

AULA 16

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$, encontrar uma **coleção máxima de intervalos** disjuntos dois a dois.

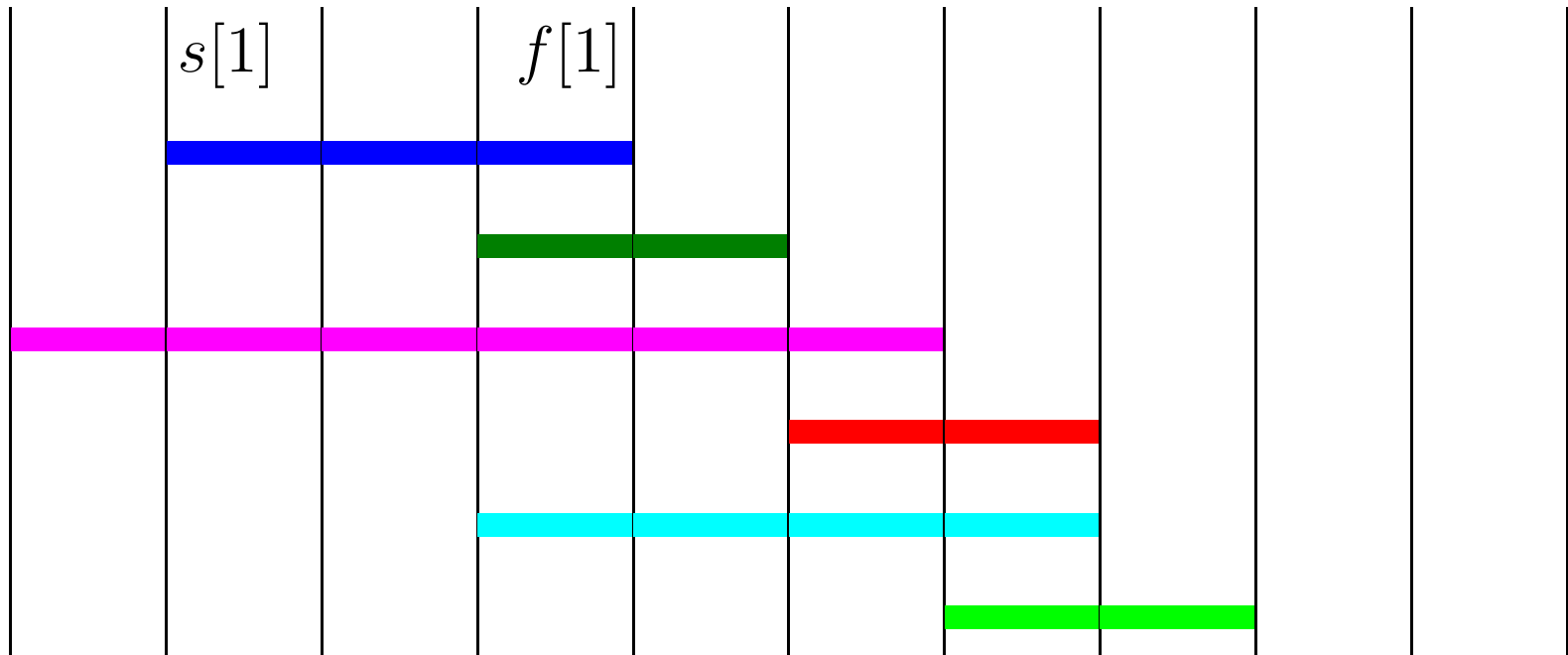
Solução é um subconjunto A de $\{1, \dots, n\}$.

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$, encontrar uma **coleção máxima de intervalos** disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \dots, n\}$.

Exemplo:

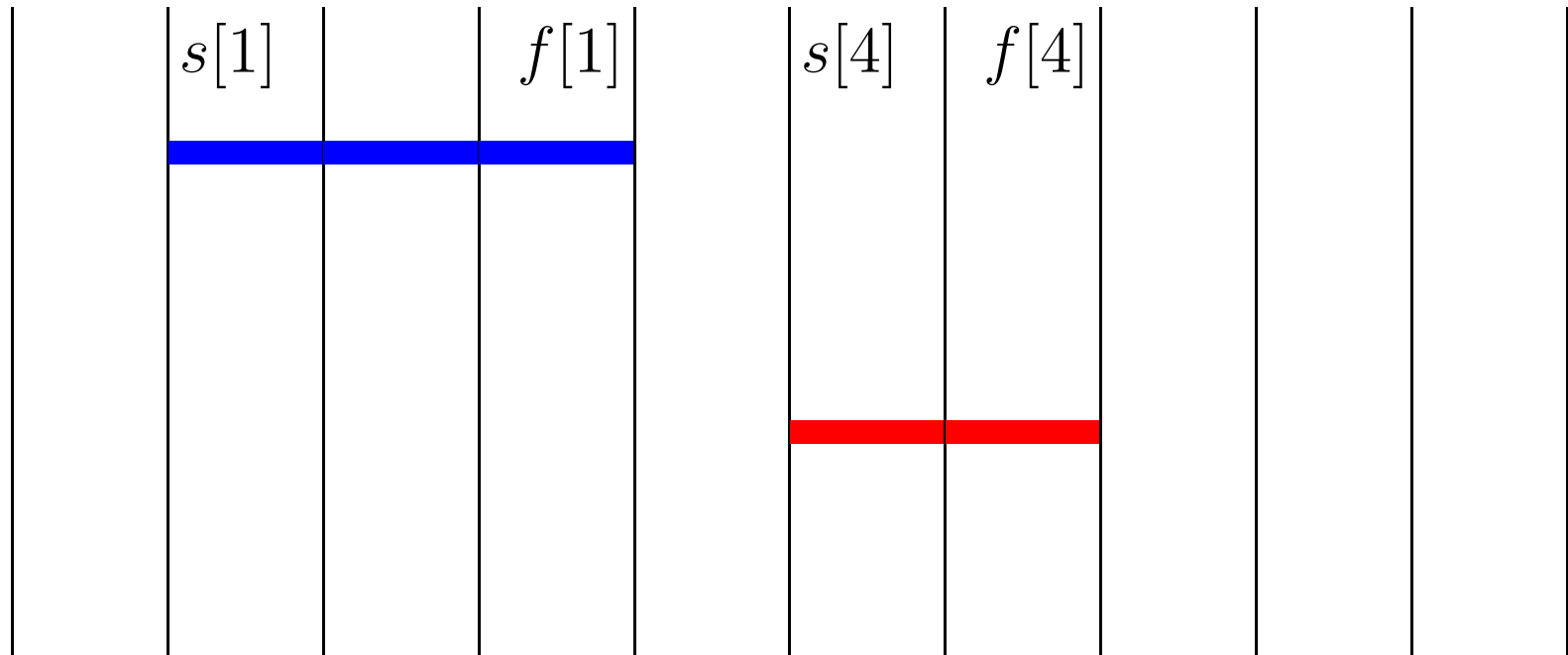


Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$, encontrar uma **coleção máxima de intervalos** disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \dots, n\}$.

Solução:

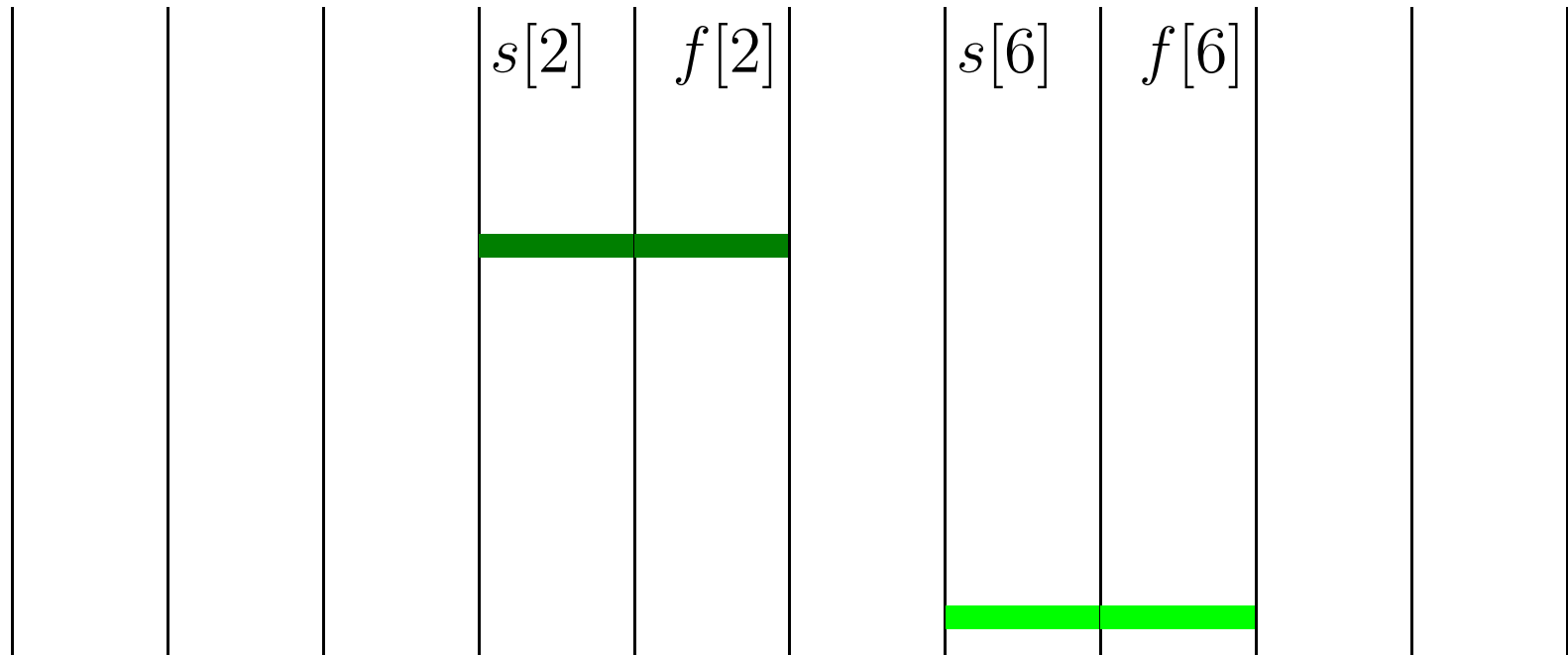


Problema dos intervalos disjuntos

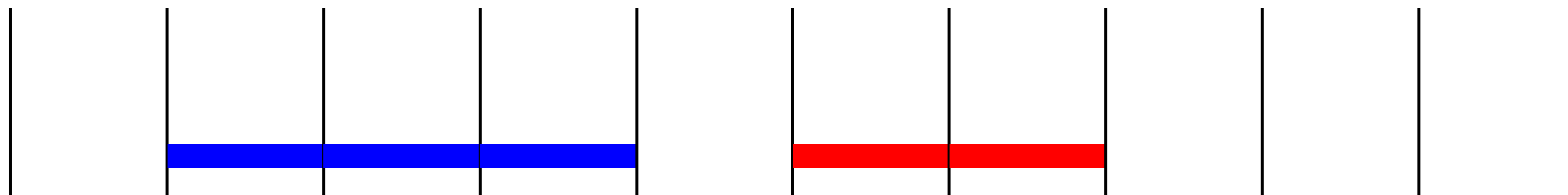
Problema: Dados intervalos $[s[1], f[1]), \dots, [s[n], f[n])$, encontrar uma **coleção máxima de intervalos** disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \dots, n\}$.

Solução:



Motivação



Se cada intervalo é uma “**atividade**”, queremos coleção disjunta máxima de atividades compatíveis (*i* e *j* são compatíveis se $f[i] \leq s[j]$)

Nome no CLRS: **Activity Selection Problem**

Subestrutura ótima

Intervalos $S := \{1, \dots, n\}$

Suponha que A é **coleção máxima** de intervalos de S disjuntos dois a dois.

Se $i \in A$

então $A - \{i\}$ é **coleção máxima** de intervalos disjuntos de $S - \{k : [s[k], f[k]) \cap [s[i], f[i]) \neq \emptyset\}$.

senão A é **coleção máxima** de intervalos disjuntos de $S - \{i\}$.

Demonstre a propriedade.

Subestrutura ótima II

Intervalos $S := \{1, \dots, n\}$

Suponha que A é **coleção máxima** de intervalos de S disjuntos dois a dois.

Se $i \in A$ é tal que $f[i]$ é **mínimo**

então $A - \{i\}$ é **coleção máxima** de intervalos disjuntos
de $\{k : s[k] \geq f[i]\}$.

$\{k : s[k] \geq f[i]\} =$ todos intervalos “à direita” de “ i ”.

Demonstre a propriedade.

Algoritmo de programação dinâmica

Suponha $f[1] \leq f[2] \leq \dots \leq f[n]$

$t[i]$ = tamanho de uma subcoleção
disjunta máxima de $\{i, \dots, n\}$

$$t[n] = 1$$

$$t[i] = \max \{t[i + 1], 1 + t[k]\} \quad \text{para } i = 1, \dots, n - 1,$$

onde k é o menor índice tal que $s[k] \geq f[i]$.

Algoritmo de programação dinâmica

DYNAMIC-ACTIVITY-SELECTOR (s, f, n)

0 ordene s e f de tal forma que

$$f[1] \leq f[2] \leq \dots \leq f[n]$$

1 $A[n + 1] \leftarrow \emptyset$

2 **para** $i \leftarrow n$ decrescendo até 1 **faça**

3 $A[i] \leftarrow A[i + 1]$

4 $k \leftarrow i + 1$

5 **enquanto** $k \leq n$ e $s[k] < f[i]$ **faça**

6 $k \leftarrow k + 1$

7 **se** $|A[i]| < 1 + |A[k]|$

8 **então** $A[i] \leftarrow \{i\} \cup A[k]$

9 **devolva** $A[1]$

Consumo de tempo é $\Theta(n^2)$.

Conclusão

Invariante: na linha 2 vale que

(i0) $A[j]$ é coleção disjunta máxima de $\{j, \dots, n\}$
para $j = i + 1, \dots, n$.

O consumo de tempo do algoritmo
DYNAMIC-ACTIVITY-SELECTOR é $\Theta(n^2)$.

Escolha gulosa

Intervalos $S := \{1, \dots, n\}$

Se $f[i]$ é mínimo em S ,

então **EXISTE** uma solução ótima A tal que $i \in A$.

Demonstre a propriedade.

Algoritmo guloso

Devolve uma coleção **máxima de intervalos** disjuntos dois a dois.

INTERVALOS-DISJUNTOS (s, f, n)

0 ordene s e f de tal forma que
 $f[1] \leq f[2] \leq \dots \leq f[n]$

1 $A \leftarrow \{1\}$

2 $i \leftarrow 1$

3 **para** $j \leftarrow 2$ **até** n **faça**

4 **se** $s[j] \geq f[i]$

5 **então** $A \leftarrow A \cup \{j\}$

6 $i \leftarrow j$

7 **devolva** A

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.

Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Conclusão

Na linha 3 vale que

(i0) A é uma **coleção máxima** de intervalos disjuntos de $(s, f, j-1)$

O consumo de tempo do algoritmo
INTERVALOS-DISJUNTOS é $\Theta(n \lg n)$.

Algoritmos gulosos

Algoritmo guloso

- procura maximal e acaba obtendo máximo
- procura ótimo local e acaba obtendo ótimo global

costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil provar que está correto

Problema precisa ter

- subestrutura ótima (como na programação dinâmica)
- propriedade da escolha gulosa (*greedy-choice property*)

Exercícios

Exercício 22.A [CLRS 16.1-1]

Escreva um algoritmo de programação dinâmica para resolver o problema dos intervalos disjuntos. (Versão simplificada do exercício: basta determinar o *tamanho* de uma coleção disjunta máxima.) Qual o consumo de tempo do seu algoritmo?

Exercício 22.B

Prove que o algoritmo guloso para o problema dos intervalos disjuntos está correto. (Ou seja, prove a propriedade da subestrutura ótima e a propriedade da escolha gulosa.)

Exercício 22.C [CLRS 16.1-2]

Mostre que a seguinte idéia também produz um algoritmo guloso correto para o problema da coleção disjunta máxima de intervalos: dentre os intervalos disjuntos dos já escolhidos, escolha um que tenha instante de início máximo. (Em outras palavras, suponha que os intervalos estão em ordem decrescente de início.)

Exercício 22.D [CLRS 16.1-4]

Nem todo algoritmo guloso resolve o problema da coleção disjunta máxima de intervalos. Mostre que nenhuma das três idéias a seguir resolve o problema. Idéia 1: Escolha o intervalo de menor duração dentre os que são disjuntos dos intervalos já escolhidos. Idéia 2: Escolha um intervalo seja disjunto dos já escolhidos e intercepte o menor número possível de intervalos ainda não escolhidos. Idéia 3: Escolha o intervalo disjunto dos já selecionados que tenha o menor instante de início.

Mais exercícios

Exercício 22.E [Coloração de intervalos. CLRS 16.1-3]

Queremos distribuir um conjunto de atividades no menor número possível de salas. Cada atividade a_i ocupa um certo intervalo de tempo $[s_i, f_i)$; duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos. Descreva um algoritmo guloso que resolve o problema. (Represente cada sala por uma cor; use o menor número possível de cores para pintar todos os intervalos.) Prove que o número de salas dado pelo algoritmo é, de fato, mínimo.

Exercício 22.F [pares de livros]

Suponha dado um conjunto de livros numerados de 1 a n . Suponha que o livro i tem peso $p[i]$ e que $0 < p[i] < 1$ para cada i . Problema: acondicionar os livros no menor número possível de envelopes de modo que cada envelope tenha no máximo 2 livros e o peso do conteúdo de cada envelope seja no máximo 1. Escreva um algoritmo guloso que calcule o número mínimo de envelopes. O consumo de tempo do seu algoritmo deve ser $O(n \lg n)$. Mostre que seu algoritmo está correto (ou seja, prove a “greedy-choice property” e a “optimal substructure” apropriadas). Estime o consumo de tempo do seu algoritmo.

Mais exercícios ainda

Exercício 22.G [Bin-packing]

São dados objetos $1, \dots, n$ e um número ilimitado de “latas”. Cada objeto i tem “peso” w_i e cada lata tem “capacidade” 1: a soma dos pesos dos objetos colocados em uma lata não pode passar de 1. Problema: Distribuir os objetos pelo menor número possível de latas. Programe e teste as seguintes heurísticas. Heurística 1: examine os objetos na ordem dada; tente colocar cada objeto em uma lata já parcialmente ocupada que tiver mais “espaço” livre sobrando; se isso for impossível, pegue uma nova lata. Heurística 2: rearranje os objetos em ordem decrescente de peso; em seguida, aplique a heurística 1. Essas heurísticas resolvem o problema? Compare com o exercício 22.F.

Para testar seu programa, sugiro escrever uma rotina que receba $n \leq 100000$ e $u \leq 1$ e gere w_1, \dots, w_n aleatoriamente, todos no intervalo $(0, u)$.

Exercício 22.H [parte de CLRS 16-4, modificado]

Seja $1, \dots, n$ um conjunto de *tarefas*. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a n . A cada tarefa t está associado um *prazo* p_t : a tarefa deveria ser executada em algum dia do intervalo $1 \dots p_t$. A cada tarefa t está associada uma *multa* não-negativa m_t . Se uma dada tarefa t é executada depois do prazo p_t , sou obrigado a pagar a multa m_t (mas a multa não depende do número de dias de atraso). Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total. Escreva um algoritmo guloso para resolver o problema. Prove que seu algoritmo está correto (ou seja, prove a “greedy-choice property” e

Análise amortizada

CLR 18 ou CLRS 17

Contador binário

Incrementa de 1 o número binário representado por $A[1 \dots k]$.

INCREMENT (A, k)

```
1   $i \leftarrow 0$ 
2  enquanto  $i < k$  e  $A[i] = 1$  faça
3       $A[i] \leftarrow 0$ 
4       $i \leftarrow i + 1$ 
5  se  $i < k$ 
6      então  $A[i] \leftarrow 1$ 
```

Contador binário

Incrementa de 1 o número binário representado por $A[1 \dots k]$.

INCREMENT (A, k)

1 $i \leftarrow 0$

2 **enquanto** $i < k$ **e** $A[i] = 1$ **faça**

3 $A[i] \leftarrow 0$

4 $i \leftarrow i + 1$

5 **se** $i < k$

6 **então** $A[i] \leftarrow 1$

Entrada:

$k-1$	3	2	1	0	
0	1	0	1	1	1

A

Contador binário

Incrementa de 1 o número binário representado por $A[1 \dots k]$.

INCREMENT (A, k)

```
1   $i \leftarrow 0$ 
2  enquanto  $i < k$  e  $A[i] = 1$  faça
3       $A[i] \leftarrow 0$ 
4       $i \leftarrow i + 1$ 
5  se  $i < k$ 
6      então  $A[i] \leftarrow 1$ 
```

Entrada:

$k-1$	3	2	1	0	
0	1	0	1	1	A

Saída:

$k-1$	3	2	1	0	
0	1	1	0	0	A

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(1)$
2	$O(k)$
3	$O(k)$
4	$O(k)$
5	$\Theta(1)$
6	$O(1)$


total $O(k) + \Theta(1) = O(k)$

“Custo” =

consumo de tempo = número de bits alterados = $O(k)$

Seqüência de n chamadas


INCR INCR ... INCR INCR INCR


 n

Consumo de tempo é $O(nk)$

Seqüência de n chamadas

INCR INCR ... INCR INCR INCR


 n

Consumo de tempo é $O(nk)$

EXAGERO!

Exemplo

$$n = 16$$

$$k = 6$$

A

5	4	3	2	1	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1

A

5	4	3	2	1	0
0	0	1	0	0	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	0	1
0	0	1	1	1	0
0	0	1	1	1	1
0	1	0	0	0	0

Exemplo

$n = 16$

$k = 6$

A						A					
5	4	3	2	1	0	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	1	0	0	1
0	0	0	0	1	0	0	0	1	0	1	0
0	0	0	0	1	1	0	0	1	0	1	1
0	0	0	1	0	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0	1	1	0	1
0	0	0	1	1	0	0	0	1	1	1	0
0	0	0	1	1	1	0	0	1	1	1	1
						0	1	0	0	0	0

$A[0]$	muda	n	vezes
$A[1]$	"	$\lfloor n/2 \rfloor$	"
$A[2]$	"	$\lfloor n/4 \rfloor$	"
$A[3]$	"	$\lfloor n/8 \rfloor$	"

Custo amortizado

Custo total:

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n = \Theta(n)$$

Custo amortizado (= custo fictício = custo faz-de-conta) de uma operação:

$$\frac{2n}{n} = \Theta(1)$$

Este foi o **método agregado** de análise: soma os custos de todas as operações para determinar o **custo amortizado de cada operação**

Conclusões

O consumo de tempo de uma seqüência de n execuções do algoritmo **INCREMENT** é $\Theta(n)$.

O consumo de tempo amortizado do algoritmo **INCREMENT** é $\Theta(1)$.

Método de análise contábil

Pague \$2 para mudar $0 \rightarrow 1$

\$0 para mudar $1 \rightarrow 0$

Custo amortizado por chamada de INCREMENT: \leq \$2

Seqüência de n chamadas de INCREMENT.

Como \$ armazenado nunca é negativo,

$$\begin{aligned} \text{soma custos reais} &\leq \\ &\leq \text{soma custos amortizados} \\ &= 2n \\ &= O(n) \end{aligned}$$

Método de análise potencial

$$A_0 \xrightarrow{1^{\text{a op}}} A_1 \xrightarrow{2^{\text{a op}}} A_2 \longrightarrow \dots \xrightarrow{n^{\text{a op}}} A_n$$

A_i = estado de A depois da i^{a} operação

Custo real da i^{a} operação: $c_i = t_i + 1$
onde t_i = número de $1 \rightarrow 0$ na i^{a} operação

Energia potencial de A_i :

$$\begin{aligned}\Phi(A_i) &= \text{número de bits "1"} \\ &= \Phi(A_{i-1}) - t_i + 1 \\ &\geq 0 \quad \textbf{Importante!}\end{aligned}$$

Custo amortizado

Custo amortizado da i^{a} operação:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(A_i) - \Phi(A_{i-1}) \\ &= c_i + \Phi(A_{i-1}) - t_i + 1 - \Phi(A_{i-1}) \\ &= c_i - t_i + 1 \\ &= (t_i + 1) - t_i + 1 \\ &= 2\end{aligned}$$

Custo amortizado

Soma dos **custos amortizados** limita a soma dos **custos reais** pois $\Phi \geq 0$:

$$\begin{aligned}\sum_{i=1}^n \hat{c}_i &= \sum (c_i + \Phi(A_i) - \Phi(A_{i-1})) \\&= \sum c_i + \Phi(A_n) - \Phi(A_0) \\&= \sum c_i + \Phi(A_n) - 0 \\&= \sum c_i + \Phi(A_n) \\&\geq \sum c_i, \quad (\text{pois } \Phi(A_n) \geq 0) \\&\geq \sum c_i \\&= \text{ soma dos custos reais}\end{aligned}$$

Conclusões

$$\sum c_i \leq \sum \hat{c}_i = 2n = O(n) .$$

Resumo da relação entre c , \hat{c} e Φ :

$$\hat{c}_i - c_i = \Phi_i - \Phi_{i-1}$$

Pilhas

PUSH(S, x) **empilha** o elemento s na pilha S

POP(S) **desempilha** e devolve o elemento no topo de S

STACK-EMPTY(S) devolve **VERDADE** se a pilha S está vazia e **FALSO** em caso contrário.

MULTIPOP (S, k)

```
1  enquanto STACK-EMPTY( $S$ ) = FALSO e  $k \neq 0$  faça
2      POP( $S$ )
3       $k \leftarrow k - 1$ 
```

Consumo de tempo de **MULTIPOP** é $O(\min\{|S|, k\})$.

Seqüência de n chamadas

PUSH PUSH ... POP PUSH MULTIPOP

$\underbrace{\hspace{15em}}_n$

Consumo de tempo é $O(n^2)$

Seqüência de n chamadas

PUSH PUSH ... POP PUSH MULTIPOP

$\underbrace{\hspace{15em}}_n$

Consumo de tempo é $O(n^2)$

EXAGERO!

Método agregado

Cada elemento pode ser desempilhado apenas 1 vez.

Logo, número de POP's em uma pilha não vazia \leq número de PUSH's.

Custo total:

$$\begin{aligned} &= \text{custo PUSH's} + \text{custo POP's} + \text{custo MULTIPOP's} \\ &\leq n + \text{custo POP's pilha ã vazia} \\ &\leq 2n = O(n) \end{aligned}$$

Custo amortizado de cada operação:

$$\frac{2n}{n} = \Theta(1)$$

Conclusões

O consumo de tempo de uma seqüência de n execuções dos algoritmos POP, PUSH e MULTIPOP é $\Theta(n)$.

O consumo de tempo amortizado de cada algoritmo é $\Theta(1)$.

Método de análise contábil

Custos reais:

PUSH \$1

POP \$1

MULTIPOP \$ $\min\{|S|, k\}$

Créditos:

Pague \$2 por um PUSH

 \$1 por um POP

 \$1 por um MULTIPOP

Custo amortizado por chamada de POP, PUSH e
MULTIPOP: \leq \$2

Método de análise contábil

Como \$ armazenado nunca é negativo,

$$\begin{aligned} \text{soma custos reais} &\leq \\ &\leq \text{soma custos amortizados} \\ &\leq 2n \\ &= O(n) \end{aligned}$$

Método de análise potencial

$$S_0 \xrightarrow{1^{\text{a op}}} S_1 \xrightarrow{2^{\text{a op}}} S_2 \longrightarrow \cdots \xrightarrow{n^{\text{a op}}} S_n$$

S_i = estado de S depois da i^{a} operação

Energia potencial de S_i :

$$\begin{aligned}\Phi(S_i) &= \text{número de objetos na pilha} \\ &\geq 0 \quad (\text{Importante!})\end{aligned}$$

Custo amortizado P_{USH}

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a i^{a} operação é **PUSH**, então $c_i = 1$ e

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\ &= c_i + 1 \\ &= 1 + 1 \\ &= 2\end{aligned}$$

Custo amortizado POP

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a i^{a} operação é **POP**, então $c_i = 1$ e

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\ &= c_i - 1 \\ &= 1 - 1 \\ &= 0\end{aligned}$$

Custo amortizado MULTIPOP

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a i^{a} operação é **MULTIPOP** e k' é o número de elementos desempilhados, então $c_i = k'$ e

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\ &= c_i - k' \\ &= k' - k' \\ &= 0\end{aligned}$$

Custo amortizado

Soma dos **custos amortizados** limita a soma dos **custos reais** pois $\Phi \geq 0$:

$$\begin{aligned}\sum_{i=1}^n c_i &= \sum (\hat{c}_i - \Phi(S_i) + \Phi(S_{i-1})) \\&= \sum \hat{c}_i - \Phi(S_n) + \Phi(S_0) \\&= \sum \hat{c}_i - \Phi(S_n) + 0 \\&= \sum \hat{c}_i - \Phi(S_n) \\&\leq \sum \hat{c}_i, \quad (\text{pois } \Phi(S_n) \geq 0) \\&\leq \sum 2 \\&= 2n\end{aligned}$$