PROBLEMA

Duas pessoas, Marcelo e Carla, trabalham em uma fazenda de maçãs na qual existem N árvores. Asárvores estão alinhadas em uma fileira e são numeradas de 1 a N. Marcelo planeja coletar maçãs de K árvores consecutivas e Carla de L árvores consecutivas. Eles querem escolher segmentosdisjuntos, ou seja, que não se sobrepõem para não interferirem na coleta do outro. Qual o maiornúmero de maçãs que os dois juntos podem coletar?

Escreva uma função:

def get_max_apples(A, K, L)

que, dado um vetor A consistindo de N inteiros denotando o número de maçãs em cada árvore, einteiros K e L, respectivamente, o número de árvores que Marcelo e Carla escolhem coletar, retorneo número máximo de maçãs coletadas ou -1 se não existem intervalos que permitam a coleita.

Por exemplo, dado um vetor A = [3, 4, 1, 7, 8, 5], K = 2, L = 3, sua função deve retornar27, já que Carla escolherá as árvores 4, 5 e 6 e coletar 7 + 8 + 5 = 20 maçãs. Marcelo, por sua vez,escolherá as árvores 1 e 2, coletando assim 3 + 4 = 7 maçãs.

SOLUÇÃO

*** BACKEND**

Partindo de um olhar técnico, podemos observar que os números inteiros K e L, representam o número de elementos de subconjutos de A, onde o somatório dos elementos dos 2 conjuntos deve ser o maior possível.

Sendo assim podemos a partir de uma matriz, representar a soma de todos os elementos dos subconjuntos possíveis, de tamanhos K e L, onde os subconjuntos de tamnho K representarão as linhas, e os subconjuntos de tamanho L representarão as colunas.

Exemplo:

Para,

A = [3, 4, 1, 7, 8, 5]

K = 2

L = 3,

temos:

	[3,4,1]	[4,1,7]	[1,7,8]	[7,8,5]
[3,4]	x	x	23	27
[4,1]	x	x	x	25
[1,7]	x	x	x	x
[7,8]	23	x	x	x
[8,5]	21	25	x	x

A matriz acima representa o somatório entre os subconjuntos, as células que possuem um **x** siginifca que não é possível obter uma soma, pois os subconjuntos se sobrepõem.

A partir dessa analogia, podemos obter o seguinte algoritmo:

```
maior = -1
para i de 1 até K faça
    para j de 1 até L faça
        se (j >= i + K) ou (i > (j+L)) então
            acumulador = 0
            para x de i até i+K faça
                acumulador += A[x]
            fimpara
            para x de j até i+L faça
                acumulador += A[x]
            fimpara
            se maior < acumulador então
                maior = acumulador
            fimse
        fimse
```

fimpara

fimpara

Não foi necessário criar de fato uma matriz, pois a partir dos somatórios, conseguimos obter o de maior valor, em tempo de execução e indica-lo.

No trecho "*se (j >= i + K) ou (i > (j+L)) então"* verificamos se é possível fazer o somatório daquela célula, pois os subconjuntos não podem se sobrepor. Isso foi possível pois os índices i e j representam o início daquele subconjunto dentro de A, ex:

O subconjunto [7,8], está na linha 4 da matriz, e dentro do conjunto A, o número 7 está na posição 4.

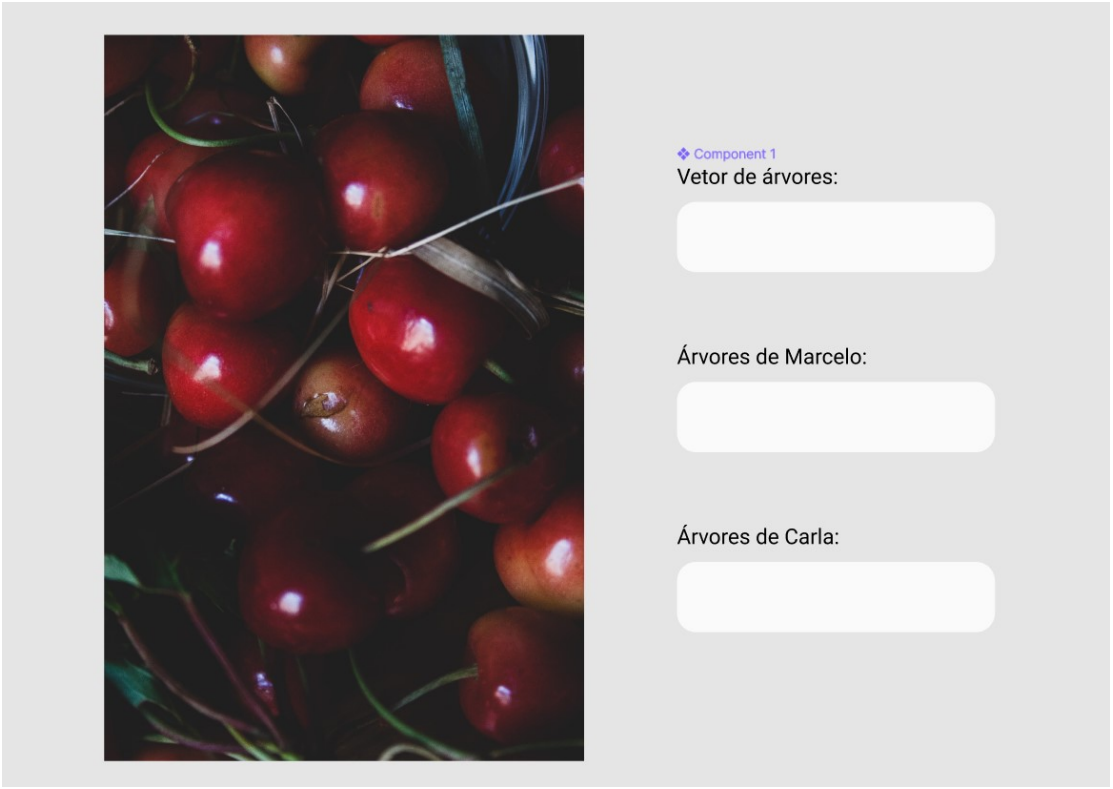
Já o subconjunto da coluna 1 da matriz, representado por [3,4,1], dentro do conjunto A, o número 3 está na posição 1.

A partir disso foi necessário apenas fazer a relação com o número de elementos de K e de L, para saber se o número de elementos passavam do início i ou j.

***FRONTEND**

Utilizando o figma

Tela inicial



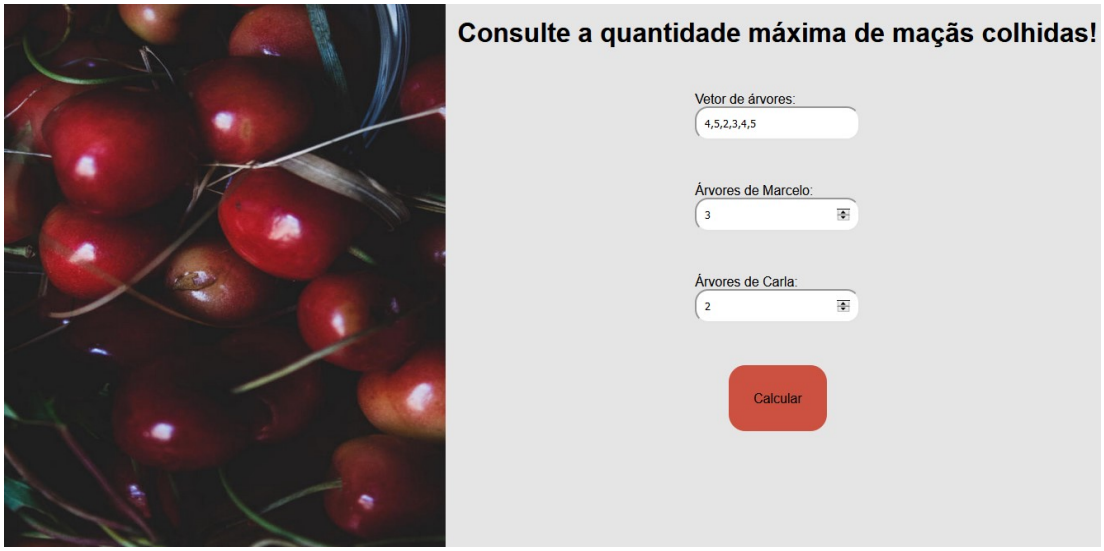
Tela do resultado

Na tela inicial temos as seguintes entradas:

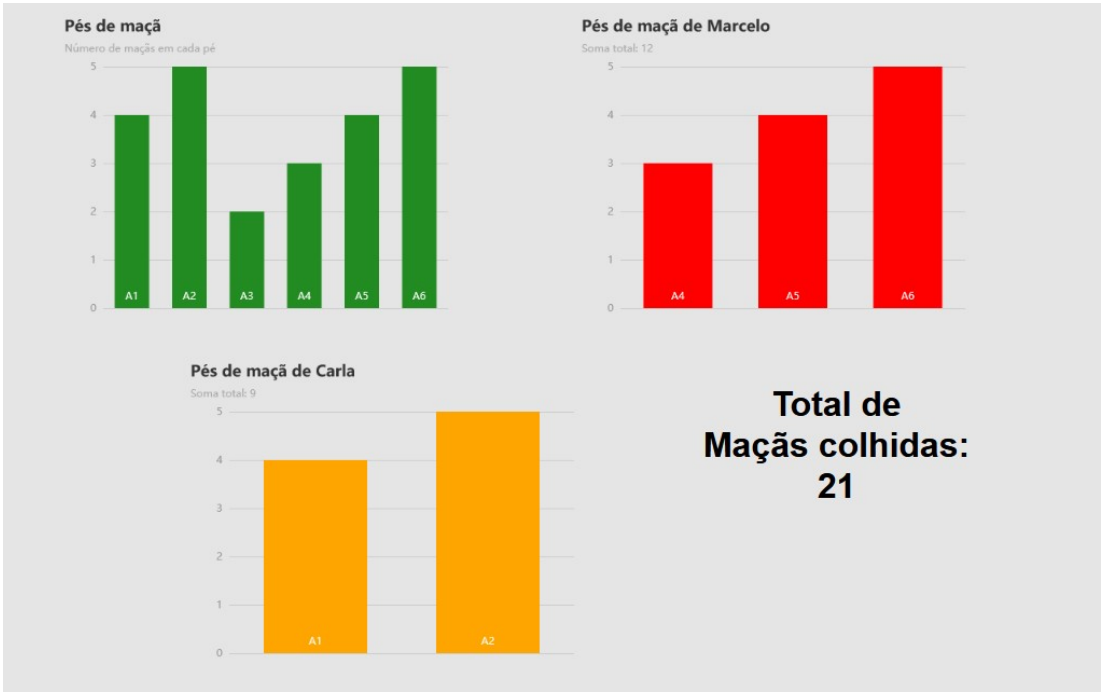
1. Vetor de árvores = deve ser no formato: X1,X2,X3...,Xn
2. Árvores de marcelo = Deve ser um inteiro > 0
3. Árvores de Carla = Deve ser um inteiro > 0

RESULTADO DAS TELAS EM PRÁTICA

TELA INCIAL 2



TELA RESULTADO 2



EXECUÇÃO:

*** Frontend:**

- npm install

- npm run serve

*** Backend (python3.8)**

- **pip install -r requirements.txt**
- **setar variável de ambiente**
    - **linux ou mac:**
        - **export FLASK_APP=backend.py**
        - **-** execução: **flask run** ou **python -m flask run**
    - **windows:**
        - **powershell: $env:FLASK_APP = "backend.py"**
        - **cmd: set FLASK_APP=backend.py**
        - **-** execução: **python -m flask run**