ISM MPSI/PCSI, Informatique
Année 2023/2024

Corrigé du concours blanc - Informatique

I Préliminaires

1 . On peut représenter une file de voitures par une liste `L` de booléens en considérant que `L[i]` est à `True` s'il y a une voiture en indice `i` et à `False` sinon avec $i \in [\![0 \,;\, n-1]\!]$ où n est la taille de `L`.

2 . On propose :

```
A=[False]*11
ind=[0,2,3,10]
for k in ind:
    A[k]=True
```

3 . On propose :

```
def occupe(L,i):
    return L[i]
```

4 . Pour chacune des n cases possibles, on a 2 choix possibles d'où 2^n files différentes de longueur n.

II Déplacement de voitures dans la file

5 . On propose :

```
def avancer(L,b):
    res=[b]+L
    return res[:len(L)]
```

6 . L'instruction `avancer(avancer(A,False),True)` renvoie `[True, False, True, False, True, True, False, False, False, False, False]`

7 . On propose :

```
def avancer_fin(L,m):
    return L[:m]+avancer(L[m:],False)
```

8 . On propose :

```
def avancer_debut(L,b,m):
    return [b]+L[:m]+L[m+1:]
```

9 . On propose :

```
def avancer_debut_bloque(L,b,m):
    for i in range(m):
        if not occupe(L,m-1-i):
            return avancer_debut(L,b,m-1-i)
    return list(L)
```

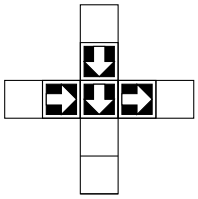
III Une étape de simulation à deux files

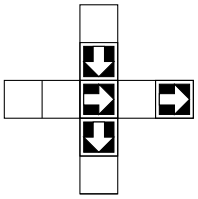
10 . On propose :

```
def avancer_files(L1,b1,L2,b2):
    m=len(L1)//2
    R1=avancer(L1,b1)
    R2=avancer_fin(L2,m)
    if occupe(R1,m) # si une voiture de L1 au croisement
        R2=avancer_debut_bloque(R2,b2,m)
    else : # si pas de voiture de L1 au croisement
        R2=avancer_debut(R2,b2,m)
    return [R1,R2]
```

11 . Avec les listes `D` et `E` considérées, l'appel `avancer_files(D,False,E,False)` renvoie

`[[False, False, True, False, True], [False, True, False, True, False]]`





IV Transitions

12 . Si toutes les cases de `L1` sont occupées à chaque étape, alors la file de voitures `L2` est indéfiniment bloquée.

13 . Il faut 5 étapes pour que les voitures de `L1`, prioritaires, franchissent le croisement. Ensuite, les deux files avancent normalement avec arrivée de nouvelles voitures dans `L1` et sortie de celles qui arrivent au bout sur 4 étapes. Il faut donc au minimum 9 étapes pour cette transition.

14 . La file `L1` étant prioritaire sur la file `L2`, les voitures de la file `L2` ne peuvent avancer qu'après le passage des voitures de la file `L1` et ne peuvent donc arriver en fin de file simultanément avec celles de la file `L1`.

V Atteignabilité

15 . On propose un tri rapide pas en place :

```
def tri(L):
    if L==[]:
        return []
    else:
        pivot,L1,L2=L[0],[],[]
        for x in L[1:]:
            if x<pivot:
                L1.append(x)
            else:
                L2.append(x)
        return tri(L1)+[pivot]+tri(L2)
```

16 . On propose :

```
def elim_double(L):
    res=[L[0]]
    n=len(L)
    for k in range(1,n):
        if L[k]!=L[k-1]:
            res.append(L[k])
    return res
```

17 . L'appel `doublons([1, 1, 2, 2, 3, 3, 3, 5])` renvoie `[1, 2, 3, 5]`.

18 . Cette fonction n'est pas adaptée pour une liste non triée puisque au cours des récursions, on compare toujours des éléments consécutifs avec le test `L[0]!=L[1]`.

19 . La fonction `recherche` renvoie un booléen. Les variables `but` et `espace` sont des listes de deux listes de même longueur impaire correspondant à des files. La fonction `successeurs` renvoie une liste de liste de deux listes de même longueur impaire correspondant à des files.

20 . Pour une file de taille n, on a 2^n files différentes possibles. Ainsi, le nombre de configurations est majoré par 2×2^n et la variable `2**(n+1)-len(espace)` est donc un variant de boucle puisque la taille de `espace` croît strictement. L'exhibition d'un variant prouve la terminaison de la boucle.

21 . Il suffit de déplacer l'instruction après le bloc `if ...` : puisque celui-ci peut provoquer une sortie de boucle auquel cas l'affectation de `stop` est superflue. On saisit :

```
def recherche(but, init):
    espace = [init]
    stop = False
    while not stop:
        ancien = espace
        espace = espace + successeurs(espace)
        espace.sort() # permet de trier espace par ordre croissant
        espace = elim_double(espace)
        if but in espace:
            return True
        stop = (ancien==espace)
    return False
```

VI Amélioration

22 . On propose une recherche dichotomique :

```
def in_triee(elt,L):
    deb,fin,trouve=0,len(L)-1,False
    while not trouve and deb<=fin:
        milieu=(deb+fin)//2
        if L[milieu]==elt:
            trouve=True
        elif L[milieu]>elt:
            fin=milieu-1
        else:
            deb=milieu+1
    return trouve
```

23 . On propose une implémentation de l'algorithme de Horner :

```
def versEntier(L):
    res=0
    for x in L:
       res=2*res+x
    return res
```

24 . La taille de la liste doit être supérieure ou égale à la taille de l'écriture binaire de l'entier non nul n qui est égale à $\lfloor \log_2(n) \rfloor + 1$. On saisit :

```
def versFile(n,taille):
    res=[False]*taille
    a,ind=n,taille-1
    while a>0:
        res[ind]=(a%2==1)
        a=a//2
        ind-=1
    return res
```

25 . On saisit :

```
def recherche(but,init):
    ...
    while not stop:
        ...
        if in_triee(but_int,espace):
            return True
        stop=(ancien==espace)
    return False

def successeurs(L,taille,N):
    res=[]
    for x in L:
        L1,L2=versFile(x//N,taille),versFile(x%N,taille)
        for flag1 in [False,True]:
            for flag2 in [False,True]:
                files=avancer_files(L1,flag1,L2,flag2)
                res.append(versEntier(files[0]+files[1]))
    return res
```