

Análise de Algoritmos

**Parte destes slides são adaptações de slides
do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.**

Políticas de caching

U : conjunto de itens.

k : capacidade do cache.

d_1, \dots, d_m : sequência de itens de U .

Políticas de caching

U : conjunto de itens.

k : capacidade do cache.

d_1, \dots, d_m : sequência de itens de U .

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

Políticas de caching

U : conjunto de itens.

k : capacidade do cache.

d_1, \dots, d_m : sequência de itens de U .

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

Exemplo: Se $k = 9$, $d = 3$ e o cache está assim:

7	2	1
8	5	9
4	0	6

Políticas de caching

U : conjunto de itens.

k : capacidade do cache.

d_1, \dots, d_m : sequência de itens de U .

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

Exemplo: Se $k = 9$, $d = 3$ e o cache está assim:

7	2	1
8	5	9
4	0	6

Quem devemos desalojar?

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

7	2
1	8

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

7	2
1	8

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

7	2
1	5

contador de falhas: 1

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

7	2
1	5

contador de falhas: 1

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

9	2
1	5

contador de falhas: 2

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

9	2
1	5

contador de falhas: 2

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	5

contador de falhas: 3

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	5

contador de falhas: 3

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	0

contador de falhas: 4

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	0

contador de falhas: 4

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	6

contador de falhas: 5

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	6

contador de falhas: 5

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	3

contador de falhas: 6

Política ótima de caching

U : conjunto de itens.

k : capacidade do cache.

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

(mais distante no futuro)

Exemplo: $d = (7, 2, 1, 2, 8, 5, 7, 9, 4, 2, 5, 0, 6, 3, 1, 4, 2, 8, 9)$:

4	2
1	3

contador de falhas: 6

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

Por que esta política é ótima?

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

Por que esta política é ótima?

Um escalonamento é **reduzido** se traz um item para o cache apenas quando este item é requisitado.

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

Por que esta política é ótima?

Um escalonamento é **reduzido** se traz um item para o cache apenas quando este item é requisitado.

Lema: Dado um escalonamento S , sempre existe um escalonamento reduzido que tem no máximo o mesmo número de falhas que S .

Prova feita na aula.

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

S_{FF} : escalonamento obtido pela política acima.

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

S_{FF} : escalonamento obtido pela política acima.

S : escalonamento reduzido que faz as mesmas primeiras j decisões que S_{FF} .

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

S_{FF} : escalonamento obtido pela política acima.

S : escalonamento reduzido que faz as mesmas primeiras j decisões que S_{FF} .

Afirmção: Existe um escalonamento reduzido S' que faz as mesmas $j + 1$ primeiras decisões que S_{FF} e tem no máximo o mesmo número de falhas que S .

Prova feita na aula.

Política ótima de caching

Algoritmo ótimo de caching:

Dada uma sequência d_1, \dots, d_m de requisições de U , a cada instante, se necessário, desalojamos do cache o item que demorará mais para ser requisitado de novo.

S_{FF} : escalonamento obtido pela política acima.

S : escalonamento reduzido que faz as mesmas primeiras j decisões que S_{FF} .

Afirmção: Existe um escalonamento reduzido S' que faz as mesmas $j + 1$ primeiras decisões que S_{FF} e tem no máximo o mesmo número de falhas que S .

Prova feita na aula.

Consequência: S_{FF} é ótimo.

Política ótima de caching

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

S_{FF} : escalonamento obtido pela política anterior.

Lema: S_{FF} é ótimo.

Política ótima de caching

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

S_{FF} : escalonamento obtido pela política anterior.

Lema: S_{FF} é ótimo.

Problema: não conhecemos d de ante-mão...

Política ótima de caching

Algoritmo de manutenção de cache:

Dada uma coleção de k itens de U e um novo item d , decidir qual dos k itens será desalojado para dar espaço para d .

S_{FF} : escalonamento obtido pela política anterior.

Lema: S_{FF} é ótimo.

Problema: não conhecemos d de ante-mão...

Era melhor uma política **online** e S_{FF} não é online...

Duas políticas online

LRU: least recently used

MRU: most recently used

Duas políticas online

LRU: least recently used

MRU: most recently used

Algoritmos de marcação por fases:

C : itens que estão no cache.

Cada item de C está marcado ou desmarcado.

Fase: no início, todos os itens desmarcados.

Duas políticas online

LRU: least recently used

MRU: most recently used

Algoritmos de marcação por fases:

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 .

Duas políticas online

LRU: least recently used

MRU: most recently used

Algoritmos de marcação por fases:

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 .

Duas políticas online

LRU: least recently used

MRU: most recently used

Algoritmos de marcação por fases:

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.

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.

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.

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.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

De fato, em cada fase, há $k + 1$ itens distintos requisitados.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

De fato, em cada fase, há $k + 1$ itens distintos requisitados.

Então $f(d) \geq r - 1$, onde r é o número de fases do algoritmo.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

De fato, em cada fase, há $k + 1$ itens distintos requisitados.

Então $f(d) \geq r - 1$, onde r é o número de fases do algoritmo.

O algoritmo de marcação faz no máximo k falhas por fase.

Logo, no total, faz no máximo $kr \leq kf(d) + k$ falhas.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

De fato, em cada fase, há $k + 1$ itens distintos requisitados.

Então $f(d) \geq r - 1$, onde r é o número de fases do algoritmo.

O algoritmo de marcação faz no máximo k falhas por fase.

Logo, no total, faz no máximo $kr \leq kf(d) + k$ falhas.

Dizemos que tal algoritmo é k -competitivo.

Análise do algoritmo de marcação

Fixe uma sequência d de requisições.

$f(d)$: número mínimo de falhas para atender d .

Em cada fase, há pelo menos uma falha.

De fato, em cada fase, há $k + 1$ itens distintos requisitados.

Então $f(d) \geq r - 1$, onde r é o número de fases do algoritmo.

O algoritmo de marcação faz no máximo k falhas por fase.

Logo, no total, faz no máximo $kr \leq kf(d) + k$ falhas.

Dizemos que tal algoritmo é k -competitivo.

Em particular, o LRU 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.

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

Análise deste algoritmo aleatorizado

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

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

Queremos estimar $E[X]$.

Análise deste algoritmo aleatorizado

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

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

Queremos estimar $E[X]$.

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 $E[X]$.

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 $E[X]$.

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 $E[X]$.

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 $E[X]$, 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 $E[X]$, 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)

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 $E[X]$, 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)

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

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

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

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 $E[X]$, 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)

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 $E[X]$, 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)

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

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