Construção e Análise de Algoritmos

4a avaliação remota

Instruções:

Existem basicamente 3 maneiras de organizar um algoritmo de programação dinâmica

I. Realizar quebras sucessivas em pontos intermediários do problema.

II. Reduzir sucessivamente o tamanho do problema a partir de uma extremidade.

III. Organizar a execução do algoritmo em termos do preenchimento de uma tabela.

Na maior parte dos casos, qualquer uma das 3 estratégias pode ser utilizada para se chegar a um algoritmo de programação dinâmica.

As duas questões que você verá a seguir ilustram essa ideia.

Na primeira questão, você será guiada(o) durante o processo de construção do algoritmo, e a sua tarefa será apenas escrever o pseudo-código e analisar a sua complexidade.

Na segunda questão, você terá que realizar a tarefa inteira sozinha(o).

1. (4,0 pontos) **Subsequência comum com soma máxima**

   Lembre que na lista 22 nós vimos o problema da subsequência comum mais longa.

   Quer dizer, dadas duas sequências de números (positivos e negativos)

   $$A = (a_1,\ a_2,\ a_3,\ \ldots,\ a_n) \qquad\qquad B = (b_1,\ b_2,\ b_3,\ \ldots,\ b_m)$$

   o problema consiste em encontrar índices $i_1, i_2, \ldots, i_k$ para a sequência A, e $j_1, j_2, \ldots, j_k$ para a sequência B, tais que

   $$a_{i_1} = b_{j_1} \qquad a_{i_2} = b_{j_2} \qquad \ldots \qquad a_{i_k} = b_{j_k}$$

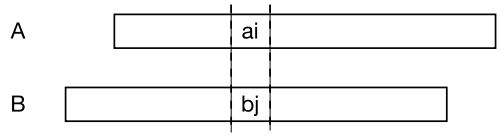
   e o objetivo é que essa subsequência comum de A e B seja o mais longa possível.

   Nessa questão, nós vamos considerar a variante do problema que consiste em *maximizar a soma dos elementos da subsequência*, e não o seu comprimento.

   A seguir, nós vamos desenvolver algoritmos de programação dinâmica para esse problema, utilizando cada uma das estratégias acima.

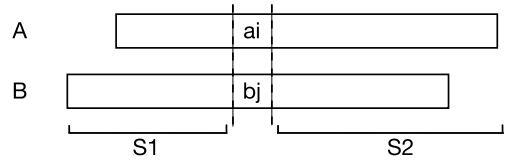
   a) *Estratégia I*

      A ideia aqui é visualizar o que acontece quando nós quebramos o problema em um ponto intermediário, selecionando elementos a_i e b_j para fazer parte da solução



Mas, uma vez que nós temos a figura, as coisas ficam mais fáceis:

- basta encontrar a subsequência comum c/ soma máxima no lado esquerdo de A e B
- encontrar a subsequência comum c/ soma máxima no lado direito de A e B



- e fazer a concatenação

$$S \;=\; S_1 \;+\; a_i \;+\; S_2$$

(aqui tanto faz usar a_i ou b_j pois eles devem ser iguais)

Pronto, aqui nós já temos a estratégia de *quebra* e *combinação*.

A última coisa que falta é o *truque da programação dinâmica*:

- Repetir esse procedimento para todo par de elementos iguais a_i e b_j em A e B

a1) Escreva o pseudo-código do algoritmo de programação dinâmica que implementa essa ideia.

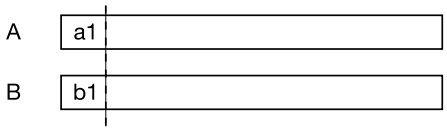
(*Importante:* não esqueça de implementar a memoização.)

a2) Estime a complexidade do seu algoritmo.

(*Importante:* não esqueça de considerar os procedimentos de quebra e combinação.)

b) *Estratégia II*

A ideia aqui é visualizar como o problema pode ser reduzido a um problema um pouco menor, examinando os elementos no início das duas sequências



As duas opções que se apresentam aqui são

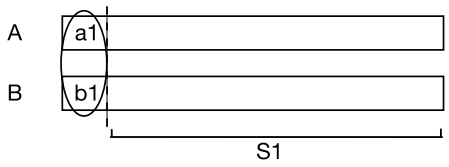
- o par (a_1, b_1) faz parte da solução (o que só pode ser o caso se $a_1 = b_1$)
- o par (a_1, b_1) não faz parte da solução

O segundo caso, no entanto, se subdivide em mais 3 possibilidades

- nem a_1 nem b_1 fazem parte da solução
- a_1 pode fazer parte da solução, associado a um outro elemento de B
- b_1 pode fazer parte da solução, associado a um outro elemento de A

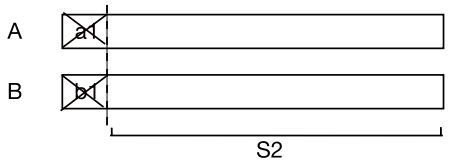
Agora, para cada um desses 4 casos, nós obtemos um subproblema diferente para resolver em seguida

caso 1:



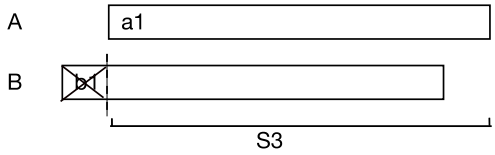
Encontrar a subsequência comum com soma máxima no restante de A e B, e concatená-la ao par (a_1, b_1).

caso 2:



Encontrar a subsequência comum com soma máxima no restante de A e B.

caso 3:



Encontrar a subsequência comum com soma máxima em A e no restante de B.

caso 4: (análogo ao anterior)

Pronto, aqui nós já temos a estratégia de *redução* do problema.

E a ideia, é claro, é escolher a melhor solução obtida a partir desses 4 casos (esse é o *truque da programação dinâmica*).

b1) Escreva o pseudo-código do algoritmo de programação dinâmica que implementa essa ideia.

(*Importante:* não esqueça de implementar a memoização.)

b2) Estime a complexidade do seu algoritmo.

(*Importante:* não esqueça de considerar os procedimentos de quebra e combinação.)

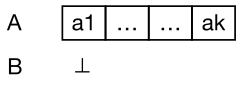
c) *Estratégia III*

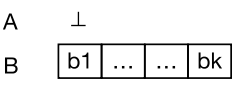
As duas estratégias anteriores colocam ênfase no processo de decomposição (quebra ou redução) do problema.

Já a estratégia III coloca ênfase no processo de composição (combinação) de soluções.

Assim, ao invés de começar visualizando um problema grande, que deve ser decomposto, nós começamos examinando um problema trivial.

Um problema trivial é aquele onde uma das sequências é vazia

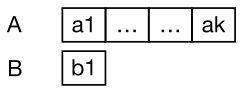


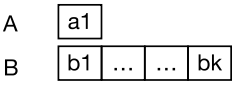


E um problema trivial, claro, possui uma solução trivial:

- a subsequência vazia (que possui soma 0)

O próximo passo, consiste em examinar problemas ligeiramente maiores



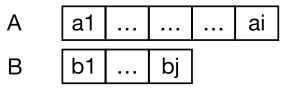


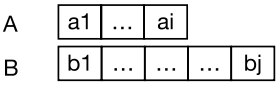
E aqui nós temos 4 casos para analisar (no exemplo da esquerda):

- o caso em que o par (a_k, b_1) faz parte da solução (caso eles sejam iguais)
- o caso em que a_k não faz parte da solução, mas b_1 pode fazer
- o caso em que b_1 não faz parte da solução, mas a_k pode fazer
- o caso em que nem a_k nem b_1 fazem parte da solução

A observação chave aqui é que em todos os casos a solução é obtida a partir da solução para um problema menor ou um problema trivial.

E para problemas maiores,





a mesma análise em 4 casos se repete.

A ideia então é utilizar uma tabela $\mathtt{T}[\mathtt{i},\mathtt{j}]$ para armazenar as soluções dos subproblemas, e preencher cada posição utilizando os valores que estão nas posições anteriores.
Os valores iniciais da tabela correspondem aos problemas triviais

$$\begin{aligned} \mathtt{T}[\mathtt{k},0] &= 0 \qquad \text{para } k = 0,\ldots,n \\ \mathtt{T}[0,\mathtt{k}] &= 0 \qquad \text{para } k = 0,\ldots,m \end{aligned}$$

E cada posição $\mathtt{T}[\mathtt{i},\mathtt{j}]$ tem o seu valor calculado da seguinte maneira

- Se $a_i = b_j$

$$\mathtt{T}[\mathtt{i},\mathtt{j}] = \mathsf{Max}\Big\{(\mathtt{T}[\mathtt{i}-1,\mathtt{j}-1]+a_i),\ \mathtt{T}[\mathtt{i},\mathtt{j}-1],\ \mathtt{T}[\mathtt{i}-1,\mathtt{j}],\ \mathtt{T}[\mathtt{i}-1,\mathtt{j}-1]\Big\}$$

- Se $a_i \neq b_j$

$$\mathtt{T}[\mathtt{i},\mathtt{j}] = \mathsf{Max}\Big\{\mathtt{T}[\mathtt{i},\mathtt{j}-1],\ \mathtt{T}[\mathtt{i}-1,\mathtt{j}],\ \mathtt{T}[\mathtt{i}-1,\mathtt{j}-1]\Big\}$$

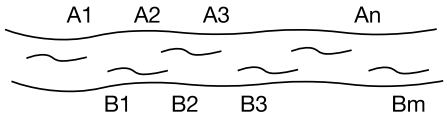
Ao final do processo, a solução do problema se encontra na posição $\mathtt{T}[\mathtt{n},\mathtt{m}]$.

c1) Escreva o pseudo-código do algoritmo de programação dinâmica que implementa essa ideia.

c2) Estime a complexidade do seu algoritmo.

2. (6,0 pontos) **O problema das pontes**

Imagine que de um lado do rio estão as cidades $A_1, A_2, \ldots, A_n$ e do outro lado do rio estão as cidades $B_1, B_2, \ldots, B_m$



Para cada par de cidades A_i, B_j, existe uma estimativa E_{ij} para o número de pessoas que precisa ir de uma cidade a outra todos os dias.

No momento essa travessia é feita de barco, mas a ideia é construir pontes para facilitar a vida das pessoas.

O objetivo, é claro, consiste em atender o maior número possível de pessoas.

Mas a restrição óbvia é que duas pontes não podem se cruzar.

Por outro lado, pode haver qualquer número de pontes em cada cidade.

Formalmente, o problema consiste em encontrar uma coleção de pares

$$(A_{i_1}, B_{j_1}),\ (A_{i_2}, B_{j_2}),\ (A_{i_3}, B_{j_3}),\ \ldots,\ (A_{i_k}, B_{j_k}),$$

que satisfaçam as restrições sobre a construção das pontes, e cuja soma

$$\sum_{l=1}^{k} E_{i_l, j_l}$$

seja a maior possível.

a) Apresente um algoritmo de programação dinâmica para esse problema, baseado na estratégia I.

Estime a complexidade do seu algoritmo.

b) Apresente um algoritmo de programação dinâmica para esse problema, baseado na estratégia II.

Estime a complexidade do seu algoritmo.

c) Apresente um algoritmo de programação dinâmica para esse problema, baseado na estratégia III.

Estime a complexidade do seu algoritmo.