

Provas
Antigas
II

Sophia Carrazza

Dois algoritmos A e B, que resolvem o mesmo problema, são analisados e é encontrado que B possui custo computacional, em termos de tempo e da notação $O(n)$, menor que o algoritmo A. A e B são então implementados (cuidadosamente e corretamente) como programas, em C, e testados em vários conjuntos de dados.

Analise as assertivas a seguir, assinalando V, se verdadeiras, ou F, se falsas.

9A J8

- F O programa A executará sempre mais rápido que o programa B. *→ depende!*
- F O programa B executará sempre mais rápido que o programa A. *→ nem sempre*
- F A medida que o tamanho da entrada cresça, as chances aumentam do programa A ser mais rápido que o programa B. *X → o contrário*
- V A medida que o tamanho da entrada cresça, as chances aumentam do programa B ser mais rápido que o programa A. *✓ → B tem menor complexidade análoga*
- F Ambos os programas sempre executarão com o mesmo tempo. *X → complexidades diferentes*
- V Não é possível fazer afirmação com relação ao tempo de execução pois eles dependerão da estruturas de dados e da memória disponível para a execução dos algoritmos.

A ordem correta, de cima para baixo, das respostas destas assertivas é:

- (a) V - F - V - F - F - V *X*
- (b) F - V - F - V - V - F *X*
- (c) E - F - V - F - F - V
- (d) F - F - F - V - F - V *X*
- (e) V - V - V - F - V - F *X*

que bom que você
não pediu pra
justificar ☺

se de $\epsilon O(n)$, + b põem complexidade $O(n^2)$, pra mim a notação big-O estabelece um limite superior. Logo, qualquer algoritmo que cresce linearmente ($O(n)$) sempre crescerá + lentamente que n^2 .

5

QUESTÃO 4 (15 %)

Analise as assertivas a seguir, assinalando V se a assertiva for verdadeira, ou F, se a assertiva for falsa.

- F Algoritmos gulosos têm como principal característica a construção incremental da solução, por meio da otimização de algum critério local, e por causa disto, não é possível obter soluções ótimas. *→ os voys podemos obter soluções ótimas norm.*
- F Algoritmos gulosos têm como principal característica a construção incremental da solução, por meio da otimização de algum critério global, sempre obtendo soluções ótimas. *→ não tanto não*
- F Algoritmos com ordem de complexidade $O(n)$ também possuem ordem de complexidade $O(n^2)$. *✓ → erros*
- F Algoritmos com ordem de complexidade $O(n)$ são sempre mais rápidos que algoritmos com ordem de complexidade $O(n^2)$ para o mesmo tamanho de instância. *X → depende de muitos outros fatores*
- V Encontrar elementos em um vetor, de forma exata, pode ser feita em $O(n)$. *✓ → precedentes os elementos do vetor*
- V É possível fazer uma busca binária, desenvolvida por meio de recursão, e uma busca binária, desenvolvida por meio de uma estrutura de repetição, terem a mesma ordem de complexidade. *→ ambos são $O(\log n)$*

Qual será a ordem correta, de cima para baixo, das respostas destas assertivas? Justifique todas as suas respostas. Caso não tenha justificativa, o item será zerado.

- () V - V - V - F - F - V
- () F - F - V - F - F - V
- () F - F - F - V - F - V
- (X) F - F - V - F - V - V *0*
- (X) F - F - V - F - V - F *0*

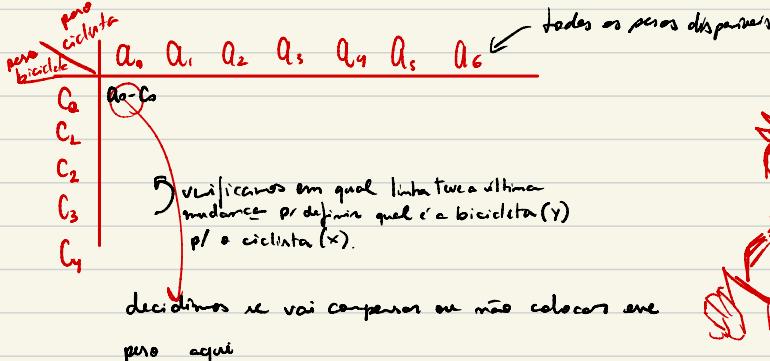
QUESTÃO 3

(30 %)

26

Seja uma competição de ciclismo de velocidade por equipes, em que todos os membros estão divididos em m equipes. Um dos desafios das equipes é fazer um percurso o mais rapidamente possível. Você é o técnico de uma das equipes, e que o seu desafio é escolher adequadamente as bicicletas de cada membro da equipe. Para garantir a velocidade média máxima de cada ciclista sabe-se que quanto mais próximos os pesos da bicicleta e do ciclista, maior a velocidade que pode ser atingida pela bicicleta, e consequentemente, maior a velocidade média do par ciclista/bicicleta. Considere que a sua equipe possui n membros, com pesos a_1, a_2, \dots, a_n , e que existam n bicicletas com pesos c_1, c_2, \dots, c_n .

Problema: o seu objetivo é escrever um algoritmo, com $O(n^2)$ para atribuir uma bicicleta a um ciclista de forma a tentar minimizar o tempo de percurso. Seu algoritmo é ótimo? Dê argumentos que mostre sua otimalidade ou um contra-exemplo que mostre que não é ótimo. Justifique suas escolhas e deixe claro como chegou ao algoritmo com o custo desejado.



$$\min \{ \text{dif}(a_0, c_{i-1}), \text{dif}(a_0, c_i) \}$$

preenchimento da tabela \downarrow verificação da tabela

$$\text{complexidade} = O(n^2 + n^2) = O(n^2)$$

é ótimo? = não, não é ótimo, pois não está considerando todas as possíveis combinações, tendo um nº limite de bicicletas

↳ Solução ótima: (que não é $O(n^2)$) Algoritmo Guloso
uma solução ótima seria ordenar ambas as listas de pesos de bicicletas e de ciclistas, e fazer um pairing entre cada posição com tupla ($[c[0], b[0]]$). Daí forma, todos os ciclistas terão suas diferenças minimizadas e, consequentemente manterão o desempenho dos ciclistas.

ou seja, o maior mº de caracteres que aparecem nos dois strings na mesma ordem (não necessariamente consecutivas).

QUESTÃO 1

Dado duas strings $x = x_1 x_2 \dots x_n$ e $y = y_1 y_2 \dots y_m$. Desejamos encontrar o comprimento da maior subsequência comum com custo computacional $O(mn)$ de forma que $x_{i_1} x_{i_2} \dots x_{i_k} = y_{j_1} y_{j_2} \dots y_{j_k}$, em que $i_1 < i_2 < \dots < i_k$ e $j_1 < j_2 < \dots < j_k$. Justifique todas as suas opções e escolhas, assim como, descreva clara sua modelagem e os custos computacionais.

quebrar roteiro
capimento (não os
letras)

Problema da Maior Subsequência Comum (LCS) \Rightarrow programação dinâmica!
criaremos uma tabela (matriz) \rightarrow longest common sequence

X	x_1	x_2	x_3	x_4
y_1	0	0	0	0
y_2	0			
y_3	0			
y_4	0			

② \rightarrow resultado

- se $i = 0$ ou $j = 0$, reencontram com 0. (se $i=0$ ou $j=0$, $c[i][j]=0$)
- se as letras coincidirem, somarmos + ao valor da célula diagonalmente anterior na posição de análise (se $x[i] = y[j]$, $c[i][j] = c[i-1][j-1] + 1$)
- se não coincidirem, colocarmos o maior valor entre a célula de cima e a célula da esquerda. (se $x[i] \neq y[j]$, $c[i][j] = \max(c[i-1][j], c[i][j-1])$)

exemplo \Rightarrow "ABCB" e "BDCAB"

complexidade $\in \Theta(n \cdot m)$

	-	B	D	C	A	B
-	0	0	0	0	0	0
A	0	0	0	0	1	1
B	0	1	1	1	1	2
C	0	1	1	2	2	2
B	0	1	1	2	2	3

o resultado é o valor no canto inferior direito da matriz

QUESTÃO 2

(100 %)

Uma subsequência é "palindrómica" se ela é lida da mesma forma da esquerda para a direita e da direita para a esquerda. Por exemplo, a sequência A, C, G, T, G, T, C, A, A, A, T, C, G possui muitas subsequências palindrómicas, dentre elas, A, C, G, C, A e A, A, A, A. No entanto, A, C, T não é palindrómica. Projete um algoritmo, que execute em $O(n^2)$ tenha como entrada uma sequência com n caracteres, e que retorne o tamanho da maior subsequência palindrómica. Justifique todas as suas opções e escolhas, assim como, deixe claro sua modelagem e os custos computacionais.

menor sequência 8 carateres

-	A	C	G	T	G	T	C	A	A	A	A	T	C	G
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	1	1	1	1	1	1	1	1	1	1	1
C	0	0	1	1	1	1	2	2	2	2	2	2	2	2
T	0	0	1	1	2	2	2	2	2	2	2	2	2	3
A	0													
A	0													
C	0													
T	0													
G	0													
C	0													
A	0													

mesma modelagem
anterior, mas considerando a string original e sua reversa.

maior subsequência palindrómica:
8 //

8

complexidade: custo de montagem / inserção da matriz (que é $n \cdot n \Rightarrow O(n^2)$).

QUESTÃO 3

(100 %)

Projetar um algoritmo para encontrar o maior valor, que pode ser obtido por uma colocação adequada de parênteses na expressão $x_1/x_2/x_3/\dots/x_{n-1}/x_n$, em que x_i é um número positivo. Justifique todas as suas opções e escolhas, assim como, deixe claro sua modelagem e os custos computacionais.

$$(8/2)/2 = 4/2 = 2$$

$$\rightarrow 8/(2/2) = 8/1 = 8 \quad \text{↑ resultado} \Rightarrow \text{↑ dividendo} \downarrow \text{divisor}$$

$$\downarrow$$

$$3/(8/5/2/2/7) \rightarrow 6$$

$$3/(1/5/2/2/7) = 428$$

$$3/1/(5/2/2/7) = 17,6$$

$m[i][j] = x_i$ se não existe x_L .

$$m[i][j] = \max \left\{ m[i][k] / m[k+1][j], m[i][k-1] \cdot (x_k / m[k+1][j]) \right\}$$

\downarrow

$K = \text{posição onde colocamos a última parêntese na expressão}$

- tempo de fim (ótimo)
 - tempo de início
 - tamanho
 - nº de intersecções
- } existe contracarro
- } algoritmos p/ encontrar a maior qntd de intervalos disjuntos
- ↓ p/ no e sem peso!

QUESTÃO 1

(20 %)

Uma sequência composta por elementos diferentes é considerada bitônica se ela possui um elemento que divide em duas subsequências sendo que a primeira parte seja uma subsequência decrescente e a segunda parte uma subsequência crescente. Projete um algoritmo em $O(\log n)$ para encontrar o índice e fim de cada uma subsequência.

- algoritmo de divisão e conquista → busca binária
 - passo a passo:
 - 1º calcularemos o elemento do meio do vetor/sequência.
 - 2º verificaremos se o elemento da direita é maior que ele e se da esquerda também é maior que ele. Se sim, enc坍hamos o ponto de inflexão. Se não, verificaremos:
 - se o elemento de meio é maior que o próximo, estamos na parte decrescente, e recalcularemos o meio com a esquerda tendo o meio atual + 1 (e repetimos o restante do código).
 - se o elemento de meio é menor que o próximo, estamos na parte crescente, e recalcularemos o meio com a direita tendo o meio atual - 1 (e repetimos o restante do código).
 - 3º quando enc坍hamos o ponto de inflexão, o ponto de inflexão é inicio e fim de cada subseqüência são o 1º elemento do vetor até meio - 1 e o meio + 1 até o último elemento do vetor
- até encontrar
- * casos base → se temos só um elemento no vetor, retornamos ele
 - 4º se o 1º elemento é menor que o 2º → retorna erro?
 - 5º se o último elemento é menor que o penultimo → erro

Seja uma competição de esqui por equipes, em que todos os membros estão divididos em m equipes. Um dos desafios das equipes é descer uma montanha o mais rapidamente possível. Você é o técnico de uma das equipes, e que o seu desafio é escolher adequadamente os esquis de cada membro da equipe. **Para garantir a velocidade máxima de cada esquiador** sabese que os esquiadores vão mais rápido quando a sua altura é parecida com o comprimento do esqui utilizado. Considere que a sua equipe possui n membros, com alturas a_1, a_2, \dots, a_n , e que existam n pares de esquis com comprimentos c_1, c_2, \dots, c_n . Portanto, o seu objetivo é escrever um algoritmo, em $O(n^2)$, para atribuir um par de esqui a cada esquiador de forma a tentar minimizar o tempo de descida. Seu algoritmo é ótimo? Dê argumentos que mostre sua otimalidade ou um contra-exemplo que mostre que não é ótimo. Justifique suas escolhas e dize claro como chegou ao algoritmo com o custo desejado.

- ↳ 1^a opção = ordenação dos dois vetores com bubble sort e fazendo um pairing entre cada posição dos dois vetores
 repetimos isso diversas vezes até não ter o que trocar e o vetor estiver ordenado. $O(n^2)$
- ↳ 2^a opção = ordenações do 1º vetor e tentarmos os resultados desse vetor ordenado com todos
- Compararemos cada posição c/ sua proxima. Se o de posição é maior que o próximo, trocamos os valores de posição e compararemos o trocado c/ prox.*

Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list, comparing adjacent elements and swapping them if they are in the wrong order. It continues this process until no swaps are needed, indicating that the list is sorted.

Here's a more detailed explanation:

- Comparison and Swapping:**
In each pass, the algorithm compares adjacent elements in the list. If the current element is greater than the next element (for ascending order), it swaps their positions.
- Passes:**
This process is repeated multiple times, with each pass moving the largest unsorted element to its correct position at the end of the list.
- Termination:**
The algorithm terminates when it completes a pass without making any swaps, indicating that the list is sorted.
- Visualization:**
Programmatic provides a visual explanation of how bubble sort works, and Runestone Academy describes how each element "bubbles" up to its correct position.
- Time Complexity:**
Bubble sort has a time complexity of $O(n^2)$ in the worst and average cases, where n is the number of elements in the list.
- Space Complexity:**
Bubble sort has a space complexity of $O(1)$ because it only requires a constant amount of extra space for swapping.

Bubble Sort					
	6	2	8	4	10
First pass	6	2	8	4	10
Next pass	2	6	8	4	10
Next pass	2	6	4	8	10
	2	4	6	8	10

Review complete

QUESTÃO 3

(30 %)

Seja $G = (V, E)$ um grafo acíclico direcionado, e seja k o número máximo de arestas em um caminho de G . É possível projetar um algoritmo em tempo linear em relação ao número de vértices e arestas que divide os vértices em, no máximo, $k + 1$ grupos tal que para dois vértices do mesmo grupo não possam caminho de um para o outro? Se for possível, projete este algoritmo em tempo linear, caso contrário, projete em $O(n^2)$. *Justifique todas as suas opções e escolhas. Deixe claros tanto sua modelagem e quanto os custos computacionais do algoritmo projetado, mostrando como calcular este custo.*

QUESTÃO 4

(25 %)

Discorra sobre a estrutura geral de algoritmos gulosos, mostrando, em especial, elementos que ilustrem custos computacionais desta estrutura.

10 14 15 12 11 / 12 3 4 3