Projeto e analíse de algoritmos

Amanda Goulart

Janeiro 2022

1 Considere o problema do Longest Common Subsequence. Aplique uma solução por programação dinâmica para o LCS entre as duas sequências: i) seu primeiro nome, ii) seu último nome. Mostre como encontrou a solução.

Temos uma sequência $X = \{x_1, x_2, ..., x_n\}$ e $Y = y_1, y_2, ..., y_n$ é uma subsequência de X se existe uma sequência crescente $\{a1, a2, a3..an\}$ de índices de X nas quais, onde j= 1,2, ... k, temos $X_{ij} = Y_j$

Dado o exemplo X = {A,M,A,N,D,A} e Y = {G,O,U,L,A,R,T}, temos que Y é uma subsequência de X.

Tendo as sequências $X = \{x_1, x_2, ..., x_n\}$ e Y = $\{y_1, y_2, ..., y_n\}$ e $Z = \{z_1, z_2, ..z_n\}$ uma subsequência de X e Y.

Temos que :

1. Se $x_m = y_n$, então $z_k = x_m = y_n$ e z_{k-1} é uma LCS de $x_{m-1} e y_{n-1}$
2. Se $x_m \neq y_n$, então $z_k \neq x_m$ que implica é uma LCS de x_{m-1} e y.
3. Se $x_m \neq y_n$, então $z_k \neq y_m$ implica que z é uma LCS de x e y_{n-1}

Analisando o item 1, neste caso pode-se agregar $x_m = y_n = z_k$ a z para encontrar uma subsequência comum de X e Y. Assim temos que z_{k-1} é uma subsequência de X_{m-1} e Y_{n-1}. Para provar que ela é uma LCS, suponha que existe uma subsequência comum W de X_{m-1} e Y_{m-1} ed comprimento maior que k-1, Enão quando agregarmos $x_m = y_n$ a W, acontece que cria uma subsequência comum > que k, o que contraditorio. Logo temos que $x_m = y_n = z_k$

Para o item 2 , considerando que $z_k \neq x_m$, logo Z é uma subsequência de $X_{m-1} e Y$. Supondo que exista uma sequência comum W de X_{m-1} e Y com comprimento maior que K, logo W também seria uma subsequência comum de X_m e Y, entrando em contradição que z é uma LCS de X e Y.

Para o item 3 , considerando que $z_k \neq x_m$, logo Z é uma subsequência de $X e Y_{n-1}$. Supondo que exista uma sequência comum W de X e Y_{n-1} com comprimento maior que K, logo W também seria uma subsequência comum de X e Y_n, entrando em contradição que z é uma LCS de X e Y.

Dando a solução recursiva de:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 ou j = 0 \\ c[i-1, j-1] + 1 & \text{if } x = i, j > 0 e x_i = y_i \\ max(c[i, j-1], c[i-1, j]) & \text{if } x = i, j > 0 e x_i = y_i \end{cases} \tag{1}$$

Com o seguinte algoritmo:

```
def lcs(X, Y, i, j):
    if i == 0 or j == 0:
        return 0
    elif X[i-1] == Y[j-1]:
```

```
        return 1 + lcs(X, Y, i-1, j-1);
    else:
        return max(lcs(X, Y, i, j-1), lcs(X, Y, i-1, j));

X = "AMANDA"
Y = "GOULART"
print ("LCS = ", lcs(X , Y, len(X), len(Y)) )
```

Resultando em:

```
LCS = 1
```

Com a Bottom-Up Solution:

	a	m	a	n	d	a
g	0	0	0	0	0	0
o	0	0	0	0	0	0
u	0	0	0	0	0	0
l	0	0	0	0	0	0
a	1	1	1	1	1	1
r	1	1	1	1	1	1
t	1	1	1	1	1	1

1.1 Qual o comprimento da solução ótima?

No caso apresentado temos que o comprimento da solução ótima é 1.

1.2 Qual é a subsequência comum encontrada?

E a subsequência apresentada é 'A'.

2 Para um certo feriado você tem disponibilidade para assistir um total de T minutos de tv escolhidos de um conjunto de N filmes no serviço de streaming de vídeo. Considerando que para cada filme i você conhece a sua duração D_i em minutos e a sua avaliação Pi , o seu objetivo é o de encontrar qual a soma máxima das avaliações dos filmes que você poderia assistir no tempo disponível T.

2.1 Demonstre a propriedade de subestrutura ótima do problema. Forneça uma fórmula recursiva para resolver o problema.

Suponhamos que temos um conjunto de $N = \{a_1, a_2, a_3, ..., a_n\}$, onde cada filme possui um tempo de I_i (tempo inicial) e F_f (tempo final), onde $0 \leq I_i < F_i < \infty$. Temos que nos atentar a dois pontos:

1. O filme selecionado occore em um intervalo semiaberto $[I_i, F_i)$
2. Os intervalos de tempos dos filmes não podem se sobrepor

Dado o conjunto $S_{ij} = a_k \in S : I_i \leq F_k < F_k \leq I_j$, onde S_{ij} é um subconjunto de S, que se inicia no filme i e termina no filme j.

Para os casos onde S_{ij} é vazio, ocorre quando $i \leq j$. Pois vamos supor que existe um filme $a_k \in$S$_{ij}$ para $i \leq j$, onde estão ordenados, logo $F_i \leq I_k < F_k \leq I_j < F_j$. Como $F_i < F_j$ cotradiz o conceito que foi suposto de ordenação, logo assumindo que ordenamos as atividades em ordem crescente de tempo de término , nosso espaço de subproblemas é selecionar um subconjunto de tamanho máximo de atividades mutuamente compatíveis de S_{ij} $\forall 0 \leq i < j \leq n+1$.

Suponhamos que a solução ótima A_{ij} para S_{ij} onde existe o filme a_k. Logo então as soluções A_{ik} para S_{ik} e A_{kj} para S_{kj} sendo soluções ótimas, logo S_{ij} também será uma solução ótima. E assim temos a subestrutura do problema.

Para a solução recursiva, seja c [i, j] o número de atividades em um subconjunto de tamanho máximo de atividades mutuamente compatíveis em S_{ij} . Temos c [i, j] = 0 sempre que S_{ij} é vazio ; em particular, c [i, j] = 0 para $i \leq j$.

Para conjuntos não vazios de S_{ij}, se A_k é um subconjunto de tamanho máximo compativel de S_{ij}, também será possivel os subproblemas S_{ik} e S_{kj}.

E assim temos a equação de recorrência:

$c[i, j] = c[i, k] + c[k, j] + 1$

Logo:

$$\text{c[i, j]} = \begin{cases} 0 & \text{if } S_{ij} = \\ max(c[i,k] + c[k,j] + 1) & \text{if } S_{ij} \neq \end{cases}$$

(2)

2.2 Forneça um critério guloso para resolver o problema. Caso este critério garanta a solução ótima, demonstre a sua propriedade gulosa. Caso contrário, mostre um exemplo para o qual esse algoritmo falha.

Suponhamos um subproblema onde S_{ij} não vazio, onde existe um filme am com tempo ótimo.

Logo, teremos que ter:

1. O filme a_m é utilizado no conjunto S_{ij}
2. O subproblema S_{im} é vazio, assim caso acontecer a escolha do filme a_m, o subproblema S_{mj} sera ó unico conjunto não vazio.

Primeiramente para o item 1, iremos supor que A_{ij} é um subconjunto de S_{ij}, ordenando os fimes em ordem crescente de acordo com o tempo. Sendo a_k é a primeira atividade de A_{ij} .

Quando $a_k = a_m$ se encerra, pois quando a_m esta inserido no subconjunto de S_{ij}. Quando $a_k \neq a_m$, será feito um subconjunto $A_{ij} = A_{ij} - a_k \cup a_m$, onde este conjunto os filmes serão disjuntos.

Para o item 2, suponhamos que S_{im} não é vazio, e existe um filme a_k tal que $F_i \leq I_k < F_k \leq I_m < F_m$. Logo a_k está contido em S_{ij} e um termino antes de a_m. Sendo assim, $S_{im} vazio$

2.3 Forneça um algoritmo que obtenha a solução ótima por programação dinâmica.

```
RECURSIVE-MOVIE-SELECTOR(s, f, i, j)
m    i + 1
while m < j and sm < fi
   do m    m + 1
if m < j
```

```
then return {am}   RECURSIVE-MOVIE-SELECTOR(s, f, m, j)
else return 0
```

3 Seja um conjunto S de inteiros e um inteiro positivo k. Considere o problema de se encontrar um subconjunto $X \in S$ de tamanho k cuja soma seja máxima entre todos possíveis subconjuntos de S.

3.1 Forneça um algoritmo guloso para resolver o problema.

```python
def get_maximum_subarray_sum(arr):
  for i in range(1, len(arr)):
    if arr[i-1] > 0:
      arr[i] += arr[i-1]
  return max(arr)
```

3.2 Mostre que o algoritmo encontra a solução ótima. Caso contrário, mostre um contra-exemplo.

Suponhamos que a soma do subconjunto P que termina da posição P[i-1]. Para achar a soma na posição P[i], adicionamos o elemento P[i-1], ao elemento P[i], somente se S[i-1] $>$ 0. Caso contrário, não vale de nada. Logo temos a equação de recorrência dada por:

$S[i] = A[i] + S[i-1]$

Logo:

S[i] $= \begin{cases} A[i] + S[i-1] & \text{if } S[i-1] > 0 \\ A[i] & \text{if } S[i-1] \leq 0 \end{cases}$

Provando por invariante de laço:

Inicilização:

Se analizarmos o loop , podeos perceber que ele se encerra no primeiro elemento. O final do array, com o primeiro elemento é o próprio valor máximo.

Manutenção:

Em cada laço, adiciona o valor máximo anteior, se e somente se este valor for maior que 0, caso não mantém o valor atual. Logo a condição de manutenção é mantida.

Encerramento:

Como a condição do laço é mantida até o seu encerramento, temos que ao finalizar estará correto e retornará a soma do conjunto P.