Nome: Patricia da Silva Rodrigues

Nota da Prova 1: 5.2

Q1 (valor 2.5): nota 1.5

- (penalidade: -1.0, etiqueta 1) Análise conclui consumo de tempo super-linear
  Note que k não é uma constante, e sim $k = \Theta(\lg n)$. Logo, $O(kn) = O(n \lg n)$.

Q2 (valor 2.5): nota 0.0

- (penalidade: -2.5, etiqueta 1) Algoritmo incompleto

Q3 (valor 2.5): nota 2.2

- (penalidade: -0.3, etiqueta 1) Off by one

Q4 (valor 2.5): nota 1.5

- (penalidade: -1.0, etiqueta 1) Algoritmo incorreto

MAC0338 Análise de Algoritmos: Prova I

NOME COMPLETO: Patricia S. Rodrigues

NUSP: 11315590

ASSINATURA: Patricia S.

ATENÇÃO

Siga as instruções abaixo **RIGOROSAMENTE**. Qualquer desvio acarretará na penalização em 1,0 ponto.

1. Escreva suas iniciais no canto inferior direito de **TODAS** as folhas de prova.
2. Mantenha suas respostas **dentro das bordas** de cada espaço reservado para elas. (Sua prova será escaneada; conteúdos fora das bordas serão perdidos.)

Iniciais: P. S. R

Exercício 1. Considere a sequência de vetores $A_k[1\,..\,2^k], A_{k-1}[1\,..\,2^{k-1}], \ldots, A_1[1\,..\,2^1]$, e $A_0[1\,..\,2^0]$. Suponha que cada um dos vetores é crescente. Queremos reunir, por meio de sucessivas operações de intercalação (= *merge*), o conteúdo dos vetores $A_0, \ldots, A_k$ em um único vetor crescente $B[1\,..\,n]$, onde $n = 2^{k+1} - 1$. Escreva um algoritmo que faça isso em $O(n)$ unidades de tempo. Use como subrotina o INTERCALA visto em aula; lembre-se de que o protótipo dessa subrotina é INTERCALA(C, p, q, r), que assume que $C[p\,..\,q]$ e $C[q+1\,..\,r]$ estão ordenados. [2,5 pontos]

Você deve escrever o pseudocódigo do seu algoritmo, descrever seu funcionamento, argumentar corretude e fazer uma análise formal do consumo de tempo.

RESPOSTA

UNIR (C, K)

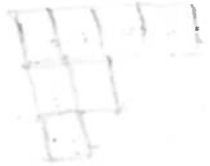
1. SE $2^K <$ TAM(C) FAÇA:
2. $q = 2^K$
3. $p = 2^{K-1}$
4. INTERCALA$(C, n-2^{K-1}, n-2^K, n)$
5. UNI (C, K+1)
6. SENÃO
7. DEVOLVA C

MAIN (VETORES)

1. C = AGLUTINA (VETORES)
2. UNIR (C, 0)

se p

2^3



se $2^{K-1} <$ tam(C)

Iniciais: P. S. R.

RESPOSTA

em main estou usando a função seguinte que junta os vetores um atrás do outro em um único vetor maior, começando $\underbrace{A \cdot 2^k}_{A_k}, \underbrace{A \cdot 2^{k-1}}_{A_{k-1}}, \ldots$ e assim por diante
ela deve consumir $O(n)$

explicação

a função unir deverá fazer a intercalação começando pelo menor A_k até o maior A_k (correremos de trás para frente no vetor C)
q = diminuirá na proporção $n-2^k$ (de trás para frente no vetor C)
p será igualmente decrescente, mas começando no $n-2^{k-1}$ e r será sempre n

Na recursão, iremos para o próximo k e o próximo q será o antigo $P(2^{k-1})$, o próximo p será o início do próximo vetor de trás para frente (2^{k-2}) e r será n. Desta maneira, sempre teremos o vetor intercalado anteriormente para intercalar com o próximo vetor.

MAIN:
1. $O(1)$
2. $O(n)$ } $O(n)$

UNIR:
1. $O(1)$
2. $O(1)$
3. $O(1)$
4. $O(n)$ } $O(n)$

chamada recursiva

6. $O(1)$ e 7. $O(1)$

1

5. temos $2^k - 1$ elementos. Se
$n = 2^k - 1 \Rightarrow \log n = \log_2(2^k) \Rightarrow O(\log n)$
(chama a linha 5 no máximo k vezes
$5 = \log n$

$\therefore$ MAIN e UNIR
$O(n) + O\lg(n) + O(n) + O(1) = O(n)$
$\Rightarrow \boxed{O(n)}$

Iniciais: P. S. R.

Exercício 2. Escreva um algoritmo aleatorizado que, dado um vetor $A[1\,..\,n]$ contendo n números inteiros distintos e um inteiro k tal que $1 \leq k \leq n$, determine os k números de A mais próximos da mediana de A. O consumo de tempo **esperado** de seu algoritmo deve ser $O(n)$. Você pode supor que n é ímpar, de modo que $A[1\,..\,n]$ possui uma única mediana. [2,5 pontos]

Você deve escrever o pseudocódigo do seu algoritmo, descrever seu funcionamento, argumentar corretude e fazer uma análise formal do consumo de tempo.

RESPOSTA

1

CALCULAPROXmed (A, k)
PARA i de 1 até n FAÇA:
ALeA _ PARTICIONE(A)
se verifique(A) == 1:
resposta = $\lceil \frac{n}{2} \rceil - \frac{k}{2} \ldots, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{n}{2} \rceil + 1, \ldots \lceil \frac{n}{2} \rceil + \frac{k}{2}$

ALeA_PARTICIONE (A)
PIVO = RAND(1, TAM(A))
devolva PARTICIONE(A, PIVO)

Não consegui pensar em uma resposta melhor que $O(n \lg n)$

 Iniciais: P.S.R

RESPOSTA

Iniciais: P. S. R

Exercício 3. Descreva dois algoritmos, Pré-Processa e Consulta-Intervalo, com os seguintes comportamentos; a ideia é que Pré-Processa é chamado uma única vez, seguido de múltiplas/numerosas chamadas a Consulta-Intervalo. [2,5 pontos]

Uma chamada Pré-Processa(A, n) recebe um vetor $A[1\,..\,n]$ de n inteiros no intervalo de 1 a k, inclusive, e devolve algum objeto, que vamos chamar de X. Tal chamada deve consumir tempo $O(n + k)$.

Uma chamada Consulta-Intervalo(X, a, b) utiliza o objeto X devolvido pela única chamada a Pré-Processa, além de dois inteiros tais que $1 \leq a \leq b \leq k$, e devolve o número de inteiros em $A[1\,..\,n]$ que estão no intervalo $[a\,..\,b]$. Cada chamada a Consulta-Intervalo deve consumir tempo $O(1)$.

Você deve escrever os pseudocódigos dos seus algoritmos, descrever seus funcionamentos, argumentar corretude e fazer uma análise formal dos consumos de tempo.

RESPOSTA

A = [1|2|3|4|6]

C[1..k]

k = 6

```
PRE_PROCESSA (A, n, k)
    C = [ ];
    PARA i de 1 até k FAÇA:
        C[i] = 0
    PARA i de 1 até n FAÇA:
        C[A[i]] = C[A[i]] + 1
    PARA i de 2 até k FAÇA:
        C[i] = C[i] + C[i-1]
    Devolva C
```

→ C = [0|0|0|0|0|0] (1 2 3 4 5 6)

→ C = [1|1|1|1|0|1]

```
CONSULTA_INTERVALO (X, a, b)
    Se a = 1
        Devolva X[b]
    SENÃO
        Devolva X[b] - X[a]
```

1

FUNCIONAMENTO: O algoritmo pré-processa no primeiro loop inicializa C com zeros, no segundo calcula a frequência de cada número que aparece em A e armazena a frequência de i em C[i]. O terceiro loop faz a soma acumulada dessas frequências. Trata-se de uma adaptação do algoritmo de ordenação em tempo linear Counting sort. O algoritmo consulta_intervalo recebe em X esse array de

 Iniciais: P. S. R.

RESPOSTA

frequência acumulada devolvida pelo pre-processa. Como em X temos a frequência acumulada, apenas calculamos a diferença no intervalo [a,b] e retornamos o valor.

Análise consumo de tempo: pre-processa

1-2: $\Theta(k)$
3-4: $\Theta(n)$
5-6: $\Theta(k)$

Total $2k + n = O(k+n)$

Análise consumo de tempo: consulta_intervalo

1. $\Theta(1)$
2. $\Theta(1)$
3. $\Theta(1)$
4. $\Theta(1)$

$O(1)$ tempo constante

argumentação

Como no 3º loop a soma é acumulativa
a crescente da frequência, conseguimos
garantir que $X[b]-X[a]$ retornará com
certeza o número de inteiros ~~em~~ no
intervalo [a,b].

Iniciais: P. S. R

Exercício 4. Considere o algoritmo de programação dinâmica visto em aula para parentização ótima de um produto $A_1 A_2 \cdots A_n$ de n matrices, reproduzido abaixo: [2,5 pontos]

```
MATRIX-CHAIN-ORDER(p, n)
1.  para i ← 1 até n faça
2.     m[i, i] ← 0
3.  para ℓ ← 2 até n faça
4.     para i ← 1 até n − ℓ + 1 faça
5.        j ← i + ℓ − 1
6.        m[i, j] ← ∞
7.        para k ← i até j − 1 faça
8.           q ← m[i, k] + p[i − 1]p[k]p[j] + m[k + 1, j]
9.           se q < m[i, j]
10.             então m[i, j] ← q
11. return m[1, n]
```

Descreva como modificar o pseudocódigo acima para que ele devolva um objeto X a mais, que será usado na construção de uma parentização ótima para o produto $A_1 A_2 \cdots A_n$. (Não é necessário copiar o código inteiro; por exemplo, você poderia escrever: "entre as linhas 3 e 4, insira uma linha 3,5 com o comando 'blá'".)

Escreva também uma função IMPRIME-PARENTIZAÇÃO(X, n), que recebe o objeto X devolvido a mais pela sua modificação, bem como o número de matrizes no produto, e imprime uma parentização ótima. Naturalmente, esse algoritmo não deve refazer os cálculos já realizados pela função MATRIX-CHAIN-ORDER. Além disso, não imprima parênteses em torno de uma única matriz. Por exemplo, uma impressão válida para $n = 3$ pode ser $((A_1 A_2) A_3)$.

Você deve escrever o pseudocódigo desse algoritmo, descrever seu funcionamento, argumentar corretude e fazer uma análise formal do consumo de tempo.

RESPOSTA

modificação proposta no Matrix-Chain-Order:

1º antes da linha 1 K de dimensões n x n (mesma dimensão que m.

2º entre as linhas 10 e 11, ainda dentro da condição da linha 9, além de armazenar o mínimo em m[i,j], armazene também:

mk[i,j] = k

Ao armazenarmos qual foi o k que retornou o mínimo para m[i,j], conseguiremos depois retornar a esse valor para imprimir uma parentalização ótima. Usarei uma árvore recursiva para construir a parentalização

→

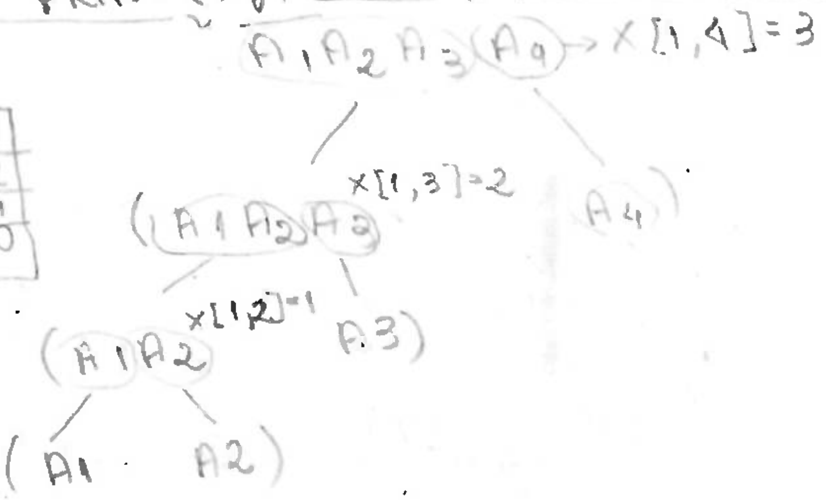
 Iniciais: P. S. R

RESPOSTA

```
IMPRIME_PARENTALIZAÇÃO (X, n)
    i = 1
    j = tamanho(X)
    Se i < j FAÇA:
        K = X[i, j] e PRINT ("(")
        IMPRIME_PARENTALIZAÇÃO (X[i..K], n)
        IMPRIME_PARENTALIZAÇÃO (X[K+1..j], n)
        PRINT (")")
    Se X[i, j] = 0 FAÇA:
        PRINT(A_i)
```

Simulação

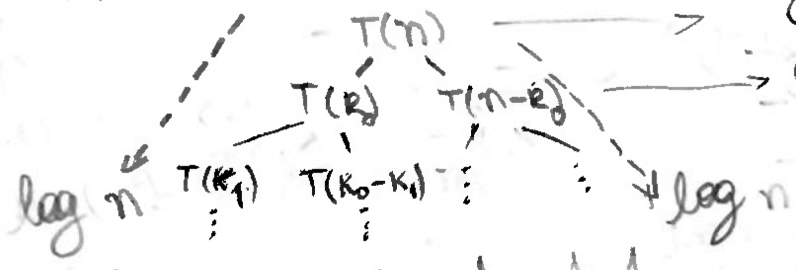
$$X = \begin{bmatrix} 0 & 1 & 2 & 3 \\ & 0 & 1 & 2 \\ & & 0 & 1 \\ & & & 0 \end{bmatrix}$$



$\boxed{(((A1\ A2)\ A3)\ A4)}$

Consumo de tempo: imprime parentalização:

$T(n) = T(K) + T(n-K) + O(1)$



$\to C.1 = 1C$

$\to C.1 + C.1 = 2C$

$\vdots$

$= nC$

$\Omega(\log n) \longrightarrow O(\log n) = \boxed{\Theta(\lg n)}$

definição Θ: c_1 e $c_2 > 0$ tal que, $\exists n \geq n_0 \Rightarrow$

$\Rightarrow c_1 g(n) \leq f(n)$ e $f(n) \leq c_2 g(n)$.

RESPOSTA

argumentação

como a matriz mk registra em i,j qual foi o k que deu a menor quantidade de multiplicações necessárias até o momento, sabemos que se colocarmos parenteses naquele k, teremos uma multiplicação ótima.

O algoritmo que implementei começa em 1,n, porque assim estamos considerando todo conjunto de matrizes

Iniciais: P. S. R