ENSAE TD noté, vendredi 5 décembre 2014

Le programme construit au fur et à mesure des questions devra être imprimé à la fin du TD et rendu au chargé de TD. ***Il ne faut pas oublier de mentionner son nom sur chaque page.*** *Les réponses autres que des parties de programme seront insérées sous forme de commentaires.* ***Il faudra aussi indiquer le numéro des exercices sous forme de commentaires.***

1

L'objectif de cet exercice est d'énumérer tous les chemins d'un graphe acyclique et orienté (on ne peut pas revenir à un nœud déjà visité, on ne parcourt les arcs que dans un sens). Si un arc relie i à j, j est appelé le **successeur direct** de i.

1) On définit un graphe de N nœuds par une matrice d'adjacence. Chaque nœud i est connecté au nœud $i+1$. Construire cette matrice. Chaque coefficient m_{ij} vaut 1 s'il existe un arc $i \rightarrow j$, 0 sinon. Exemple pour trois nœuds :

$$adj[i][j] = \begin{pmatrix} . & 1 & . \\ . & . & 1 \\ . & . & . \end{pmatrix} \quad graphe : \quad \textcircled{0} \rightarrow \textcircled{1} \rightarrow \textcircled{2}$$

(2 points)

2) On ajoute 5 arcs aléatoires entre les nœuds i et j avec $i < j$ (module `random`). (2 points)

3) Ecrire une fonction `successeurs(adj,i)` qui détermine pour un nœud i tous ses successeurs directs. (2 points)

4) Ecrire une fonction `successeurs_dico(adj)` qui constuit un dictionnaire dont les clés sont les nœuds (0 à $N-1$) et les valeurs la liste des successeurs directs. (2 points)

5) On représente un chemin sous la forme d'une liste d'entiers. Il existe autant de manières de continuer ce chemin qu'il y a de successeurs directs. Ecrire une fonction `suites_chemin(chemin,dico)` qui prend comme argument un chemin `c` et le dictionnaire construit par la fonction précédente et qui retourne tous les suites possibles du chemin `c`. Par exemple, pour l'exemple de la question 1 :

```
suites_chemin( [ 0, 1 ], dico ) --> [ [ 0, 1, 2 ] ]
suites_chemin( [ 0, 1, 2 ], dico ) --> [  ]   # il n'y pas de suite possible
```

(2 points)

6) Par construction, le nœud 0 est normalement le seul nœud qui n'est pas un successeur. On l'appelle la **racine**. On veut énumérer tous les chemins partant de ce nœud. Implémentez l'algorithme suivant :

1. Initialisation de `chemins` = [[`0`]]
2. A l'itération i, on initialise la liste `chemins2` = []
3. Pour chaque élément de `chemins`, on lui applique la fonction `suites_chemin`. On ajoute le résultat à la liste `chemins2`.
4. On remplace `chemins` par `chemins2` puis on retourne à l'étape 2 jusqu'à ce que plus rien de change.

Il faut faire attention à ce que deviennent les chemins qui sont *terminés* : le dernier nœud n'a pas de successeur direct.

(4 points)

7) On distingue deux types de parcours d'arbres : en *profondeur d'abord* ou en *largeur d'abord*. Qu'est-ce qui le correspond le mieux à cet algorithme ? Et pourquoi ? (1 point)

8) On considère un autre graphe dont la matrice d'adjence vérifie $m_{ij} = 1$ pour tout $i < j$ et 0 pour tous les autres éléments. Quel est le nombre de chemins possibles ? Justifiez. (2 points)

2

1) On suppose qu'on dispose d'un tableau de nombres non trié. Ecrire une fonction qui retourne les trois éléments minimaux.

(2 points pour un coût de $O(n \ln n)$)

(3 points pour un coût inférieur ou égal à $O(n)$)

ENSAE TD noté, vendredi 5 décembre 2014

Le programme construit au fur et à mesure des questions devra être imprimé à la fin du TD et rendu au chargé de TD. ***Il ne faut pas oublier de mentionner son nom sur chaque page.*** *Les réponses autres que des parties de programme seront insérées sous forme de commentaires.* ***Il faudra aussi indiquer le numéro des exercices sous forme de commentaires.***

3

Quand on pense à la programmation dynamique, le premier exemple qu'on cite est l'algorithme du calcul du plus court chemin dans un graphe. La distance d'édition entre deux mots tient aussi de la programmation dynamique. Elle est définie par récurrence pour deux mots $A = (a_1, ..., a_p)$ et $B = (b_1, ..., b_q)$:

$$d(i,0) = d(0,i) = i \tag{1}$$

$$d(i,j) = \min \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1) + cout(a_i, b_j) \end{cases} \tag{2}$$

La valeur $d(i,j)$ définit la distance entre les débuts de mots $(a_1, ..., a_i)$ et $(b_1, ..., b_j)$. La fonction $cout(a_i, b_j) = 0$ si les deux caractères vérifient $a_i = b_j$, 1 sinon. On donne quelques exemples :

- distance(ami,amie) = 1 : on a ajouté e à la fin du second mot
- distance(amie,ami) = 1 : on a supprimé e à la fin du premier mot
- distance(ami,agi) = 1 : on a remplacé m par g

On va transformer ce problème en un problème du plus court chemin. Dans la suite, on considère deux `mot1='python'` et `mot2='piton'`. On rappelle que les deux notations suivantes sont équivalentes pour un dictionnaire :

```
dictionnaire [  i,j  ]
dictionnaire [ (i,j) ]
```

1) Ecrire une fonction `word2dict(mot)` qui prend comme argument un mot `mot` et qui construit un dictionnaire qui associe à tous les positions `i` la sous chaîne de caractères formée des premiers caractères de 1 à `i` exclu. Exemple :

```
mot --> { 0:'', 1:'m', 2:'mo', 3:'mot' }
```

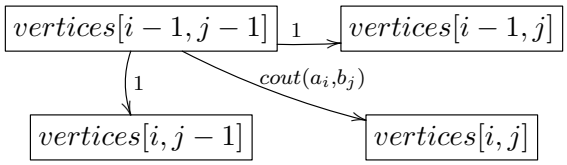
On applique cette fonction aux deux mots `mot1` et `mot2`. Indice : utiliser `mot[0 : i]`. (2 points)

2) Une seconde fonction `two_words2dicts(d1,d2)` qui construit un troisième dictionnaire à partir des deux premiers qu'on définit pour toutes les clés `i` de `d1` et toutes les clés `j` de `d2`. (2 points)

```
vertices = {  (i,j): (d1[i],d2[j]) }
```

3) Combien y a-t-il de clés dans ce dictionnaire ? (en fonction de p et q les tailles des deux mots) (1 point)

4) Comme le nom du dictionnaire choisi précédemment le suggère, le dictionnaire de la question précédente représente les noeuds du graphe. Il faut maintenant construire les arcs.



On pourra ensuite appliquer l'algorithme du plus court chemin. Dans ce cas, on définit l'ensemble des arcs du graphe (la matrice d'adjacence) par un dictionnaire :

```
edges = { (i1,j1),(i2,j2) : valeur }
```

Au début, ce dictionnaire est vide. Dans un premier temps, on ajoute les clés (arcs) suivants :

```
edges [ (i1,j1),(i2,j2) ] = 1 si  (i2-i1 == 1 et j1 == j2) ou (j2-j1 == 1 et i1 == i2)
```

Ecrire la fonction `add_edge_hv(vertices)` qui ajoute les arcs horizontaux et verticaux. (2 points)

5) Combien a-t-on ajouté d'arcs ? (en fonction de p et q les tailles des deux mots) (1 point)

6) On s'intéresse maintenant au cas où $\texttt{i2} - \texttt{i1} = \texttt{j2} - \texttt{j1} = 1$. Dans ce cas, la valeur ajoutée au dictionnaire est soit 1, soit 0 ($= cout(a_i, b_j)$). C'est 0 si les deux chaînes référencées par $\texttt{vertices}[\texttt{i2}, \texttt{j2}]$ se terminent par le même caractères, 1 sinon. (2 points)

7) Il ne reste plus qu'à implémenter l'algorithme du plus court chemin entre les nœuds $(0, 0)$ et (p, q). La seule difficulté est peut-être que les nœuds sont référencés par deux indices au lieu d'un entier. On crée un dernier dictionnaire $distance[i, j]$ qui contient la distance minimum entre le nœuds (i, j) et le nœuds $(0, 0)$. On initialise le dictionnaire avec $distance[0, 0] = 0$. On crée uen fonction `loop_on_edges` qui met à jour ce dictionnaire de la façon suivante :

– pour toutes les arcs $(\texttt{i1}, \texttt{j1}), (\texttt{i2}, \texttt{j2})$:
$\texttt{distance}[\texttt{i2}, \texttt{j2}] = \texttt{min}(\texttt{distance}[\texttt{i2}, \texttt{j2}], \texttt{distance}[\texttt{i1}, \texttt{j1}] + \texttt{edges}[(\texttt{i1}, \texttt{j1}), (\texttt{i2}, \texttt{j2})])$

(3 points)

8) La dernière fonction exécute la fonction `loop_on_edges` de nombreuses fois jusqu'à ce que le dictionnaire `distance` n'évolue plus. (3 points)

9) Appliquer le résultat à la paire `mot1='python'` et `mot2='piton'`. (1 point)

4

On a un tableau d'entiers $\texttt{l} = [1, 8, 5, 7, 3, 6, 9]$. On veut placer les entiers pairs en premiers et les entiers impairs en derniers : $8, 6, 1, 5, 7, 3, 9$. Ecrire une fonction qui fait cela.

(2 points si elle le fait avec un coût de $O(n \ln n)$)

(3 points si elle le fait avec un coût de $O(n)$