Concours Blanc : TP d'Informatique

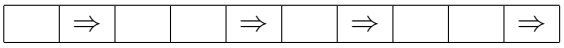
PTSI Lycée Eiffel

8 juin 2018

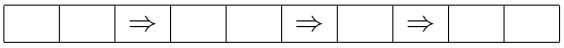
Dans ce TP, nous allons nous intéresser à des modèles élémentaires de gestion de traffic routier. Dans un premier temps, nous nous intéresserons à une « route » constituée d'une seule file dans laquelle les voitures peuvent prendre des emplacements prédéterminés, puis nous tenterons de modéliser une intersection de deux telles routes et d'étendre ces principes à des cas plus complexes.

I. Modélisation d'une route à une seule file.

La route sera simplement modélisée en Python par une liste de longueur n, chacun des emplacements de la liste pouvant contenir une voiture ou être vide (ce qu'on modélisera simplement par les valeurs 1 et 0). On suppose que les voitures se déplacent toutes à la même vitesse, d'une unité vers la droite à chaque fois que passe une unité de temps. Une voiture qui est située à droite de la liste disparaitra au prochain déplacement, et une nouvelle voiture peut apparaitre à gauche de la liste à chaque déplacement. Ainsi, pour $n = 10$, une route où les emplacements 1, 4, 6 et 9 sont occupés (en cohérence avec les conventions de Python, les emplacements sont numérotés de 0 à 9) pourra être représentée comme ceci (les voitures étant indiquées sous forme de flèches) :



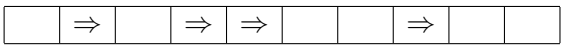
Cette route sera représentée en Python par la liste $[0, 1, 0, 0, 1, 0, 1, 0, 0, 1]$. Après un déplacement, en supposant qu'aucune voiture n'apparait à gauche, la route ressemblera à ceci :



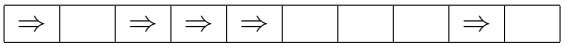
Après un deuxième déplacement où une voiture apparait à gauche, la route sera désormais :

1. Donner une commande Python permettant de créer une route de longueur n sur laquelle ne se trouvent pas de voitures.
2. Soit L une liste représentant une route de longueur n et i un entier naturel. Écrire une fonction Python **occupee(L,i)** qui renvoie True si la case d'indice i de la liste est occupée par une voiture, et False sinon. Dans le cas où $i \geqslant n$, on renverra un message d'erreur indiquant que la route n'est pas assez longue.
3. Écrire une fonction Python **nbvoitures(L)** qui renvoie le nombre de voitures présentes sur une route L (on identifiera désormais les routes et les listes qui les représentent).
4. Écrire une fonction Python **egal(L1,L2)** qui détermine si deux routes sont identiques (même longueur, même nombre de voitures situées aux mêmes emplacements). Cette fonction renverra True ou False.
5. Écrire une fonction Python **deplace(L,a)** qui effectue un déplacement sur la route L, en ajoutant une voiture à gauche de la route si $a = 1$ (et en laissant la case de gauche vide si $a = 0$). Cette fonction renverra une liste de même longueur que L.

6. Écrire une fonction Python **deplacemultiple(L,k,l)** qui effectue une suite de k déplacements de la route L, en ajoutant ou non de nouvelles voitures à chaque étape selon les valeurs prises par la liste l, qui est une liste de longueur k ne contenant que des 0 et des 1. On considèdera que l'élément d'indice 0 de la liste l gère l'ajout de voiture du premier déplacement, celui d'indice 1 correspond au deuxième déplacement etc. Ainsi, la commande **deplacemultiple([0,1,0,0,1,0,1,0,0,1],2,[0,1])** devrait renvoyer la liste [1,0,0,1,0,0,1,0,1,0] (cf exemples donnés ci-dessus).
7. On suppose pour cette question que la voiture située en position b dans la route L est en panne. Lors d'un déplacement, cette voiture ne bougera donc pas, les voitures situées à sa droite avanceront normalement, et les voitures situées à sa gauche avanceront si et seulement si l'emplacement situé à leur droite s'est libéré (ainsi, une voiture située juste à gauche de la voiture en panne restera par exemple elle aussi immobile). Ainsi, si on suppose la voiture en position 4 en panne sur la route suivante :



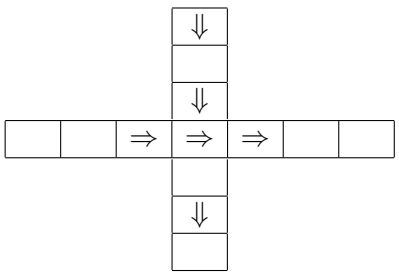
un déplacement sur cette route avec ajout de voiture à gauche donnera la situation suivante :



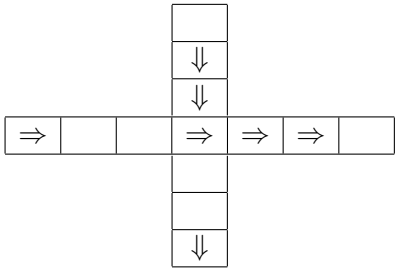
Écrire une fonction Python **deplacepanne(L,b,a)** qui effectue un déplacement sur la route L, avec une voiture en panne en position b, et un ajout de voiture à gauche si $a = 1$.

II. Intersection de deux routes à une seule file.

On considère désormais deux routes $L1$ et $L2$ de même longueur impaire qui se croisent au niveau de leur case centrale (ainsi, si les deux routes sont de longueur 15, la case 7 de chacune des deux routes est en fait la même). La route $L1$ est supposée prioritaire sur la route $L2$. Les voitures restent toujours sur leur route, et la case commune ne peut bien sûr contenir qu'une seule voiture à la fois. Lors d'un déplacement, les deux routes vont se déplacer simultanément, mais si les cases précédant la case centrale sont occupées dans chacune des deux listes, c'est la voiture de la liste $L1$ (qui a la priorité) qui ira sur la case centrale, celle de la liste $L2$ jouant donc pour ce déplacement le rôle d'une voiture en panne. Ainsi, le croisement suivant ($L1$ de gauche à droite, $L2$ de haut en bas) :



après déplacement avec ajout d'un véhicule dans $L1$ mais pas dans $L2$, deviendra :



1. Écrire une fonction Python **deplacecroisement(L1,L2,a1,a2)** qui effectue le déplacement correspondant à un croisement, en ajoutant une voiture au début de la route $L1$ si $a1 = 1$ (et similairement pour $L2$).
2. Donner un exemple de situation sur un croisement de deux routes de longueur 7 qu'on ne peut pas atteindre à partir d'une situation initiale où les deux routes sont vides (on expliquera la raison de cette impossibilité). Les questions suivantes permettent d'écrire un programme effectuant la liste des situations atteignables à partir d'un croisement de routes initialement vides.
3. Écrire une fonction Python **elimdouble(L)** qui prend comme argument une liste L *triée dans l'ordre croissant*, et qui élimine les éléments apparaisant plusieurs fois dans la liste L. Ainsi, **elimdouble([1,1,3,3,3,,5,7,7])** doit renvoyer $[1, 3, 5, 7]$.
4. Écrire une fonction Python **successeurs(L1,L2)** qui renvoie les quatre situations possibles après un déplacement sur le croisement des routes $L1$ et $L2$, selon qu'on ajoute ou non une voiture dans $L1$ et dans $L2$. Pour simplifier l'écriture du programme suivant, on renverra le résultat sous forme d'une liste de quatre listes, chacune de ces listes étant obtenue en concaténant les deux routes $L1$ et $L2$ après le déplacement correspondant.
5. On souhaite écrire un programme **situationspossibles(n)** calculant toutes les situations possibles au croisement de deux routes de longueur n (où n est un entier impair), en partant de deux routes initialement vides. Pour celà, on respectera les étapes suivantes :
   - on crée une liste **situations** qui contient initialement un seul élément : la liste de longueur $2n$ ne contenant que des 0 (concaténation des deux routes vides)
   - à chaque nouvelle étape, on ajoute à la liste **situations** les successeurs de tous les croisements de routes déjà présents dans la liste (attention à bien séparer en deux routes distinctes au moment d'appliquer la fonction **successeurs**).
   - on trie la liste **situations** (la méthode **sort** en Python gère parfaitement le tri d'une liste elle-même constituée de listes d'entiers), puis on élimine les listes apparaissant plusieurs fois.
   - si la longueur de la liste **situations** n'a pas augmenté lors de la dernière étape effectuée, on s'arrête là.

   Écrire cet algorithme en Python, et déterminer le nombre total de situations atteignables pour $n = 7$ (attention à ne pas tester le programme sur de trop grandes valeurs de n, il est particulièrement peu subtil et donc très lent). On pourra comparer au nombre total de situations possibles si on oublie les règles de priorité.

III. Gestion de vitesses différentes.

On revient désormais au cas d'une seule route, mais sur laquelle les voitures peuvent avoir des vitesses variables. Une telle route sera représentée en Python par une liste d'entiers naturels, le 0 correspondant toujours à un emplacement vide, et l'entier $k > 0$ correspondant à une voiture de vitesse k, c'est-à-dire une voiture pouvant se déplacer de k cases à chaque déplacement.

1. On suppose dans cette question que les dépassements sont impossibles sur notre route. Une voiture de vitesse k se déplacera donc d'un maximum de k cases à chaque déplacement, en s'arrêtant dès qu'elle est bloquée par une voiture plus lente.

   Écrire une fonction Python **deplacevitesse(L,a)** qui effectue un déplacement sur la route L en suivant cette règle, en ajoutant une voiture à gauche de la route si $a > 0$ (dans ce cas, a est un entier naturel représentant la vitesse de la voiture à ajouter).
2. On suppose maintenant que les dépassements sont possibles, et qu'une voiture peut même en dépasser plusieurs en un seul déplacement (et une même voiture lente peut aussi se faire dépasser par plusieurs voitures lors d'un seul déplacement). Une voiture de vitesse $k > 1$ se

déplacera donc d'un nombre de cases maximal inférieur ou égal à k (on cherche la position libre la plus éloignée de sa position de départ, avec la limite de k cases pour un déplacement).

Écrire une fonction Python **deplacedepasse(L,a)** qui effectue un déplacement sur la route L en suivant cette règle.

3. On considère désormais une route constituée de deux files $L1$ et $L2$ où les déplacements s'effectuent en sens inverse l'une de l'autre :
   - sur la route $L2$, les voitures se déplacent *de droite à gauche* mais sont toutes de vitesse 1 (il peut bien sûr y avoir des emplacements vides).
   - sur la route $L1$, les voitures se déplacent de gauche à droite, et peuvent avoir une vitesse k quelconque. Elle peuvent se doubler comme à la question précédente, mais avec la condition supplémentaire suivante : une voiture ne peut se déplacer de la case c à la case c' en doublant une autre voiture que si les cases c à c' de la route $L2$ sont libres (la voiture va doubler en passant sur l'autre file). Ainsi, une voiture de vitesse 3 positionnée à la case d'indice 4 ne pourra dépasser une voiture de vitesse 1 positionnée en case 5 (et qui va donc avancer jusqu'à la case 6), que si la case $L1[7]$, mais aussi les cases $L2[4]$, $L2[5]$, $L2[6]$ et $L2[7]$ sont toutes libres.

   Pour simplifier, on considèrera lors d'un déplacement qu'on déplace d'abord toutes les voitures de la route $L1$, puis toutes celles de la route $L2$.

   Écrire une fonction Python **deplacedeuxfiles(L1,L2,a1,a2)** effectuant un déplacement suivant ces règles, en ajoutant ou non une voiture à gauche de $L1$ (de vitesse $a1$) et à droite de $L2$ en fonction des valeurs de $a1$ et $a2$.