Algoritmos de marcação para caching

Sec 13.8 do KT

Cache

Modelo: itens do mesmo tamanho

Memória cache de tamanho k (itens)

Sequência de requisições:

$$S : d_1, d_2, \ldots, d_m$$

Cache hit: acerto – item está no cache

Cache miss: falha – item não está no cache

Se o cache está cheio e entra um novo item,
outro tem que sair (é despejado).

Políticas de despejo

Política de despejo (*replacement policy*):
algoritmo que decide quando trazer um
item para o cache e que item despejar.

Política preguiçosa (*lazy policy*):
um item só é trazido para o cache se ele foi requisitado.

Política FF (*farthest first* - algoritmo de Bélády): despeja
o item que será acessado novamente no futuro mais distante.

Vimos na aula passada que a política FF é ótima:
provoca o menor número possível de falhas.

Duas políticas online

LRU: least recently used

MRU: most recently used

Para variantes e mais referências, veja a página
https://en.wikipedia.org/wiki/Cache_replacement_policies

Algoritmos de marcação por fases

Algoritmo:

C: itens que estão no cache.
Cada item de C está marcado ou desmarcado.

Fase: no início, todos os itens desmarcados.
recebe requisição do item s.
marca s.
se $s \in C$, atende s e passa para o próximo.

se $s \notin C$,
  se C está todo marcado
    desmarca todos os itens e começa nova
    fase deixando s para ser atendido nela.
  senão
    despeja de C um dos itens desmarcados,
    traz s no seu lugar e passa para o próximo.

Algoritmos de marcação por fases

Algoritmo:

C: itens que estão no cache.

Cada item de C está marcado ou desmarcado.

Fase: no início, todos os itens desmarcados.

recebe requisição do item s.

marca s.

se $s \in C$, atende s e passa para o próximo.

se $s \notin C$,

se C está todo marcado

desmarca todos os itens e começa nova fase deixando s para ser atendido nela.

senão

despeja de C um dos itens desmarcados, traz s no seu lugar e passa para o próximo.

Note que LRU é um algoritmo de marcação.

Algoritmos de marcação por fases

Algoritmo:

C: itens que estão no cache.

Cada item de C está marcado ou desmarcado.

Fase: no início, todos os itens desmarcados.

recebe requisição do item s.

marca s.

se $s \in C$, atende s e passa para o próximo.

se $s \notin C$,

se C está todo marcado

desmarca todos os itens e começa nova fase deixando s para ser atendido nela.

senão

despeja de C um dos itens desmarcados, traz s no seu lugar e passa para o próximo.

Vimos que tal algoritmo é k-competitivo.

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Análise:

Fase j:
item requisitado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Análise:

Fase j:
item requisitado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Análise:

Fase j:
item requisitado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Análise:

Fase j:
item requisitado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Um algoritmo melhor

Sorteie um desmarcado uniformemente para despejar!

Análise:

Fase j:
item requisitado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Prova: Na fase j, há requisição para k itens distintos.

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Prova: Na fase j, há requisição para k itens distintos.

Na fase $j+1$, há requisição para c_{j+1} itens distintos destes.

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$.

$f_j(d)$: número de falhas da política ótima na fase j.

$$f(d) = \sum_{j=1}^{r} f_j(d)$$

c_j: número de itens frescos na fase j.

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Prova: Na fase j, há requisição para k itens distintos.

Na fase $j+1$, há requisição para c_{j+1} itens distintos destes.

Então um algoritmo ótimo incorre em $\geq c_{j+1}$ falhas. □

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.
$f(d) = \sum_{j=1}^{r} f_j(d)$.
c_j: número de itens frescos na fase j. (Tome $c_1 = 0$.)

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.
$f(d) = \sum_{j=1}^{r} f_j(d)$.
c_j: número de itens frescos na fase j. (Tome $c_1 = 0$.)

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Consequência:

$$2f(d) - f_1(d) - f_r(d) \;=\; \sum_{j=1}^{r-1}(f_j(d) + f_{j+1}(d)) \;\geq\; \sum_{j=1}^{r-1} c_{j+1}$$

Um algoritmo melhor

Fase j:
item desmarcado é fresco se não foi marcado na fase $j-1$
e é amanhecido caso contrário.

$f_j(d)$: número de falhas da política ótima na fase j.
$f(d) = \sum_{j=1}^{r} f_j(d)$.
c_j: número de itens frescos na fase j. (Tome $c_1 = 0$.)

Lema: $f_j(d) + f_{j+1}(d) \geq c_{j+1}$

Consequência:

$$2f(d) - f_1(d) - f_r(d) \;=\; \sum_{j=1}^{r-1}(f_j(d) + f_{j+1}(d)) \;\geq\; \sum_{j=1}^{r-1} c_{j+1}$$

Logo, $2\,f(d) \geq \sum_{j=1}^{r} c_j$.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X = \sum_{j=1}^{r} X_j$.

Queremos estimar $\mathrm{E}[X]$.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X = \sum_{j=1}^{r} X_j$.

Queremos estimar $\mathrm{E}[X]$.

Note que
$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X = \sum_{j=1}^{r} X_j$.

Queremos estimar $\mathrm{E}[X]$.

Note que
$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Considere a requisição do i-ésimo amanhecido na fase j.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X = \sum_{j=1}^{r} X_j$.

Queremos estimar $\mathrm{E}[X]$.

Note que
$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Considere a requisição do i-ésimo amanhecido na fase j.

Cache:	frescos	$c \leq c_j$
	amanhecidos marcados	$i - 1$
	amanhecidos desmarcados	$k - c - i + 1$

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X = \sum_{j=1}^{r} X_j$.

Queremos estimar $\mathrm{E}[X]$.

Note que
$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Considere a requisição do i-ésimo amanhecido na fase j.

Cache:	frescos	$c \leq c_j$
	amanhecidos marcados	$i - 1$
	amanhecidos desmarcados	$k - c - i + 1$

Logo o número de amanhecidos fora do cache é c.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

$$\Pr[\text{falha no } i\text{-ésimo amanhecido}] = \frac{c}{k-i+1} \leq \frac{c_j}{k-i+1}.$$

($k-i+1$: número de amanhecidos desmarcados (dentro e fora do cache))

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

$$\Pr[\text{falha no } i\text{-ésimo amanhecido}] = \frac{c}{k-i+1} \leq \frac{c_j}{k-i+1}.$$

($k-i+1$: número de amanhecidos desmarcados (dentro e fora do cache))

Logo $\mathrm{E}[X_j] \leq c_j + \sum_{i=c_j+1}^{k} \frac{c_j}{i} = c_j(1 + H_k - H_{c_j}) \leq c_j H_k$ e

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

$$\Pr[\text{falha no } i\text{-ésimo amanhecido}] \;=\; \frac{c}{k-i+1} \;\le\; \frac{c_j}{k-i+1}.$$

($k-i+1$: número de amanhecidos desmarcados (dentro e fora do cache))

Logo $\mathrm{E}[X_j] \le c_j + \sum_{i=c_j+1}^{k} \frac{c_j}{i} = c_j(1 + H_k - H_{c_j}) \le c_j H_k$ e

$$\mathrm{E}[X] \;=\; \sum_{j=1}^{r} \mathrm{E}[X_j] \;\le\; \sum_{j=1}^{r} c_j H_k \;\le\; 2H_k f(d) \;=\; \mathrm{O}(\lg k)\, f(d).$$

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

$$\Pr[\text{falha no } i\text{-ésimo amanhecido}] = \frac{c}{k-i+1} \leq \frac{c_j}{k-i+1}.$$

($k - i + 1$: número de amanhecidos no cache)

Temos que $\mathrm{E}[X] = \mathrm{O}(\lg k) f(d)$.

Análise deste algoritmo aleatorizado

X_j: número de falhas do algoritmo aleatorizado na fase j.

$X_j = c_j$ falhas por itens frescos + falhas por amanhecidos.

Queremos estimar $\mathrm{E}[X]$, onde $X = \sum_{j=1}^{r} X_j$.

Considere a requisição do i-ésimo amanhecido na fase j.

Número de amanhecidos fora do cache é c, assim

$$\Pr[\text{falha no } i\text{-ésimo amanhecido}] = \frac{c}{k-i+1} \leq \frac{c_j}{k-i+1}.$$

($k - i + 1$: número de amanhecidos no cache)

Temos que $\mathrm{E}[X] = \mathrm{O}(\lg k) f(d)$.

Portanto esse algoritmo é $\mathrm{O}(\lg k)$-competitivo.

Tópicos de Análise de Algoritmos

Análise amortizada

Cap 17 do CLRS

Análise amortizada

Serve para analisar uma sequência de operações ou iterações onde o pior caso individual não reflete o pior caso da sequência.

Análise amortizada

Serve para analisar uma sequência de operações ou iterações onde o pior caso individual não reflete o pior caso da sequência.

Em outras palavras, serve para melhorar análises de pior caso que baseiem-se diretamente no pior caso de uma operação/iteração e que deem uma delimitação frouxa para o tempo de pior caso da sequência.

Análise amortizada

Serve para analisar uma sequência de operações ou iterações onde o pior caso individual não reflete o pior caso da sequência.

Em outras palavras, serve para melhorar análises de pior caso que baseiem-se diretamente no pior caso de uma operação/iteração e que deem uma delimitação frouxa para o tempo de pior caso da sequência.

Métodos:

- agregado
- por créditos
- potencial

Odômetro binário

Considere um contador binário, inicialmente zerado,
representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: incrementa.

Odômetro binário

Considere um contador binário, inicialmente zerado, representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: incrementa.

```
Incrementa (A, n)
1   i ← 0
2   enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5   se i < n
6         então A[i] ← 1
```

Odômetro binário

Considere um contador binário, inicialmente zerado, representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: incrementa.

```
Incrementa (A, n)
1   i ← 0
2   enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5   se i < n
6         então A[i] ← 1
```

Consumo de tempo no pior caso: $\Theta(n)$

Odômetro binário

Considere um contador binário, inicialmente zerado,
representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: Incrementa.

Consumo de tempo no pior caso: $\Theta(n)$.

O odômetro dá uma volta completa
a cada 2^n execuções do Incrementa.

Odômetro binário

Considere um contador binário, inicialmente zerado,
representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: Incrementa.

Consumo de tempo no pior caso: $\Theta(n)$.

O odômetro dá uma volta completa
a cada 2^n execuções do Incrementa.

Quanto tempo leva para o odômetro dar uma volta completa?

Odômetro binário

Considere um contador binário, inicialmente zerado,
representado em um vetor $A[0\,..\,n-1]$, onde cada $A[i]$ vale 0 ou 1.

Operação: Incrementa.

Consumo de tempo no pior caso: $\Theta(n)$.

O odômetro dá uma volta completa
a cada 2^n execuções do Incrementa.

Quanto tempo leva para o odômetro dar uma volta completa?

Leva $\mathrm{O}(n2^n)$.

Será que é $\Theta(n2^n)$?

Odômetro binário

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Odômetro binário

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Método agregado

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

Método agregado

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

bit 0 muda 2^n vezes
bit 1 muda 2^{n-1} vezes
bit 2 muda 2^{n-2} vezes
...
bit $n-2$ muda 4 vezes
bit $n-1$ muda 2 vezes

Método agregado

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

bit 0 muda 2^n vezes
bit 1 muda 2^{n-1} vezes
bit 2 muda 2^{n-2} vezes
...
bit $n-2$ muda 4 vezes
bit $n-1$ muda 2 vezes

Total de alterações de bits: $\sum_{i=1}^{n} 2^i < 2 \cdot 2^n$.

Método agregado

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

bit 0 muda 2^n vezes
bit 1 muda 2^{n-1} vezes
bit 2 muda 2^{n-2} vezes
...
bit $n-2$ muda 4 vezes
bit $n-1$ muda 2 vezes

Total de alterações de bits: $\sum_{i=1}^{n} 2^i < 2 \cdot 2^n$.

Custo da volta completa: $\Theta(2^n)$.

Método agregado

i	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

bit 0 muda 2^n vezes
bit 1 muda 2^{n-1} vezes
bit 2 muda 2^{n-2} vezes
...
bit $n-2$ muda 4 vezes
bit $n-1$ muda 2 vezes

Total de alterações de bits: $\sum_{i=1}^{n} 2^i < 2 \cdot 2^n$.

Custo da volta completa: $\Theta(2^n)$.

Custo amortizado por Incrementa: $\Theta(1)$.

Análise por créditos

Atribuímos um número fixo de créditos por operação
Incrementa de modo a pagar por toda alteração de bit.

Análise por créditos

Atribuímos um número fixo de créditos por operação
Incrementa de modo a pagar por toda alteração de bit.

Objetivo: atribuir o menor número possível de créditos
que seja ainda suficiente para pagar por todas as alterações.

Análise por créditos

Atribuímos um número fixo de créditos por operação Incrementa de modo a pagar por toda alteração de bit.

Objetivo: atribuir o menor número possível de créditos que seja ainda suficiente para pagar por todas as alterações.

Relembre...

```
Incrementa (A, n)
1   i ← 0
2   enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
5         i ← i + 1
6   se i < n
7         então A[i] ← 1
```

Análise por créditos

Atribuímos 2 créditos por Incrementa.

```
Incrementa (A, n)
1    i ← 0
2    enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5    se i < n
6         então A[i] ← 1
```

Um é usado para pagar pela alteração da linha 6.

Análise por créditos

Atribuímos 2 créditos por Incrementa.

```
Incrementa (A, n)
1    i ← 0
2    enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5    se i < n
6         então A[i] ← 1
```

Um é usado para pagar pela alteração da linha 6.

O outro fica armazenado sobre o bit alterado na linha 6.

Análise por créditos

Atribuímos 2 créditos por Incrementa.

```
Incrementa (A, n)
1    i ← 0
2    enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5    se i < n
6         então A[i] ← 1
```

Um é usado para pagar pela alteração da linha 6.

O outro fica armazenado sobre o bit alterado na linha 6.

Há um crédito armazenado sobre cada bit que vale 1.

Alterações da linha 3 são pagas por créditos armazenados por chamadas anteriores do Incrementa.

Análise por créditos

Atribuímos 2 créditos por Incrementa.

```
Incrementa (A, n)
1    i ← 0
2    enquanto i < n e A[i] = 1 faça
3         A[i] ← 0
4         i ← i + 1
5    se i < n
6         então A[i] ← 1
```

Um é usado para pagar pela alteração da linha 6.

O outro fica armazenado sobre o bit alterado na linha 6.

O número de créditos armazenados em cada instante
é o número de bits que valem 1, logo é sempre não negativo.

Custo amortizado por Incrementa: 2

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.

Note que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.

Note que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.

Note que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa.

Note que $c_i \leq 1 + t_i$
onde t_i é o número de bits 1 consecutivos no final do contador A_{i-1}.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.

Note que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa.

Note que $c_i \leq 1 + t_i$
onde t_i é o número de bits 1 consecutivos no final do contador A_{i-1}.

Note que $\phi(A_i) - \phi(A_{i-1}) \leq 1 - t_i$.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.

Note que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa.

Note que $c_i \leq 1 + t_i$
onde t_i é o número de bits 1 consecutivos no final do contador A_{i-1}.

Note que $\phi(A_i) - \phi(A_{i-1}) \leq 1 - t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.
Temos que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa e
t_i é o número de bits 1 consecutivos no final do contador A.
Temos que $c_i \leq 1 + t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.
Temos que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa e
t_i é o número de bits 1 consecutivos no final do contador A.
Temos que $c_i \leq 1 + t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Então o custo da volta completa é

$$c = \sum_{i=1}^{2^n} c_i$$

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.
Temos que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa e
t_i é o número de bits 1 consecutivos no final do contador A.
Temos que $c_i \leq 1 + t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Então o custo da volta completa é

$$c \;=\; \sum_{i=1}^{2^n} c_i \;=\; \sum_{i=1}^{2^n} \hat{c}_i + \phi(A_0) - \phi(A_{2^n})$$

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.
Temos que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa e
t_i é o número de bits 1 consecutivos no final do contador A.
Temos que $c_i \leq 1 + t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Então o custo da volta completa é

$$c \;=\; \sum_{i=1}^{2^n} c_i \;=\; \sum_{i=1}^{2^n} \hat{c}_i + \phi(A_0) - \phi(A_{2^n}) \;\leq\; \sum_{i=1}^{2^n} \hat{c}_i \;\leq\; 2 \cdot 2^n.$$

Método do potencial

Seja $\phi(A)$ o número de bits que valem 1 em $A[0\,..\,n-1]$.

Seja A_i o estado do contador A após o i-ésimo Incrementa.
Temos que $\phi(A_0) = 0$ e $\phi(A_i) \geq 0$.

Seja c_i o número de bits alterados no i-ésimo Incrementa e
t_i é o número de bits 1 consecutivos no final do contador A.
Temos que $c_i \leq 1 + t_i$.

Seja $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$.

Então o custo da volta completa é

$$c \;=\; \sum_{i=1}^{2^n} c_i \;=\; \sum_{i=1}^{2^n} \hat{c}_i + \phi(A_0) - \phi(A_{2^n}) \;\leq\; \sum_{i=1}^{2^n} \hat{c}_i \;\leq\; 2 \cdot 2^n.$$

Custo amortizado por Incrementa: 2 ▷ valor da delimitação no $\hat{c}_i$