EA4 – Éléments d'algorithmique
TP noté (6 mai 2015)

Durée : 1h45

Documents autorisés.

Télécharger sur DidEL le fichier `mercredi.py`. Il contient les entêtes des fonctions demandées, ainsi que des outils vous permettant de les tester. Télécharger également le fichier `liste_mots`.

Exercice 1 : préfixe commun

On considère ici des listes de mots, parmi lesquels on souhaite identifier tous ceux qui possèdent un préfixe donné.

1. Compléter la fonction `liste_prefixe(pref, L)` pour qu'elle retourne la liste des mots d'une liste quelconque `L` qui commencent par le préfixe `pref`. Vous pouvez utiliser la méthode `startswith()`.
2. On suppose maintenant la liste `L` *triée en ordre alphabétique* (qui est l'ordre par défaut pour les chaînes de caractères en Python). Compléter la fonction `liste_prefixe_dicho(pref, L)` pour qu'elle retourne la liste des mots de `L` qui ont pour préfixe `pref` en utilisant une recherche dichotomique.
3. Modifier les fonctions précédentes pour qu'elles a□chent le nombre de comparaisons e□ectuées.

Exercice 2 : évolution de la hauteur d'un ABR

On souhaite étudier l'évolution de la hauteur d'un ABR au fur et à mesure de sa construction. Pour cela, et pour éviter de recalculer la hauteur de l'ABR à chaque étape, on enrichit la structure avec la donnée, pour chaque nœud, de la hauteur du sous-arbre correspondant.
Dans cet exercice, on représentera les arbres binaires par des listes imbriquées, chaque nœud étant représenté par une liste de longueur 4, `[valeur, fils_gauche, fils_droit, hauteur]`. L'arbre vide sera représenté par la liste vide. Par exemple, une feuille sera représentée par `[valeur, [], [], 0]`.

1. Compléter la fonction `insertion(arbre, x)` pour qu'elle insère l'élément `x` dans l'ABR `arbre` en mettant à jour le champ `hauteur` des nœuds concernés.
2. Compléter la fonction `creation(liste)` pour qu'elle retourne le couple `(A, H)` formé de l'ABR `A` obtenu par insertion successive des éléments de `liste`, et de la liste `H` des hauteurs successives des ABR intermédiaires.

Exercice 3 : distance minimale

On souhaite programmer la méthode de calcul de la distance minimale dans un nuage de points vue en cours. Le nuage est représenté par une liste de couples (les coordonnées cartésiennes des points). On rappelle que l'algorithme repose sur le principe « diviser pour régner » :

- couper le nuage en deux selon les abscisses ;
- calculer récursivement les distances minimales dans chaque demi-nuage ;
- chercher s'il existe deux points plus proches dans la bande médiane (verticale) que dans les demi-nuages ; pour cela, on peut se contenter de tester, pour chaque point de la bande, les cinq points suivants (pour l'ordre défini par les ordonnées des points).

1. Remplacer le corps de la fonction `tri_selon_x(L)` pour utiliser un tri par tas.
2. Compléter `distance_min(L)` et `distance_min_aux(Lx, Ly, debut, fin)` pour e□ectuer le calcul selon la méthode du cours.

Exercice 4 : opération ensembliste

On considère des ensembles d'entiers représentés par des listes triées sans doublon.
Compléter la fonction `deux_sans_trois(E, F, G)` pour qu'elle retourne la liste des entiers appartenant à *exactement* deux des trois ensembles représentés par `E`, `F` et `G`. Cette fonction doit *impérativement* s'exécuter en temps linéaire en la somme des longueurs des listes.