Instituto Superior Técnico

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

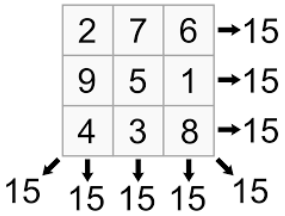
Elementos de Programação

17 de Janeiro de 2020 Exame 1 Duração: 2h30

Grupo I (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 matriz quadrada diz-se *quase-mágica* se as somas de cada uma das suas linhas, colunas e diagonais principais tiver exactamente o mesmo valor (15, no exemplo apresentado).



Defina imperativamente em *Python* usando dois ciclos encaixados uma função `qmag` que dada uma matriz devolve `True` se a matriz for quase-mágica, e `False` caso contrário.

Resolução:

```
def qmag(m):
    d1=0
    d2=0
    start=True
    ok=True
    i=0
    while ok and i<len(m):
        d1=d1+m[i][i]
        d2=d2+m[i][len(m)-1-i]
        lin=0
        col=0
        for j in range(len(m)):
            lin=lin+m[i][j]
            col=col+m[j][i]
        if start:
            start=False
            const=lin
        ok=(lin==col==const)
        i=i+1
    return ok and (d1==d2==const)
```

Grupo II (5+3 valores)

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

- `zeros(n)`: lista esparsa com `n` elementos, todos zero;
- `junta(sp1,sp2)`: lista esparsa que resulta de concatenar as listas esparsas `sp1` e `sp2`;
- `corta(sp,i,j)`: lista esparsa que resulta de tomar apenas os valores na posições entre `i` e `j` (com `i`$\leq$`j` e excluindo `j`) da lista esparsa `sp`;
- `compr(sp)`: número de elementos da lista esparsa `sp`;
- `nzpos(sp)`: lista das posições da lista esparsa `sp` cujos valores são diferentes de zero;
- `val(sp,i)`: valor que está na posição `i` da lista esparsa `sp`;
- `atr(sp,i,x)`: lista esparsa que resulta de atribuir o valor `x` à posição `i` da lista esparsa `sp`.

a) Em *Python*, pretende-se representar uma lista esparsa como um par `(n,nuc)` onde `n` é o número de elementos da lista, e `nuc` é uma lista de pares da forma `[...,(i,x),...]`, para cada posição `i` da lista esparsa cujo valor associado `x` seja diferente de zero, ordenada pelas posições.

Apresente implementações eficientes para as operações (valoriza-se que a pesquisa da posição correcta em `nuc` nas operações `val` e `atr` seja obtida por pesquisa binária).

Resolução:

```
def zeros(n):
    return (n,[])

def junta(sp1,sp2):
    (n1,nuc1)=sp1
    (n2,nuc2)=sp2
    return (n1+n2,nuc1+[(n1+i,x) for (i,x) in nuc2])

def corta(sp,i,j):
    (n,nuc)=sp
    return (j-i,[(k-i,x) for (k,x) in nuc if i<=k<j])

def compr(sp):
    (n,nuc)=sp
    return n

def nzpos(w):
    (n,nuc)=w
    return [i for (i,y) in nuc]

def posnuc(i,nuc):
    if nuc==[]:
        return (False,0,0)
    else:
        found=False
        a=0
        b=len(nuc)-1
        while not(found) and a<=b:
            m=(a+b)//2
            if i==nuc[m][0]:
                found=True
            elif i<nuc[m][0]:
                b=m-1
            else:
                a=m+1
        return (found,m,b+1)

def val(sp,i):
    (n,nuc)=sp
    (existe,pos,npos)=posnuc(i,nuc)
    if existe:
        return nuc[pos][1]
    else:
        return 0

def atr(sp,i,x):
    (n,nuc)=sp
    nnuc=nuc[:]
    (existe,pos,npos)=posnuc(i,nuc)
    if existe and x!=0:
        nnuc[pos]=(i,x)
    elif existe and x==0:
        nnuc.pop(pos)
    elif not(existe) and x!=0:
        nnuc.insert(npos,(i,x))
    return (n,nnuc)
```

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

b1) `pertence`, que recebendo um valor `x` e uma lista esparsa `sp` devolve `True` se `x` ocorre em `sp`, e `False` caso contrário;

Resolução:

```
def pertence(x,sp):
    if x==0:
        return compr(sp)!=len(nzpos(sp))
    else:
        return x in [val(sp,i) for i in nzpos(sp)]
```

b2) `prod`, que recebendo duas listas esparsas `sp1,sp2` com o mesmo número de elementos devolve a lista esparsa obtida multiplicando os valores das duas listas em posições correspondentes.

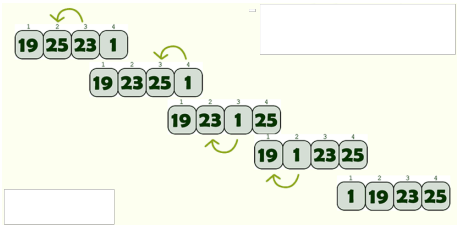
Resolução:

```
def prod(sp1,sp2):
    w1=nzpos(sp1)
    w2=nzpos(sp2)
    sp=zeros(compr(sp1))
    i1=0
    i2=0
    while i1<len(w1) and i2<len(w2):
        if w1[i1]<w2[i2]:
            i1=i1+1
        elif w1[i1]>w2[i2]:
            i2=i2+1
        else:
            sp=atr(sp,w1[i1],val(sp1,w1[i1])*val(sp2,w1[i1]))
            i1=i1+1
            i2=i2+1
    return sp
```

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.

No algoritmo *bubble sort* uma lista de valores numéricos é ordenada (por ordem crescente) por troca sucessiva de valores $x > y$ que surjam na lista em posições consecutivas. Veja-se o exemplo.



Implemente funcionalmente em *Python* o algoritmo *bubble sort*.

Resolução:

```
def bubblesort(w):
    def troca(u,i):
        if u[i]>u[i+1]:
            return u[:i]+[u[i+1],u[i]]+u[i+2:]
        else:
            return u

    def bubble(u):
        return reduce(troca,range(len(u)-1),u)

    return fixedpoint(bubble,w)
```

Grupo IV (3 valores)

Considere o seguinte programa imperativo PROG.

```
b=True
i=0
while b and i!=len(w):
    if w[i]>=m:
        b=False
    i=i+1
```

Demonstre que é válida a asserção

$$\{\texttt{True}\}\ \texttt{PROG}\ \{\texttt{b} == \forall_{0\leq \texttt{j}<\texttt{len(w)}}\texttt{w[j]} < \texttt{m}\}.$$

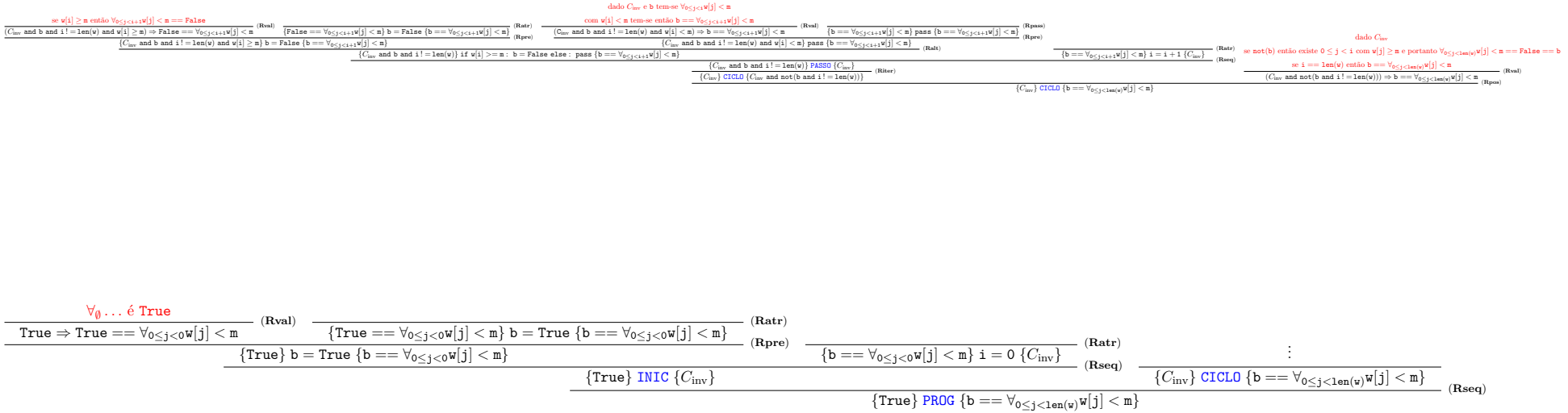
Resolução: Considera-se a estrutura de ciclo inicializado

$$\texttt{PROG}\left\{\begin{array}{l}\left.\begin{array}{l}\texttt{b} = \texttt{True}\\ \texttt{i} = 0\end{array}\right\}\texttt{INIC}\\ \left.\begin{array}{l}\texttt{while b and i} != \texttt{len(w)}:\\ \quad\left.\begin{array}{l}\texttt{if w[i]} >= \texttt{m}:\\ \quad\texttt{b} = \texttt{False}\\ \texttt{i} = \texttt{i} + 1\end{array}\right\}\texttt{PASSO}\end{array}\right\}\texttt{CICLO}\end{array}\right.$$

e a condição invariante do ciclo

$$C_{\text{inv}} \equiv (\texttt{b} == \forall_{0\leq \texttt{j}<\texttt{i}}\texttt{w[j]} < \texttt{m}).$$

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



$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall_{\emptyset}\ldots\text{ é True}}{\texttt{True} \Rightarrow \texttt{True} == \forall_{0\leq j<0} \texttt{w}[\texttt{j}] < \texttt{m}}\,(\textbf{Rval}) \quad \dfrac{}{\{\texttt{True} == \forall_{0\leq j<0} \texttt{w}[\texttt{j}] < \texttt{m}\}\ \texttt{b} = \texttt{True}\ \{\texttt{b} == \forall_{0\leq j<0} \texttt{w}[\texttt{j}] < \texttt{m}\}}\,(\textbf{Ratr})}{\{\texttt{True}\}\ \texttt{b} = \texttt{True}\ \{\texttt{b} == \forall_{0\leq j<0} \texttt{w}[\texttt{j}] < \texttt{m}\}}\,(\textbf{Rpre}) \quad \dfrac{}{\{\texttt{b} == \forall_{0\leq j<0} \texttt{w}[\texttt{j}] < \texttt{m}\}\ \texttt{i} = 0\ \{C_{\text{inv}}\}}\,(\textbf{Ratr})}{\{\texttt{True}\}\ \texttt{INIC}\ \{C_{\text{inv}}\}}\,(\textbf{Rseq}) \quad \dfrac{\vdots}{\{C_{\text{inv}}\}\ \texttt{CICLO}\ \{\texttt{b} == \forall_{0\leq j<\texttt{len(w)}} \texttt{w}[\texttt{j}] < \texttt{m}\}}}{\{\texttt{True}\}\ \texttt{PROG}\ \{\texttt{b} == \forall_{0\leq j<\texttt{len(w)}} \texttt{w}[\texttt{j}] < \texttt{m}\}}\,(\textbf{Rseq})$$