DS 2
Informatique
Algorithmie
Extrait Concours Mines-Ponts 2017
Correction

Simon Dauguet
simon.dauguet@gmail.com

Mercredi 27 Novembre 2024

Partie I : Préliminaires

1. On peut représenter une file de voiture par une liste de booléens, le `True` représentant la présence d'une voiture et `False` représentant une absence de voiture. Par exemple `[True,False,True]` pourrait représenter une file de 3 places avec une voiture au début et à la fin, et un espace vide entre les deux.

2. On peut écrire

```
A=[True, False, True, True]+[False]*6+[True]
```

pour représenter la situation de la figue 1a.

3.
```
def occupe(L:list, i:int) -> bool :
    return(L[i])
```

Cette fonction renvoie `True` s'il y a `True` en position i dans la liste L et donc s'il y a une voiture en position i dans la file. Et inversement avec l'absence de voiture.

4. Pour une file de longueur n, à chaque place, on a 2 situations possibles seulement (une voiture, ou pas de voiture). On a donc 2^n listes possibles.

5.
```
def egal(L1:list, L2:list) :
    return(L1==L2)
```

Cette fonction renvoie le booléen correspondant au fait que les deux listes sont égales. Si elles diffèrent d'une seule position au moins, elle renverra `False`.

6. Comme indiqué dans la définition de la fonction `egal` dans l'énoncé de la question précédent, cette fonction renvoie un booléen.

Partie II : Déplacement de voitures dans la file

7. l'appel de `avancer(A,False)` renvoie la liste :

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

Et donc l'appel de `avancer(avancer(A,False)True)` renvoie la liste :

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

8.
```
def avancer_fin(L:list, m:int) -> list :
    return(L[:m]+avancer(L[m:],False))
```

9.
```
def avancer_debut(L:list, b:bool, m:int) -> list:
    return(avancer(L[:m+1],b)+L[m+1:])
```

10.
```
def avancer_debut_bloque(L:list, b:bool, m:int) -> list :
    LL=L[m:]
    for k in range(1,m) :
        if L[m-k]==False :
            LL=[L[m-k-1]]+LL
    return(LL)
```

Partie III : Une étape de simulation à deux files

11.
```
def avancer_file(L1:list, b1:bool, L2:list, b2:bool) -> list :
    m=(len(L1)-1)//2
    R2=avancer_fin(L2,m)
    R1=avancer(L1,b1) # Les voitures de L1 ne peuvent pas être bloquées
    if occupe(R1,m) :
        R2=avancer_fin_bloque(R2,b2,m)
    else :
        R2=avancer_debut(R2,b2,m)
    return([R1,R2])
```

12. L'appel `avancer_file(D,False,E,False)` renvoie :

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

Partie IV. Transitions

13. Considérons la situation suivante :

- La file `L1` est pleine.
- À chaque étape de la simulation, on ajoute une nouvelle voiture à `L1`.
- Une voiture est sur la case $m-1$ de la file `L2`.

La voiture en $m-1$ de `L2` ne pourra jamais avancé car la place m de `L1` ne sera jamais libérée. Et donc la file `L2` est indéfiniment bloquée.

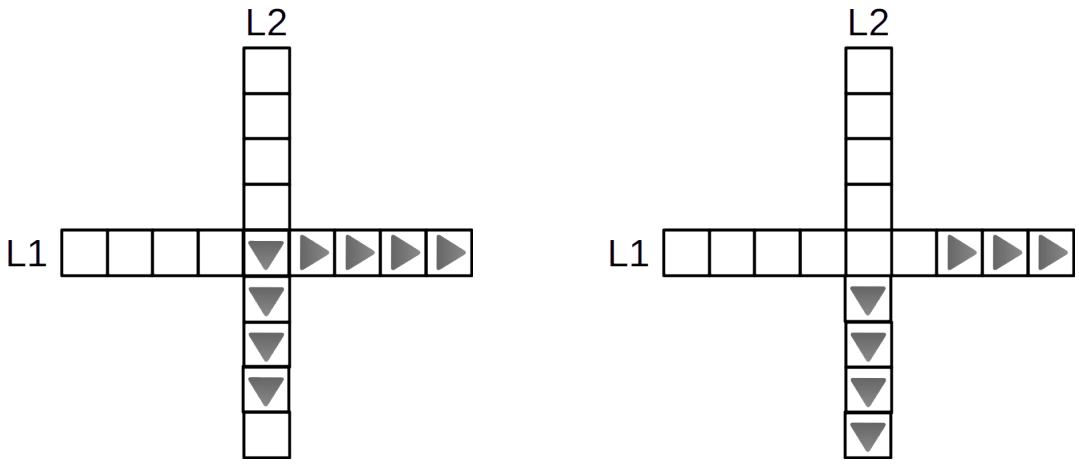
14. À cause de la priorité à droite, les voitures de `L2` ne peuvent se déplacer qu'une fois que toutes les voitures de `L1` ont passées le croisement. Il faut donc déjà attendre 4 étapes pour que les voitures de `L1` passent le croisement. Il ne faut pas rajouter de voitures supplémentaires dans `L1`.

Puis, à l'étape 5, le croisement est libéré par les voitures de `L1` et celles de `L2` peuvent commencer à avancer. Il faut alors 5 étapes de plus pour que les 4 voitures de `L2` dépassent toutes le croisement.

En parallèle de l'avancée des voitures de `L2`, on introduit de nouvelles dans `L1`. Mais ça ne rajoute pas d'étapes supplémentaires.

Il faut donc un minimum de 9 étapes pour passer de la configuration 4a à la configuration 4b.

15. La configuration 4c est impossible. En effet, à cause de la priorité à droite, les voitures de `L2` ne peuvent passer que si toutes les voitures de `L1` sont déjà passées. Or, à chaque étape, les voitures des deux files avances en même temps. L'étape précédente de la configuration 4c, reviendrais donc à avoir la dernière voiture de `L2` au croisement. Et toutes les voitures de `L1` déjà passées. Mais dans ce cas, à l'étape suivante (donc théoriquement la configuration 4c), les voitures de `L1` auraient continuées à avancer et il y aurait une place vide entre le croisement et la dernière voiture de `L1`. Ce qui n'est pas le cas.



(a) (b)

Partie V : Atteignabilité

16.
```
def elim_double(L:list) -> list :
    R=[L[0]]
    for k in L :
        if k not in R :
            R=R+[k]
    return(R)
```

17. On présente les choses sous forme d'un tableau :

Objets	`recherche`	`but`	`espace`	`successeurs`
Types	booléen	liste de liste de booléens	liste de listes de listes de booléen	liste de listes de listes de booléens

18.
```
def versEntier(L:list) -> int  :
    N=0
    for k in range(0,len(L)) :
        N=N+int(L[k])*2**(len(L)-k-1)
    return(N)
```

19. `taille` doit avoir au moins la valeur du nombre de chiffre du codage de `n` en base 2. Autrement dit, `taille` doit prendre au moins la valeur $\lfloor \log_2(n) \rfloor + 1$. À la ligne 4, il faut écrire "`i >= 0`" ou "`n != 0`".

20. On modifie la fonction `recherche` de la manière suivante :

```
def recherche(but:list, init:list) -> object :
    if egal(init,but) :
        return(0)
    n=0
    espace = [init]
    stop = False
    while not stop :
        n = n+1
        ancien = espace
        espace = espace + successeur(espace)
        espace.sort()
        espace = elim_double(espace)
        stop = egal(ancien,espace)
        if but in espace :
            return(n)
    return(False)
```