ENSAE TD noté, mardi 27 décembre 2013

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 inséré en commentaire au début du programme et de l'ajouter sur chaque page.*** *Les réponses autres que des parties de programme seront insérées sous forme de commentaires. Les squelettes de fonctions proposés ne sont que des suggestions.* ***Il faudra aussi indiquer le numéro des exercices sous forme de commentaires.***

1

La recherche dichotomique est assez simple. On cherche un élément e dans un tableau trié :

1. On compare e à l'élément du milieu.
2. S'ils sont égaux, on s'arrête. Si e est inférieur, on cherche dans la première partie du tableau. On cherche dans la partie supérieure dans l'autre cas.

On part de l'algorithme implémenté ici de façon récursive ou non et disponible ici : http://www.xavierdupre.fr/blog/2013-12-01_nojs.html.

1) Récupérez l'algorithme de la recherche dichotomique et appliquez-le à la liste [0, 2, 4, 6, 8, 100, 1000] et le nombre 100. (1 point)

2) On suppose qu'on dispose de deux listes triées de nombres, une pour les nombres impairs et une autre pour les nombres pairs. On veut écrire une fonction qui recherche un entier dans une de ces deux listes sachant que la parité du nombre indique dans quelle liste chercher et que le nombre cherché s'y trouve.

```
def deux_recherches(element, liste_impaire, liste_paire) :
    # ....
    return position_dans_liste_impaire, position_dans_liste_paire
```

Vous pouvez appliquer votre fonction à l'élement 100 et aux listes [0, 2, 4, 6, 8, 100, 1000], [1,3,5]. Le résultat attendu est (-1,5).

(2 points)

3) On suppose que les deux listes (nombres impairs, nombres pairs) sont de même tailles n. Dans le cas d'une recherche simple, le fait de couper la liste en deux est un avantage, est-ce toujours le cas dans le cas d'une recherche dichotomique ? Justifiez. (1 point)

4) Adaptez l'algorithme (de la question 1) pour retourner -1 lorsque l'élément à chercher n'est pas dans la liste. Il peut être utile de vérifier que vous arrivez bien à retrouver tous les éléments. (3 points)

```
l = [ 0, 2, 4, 6, 8, 100, 1000 ]
for i in l :
    print (i,recherche_dichotomique(i, l))
```

5) On modifie maintenant la fonction (de la question 2) de telle sorte qu'on cherche d'abord dans la liste des nombres impairs et si on ne trouve pas, on cherche dans la liste des nombres pairs. (2 points)

6) On doit chercher 1000 nombres impairs et 1 nombre pair. Quelle est la fonction la plus rapide pour $\ln_2(n) = 30$ ($n = 2^{30} \sim 10^9$)) ? Pourquoi ? (1 point)

2

On s'intéresse à la distance de Levensthein qui est une distance entre deux mots. Pour deux mots $L = (l_1, .., l_m)$ et $K = (k_1, ..., k_n)$, elle se résume par la relation de récurrence suivante :

$$d(i,j) = \min \left\{ \begin{array}{l} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1)+\mathbb{1}_{\{l_i \neq k_j\}} \end{array} \right\} \quad (1)$$

Le code est disponible à l'adresse : http://www.xavierdupre.fr/blog/2013-12-02_nojs.html

1) Récupérez la fonction sur Internet et appliquez cette fonction sur les couples de mots suivant :

1. $d(levenstein, levenshtein)$
2. $d(bonbon, bombon)$
3. $d(example, exemples)$
4. $d(esche, eche)$

(1 point)

2) Vérifiez que la distance est symétrique pour ces exemples. (1 point)

3) La fonction donne le même poids à toutes les confusions entre deux caractères. On souhaite que confondre n et m ait un coût de 0,5 au lieu de 1. Que vaut cette nouvelle distance pour la paire $d(bonbon, bombon)$? Modifiez le code de la fonction récupérée à la question 1. (3 points)

4) On veut faire en sorte qu'ajouter (ou supprimer) un s ait un coût de 0.5 ? Modifiez le code de la fonction. Que vaut la nouvelle distance $d(example, exemples)$? (3 points)

5) On veut faire en sorte qu'ajouter (ou supprimer) un s à la fin d'un mot ait un coût de 0,2 ? Que vaut la nouvelle distance $d(example, exemples)$. Modifiez le code de la fonction (de la question précédente). (2 points)

ENSAE TD noté, mardi 27 décembre 2013

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 inséré en commentaire au début du programme et de l'ajouter sur chaque page.*** *Les réponses autres que des parties de programme seront insérées sous forme de commentaires. Les squelettes de fonctions proposés ne sont que des suggestions.* ***Il faudra aussi indiquer le numéro des exercices sous forme de commentaires.***

3

On s'intéresse à la distance de Levenshtein qui est une distance entre deux mots. Pour deux mots $L = (l_1, .., l_m)$ et $K = (k_1, ..., k_n)$, elle se résume par la relation de récurrence suivante :

$$d(i,j) = \min \left\{ \begin{array}{l} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1) + \mathbb{1}_{\{l_i \neq k_j\}} \end{array} \right\} \tag{2}$$

Le code est disponible à l'adresse : http://www.xavierdupre.fr/blog/2013-12-02_nojs.html

1) Récupérez la fonction sur Internet et appliquez cette fonction sur les couples de mots suivant :

1. $d(levenstein, levenstien)$
2. $d(bonbbon, bonbon)$
3. $d(example, exemples)$

(1 point)

2) Vérifiez que la distance est symétrique pour ces exemples. (1 point)

3) La fonction donne le même poids à toutes les confusions entre deux caractères. On veut modifier la fonction de telle sorte qu'elle donne un coût deux fois moindre aux inversions de lettres. Modifiez le code de la fonction récupérée à la question 1. Quel est le nouveau coût de la paire $d(levenstein, levenstien)$? (4 points)

4) On veut donner à un coût de 0.45 à toutes les répétitions de lettres. Quel est le nouveau coût de la paire $d(bonbbon, bonbon)$? Modifiez le code de la fonction (de la question précédente). (4 points)

4

La recherche dichotomique est assez simple. On cherche un élément e dans un tableau trié :

1. On compare e à l'élément du milieu.
2. S'ils sont égaux, on s'arrête. Si e est inférieur, on cherche dans la première partie du tableau. On cherche dans la partie supérieure dans l'autre cas.

On part de l'algorithme implémenté ici de façon récursive ou non et disponible ici : http://www.xavierdupre.fr/blog/2013-12-01_nojs.html.

1) Récupérez l'algorithme de la recherche dichotomique et appliquez-le à la liste [0, 2, 3, 5, 10, 100, 340] et le nombre 100. (1 point)

2) Adaptez l'algorithme pour retourner -1 lorsque l'élément à chercher n'est pas dans la liste. Il peut être utile de vérifier que vous arrivez bien à retrouver tous les éléments. (3 points)

```
l = [ 0, 2, 4, 6, 8, 100, 1000 ]
for i in l :
    print (i,recherche_dichotomique(i, l))
```

3) On suppose qu'on a maintenant deux listes triées, il faut écrire une fonction qui cherche un élément dans chacune des deux listes et qui retourne deux positions. (2 points)

```
def deux_recherches(element, liste1, liste2) :
    # ....
    return position_dans_liste1, position_dans_liste2
```

4) On suppose que la liste 1 est 10 fois plus petite que la liste 2. On effectue 1010 recherches. Parmi elles, 1000 nombres sont dans la liste 1, 10 sont dans la liste 2. On considère deux options :

1. On cherche d'abord dans la liste 1. Si rien n'a été trouvé, on cherche dans la liste 2.
2. On cherche dans une liste triée qui contient les deux listes.

Quelle est la plus rapide ? (2 points)

5) On suppose que tous les éléments de la liste 1 sont inférieurs à tous les éléments de la liste 2. Comment utiliser cette information pour ne faire qu'une seule recherche. (2 points)