Tópicos de Análise de Algoritmos

Método guloso

Sec 4.3 do KT

Políticas de caching

U: conjunto de itens.

k: capacidade do cache.

$d_1, \ldots, d_m$: sequência de itens de U.

Políticas de caching

U: conjunto de itens.

k: capacidade do cache.

$d_1, \ldots, d_m$: 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.

$d_1, \ldots, d_m$: 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.

$d_1, \ldots, d_m$: 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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 $d_1, \ldots, d_m$ 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

Por que esta política é ótima?

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

Por que esta política é ótima?

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

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

Por que esta política é ótima?

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

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

O escalonamento S_{FF} é reduzido.

Por que esta política é ótima?

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

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

O escalonamento S_{FF} é reduzido.

Para mostrar que S_{FF} é ótimo,
usaremos alguns escalonamentos que não são reduzidos.

Escalonamentos reduzidos

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

Escalonamentos reduzidos

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

Prova: Em cada passo i em que S traz um item que não foi requisitado, o escalonamento $\bar{S}$ finge fazer isso mas não faz.

Escalonamentos reduzidos

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

Prova: Em cada passo i em que S traz um item que não foi requisitado, o escalonamento $\bar{S}$ finge fazer isso mas não faz.

O escalonamento $\bar{S}$ apenas traz esse item para o cache no passo $j > i$ em que esse item foi requisitado.

Escalonamentos reduzidos

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

Prova: Em cada passo i em que S traz um item que não foi requisitado, o escalonamento $\bar{S}$ finge fazer isso mas não faz.

O escalonamento $\bar{S}$ apenas traz esse item para o cache no passo $j > i$ em que esse item foi requisitado.

A falha de cache em $\bar{S}$ no passo j pode ser paga pela operação feita por S no passo i. □

Escalonamentos reduzidos

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

Prova: Em cada passo i em que S traz um item que não foi requisitado, o escalonamento $\bar{S}$ finge fazer isso mas não faz.

O escalonamento $\bar{S}$ apenas traz esse item para o cache no passo $j > i$ em que esse item foi requisitado.

A falha de cache em $\bar{S}$ no passo j pode ser paga pela operação feita por S no passo i. □

Note que o número de itens trazidos para o cache por um escalonamento reduzido é exatamente seu número de falhas de cache.

Análise do S_{FF}

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

Análise do S_{FF}

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

Ideia: transformar um escalonamento ótimo no S_{FF}
sem aumentar o número de falhas de cache.

Análise do S_{FF}

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

Ideia: transformar um escalonamento ótimo no S_{FF}
sem aumentar o número de falhas de cache.

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

Análise do S_{FF}

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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.

Ideia: transformar um escalonamento ótimo no S_{FF}
sem aumentar o número de falhas de cache.

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

Afirmação: 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 da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Se d está no cache de S_{FF},
então d também está no cache de S (pois eles coincidem até j).

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Se d está no cache de S_{FF},
então d também está no cache de S (pois eles coincidem até j).

Nesse caso, tome $S' = S$.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Se d está no cache de S_{FF},
então d também está no cache de S (pois eles coincidem até j).

Nesse caso, tome $S' = S$.

Se d não está no cache, então S_{FF} desaloca um elemento e,
e S desaloca um elemento f.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Se d está no cache de S_{FF},
então d também está no cache de S (pois eles coincidem até j).

Nesse caso, tome $S' = S$.

Se d não está no cache, então S_{FF} desaloca um elemento e,
e S desaloca um elemento f.

Se $f = e$, novamente tome $S' = S$.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.

Se d está no cache de S_{FF},
então d também está no cache de S (pois eles coincidem até j).

Nesse caso, tome $S' = S$.

Se d não está no cache, então S_{FF} desaloca um elemento e,
e S desaloca um elemento f.

Se $f = e$, novamente tome $S' = S$.

Se $f \neq e$...

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.

   S' despeja f, sincronizando os caches de S e S'.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.

Se $e' = e$, os caches ficam sincronizados
e S' teve uma falha de cache a menos, contradição.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.

Se $e' \neq e$, então S' atende f, despeja e',
e traz e para o cache, para sincronizar (não reduzido).

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.
c) $d' = e$.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.
c) $d' = e$.

Não acontece! Por que?

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.
b) $d' = f$ e S despeja e'.
c) $d' = e$.

Não acontece antes de b) pela escolha do S_{FF}:
f é requisitado antes de e.

Prova da afirmação

Afirmação: 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: Seja d a requisição do passo $j+1$.
Suponha que d não está no cache, S_{FF} desaloca e e S desaloca $f \neq e$.

Seja S' o escalonamento que segue S_{FF}, despejando e,
e depois segue S enquanto possível.

Quando dá o primeiro problema?

Numa requisição d' tal que

a) $d' \neq e, f$, sendo que d' não está no cache e S quer despejar e.

b) $d' = f$ e S despeja e'.

c) $d' = e$.

S' satisfaz a afirmação. □

Consequência

Algoritmo ótimo de caching:
Dada uma sequência $d_1, \ldots, d_m$ 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 ótimo que faz
as mesmas primeiras j decisões que S_{FF}.

Afirmação: Existe um escalonamento reduzido S'
que faz as mesmas $j+1$ primeiras decisões que S_{FF} e
tem o mesmo número de falhas que S, ou seja, S' também ótimo.

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.