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

Política ótima de caching

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.

Por que esta política é ótima?

Política ótima de caching

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.

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

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 $d_1, \ldots, d_m$ 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.

Política ótima de caching

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 que faz
as mesmas primeiras j decisões que S_{FF}.

Política ótima de caching

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 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 feita na aula.

Política ótima de caching

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 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 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.