Coding dojo

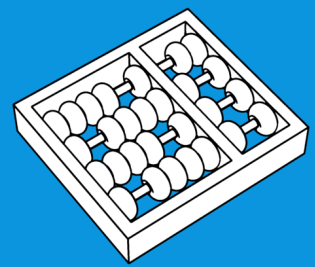
MC102 - Algoritmos e Programação de Computadores

04/25

8

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

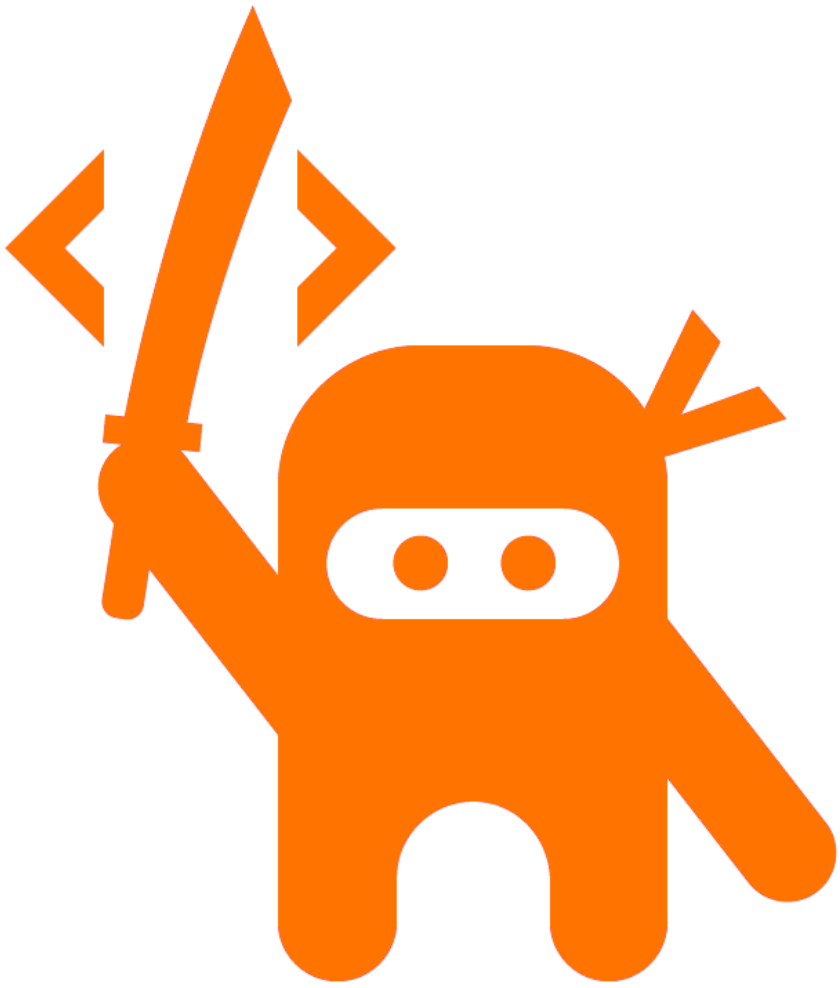
UNICAMP



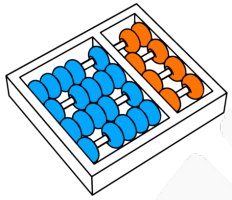
"O dojo é um local de superação, vinculo familiar e propagação da sabedoria ninja."

Sensei Cícero Melo Hosho Ryu Ninpo.

Soluções para os exercícios!



CODE
DOJO



Mínimos locais

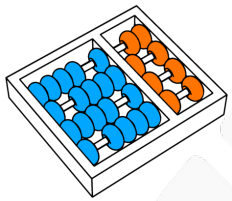
Dada uma sequência de números reais S, um valor é um mínimo local, se os dois adjacentes a ele forem maiores (isto é, $s_{i-1} > s_i < s_{i+1}$). Faça um programa que leia uma sequência de inteiros S e imprima as posições dos mínimos locais nela.

Exemplo de entrada:

8 2 1 2 5 2 4 7 5 2 6 2 4

Exemplo de saída (para a entrada do exemplo):

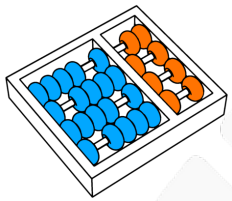
3 6 10 12



Solução

```
1  seq = list(map(int, input().split()))
2
3  for i in range(1, len(seq) - 1):
4          if seq[i] < seq[i-1] and seq[i] < seq[i+1]:
5                  print(i+1, end=' ')
6
7  print()
```

Os possíveis mínimos locais vão do índice 1 até o len(seq) - 2, ou seja excluímos o primeiro (0) e último (len(seq)-1) elementos da nossa análise. Para cada índice i a ser analisado, verificamos se o elemento é mínimo local, para isso deve ser menor que o do índice i-1 (antecessor) e o do índice i+1 (sucessor). Ao imprimir, colocamos i+1, pois no exercício as posições devem começar por 1 e não por 0 (ou seja, cada posição é 1 a mais do que o índice). Também modificamos o fim de linha da impressão, para que todos os números fiquem na mesma linha. Já o último print vazio é para imprimir uma troca de linha.



Menor, maior, média, mediana e moda

Faça um programa que leia uma sequência de inteiros S e imprima o menor valor da sequência, o maior, a média (até duas casas decimais), a mediana (elemento com igual número de maiores (ou iguais) que de menores ou iguais) e a moda (elemento que mais se repete).

Exemplo de entrada:

8 2 1 2 5 2 4 7 5 2 6 2 4

Exemplo de saída (para a entrada do exemplo):

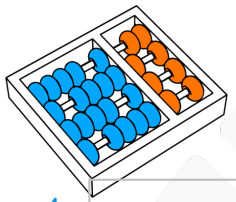
Menor: 1

Maior: 8

Media: 3.84

Mediana: 4

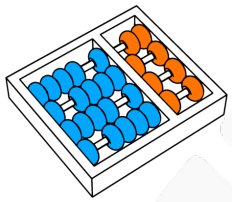
Moda:2



Solução. Código

```
1  seq = list(map(int, input().split()))
2
3  seq.sort()
4
5  print('Menor:', seq[0]) # O menor é o primeiro
6  print('Maior:', seq[-1]) # O maior é o último
7  print('Media:', '{:.2f}'.format(sum(seq)/len(seq)))
8
9  if len(seq) % 2 == 1:
10         print('Mediana:', seq[len(seq)//2])
11 else:
12         print('Mediana:', '{:.2f}'.format((seq[len(seq)//2-1] + seq[len(seq)//2]) / 2))
13
14 moda = atual = seq[0]
15 moda_reps = atual_reps = 1
16 for i in range(1, len(seq)):
17         if atual == seq[i]:
18                 atual_reps += 1
19                 if atual_reps > moda_reps:
20                         moda = atual
21                         moda_reps = atual_reps
22         else:
23                 atual = seq[i]
24                 atual_reps = 1
25 print('Moda:', moda)
```

Alguns comentários no próximo slide.

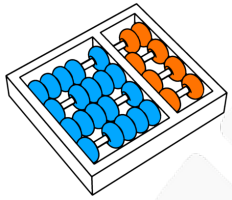


Solução. Comentários

A forma mais simples de encontrar os valores é ordenando a sequência:

- O menor é o primeiro, o maior o último e a mediana o que está no meio (se houver dois no meio, quantidade par, então é a média entre eles).
- Para a média, basta somar (usando a função sum) e dividir pela quantidade.
- No caso da moda, como os elementos estão já ordenados, os iguais ficam juntos.

  Assim, podemos percorrer o vetor uma única vez para encontrar a moda.
  - Para isso, usamos duas variáveis, uma com o candidato a moda encontrado até o momento e outra com o valor do número que estamos atualmente analisando (moda e atual).
  - Para ambos, contamos o número de repetições (moda_reps e atual_reps).
  - Ao percorrer o vetor, verificamos se o elemento na posição corrente é igual ao que consideramos 'atual', se for, então aumentamos o número de repetições dele, e nesse caso podemos verificar se precisamos atualizar a moda.
  - Se o elemento na posição corrente não for igual ao 'atual', então já temos um novo atual, com uma ocorrência.



Subsequência

Uma sequência S_1 é subsequência de uma S_2 se todos os elementos de S_1 aparecem de forma consecutiva e na mesma ordem em S_2. Faça um programa que leia duas sequências de inteiros S_1 e S_2 e imprima se S_1 é ou não subsequência de S_2.

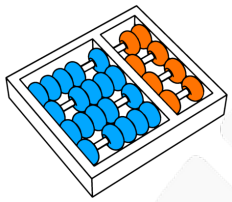
Exemplo de entrada:

2 6 2 4

8 2 1 2 6 2 4 7 5

Exemplo de saída (para a entrada do exemplo):

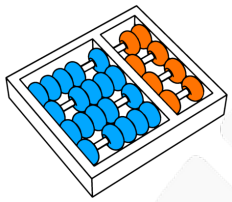
[2, 6, 2, 4] eh subsequencia de [8, 2, 1, 2, 6, 2, 4, 7, 5].



Solução

```
1  sub = input()
2  seq = input()
3
4  n_sub = len(sub)
5  n_seq = len(seq)
6
7  esta = False
8  i = 0
9  while not esta and i + n_sub <= n_seq:
10         j = 0
11         while j < n_sub and sub[j] == seq[i + j]:
12                 j += 1
13         if j == n_sub:
14                 esta = True
15         i += 1
16
17 if esta:
18         print(sub, 'eh subsequencia de', seq)
19 else:
20         print(sub, 'nao eh subsequencia de', seq)
```

Para que uma sequência seja subsequência de outra, na sequência maior, deve haver uma posição i, tal que a menor comece (por exemplo 'ana' começa na posição 1 de 'banana'). Então, por cada posição i da sequência maior, testamos se cada elemento da menor está a partir de i. Para isso, iteramos com uma variável j cada posição da sequência menor e verificamos se o elemento dessa posição é o mesmo que o da sequência maior na posição i+j. Se conseguimos chegar até o final da sequência menor nessa checagem, significa que ela está na maior.



União de vetores ordenados

Faça um programa que:

(a) Leia duas sequências de números reais S_1 e outra S_2. Assumindo que os elementos de S_1 estejam ordenados e que os de S_2 também estejam ordenados, imprima (em ordem) os elementos da união entre S_1 e S_2.

(b) Solucione o item anterior sem usar **sort()**.

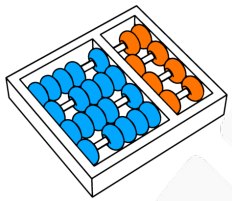
Exemplo de entrada:

2 4 7

1 5 6 8

Exemplo de saída (para a entrada do exemplo):

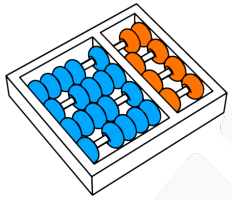
1 2 4 5 6 7 8



Solução

```
1  vetor1 = list(map(int, input().split()))
2  vetor2 = list(map(int, input().split()))
3
4  n1 = len(vetor1)
5  n2 = len(vetor2)
6
7  uniao = [0] * (n1 + n2)
8  i = j = 0
9  for k in range(len(uniao)):
10         if i < n1 and (j == n2 or vetor1[i] <= vetor2[j]):
11                 uniao[k] = vetor1[i]
12                 i += 1
13         else:
14                 uniao[k] = vetor2[j]
15                 j += 1
16
17 print(uniao)
```

Sabemos que o tamanho do vetor resultante é a soma dos tamanhos dos outros dois, assim criamos esse vetor e preenchemos cada posição dele. Para preencher as posições, mantemos dois índices para cada um dos vetores menor, o i que indica o elemento do primeiro vetor que devemos adicionar e o j o do segundo. Para adicionar o elemento da posição i no primeiro vetor, este deve ser menor que o da posição j do segundo. Nesse caso, como já adicionamos o i, o próximo elemento do primeiro vetor a ser adicionado deve ser o i+1. Nessa análise, alguns cuidados são importante, como garantir que não tenhamos adicionado já todos os elementos do primeiro vetor (i < len(vetor1)). Note que se não formos adicionar o elemento na posição i do primeiro vetor, é porque alguma condição não se satisfaz e portanto adicionamos o da posição j do segundo.



Subsequência de soma máxima

Faça um programa que leia uma sequência de reais S e encontrar duas posições i e j na sequência, tais que a soma dos elementos desde a posição i até a j seja máxima ($i <= j$). Imprima as posições e o valor da soma dos elementos da posição i até a j.

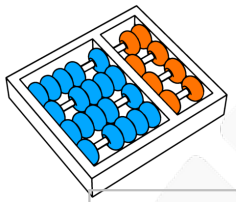
Exemplo de entrada:

$$3\ 4\ -5\ 3\ 3\ -8\ 2$$

Exemplo de saída (para a entrada do exemplo):

Intervalo: [1, 5]

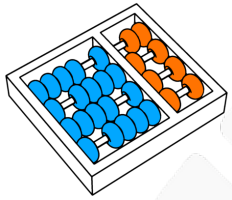
Soma: 8



Solução

```
1  seq  =list(map(int, input().split()))
2
3  soma_max = seq[0]
4  i_max = f_max = 0
5  soma_sufix = 0
6  i_sufix = 0
7  for i in range(len(seq)):
8          soma_sufix += seq[i]
9          if soma_sufix > soma_max:
10                 soma_max = soma_sufix
11                 i_max = i_sufix
12                 f_max = i
13         if soma_sufix < 0:
14                 soma_sufix = 0
15                 i_sufix = i + 1
16
17 print('Intervalo:', [i_max + 1, f_max + 1])
18 print('Soma:', soma_max)
```

Imaginemos que calculamos a soma máxima considerando só até a posição $i - 1$. Para calcular o da posição i temos duas opções. A sequência com maior soma usa o elemento na posição i, pelo que é o sufixo do sub-vetor até a posição i com maior soma. Ou o elemento da posição i não está, pelo que a sequência com maior soma é a mesma da $i - 1$. Assim, mantemos duas somas, a melhor até o momento (soma_max), com seus índices de início e fim (i_max e f_max); e a soma do melhor sufixo (soma_sufix) com seu índice inicial (i_sufix), note que o final é o índice atual i. A cada iteração, só precisamos atualizar a soma do sufixo, se esta for maior que a melhor soma, então atualizamos. Se não, a soma do sufixo certamente não aumentará se ela fica negativa, nesse caso é melhor reiniciar ela para a próxima posição. Na hora de imprimir os intervalos somamos 1, posi no exercício as posições devem começar por 1 e não por 0.



Três somam zero?

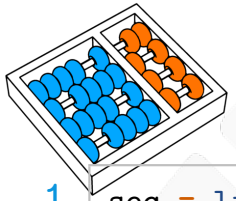
Faça um programa que leia uma sequência de reais S e, se houver três valores na sequência cuja soma é zero, os imprima, caso contrário imprima que não há três valores cuja soma é zero.

Exemplo de entrada:

$$3\ 4\ -5\ 3\ 3\ -8\ 2$$

Exemplo de saída (para a entrada do exemplo):

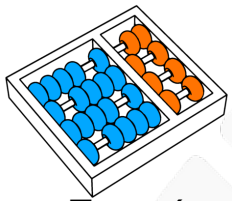
$$3\ +\ -5\ +\ 2\ =\ 0$$



Solução

```
1  seq = list(map(int, input().split()))
2  seq.sort()
3  n = len(seq)
4  ha = False
5  k = 0
6  while not ha and k < n:
7          i = k + 1
8          j = n - 1
9          while i < j and seq[k] + seq[i] + seq[j] != 0:
10                 if seq[k] + seq[i] + seq[j] > 0:
11                         j -= 1
12                 else:
13                         i += 1
14         if i < j:
15                 print(seq[k], '+', seq[i], '+', seq[j], '= 0')
16                 ha = True
17         K += 1
18 if not ha:
19         print('Nao ha tres que somem zero')
```

Se ordenamos primeiro os elementos do vetor, uma solução pode ser a seguinte. Para cada índice k verificamos se há dois índices $i < j$ maiores que k, tal que os elementos nesses índices somem zero. Para cada k podemos começar com i sendo o próximo $(k + 1)$ e j o último $(n - 1)$. Se a soma dos elementos nessas posições for positiva, isso significa que estamos somando mais do necessário ao elemento da posição k, ou seja temos que somar menos (aumentar o i só faria somar mais pois os elementos estão ordenados, portanto a única opção é diminuir o j). Já se a soma ficar negativa, é o contrário, aumentamos i pois está faltando algo a ser somado. Repetimos esse processo até encontrar uma soma que seja zero, ou que o i fique maior que o j, se isto último acontecer é porque com esse k não há dois maiores que somem zero. Para saber se foi encontrada ou não a soma zero, usamos a variável booleana 'ha', e repetimos o processo para cada índice k enquanto 'ha' se mantenha negativa ou não cheguemos no final do vetor.



Escolhendo números

Este é um jogo entre duas pessoas A e B que se alternam escolhendo números em uma dada sequência S com quantidade par de elementos. As regras são as seguintes:

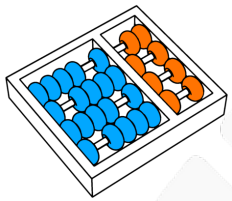
- Começa o jogador A.
- Na sua vez, o jogador escolhe um dos extremos da sequência S e o remove da sequência, passando o turno para o outro jogador.
- O jogo acaba quando todos os números foram removidos.
- Ganha o jogador que removeu a maior soma.

Sabendo que você é o jogador A e supondo que o B é um especialista no jogo. Crie uma estratégia em que A nunca perde! Faça um programa que leia a sequência de números e imprima a soma dos números selecionados por A seguindo sua estratégia.

Exemplo de entrada:

```
10 1 1 1 1 1
```

Exemplo de saída (para a entrada do exemplo):



Solução

```
1 seq = list(map(int, input().split()))
2 soma_par = sum(seq[::2])
3 soma_impar = sum(seq[1::2])
4 if soma_par >= soma_impar:
5         print(soma_par)
6 else:
7         print(soma_impar)
```

Aqui basta observar que, como o vetor tem um número par de elementos, o primeiro jogador pode sempre forçar o segundo a só escolher as posições pares ou só escolher as posições ímpares. O primeiro jogador pode escolher entre a posição 0 (primeira) e a $n-1$ (última). Como n é par, $n-1$ é ímpar. Se o primeiro jogador escolhe a 0, então o segundo só pode escolher a 1 ou a $n-1$, ambas ímpares. Se o primeiro escolhe a $n-1$, então o segundo só pode escolher a 0 ou a $n-2$, ambas pares. Essa ideia pode ser mantida até o fim do jogo, controlando sempre a opção de escolha do segundo jogador. Desta forma, o primeiro jogador só precisa somar os elementos das posições ímpares e os das pares, escolhendo aquele que tiver maior soma.

Coding dojo

MC102 - Algoritmos e Programação de Computadores

04/25

8

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP