Problema de Programação Dinâmica

10465 Homer Simpson

Cristiano Medeiros Dalbem Juliano Leal Gonçalves

Universidade Federal do Rio Grande do Sul

11 de Novembro de 2009

Definição do problema

Homer Simpson adora comer hamburguers. Além do seu habitual hamburguer do Krusty, existe um novo tipo de hamburguer na praça, que Homer tambem gosta. Para comer o primeiro, Homer leva m minutos, e o segundo, n minutos. Dados t minutos, tu tens que descobrir qual o maior número de hamburguers que Homer consegue comer sem perder tempo. Se ele tiver que perder tempo, ele tomará cerveja.

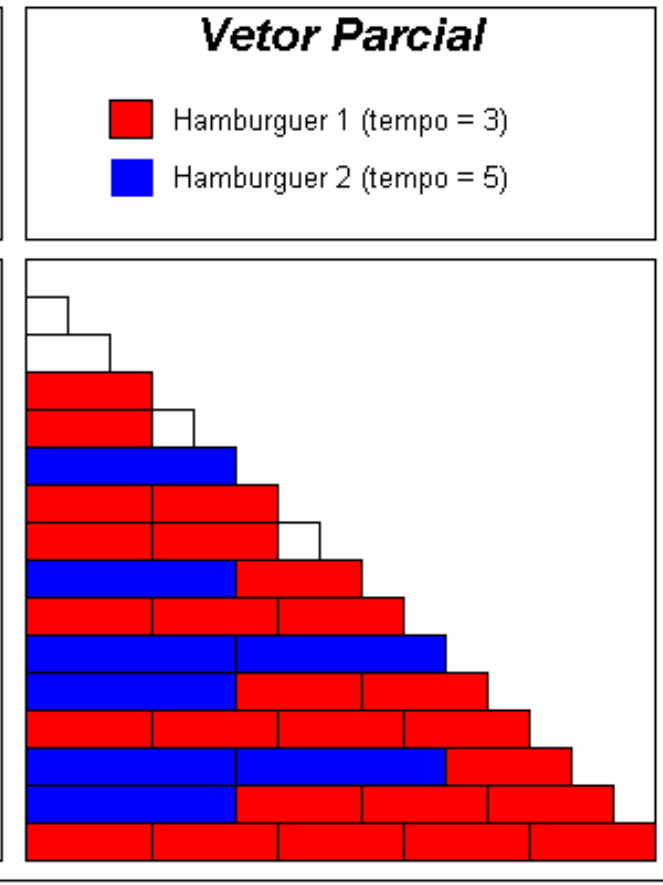
Definição recursiva

$$\text{Qtd\_cerveja}[x] = \begin{cases} x & , \text{se } x < m, n \\ Qtd\_cerveja[x-n] & , \text{se } n \leq x < m \\ Qtd\_cerveja[x-m] & , \text{se } m \leq x < n \\ min(Qtd\_cerveja[x-n], Qtd\_cerveja[x-m]) & , \text{se } x \geq m, n \end{cases}$$

$$\text{Qtd\_hamburguer}[x] = \begin{cases} 0 & , \text{se } x < m, n \\ 1 + Qtd\_hamburguer[x-n] & , \text{se}(n \leq x < m) ou (x \geq m, n \wedge Qtd\_cerveja[x-n] \leq Qtd\_cerveja[x-m]) \\ 1 + Qtd\_hamburguer[x-m] & , \text{se}(m \leq x < n) ou (x \geq m, n \wedge Qtd\_cerveja[x-m] < Qtd\_cerveja[x-n]) \end{cases}$$

Solução

Iteração	Hamburgers	Beer
0	0	0
1	0	1
2	0	2
3	1	0
4	1	1
5	1	0
6	2	0
7	2	1
8	2	0
9	3	0
10	2	0
11	3	0
12	4	0
13	3	0
14	4	0
15	5	0



Hamburger	0	0	0	1	1	1	2	2	2	3	2	3	4	3	4	5
Beer	0	1	2	0	1	0	0	1	0	0	0	0	0	0	0	0

```
1           structure vetor [ t + 1 ]
2
3           smaller := min(m, n)
4           bigger  := max(m, n)
5
6           for i := 0,...,t+1 do
7           {
8            if (i < smaller)
9
10            vetor[i].beerCount = i
11            vetor[i].hambCount = 0
12
13           else if (i < bigger)
14
15            vetor[i].beerCount := vetor[i - smaller].beerCount
16            vetor[i].hambCount := 1 + vetor[i - smaller].hambCount;
17
18                 else
19
20                   estadoMinimo := minimo_beerCount_entre(vetor[i - n],
                          vetor[i - m]);
21
22                   vetor[i].beerCount := estadoMinimo.beerCount
23                   vetor[i].hambCount += 1 + estadoMinimo.hambCount
24          }
25
26          if( vetor[t].beerCount )
27           print ( vetor[t].hambCount, vetor[t].beerCount )
28          else
29           print ( vetor[t].hambCount )
```

Solved problems

Problem	Ranking	Submission	Date	Run time
10465	768	7533720	2009-11-04 13:56:27	0.348

Análise de complexidade

Todas estas funções têm custo constante:

Descrição	Linhas
atribuições e declarações fora do laço principal	1-4
comparações em estruturas IF	8,13,26
atribuições dentro do laço principal	10-11,15-16,20-23
recuperação em vetores	15-16,22-23
soma de inteiros	23
função de comparação entre inteiros	20
funções de saída	27,29

O algoritmo itera em t independentemente dos valores de m e n. Portanto, a complexidade, além de ser a mesma para qualquer caso, depende apenas de t.

$$\sum_{0}^{t+1} c = (t+1)c = \Omega(t) = O(t) = \theta(t)$$

Perguntas???