1 Programmation

2 Étude de trafic routier

1)

```
A = [False for i in range (11)]
A[0] = True
A[2] = True
A[3] = True
A[10] = True
```

2)

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

3)
Il y a deux choix possibles pour chaque case (*True* ou *False*), chaque case est indépendante du reste, donc pour n cases, il y a 2^n possibilités.

4) Bien faire attention de vérifier la différence de longueur.

```
def egal(L1,L2) :
    if len(L1) != len(L2) :
        return False
    for i in range (len(L2)) :
        if L1[i] != L2[i] :
            return False
    return True
```

5)
Dans le pire cas, on effectue la boucle un nombre égal à len(L1) fois. Ce qui est à l'intérieur de la boucle est en complexité O(1) et ce qui est en dehors est en O(1). Donc la complexité du tout est linéaire en len(L1), soit O(len(L1)).

6)
avancer(A,False) renvoie la liste B de l'exemple, on applique ensuite avancer en ajoutant une voiture à gauche et on obtient `[1,0,1,0,1,1,0,0,0,0,0]`.

7)
Le sujet est beaucoup plus simple si on n'oublie pas d'utiliser avancer et que l'on utilise les *slicings* de liste.

```
def avancer_fin(L,m) :
    L2 = L[0:len(L)]
    L2[m:len(L)] = avancer(L[m:len(L)], False)
    return L2
```

8)

```
def avancer_debut(L,b,m) :
    L2 = L[0:len(L)]
    if m >0:
        L2[m]=L2[m-1]
    L2[0:m-1] = avancer(L[0:m-1],b)
    return L
```

9)

```
def avancer_debut_bloque(L,b,m):
    if m = 0 :
        return L
    else :
        if occupe(L,m-1) :
            avancer_debut_bloque(L,b,m-1)
        else :
            return avancer_debut(L,b,m-1)
```

10) m est ici l'indice de la case du milieu. Donc $m = len(L)//2$, parce que l'on commence à 0 dans une liste. On peut vérifier sur l'exemple si ça marche.

11)

```
def avancer_files(L1,b1,L2,b2):
    m = len(L1)//2
    R1 = avancer(L1,b)
    R2 = avancer_fin(L2,m)
    if R1[m]:
        R2 = avancer_debut_bloque(R2,b2,m)
    else:
        R2 = avancer_debut(R2,b2,m)
return [R1 , R2]
```

12)
L'appel avancer_files(D,False,E,False) renvoie [[False,False,True,False,True],[False,True,False,True,False]].

13)

Si une voiture est toujours introduite sur L1, et que toutes les cases sont pleines avant l'intersection dans L1, la première voiture avant l'intersection dans L2 attendra toujours.

14)

L1 est prioritaire, donc la voiture la plus à droite va s'engager, et comme les autres voitures vont suivre, les 4 voitures de L1 vont passer avant que celle de L2 puisse passer, sachant que la 4ème voiture de L2 doit passer l'intersection dans la configuration b, il faut que les 4 voitures de L1 passent, donc 4 étapes.

À la 5ème étape, elles laissent libre l'intersection et la première voiture de L2 s'engage, 4 étapes plus tard la 4ème voiture s'engage, et à l'étape suivante elle laisse l'intersection vide.

À chacune des 4 étapes précédentes on a fait entrer une voiture dans L1. Il faut donc 9 étapes au minimum, car moins d'étapes strictement fait qu'il restera des voitures de L2 en fin de file.

15)

La configuration 4c n'est pas atteignable depuis une autre puisqu'à l'étape précédente, il devrait y avoir deux voitures sur la case d'intersection, ce qui est impossible, les voitures y passant une par une.

16)

```
def doublons(L) :
    rep = [L[0]]
    for i in range (1,len(L)) :
        if rep[-1] != L[i] :
            rep.append(L[i])
    return rep
```