Entrega 4

Daniel Brito

12) Dado um conjunto de inteiros $A = \{a_1, a_2, ..., a_n\}$ e um inteiro t, o seguinte algoritmo decide se o somatório de algum subconjunto de A é maior que t:

```
import sys

def somatorioMaiorT(a, t):

    soma = 0
    l = sys.maxsize
    n = len(a)

    esq = 0

    for dir in range(n):
        soma += a[dir]

        while soma > t and esq <= dir:
            l = min(l, dir - esq + 1)

            soma -= a[esq]
            esq += 1

    return l != sys.maxsize
```

13) Dado um conjunto de inteiros $A = \{a_1, a_2, ..., a_n\}$ e um inteiro t, o seguinte algoritmo decide se o somatório de algum subconjunto de A é igual a t:

```
def somatorioIgualT(a, n, soma):

  sub = ([[False for i in range(soma+1)] for i in range(n+1)])

  for i in range(n+1):
    sub[i][0] = True

  for i in range(1, soma+1):
    sub[0][i]= False

  for i in range(1, n+1):
    for j in range(1, soma+1):

      if j < a[i-1]:
        sub[i][j] = sub[i-1][j]

      if j >= a[i-1]:
        sub[i][j] = (sub[i-1][j] or sub[i-1][j - a[i-1]])

  return sub[n][soma]
```

14) Dado um conjunto de inteiros $A = \{a_1, a_2, ..., a_n\}$, o seguinte algoritmo determina se é possível particioná-lo em duas partes, de maneira que o somatório dos elementos de cada parte é igual, retornando as partições ou -1, caso não seja possível:

```
def particaoIgual(A):
  for i in range(len(A)):
    soma_esq = 0

    for j in range(i):
      soma_esq += A[j]

    soma_dir = 0

    for j in range(i, len(A)):
      soma_dir += A[j]

    if soma_esq == soma_dir:
      return [A[:i], A[i:]]

  return -1
```

15) Dado um conjunto de inteiros $A = \{a_1, a_2, ..., a_n\}$, o seguinte algoritmo determina se é possível particioná-lo em duas partes, de maneira que a diferença absoluta entre o somatório de cada parte seja mínima:

```
import sys

def findMin(A):
  soma = sum(A)
  n = len(A)

  dp = [[0 for i in range(soma+1)]
       for j in range(n+1)]

  for i in range(n+1):
    dp[i][0] = True

  for j in range(1, soma+1):
    dp[0][j] = False

  for i in range(1, n+1):
    for j in range(1, soma+1):

      dp[i][j] = dp[i-1][j]

      if a[i-1] <= j:
        dp[i][j] |= dp[i-1][j-a[i-1]]

  dif = sys.maxsize

  for j in range(soma//2, -1, -1):
    if dp[n][j] == True:
      dif = soma - (2*j)
      break

  return dif
```

16) Dado um conjunto de inteiros $A = \{a_1, a_2, ..., a_n\}$, `particionaTrio` particiona A em três partes cujo somatório é igual entre si. Logo após, `verificaParticoes` verifica se foi possível realizar a divisão com base nesta restrição, imprimindo o devido resultado:

```
def particionaTrio(A, pos1, pos2):
  n = len(A);

  prefixo = [0] * n;
  soma = 0;

  for i in range(n):
    soma += A[i];
    prefixo[i] = soma;

  sufixo = [0] * n;
  soma = 0;

  for i in range(n - 1, -1, -1):
    soma += A[i];
    sufixo[i] = soma;

  somaTotal = soma;

  i = 0;
  j = n - 1;

  while i < j-1:
    if prefixo[i] == somaTotal // 3:
      pos1 = i;

    if sufixo[j] == somaTotal // 3:
      pos2 = j;

    if pos1 != -1 and pos2 != -1:
      if sufixo[pos1 + 1] - sufixo[pos2] == somaTotal // 3:
        return [True, pos1, pos2];
      else:
        return [False, pos1, pos2];

    if prefixo[i] < sufixo[j]:
      i += 1;
    else:
      j -= 1;

  return [False, pos1, pos2];
```

```
def verificaParticoes(A):

  pos1 = -1;
  pos2 = -1;

  res = particionaTrio(A, pos1, pos2);

  pos1 = res[1];
  pos2 = res[2];

  if res[0]:
    for i in range(pos1 + 1):
      print(A[i], end = " ");

    print("")

    for i in range(pos1 + 1, pos2):
      print(A[i], end = " ");

    print("")

    for i in range(pos2, len(A)):
      print(A[i], end = " ");
  else:
    print("Nao foi possivel particionar");
```

17) Abaixo, o algoritmo de Bellman-Ford alterado de forma que o conteúdo do primeiro laço precise ser executado apenas $m + 1$ vezes, mesmo que o valor de m seja desconhecido:

```
BELLMAN-FORD-ALT(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
changes = True
while changes == True:
    changes = FALSE
    for each edge (u, v) in G.E.
        RELAX-M(u, v, w)

RELAX-M(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.PI = u
    changes = True
```

18) TO-DO

19) TO-DO

20) TO-DO