Melhores momentos

AULA 17

Segmento de soma máxima

Um **segmento** de um vetor v[0 . . n−1] é qualquer subvetor da forma v[e . . d].

Problema: Dado um vetor v[0 . . n−1] de números inteiros, determinar um segmento v[e . . d] de soma máxima.

Entra:

	0									n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

Segmento de soma máxima

Sai:

	0		2				6			n−1
v	31	−41	59	26	−53	58	97	−93	−23	84

v[e . . d] = v[2 . . 6] é segmento de soma máxima.

v[2 . . 6] tem soma **187**.

Conclusões

O consumo de tempo do algoritmo segMax3 é proporcional a n^3.

O consumo de tempo do algoritmo segMax2 é proporcional a n^2.

O consumo de tempo do algoritmo segMax é proporcional a n.

Algumas técnicas

- Evitar recomputações. Usar espaço para armazenar resultados a fim de evitar recomputá-los (segMax2, segMax).
- Algoritmos incrementais/varredura. Solução de um subproblema é estendida a uma solução do problema original (segMax).
- Delimitação inferior. Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível (segMax).

AULA 18

Análise de algoritmo (continuação)

Programming Pearls: Algorithm Design Techniques,
Jon Bentley, Addison-Wesley, 1986

Análise experimental de algoritmos

"O interesse em experimentação, é devido ao reconhecimento de que os resultados teóricos, freqüentemente, não trazem informações referentes ao desempenho do algoritmo na prática."

Análise experimental de algoritmos

Segundo D.S. Johnson, pode-se dizer que existem quatro motivos básicos que levam a realizar um trabalho de implementação de um algoritmo:

- usar o código em uma aplicação particular, cujo propósito é descrever o impacto do algoritmo em um certo contexto;
- proporcionar evidências da superioridade de um algoritmo;

Análise experimental de algoritmos

- melhor compreensão dos pontos fortes e fracos e do desempenho das operações algorítmicas na prática; e
- produzir conjecturas sobre o comportamento do algoritmo no caso-médio sob distribuições específicas de instâncias onde a análise probabilística direta é muito difícil.

Ambiente experimental

A plataforma utilizada nos experimentos foi um computador rodando Ubuntu GNU/Linux 3.2.0-30

As especificações do computador que geraram as saídas a seguir são

```
model name: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GH:
cpu MHz   : 1596.000
cache size: 4096 KB

MemTotal  : 3354708 kB
```

Ambiente experimental

Os códigos foram compilados com o gcc 4.6.3 e com opções de compilação

-Wall -ansi -O2 -pedantic -Wno-unused-result

As implementações comparadas neste experimento são `segMax3`, `segMax2` e `segMax`.

Ambiente experimental

A estimativa do tempo é calculada utilizando-se:

```
#include <time.h>
[...]
clock_t start, end;
double time;

start = clock();

[...implementação...]

end  = clock();
time = ((double)(end - start))/CLOCKS_PER_SEC;
```

Resultados experimentais

segMax3		
n	tempo (s)	comentário
256	0.00	
512	0.02	
1024	0.12	
2048	0.89	
4096	6.99	
8192	55.55	$\approx$ 1 min
16384	444.25	$>$ 7 min
32768	59m15.550s	$\approx$ 1 hora

Resultados experimentais

segMax2		
n	tempo (s)	comentário
2048	0.00	
4096	0.01	
8192	0.02	
16384	0.13	
32768	0.53	
65536	2.12	
131072	8.52	
262144	34.08	$\approx$ 0.5 min
524288	136.56	$>$ 2 min
1048576	561.41	$>$ 9 min

Resultados experimentais

segMax		
n	tempo (s)	comentários
1048576	0.00	
2097152	0.01	
4194304	0.01	
8388608	0.01	
16777216	0.02	
33554432	0.05	
67108864	0.09	
134217728	0.19	$>$ 134 milhões
268435456	0.37	$>$ 268 milhões
536870912	0.75	$>$ 0.5 bilhões

Notação assintótica

CLRS 3.1

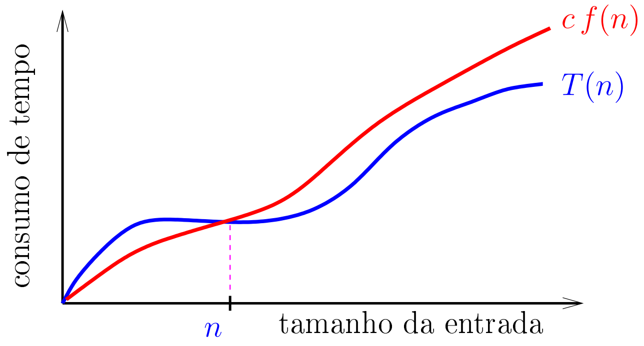
Notação assintótica

Sejam $\mathtt{T}(\mathtt{n})$ e $\mathtt{f}(\mathtt{n})$ funções dos inteiros nos reais. Dizemos que $\mathtt{T}(\mathtt{n})$ **é** $\mathrm{O}(\mathtt{f}(\mathtt{n}))$ se existem constantes positivas $\mathtt{c}$ e $\mathtt{n}_0$ tais que

$$\mathtt{T}(\mathtt{n}) \leq \mathtt{c}\,\mathtt{f}(\mathtt{n})$$

para todo $\mathtt{n} \geq \mathtt{n}_0$.

Notação assintótica

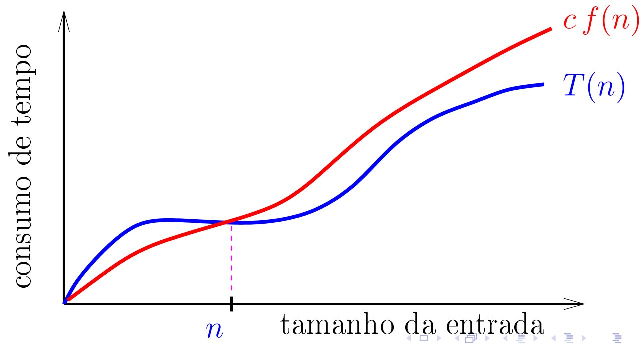


Mais informal

$\mathtt{T(n)}$ é $\mathrm{O}(\mathtt{f(n)})$ se existe $\mathtt{c} > 0$ tal que

$$\mathtt{T(n)} \leq \mathtt{c\,f(n)}$$

para todo n suficientemente GRANDE.



Consumo de tempo segMax3

Se a execução de cada linha de código consome 1 **unidade** de tempo o consumo total é:

linha		todas as execuções da linha	
1	$=$	1	$= \mathrm{O}(1)$
2	$=$	$n+1$	$= \mathrm{O}(n)$
3	$=$	$(n+1)+n+(n-1)+\cdots+1$	$= \mathrm{O}(n^2)$
4	$=$	$n+(n-1)+\cdots+1$	$= \mathrm{O}(n^2)$
5	$=$	$(2+\cdots+(n+1))+(2+\cdots+n)+\cdots+2$	$= \mathrm{O}(n^3)$
6	$=$	$(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+1$	$= \mathrm{O}(n^3)$
7	$=$	$n+(n-1)+(n-2)+\cdots+1$	$= \mathrm{O}(n^2)$
8	$\leq$	$n+(n-1)+(n-2)+\cdots+1$	$= \mathrm{O}(n^2)$
total	$=$	$\mathrm{O}(2n^3+4n^2+n+1)$	$= \mathrm{O}(n^3)$

Consumo de tempo segMax2

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1	$=$	1	$= \mathrm{O}(1)$
2	$=$	$n+1$	$= \mathrm{O}(n)$
3	$=$	n	$= \mathrm{O}(n)$
4	$=$	$(n+1)+n+\cdots+2$	$= \mathrm{O}(n^2)$
5	$=$	$n+(n-1)+\cdots+1$	$= \mathrm{O}(n^2)$
6	$=$	$n+(n-1)+\cdots+1$	$= \mathrm{O}(n^2)$
7	$\leq$	$n+(n-1)+\cdots+1$	$= \mathrm{O}(n^2)$
total	$=$	$\mathrm{O}(4n^2+2n+1)$	$= \mathrm{O}(n^2)$

Consumo de tempo segMaxI

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1	$=$	1	$= \mathrm{O}(1)$
2	$=$	$n+1$	$= \mathrm{O}(n)$
3	$=$	n	$= \mathrm{O}(n)$
4	$=$	$2+3+\ldots+(n+1)$	$= \mathrm{O}(n^2)$
5	$=$	$1+2+\ldots+n$	$= \mathrm{O}(n^2)$
6	$=$	$1+2+\ldots+n$	$= \mathrm{O}(n^2)$
7	$=$	n	$= \mathrm{O}(n)$
8	$\leq$	n	$= \mathrm{O}(n)$
total	$=$	$\mathrm{O}(3n^2+4n+1)$	$= \mathrm{O}(n^2)$

Conclusões

O consumo de tempo do algoritmo segMax3 é $\mathrm{O}(n^3)$.

O consumo de tempo do algoritmo segMax2 é $\mathrm{O}(n^2)$.

O consumo de tempo do algoritmo segMax é $\mathrm{O}(n)$.

$(3/2)n^2 + (7/2)n - 4$ versus $(3/2)n^2$

n	$(3/2)n^2 + (7/2)n - 4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	393216
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

$(3/2)n^2 + (7/2)n - 4$ versus $(3/2)n^2$

n	$(3/2)n^2 + (7/2)n - 4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	393216
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

$(3/2)n^2$ domina os outros termos

Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ($1\mu s$).

consumo de tempo(μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\,\lceil \lg n \rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Michael T. Goodrich e Roberto Tamassia, *Projeto de Algoritmos*, Bookman.

Crescimento de algumas funções

n	$\lg n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	$1{,}8\ 10^{19}$
128	7	11	896	16384	2097152	$3{,}4\ 10^{38}$
256	8	16	1048	65536	16777216	$1{,}1\ 10^{77}$
512	9	23	4608	262144	134217728	$1{,}3\ 10^{154}$
1024	10	32	10240	1048576	$1{,}1\ 10^{9}$	$1{,}7\ 10^{308}$

Nomes de "classes" O

classe	nome
$O(1)$	constante
$O(\lg n)$	logarítmica
$O(n)$	linear
$O(n \lg n)$	$n \log n$
$O(n^2)$	quadrática
$O(n^3)$	cúbica
$O(n^k)$ com $k \geq 1$	polinomial
$O(2^n)$	exponencial
$O(a^n)$ com $a > 1$	exponencial

Busca em vetor ordenado

PF 7.1 a 7.8

http://www.ime.usp.br/~pf/algoritmos/aulas/bub

Busca em vetor ordenado

Um vetor v[0 . . n−1] é **crescente** se

$$v[0] \leq v[1] \leq v[2] \cdots \leq v[n-1].$$

Problema: Dado um número x e um vetor crescente v[0 . . n−1] encontrar um índice m tal que v[m]==x.

Entra: x == 50

	0							7			n−1
v	10	20	25	35	38	40	44	50	55	65	99

Sai: m == 7

Busca em vetor ordenado

Entra: x == 57

	0										n−1
v	10	20	25	35	38	40	44	50	55	65	99

Sai: m == -1 (x não está em v)

Busca sequencial

```
int buscaSequencial(int x, int n, int v[])
 {
   int m = 0;
   while (/*1*/ m < n && v[m] < x) ++m;
   if (m < n && v[m] == x)
       return m;
   return -1;
 }
```

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	m										10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	m										10
v	10	20	25	35	38	40	44	50	55	65	99

	0	m									10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	m										10
v	10	20	25	35	38	40	44	50	55	65	99

	0	m									10
v	10	20	25	35	38	40	44	50	55	65	99

	0		m								10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	m										10
v	10	20	25	35	38	40	44	50	55	65	99

	0	m									10
v	10	20	25	35	38	40	44	50	55	65	99

	0		m								10
v	10	20	25	35	38	40	44	50	55	65	99

	0			m							10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

	0					m					10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

	0					m					10
v	10	20	25	35	38	40	44	50	55	65	99

	0						m				10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

	0					m					10
v	10	20	25	35	38	40	44	50	55	65	99

	0						m				10
v	10	20	25	35	38	40	44	50	55	65	99

	0							m			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 55

	0				m						10
v	10	20	25	35	38	40	44	50	55	65	99

	0					m					10
v	10	20	25	35	38	40	44	50	55	65	99

	0						m				10
v	10	20	25	35	38	40	44	50	55	65	99

	0							m			10
v	10	20	25	35	38	40	44	50	55	65	99

	0								m		10
v	10	20	25	35	38	40	44	50	55	65	99

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[m-1] < x$. ♡

x == 55

	0							m			10
v	10	20	25	35	38	40	44	50	55	65	99

A relação (i0) vale no começo da primeira iteração se supusermos que $v[-1] = -\infty$.

No início da última iteração $m \geq n$ ou $v[m] \geq x$.

Portanto, se a função devolve -1, então x não está em $v[0\,.\,.\,n-1]$

Consumo de tempo buscaSequencial

Se a execução de cada linha de código consome 1 **unidade** de tempo o consumo total é:

linha	todas as execuções da linha		
1	$=$	1	$= 1$
2	$\leq$	$n+1$	$\approx n$
3	$=$	1	$= 1$
4	$\leq$	1	≤ 1
5	$\leq$	1	≤ 1
total	$\leq$	$n+3$	$= O(n)$

Conclusão

O consumo de tempo do algoritmo buscaSequencial no pior caso é proporcional a n.

O consumo de tempo do algoritmo buscaSequencial é $O(n)$.

Busca binária

```
int buscaBinaria(int x, int n, int v[]) {
   int e, m, d;
   e = 0; d = n-1;
   while (/*1*/ e <= d) {
      m = (e + d)/2;
      if (v[m] == x) return m;
      if (v[m] < x) e = m + 1;
      else d = m - 1;
   }
   return -1;
}
```

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	e										d
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	e										d
v	10	20	25	35	38	40	44	50	55	65	99

	e					m					d
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	e										d
v	10	20	25	35	38	40	44	50	55	65	99

	e					m					d
v	10	20	25	35	38	40	44	50	55	65	99

	0						e				d
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0										10
v	10	20	25	35	38	40	44	50	55	65	99

	e										d
v	10	20	25	35	38	40	44	50	55	65	99

	e					m					d
v	10	20	25	35	38	40	44	50	55	65	99

	0						e				d
v	10	20	25	35	38	40	44	50	55	65	99

	0						e		m		d
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

							m				
	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0						e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

	0						m e	d			10
v	10	20	25	35	38	40	44	50	55	65	99

	0							e d			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0							m e d			10
v	10	20	25	35	38	40	44	50	55	65	99

Exemplo

x == 48

	0							m e d			10
v	10	20	25	35	38	40	44	50	55	65	99

	0						d	e			10
v	10	20	25	35	38	40	44	50	55	65	99

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

A relação (i0) vale no começo da primeira iteração se supusermos que $v[-1] = -\infty$ e $v[n] = +\infty$.

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

No início da última iteração quando $e > d$ nenhum elemento é "$> v[e-1]$" e "$< v[d+1]$", pois o vetor é crescente (!). Logo, x não está em $v[0\,.\,.\,n-1]$ e função devolve -1

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[e-1] < x < v[d+1]$. ♡

x == 48

	0					e		d			n-1
v	10	20	25	35	38	40	44	50	55	65	99

O valor de $d - e$ diminui a cada iteração. Portanto, se a função não encontra m tal que v[m]==x, então a função para quando $d - e < 0$.

Consumo de tempo buscaBinaria

O consumo de tempo da função `buscaBinaria` é proporcional ao número k de iterações do `while`.

No início da 1a. iteração tem-se que $d - e = n - 1 \approx n$.

Sejam

$$(e_0, d_0), (e_1, d_1), \ldots, (e_k, d_k),$$

os valores das variáveis e e d no início de cada uma das iterações. No pior caso x não está em v.
Assim, $d_{k-1} - e_{k-1} \geq 0$ **e** $d_k - e_k < 0$

Número iterações

Estimaremos o valor de k em função de $d - e$.

Note que $d_{i+1} - e_{i+1} \leq (d_i - e_i)/2$ para i=1,2,...,k-1.

Desta forma tem-se que

$$\begin{array}{lcccccc}
d_0 - e_0 & = & n-1 & < & n & & \\
d_1 - e_1 & \leq & (d_0 - e_0)/2 & < & n/2 & & \\
d_2 - e_2 & \leq & (d_1 - e_1)/2 & < & (n/2)/2 & = & n/2^2 \\
d_3 - e_3 & \leq & (d_2 - e_2)/2 & < & (n/2^2)/2 & = & n/2^3 \\
d_4 - e_4 & \leq & (d_3 - e_3)/2 & < & (n/2^3)/2 & = & n/2^4 \\
\vdots & & \vdots & & \vdots & & \vdots
\end{array}$$

Número iterações

Percebe-se que depois de cada iteração o valor de $d - e$ é reduzido pela metade.
Seja t o número inteiro tal que

$$2^t \leq n < 2^{t+1}$$

Da primeira desigualdade temos que

$$t \leq \lg n,$$

onde $\lg n$ denota o logaritmo de n na base 2.

Número iterações

Da desigualde estrita, concluímos que

$$0 \leq (d_{k-1} - e_{k-1})/2^{k-1} < \underline{n}/2^{k-1} < \underline{2^{t+1}}/2^{k-1}.$$

Assim, em particular temos que

$$1 \leq 2^{t+1}/2^{k-1}$$

ou, em outras palavras

$$k \leq t + 2.$$

Portanto , o número k de iterações é não superior a

$$t + 2 \leq \lg n + 2.$$

Conclusão

O consumo de tempo do algoritmo `buscaBinaria` no pior caso é proporcional a $\lg n$.

O consumo de tempo do algoritmo `buscaBinaria` é $O(\lg n)$.

Número de iterações

buscaSequencial	buscaBinaria
n	$\lg n$
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
262144	18
1048576	20
⋮	⋮
4294967296	32

Versão recursiva da busca binária

Para formular uma versão recursiva é necessário generalizar um pouco o problema trocando v[0 . . n−1] por v[e . . d].

```
int buscaBinaria(int x, int n, int v[])
 {
   return buscaBinariaR(x,0,n-1,v);
 }
```

Versão recursiva da busca binária

Recebe um vetor crescente v[e . . d] e devolve um índice m tal que v[m] == x. Se tal m não existe, devolve -1 .

```
int
buscaBinariaR(int x,int e,int d,int v[]) {
   int m;
   if (d < e) return -1;
   m = (e + d)/2;
   if (v[m] == x) return m;
   if (v[m] < x)
      return buscaBinariaR(x,m+1,d,v);
   return buscaBinariaR(x,e,m-1,v);
 }
```

Outra versão recursiva

Observações:

- As declarações `int v[]` e `int *v` no protótipos de funções são equivalentes. Abaixo escolhemos `int *v` apenas para deixar **mais explicito** que em ambos os casos o que está sendo passado como parâmetro é um **endereço**(!).
- As expressões "&v[m+1]" e "v+m+1" são equivalentes (=tem o memo valor =representam o mesmo endereço).
- Tem um problema . . .

Outra versão recursiva

A função abaixo não resolve o problema. . .

Por quê? Como consertar?

```
int
buscaBinariaR(int x,int n, int *v) {
  int m;
  if (n == 0) return -1;
  m = n/2;
  if (v[m] == x) return m;
  if (v[m] < x)
     return buscaBinariaR(x,n-m-1,&v[m+1]);
  return buscaBinariaR(x,m,v);
 }
```