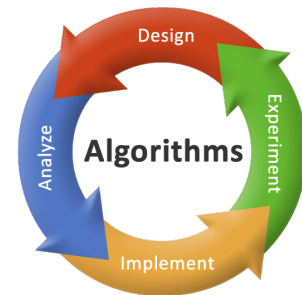


Análise de algoritmos



Fonte: <http://openclassroom.stanford.edu/>

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

AULA 16

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**.

Segmento de soma máxima

Problema (versão simplificada): Determinar a **soma máxima** de um segmento de um dado vetor $v[0..n-1]$.

Entra:

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

Sai:

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

A soma máxima é **187**.

Algoritmo café-com-leite

```
void segMax3(int v[], int n, int *e, int *d,
             int *sMax){
    int i, j, k, s;
    1 *sMax = 0; *e = *d = -1;
    2 for (i = 0; /*1*/ i < n; i++){
    3     for (j = i; j < n; j++){
    4         s = 0;
    5         for (k = i; k <= j; k++){
    6             s += v[k];
    7             if (/*2*/ s > *sMax){
    8                 *sMax = s; *e = i; *d = j;
            }
        }
    }
}
```

Correção de algoritmos

Estrutura “típica” de demonstrações da correção de algoritmos iterativos através de suas **relações invariantes** consiste em:

1. **verificar que** a relação **vale no início** da primeira iteração;
2. **demonstrar que** se a relação **vale no início** da iteração, **então** ela **vale no final** da iteração (com os papéis de alguns atores possivelmente trocados);
3. **concluir que**, se **relação vale** no início da **última iteração**, **então** a **relação junto com a condição** de parada **implicam na correção** do algoritmo.

Correção

Mais relações **invariantes**:

(i1) em /*1*/ vale que:

$$*sMax = v[*e] + v[*e+1] + v[*e+2] + \dots + v[*d];$$

(i2) em /*2*/ vale que:

$$s = v[i] + v[i+1] + v[i+2] + \dots + v[j].$$

Conclusão

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

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e..*d]$ é um segmento de soma máxima com $*e < i$. ♥

	*e		i			*d			n-1	
v	31	-41	59	26	-53	58	97	-93	-23	84

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 = 1
2	=	$n + 1 \approx n$
3	=	$(n + 1) + n + (n - 1) + \dots + 1 \leq n^2$
4	=	$n + (n - 1) + \dots + 1 \leq n^2$
5	=	$(2 + \dots + (n + 1)) + (2 + \dots + n) + \dots + 2 \leq n^3$
6	=	$(1 + \dots + n) + (1 + \dots + (n - 1)) + \dots + 1 \leq n^3$
7	=	$n + (n - 1) + (n - 2) + \dots + 1 \leq n^2$
8	≤	$n + (n - 1) + (n - 2) + \dots + 1 \leq n^2$
total	≤	$2n^3 + 4n^2 + n + 1 \sim n^3$

Resultados experimentais

segMax3		
n	tempo (s)	comentário
256	0.01	
512	0.07	
1024	0.53	
2048	4.23	
4096	33.72	
8192	269.89	> 4 min
16384	2120	> 35 min

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

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e.. *d]$ é um segmento de soma máxima com $*e < i$. ♥

	*e						*d	i		n-1
v	31	-41	59	26	-53	58	97	-93	-23	84

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	= 1
2	= n + 1	≈ n
3	= n	= n
4	= (n + 1) + n + ... + 2	≤ n ²
5	= n + (n - 1) + ... + 1	≤ n ²
6	= n + (n - 1) + ... + 1	≤ n ²
7	≤ n + (n - 1) + ... + 1	≤ n ²
total	≤ 4n ² + 2n + 1	~ n ²

Algoritmo arroz-com-feijão

```
void segMax2(int v[],int n,int *e,int *d,
             int *sMax){
    int i, j, s;
    1 *sMax = 0; *e = *d = -1;
    2 for (i = 0; /*1*/ i < n; i++) {
    3     s = 0;
    4     for (j = i; j < n; j++){
    5         s += v[j];
    6         if (/*2*/ s > *sMax){
    7             *sMax = s; *e = i; *d = j;
        }
    }
}
```

Correção

Mais relações invariante:

(i1) em /*1*/ vale que:

$$sMax = v[*e] + v[*e+1] + v[*e+2] + \dots + v[*d];$$

(i2) em /*2*/ vale que:

$$s = v[i] + v[i+1] + v[i+2] + \dots + v[j].$$

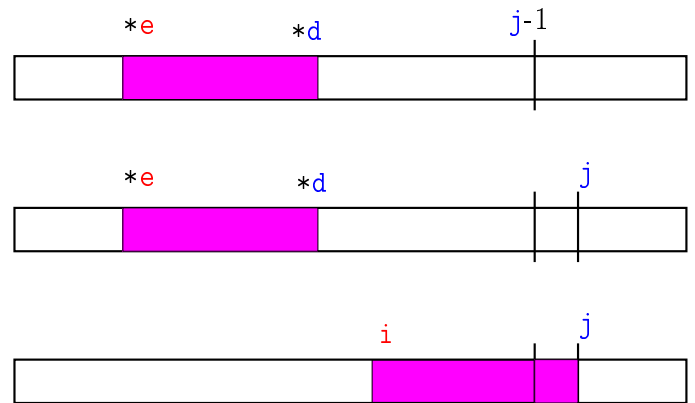
Conclusão

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

Resultados experimentais

segMax2		
n	tempo (s)	comentário
2048	0.00	
4096	0.02	
8192	0.06	
16384	0.23	
32768	0.92	
65536	3.71	
131072	14.83	
262144	59.34	≈ 1 min
524288	237.26	≈ 4 min
1048576	957.40	≈ 16 min

Nova ideia (indutiva)



Implementação ingênua

Determina um segmento de soma máxima de $v[0..n-1]$.

```
void segMaxI(int v[], int n, int *e, int *d,
             int *sMax){
    int i, j, k, sk, s;
    1 s = *sMax = 0; *e = *d = -1;
```

Implementação ingênua

```
2 for (i = j = 0; /*1*/ j < n; j++) {
3     s = sk = v[j]; i = j;
4     for (k = j-1; k >= 0; k--) {
5         sk += v[k];
6         if (sk > s){ s = sk; i = k; }
7     }
8     if (/*2*/ s > *sMax){
9         *sMax = s; *e = i; *d = j;
10    }
11 }
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e..*d]$ é segmento de soma máxima com $*d \leq j - 1$. ♥

	*e	*d	j	n-1						
v	31	-41	59	26	-53	58	97	-93	-23	84

Mais uma relação **invariante**:

(i1) em /*1*/ vale que:

$$*sMax = v[*e] + v[*e+1] + v[*e+2] + \dots + v[*d].$$

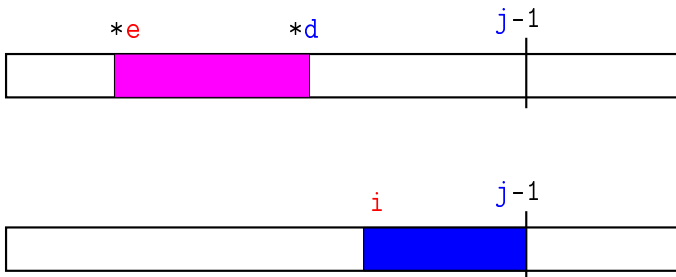
Mais relações invariantes

Em /*1*/ vale que:

(i2) $v[i..j-1]$ é segmento de soma máxima com término em $j-1$ e contendo $v[j-1]$;

(i3) $s = v[i] + v[i+1] + v[i+2] + \dots + v[j-1]$;

Invariantes (i0) e (i2)



Ainda mais relações invariantes

Em /*2*/ vale que:

(i2') $v[i..j]$ é segmento de soma máxima com término em j e contendo $v[j]$;

(i3') $s = v[i] + v[i+1] + v[i+2] + \dots + v[j]$;

(i4) para $k = i, i+1, \dots, j$, vale que

$$v[k] + v[k+1] + \dots + v[j-1] \geq 0;$$

(i5) para $k = 0, 1, \dots, i-1$, vale que

$$v[k] + v[k+1] + \dots + v[i-1] < 0;$$

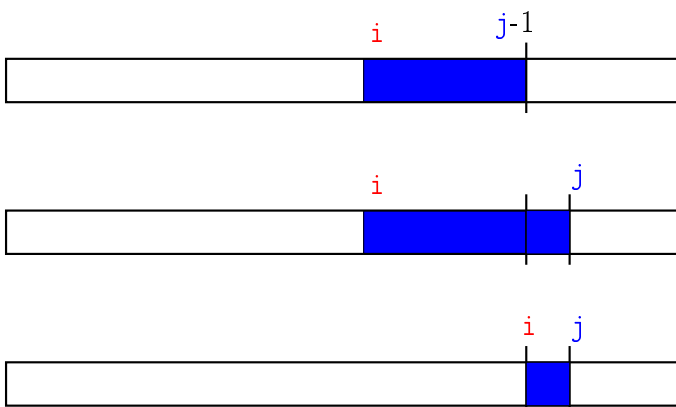
Consumo de tempo segMaxI

linha	todas as execuções da linha	
1	= 1	= 1
2	= $n + 1$	$\approx n$
3	= n	= n
4	= $1 + 2 + \dots + n$	$\leq n^2$
5	= $1 + 2 + \dots + n-1$	$\leq n^2$
6	= $1 + 2 + \dots + n-1$	$\leq n^2$
7	= n	= n
8	$\leq n$	= n
total	$\leq 3n^2 + 4n + 1$	$\sim n^2$

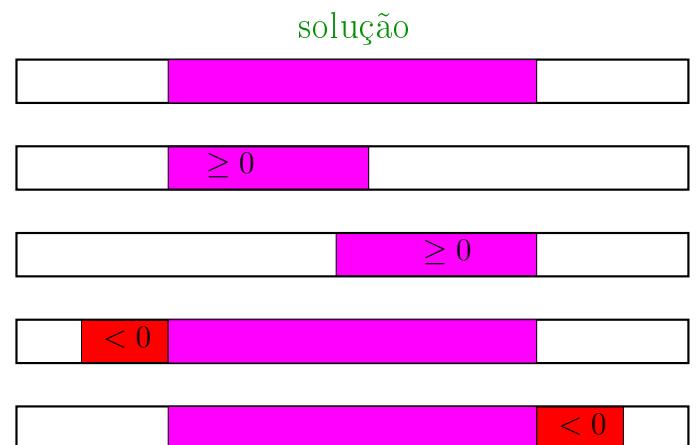
Conclusão

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

Segmento de soma máxima contendo $v[j]$



Cara da solução



Conclusões

Se $v[*e..*d]$ é um **seg de soma máx.** então:

- ▶ para $k = *e, *e+1, \dots, *d$, vale que $v[*e] + v[*e+1] + \dots + v[k] \geq 0$;
- ▶ para $k = *e, *e+1, \dots, *d$, vale que $v[k] + v[k+1] + \dots + v[*d] \geq 0$;
- ▶ para $k = 1, 2, \dots, *e-1$, vale que $v[k] + v[k+1] + \dots + v[*e-1] \leq 0$;
- ▶ para todo $k = *d+1, *d+2, \dots, n$, vale $v[*d+1] + v[*d+2] + \dots + v[k] \leq 0$.

Mais conclusões

Se $v[i..j]$ é um segmento de soma máxima terminando em j e **contendo** $v[j]$ então:

- ▶ para $k = i, i+1, \dots, j-1$, vale que $v[i] + v[i+1] + \dots + v[k] \geq 0$;
- ▶ para $k = 0, 1, 2, \dots, i-1$, vale que $v[k] + v[k+1] + \dots + v[i-1] \leq 0$.

Algoritmo linear

```
void segMax(int v[], int n, int *e, int *d, int *sMax){
    int i, j, s, sMax;
    1 s = sMax = 0; *e = *d = -1;
    2 for (i = j = 0; /*1*/ j < n; j++) {
    3   if (s < 0){
    4     /*3*/ i = j; s = v[j];
    5   } else s += v[j];
    6   if (/*2*/ s > sMax){
    7     sMax = s; *e = i; *d = j;
    }
  }
}
```

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e..*d]$ é segmento de soma máxima com $*d \leq j - 1$. ♥

		*e	*d	j					n-1	
v	31	-41	59	26	-53	58	97	-93	-23	84

Mais uma relação **invariante**:

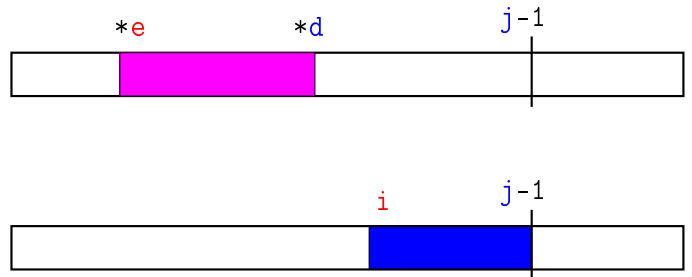
(i1) em /*1*/ vale que:
 $sMax = v[*e] + v[*e+1] + v[*e+2] + \dots + v[*d]$.

Mais relações invariantes

Em /*1*/ vale que:

- (i2) $v[i..j-1]$ é **segmento de soma máxima** com término em $j-1$ e **contendo** $v[j-1]$;
- (i3) $s = v[i] + v[i+1] + v[i+2] + \dots + v[j-1]$;

Invariantes (i0) e (i2)



Ainda mais relações invariantes

Em /*2*/ vale que:

(i2) $v[i..j]$ é segmento de soma máxima com término em j e contendo $v[j]$;

(i3) $s = v[i] + v[i+1] + v[i+2] + \dots + v[j]$;

(i4) para $k = i, i+1, \dots, j$, vale que

$$v[k] + v[k+1] + \dots + v[j-1] \geq 0;$$

(i5) para $k = 0, 1, \dots, i-1$, vale que

$$v[k] + v[k+1] + \dots + v[i-1] < 0;$$

◀ ▶ ↻ 🔍

Consumo de tempo

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	= $n + 1$	$\approx n$
3	= n	= n
4-5	= n	= n
6	= n	= n
7	$\leq n$	= n
total	$\leq 5n + 2$	$\sim n$

◀ ▶ ↻ 🔍

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

◀ ▶ ↻ 🔍

Ainda mais relações invariantes

Em /*3*/ vale que:

(i6) $s = v[i] + v[i+1] + \dots + v[j-1] < 0$

As relações invariantes (i4) e (i6) implicam que em /*3*/:

(i7) para $k = 0, 1, 2, \dots, j-1$, vale que

$$v[k] + v[k+1] + \dots + v[j-1] < 0;$$

◀ ▶ ↻ 🔍

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:** **segMax2**, **segMax**
Algumas vezes usamos espaço para armazenar resultados a fim de evitar recomputá-los.
- ▶ **Algoritmos incrementais/varredura:** **segMax**
Solução de um subproblema é estendida a uma solução do problema original.
- ▶ **Algoritmo ótimo:** **segMax**
Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível.

◀ ▶ ↻ 🔍