23 / 09 / 19

☒ T Q Q S S D

Correção da 1ª lista

01) (+0,5)

(7) $n!$	(1) $\log n$	(6) 3^n	(1) $2.\log n^3$	(6) 3^{n+1}
(5) fib(n)	(4) n^3	(2) $n \log n$	(3) $6n^2$	(3) $2n + n^2$

02) (+0,5)

03)

- Melhor caso: Se $m \geq n$, o melhor caso ocorre quando A[0] não está contido em B. Se $m < n$, o melhor caso ocorre quando $A[i] = B[0]$ $(i = 0, 1, \ldots, m)$. A complexidade temporal no melhor caso é $\Theta(\min(m,n))$.
- Pior caso: Quando todos os elementos de A são iguais somente ao $B[n-1]$. A complexidade temporal no pior caso é $\Theta(m.n)$.
- Complexidade espacial: $O(1)$
- Região crítica: laço enquanto
- O algoritmo é eficiente, pois, no pior caso, requer tempo quadrático no tamanho da entrada.

	TPC $[m \cdot n]$	TG $[m+n]$
Se $m \geq n$	$O(m^2)$	$O(m)$
Se $m < n$	$O(n^2)$	$O(n)$

- Não é de cota inferior. Se ordenarmos um dos vetores, podemos resolver o problema em tempo linearítmico no tamanho da entrada.
- Prova:

Teorema: o algoritmo contido é correto

Prova: suponha que A está contido em B, nesse caso, a condição do Se nunca será satisfeita, e o algoritmo devolverá SIM. Suponha agora que A não está contido em B. Seja K a primeira posição de A cujo conteúdo não ocorre em B. Quando $i = K$, j será incrementado até tornar-se igual a n, logo a condição do Se será satisfeita e o algoritmo devolverá NÃO. ∎

04)

Algoritmo temPar

Entrada: um vetor de números v com n posições

Vaida: Sim, se v contém um par;
não, caso contrário

Para i = 0 até n-1
De v[i] % 2 = 0
Devolva SIM e pare
Devolva NÃO

- Melhor caso: V[0] é par CT: O(1)
- Pior caso: V não contém par. CT: O(n)
- Complexidade espacial: O(1)
- É eficiente, pois gasta tempo linear no tamanho da entrada.
- Só podemos afirmar com segurança que não há número par em V após inspecionarmos todas as posições de v. Logo, todo algoritmo correto para esse problema requer tempo $\Omega(n)$ no pior caso. Concluímos que o algoritmo é de cota inferior.

- Prova:

Teorema: o algoritmo temPar é correto

Prova: Suponha que V não contenha par. Nesse caso, a condição do De nunca será satisfeita. Ao sair do laço, o algoritmo devolverá NÃO. Suponha agora que V contenha um número par. Seja K a primeira posição de V cujo conteúdo é par ($0 \leq K \leq n-1$). Como i varia de 0 a n-1, quando i = K, a condição do De será satisfeita e o algoritmo devolverá SIM

05) a) $\begin{cases} T(n) = 2T(\lceil n/2 \rceil) + n \\ T(1) = 1 \end{cases}$

Vamos supor $n = 2^K \rightarrow K = \log_2 n$

ca 0 $T(n) = 2T(n/2) + n$

ca 1 $2T(n/2) = 4T(n/4) + n$

ca 2 $4T(n/4) = 8T(n/8) + n$

$\vdots$

ca k-1 $2^{k-1}T(n/2^{k-1}) = 2^k T(n/2^k) + n$

ca k $2^k T(n/2^k) = 2^k$

$T(n) = nK + 2^K$

$\triangleright T(n) = n \log_2 n + n$

$T(n) \in \Theta(n \log n)$

__/__/__ S T Q Q S S D

b) $T(n) = T(n-1) + n/2$, $T(1) = 1/2$

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

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

$T(2) = T(1) + 2/2$

$T(1) = 1/2$

$T(n) = \frac{1}{2} + \frac{2}{2} + \dots + \frac{(n-1)}{2} + \frac{n}{2} = \frac{1}{2}(1 + 2 + \dots + n)$

$T(n) = \frac{1}{2}\cdot\frac{(1+n)\cdot n}{2} = \frac{n^2+n}{4}$; $T(n) \in \Theta(n^2)$

06)

$L = \{8, 3, 5, 2, 9\}$ (posições 0, 1, 2, 3, 4)

$enigma(L,4) = 9 * enigma(L,3) = 9 * 2 * enigma(L,2) =$

$9 * 2 * 5 * enigma(L,1) = 9 * 2 * 5 * 3 * enigma(L,0) = 9 * 2 * 5 * 3 * 8 = 2160$

A complexidade temporal do algoritmo enigma é dado por:

$\begin{cases} T(n) = T(n-1) + C \Rightarrow T(n) \in \Theta(n) \\ T(0) = C \end{cases}$

O algoritmo enigma devolve o resultado da multiplicação dos números contidos entre as posições 0 e n de V. Ele é eficiente.

Extra: é de cota inferior

Prova:

Teorema: o algoritmo enigma é correto

Prova: por indução em n. Base: n = 0, trivial. (H.I)

Suponha agora que enigma(L, n-1) resulta em $L[0] * L[1] * \dots * L[n-1]$. Observe que enigma(L, n) resulta em:

$L(n) * enigma(L, n-1) = L[0] * L[1] * \dots * L[n-1] * L[n]$ ■

S T Q Q S S D

__/__/__

07) Algoritmo: potencia
Ent.: a e b ∈ ℕ
Saída: a^b
Se a = 0 e b = 0
Devolva ERRO e pare
Se b = 0
Devolva 1 e pare
aux = potencia(a, $\lfloor b/2 \rfloor$)
Se b for par
Devolva aux * aux
Se não
Devolva aux * aux * a

A complexidade temporal e espacial do potencia é dada por:
$\begin{cases} T(a,b) = T(a, \lfloor b/2 \rfloor) + C \\ T(a,0) = C \end{cases} \Rightarrow T(a,b) \in \Theta(\log b)$

O algoritmo é eficiente!
-Prova:
Teorema: O algoritmo potencia é correto.
Prova: por indução em b. Base: b = 0, trivial.
Suponha agora que potencia(a, $\lfloor b/2 \rfloor$) resulta em $a^{\lfloor b/2 \rfloor}$ (H.I.)
Se b for par e > 0, potencia(a,b) resulta em:
aux * aux = potencia(a, $\lfloor b/2 \rfloor$) * potencia(a, $\lfloor b/2 \rfloor$) = $a^{\lfloor b/2 \rfloor} * a^{\lfloor b/2 \rfloor} = a^{b/2} * a^{b/2} = a^b$
Se b for impar, potencia(a,b) resulta em:
aux * aux * a = potencia(a, $\lfloor b/2 \rfloor$) * potencia(a, $\lfloor b/2 \rfloor$) * a = $a^{\lfloor b/2 \rfloor} * a^{\lfloor b/2 \rfloor} * a =$
$a^{\frac{b-1}{2}} * a^{\frac{b-1}{2}} * a = a^b$ ∎

__/__/__

S T Q Q S S D

08) O algoritmo de Fleinholzer requer tempo $O(m)$, onde m é a quantidade de arestas. Como o tamanho da saída é $\Theta(m)$, concluímos que esse algoritmo é de cota superior, logo também é de cota superior.

09) Algoritmo: somaMatriz
Entrada: uma matriz de números A, com m linhas e n colunas
Saída: a soma dos números contidos em A

```
soma = 0
Para i = 0 até m-1
    Para j = 0 até n-1
        soma = soma + A[i][j]
Devolva soma
```

O algoritmo somaMatriz requer tempo $\Theta(m \cdot n)$ e espaço $O(1)$. Para somar os elementos de uma matriz, é preciso acessar as suas posições, portanto será necessário gastar tempo $\Omega(m \cdot n)$. Como o algoritmo requer tempo $O(m \cdot n)$, concluímos que é de cota inferior.

10)
Método contábil: vamos atribuir custo amortizado 3 para a operação de empilhamento em P_1 (gera crédito), custo amortizado 0 para as operações de desempilhamento em P_1 e empilhamento em P_2 (gram débito) e custo amortizado 1 para o desempilhamento em P_2. Com tais custos amortizados, podemos garantir que o saldo nunca fica negativo, logo a soma dos custos amortizados é maior ou igual à soma dos custos reais. O custo amortizado do removeDaFila é 1, logo a soma dos custos reais é $O(n)$

Método da agregação
custos dos empilham em $P_1 \leq n$
" " desempilham em $P_1 \leq n$
" " empilham em $P_2 \leq n$
" " desempilham, " " $\leq n$

S T Q Q S S D

__/__/__

- Método potencial

Vamos associar P_1 ao removeDaFila e definir $\Phi(D_i)$ como sendo igual ao dobro dos elementos contidos em P_1. Note que $\Phi(D_0) = 0$ e $\Phi(D_n) \geq 0$, logo $\Phi(D_n) \geq \Phi(D_0)$ e portanto a soma dos custos amortizados é maior ou igual à soma dos custos reais

O custo amortizado da operação i é:

- Se a operação é um insere na fila

$$\hat{C}_i = 1 + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{2} = 3$$

- Se a operação é um removeDaFila e P_2 não estava vazia

$$\hat{C}_i = 1 + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{0} = 1$$

- Se a operação é um removeDaFila e P_2 estava vazia, sendo K a quantidade de elementos em P_1

$$\hat{C}_i = 2K + 1 + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{-2K} = 1$$

$\Phi(D_i)$: 0; $\Phi(D_{i-1})$: $2K$