればお願いします！

ありがとうございました😊