

計算量のはなし2

～DequeはStack2つでつくれる～

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method

有名なことわざ

DequeはStack2つでつくれる

Deque

- 次の操作に対応したデータ構造
 - 列の先頭に値を追加 (pushF)
 - 列の末尾に値を追加 (pushB)
 - 列の先頭から値を取り出す (popF)
 - 列の末尾から値を取り出す (popB)

Deque: 図示

- ここに空のDequeがあるじゃろ
 - 左が先頭(Front), 右が末尾 (Back)

Deque: 図示

- pushF(3)



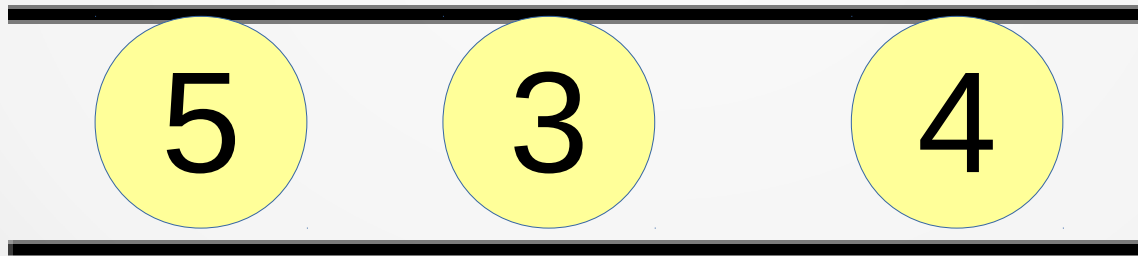
Deque: 図示

- pushF(3), pushB(4)



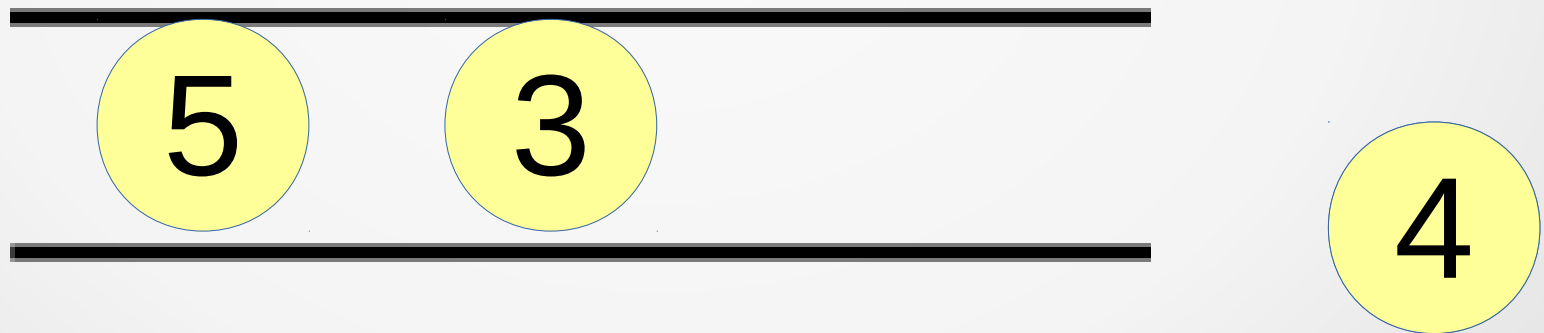
Deque: 図示

- pushF(3), pushB(4), pushF(5)



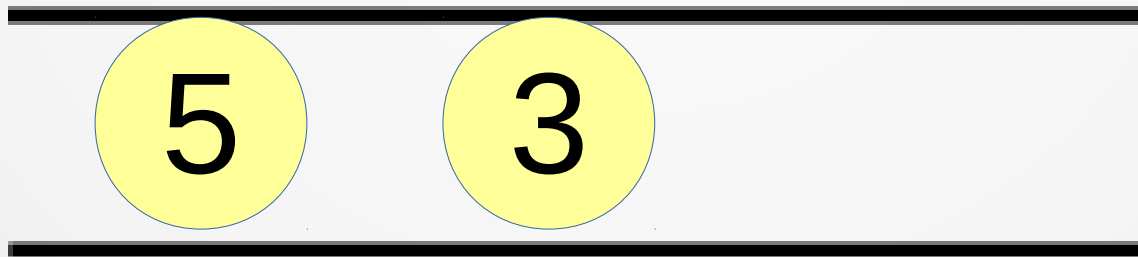
Deque: 図示

- `pushF(3)`, `pushB(4)`, `pushF(5)`, `popB()`



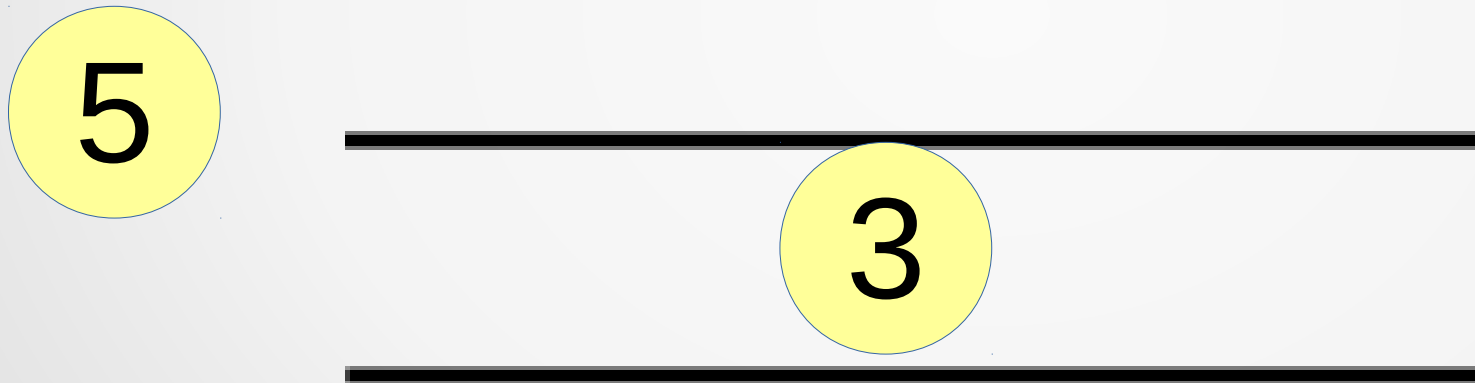
Deque: 図示

- `pushF(3)`, `pushB(4)`, `pushF(5)`, `popB()`



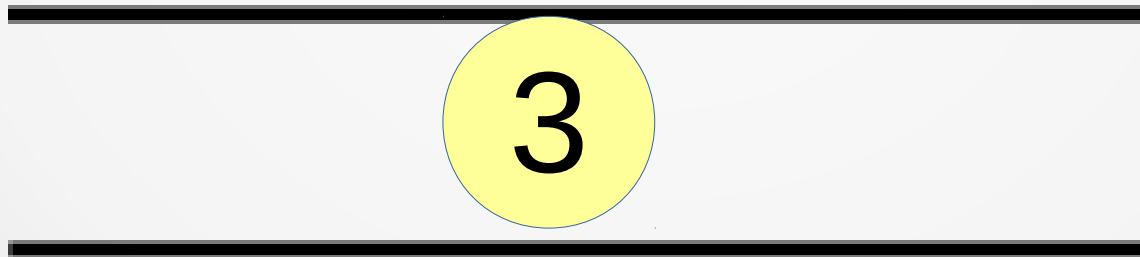
Deque: 図示

- pushF(3), pushB(4), pushF(5), popB(), popF()



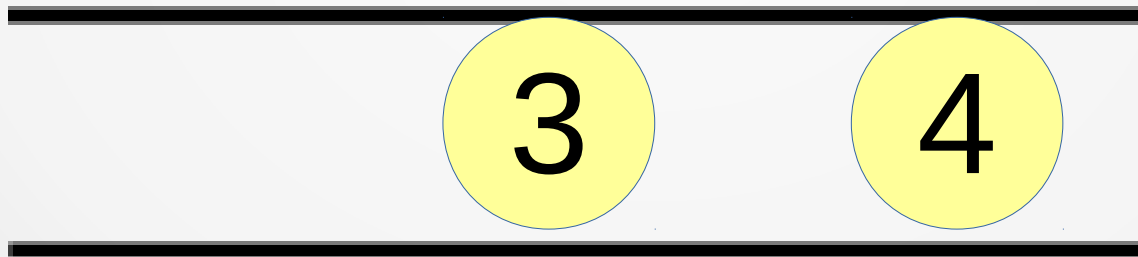
Deque: 図示

- pushF(3), pushB(4), pushF(5), popB(), popF()



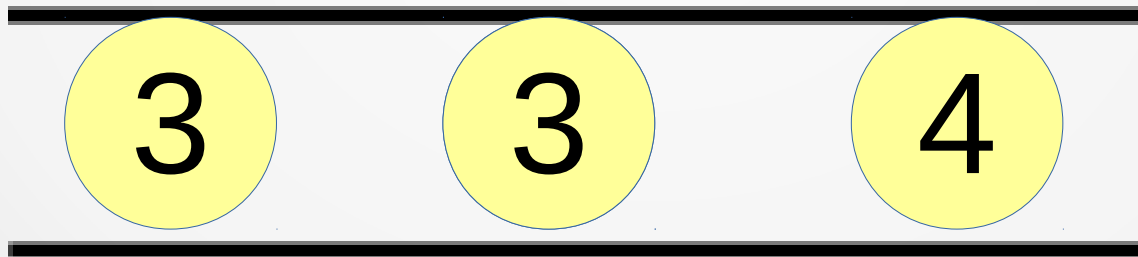
Deque: 図示

- pushF(3), pushB(4), pushF(5), popB(), popF()
pushB(4)



Deque: 図示

- pushF(3), pushB(4), pushF(5), popB(), popF()
pushB(4), pushF(3)



有名なことわざ(再掲)

DequeはStack2つでつくれる

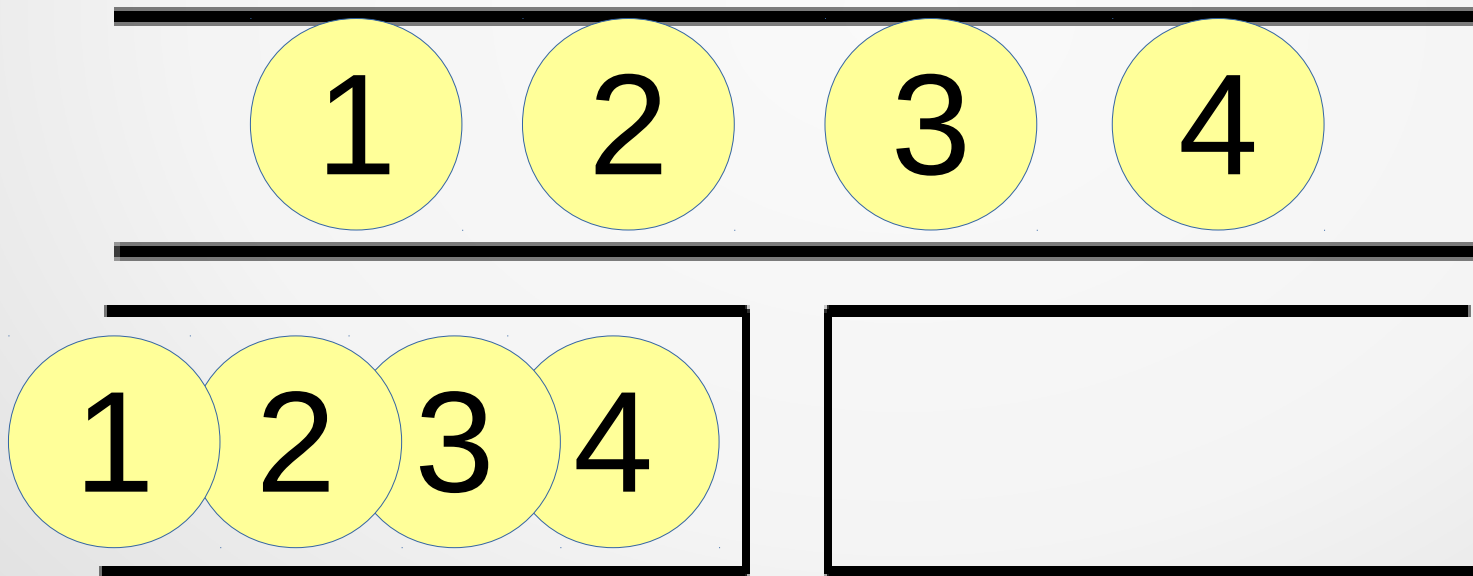
Deque (Stack × 2): 図示

- 空のDequeの下にStackが2つあるじゃろ
 - 左が先頭(Front), 右が末尾 (Back)



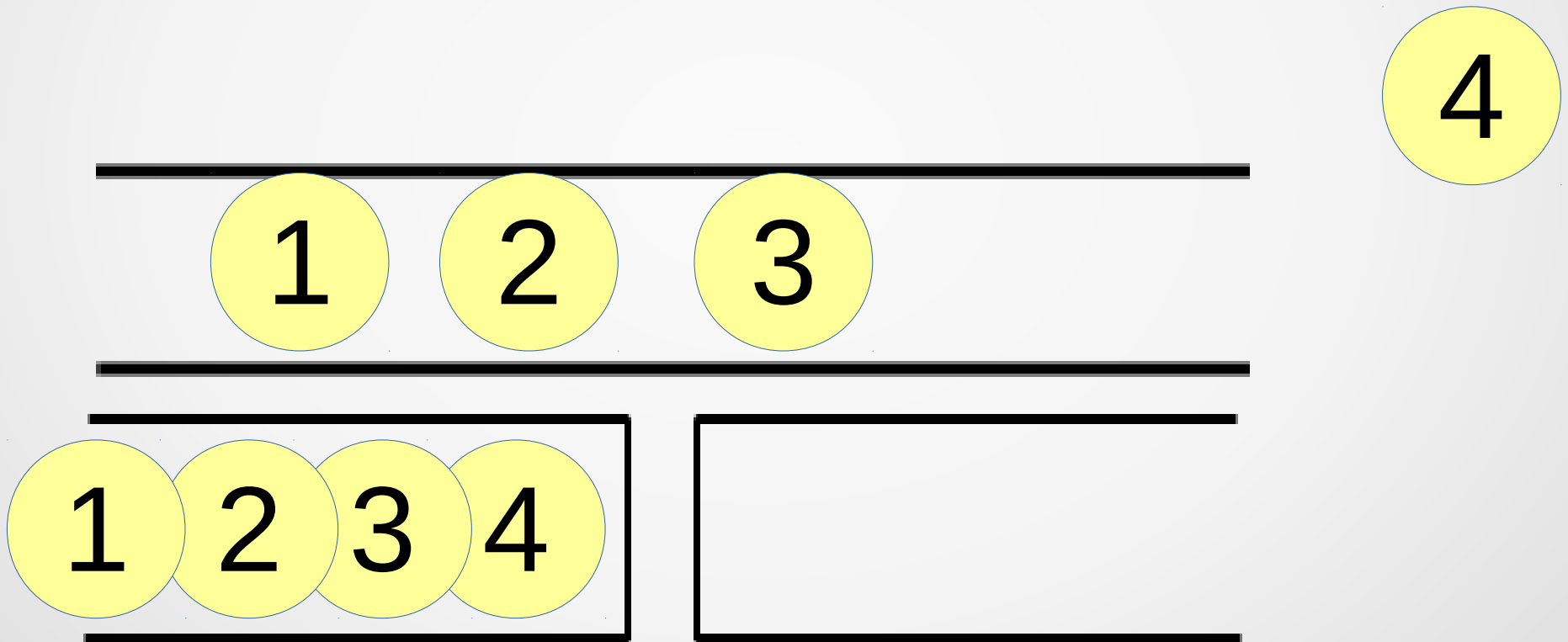
Deque (Stack \times 2): 図示

- pushF(1), pushF(2), pushF(3), pushF(4)



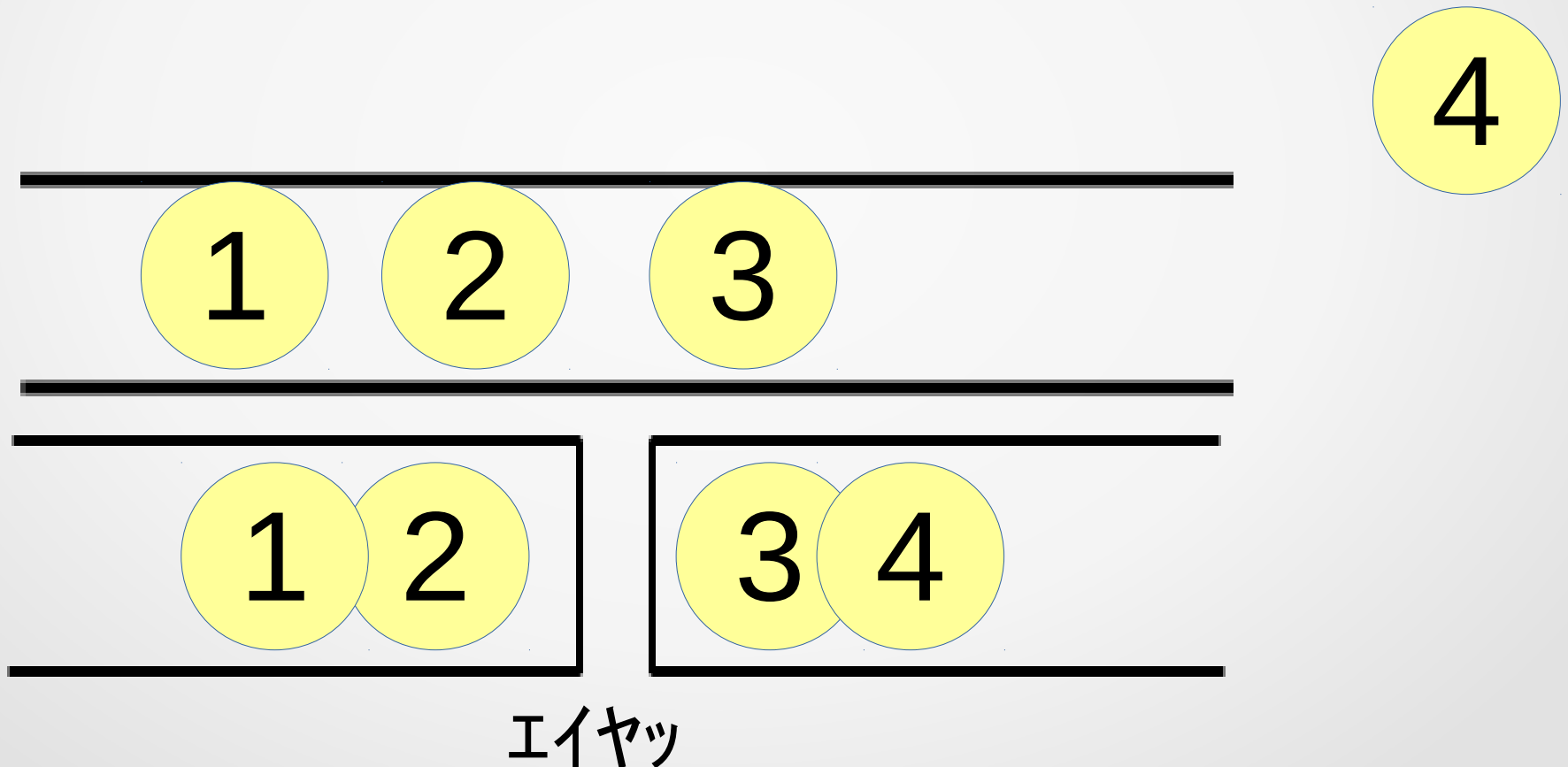
Deque (Stack \times 2): 図示

- pushF(1), pushF(2), pushF(3), pushF(4)
popB()



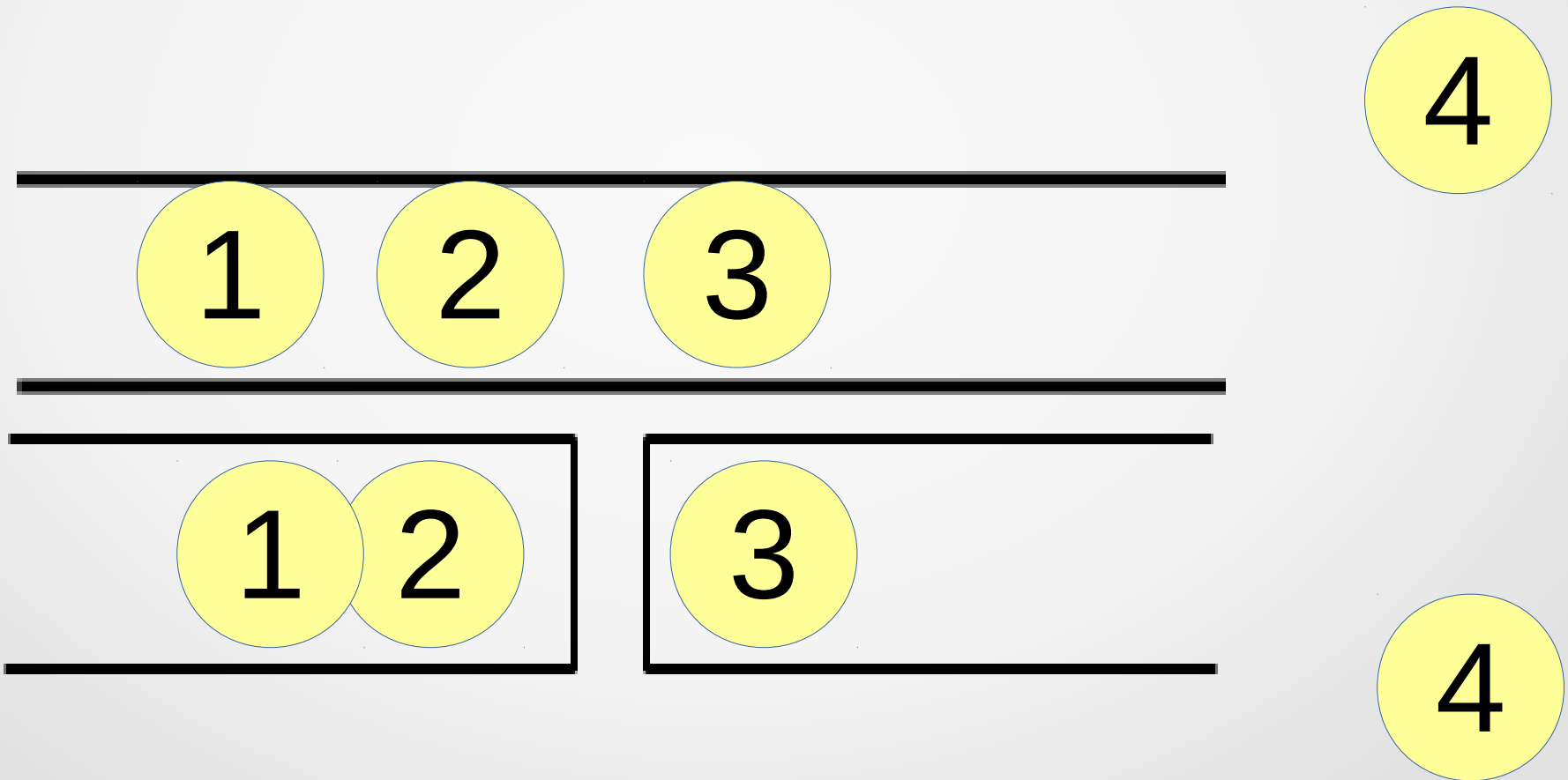
Deque (Stack × 2): 図示

- pushF(1), pushF(2), pushF(3), pushF(4)
popB()



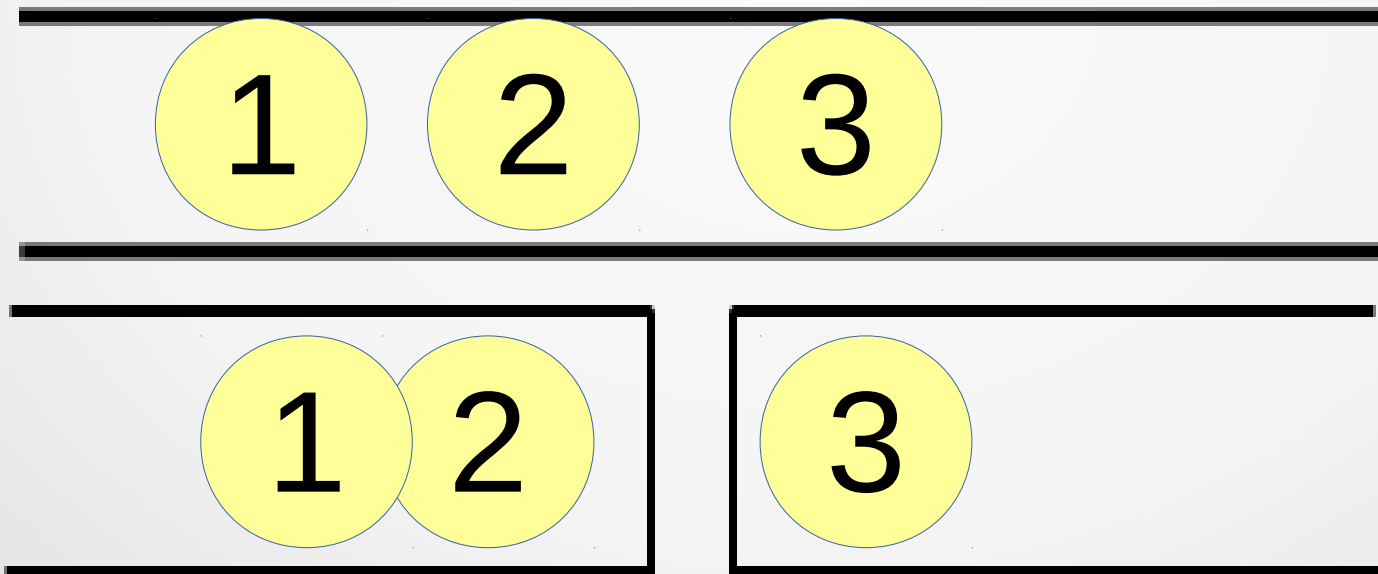
Deque (Stack × 2): 図示

- pushF(1), pushF(2), pushF(3), pushF(4)
popB()



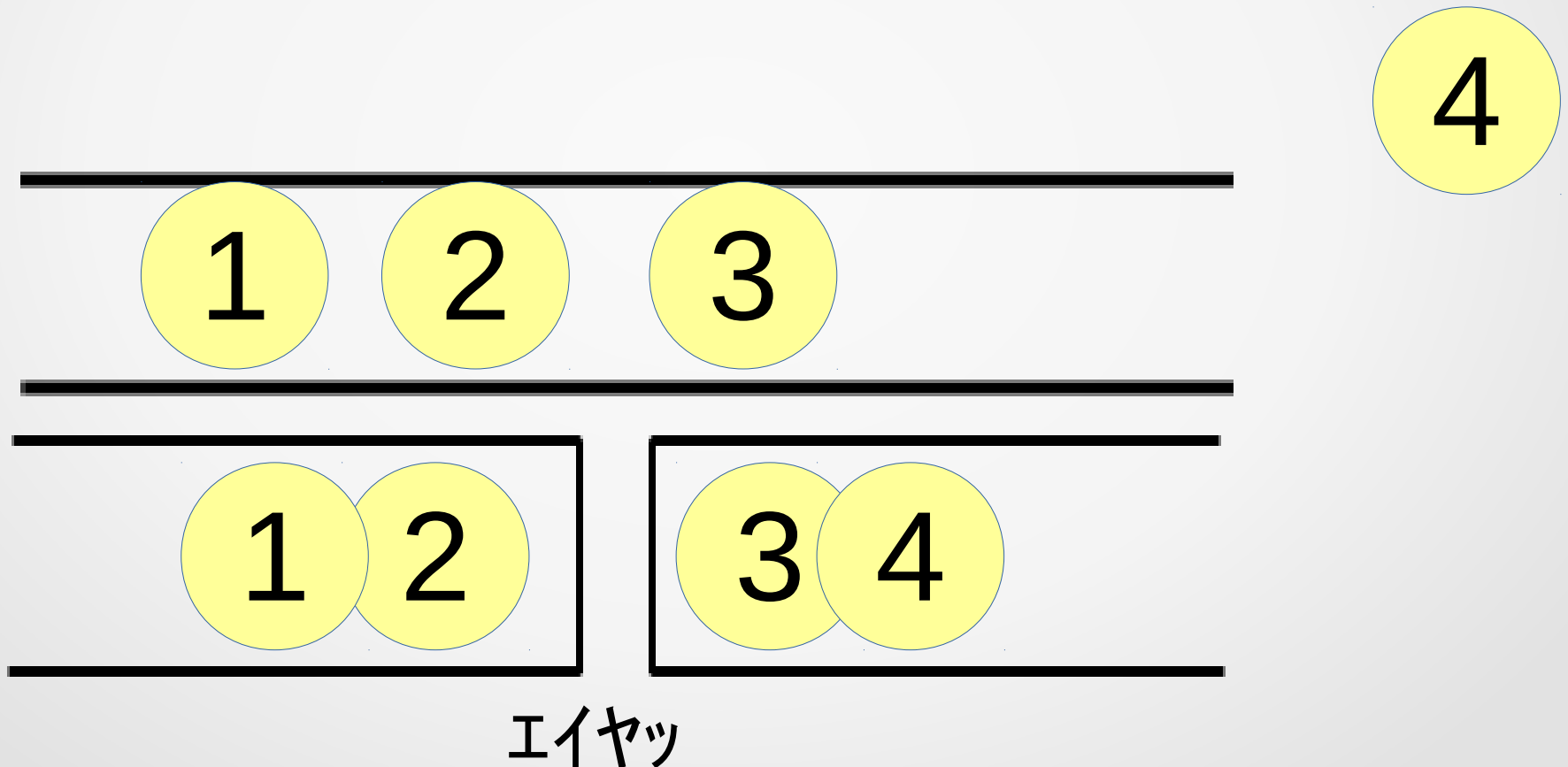
Deque (Stack × 2): 図示

- pushF(1), pushF(2), pushF(3), pushF(4)
popB()



Deque (Stack \times 2): 大事な所

- PopしたいのにStackがからの時、もう片方のStackの半分を移してくる



フハハ

Dequeはこのようなして
Stack2つでつくれるのじゃ

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method

半分うつすのやばそう

- Stack2つで実装したDequeの計算量を評価してみよう

操作一覽

- pushBack
- pushFront
- popBack
- popFront

操作一覧

- pushBack どうせ $O(1)$
- pushFront どうせ $O(1)$
- popBack
- popFront

操作一覧

- pushBack どうせ $O(1)$
- pushFront どうせ $O(1)$
- popBack だいたい $O(1)$
- popFront だいたい $O(1)$

半分移すとき時間かかる

- pushBack どうせ $O(1)$
- pushFront どうせ $O(1)$
- popBack だいたい $O(1)$ ときどき $O(N)$
- popFront だいたい $O(1)$ ときどき $O(N)$

半分移すとき時間かかる

- pushBack どうせ $O(1)$
 - pushFront どうせ $O(1)$
 - popBack だいたい $O(1)$ ときどき $O(N)$
 - popFront だいたい $O(1)$ ときどき $O(N)$
-
- でも $O(N)$ であることは本当に「ときどき」で、まれにしか起こらない

半分移すとき時間かかる

- pushBack どうせ $O(1)$
 - pushFront どうせ $O(1)$
 - popBack だいたい $O(1)$ ときどき $O(N)$
 - popFront だいたい $O(1)$ ときどき $O(N)$
-
- でも $O(N)$ であることは本当に「ときどき」で、まれにしか起こらない

半分移すとき時間かかる

- pushBack どうせ $O(1)$
- pushFront どうせ $O(1)$
- popBack だいたい $O(1)$ ときどき $O(N)$
- popFront だいたい $O(1)$ ときどき $O(N)$

- でも $O(N)$ であることは本当に「ときどき」で、まれにしか起こらない
- $O(N)$ っていいたくない

そんなときに便利なのが

ならし計算量

ならし計算量とは

- データ構造に対する操作の計算量の評価で使われる「計算量もどき」
- 時と場合によって計算量が変わるが、均すと $O(f(x))$ になるとき

「ならし計算量が $O(f(x))$ である」

という

ならし計算量とは

- データ構造に対する操作の計算量の評価で使われる「計算量もどき」
- 時と場合によって計算量が変わるが、均すと $O(f(x))$ になるとき

「ならし計算量が $O(f(x))$ である」

という

ならし計算量とは

- データ構造に対する操作の計算量の評価で使われる「計算量もどき」
- 時と場合によって計算量が変わるが、均すと $O(f(x))$ になるとき
?

「ならし計算量が $O(f(x))$ である」

という

「均す」とは

- データ構造の中で、操作をN回やった時、「実際の計算量」の総和と「ならし計算量」の総和がだいたい同じになる
- 例：
 - Dequeのpopの実際の計算量
 - $O(1)$ だったり $O(N)$ だったり
 - Dequeのpopのならし計算量
 - $O(1)$

Dequeの計算量

- Dequeの操作のならし計算量は全て $O(1)$
 - ほんまか
- ならし計算量が $O(1)$ ということは N 回操作したときの「実際の計算量」が $O(N)$ であるということ
 - 実験してみるとそれっぽい
- これから2つの方法でこれを証明する

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method

AccountingMethod

- 直訳: 会計法 (法律っぽい)
- 初めあなたの所持金は0円です
- あなたが操作をするたびに、「ならし計算量」円だけお金がふってきます
- あなたが操作をするたびに、「実際の計算量」円払わなければなりません
- あなたはどのタイミングでも所持金が負になってはいけません

AccountingMethod

- 何円ふってくるか、うまく決めて所持金が負にならなければそれがならし計算量
- うまく思いつこう！

AccountingMethod

- 以下のように設定するとうまくいく
 - 操作をすると 5 円ふってくる

AccountingMethod

- 以下のように設定するとうまくいく
 - 操作をすると 5 円ふってくる
- これでうまくいくことをこれから証明

AccountingMethod:5円の証明

- pushBack,pushFront
 - 実際の計算量は 1
 - よって1円払わないといけない
- popBack,popFrontのエイヤッしないやつ
 - 実際の計算量は 1
 - 1円払わないといけない
- どちらの場合も差し引いて4円貯金が増える

AccountingMethod:5円の証明

- popBack, popFrontのエイヤツするとき
 - もう片方のStackの中身(N 個とする)をいったん全部出して、半分ずつ入れ分ける
 - 実際の計算量は $2N$
 - 差し引いて $2N-5$ 円貯金が減る
- このとき借金しなければよい

AccountingMethod:5円の証明

- 実は以下の命題が成り立つ
 - 左側のStackにL個、右側のStackにR個の要素が入ってる時、 $4 \times \max(L, R)$ 円以上の貯金がある
 - 帰納法で証明可能

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い
 - pushとエイヤッしないpopは 貯金が4円増えるので成立し続ける

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い
 - pushとエイヤッしないpopは 貯金が4円増えるので成立し続ける
 - エイヤッするとき、もともと貯金が $4N$ 円以上あるので操作後は $2N$ 円以上が保証される

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い
 - pushとエイヤッしないpopは 貯金が4円増えるので成立し続ける
 - エイヤッするとき、もともと貯金が $4N$ 円以上あるので操作後は $2N$ 円以上が保証される
 - エイヤッ後は $4 \times \max(L, R)$ は $2N$ になる

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い
 - pushとエイヤッしないpopは 貯金が4円増えるので成立し続ける
 - エイヤッするとき、もともと貯金が $4N$ 円以上あるので操作後は $2N$ 円以上が保証される
 - エイヤッ後は $4 \times \max(L, R)$ は $2N$ になる 成立し続ける

AccountingMethod: $4 \times \max(L, R)$ の証明

- 命題: 貯金が $4 \times \max(L, R)$ 円以上
- はじめの状態(空の状態)では成立
- 成立している状態からどの操作をしても命題が成立し続けることを示せば良い
 - pushとエイヤッしないpopは 貯金が4円増えるので成立し続ける
 - エイヤッするとき、もともと貯金が $4N$ 円以上あるので操作後は $2N$ 円以上が保証される
 - エイヤッ後は $4 \times \max(L, R)$ は $2N$ になる 成立し続ける

(Q.E.D.)

これが

Accounting Method

AccountingMethod

- ならし $O(1)$ を求めるのに向いてる
 - ふってくる値段を貯金がなくならないように充分大きくするだけ
- ふってくる値段は操作毎に変えても良い
 - popは4円
 - pushは1円とかでも大丈夫
- 今回のようにデータ構造の状態に対して貯金の下限を示したりして使う

AccountingMethod:演習問題

- 以下をAccountingMethodで示してみよ
- 動的にメモリを確保する配列(vector)のpush_backのならし計算量: $O(1)$
 - メモリがなくなったら別の場所にいまの2倍の大きさのメモリを確保します

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method

PotentialMethod

- データ構造の状態に対してPotentialという値を定義する
- ある操作をしてPotentialがP1からP2に変わったとする
- そのならし計算量を
「実際の計算量」+ $P2 - P1$
として定義する

PotentialMethod:なんでそんなことするの

- 長い処理を考える

PotentialMethod:なんでそんなことするの

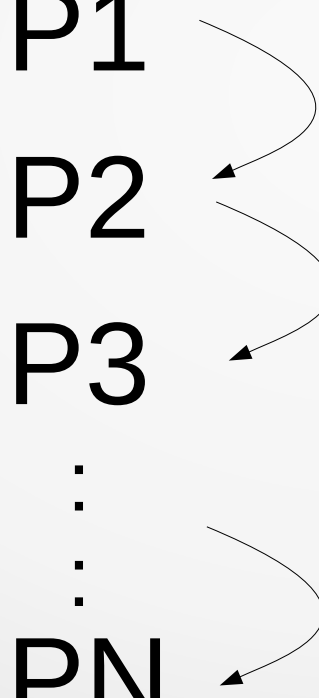
- 長い処理を考える

状態	Potential	実計算量	均計算量
C1	P1	X1	
C2	P2	X2	
C3	P3	⋮	
⋮	⋮	⋮	
CN	PN	XN	

PotentialMethod:なんでそんなことするの

- ポテンシャルと実計算量から均計算量を出す


状態	Potential	実計算量	均計算量
C1	P1	X1	$X1 + P2 - P1$
C2	P2	X2	$X2 + P3 - P2$
C3	P3	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots
CN	PN	XN	$XN + PN - (ry$



PotentialMethod:なんでそんなことするの

- 計算量の総和を考えてみる

状態	Potential	実計算量	均計算量
C1	P1	X1	$X1 + P2 - P1$
C2	P2	X2	$X2 + P3 - P2$
C3	P3	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots
CN	PN	XN	$XN + PN - (ry$



PotentialMethod:なんでそんなことするの

- 計算量の総和を考えてみる
- 実計算量
 - $X_1 + X_2 + X_3 + \dots + X_N$
- 均計算量
 - $(X_1 + P_2 - P_1) + (X_2 + P_3 - P_2) + \dots$ (ry
 - ↓
 - $X_1 + X_2 + X_3 + \dots + X_N + (P_N - P_1)$

PotentialMethod:なんでそんなことするの

- 計算量の総和を考えてみる
- 実計算量
 - $X_1 + X_2 + X_3 + \dots + X_N$
- 均計算量
 - $(X_1 + P_2 - P_1) + (X_2 + P_3 - P_2) + \dots$ (ry
 - ↓
 - $X_1 + X_2 + X_3 + \dots + X_N + (P_N - P_1)$

PotentialMethod:なんでそんなことするの

- 「実計算量の総和」と「均計算量の総和」の差が $P_N - P_1$
 - 終状態と始状態のPotentialの差
 - これが「均計算量の総和」以下のオーダーなら、実計算量と均計算量のオーダーが一致する
 - 均計算量の定義と合致

Potential Method: Dequeのポテンシャル

- うまいポテンシャルを見つけて、どんな操作も均計算量が $O(1)$ であることを示せば良い

Potential Method: Dequeのポテンシャル

- うまいポテンシャルを見つけて、どんな操作も均計算量が $O(1)$ であることを示せば良い
- 今回は左側のStackの要素数を L 、右側のStackの要素数を R として
$$4 * \max(L, R)$$
をPotentialとするとうまくいく

Potential Method: $4 * \max(L, R)$ の証明

- 各操作の均計算量
 - push
 - ポテンシャルはたかだか4増える
 - 実計算量は1
 - よってならし計算量は5

PotentialMethod: $4 * \max(L, R)$ の証明

- 各操作の均計算量
 - pop(エイヤッなし)
 - ポテンシャルは増えない
 - 実計算量は1
 - よってならし計算量は1以下

PotentialMethod: $4 * \max(L, R)$ の証明

- 各操作の均計算量
 - pop(エイヤッあり)
 - ポテンシャルは $4 * (N/2)$ 減る
 - 実計算量は $2N$
 - よってならし計算量は 0

PotentialMethod: $4 * \max(L, R)$ の証明

- PN-P1
 - N回操作した後の均計算量の総和は $O(N)$
 - PN-P1の最大値は $4N = O(N)$
 - ずっと同じ方からpushしたとき
 - よっしゃ

Potential Method: $4 * \max(L, R)$ の証明

- ポテンシャル $4 * \max(L, R)$ での均計算量がすべて $O(1)$ でした

PotentialMethod: $4 * \max(L, R)$ の証明

- ポテンシャル $4 * \max(L, R)$ での均計算量がすべて $O(1)$ でした

(Q.E.D.)

これが

Potential Method

PotentialMethod

- ならし $O(1)$ 以外を求めるのにも有用
 - フィボナッチヒープの $O(\log N)$
 - スプレー木の $O(\log N)$
 - UnionFind木の $O(\alpha(n))$ (アッカーマン関数の逆関数)
- $4 * \max(L, R)$ はAccountingMethodにもでてきた値
 - AccountingもPotentialも歩み寄り方が違うだけで、やりたい証明は同じ

PotentialMethod:演習問題

- 以下をPotentialMethodで示してみよ
- 動的にメモリを確保する配列(vector)のpush_backのならし計算量: $O(1)$
 - メモリがなくなったら別の場所にいまの2倍の大きさのメモリを確保します

目次

- 有名なことわざの紹介
- ならし計算量のはなし
- Accounting Method
- Potential Method
- おしまい