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**
    $B[i] \leftarrow \text{NIL}$
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 $A[1\,..\,n]$ 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 $X_i = \sum_j X_{ij}$.

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 $Y_i \le {X_i}^2$.

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

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

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

Logo,

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

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)
1   se n ≤ 1
2       então devolva n
3       senão a ← FIBO-REC (n − 1)
4             b ← FIBO-REC (n − 2)
5             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	38.4[illegible]

Recorrência

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

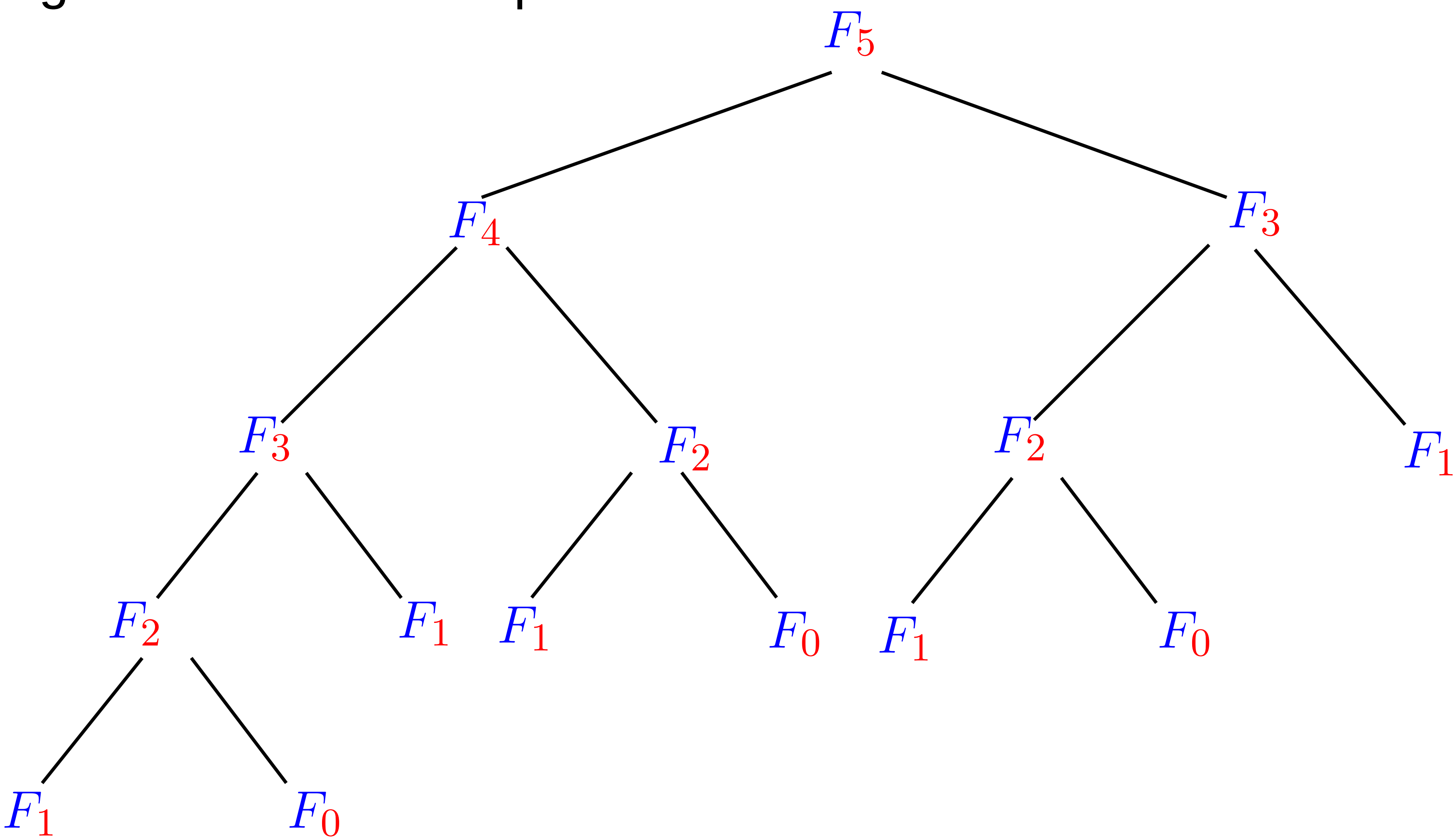
$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\overset{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2+1)\,(3/2)^{n-2} + 1 \\
&> (5/2)\,(3/2)^{n-2} \\
&> (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n\,.
\end{aligned}
$$

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] ← 0
    f[1] ← 1
    para i ← 2 até n faça
        f[i] ← f[i − 1] + f[i − 2]
    devolva f[n]
```

Note a tabela $f[0 \mathinner{..} n{-}1]$.

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 $\leftarrow 0$
f_atual $\leftarrow 1$
para $i \leftarrow 2$ **até** n **faça**
    f_prox $\leftarrow$ f_atual $+$ f_ant
    f_ant $\leftarrow$ f_atual
    f_atual $\leftarrow$ f_prox
devolva f_atual

Versão recursiva eficiente

MEMOIZED-FIBO (f, n)

```
para i ← 0 até n faça
        f[i] ← −1
devolva LOOKUP-FIBO (f, n)
```

LOOKUP-FIBO (f, n)

```
se f[n] ≥ 0
        então devolva f[n]
se n ≤ 1
        então f[n] ← n
        senão f[n] ← 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 $c[1,n] + x[1] \leq c[2,n] + x[2]$

  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

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)

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

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 $\Theta(n)$.