AULA 11

Análise de algoritmo

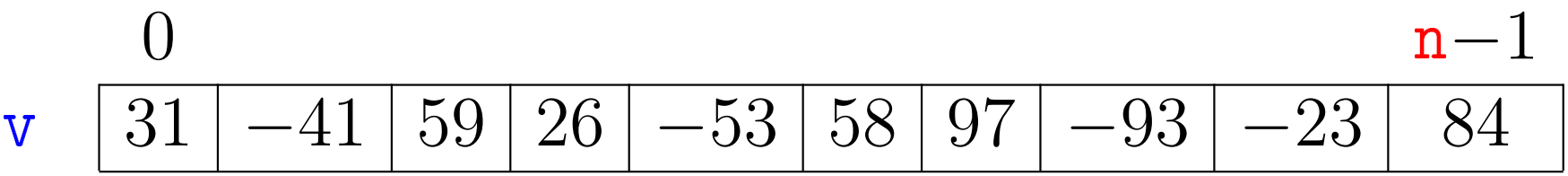
Programming Pearls: Algorithm Design Techniques,
Jon Bentley, Addison-Wesley, 1986

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:



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; /*2*/ k <= j; k++)
6           s += v[k];
7        if (s > *smax){
8           *smax = s; *e = i; *d = j;
         }
      }
}
```

Correção de algoritmos

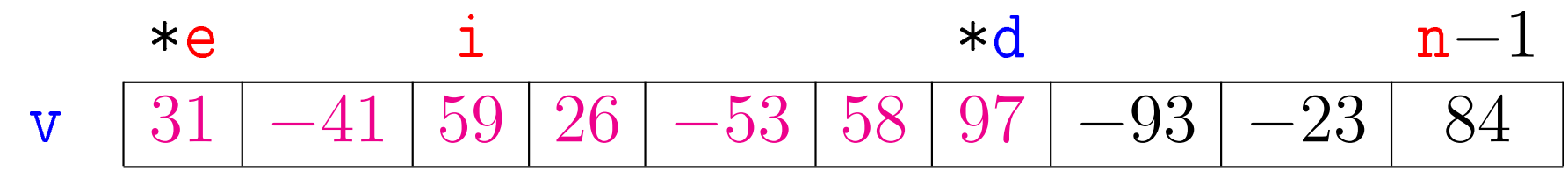
Estrutura "típica" de demonstrações da correção de algoritmos iterativos através de suas **relações invariantes** consiste em:

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 a relação junto com a condição de parada implicam na correção do algoritmo.

Correção

Relação **invariante** chave:

(i0) em `/*1*/` vale que: $\mathtt{v}[*\mathtt{e} \,..\, *\mathtt{d}]$ é um segmento de soma máxima com $*\mathtt{e} < \mathtt{i}$. ♡



Correção

Mais relações **invariantes**:

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

$$*\texttt{smax} = \texttt{v}[*\texttt{e}] + \texttt{v}[*\texttt{e}+1] + \texttt{v}[*\texttt{e}+2] + \cdots + \texttt{v}[*\texttt{d}];$$

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

$$\texttt{s} = \texttt{v}[\texttt{i}] + \texttt{v}[\texttt{i}+1] + \texttt{v}[\texttt{i}+2] + \cdots + \texttt{v}[\texttt{k}-1].$$

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

Conclusão

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

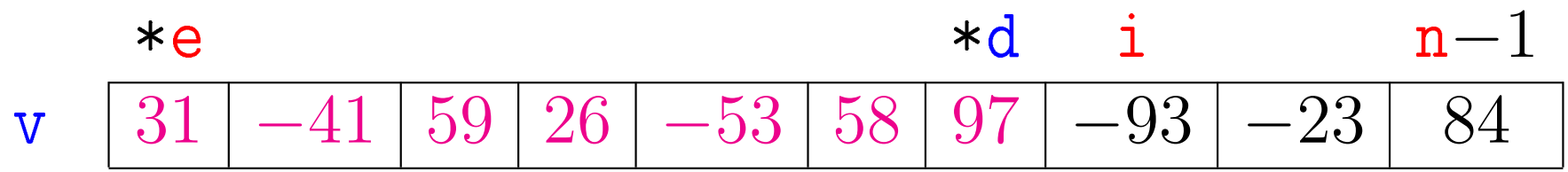
Algoritmo arroz-com-feijão

```
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

Relação **invariante** chave:

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



Correção

Mais relações invariante:

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

$$\texttt{smax} = \texttt{v}[*\texttt{e}] + \texttt{v}[*\texttt{e}+1] + \texttt{v}[*\texttt{e}+2] + \cdots + \texttt{v}[*\texttt{d}];$$

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

$$\texttt{s} = \texttt{v}[\texttt{i}] + \texttt{v}[\texttt{i}+1] + \texttt{v}[\texttt{i}+2] + \cdots + \texttt{v}[\texttt{j}].$$

Consumo de tempo segMax2

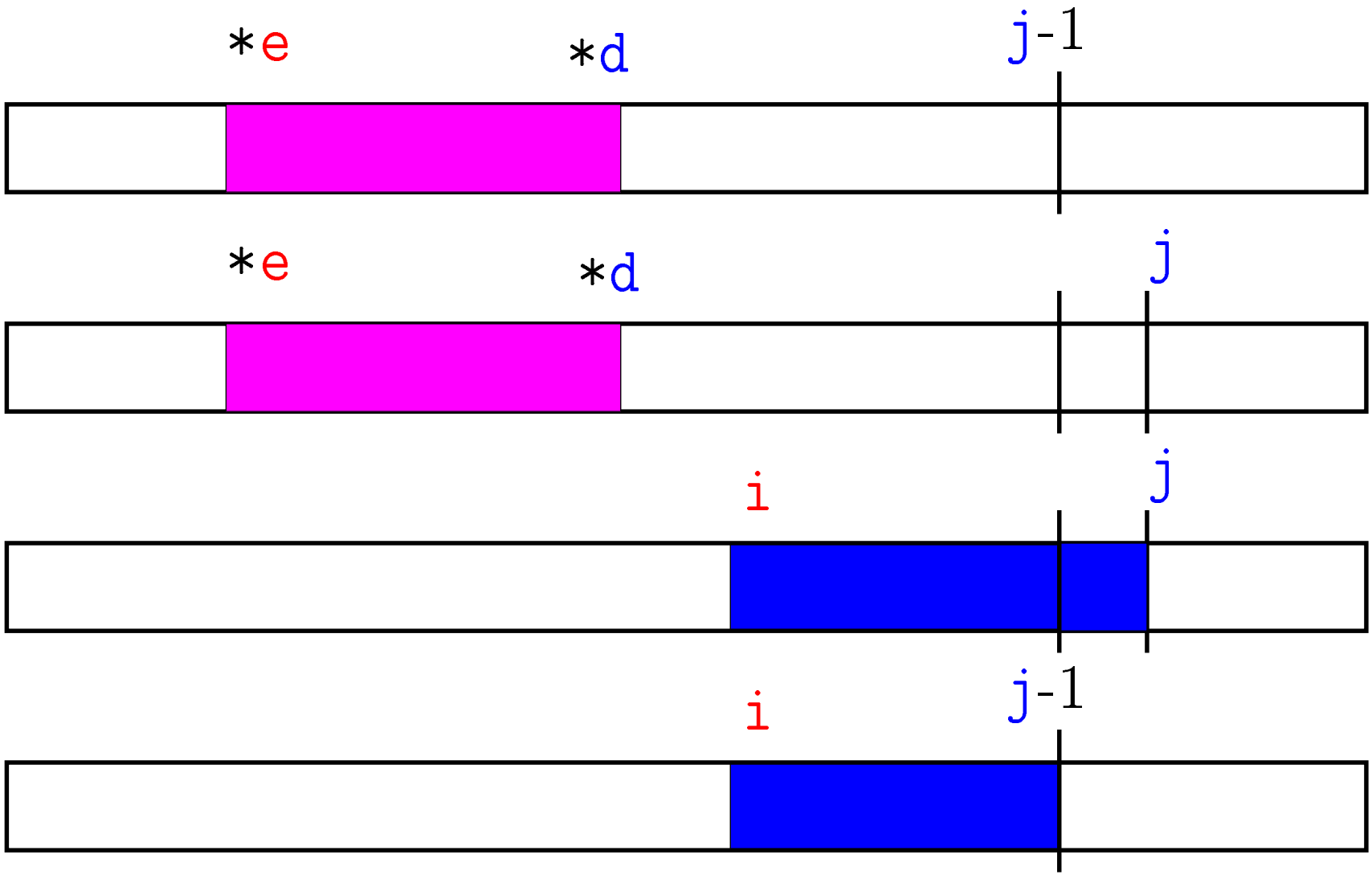
Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

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

Conclusão

O consumo de tempo do algoritmo `segMax2` é proporcional a $\mathtt{n}^2$.

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 *smax = 0; *e = *d = -1;
```

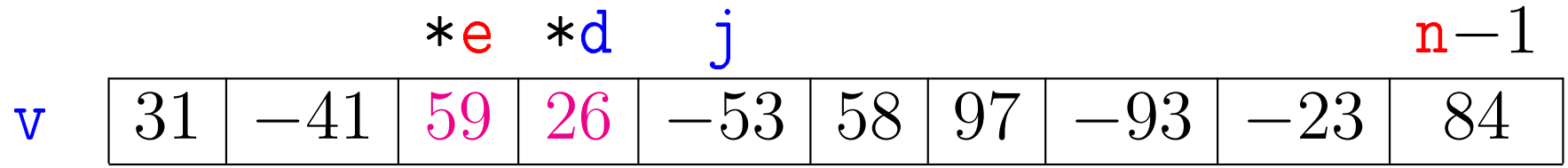
Implementação ingênua

```
2 for(j = 0; /*1*/ j < n; j++) {
3     s = sk = 0; i = j + 1;
4     for(k = j; k >= 0; k--) {
5         sk += v[k];
6         if (sk > s){ s = sk; i = k; }
      }
7     if (/*2*/ s > *smax){
8         *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 ≤ j − 1. ♡



Mais uma relação **invariante**:

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

$$\texttt{smax} = \texttt{v}[*\texttt{e}] + \texttt{v}[*\texttt{e}+1] + \texttt{v}[*\texttt{e}+2] + \cdots + \texttt{v}[*\texttt{d}].$$

Mais relações invariantes

Em /*2*/ vale que:

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

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

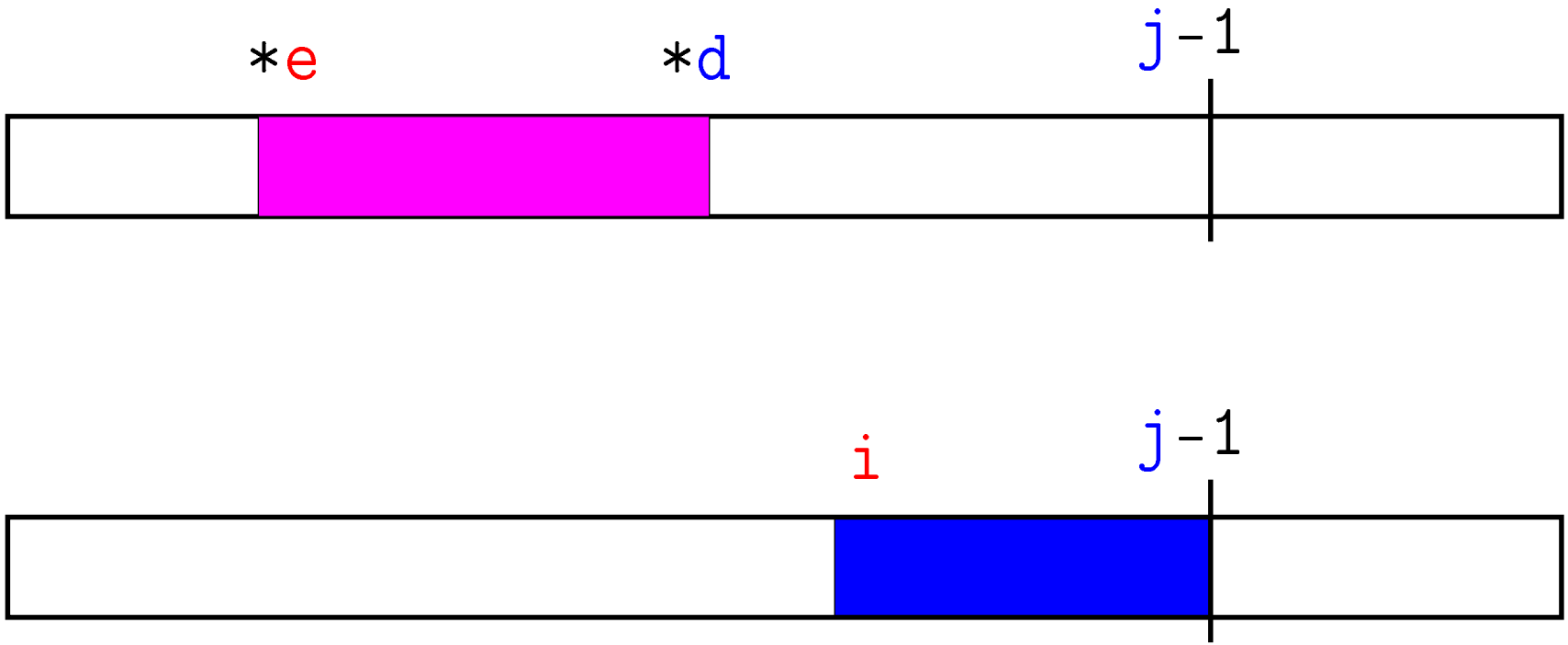
(i4) para $k = i, i+1, \ldots, j$, vale que

$$v[k] + v[k+1] + \cdots + v[j] \geq 0\,;$$

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

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

Invariantes (i0) e (i2)



Consumo de tempo segMaxI

linha	todas as execuções da linha		
1	$=$	1	$= 1$
2	$=$	$n + 1$	$\approx n$
3	$=$	n	$= n$
4	$=$	$2 + 3 + \ldots + (n + 1)$	$\leq n^2$
5	$=$	$1 + 2 + \ldots + n$	$\leq n^2$
6	$=$	$1 + 2 + \ldots + n$	$\leq n^2$
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 $\mathtt{n}^2$.