Ordenação em tempo linear

CLRS cap 8

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,.\,.\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

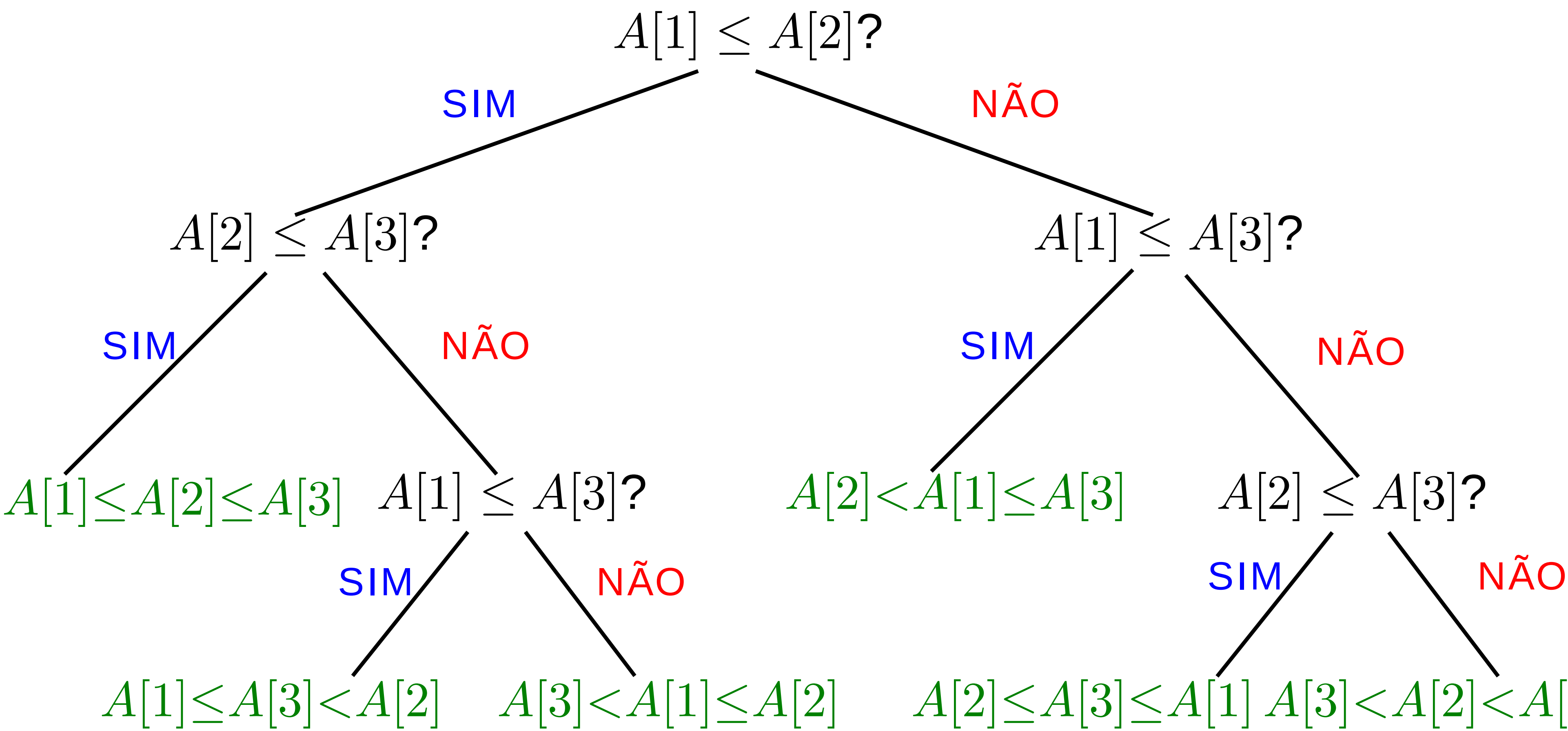
NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações
é uma "árvore de decisão".

Exemplo

ORDENA-POR-INSERÇÃO $(A[1\,..\,3])$:



Limite inferior

Considere uma árvore de decisão para $A[1 \,.\,.\, n]$.

Limite inferior

Considere uma árvore de decisão para $A[1 \mathinner{.\,.} n]$.

Número de comparações, no pior caso?

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1,\ldots,n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h. A afirmação vale para $h = 0$.
Suponha que a afirmação vale para toda árvore binária de altura menor que h, para $h \geq 1$.
O número de folhas de uma árvore de altura h é a soma do número de folhas de suas sub-árvores, que têm altura $\leq h-1$. Logo, o número de folhas de uma árvore de altura h é não superior a

$$2 \times 2^{h-1} = 2^h.$$

Limite inferior

Assim, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 \;=\; \prod_{i=0}^{n-1} (n-i)(i+1) \;\geq\; \prod_{i=1}^{n} n \;=\; n^n$$

Portanto,

$$h \;\geq\; \lg(n!) \;\geq\; \frac{1}{2}\, n \lg n.$$

Alternativamente, a fórmula de Stirling diz que

$$n! \;=\; \sqrt{2\pi\, n}\Big(\frac{n}{e}\Big)^n\big(1+\Theta(\frac{1}{n})\big).$$

Disso, temos que $h \;\geq\; \lg(n!) \;\geq\; \lg\left(\frac{n}{e}\right)^n \;=\; n(\lg n - \lg e)$.

Conclusão

Todo algoritmo de ordenação baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Devolve um vetor $B[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

Counting Sort

Recebe inteiros n e k, e um vetor $A[1\,..\,n]$ onde cada elemento é um inteiro entre 1 e k.

Devolve um vetor $B[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

COUNTINGSORT(A, n)

para $i \leftarrow 1$ **até** k **faça**
  $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $C[A[j]] \leftarrow C[A[j]] + 1$
para $i \leftarrow 2$ **até** k **faça**
  $C[i] \leftarrow C[i] + C[i-1]$
para $j \leftarrow n$ **decrescendo até** 1 **faça**
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
devolva B

Consumo de tempo

linha	consumo na linha
1	$\Theta(k)$
2	$O(k)$
3	$\Theta(n)$
4	$O(n)$
5	$\Theta(k)$
6	$O(k)$
7	$\Theta(n)$
8	$O(n)$
9	$O(n)$
10	$\Theta(1)$
total	????

Consumo de tempo

linha	consumo na linha
1	$\Theta(k)$
2	$O(k)$
3	$\Theta(n)$
4	$O(n)$
5	$\Theta(k)$
6	$O(k)$
7	$\Theta(n)$
8	$O(n)$
9	$O(n)$
10	$\Theta(1)$
total	$\Theta(k+n)$

Counting Sort

COUNTINGSORT(A, n)

para $i \leftarrow 1$ **até** k **faça**
  $C[i] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $C[A[j]] \leftarrow C[A[j]] + 1$
para $i \leftarrow 2$ **até** k **faça**
  $C[i] \leftarrow C[i] + C[i-1]$
para $j \leftarrow n$ **decrescendo até** 1 **faça**
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
devolva B

Consumo de tempo: $\Theta(k + n)$

Se $k = \mathrm{O}(n)$, o consumo de tempo é $\Theta(n)$.

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

RADIXSORT(A, n, d)
1 **para** $i \leftarrow 1$ **até** d **faça**
2 ORDENE(A, n, i)

Radix Sort

Algoritmo usado para ordenar

- inteiros não-negativos com d dígitos
- cartões perfurados
- registros cuja chave tem vários campos

dígito 1: menos significativo
dígito d: mais significativo

RADIXSORT(A, n, d)
1 **para** $i \leftarrow 1$ **até** d **faça**
2 ORDENE(A, n, i)

ORDENE(A, n, i): ordena $A[1 \,..\, n]$ pelo i-ésimo dígito dos números em A por meio de um algoritmo estável.

Consumo de tempo

Depende do algoritmo ORDENE.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Consumo de tempo

Depende do algoritmo ORDENE.

Se cada dígito é um inteiro de 1 a k,
então podemos usar o COUNTINGSORT.

Neste caso, o consumo de tempo é $\Theta(d(k+n))$.

Se d é limitado por uma constante (ou seja, se $d = \mathrm{O}(1)$)
e $k = \mathrm{O}(n)$, então o consumo de tempo é $\Theta(n)$.

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Bucket Sort

Recebe um inteiro n e um vetor $A[1 \,.\,.\, n]$ onde cada elemento é um número no intervalo $[0, 1)$.

Devolve um vetor $C[1 \,.\,.\, n]$ com os elementos de $A[1 \,.\,.\, n]$ em ordem crescente.

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0,1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

BUCKETSORT(A, n)
para $i \leftarrow 0$ **até** $n-1$ **faça**
  $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
  INSIRA$(B[\lfloor n\,A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n-1$ **faça**
  ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**
  $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
  INSIRA$(B[\lfloor n\,A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
  ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

INSIRA(p, x): insere x na lista apontada por p

ORDENELISTA(p): ordena a lista apontada por p

CONCATENE(B, n): devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

Bucket Sort

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Se os números em $A[1 \mathinner{.\,.} n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o consumo de tempo esperado é linear em n.

Bucket Sort

Se os números em $A[1\,.\,.\,n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o número esperado de elementos de $A[1\,.\,.\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Bucket Sort

Se os números em $A[1\,..\,n]$ forem uniformemente distribuídos no intervalo $[0, 1)$, então o número esperado de elementos de $A[1\,..\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Logo, o consumo de tempo esperado para ordenar cada uma das listas $B[i]$ é linear $\Theta(1)$.

Assim, o consumo de tempo esperado do algoritmo BUCKETSORT neste caso é $\Theta(n)$.

Bucket Sort

Se os números em $A[1\,..\,n]$ forem uniformemente distribuídos no intervalo $[0,1)$, então o número esperado de elementos de $A[1\,..\,n]$ em cada lista $B[i]$ é $\Theta(1)$.

Logo, o consumo de tempo esperado para ordenar cada uma das listas $B[i]$ é linear $\Theta(1)$.

Assim, o consumo de tempo esperado do algoritmo BUCKETSORT neste caso é $\Theta(n)$.

Exercícios

Exercício 10.A

Desenhe a árvore de decisão para o SELECTIONSORT aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 10.B [CLRS 8.1-1]

Qual o menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

Exercício 10.C [CLRS 8.1-2]

Mostre que $\lg(n!) = \Omega(n \lg n)$ sem usar a fórmula de Stirling. Sugestão: Calcule $\sum_{k=n/2}^{n} \lg k$. Use as técnicas de CLRS A.2.

Exercícios

Exercício 10.D [CLRS 8.2-1]
Simule a execução do COUNTINGSORT usando como entrada o vetor
$A[1\,..\,11] = \langle 7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3 \rangle$.

Exercício 10.E [CLRS 8.2-2]
Mostre que o COUNTINGSORT é estável.

Exercício 10.F [CLRS 8.2-3]
Suponha que o **para** da linha 7 do COUNTINGSORT é substituído por

7 **para** $j \leftarrow 1$ **até** n **faça**

Mostre que o ainda funciona. O algoritmo resultante continua estável?

Bucketsort

CLRS sec 8.4

Bucket Sort

Recebe um inteiro n e um vetor $A[1\,..\,n]$ onde cada elemento é um número no intervalo $[0,1)$.

Devolve um vetor $C[1\,..\,n]$ com os elementos de $A[1\,..\,n]$ em ordem crescente.

BUCKETSORT(A, n)
para $i \leftarrow 0$ **até** $n-1$ **faça**
  $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
  INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n-1$ **faça**
  ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

Bucket Sort

BUCKETSORT(A, n)
para $i \leftarrow 0$ **até** $n-1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n-1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

INSIRA(p, x): insere x na lista apontada por p

ORDENELISTA(p): ordena a lista apontada por p

CONCATENE(B, n): devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0,1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0,1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0,1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} \quad = \quad \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Consumo de tempo

Suponha que os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

Consumo de tempo

Suponha que os números em $A[1 \mathinner{.\,.} n]$ são uniformemente distribuídos no intervalo $[0, 1)$.

Suponha que o ORDENELISTA seja o INSERTIONSORT.

Seja X_i o número de elementos na lista $B[i]$.

Seja

$$X_{ij} \quad = \quad \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Observe que $X_i = \sum_j X_{ij}$.

Y_i: número de comparações para ordenar a lista $B[i]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq X_i{}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[X_i{}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j X_{ij}{}^2 + \sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \mathrm{E}[\sum_j {X_{ij}}^2] + \mathrm{E}[\sum_j \sum_{k \neq j} X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$.

Logo $\mathrm{E}[Y_i] \leq \mathrm{E}[{X_i}^2] = \mathrm{E}[(\sum_j X_{ij})^2]$.

$$\begin{aligned} \mathrm{E}[(\sum_j X_{ij})^2] &= \mathrm{E}[\sum_j \sum_k X_{ij} X_{ik}] \\ &= \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}] \end{aligned}$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Consumo de tempo

X_i: número de elementos na lista $B[i]$

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

Y_i: número de comparações para ordenar a lista $B[i]$.

Observe que $Y_i \leq {X_i}^2$. Ademais,

$$\mathrm{E}[Y_i] \leq \sum_j \mathrm{E}[{X_{ij}}^2] + \sum_j \sum_{k \neq j} \mathrm{E}[X_{ij} X_{ik}].$$

Observe que ${X_{ij}}^2$ é uma variável aleatória binária. Vamos calcular sua esperança:

$$\mathrm{E}[{X_{ij}}^2] = \Pr[{X_{ij}}^2 = 1] = \Pr[X_{ij} = 1] = \frac{1}{n}.$$

Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Consumo de tempo

Para calcular $\mathrm{E}[X_{ij}X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.

Portanto, $\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}]$.

Ademais, $\mathrm{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$.

Logo,

$$
\begin{aligned}
\mathrm{E}[Y_i] &\leq \sum_j \frac{1}{n} + \sum_j \sum_{k \neq j} \frac{1}{n} \\
&= \frac{n}{n} + n(n-1)\frac{1}{n^2} \\
&= 1 + (n-1)\frac{1}{n} \\
&= 2 - \frac{1}{n}.
\end{aligned}
$$

Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que Y é o número de comparações realizadas pelo BUCKETSORT no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BUCKETSORT.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

Consumo de tempo

Agora, seja $Y = \sum_i Y_i$.

Note que Y é o número de comparações realizadas pelo BUCKETSORT no total.

Assim $\mathrm{E}[Y]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do BUCKETSORT.

Mas então $\mathrm{E}[Y] = \sum_i \mathrm{E}[Y_i] \leq 2n - 1 = \mathrm{O}(n)$.

O consumo de tempo esperado do BUCKETSORT quando os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0, 1)$ é $\mathrm{O}(n)$.

Programação dinâmica

CLRS cap 15

= “recursão–com–tabela”
= transformação inteligente de recursão em iteração

Programação dinâmica

"Dynamic programming is a fancy name for divide-and-conquer with a table. Instead of solving subproblems recursively, solve them sequentially and store their solutions in a table. The trick is to solve them in the right order so that whenever the solution to a subproblem is needed, it is already available in the table. Dynamic programming is particularly useful on problems for which divide-and-conquer appears to yield an exponential number of subproblems, but there are really only a small number of subproblems repeated exponentially often. In this case, it makes sense to compute each solution the first time and store it away in a table for later use, instead of recomputing it recursively every time it is needed."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n:

FIBO-REC (n)
se $n \leq 1$
  então devolva n
  senão $a \leftarrow$ FIBO-REC $(n-1)$
    $b \leftarrow$ FIBO-REC $(n-2)$
    devolva $a + b$

Consumo de tempo

```
FIBO-REC (n)
    se n ≤ 1
        então devolva n
        senão a ← FIBO-REC (n − 1)
              b ← FIBO-REC (n − 2)
              devolva a + b
```

n	16	32	40	41	42	43	44	45	47
tempo	0.002	0.06	2.91	4.71	7.62	12.37	19.94	32.37	84.50

tempo em segundos.

$F_{47} = 2971215073$

Consumo de tempo

$T(n) :=$ número de somas feitas por FIBO-REC (n)

linha	número de somas
1-2	$= 0$
3	$= T(n-1)$
4	$= T(n-2)$
5	$= 1$

$$T(n) = T(n-1) + T(n-2) + 1$$

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Solução: $T(n) > (3/2)^n$ para $n \geq 6$.

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
$(3/2)^n$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.44

Recorrência

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e $T(7) = 20 > 18 > (3/2)^7$.
Se $n \geq 8$, então

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\overset{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2+1)\,(3/2)^{n-2} + 1 \\
&> (5/2)\,(3/2)^{n-2} \\
&> (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n \,.
\end{aligned}$$

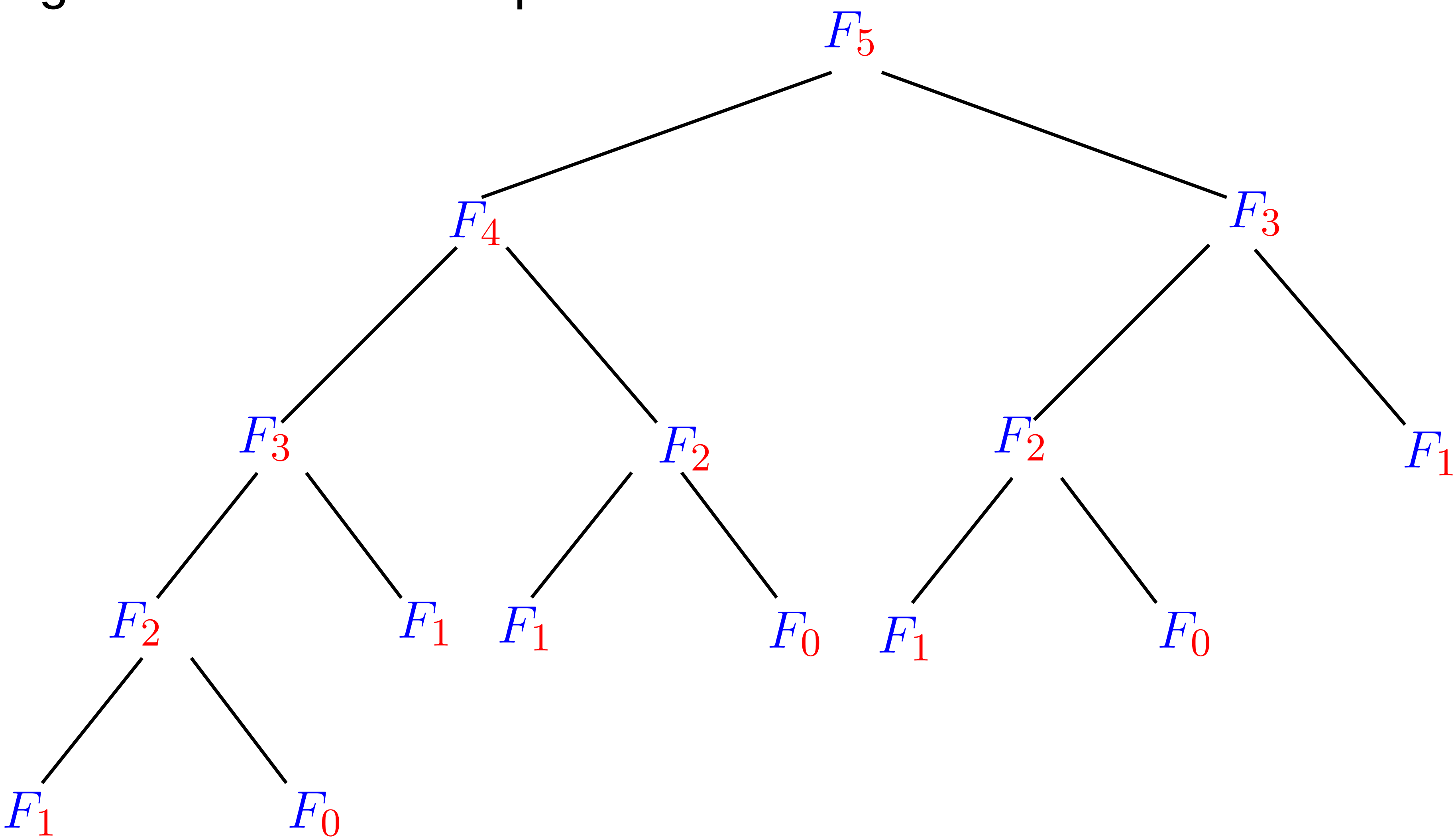
Logo, $T(n)$ é $\Omega((3/2)^n)$.

Verifique que $T(n)$ é $\mathrm{O}(2^n)$.

Consumo de tempo

Consumo de tempo é exponencial.

Algoritmo resolve subproblemas muitas vezes.



Resolve subproblemas muitas vezes

```
FIBO-REC(5)
  FIBO-REC(4)
    FIBO-REC(3)
      FIBO-REC(2)
        FIBO-REC(1)
        FIBO-REC(0)
      FIBO-REC(1)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
  FIBO-REC(3)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
    FIBO-REC(1)
```

FIBO-REC(5) = 5

Resolve subproblemas muitas vezes

```
FIBO-REC(8)
  FIBO-REC(7)
    FIBO-REC(6)
      FIBO-REC(5)
        FIBO-REC(4)
          FIBO-REC(3)
            FIBO-REC(2)
              FIBO-REC(1)
              FIBO-REC(0)
            FIBO-REC(1)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
  FIBO-REC(6)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
    FIBO-REC(4)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
      FIBO-REC(2)
        FIBO-REC(1)
        FIBO-REC(0)
```

Algoritmo de programação dinâmica

FIBO (n)
$f[0] \leftarrow 0$
$f[1] \leftarrow 1$
para $i \leftarrow 2$ **até** n **faça**
    $f[i] \leftarrow f[i-1] + f[i-2]$
devolva $f[n]$

Note a tabela $f[0\,..\,n-1]$.

f					$\star$	$\star$	$??$			

Consumo de tempo é $\Theta(n)$.

Algoritmo de programação dinâmica

Versão com economia de espaço.

```
FIBO (n)
  se n = 0 então devolva 0
  f_ant ← 0
  f_atual ← 1
  para i ← 2 até n faça
      f_prox ← f_atual + f_ant
      f_ant ← f_atual
      f_atual ← f_prox
  devolva f_atual
```

Versão recursiva eficiente

MEMOIZED-FIBO (f, n)

para $i \leftarrow 0$ **até** n **faça**
    $f[i] \leftarrow -1$
devolva LOOKUP-FIBO (f, n)

LOOKUP-FIBO (f, n)

se $f[n] \geq 0$
    então devolva $f[n]$
se $n \leq 1$
    então $f[n] \leftarrow n$
    senão $f[n] \leftarrow$ LOOKUP-FIBO$(f, n-1)$
                        $+$ LOOKUP-FIBO$(f, n-2)$
devolva $f[n]$

Não recalcula valores de f.

Linha de produção

LINHA-de-PRODUÇÃO (a, t, e, x, n)

$c[1,1] \leftarrow e[1] + a[1,1]$
$c[2,1] \leftarrow e[2] + a[2,1]$
para $j \leftarrow 2$ **até** n **faça**
  se $c[1, j-1] + a[1, j] \leq c[2, j-1] + t[2, j-1] + a[1, j]$
    então $c[1, j] \leftarrow c[1, j-1] + a[1, j]$
      $l[1, j] \leftarrow 1$
    senão $c[1, j] \leftarrow c[2, j-1] + t[2, j-1] + a[1, j]$
      $l[1, j] \leftarrow 2$
  se $c[2, j-1] + a[2, j] \leq c[1, j-1] + t[1, j-1] + a[2, j]$
    então $c[2, j] \leftarrow c[2, j-1] + a[2, j]$
      $l[2, j] \leftarrow 2$
    senão $c[2, j] \leftarrow c[1, j-1] + t[1, j-1] + a[2, j]$
      $l[2, j] \leftarrow 1$
$\ldots$

Linha de produção

LINHA-de-PRODUÇÃO (a, t, e, x, n)

. . .

```
se c[1,n] + x[1] ≤ c[2,n] + x[2]
    então c* ← c[1,n] + x[1]
          l* ← 1
    senão c* ← c[2,n] + x[2]
          l* ← 2
devolva c*, l*, l
```

Linha de produção

LINHA-de-PRODUÇÃO (a, t, e, x, n)

...

se $c[1, n] + x[1] \leq c[2, n] + x[2]$

  então $c^* \leftarrow c[1, n] + x[1]$

    $l^* \leftarrow 1$

  senão $c^* \leftarrow c[2, n] + x[2]$

    $l^* \leftarrow 2$

devolva c^*, l^*, l

O tempo desse algoritmo é $\Theta(n)$.

Linha de produção

IMPRIME-MÁQUINAS (l, l^*, n)

```
i ← l*
imprima "linha i, máquina n"
para j ← n até 2 faça
    i ← l[i, j]
    imprima "linha i, máquina j − 1"
```

Linha de produção

IMPRIME-MÁQUINAS (l, l^*, n)

$i \leftarrow l^*$
imprima "linha i, máquina n"
para $j \leftarrow n$ **até** 2 **faça**
    $i \leftarrow l[i, j]$
    imprima "linha i, máquina $j - 1$"

O tempo desse algoritmo é $\Theta(n)$.

Programação dinâmica

CLRS 15.2–15.3

= “recursão–com–tabela”
= transformação inteligente de recursão em iteração

Multiplicação iterada de matrizes

Se A é $p \times q$ e B é $q \times r$ então AB é $p \times r$.

$(AB)[i,j] = \sum_k A[i,k]\, B[k,j]$

MULT-MAT (p, A, q, B, r)

    para $i \leftarrow 1$ **até** p **faça**
        para $j \leftarrow 1$ **até** r **faça**
            $AB[i,j] \leftarrow 0$
            para $k \leftarrow 1$ **até** q **faça**
                $AB[i,j] \leftarrow AB[i,j] + A[i,k] \cdot B[k,j]$

Número de multiplicações escalares = $p \cdot q \cdot r$

Multiplicação iterada

Problema: Encontrar número mínimo de multiplicações escalares necessário para calcular produto $A_1 A_2 \cdots A_n$.

$$\begin{array}{ccccccccccc} p[0] & & p[1] & & p[2] & \ldots & p[n-1] & & p[n] \\ & A_1 & & A_2 & & \ldots & & A_n & \end{array}$$

cada A_i é $p[i-1] \times p[i]$ ($A_i[1\,..\,p[i-1], 1\,..\,p[i]]$)

Exemplo: $A_1 \cdot A_2 \cdot A_3$

$$\begin{array}{ccccccc} 10 & A_1 & 100 & A_2 & 5 & A_3 & 50 \end{array}$$

$((A_1\ A_2)\ A_3)$	7500	multiplicações escalares
$(A_1\ (A_2\ A_3))$	75000	multiplicações escalares

Soluções ótimas contêm soluções ótimas

Se

$$(A_1 A_2)\,(A_3((A_4 A_5) A_6))$$

é ordem ótima de multiplicação então

$$(A_1 A_2) \quad \text{e} \quad (A_3((A_4 A_5) A_6))$$

também são ordens ótimas.

Soluções ótimas contêm soluções ótimas

Se

$$(A_1 A_2)\,(A_3((A_4 A_5) A_6))$$

é ordem ótima de multiplicação então

$$(A_1 A_2) \quad \text{e} \quad (A_3((A_4 A_5) A_6))$$

também são ordens ótimas.

Decomposição: $(A_i \cdots A_k)\,(A_{k+1} \cdots A_j)$

$m[i,j] =$ número mínimo de multiplicações escalares para calcular $A_i \cdots A_j$

Recorrência

$m[i, j] =$ número mínimo de multiplicações escalares para calcular $A_i \cdots A_j$

se $i = j$ então $m[i, j] = 0$

se $i < j$ então

$$m[i, j] = \min_{i \le k < j} \{ m[i, k] + p[i-1]p[k]p[j] + m[k+1, j] \}$$

Exemplo:

$$m[3, 7] = \min_{3 \le k < 7} \{ m[3, k] + p[2]p[k]p[7] + m[k+1, 7] \}$$

Algoritmo recursivo

Recebe $p[i-1\,..\,j]$ e devolve $m[i,j]$

```
REC-MAT-CHAIN (p, i, j)
se i = j
      então devolva 0
m[i, j] ← ∞
para k ← i até j − 1 faça
      q1 ← REC-MAT-CHAIN (p, i, k)
      q2 ← REC-MAT-CHAIN (p, k + 1, j)
      q ← q1 + p[i − 1]p[k]p[j] + q2
      se q < m[i, j]
            então m[i, j] ← q
devolva m[i, j]
```

Consumo de tempo?

Consumo de tempo

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

O programa foi compilado com o gcc versão ?? e opção de compilação "-O2".

n	3	6	10	20	25
tempo	$0.0s$	$0.0s$	$0.01s$	$201s$	$567m$

Consumo de tempo

$T(n) =$ número comparações entre q e $m[\star, \star]$
na linha 8 quando $n := j - i + 1$

$$
\begin{aligned}
T(1) &= 0 \\
T(n) &= \sum_{h=1}^{n-1} (T(h) + T(n-h) + 1) \\
&= 2 \sum_{h=2}^{n-1} T(h) \ + \ (n-1) \\
&= 2(T(2) + \cdots + T(n-1)) \ + \ (n-1) \ \text{ para } n \geq 2
\end{aligned}
$$

Consumo de tempo

$T(n) =$ número comparações entre q e $m[\star, \star]$ na linha 8 quando $n := j - i + 1$

$$
\begin{aligned}
T(1) &= 0 \\
T(n) &= \sum_{h=1}^{n-1} (T(h) + T(n-h) + 1) \\
&= 2 \sum_{h=2}^{n-1} T(h) \; + \; (n-1) \\
&= 2(T(2) + \cdots + T(n{-}1)) \; + \; (n-1) \quad \text{para } n \geq 2
\end{aligned}
$$

Fácil verificar: $T(n) \geq 2^{n-2}$ para $n \geq 2$.

Recorrência

n	1	2	3	4	5	6	7	8
$T(n)$	0	1	4	13	40	121	364	1093
2^{n-2}	0.5	1	2	8	16	32	64	128

Prova: Para $n = 2$, $T(2) = 1 = 2^{2-2}$.
Para $n \geq 3$,

$$T(n) = 2(T(2) + \cdots + T(n-1)) + n - 1$$

$$\stackrel{\text{hi}}{\geq} 2(2^0 + \cdots + 2^{n-3}) + n - 1$$

$$> 2^0 + \cdots + 2^{n-3} + n - 1$$

$$= 2^{n-2} - 1 + n - 1$$

$$> 2^{n-2} \quad \text{(pois } n \geq 3\text{).}$$

Conclusão

O consumo de tempo do algoritmo
REC-MAT-CHAIN é $\Omega(2^n)$.

Resolve subproblemas muitas vezes

$p[0] = 10 \quad p[1] = 100 \quad p[2] = 5 \quad p[3] = 50$

```
REC-MAT-CHAIN(p, 1, 3)
  REC-MAT-CHAIN(p, 1, 1)
  REC-MAT-CHAIN(p, 2, 3)
    REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 3)
  REC-MAT-CHAIN(p, 1, 2)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 2)
  REC-MAT-CHAIN(p, 3, 3)
```

Número mínimo de mults = 7500

Resolve subproblemas muitas vezes

```
REC-MAT-CHAIN(p, 1, 5)
  REC-MAT-CHAIN(p, 1, 1)
  REC-MAT-CHAIN(p, 2, 5)
    REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 5)
      REC-MAT-CHAIN(p, 3, 3)
      REC-MAT-CHAIN(p, 4, 5)
        REC-MAT-CHAIN(p, 4, 4)
        REC-MAT-CHAIN(p, 5, 5)
      REC-MAT-CHAIN(p, 3, 4)
        REC-MAT-CHAIN(p, 3, 3)
        REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 2, 3)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 4, 5)
      REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 2, 4)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 4)
        REC-MAT-CHAIN(p, 3, 3)

      REC-MAT-CHAIN(p, 4, 4)
    REC-MAT-CHAIN(p, 5, 5)
  REC-MAT-CHAIN(p, 1, 2)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 2)
  REC-MAT-CHAIN(p, 3, 5)
    REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 4, 5)
      REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 3, 4)
      REC-MAT-CHAIN(p, 3, 3)
      REC-MAT-CHAIN(p, 4, 4)
    REC-MAT-CHAIN(p, 5, 5)
  REC-MAT-CHAIN(p, 1, 3)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 3)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 1, 2)
      REC-MAT-CHAIN(p, 1, 1)
      REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 3)

REC-MAT-CHAIN(p, 1, 1)
REC-MAT-CHAIN(p, 2, 4)
  REC-MAT-CHAIN(p, 2,
  REC-MAT-CHAIN(p, 3,
    REC-MAT-CHAIN(p, 3
    REC-MAT-CHAIN(p, 4
  REC-MAT-CHAIN(p, 2,
    REC-MAT-CHAIN(p, 2
    REC-MAT-CHAIN(p, 3
  REC-MAT-CHAIN(p, 4,
REC-MAT-CHAIN(p, 1, 2)
  REC-MAT-CHAIN(p, 1,
  REC-MAT-CHAIN(p, 2,
REC-MAT-CHAIN(p, 3, 4)
  REC-MAT-CHAIN(p, 3,
  REC-MAT-CHAIN(p, 4,
REC-MAT-CHAIN(p, 1, 3)
  REC-MAT-CHAIN(p, 1,
  REC-MAT-CHAIN(p, 2,
    REC-MAT-CHAIN(p, 2
    REC-MAT-CHAIN(p, 3
  REC-MAT-CHAIN(p, 1,
    REC-MAT-CHAIN(p, 1
```

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2, 6]$ preciso de ...

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2,6]$ preciso de ...

$m[2,2]$, $m[2,3]$, $m[2,4]$, $m[2,5]$ e de
$m[3,6]$, $m[4,6]$, $m[5,6]$, $m[6,6]$.

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2,6]$ preciso de . . .

$m[2,2]$, $m[2,3]$, $m[2,4]$, $m[2,5]$ e de
$m[3,6]$, $m[4,6]$, $m[5,6]$, $m[6,6]$.

Calcule todos os $m[i,j]$ com $j-i+1=2$,
depois todos com $j-i+1=3$,
depois todos com $j-i+1=4$,
etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	j
1	0								
2		0	$\star$	$\star$	$\star$	??			
3			0			$\star$			
4				0		$\star$			
5					0	$\star$			
6						0			
7							0		
8								0	
i									

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	??				
2		0				
3			0			
4				0		
5					0	
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0					
3			0				
4				0			
5					0		
6						0	

$$m[1,1] + p[1{-}1]p[1]p[2] + m[1{+}1,2] = 0 + 2000 + 0 = 2000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	??				
3			0				
4				0			
5					0		
6						0	
i							

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0				
4				0			
5					0		
6						0	

$$m[2,2] + p[2{-}1]p[2]p[3] + m[2{+}1,3]{=}0{+}6000{+}0{=}6000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	??		
4				0		
5					0	
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0	6000			
4				0			
5					0		
6						0	

$$m[3,3] + p[3-1]p[3]p[4] + m[3+1,4] = 0 + 6000 + 0 = 6000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	6000		
4				0	??	
5					0	
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0	6000			
4				0	4500		
5					0		
6						0	

$$m[4,4] + p[4{-}1]p[4]p[5] + m[4{+}1,5]{=}0{+}4500{+}0{=}4500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	6000		
4				0	4500	
5					0	??
6						0

Simulação

$p[0]=10 \quad p[1]=10 \quad p[2]=20 \quad p[3]=30 \quad p[4]=10 \quad p[5]=15 \quad p[6]=30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$$m[5,5] + p[5-1]p[5]p[6] + m[5+1,6] = 0 + 4500 + 0 = 4500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	??				
2		0	6000				
3			0	6000			
4				0	4500		
5					0	4500	
6						0	
i							

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	9000				
2		0	6000				
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$m[1,1] + p[1{-}1]p[1]p[3] + m[1{+}1,3] {=} 0{+}3000{+}6000{=}9000$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000				
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$$m[1,2] + p[1{-}1]p[2]p[3] + m[2{+}1,3]{=}2000{+}6000{+}0{=}8000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000	8000			
2		0	6000	??		
3			0	6000		
4				0	4500	
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$$m[2,2] + p[2-1]p[2]p[4] + m[2+1,4] = 0 + 2000 + 6000 = 8000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$$m[2,3] + p[2-1]p[3]p[4] + m[3+1,4] = 6000 + 3000 + 0 = 9000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000	8000			
2		0	6000	8000		
3			0	6000	??	
4				0	4500	
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	13500		
4				0	4500		
5					0	4500	
6						0	

$$m[3,3] + p[3-1]p[3]p[5] + m[3+1,5] = 0 + 9000 + 4500 = 13500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500		
5					0	4500	
6						0	

$$m[3,4] + p[3{-}1]p[4]p[5] + m[4{+}1,5] {=} 6000{+}3000{+}0{=}9000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000			
2		0	6000	8000		
3			0	6000	9000	
4				0	4500	??
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[4,4] + p[4{-}1]p[4]p[6] + m[4{+}1,6] = 0 + 9000 + 4500 = 13500$$

Simulação

$p[0]{=}10\quad p[1]{=}10\quad p[2]{=}20\quad p[3]{=}30\quad p[4]{=}10\quad p[5]{=}15\quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[4,5] + p[4-1]p[5]p[6] + m[5+1,6] = 4500 + 13500 + 0 = 18000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	??		
2		0	6000	8000		
3			0	6000	9000	
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	8000	9000		
2		0	6000	8000		
3			0	6000	9000	
4				0	4500	13500
5					0	4500
6						0

$$m[1,1] + p[1{-}1]p[1]p[4] + m[1{+}1,4] {=} 0{+}1000{+}8000{=}9000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,2] + p[1{-}1]p[2]p[4] + m[2{+}1,4]{=}2000{+}2000{+}6000{=}10000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,3] + p[1{-}1]p[3]p[4] + m[3{+}1,4]{=}8000{+}3000{+}0{=}11000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000	8000	9000		
2		0	6000	8000	??	
3			0	6000	9000	
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	12000		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,2] + p[2{-}1]p[2]p[5] + m[2{+}1,5]{=}0{+}3000{+}9000{=}12000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	12000		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,3] + p[2{-}1]p[3]p[5] + m[3{+}1,5]{=}6000{+}4500{+}4500{=}15000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,4] + p[2{-}1]p[4]p[5] + m[4{+}1,5]{=}8000{+}1500{+}0{=}9500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	8000	9000		
2		0	6000	8000	9500	
3			0	6000	9000	??
4				0	4500	13500
5					0	4500
6						0

i

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000	31500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[3,3] + p[3{-}1]p[3]p[6] + m[3{+}1,6] = 0 + 18000 + 13500 = 31500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[3,4] + p[3{-}1]p[4]p[6] + m[4{+}1,6] = 6000 + 6000 + 4500 = 16500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[3,5] + p[3-1]p[5]p[6] + m[5+1,6] = 9000 + 9000 + 0 = 18000$$

Simulação

$p[0]=10$ $p[1]=10$ $p[2]=20$ $p[3]=30$ $p[4]=10$ $p[5]=15$ $p[6]=30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000	8000	9000	??	
2		0	6000	8000	9500	
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	11000		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,1] + p[1{-}1]p[1]p[5] + m[1{+}1,5] {=} 0{+}1500{+}9500{=}11000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	11000		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,2] + p[1{-}1]p[2]p[5] + m[2{+}1,5] {=} 2000{+}3000{+}9000{=}14000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	11000		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,3] + p[1{-}1]p[3]p[5] + m[3{+}1,5]{=}8000{+}4500{+}4500{=}17000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,4] + p[1{-}1]p[4]p[5] + m[4{+}1,5] {=} 9000{+}1500{+}0 {=} 10500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000	8000	9000	10500	
2		0	6000	8000	9500	??
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	22500	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,2] + p[2{-}1]p[2]p[6] + m[2{+}1,6]{=}0{+}6000{+}16500{=}22500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	22500	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,3] + p[2-1]p[3]p[6] + m[3{+}1,6] = 6000 + 9000 + 13500 = 28500$$

Simulação

$p[0]{=}10\quad p[1]{=}10\quad p[2]{=}20\quad p[3]{=}30\quad p[4]{=}10\quad p[5]{=}15\quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	15500	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,4]+p[2-1]p[4]p[6]+m[4+1,6]{=}8000{+}3000{+}4500{=}15500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,5] + p[2{-}1]p[5]p[6] + m[5{+}1,6] = 9500{+}4500{+}0 = 14000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000	10500	??
2		0	6000	8000	9500	14000
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	17000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,1] + p[1{-}1]p[1]p[6] + m[1{+}1,6] {=} 0 {+} 3000 {+} 14000 {=} 17000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	17000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,2] + p[1{-}1]p[2]p[6] + m[2{+}1,6] {=} 2000{+}6000{+}16500{=}24500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	17000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,3] + p[1{-}1]p[3]p[6] + m[3{+}1,6]{=}8000{+}9000{+}13500{=}30500$$

Simulação

$p[0]=10 \quad p[1]=10 \quad p[2]=20 \quad p[3]=30 \quad p[4]=10 \quad p[5]=15 \quad p[6]=30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	16500	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,4] + p[1-1]p[4]p[6] + m[4+1,6] = 9000 + 3000 + 4500 = 16500$$

Simulação

$p[0]=10 \quad p[1]=10 \quad p[2]=20 \quad p[3]=30 \quad p[4]=10 \quad p[5]=15 \quad p[6]=30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	15000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,5] + p[1-1]p[5]p[6] + m[5+1,6] = 10500 + 4500 + 0 = 15000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	15000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	
i							

Algoritmo de programação dinâmica

Recebe $p[0 \,..\, n]$ e devolve $m[1, n]$.

MATRIX-CHAIN-ORDER (p, n)

para $i \leftarrow 1$ **até** n **faça**
    $m[i, i] \leftarrow 0$
para $l \leftarrow 2$ **até** n **faça**
    para $i \leftarrow 1$ **até** $n - l + 1$ **faça**
        $j \leftarrow i + l - 1$
        $m[i, j] \leftarrow \infty$
        para $k \leftarrow i$ **até** $j - 1$ **faça**
            $q \leftarrow m[i, k] + p[i-1]p[k]p[j] + m[k{+}1, j]$
            se $q < m[i, j]$
                então $m[i, j] \leftarrow q$
devolva $m[1, n]$

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: ???

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Curioso verificar que consumo de tempo é $\Omega(n^3)$:
Número de execuções da linha 8:

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Curioso verificar que consumo de tempo é $\Omega(n^3)$:
Número de execuções da linha 8:

l	i	execs linha 8
2	$1,\ldots,n-1$	$(n-1)\cdot 1$
3	$1,\ldots,n-2$	$(n-2)\cdot 2$
4	$1,\ldots,n-3$	$(n-3)\cdot 3$
$\ldots$	$\ldots$	$\ldots$
$n-1$	$1,2$	$2\cdot(n-2)$
n	1	$1\cdot(n-1)$
	total	$\sum_{h=1}^{n-1} h(n-h)$

Consumo de tempo

Para $n \geq 6$, $\sum_{h=1}^{n-1} h(n-h) \;=$

$$
\begin{aligned}
&= \; n\sum_{h=1}^{n-1} h \;-\; \sum_{h=1}^{n-1} h^2 \\
&= \; n\tfrac{1}{2}n(n-1) - \tfrac{1}{6}(n-1)n(2n-1) \qquad \text{(CLRS p.1060)} \\
&\geq \; \tfrac{1}{2}n^2(n-1) - \tfrac{1}{6}2n^3 \\
&\geq \; \tfrac{1}{2}n^2\tfrac{5n}{6} - \tfrac{1}{3}n^3 \\
&= \; \tfrac{5}{12}n^3 - \tfrac{1}{3}n^3 \\
&= \; \tfrac{1}{12}n^3
\end{aligned}
$$

Consumo de tempo é $\Omega(n^3)$

Conclusão

O consumo de tempo do algoritmo
MATRIX-CHAIN-ORDER é $\Theta(n^3)$.

Versão recursiva eficiente

MEMOIZED-MATRIX-CHAIN-ORDER (p, n)

```
para i ← 1 até n faça
    para j ← 1 até n faça
        m[i, j] ← ∞
devolva LOOKUP-CHAIN (p, 1, n)
```

Versão recursiva eficiente

LOOKUP-CHAIN (p, i, j)

se $m[i, j] < \infty$
    então devolva $m[i, j]$
se $i = j$
    então $m[i, j] \leftarrow 0$
    senão para $k \leftarrow i$ **até** $j - 1$ **faça**
        $q \leftarrow$ LOOKUP-CHAIN (p, i, k)
            $+\ p[i-1]p[k]p[j]$
            $+$ LOOKUP-CHAIN $(p, k+1, j)$
        se $q < m[i, j]$
            então $m[i, j] \leftarrow q$
devolva $m[1, n]$

Ingredientes de programação dinâmica

- Subestrutura ótima: soluções ótimas contém soluções ótimas de subproblemas.
- Subestrutura: decomponha o problema em subproblemas menores e, com sorte, mais simples.
- *Bottom-up*: combine as soluções dos problemas menores para obter soluções dos maiores.
- Tabela: armazene as soluções dos subproblemas em uma tabela, pois soluções dos subproblemas são consultadas várias vezes.
- Número de subproblemas: para a eficiência do algoritmo é importante que o número de subproblemas resolvidos seja ‘pequeno’.
- Memoized: versão *top-down*, recursão com tabela.

Exercício

O algoritmo MATRIX-CHAIN-ORDER determina o número mínimo de multiplicações escalares necessário para calcular produto $A_1 A_2 \cdots A_n$.

Na aula, mencionamos uma maneira de obter uma parentização ótima a partir dos cálculos feitos, usando para isso um dado a mais que podemos guardar no decorrer do algoritmo.

Faça os ajustes sugeridos na aula, de modo a guardar esse dado extra, e devolvê-lo junto com o valor $m[1, n]$.

Faça uma rotina que recebe a informação extra armazenada pelo algoritmo acima e imprime uma parentização ótima das matrizes $A_1 A_2 \cdots A_n$.

Exercícios

Exercício 19.A [CLRS 15.2-1]
Encontre a maneira ótima de fazer a multiplicação iterada das matrizes cujas dimensões são $(5, 10, 3, 12, 5, 50, 6)$.

Exercício 19.B [CLRS 15.2-5]
Mostre que são necessários exatamente $n-1$ pares de parênteses para especificar exatamente a ordem de multiplicação de $A_1 \cdot A_2 \cdots A_n$.

Exercício 19.C [CLRS 15.3-2]
Desenhe a árvore de recursão para o algoritmo MERGE-SORT aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

Exercício 19.D [CLRS 15.3-5 expandido]
Considere o seguinte algoritmo para determinar a ordem de multiplicação de uma cadeia de matrizes $A_1, A_2, \ldots, A_n$ de dimensões $p_0, p_1, \ldots, p_n$: primeiro, escolha k que minimize p_k; depois, determine recursivamente as ordens de multiplicação de $A_1, \ldots, A_k$ e $A_{k+1}, \ldots, A_n$. Esse algoritmo produz uma ordem que minimiza o número total de multiplicações escalares? E se k for escolhido de modo a maximizar p_k? E se k for escolhido de modo a minimizar p_k?

Mais exercícios

Exercício 19.E
Prove que o número de execuções da linha 9 em MATRIX-CHAIN-ORDER é $\mathrm{O}(n^3)$.

Exercício 19.F [Subset-sum. CLRS 16.2-2 simplificado]
Escreva um algoritmo de programação dinâmica para o seguinte problema: dados números inteiros não-negativos $w_1, \ldots, w_n$ e W, encontrar um subconjunto K de $\{1, \ldots, n\}$ que satisfaça $\sum_{k \in K} w_k \leq W$ e maximize $\sum_{k \in K} w_k$. (Imagine que $w_1, \ldots, w_n$ são os tamanhos de arquivos digitais que você deseja armazenar em um disquete de capacidade W.)

Exercício 19.G [Mochila 0-1. CLRS 16.2-2]
O problema da mochila 0-1 consiste no seguinte: dados números inteiros não-negativos $v_1, \ldots, v_n$, $w_1, \ldots, w_n$ e W, queremos encontrar um subconjunto K de $\{1, \ldots, n\}$ que

$$\text{satisfaça } \sum_{k \in K} w_k \leq W \text{ e maximize } \sum_{k \in K} v_k.$$

(Imagine que w_i é o *peso* e v_i é o *valor* do objeto i.) Resolva o problema usando programação dinâmica.

Mais um exercício

Exercício 19.H [Partição equilibrada]
Seja S o conjunto das raízes raízes quadradas dos números $1, 2, \ldots, 500$. Escreva e teste um programa que determine uma partição (A, B) de S tal que a soma dos números em A seja tão próxima quanto possível da soma dos números em B. Seu algoritmo resolve o problema? ou só dá uma solução “aproximada”?
Uma vez calculados A e B, seu programa deve imprimir a diferença entre a soma de A e a soma de B e depois imprimir a lista dos quadrados dos números em um dos conjuntos.

Mais programação dinâmica

CLRS 15.4 e 15.5

= "recursão–com–tabela"
= transformação inteligente de recursão em iteração

Subseqüências

$\langle z_1, \dots, z_k \rangle$ é subseqüência de $\langle x_1, \dots, x_m \rangle$
se existem índices $i_1 < \cdots < i_k$ tais que

$$z_1 = x_{i_1} \qquad \dots \qquad z_k = x_{i_k}$$

EXEMPLOS:

$\langle 5, 9, 2, 7 \rangle$ é subseqüência de $\langle 9, 5, 6, 9, 6, 2, 7, 3 \rangle$

⟨A,A, D, A, A⟩ é subseqüência de
⟨A, B, R, A, C, A, D, A, B, R, A⟩

```
A           A           D   A           A
|           |           |   |           |
A   B   R   A   C   A   D   A   B   R   A
```

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)

$i \leftarrow m$
$j \leftarrow n$
enquanto $i \geq 1$ **e** $j \geq 1$ **faça**
    se $Z[i] = X[j]$
        então $i \leftarrow i - 1$
    $j \leftarrow j - 1$
se $i \geq 1$
    então devolva "`não é subseqüência`"
    senão devolva "`é subseqüência`"

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)

$i \leftarrow m$
$j \leftarrow n$
enquanto $i \geq 1$ **e** $j \geq 1$ **faça**
    se $Z[i] = X[j]$
        então $i \leftarrow i - 1$
    $j \leftarrow j - 1$
se $i \geq 1$
    então devolva "`não é subseqüência`"
    senão devolva "`é subseqüência`"

Consumo de tempo é $\mathrm{O}(m + n)$ e $\Omega(\min\{m, n\})$.

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)

$i \leftarrow m$
$j \leftarrow n$
enquanto $i \geq 1$ **e** $j \geq 1$ **faça**
  se $Z[i] = X[j]$
    então $i \leftarrow i - 1$
  $j \leftarrow j - 1$
se $i \geq 1$
  então devolva "`não é subseqüência`"
  senão devolva "`é subseqüência`"

Invariantes:

(i0) $Z[i{+}1\,..\,m]$ é subseqüência de $X[j{+}1\,..\,n]$

(i1) $Z[i\,..\,m]$ **não** é subseqüência de $X[j{+}1\,..\,n]$

Subseqüência comum máxima

Z é subseq comum de X e Y

se Z é subseqüência de X e de Y

ssco $=$ subseq comum

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

Outra ssco = B D A B

Problema

Problema: Encontrar uma ssco máxima de X e Y.

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

ssco maximal = A B A

ssco máxima = B C A B

Outra ssco máxima = B D A B

LCS = Longest Common Subsequence

diff

```
> more abracadabra
A
B
R
A
C
A
D
A
B
R
A
```

```
> more yabbadabbadoo
Y
A
B
B
A
D
A
B
B
A
D
D
O
```

diff -u abracadabra yabbadabbadoo

```
+Y
 A
 B
-R
-A
-C
+B
 A
 D
 A
 B
-R
+B
 A
+D
+D
+O
```

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$, então $Z[k] = X[m] = Y[n]$ e $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq X[m]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq Y[n]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n{-}1]$.

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

Simplificação

Problema: encontrar o comprimento de uma ssco máxima.

$c[i, j] =$ comprimento de uma ssco máxima
de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Recorrência:

$c[0, j] = c[i, 0] = 0$

$c[i, j] = c[i{-}1, j{-}1] + 1$ se $X[i] = Y[j]$

$c[i, j] = \max\,(c[i, j{-}1], c[i{-}1, j])$ se $X[i] \neq Y[j]$

Algoritmo recursivo

Devolve o comprimento de uma ssco máxima de $X[1\,.\,.\,i]$ e $Y[1\,.\,.\,j]$.

REC-LCS-LENGTH (X, i, Y, i)

se $i = 0$ **ou** $j = 0$
    então devolva 0
se $X[i] = Y[j]$
    então
$c[i, j] \leftarrow$REC-LCS-LENGTH $(X, i-1, Y, j-1) + 1$
    senão $q_1 \leftarrow$ REC-LCS-LENGTH $(X, i-1, Y, j)$
        $q_2 \leftarrow$ REC-LCS-LENGTH $(X, i, Y, j-1)$
        se $q_1 \geq q2$
            então $c[i, j] \leftarrow q_1$
            senão$c[i, j] \leftarrow q_2$
devolva $c[i, j]$

Consumo de tempo

$T(m, n) :=$ número de comparações feitas por
REC-LCS-LENGTH (X, m, Y, n)

Recorrência

$$T(0, n) = 0$$
$$T(m, 0) = 0$$
$$T(m, n) \leq T(m-1, n) + T(m, n-1) + 1 \text{ para } n \geq 0 \text{ e } m \geq 0$$

A que classe Ω pertence $T(m, n)$?

Recorrência

Note que $T(m,n) = T(n,m)$ para $n = 0,1,\ldots$ e $m = 0,1,\ldots$.

Seja $k := \min\{m,n\}$. Temos que

$$T(m,n) \geq T(k,k) \geq S(k),$$

onde

$$S(0) = 0$$
$$S(k) = 2S(k-1) + 1 \quad \text{para } k = 1,2,\ldots$$

$S(k)$ é $\Theta(2^k) \Rightarrow T(m,n)$ é $\Omega(2^{\min\{m,n\}})$

$T(m,n)$ é exponecial

Conclusão

O consumo de tempo do algoritmo
REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1 \,..\, i] \quad \text{e} \quad Y[1 \,..\, j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

$c[4,5]$, $c[3,6]$ e de $c[3,5]$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,.\,.\,i] \quad \text{e} \quad Y[1\,.\,.\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[4,6]$ preciso de ...

$c[4,5]$, $c[3,6]$ e de $c[3,5]$.

Calcule todos os $c[i,j]$ com $i = 1, j = 0, 1, \ldots, n$,
depois todos com $i = 2, j = 0, 1, \ldots, n$,
depois todos com $i = 3, j = 0, 1, \ldots, n$,
etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	j
1	0	0	0	0	0	0	0	0	
2	0								
3	0				$\star$	$\star$			
4	0				$\star$	??			
5	0								
6	0								
7	0								
8	0								

i

Simulação

Y			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	??						
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	??					
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	??				
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	??			
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	??		
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	??	
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	??						
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	??					
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	??				
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	??			
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	??		
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	??	
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	??						
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	??					
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	??				
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	??			
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	??		
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	??	
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	??						
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	??					
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	??				
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	??			
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	??		
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	??	
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	??						
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	??					
A	6	0							
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	??				
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	??			
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	??		
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	??	
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	??						
B	7	0							
	i								

Simulação

Y			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	??					
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	??				
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	??			
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	??		
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	??	
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	??						
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	??					
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	??				
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	??			
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	??		
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	??	

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	4	
	i								

Algoritmo de programação dinâmica

Devolve o comprimento de uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

LEC-LENGTH (X, m, Y, n)

para $i \leftarrow 0$ **até** m **faça**
    $c[i, 0] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
    $c[0, j] \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
    para $j \leftarrow 1$ **até** n **faça**
        se $X[i] = Y[j]$
            então $c[i, j] \leftarrow c[i-1, j-1] + 1$
            senão se $c[i-1, j] \geq c[i, j-1]$
                então $c[i, j] \leftarrow c[i-1, j]$
                senão $c[i, j] \leftarrow c[i, j-1]$
devolva $c[m, n]$

Conclusão

O consumo de tempo do algoritmo LEC-LENGTH é $\Theta(mn)$.

Subseqüência comum máxima

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	
A	1	$\star$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	
B	2	$\star$	$\nwarrow$	$\uparrow$	$\uparrow$	$\leftarrow$	$\nwarrow$	$\uparrow$	
C	3	$\star$	$\leftarrow$	$\leftarrow$	$\nwarrow$	$\uparrow$	$\leftarrow$	$\leftarrow$	
B	4	$\star$	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\nwarrow$	$\uparrow$	
D	5	$\star$	$\leftarrow$	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	
A	6	$\star$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\nwarrow$	$\leftarrow$	$\nwarrow$	
B	7	$\star$	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\nwarrow$	$\leftarrow$	

i

Algoritmo de programação dinâmica

LEC-LENGTH (X, m, Y, n)

para $i \leftarrow 0$ **até** m **faça**
  $c[i, 0] \leftarrow 0$
para $j \leftarrow 1$ **até** n **faça**
  $c[0, j] \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
  para $j \leftarrow 1$ **até** n **faça**
    se $X[i] = Y[j]$
      então $c[i, j] \leftarrow c[i-1, j-1] + 1$
        $b[i, j] \leftarrow$ "$\nwarrow$"
      senão se $c[i-1, j] \geq c[i, j-1]$
        então $c[i, j] \leftarrow c[i-1, j]$
          $b[i, j] \leftarrow$ "$\uparrow$"
        senão $c[i, j] \leftarrow c[i, j-1]$
          $b[i, j] \leftarrow$ "$\leftarrow$"
devolva c e b

Get-LCS

GET-LCS $(X, m, n, b,$ máxcomp$)$

$k \leftarrow$ máxcomp
$i \leftarrow m$
$j \leftarrow n$
enquanto $i > 0$ **e** $j > 0$ **faça**
    se $b[i, j] =$ "$\nwarrow$"
        então $Z[k] \leftarrow X[i]$
            $k \leftarrow k - 1 \quad i \leftarrow i - 1 \quad j \leftarrow j - 1$
        senão se $b[i, j] =$ "$\leftarrow$"
            então $i \leftarrow i - 1$
            senão $j \leftarrow j - 1$
devolva Z

Consumo de tempo é $O(m + n)$ e $\Omega(\min\{m, n\})$.

Exercícios

Exercício 20.A

Escreva um algoritmo para decidir se $\langle z_1, \ldots, z_k \rangle$ é subseqüência de $\langle x_1, \ldots, x_m \rangle$. Prove rigorosamente que o seu algoritmo está correto.

Exercício 20.B

Suponha que os elementos de uma seqüência $\langle a_1, \ldots, a_n \rangle$ são distintos dois a dois. Quantas subseqüências tem a seqüência?

Exercício 20.C

Uma subseqüência crescente Z de uma seqüência X e é *máxima* se não existe outra subseqüência crescente mais longa. A subseqüência $\langle 5, 6, 9 \rangle$ de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ é máxima? Dê uma seqüência crescente máxima de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$. Mostre que o algoritmo "guloso" óbvio não é capaz, em geral, de encontrar uma subseqüência crescente máxima de uma seqüência dada. (Algoritmo guloso óbvio: escolha o menor elemento de X; a partir daí, escolha sempre o próximo elemento de X que seja maior ou igual ao último escolhido.)

Exercício 20.D

Escreva um algoritmo de programação dinâmica para resolver o problema da subseqüência crescente máxima.

Mais exercícios

Exercício 20.E [CLRS 15.4-5]
Mostre como o algoritmo da subseqüência comum máxima pode ser usado para resolver o problema da subseqüência crescente máxima de uma seqüência numérica. Dê uma delimitação justa, em notação Θ, do consumo de tempo de sua solução.

Exercício 20.F [Printing neatly. CLRS 15-2]
Considere a seqüência $P_1, P_2, \ldots, P_n$ de palavras que constitui um parágrafo de texto. A palavra P_i tem l_i caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo M caracteres. Se uma determinada linha contém as palavras $P_i, P_{i+1}, \ldots, P_j$ (com $i \leq j$) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

$$M - (l_i + 1 + l_{i+1} + 1 + \cdots + 1 + l_j)\,.$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas $1, 2, \ldots, L$ e b_p espaços no fim da linha p, queremos minimizar $b_1^3 + b_2^3 + \cdots + b_{L-1}^3$).
Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a “optimal substructure property” para esse problema? Faça uma análise do consumo de tempo do algoritmo.

Buscas em um conjunto conhecido

Considere um inteiro n e um vetor $v[1\,..\,n]$ de inteiros.

Problema: Dado $v[1\,..\,n]$ e uma sequência de k inteiros, decidir se cada inteiro está ou não em v.

Buscas em um conjunto conhecido

Considere um inteiro n e um vetor $v[1\,..\,n]$ de inteiros.

Problema: Dado $v[1\,..\,n]$ e uma sequência de k inteiros, decidir se cada inteiro está ou não em v.

Se k é grande, como devemos armazenar o v?

Buscas em um conjunto conhecido

Considere um inteiro n e um vetor $v[1\,..\,n]$ de inteiros.

Problema: Dado $v[1\,..\,n]$ e uma sequência de k inteiros, decidir se cada inteiro está ou não em v.

Se k é grande, como devemos armazenar o v?

E se v armazena um conjunto bem conhecido, como por exemplo as palavras de uma língua? (A ser usado por um tradutor, ou um speller.)

Buscas em um conjunto conhecido

Considere um inteiro n e um vetor $v[1\,..\,n]$ de inteiros.

Problema: Dado $v[1\,..\,n]$ e uma sequência de k inteiros, decidir se cada inteiro está ou não em v.

Se k é grande, como devemos armazenar o v?

E se v armazena um conjunto bem conhecido,
como por exemplo as palavras de uma língua?
(A ser usado por um tradutor, ou um speller.)

Podemos fazer algo melhor?

Buscas em conjunto conhecido

Dadas estimativas do número de acessos a cada elemento de $v[1\,.\,.\,n]$, qual é a melhor estrutura de dados para v?

Buscas em conjunto conhecido

Dadas estimativas do número de acessos a cada elemento de $v[1\,..\,n]$, qual é a melhor estrutura de dados para v?

Árvore de busca binária (ABB)?

Buscas em conjunto conhecido

Dadas estimativas do número de acessos a cada elemento de $v[1\,..\,n]$, qual é a melhor estrutura de dados para v?

Árvore de busca binária (ABB)?

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

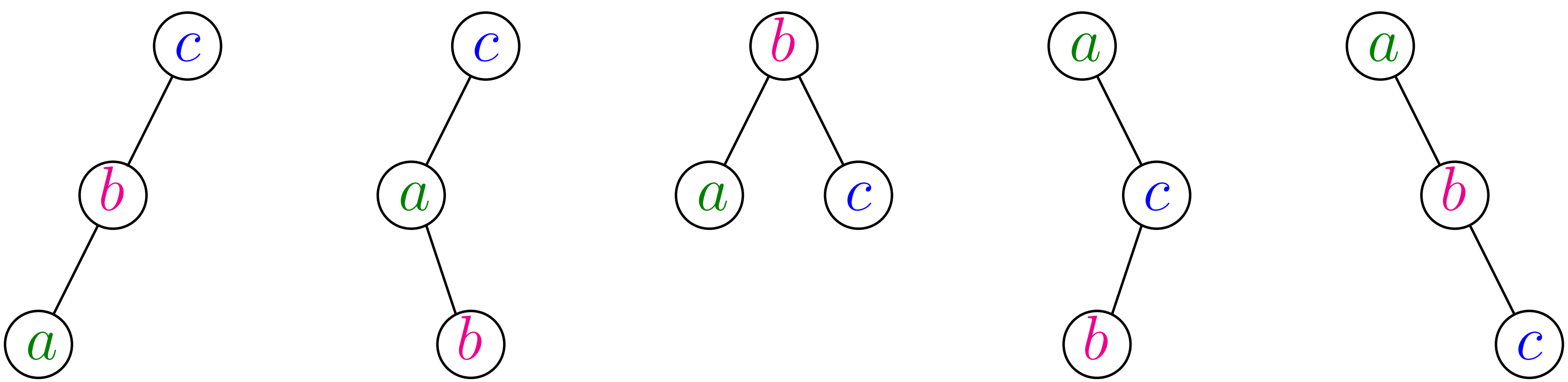
Buscas em conjunto conhecido

Dadas estimativas do número de acessos a cada elemento de $v[1\,..\,n]$, qual é a melhor estrutura de dados para v?

Árvore de busca binária (ABB)?

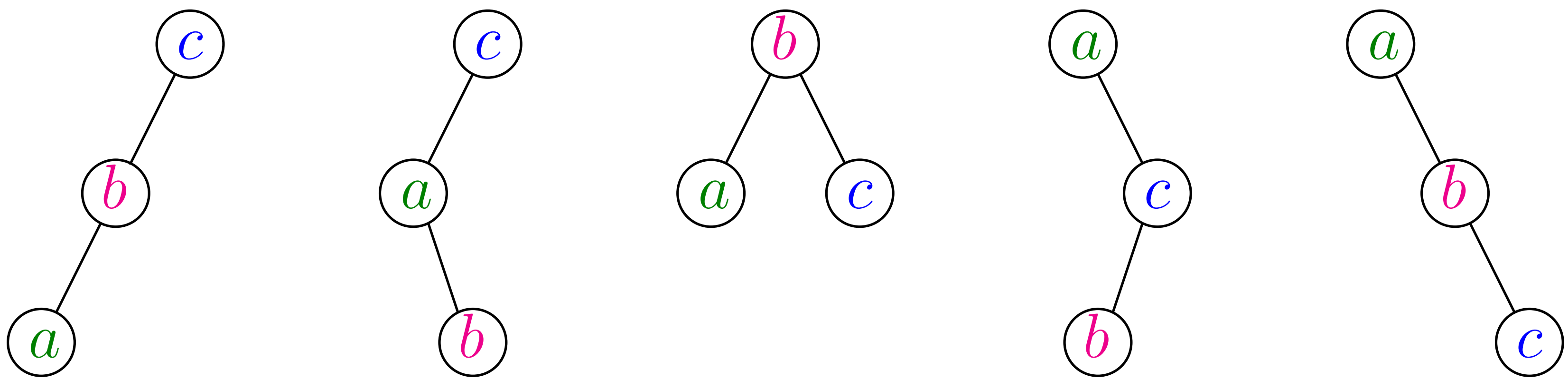
Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

Qual a melhor das ABBs?



Exemplo

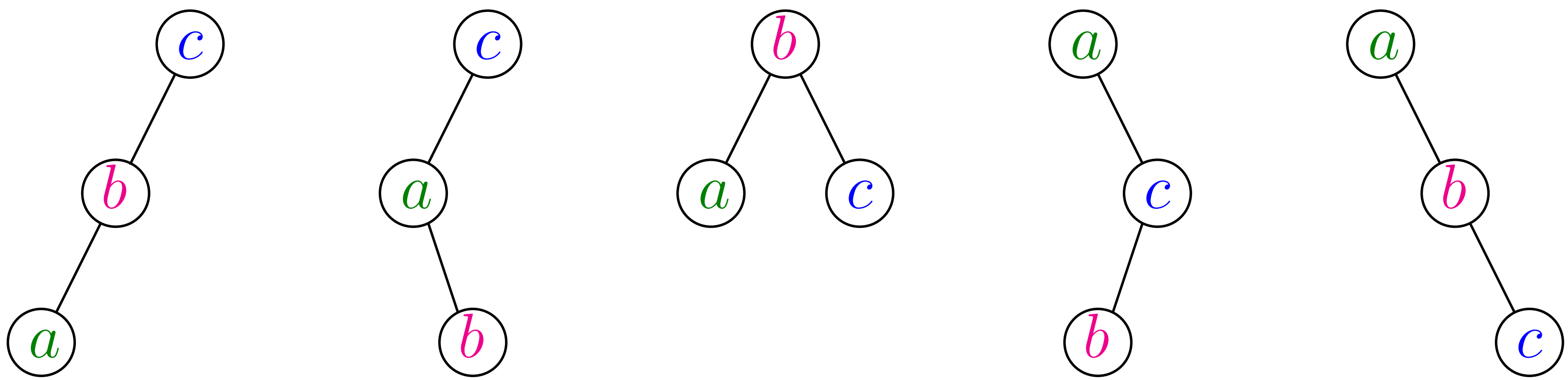
Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Qual a melhor das ABBs?

Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

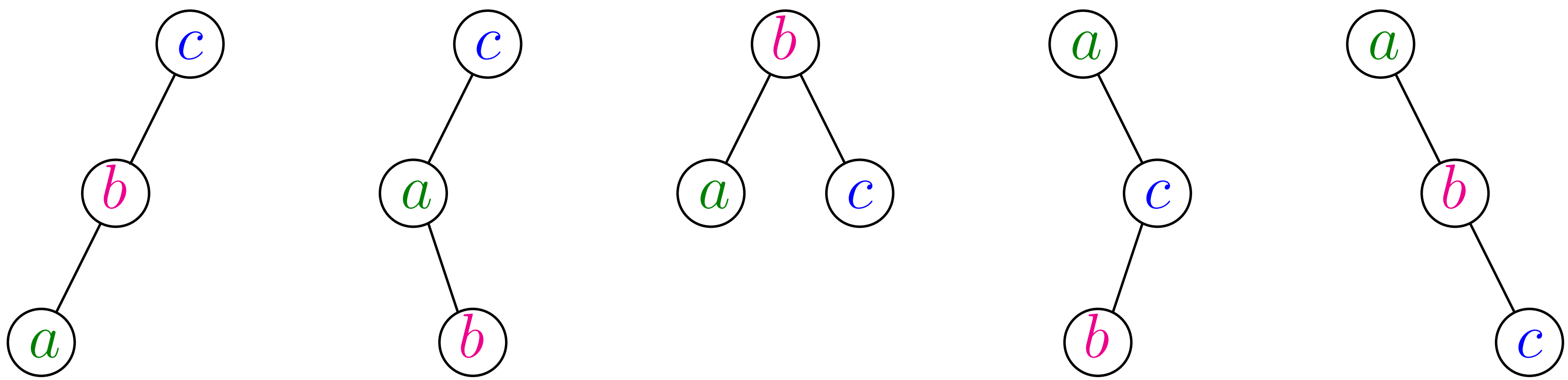


Número esperado de comparações:

- $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$
- $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 2 = 170$

Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Número esperado de comparações:

- $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$ $\longleftarrow$ ABB ótima
- $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 2 = 170$

Árvore de busca ótima

Considere um vetor $e[1\,..\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1, \ldots, n\}$.

Árvore de busca ótima

Considere um vetor $e[1\,..\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1,\ldots,n\}$.

Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1,\ldots,n\}$ que minimiza o número

$$\sum_{i=1}^{n} h_i\, e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

Árvore de busca ótima

Considere um vetor $e[1\,.\,.\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1,\ldots,n\}$.

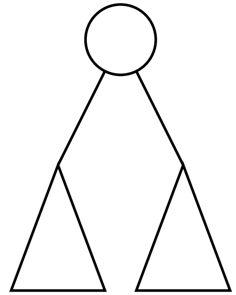
Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1,\ldots,n\}$ que minimiza o número

$$\sum_{i=1}^{n} h_i\, e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

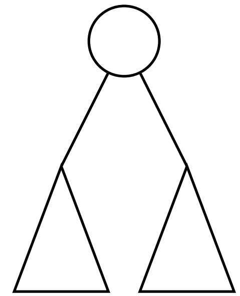
Problema (ABB Ótima): Dado $e[1\,.\,.\,n]$, encontrar uma árvore de busca binária ótima com respeito a e.

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

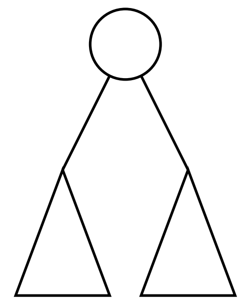
Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima
são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

Subestrutura ótima

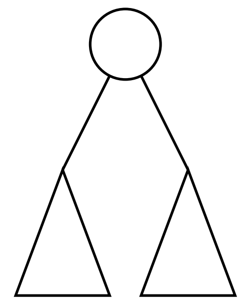


Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

$c[i, j]$: custo min de uma ABB para $e[i \mathinner{..} j]$
$s[i, j]$: soma dos acessos em $e[i \mathinner{..} j]$

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

$c[i,j]$: custo min de uma ABB para $e[i \,..\, j]$
$s[i,j]$: soma dos acessos em $e[i \,..\, j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j} \{c[i,k-1] + c[k+1,j] + s[i,j]\} & \text{se } i \le j \end{cases}$$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i \mathinner{.\,.} j]$
$s[i,j]$: soma dos acessos em $e[i \mathinner{.\,.} j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k-1] + c[k+1,j] + a[i,j]\} & \text{se } i \le j \end{cases}$$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$
$s[i,j]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j} \{c[i,k-1] + c[k+1,j] + a[i,j]\} & \text{se } i \le j \end{cases}$$

$$s[i,j] = \begin{cases} 0 & \text{se } i > j \\ s[i,j-1] + e_j & \text{se } i \le j \end{cases}$$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$
$s[i,j]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k-1] + c[k+1,j] + a[i,j]\} & \text{se } i \le j \end{cases}$$

$$s[i,j] = \begin{cases} 0 & \text{se } i > j \\ s[i,j-1] + e_j & \text{se } i \le j \end{cases}$$

Como preencher as matrizes c e s?
Em que ordem?

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$
$s[i,j]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k-1] + c[k+1,j] + a[i,j]\} & \text{se } i \le j \end{cases}$$

$$s[i,j] = \begin{cases} 0 & \text{se } i > j \\ s[i,j-1] + e_j & \text{se } i \le j \end{cases}$$

Como preencher as matrizes c e s?
Em que ordem?
Como no problema da parentização! Pelas diagonais!

Árvore de busca ótima

ABB-ÓTIMA (e, n)

$s[0] = 0$
para $i \leftarrow 1$ **até** n **faça**
    $s[i] \leftarrow s[i-1] + e[i]$
para $i \leftarrow 1$ **até** $n+1$ **faça**
    $c[i][i-1] \leftarrow 0$
para $\ell \leftarrow 1$ **até** n **faça**
    para $i \leftarrow 1$ **até** $n-\ell+1$ **faça**
        $j \leftarrow i+\ell-1$
        $c[i][j] \leftarrow c[i+1][j]$
        para $k \leftarrow i+1$ **até** j **faça**
            se $c[i][k-1] + c[k+1][j] < c[i][j]$
            então $c[i][j] \leftarrow c[i][k-1] + c[k+1][j]$
        $c[i][j] \leftarrow c[i][j] + s[j] - s[i-1]$
devolva $c[1, n]$

Árvore de busca ótima

Exercício: Como fazer para obter uma ABB ótima e não apenas o seu custo? Complete o serviço!

Lista 5 na página em breve!

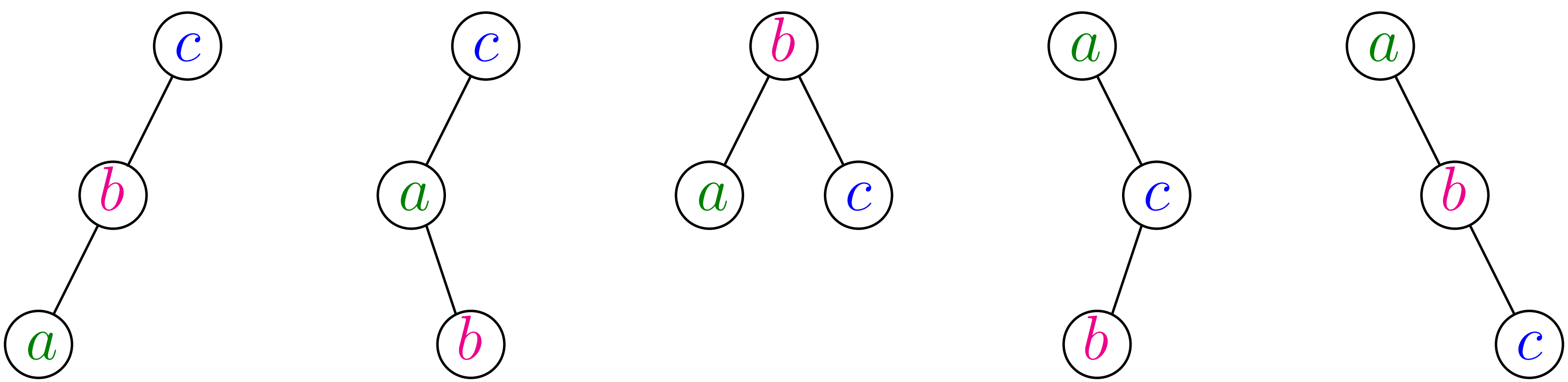
Buscas em conjunto conhecido

Dadas estimativas do número de acessos a cada elemento de $v[1\,..\,n]$, qual é a melhor estrutura de dados para v?

Árvore de busca binária (ABB)

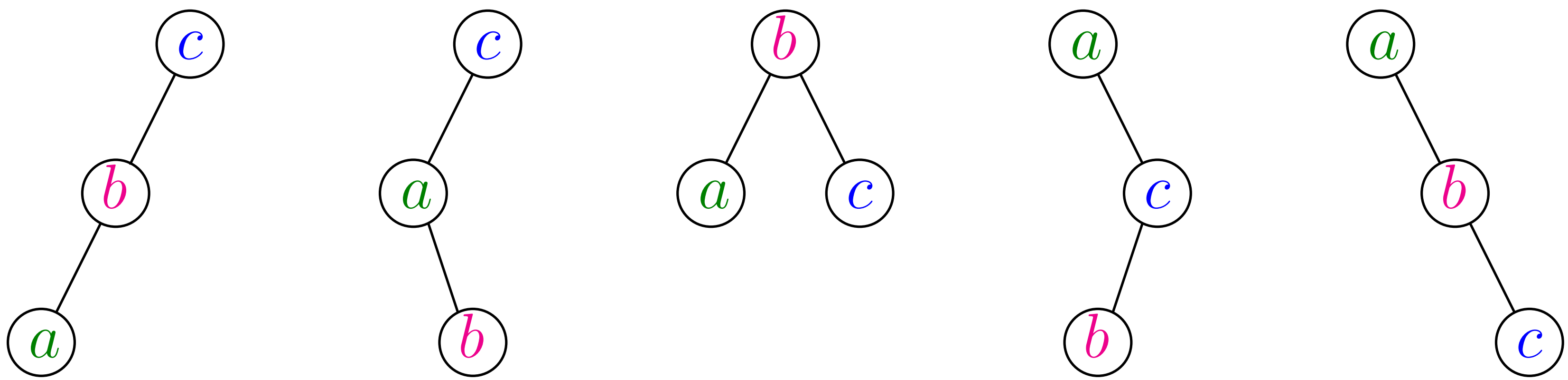
Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

Qual a melhor das ABBs?



Exemplo

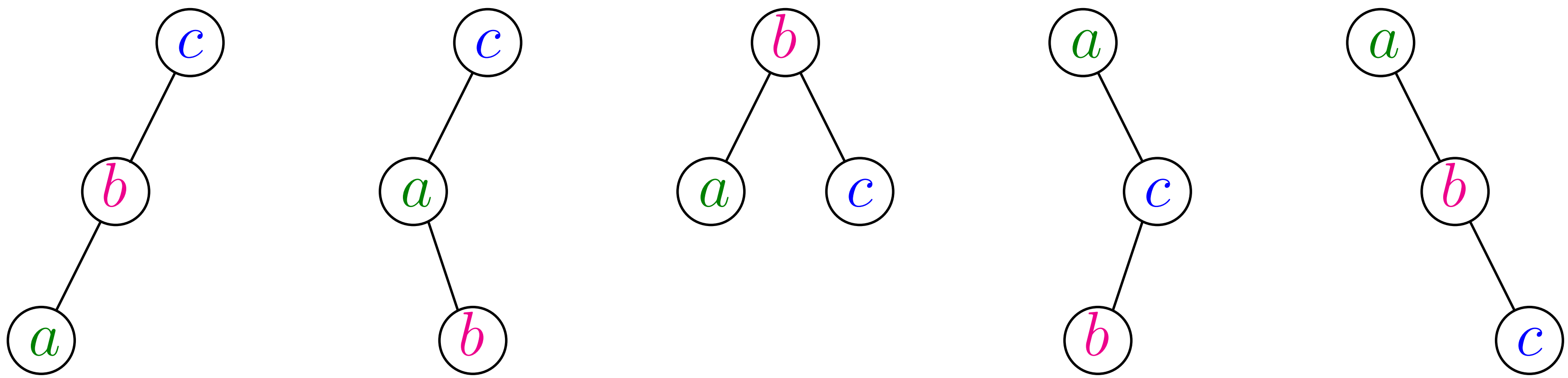
Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Qual a melhor das ABBs?

Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

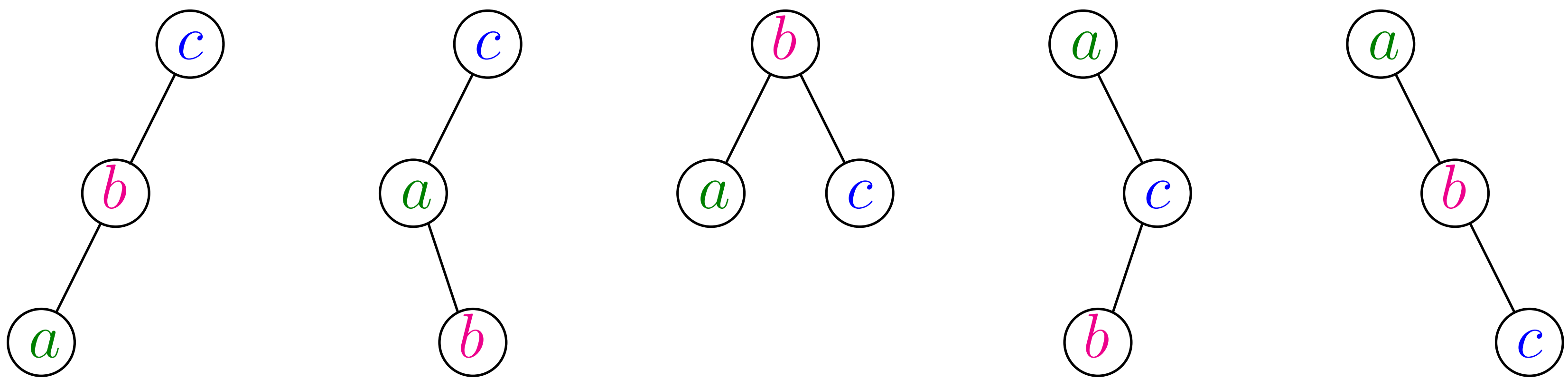


Número esperado de comparações:

- $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$
- $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 2 = 170$

Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Número esperado de comparações:

- $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$ $\longleftarrow$ ABB ótima
- $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 2 = 170$

Árvore de busca ótima

Considere um vetor $e[1\,..\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1, \ldots, n\}$.

Árvore de busca ótima

Considere um vetor $e[1\,..\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1, \ldots, n\}$.

Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1, \ldots, n\}$ que minimiza o número

$$\sum_{i=1}^{n} h_i \, e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

Árvore de busca ótima

Considere um vetor $e[1\,..\,n]$ de inteiros com uma estimativa do número de acessos a cada elemento de $\{1, \ldots, n\}$.

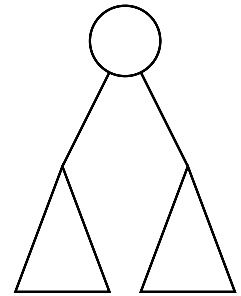
Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1, \ldots, n\}$ que minimiza o número

$$\sum_{i=1}^{n} h_i \, e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

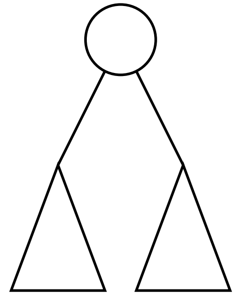
Problema (ABB Ótima): Dado $e[1\,..\,n]$, encontrar uma árvore de busca binária ótima com respeito a e.

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima
são ABBs ótimas.

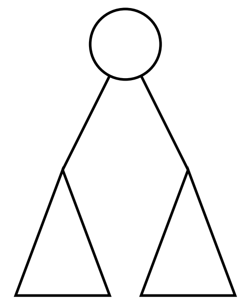
Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima
são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

Subestrutura ótima

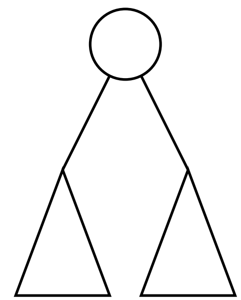


Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$
$s[i,j]$: soma dos acessos em $e[i\,..\,j]$

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

$c[i, j]$: custo min de uma ABB para $e[i \mathrel{.\,.} j]$
$s[i, j]$: soma dos acessos em $e[i \mathrel{.\,.} j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j] + s[i, j]\} & \text{se } i \leq j \end{cases}$$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$

$s[j]$: soma dos acessos em $e[1\,..\,j]$
$s[j]-s[i-1]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i\le k\le j}\{c[i,k-1]+c[k+1,j]+s[j]-s[i-1]\} & \text{se } i \le j \end{cases}$$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i \mathinner{..} j]$

$s[j]$: soma dos acessos em $e[1 \mathinner{..} j]$
$s[j] - s[i{-}1]$: soma dos acessos em $e[i \mathinner{..} j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k{-}1] + c[k{+}1,j] + s[j] - s[i{-}1]\} & \text{se } i \le j \end{cases}$$

Para calcular s:

1 $s[0] = 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad s[i] \leftarrow s[i{-}1] + e[i]$

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$

$s[j]$: soma dos acessos em $e[1\,..\,j]$
$s[j]-s[i-1]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k-1] + c[k+1,j] + s[j] - s[i-1]\} & \text{se } i \le j \end{cases}$$

Como preencher a matriz c?
Em que ordem?

Custo de uma ABB ótima

$c[i,j]$: custo min de uma ABB para $e[i\,..\,j]$

$s[j]$: soma dos acessos em $e[1\,..\,j]$
$s[j]-s[i{-}1]$: soma dos acessos em $e[i\,..\,j]$

$$c[i,j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \le k \le j}\{c[i,k{-}1] + c[k{+}1,j] + s[j] - s[i{-}1]\} & \text{se } i \le j \end{cases}$$

Como preencher a matriz c?
Em que ordem?

Como no problema da parentização! Pelas diagonais!

Programação dinâmica

	0	1	2	3	4	5	6	7	j
1	0								
2		0	$\star$	$\star$	$\star$	??			
3			0			$\star$			
4				0		$\star$			
5					0	$\star$			
6						0			
7							0		
i								0	

Simulação

$e[1]=10 \quad e[2]=20 \quad e[3]=30 \quad e[4]=15 \quad e[5]=30$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

	0	1	2	3	4	5	j
1	0	??					
2		0					
3			0				
4				0			
5					0		
6						0	
i							

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10				
2		0				
3			0			
4				0		
5					0	
6						0

$$c[1, 1{-}1] + e[1] + c[1{+}1, 1] = 0{+}10{+}0 = 10$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10				
2		0	??			
3			0			
4				0		
5					0	
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10				
2		0	20			
3			0			
4				0		
5					0	
6						0

$$c[2, 2{-}1] + e[2] + c[2{+}1, 2] = 0{+}20{+}0 = 20$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

	0	1	2	3	4	5	j
1	0	10					
2		0	20				
3			0	??			
4				0			
5					0		
6						0	
i							

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

	0	1	2	3	4	5	j
1	0	10					
2		0	20				
3			0	30			
4				0			
5					0		
6						0	
i							

$$c[3, 3{-}1] + e[3] + c[3{+}1, 3] = 0{+}30{+}0 = 30$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10				
2		0	20			
3			0	30		
4				0	??	
5					0	
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

	0	1	2	3	4	5	j
1	0	10					
2		0	20				
3			0	30			
4				0	15		
5					0		
6						0	
i							

$$c[4,4{+}1] + e[4] + c[4{+}1,4] = 0{+}15{+}0 = 15$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10				
2		0	20			
3			0	30		
4				0	15	
5					0	??
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

	0	1	2	3	4	5
1	0	10				
2		0	20			
3			0	30		
4				0	15	
5					0	30
6						0

i (rows), j (columns)

$$c[5, 5{+}1] + e[5] + c[5{+}1, 5] = 0{+}3000{+}0 = 30$$

Simulação

$e[1]{=}10$ $e[2]{=}20$ $e[3]{=}30$ $e[4]{=}15$ $e[5]{=}30$

	0	1	2	3	4	5	j
1	0	10	??				
2		0	20				
3			0	30			
4				0	15		
5					0	30	
6						0	
i							

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	50			
2		0	20			
3			0	30		
4				0	15	
5					0	30
6						0

$$c[1, 1{-}1] + (e[1] + e[2]) + c[1{+}1, 2] = 0{+}30{+}20 = 50$$

Simulação

	$e[1]{=}10$	$e[2]{=}20$	$e[3]{=}30$	$e[4]{=}15$	$e[5]{=}30$	
	0	1	2	3	4	5	j
1	0	10	40			
2		0	20			
3			0	30		
4				0	15	
5					0	30
6						0
i						

$$c[1, 2{-}1] + (e[1] + e[2]) + c[2{+}1, 2] = 10{+}30{+}0 = 40$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	??		
3			0	30		
4				0	15	
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	80		
3			0	30		
4				0	15	
5					0	30
6						0

$$c[2, 2{-}1] + (e[2] + e[3]) + c[2{+}1, 3] = 0{+}50{+}30 = 80$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30		
4				0	15	
5					0	30
6						0

$$c[2, 3{-}1] + (e[2] + e[3]) + c[3{+}1, 3] = 20{+}50{+}0 = 70$$

Simulação

$e[1]{=}10$ $e[2]{=}20$ $e[3]{=}30$ $e[4]{=}15$ $e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	??	
4				0	15	
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	60	
4				0	15	
5					0	30
6						0

$$c[3, 3{-}1] + (e[3] + e[4]) + c[3{+}1, 4] = 0{+}45{+}15 = 60$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	60	
4				0	15	
5					0	30
6						0

$$c[3, 4{-}1] + (e[3] + e[4]) + c[4{+}1, 4] = 30{+}45{+}0 = 75$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	60	
4				0	15	??
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	60	
4				0	15	75
5					0	30
6						0

$$c[4, 4{-}1] + (e[4] + e[5]) + c[4{+}1, 5] = 0{+}45{+}30{=}75$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40			
2		0	20	70		
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[4,5{-}1] + (e[4] + e[5]) + c[5{+}1,5] = 15{+}45{+}0 = 60$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	??		
2		0	20	70		
3			0	30	60	
4				0	15	60
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	130		
2		0	20	70		
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[1, 1{-}1] + (e[1] + e[2] + e[3]) + c[1{+}1, 3] = 0{+}60{+}70 = 130$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70		
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[1,2{-}1] + (e[1]) + e[2] + e[3]) + c[2{+}1,3] = 10{+}60{+}30 = 100$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70		
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[1, 3{-}1] + (e[1] + e[2] + e[3]) + c[3{+}1, 3] = 40{+}60{+}0 = 100$$

Simulação

$e[1]{=}10$ $e[2]{=}20$ $e[3]{=}30$ $e[4]{=}15$ $e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	??	
3			0	30	60	
4				0	15	60
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	125	
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[2, 2{-}1] + (e[2] + e[3] + e[4]) + c[2{+}1, 4] = 0{+}65{+}60 = 125$$

Simulação

$e[1]=10 \quad e[2]=20 \quad e[3]=30 \quad e[4]=15 \quad e[5]=30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[2,3-1] + (e[2] + e[3] + e[4]) + c[3+1,4] = 20+65+15 = 100$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	
4				0	15	60
5					0	30
6						0

$$c[2, 4-1] + (e[2] + e[3] + e[4]) + c[4+1, 4] = 70+65+0 = 135$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	??
4				0	15	60
5					0	30
6						0

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	135
4				0	15	60
5					0	30
6						0

$$c[3, 3{-}1] + (e[3] + e[4] + e[5]) + c[3{+}1, 5] = 0{+}75{+}60 = 135$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

$i \backslash j$	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	135
4				0	15	60
5					0	30
6						0

$$c[3, 4{-}1] + (e[3] + e[4] + e[5]) + c[4{+}1, 5] = 30{+}75{+}30 = 135$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	135
4				0	15	60
5					0	30
6						0

$$c[3,5{-}1] + (e[3] + e[4] + e[5]) + c[5{+}1,5] = 60{+}75{+}0 = 135$$

Simulação

$e[1]{=}10 \quad e[2]{=}20 \quad e[3]{=}30 \quad e[4]{=}15 \quad e[5]{=}30$

i \ j	0	1	2	3	4	5
1	0	10	40	100		
2		0	20	70	100	
3			0	30	60	135
4				0	15	60
5					0	30
6						0

Exercício: Preencha o que falta!

Árvore de busca ótima

ABB-ÓTIMA (e, n)

$s[0] = 0$
para $i \leftarrow 1$ **até** n **faça**
    $s[i] \leftarrow s[i-1] + e[i]$
para $i \leftarrow 1$ **até** $n+1$ **faça**
    $c[i][i-1] \leftarrow 0$
para $\ell \leftarrow 1$ **até** n **faça**
    para $i \leftarrow 1$ **até** $n-\ell+1$ **faça**
        $j \leftarrow i+\ell-1$
        $c[i][j] \leftarrow c[i+1][j]$
        para $k \leftarrow i+1$ **até** j **faça**
            se $c[i][k-1] + c[k+1][j] < c[i][j]$
            então $c[i][j] \leftarrow c[i][k-1] + c[k+1][j]$
        $c[i][j] \leftarrow c[i][j] + s[j] - s[i-1]$
devolva $c[1, n]$

Árvore de busca ótima

Exercício: Como fazer para obter uma ABB ótima e não apenas o seu custo? Complete o serviço!

Vários exercícios na lista 5.

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \le W \quad \text{e} \quad 0 \le x[i] \le 1 \text{ para todo } i$$

O valor de uma mochila é o número $x \cdot v$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1\,..\,n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila boolena ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$$(w, v, n-1, W - w[n])$$

senão $x[1\,..\,n]$ é mochila boolena ótima para
$$(w, v, n-1, W)$$

NOTA. Não há nada de especial acerca do índice n. Uma afirmação semelhante vale para qualquer índice i.

Simplificação

Problema: encontrar o valor de uma mochila booleana ótima.

$$
\begin{aligned}
t[i, Y] = &\ \text{valor de uma mochila booleana ótima} \\
&\ \text{para } (w, v, i, Y) \\
= &\ \text{valor da expressão } x \cdot v \text{ sujeito às restrições}
\end{aligned}
$$

$$x \cdot w \leq Y,$$

onde x é uma mochila booleana ótima

Possíveis valores de Y: $0, 1, 2, \ldots, W$

Recorrência

$t[i, Y]$ = valor da expressão $x \cdot v$ sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i{-}1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i{-}1, Y{-}w[i]] + v[i]\}$ se $w[i] \leq Y$

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, n, W)
se $n = 0$ **ou** $W = 0$
  então devolva 0
se $w[n] > W$
  então devolva REC-MOCHILA $(w, v, n-1, W)$
$a \leftarrow$ REC-MOCHILA $(w, v, n-1, W)$
$b \leftarrow$ REC-MOCHILA $(w, v, n-1, W-w[n]) + v[n]$
devolva $\max\{a, b\}$

Consumo de tempo no pior caso é $\Omega(2^n)$

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Olhe a recorrência e pense...

$t[i, Y] = t[i{-}1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i{-}1, Y{-}w[i]] + v[i]\}$ se $w[i] \leq Y$

Programação dinâmica

i \ Y	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0	$\star$	$\star$	$\star$	$\star$	$\star$		
3	0					??		
4	0							
5	0							
6	0							
7	0							

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

	0	1	2	3	4	5 Y
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	400	700	800
4	0	300	400	450	750	850
i						

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

MOCHILA-BOOLEANA (w, v, n, W)

1 **para** $Y \leftarrow 0$ **até** W **faça**
2 $\quad t[0, Y] \leftarrow 0$
3 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
4 $\quad\quad a \leftarrow t[i{-}1, Y]$
5 $\quad\quad$ **se** $w[i] > Y$
6 $\quad\quad\quad$ **então** $b \leftarrow 0$
7 $\quad\quad\quad$ **senão** $b \leftarrow t[i{-}1, Y{-}w[i]] + v[i]$
8 $\quad\quad t[i, Y] \leftarrow \max\{a, b\}$
9 **devolva** $t[n, W]$

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo
MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o “tamanho” de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
mais lento que o algoritmo força bruta!

Obtenção da mochila

MOCHILA (w, n, W, t)

$Y \leftarrow W$

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

  se $t[i, Y] = t[i-1, Y]$

    então $x[i] \leftarrow 0$

    senão $x[i] \leftarrow 1$

      $Y \leftarrow Y - w[i]$

devolva x

Consumo de tempo é $\Theta(n)$.

Versão recursiva

```
MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)
    para i ← 0 até n faça
        para Y ← 0 até W faça
            t[i, Y] ← ∞
    devolva LOOKUP-MOC (w, v, n, W)
```

Versão recursiva

LOOKUP-MOC (w, v, i, Y)

    se $t[i, Y] < \infty$
        então devolva $t[i, Y]$
    se $n = 0$ **ou** $Y = 0$ **então** $t[i, Y] \leftarrow 0$
    senão
        se $w[n] > Y$
            então
                $t[i, Y] \leftarrow$ LOOKUP-MOC $(w, v, n-1, Y)$
            senão
                $a \leftarrow$ LOOKUP-MOC $(w, v, i-1, Y)$
                $b \leftarrow$ LOOKUP-MOC $(w, v, i-1, Y-w[i]) + v[i]$
                $t[i, Y] \leftarrow \max\{a, b\}$
    devolva $t[i, Y]$

Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

O valor de uma mochila é o número $x \cdot v$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1\,..\,n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

MOCHILA-BOOLEANA (w, v, n, W)

para $Y \leftarrow 0$ **até** W **faça**
    $t[0, Y] \leftarrow 0$
    para $i \leftarrow 1$ **até** n **faça**
        $a \leftarrow t[i{-}1, Y]$
        se $w[i] > Y$
            então $b \leftarrow 0$
            senão $b \leftarrow t[i{-}1, Y{-}w[i]] + v[i]$
        $t[i, Y] \leftarrow \max\{a, b\}$
devolva $t[n, W]$

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo
MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
mais lento que o algoritmo força bruta!

Obtenção da mochila

MOCHILA (w, n, W, t)
```
Y ← W
para i ← n decrescendo até 1 faça
    se t[i, Y] = t[i−1, Y]
        então x[i] ← 0
        senão x[i] ← 1
              Y ← Y − w[i]
devolva x
```

Consumo de tempo é $\Theta(n)$.

Versão recursiva

MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)

para $i \leftarrow 0$ **até** n **faça**
  para $Y \leftarrow 0$ **até** W **faça**
    $t[i, Y] \leftarrow \infty$
devolva LOOKUP-MOC (w, v, n, W)

Versão recursiva

LOOKUP-MOC (w, v, i, Y)

```
se t[i, Y] < ∞
    então devolva t[i, Y]
se n = 0 ou Y = 0 então t[i, Y] ← 0
senão
    se w[n] > Y
        então
            t[i, Y] ← LOOKUP-MOC (w, v, n−1, Y)
        senão
            a ← LOOKUP-MOC (w, v, i−1, Y)
            b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
            t[i, Y] ← max {a, b}
devolva t[i, Y]
```

Algoritmos gulosos (*greedy*)

CLRS 16.1–16.3

Algoritmos gulosos

"A *greedy algorithm* starts with a solution to a very small subproblem and augments it successively to a solution for the big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, without regard to whether it will lead to a good long-term or global solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

"One thing you will notice about greedy algorithms is that they are usually easy to design, easy to implement, easy to analyse, and they are very fast, but they are *almost always difficult to prove correct*."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Problema fracionário da mochila

Problema: Dados (w, v, n, W), encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$
x	1	$1/3$	0	0	valor $= 1040$

A propósito . . .

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$\begin{array}{rrll} \text{maximize} & x \cdot v & & \\ \text{sob as restrições} & x \cdot w & \leq W & \\ & x[i] & \geq 0 & \text{para } i = 1, \ldots, n \\ & x[i] & \leq 1 & \text{para } i = 1, \ldots, n \end{array}$$

PL's podem ser resolvidos por

SIMPLEX: no pior caso consome tempo exponencial
na prática é muito rápido

ELIPSÓIDES: consome tempo polinomial
na prática é lento

PONTOS-INTERIORES: consome tempo polinomial
na prática é rápido

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila ótima para o problema (w, v, n, W).

Se $x[n] = \delta$

então $x[1\,..\,n{-}1]$ é mochila ótima para

$$(w, v, n-1, W - \delta\, w[n])$$

NOTA. Não há nada de especial acerca do índice n. Uma afirmação semelhante vale para qualquer índice i.

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i.

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i

então EXISTE uma mochila ótima $x[1\,..\,n]$ tal que

$$x[n] = \min\left\{1, \frac{W}{w[n]}\right\}$$

Algoritmo guloso

Esta propriedade da escolha gulosa sugere um algoritmo que atribui os valores de $x[1\,.\,.\,n]$ supondo que os dados estejam em ordem decrescente de “valor específico” :

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \cdots \leq \frac{v[n]}{w[n]}$$

É nessa ordem “mágica” que está o segredo do funcionamento do algoritmo.

Algoritmo guloso

Devolve uma mochila ótima para (w, v, n, W).

MOCHILA-FRACIONÁRIA (w, v, n, W)

```
0   ordene w e v de tal forma que
            v[1]/w[1] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
1   para i ← n decrescendo até 1 faça
2       se w[i] ≤ W
3           então x[i] ← 1
4                 W ← W − w[i]
5           senão x[i] ← W/w[i]
6                 W ← 0
7   devolva x
```

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.
Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Invariante

Seja W_0 o valor original de W.
No início de cada execução da linha 1 vale que

(i0) $x' = x[i{+}1 \mathinner{..} n]$ é mochila ótima para

$$(w', v', n', W_0)$$

onde

$$w' = w[i{+}1 \mathinner{..} n]$$
$$v' = v[i{+}1 \mathinner{..} n]$$
$$n' = n - i$$

Na última iteração $i = 0$ e portanto $x[1 \mathinner{..} n]$ é mochila ótima para (w, v, n, W_0).

Conclusão

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

Escolha gulosa

Precisamos mostrar que se $x[1\,.\,.\,n]$ é uma mochila ótima, então podemos supor que

$$x[n] = \alpha := \min\left\{1, \frac{W}{w[n]}\right\}$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica: transformar uma solução ótima em uma solução ótima 'gulosa'.

Esta transformação é semelhante ao processo de pivotação feito pelo algoritmo SIMPLEX para programação linear.

Algoritmos gulosos

Algoritmo guloso

- 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*)

Exercício: O problema da mochila booleana pode ser resolvido por um algoritmo guloso?

Algoritmos gulosos (*greedy*)

CLRS 16.1 e mais...

Algoritmos gulosos

Algoritmo guloso

- 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*)

Problema dos intervalos disjuntos

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

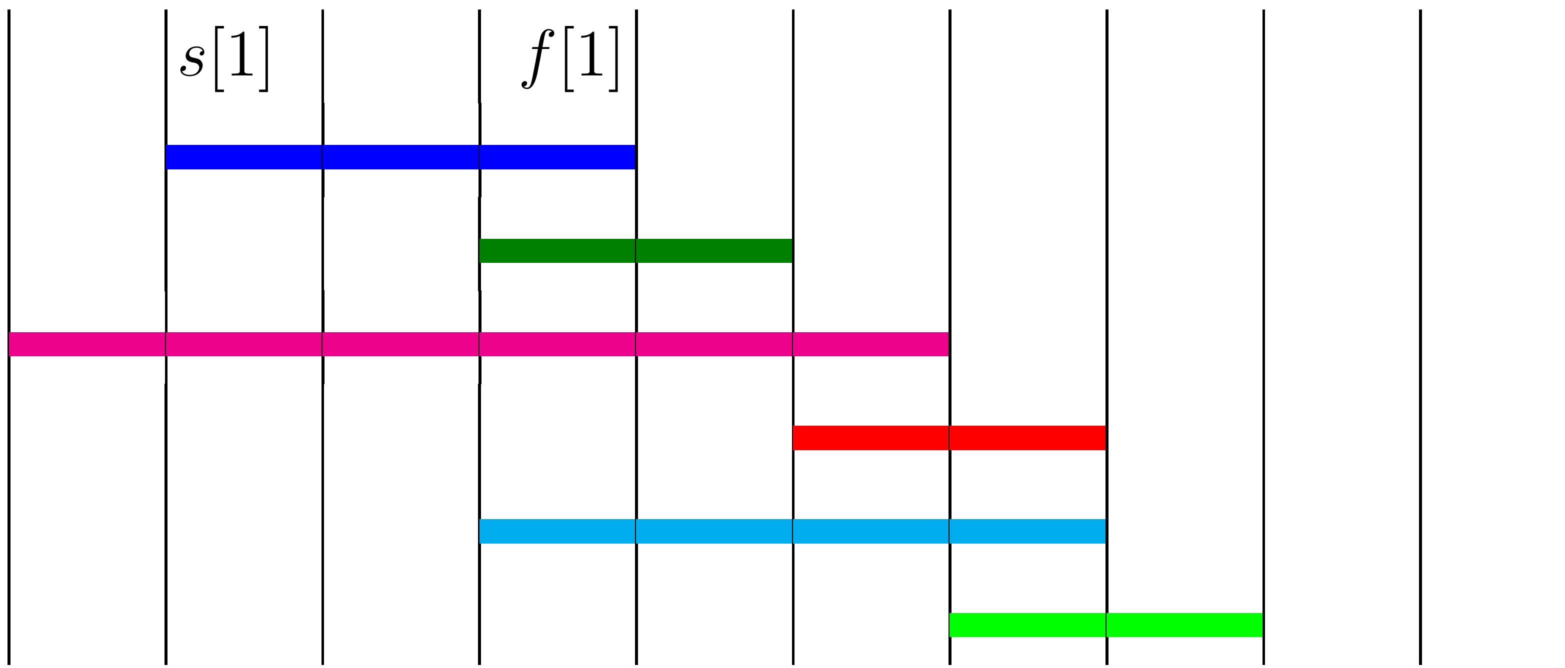
Solução é um subconjunto A de $\{1, \ldots, n\}$.

Problema dos intervalos disjuntos

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

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

Exemplo:

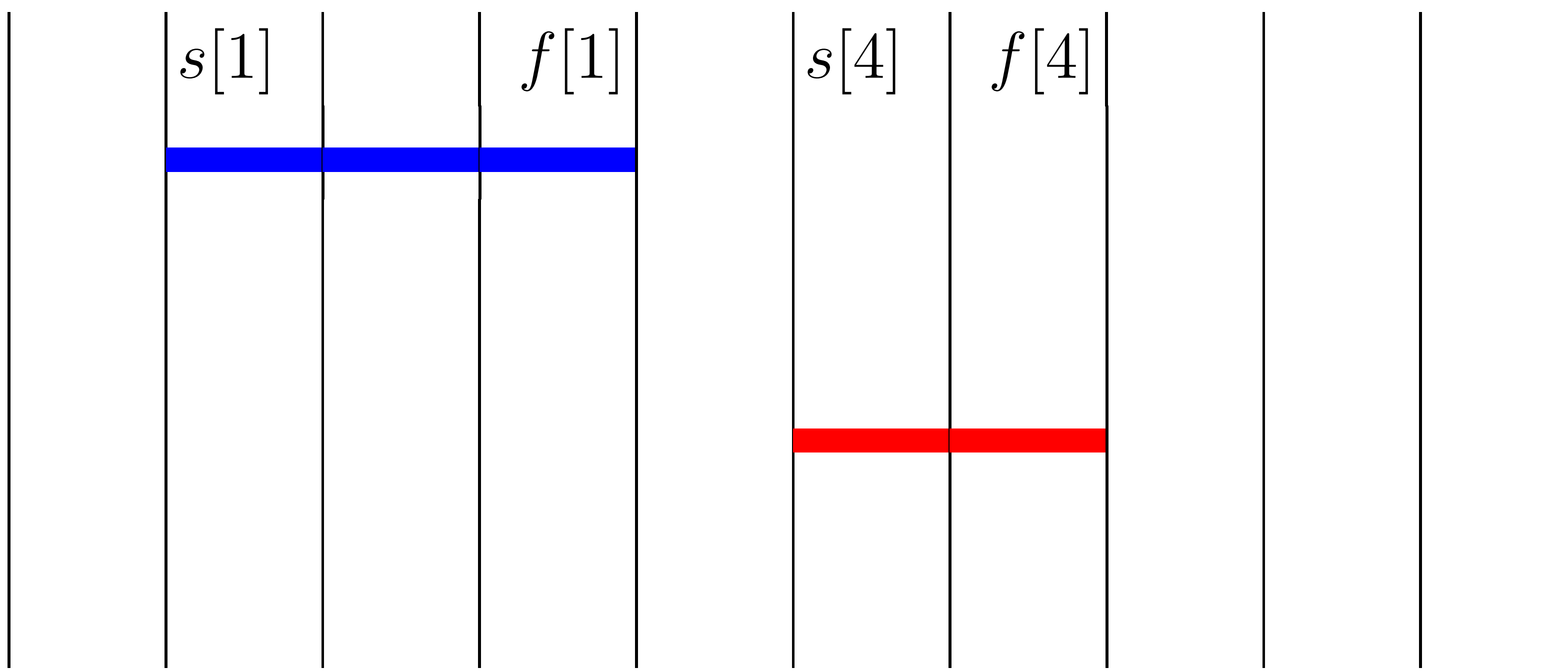


Problema dos intervalos disjuntos

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

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

Solução:

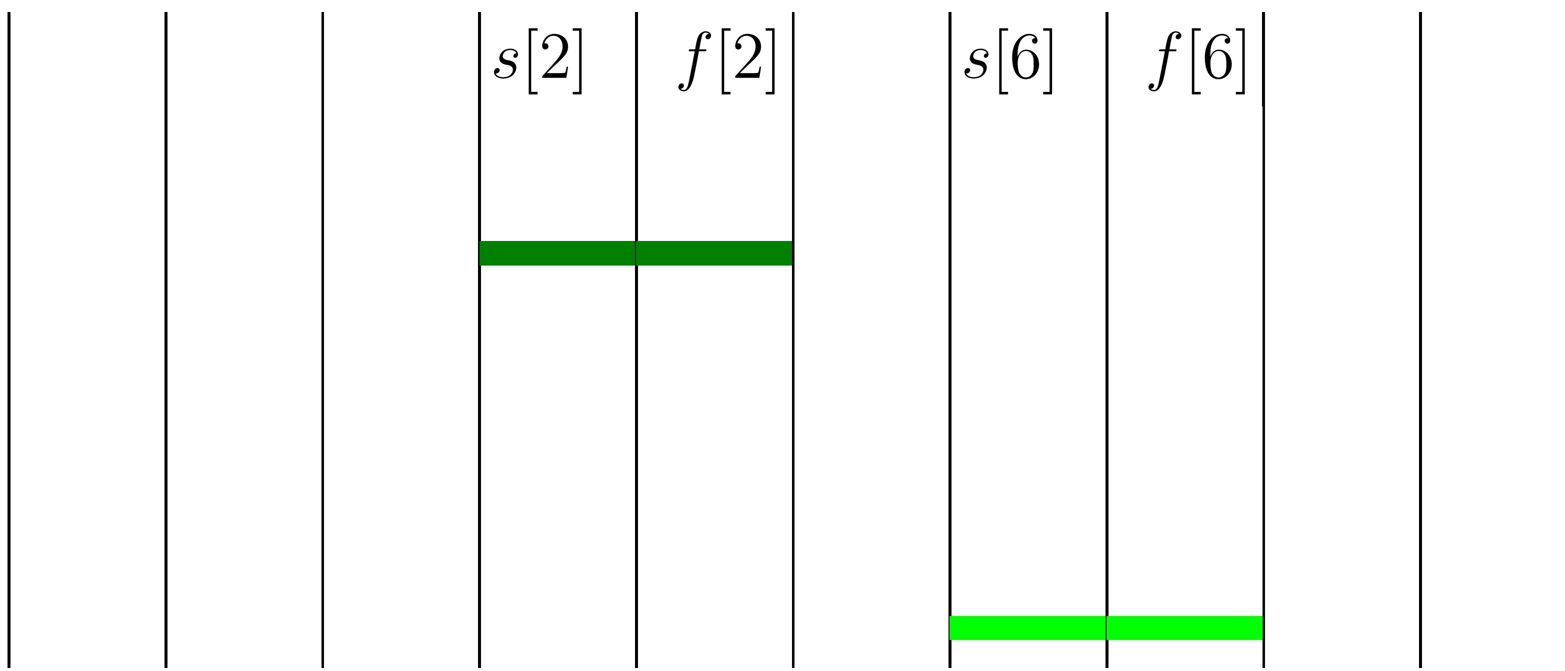


Problema dos intervalos disjuntos

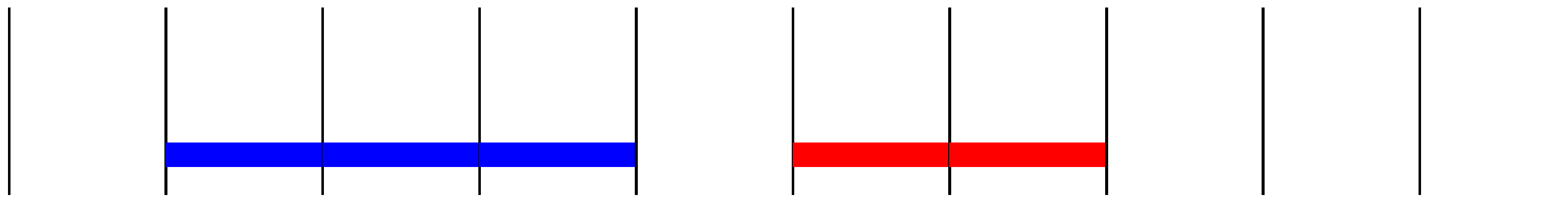
Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

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

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 $S := \{1, \ldots, n\}$

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 $S := \{1, \ldots, n\}$

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 $s[1] \leq s[2] \leq \cdots \leq s[n]$

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

$$t[n] = 1$$

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

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

Algoritmo de programação dinâmica

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

ordene s e f de tal forma que

$$s[1] \leq s[2] \leq \cdots \leq s[n]$$

$A[n+1] \leftarrow \emptyset$
para $i \leftarrow n$ **decrescendo até** 1 **faça**
    $A[i] \leftarrow A[i+1]$
    $k \leftarrow i+1$
    enquanto $k \leq n$ e $s[k] < f[i]$ **faça**
        $k \leftarrow k+1$
    se $|A[i]| < 1 + |A[k]|$
        então $A[i] \leftarrow \{i\} \cup A[k]$
devolva $A[1]$

Consumo de tempo é $\Theta(n^2)$.

Conclusão

Invariante: na linha 2 vale que

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

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

Escolha gulosa

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

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

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

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 $\Theta(n \lg n)$.

Coloração de intervalos

Problema: Dados intervalos de tempo $[s_1, f_2), \ldots, [s_n, f_n)$, encontrar uma coloração dos intervalos com o menor número possível de cores em que dois intervalos de mesma cor sempre sejam disjuntos.

Solução: uma partição de $\{1, \ldots, n\}$ em coleções de intervalos dois a dois disjuntos.

Motivação

Queremos distribuir um conjunto de atividades no menor número possivel de salas.

Cada atividade a_i ocupa um certo intervalo de tempo $[s_i, f_i)$ e duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos.

Cada sala corresponde a uma cor. Queremos usar o menor número possível de cores para pintar todos os intervalos.

Coloração de intervalos

Estratégias gulosas:

- Encontre uma coleção disjunta máxima de intervalos, pinte com a próxima cor disponível e repita a idéia para os intervalos restantes.
- Ordene as atividades de maneira que $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.
- Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Quais destas estratégias funcionam?
Quais não funcionam?

Coloração de intervalos

Problema: Dados intervalos de tempo $[s_1, f_1), \ldots, [s_n, f_n)$, encontrar uma coloração dos intervalos com o menor número possível de cores em que dois intervalos de mesma cor sempre sejam disjuntos.

Solução: uma partição de $\{1, \ldots, n\}$ em coleções de intervalos dois a dois disjuntos.

Motivação

Queremos distribuir um conjunto de atividades no menor número possivel de salas.

Cada atividade a_i ocupa um certo intervalo de tempo $[s_i, f_i)$ e duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos.

Cada sala corresponde a uma cor. Queremos usar o menor número possível de cores para pintar todos os intervalos.

Coloração de intervalos

Estratégias gulosas:

- Encontre uma coleção disjunta máxima de intervalos, pinte com a próxima cor disponível e repita a idéia para os intervalos restantes.
- Ordene as atividades de maneira que $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.
- Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Quais destas estratégias funcionam?
Quais não funcionam?

Estratégia 1

Encontre uma coleção disjunta máxima de intervalos,
pinte com a próxima cor disponível
e repita a idéia para os intervalos restantes.

Estratégia 1

Encontre uma coleção disjunta máxima de intervalos,
pinte com a próxima cor disponível
e repita a idéia para os intervalos restantes.

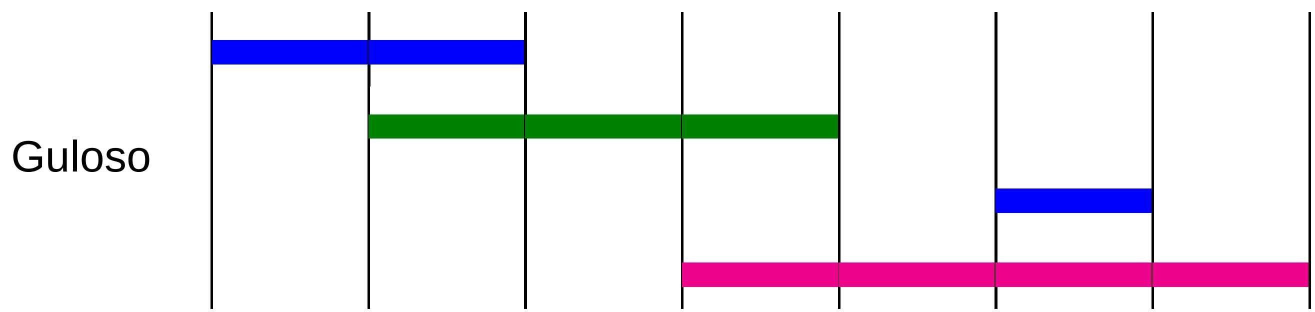
Não funciona...

Estratégia 1

Encontre uma coleção disjunta máxima de intervalos,
pinte com a próxima cor disponível
e repita a idéia para os intervalos restantes.

Não funciona...

Exemplo:



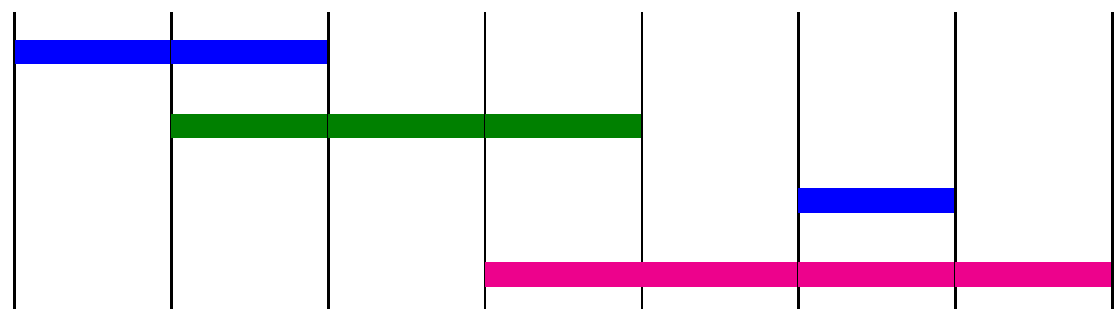
Estratégia 1

Encontre uma coleção disjunta máxima de intervalos,
pinte com a próxima cor disponível
e repita a idéia para os intervalos restantes.

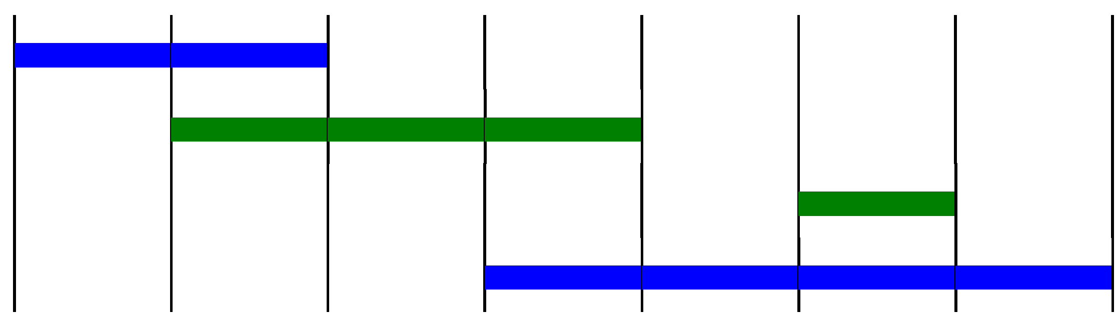
Não funciona...

Exemplo:

Guloso



Ótimo



Estratégia 2

Ordene as atividades de maneira que $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Estratégia 2

Ordene as atividades de maneira que
$f[1] \le f[2] \le \cdots \le f[n]$ e pinte uma a uma nesta ordem
sempre usando a menor cor possível para aquela atividade.

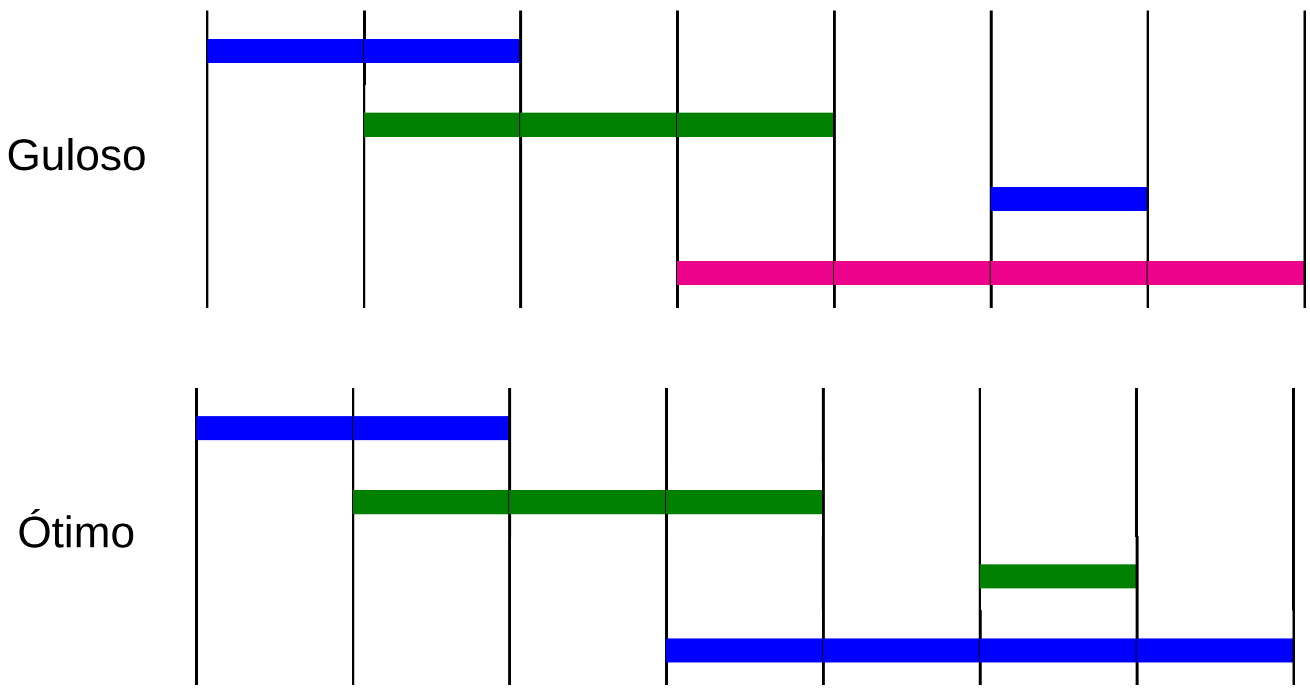
Não funciona de novo...

Estratégia 2

Ordene as atividades de maneira que $f[1] \leq f[2] \leq \cdots \leq f[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Não funciona de novo...

Mesmo exemplo:



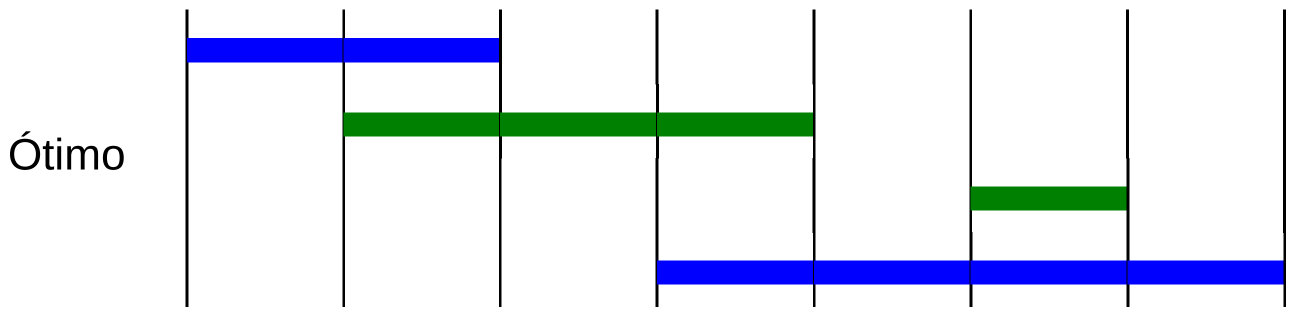
Estratégia 3

Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Estratégia 3

Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

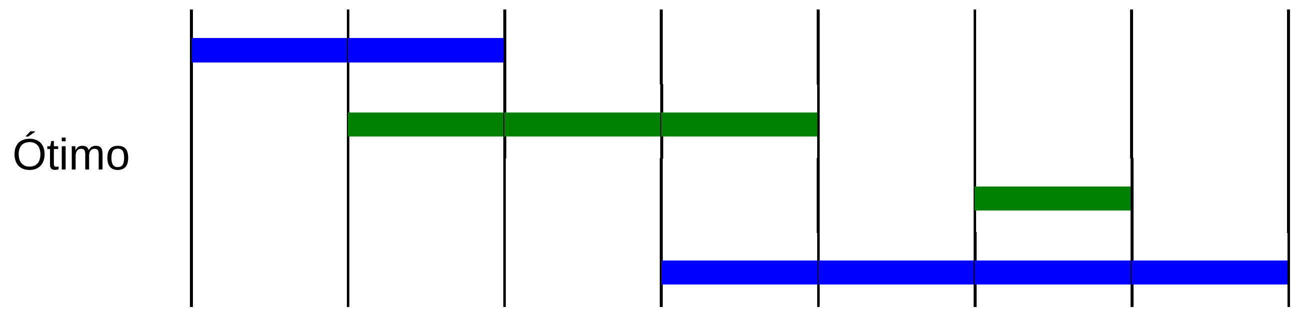
Pelo menos funciona para o exemplo...



Estratégia 3

Ordene as atividades de maneira que $s[1] \leq s[2] \leq \cdots \leq s[n]$ e pinte uma a uma nesta ordem sempre usando a menor cor possível para aquela atividade.

Pelo menos funciona para o exemplo...



De fato, funciona sempre!

A seguir, apresentamos o algoritmo guloso obtido.

No algoritmo, para uma cor j, o número $\ell[j]$ indica o final da última tarefa que foi colorida com a cor j.

Algoritmo guloso

Devolve coloração dos intervalos com menor número de cores.

COLORAÇÃO-INTERVALOS (s, f, n)

ordene s e f de tal forma que $s[1] \leq s[2] \leq \cdots \leq s[n]$
$k \leftarrow 0$
para $i \leftarrow 1$ **até** n **faça**
    $j \leftarrow 1$
    enquanto $j \leq k$ **e** $\ell[j] > s[i]$ **faça** $j \leftarrow j + 1$
    se $j > k$
        então $k \leftarrow k + 1$
            $D[k] \leftarrow \{i\}$
            $\ell[k] \leftarrow f[i]$
        senão $D[j] \leftarrow D[j] \cup \{i\}$
            $\ell[j] \leftarrow f[i]$
devolva $D[1], \ldots, D[k]$

Consumo de tempo

Observe que o algoritmo consome tempo $\mathrm{O}(n^2)$.

Como observado na aula, é possível obter um certificado de que o algoritmo apresentado utiliza o menor número de cores.

Exercício: Modifique o algoritmo para que ele devolva um certificado de que sua coloração usa um número mínimo de cores.

Demonstração

Seja $B[1], \ldots, B[k^*]$ uma solução ótima que coincide com a solução gulosa $D[1], \ldots, D[k]$ quando restrita aos intervalos em $\{1, \ldots, i-1\}$, com i o maior possível.

Se $i = n+1$, então não há nada a provar.
Senão, seja j tal que $i \in D[j]$ e j^* tal que $i \in B[j^*]$.

Seja X o conjunto $B[j] \cap \{i, \ldots, n\}$ e
Y o conjunto $B[j^*] \cap \{i, \ldots, n\}$.
Seja B' a partição de $\{1, \ldots, n\}$ tal que

$$
\begin{aligned}
B'[t] &= B[t] \quad \text{se } t \neq j \text{ e } t \neq j^* \\
B'[j] &= B[j] \setminus X \cup Y \\
B'[j^*] &= B[j^*] \setminus Y \cup X
\end{aligned}
$$

B' é uma solução ótima que contraria a escolha de B.

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \ldots, n$

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \ldots, n$

t_i: duração da tarefa i

d_i: prazo de entrega da tarefa i

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \ldots, n$

t_i: duração da tarefa i

d_i: prazo de entrega da tarefa i

Escalonamento: permutação de 1 a n

Para um escalonamento π, o tempo de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)}$$

(soma da duração das tarefas anteriores a i).

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \ldots, n$

t_i: duração da tarefa i

d_i: prazo de entrega da tarefa i

Escalonamento: permutação de 1 a n

Para um escalonamento π, o tempo de início da tarefa i é

$$s_i = \sum_{j=1}^{\pi(i)-1} t_{\pi^{-1}(j)}$$

(soma da duração das tarefas anteriores a i).

O tempo de término da tarefa i é $f_i = s_i + t_i$.

Problema de escalonamento

Para um escalonamento π, o tempo de início da tarefa i é soma da duração das tarefas anteriores a i.

O tempo de término da tarefa i é $f_i = s_i + t_i$.

Problema de escalonamento

Para um escalonamento π, o tempo de início da tarefa i é soma da duração das tarefas anteriores a i.

O tempo de término da tarefa i é $f_i = s_i + t_i$.

O atraso da tarefa i é o número

$$\ell_i = \begin{cases} 0 & \text{se } f_i \leq d_i \\ f_i - d_i & \text{se } f_i > d_i . \end{cases}$$

Problema de escalonamento

Para um escalonamento π, o tempo de início da tarefa i é soma da duração das tarefas anteriores a i.

O tempo de término da tarefa i é $f_i = s_i + t_i$.

O atraso da tarefa i é o número

$$\ell_i = \begin{cases} 0 & \text{se } f_i \leq d_i \\ f_i - d_i & \text{se } f_i > d_i. \end{cases}$$

Problema: Dados $t_1, \ldots, t_n$ e $d_1, \ldots, d_n$, encontrar um escalonamento com o menor atraso máximo.

Ou seja, que minimize $L = \max_i \ell_i$.

Estratégias gulosas

- escalonar primeiro as tarefas mais curtas
  (ignoro o prazo de entrega)

Estratégias gulosas

- escalonar primeiro as tarefas mais curtas
  (ignoro o prazo de entrega)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 10$, $t_2 = 8$, $d_2 = 8$

  Guloso: $1, 2, L = 1$ Solução: $2, 1, L = 0$

Estratégias gulosas

- escalonar primeiro as tarefas mais curtas
  (ignoro o prazo de entrega)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 10$, $t_2 = 8$, $d_2 = 8$

  Guloso: 1, 2, $L = 1$ Solução: 2, 1, $L = 0$

- escalonar primeiro as tarefas com menos $d_i - t_i$
  (tarefas com menor folga)

Estratégias gulosas

- escalonar primeiro as tarefas mais curtas
  (ignoro o prazo de entrega)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 10$, $t_2 = 8$, $d_2 = 8$

  Guloso: $1, 2, L = 1$ Solução: $2, 1, L = 0$

- escalonar primeiro as tarefas com menos $d_i - t_i$
  (tarefas com menor folga)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 2$, $t_2 = 10$, $d_2 = 10$

  Guloso: $2, 1, L = 9$ Solução: $1, 2, L = 1$

Estratégias gulosas

- escalonar primeiro as tarefas mais curtas
  (ignoro o prazo de entrega)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 10$, $t_2 = 8$, $d_2 = 8$

  Guloso: $1, 2, L = 1$ Solução: $2, 1, L = 0$

- escalonar primeiro as tarefas com menos $d_i - t_i$
  (tarefas com menor folga)

  Não funciona...

  Exemplo: $t_1 = 1$, $d_1 = 2$, $t_2 = 10$, $d_2 = 10$

  Guloso: $2, 1, L = 9$ Solução: $1, 2, L = 1$

- escalonar primeiro as tarefas com menor prazo
  (ignoro a duração)

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \cdots \leq d[\pi[n]]$$

2 **devolva** π

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \cdots \leq d[\pi[n]]$$

2 **devolva** π

Consumo de tempo: $\mathrm{O}(n \lg n)$.

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)
1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \cdots \leq d[\pi[n]]$$

2 **devolva** π

Consumo de tempo: $\mathrm{O}(n \lg n)$.

Na aula, vimos a prova de que este algoritmo está correto (devolve um escalonamento com atraso máximo mínimo).

Ingredientes da prova

Uma **inversão** em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Ingredientes da prova

Uma inversão em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Ingredientes da prova

Uma inversão em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Mostre que se um escalonamento ótimo tem uma inversão, então ele tem uma inversão do tipo $(i, i+1)$.

Ingredientes da prova

Uma inversão em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Mostre que se um escalonamento ótimo tem uma inversão, então ele tem uma inversão do tipo $(i, i+1)$.

Mostre então que, se trocarmos $\pi[i]$ e $\pi[i+1]$, obteremos um escalomento com atraso máximo não superior ao atraso máximo de π.

Exercícios

Exercício 22.A [CLRS 16.1-1]
Escreva um algoritmo de programação dinâmica para resolver o problema dos intervalos disjuntos. (Versão simplificadda 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.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 $\mathrm{O}(n \lg n)$. 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.

Exercício 22.G [Bin-packing]
São dados objetos $1, \ldots, n$ 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 seguinte 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 herísticas 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 $w_1, \ldots, w_n$ aleatoriamente, todos no intervalo $(0, u)$.

Mais exercícios ainda

Exercício 22.H [parte de CLRS 16-4, modificado]
Seja $1, \ldots, n$ 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 $1\,..\,p_t$. 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 a “optimal substructure” apropriadas). Analise o consumo de tempo.