Instituto Superior Técnico

Lic. em Matemática Aplicada e Computação
Mestrado Integrado em Eng. Biomédica

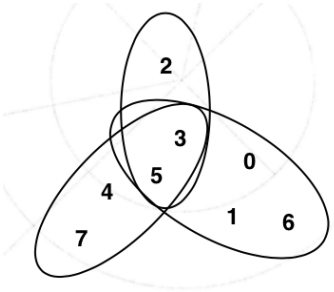
Elementos de Programação

3 de Fevereiro de 2020 Exame 2 Duração: 2h30

Grupo I (2.5+2.5 valores)

Neste exercício não pode usar definições por compreensão nem métodos. As únicas operações sobre listas permitidas são: lista vazia ([]), acesso aos elementos da lista por posição (lista[posição]), seccionamento da lista (lista[posição:posição]), comparação com a lista vazia (==[]), cálculo do comprimento (`len`) e concatenação (+).

Uma lista de listas de números (que designamos por *pétalas* e que assumimos ordenadas e sem repetições) forma um *girassol* se a intersecção de quaisquer duas pétalas distintas é constante (o centro do girassol, também ordenado e sem repetições).
Na figura, podemos observar o girassol `[[2,3,5],[0,1,3,5,6],[3,4,5,7]]`, com 3 pétalas, cujo centro é `[3,5]`.



Comece por definir imperativamente em *Python*, de forma eficiente, uma função `inter` que dadas duas pétalas `u,v` e um centro `s` devolve o par `(r,q)` onde `r` é a intersecção das pétalas `u` e `v`, e `q` é um valor Booleano, `True` se e só se `r` é igual ao centro `s`.

Use-a, depois, para definir, de forma eficiente, uma função `giraQ` que dada uma lista de pétalas `w` devolve o par `(b,c)` onde `b` é um valor Booleano, `True` se e só se `w` forma um girassol, caso em que `c` deverá ser o centro do mesmo.

Resolução:

```
def inter(u,v,s):
    r=[]
    i=0
    j=0
    k=0
    ok=True
    while i<len(u) and j<len(v):
        if u[i]<v[j]:
            i=i+1
        elif u[i]>v[j]:
            j=j+1
        else:
            r=r+[u[i]]
            if ok and k<len(s) and u[i]==s[
                k=k+1
            else:
                ok=False
            i=i+1
            j=j+1
    return (r,ok and k==len(s))
```

```
def giraQ(w):
    inic=True
    ok=True
    i=0
    while ok and i<len(w)-1:
        j=i+1
        while ok and j<len(w):
            if inic:
                c=inter(w[i],w[j],[])[0]
                inic=False
            else:
                ok=inter(w[i],w[j],c)[1]
            j=j+1
        i=i+1
    return (ok,c)
```

Grupo II (4+4 valores)

Considere *matrizes esparsas* (de números, com as posições numeradas a partir de zero como é usual em *Python*) com as operações:

- `const(n,m,x)`: matriz esparsa com `n` linhas, `m` colunas, e todas as entradas com valor `x`;
- `hjunta(sm1,sm2)`: matriz esparsa que resulta de concatenar horizontalmente as matrizes esparsas `sm1` e `sm2` (com o mesmo número de linhas);
- `vjunta(sm1,sm2)`: matriz esparsa que resulta de concatenar verticalmente as matrizes esparsas `sm1` e `sm2` (com o mesmo número de colunas);
- `sub(sm,i1,i2,j1,j2)`: matriz esparsa que resulta de tomar apenas as linhas entre `i1` e `i2` (com `i1`$\leq$`i2` e excluindo `i2`) e as colunas entre `j1` e `j2` (com `j1`$\leq$`j2` e excluindo `j2`) da matriz esparsa `sm`;
- `dim(sm)`: dimensão da matriz esparsa `sm`, i.e., par formado pelos seus números de linhas e de colunas;
- `nzpos(sm)`: lista das posições `(i,j)` da matriz esparsa `sm` a que correspondem entradas diferentes de zero;
- `val(sm,i,j)`: valor que está na posição `(i,j)` da matriz esparsa `sm`;
- `adic(sm,i,j,x)`: matriz esparsa que resulta de somar o valor `x` na posição `(i,j)` da matriz esparsa `sm`.

a) Em *Python*, pretende-se representar uma matriz esparsa como um triplo `(n,m,nuc)` onde `n` é o número de linhas da matriz e `m` o número de colunas, havendo duas opções para a representação do núcleo `nuc` (que conterá a informação relevante sobre as entradas não nulas da matriz).

Opção 1: `nuc` é um dicionário da forma `{...,(i,j):x,...}`, para cada posição `(i,j)` da matriz esparsa cujo valor `x` seja diferente de zero.

Opção 2: `nuc` é uma lista de triplos da forma `[...,(i,j,x),...]`, para cada posição `(i,j)` da matriz esparsa cujo valor `x` seja diferente de zero.

Escolha uma das duas opções, indicando-a claramente, e apresente implementações eficientes para todas as operações.

Resolução:

```
############## Opção 1 ##############

def const(n,m,x):
    if x==0:
        return (n,m,{})
    else:
        return (n,m,{(i,j):x for i in range(n) for j in range(m)})

def hjunta(sm1,sm2):
    (n1,m1,nuc1)=sm1
    (n2,m2,nuc2)=sm2
    assert n1==n2
    nuc=nuc1.copy()
    for ((i,j),x) in nuc2.items():
        nuc[(i,j+m1)]=x
    return (n1,m1+m2,nuc)

def vjunta(sm1,sm2):
    (n1,m1,nuc1)=sm1
    (n2,m2,nuc2)=sm2
    assert m1==m2
    nuc=nuc1.copy()
    for ((i,j),x) in nuc2.items():
        nuc[(i+n1,j)]=x
    return (n1+n2,m1,nuc)

def sub(sm,i1,i2,j1,j2):
    (n,m,nuc)=sm
    return (i2-i1,j2-j1,{(i-i1,j-j1):x for ((i,j),x) in nuc.items() if i1<=i<i2 and j1<=j<j2})

def dim(sm):
    (n,m,nuc)=sm
    return (n,m)

def nzpos(sm):
    (n,m,nuc)=sm
    return list(nuc.keys())

def val(sm,i,j):
    (n,m,nuc)=sm
    return nuc.get((i,j),0)

def add(sm,i,j,x):
    (n,m,nuc)=sm
    new=nuc.copy()
    old=nuc.get((i,j),0)
    if x!=0 and old+x!=0:
        new[(i,j)]=old+x
    elif x!=0:
        new.pop((i,j))
    return (n,m,new)
```

```
############## Opção 2 ##############

def const(n,m,x):
    if x==0:
        return (n,m,[])
    else:
        return (n,m,[(i,j,x) for i in range(n) for j in range(m)])

def hjunta(sm1,sm2):
    (n1,m1,nuc1)=sm1
    (n2,m2,nuc2)=sm2
    assert n1==n2
    return (n1,m1+m2,nuc1+[(i,j+m1,x) for (i,j,x) in nuc2])

def vjunta(sm1,sm2):
    (n1,m1,nuc1)=sm1
    (n2,m2,nuc2)=sm2
    assert m1==m2
    return (n1+n2,m1,nuc1+[(i+n1,j,x) for (i,j,x) in nuc2])

def sub(sm,i1,i2,j1,j2):
    (n,m,nuc)=sm
    return (i2-i1,j2-j1,[(i-i1,j-j1,x) for (i,j,x) in nuc if i1<=i<i2 and j1<=j<j2])

def dim(sm):
    (n,m,nuc)=sm
    return (n,m)

def nzpos(sm):
    (n,m,nuc)=sm
    return [(i,j) for (i,j,x) in nuc]

def ocorr(i,j,nuc):
    aux=[k for k in range(len(nuc)) if nuc[k][0:2]==(i,j)]
    if aux==[]:
        return (False,0)
    else:
        return (True,aux[0])

def val(sm,i,j):
    (n,m,nuc)=sm
    (b,k)=ocorr(i,j,nuc)
    if b:
        return nuc[k][2]
    else:
        return 0

def add(sm,i,j,x):
    if x==0:
        return sm
    else:
        (n,m,nuc)=sm
        (b,k)=ocorr(i,j,nuc)
        nnuc=nuc[:]
        if not(b):
            nnuc=nnuc+[(i,j,x)]
        elif x+nuc[k][2]!=0:
            nnuc[k]=(i,j,x+nuc[k][2])
        elif x+nuc[k][2]==0:
            nnuc=nnuc[:k]+nnuc[k+1:]
        return (n,m,nnuc)
```

b) Desenvolva eficientemente, sobre a camada de abstracção obtida acima e assegurando a independência da implementação, as seguintes funções:

b1) `mapeia`, que recebendo uma função `f` e uma matriz esparsa `sm` devolve a matriz esparsa que resulta de aplicar a função `f` em cada uma das entradas de `sm`;

Resolução:

```
def mapeia(f,sm):
    (n,m)=dim(sm)
    z=f(0)
    res=const(n,m,z)
    pos=nzpos(sm)
    for (i,j) in pos:
        x=f(val(sm,i,j))
        if x!=z:
            res=add(res,i,j,x-z)
    return res
```

b2) `prodm`, que recebendo uma matriz esparsa `sm` e uma matriz (usual) `mat` (com tantas linhas quantas as colunas de `sm`) devolve a matriz esparsa correspondente ao produto das duas.

Resolução:

```
def prodm(sm,mat):
    (n,m)=dim(sm)
    assert m==len(mat)
    res=const(n,len(mat[0]),0)
    pos=nzpos(sm)
    for (i,j) in pos:
        x=val(sm,i,j)
        for k in range(len(mat[0])):
            if mat[j][k]!=0:
                res=add(res,i,k,x*mat[j][k])
    return res
```

Grupo III (4 valores)

Neste exercício não pode usar recursão, ciclos ou atribuições. Pode usar, sem necessitar de os definir, os combinadores `map`, `reduce`, `any`, `all`, `filter`, `nest`, `fixedpoint`, bem como definições `lambda` e por compreensão.

Implemente funcionalmente em *Python* o algoritmo de ordenação por inserção binária, que procede tal como o algoritmo usual de ordenação por inserção, mas em que a posição de inserção de cada elemento na sublista já ordenada é determinada por pesquisa binária.

Resolução:

```
def insere(u,x):
    def isola(int):
        if int[0]==int[1]:
            return int
        else:
            if x>u[sum(int)//2]:
                return (1+sum(int)//2,int[1])
            else:
                return (int[0],sum(int)//2)

    u.insert(fixedpoint(isola,(0,len(u)))[1],x)
    return u

def ordenacao(w):
    return reduce(insere,w,[])
```

Grupo IV (3 valores)

Considere o seguinte programa imperativo PROG.

```
i=0
b=False
while not(b) and i<len(w):
    a=a-w[i]
    i=i+1
    if a==0:
        b=True
```

Demonstre que é válida a asserção

$$\{\texttt{a} == \texttt{X}\}\ \texttt{PROG}\ \{\texttt{not(b) or} \sum_{0 \leq \texttt{j} < \texttt{i}} \texttt{w[j]} == \texttt{X}\}.$$

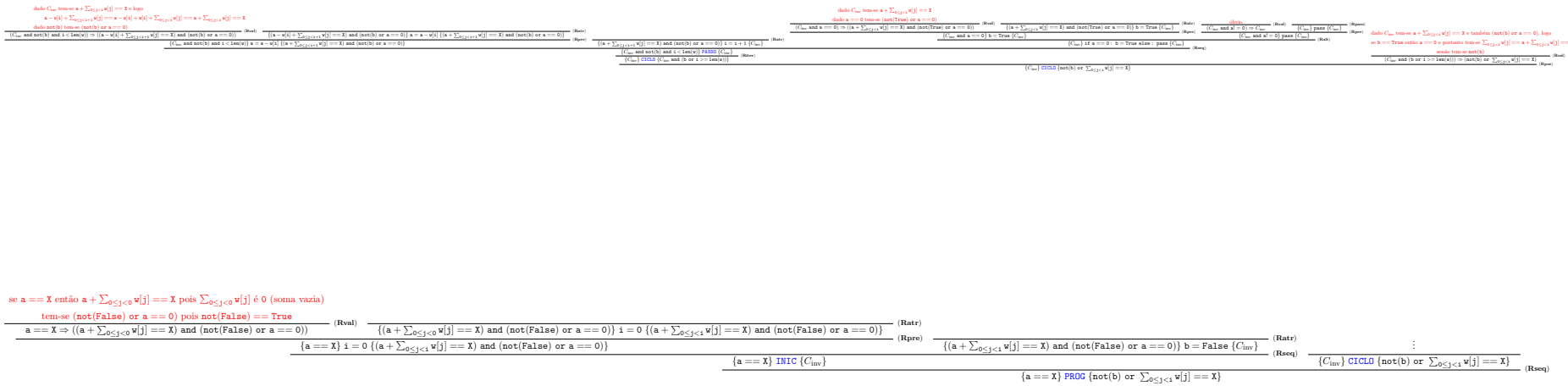
Resolução: Considera-se a estrutura de ciclo inicializado

$$\texttt{PROG} \begin{cases} \left.\begin{array}{l} \texttt{i} = 0 \\ \texttt{b} = \texttt{False} \end{array}\right\} \texttt{INIC} \\ \texttt{while not(b) and i} < \texttt{len(w)}: \left\{\begin{array}{l} \texttt{a} = \texttt{a} - \texttt{w[i]} \\ \texttt{i} = \texttt{i} + 1 \\ \texttt{if a} == 0: \\ \quad \texttt{b} = \texttt{True} \end{array}\right. \texttt{PASSO} \Big\} \texttt{CICLO} \end{cases}$$

e a condição invariante do ciclo

$$C_{\text{inv}} \equiv (\texttt{a} + \sum_{0 \leq \texttt{j} < \texttt{i}} \texttt{w[j]} == \texttt{X})\ \texttt{and}\ (\texttt{not(b) or a} == 0).$$

A demonstração da asserção segue abaixo, onde surgem a vermelho justificações para a validade das condições Booleanas.


dado C_{inv} tem-se $a + \sum_{0\le j<i} w[j] == X$ e logo
$a - w[i] + \sum_{0\le j<i+1} w[j] == a - w[i] + w[i] + \sum_{0\le j<i} w[j] == a + \sum_{0\le j<i} w[j] == X$
dado not(b) tem-se (not(b) or a == 0)
$(C_{\text{inv}}$ and not(b) and $i < \text{len}(w)) \Rightarrow [(a - w[i] + \sum_{0\le j<i+1} w[j] == X)$ and (not(b) or a == 0))
(Rval)
$[(a - w[i] + \sum_{0\le j<i+1} w[j] == X)$ and (not(b) or a == 0)] a = a − w[i] $[(a + \sum_{0\le j<i+1} w[j] == X)$ and (not(b) or a == 0)]
(Ratr)
$(C_{\text{inv}}$ and not(b) and i < len(w)) a = a − w[i] $[(a + \sum_{0\le j<i+1} w[j] == X)$ and (not(b) or a == 0)]
(Rpre)
$[(a + \sum_{0\le j<i+1} w[j] == X)$ and (not(b) or a == 0)] i = i + 1 $\{C_{\text{inv}}\}$
(Ratr)
$(C_{\text{inv}}$ and not(b) and i < len(w)) PASSE $\{C_{\text{inv}}\}$
(Ritw)
$\{C_{\text{inv}}\}$ CICLO $(C_{\text{inv}}$ and (b or i >= len(w)))
dado C_{inv} tem-se $a + \sum_{0\le j<i} w[j] == X$
dado a == 0 tem-se (not(True) or a == 0)
$(C_{\text{inv}}$ and a == 0) ⇒ $((a + \sum_{0\le j<i} w[j] == X)$ and (not(True) or a == 0))
(Rval)
$((a + \sum_{0\le j<i} w[j] == X)$ and (not(True) or a == 0)] b = True $\{C_{\text{inv}}\}$
(Ratr)
$(C_{\text{inv}}$ and a == 0] b = True $\{C_{\text{inv}}\}$
(Rpre)
$\{C_{\text{inv}}\}$ if a == 0 : b = True else : pass $\{C_{\text{inv}}\}$
(Rseq)
dado
$(C_{\text{inv}}$ and a' = 0) ⇒ C_{inv}
(Rval)
$\{C_{\text{inv}}\}$ pass $\{C_{\text{inv}}\}$
(Rpass)
$(C_{\text{inv}}$ and a' = 0] pass $\{C_{\text{inv}}\}$
(Ratr)
$\{C_{\text{inv}}\}$ CICLO (not(b) or $\sum_{0\le j<i} w[j] == X$)
dado C_{inv} tem-se $a + \sum_{0\le j<i} w[j] == X$ e também (not(b) or a == 0), logo
se b == True então a == 0 e portanto tem-se $\sum_{0\le j<i} w[j] == a + \sum_{0\le j<i} w[j] == X$
então tem-se not(b)
$(C_{\text{inv}}$ and (b or i >= len(w))) ⇒ (not(b) or $\sum_{0\le j<i} w[j] == X$)
(Rval)
(Rpass)

se a == X então a + $\sum_{0\le j<0} w[j] == X$ pois $\sum_{0\le j<0} w[j]$ é 0 (soma vazia)
tem-se (not(False) or a == 0) pois not(False) == True
a == X ⇒ ((a + $\sum_{0\le j<0} w[j] == X$) and (not(False) or a == 0))
(Rval)
{(a + $\sum_{0\le j<0} w[j] == X$) and (not(False) or a == 0)} i = 0 {(a + $\sum_{0\le j<i} w[j] == X$) and (not(False) or a == 0)}
(Ratr)
{a == X} i = 0 {(a + $\sum_{0\le j<i} w[j] == X$) and (not(False) or a == 0)}
(Rpre)
{(a + $\sum_{0\le j<i} w[j] == X$) and (not(False) or a == 0)} b = False $\{C_{\text{inv}}\}$
(Ratr)
{a == X} INIC $\{C_{\text{inv}}\}$
(Rseq)
$\{C_{\text{inv}}\}$ CICLO {not(b) or $\sum_{0\le j<i} w[j] == X$}
{a == X} PROG {not(b) or $\sum_{0\le j<i} w[j] == X$}
(Rseq)