4TPM205U: Algorithmique des tableaux: feuille 5

Tableaux : 3

Travaux dirigés sur machine

Rappel : le fichier `bibTableau.py` *doit se trouver dans le même répertoire que vos programmes* Python.

Depuis la page `http://dept-info.labri.fr/ENSEIGNEMENT/algotab/src/` téléchargez le fichier `tp05.py`. Il contient des fonctions utilisées pour les tests. Vous y ajouterez les fonctions demandées.

Sauf indication contraire les tableaux manipulés contiennent des nombres.

Exercice 1 – Dans cet exercice on veut écrire une fonction pour *fusionner* les éléments de deux tableaux `t` et `s` triés dans l'ordre croissant non pas dans un troisième tableau mais dans le tableau `t` lui même (on supposera que la taille de `t` est suffisante).

Exemple : si `t` a une taille de 10 et contient 5 éléments : `t=[4,5,5,8,9,None,None,None,None,None]` et `s` a une taille de 5 et contient 3 éléments : `s=[1,3,5,None,None]`, suite à l'appel de cette fonction on aura `t=[1,3,4,5,5,5,8,9,None,None]`.

La fonction n'utilisera pas de tableau auxiliaire et **ne devra pas déplacer plus d'une fois** chacun des éléments des deux tableaux.

- Si on adopte l'algorithme de fusion de l'**exercice 1** de la feuille de TD#5 on commence par comparer le premier élément de `t` avec le premier élément de `s`. Sur l'exemple ci-dessus il faudra placer le premier élément de `s` à sa place définitive dans `t`. Est-ce que ce choix d'algorithme permet de respecter la consigne de déplacer au maximum une seule fois chacun des éléments de `t` ou de `s` ?
- Si l'on veut minimiser le nombre de déplacements d'éléments de `t`, quel est le couple d'éléments (l'un dans `t` et l'autre dans `s`) qu'on a intérêt à comparer en premier ?
- À quel indice se trouvera en fin de compte le plus grand parmi les éléments de `t` et `s` ?
- Lorsqu'on ne pourra plus comparer des couples d'éléments, puisque tous les éléments de l'un des deux tableaux seront à leur place définitive, comment procéder avec les éléments qui n'auront pas encore été traités ?

Testez vos hypothèses sur le couple de tableaux donné en exemple ci-dessus, puis sur le couple `t=[1,2,5,8,None,None,None,None,None]` à 4 éléments et `s=[4,5,7,10,13]` à 5 éléments. (Rappel : la fusion se fait dans le tableau `t`).

Dans le fichier `tp05.py` fourni écrire la fonction `insererTableauTrieDansTableauTrie(t,nt,s,ns)` qui insère les `ns` éléments d'un tableau `s` trié en ordre croissant dans un autre tableau `t` contenant `nt` éléments triées en ordre croissant. La fonction renverra le nouveau nombre d'éléments du tableau `t`.

Exercice 2

1. Écrire la fonction `supprimerOccurrences` étudiée à l'**exercice 2** de la feuille de TD#5.
2. En utilisant les fonctions `creerTableauAleatoire` et `tempsExecution` du fichier `tp05.py`, remplir le tableau suivant qui recense le temps d'exécution (en millisecondes) des fonctions `mystere` et `supprimerOccurrences` pour effacer toutes les occurrences de 0, en fonction du nombre d'éléments dans le tableau donné en paramètre.

   Attention : pour pouvoir comparer les temps d'exécution des deux fonctions, il faut qu'elles travaillent sur les mêmes données.

`nombre d'élements`	100	1000	10 000	
`mystere`				
`supprimerOccurrences`				

3. Ecrire une fonction `tailleMini(ms)` qui calcule le nombre minimal d'éléments d'un tableau aléatoire `t` pour que l'écart de temps de calcul entre `mystere` et `supprimerOccurrences` appliquées à `t` soit supérieur à `ms` millisecondes. Le résultat sera donné à un facteur 2 près. Pour cela on calculera la différence entre le temps d'exécution des deux fonctions seulement pour des tableaux ayant pour taille les puissances successives de 2.
4. Que vaut `tailleMini(30000)` (30 secondes) ? Sur un tableau de cette taille, quel est le temps d'exécution de `mystere` ? Celui de `supprimerOccurrences` ? Conclure.

Exercice complémentaire 1 – On souhaite écrire une fonction `ligneTrianglePascal(n)` qui crée et calcule dans un tableau `t` de taille $n+1$, la ligne numéro n du triangle de Pascal ($n \geq 0$). Par exemple pour $n = 4$ le tableau devra contenir `[1, 4, 6, 4, 1]`.
On rappelle que le triangle de Pascal pour $0 \leq n \leq 4$ est :

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Les éléments du triangle sont les coefficients binomiaux C_n^p où n représente la ligne et p la colonne ($0 \leq p \leq n$) dans le triangle de Pascal. Pour construire le triangle on pose $C_0^0 = 1$ puis, pour $n > 0$ et pour $1 \leq p \leq n-1$, on a $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$, c'est-à-dire que l'élément en ligne n et en colonne p se calcule à partir des éléments dans les colonnes p et $p-1$ de la ligne $n-1$.

La fonction à écrire **ne devra pas utiliser d'autres tableaux** que celui de taille $n+1$ créé au sein de la fonction. Il faut donc trouver le moyen de calculer chaque élément de la ligne n en utilisant les valeurs (qui représentent la ligne $n-1$) déjà présentes dans le tableau (et calculées à partir des valeurs représentant la ligne $n-2$, ...). La fonction n'effectuera aucune multiplication, la seule opération arithmétique nécessaire étant l'addition.

– Avec quelle valeur initialiser toutes les cases du tableau ?
– Quelle est la première case de la ligne n que l'on peut calculer sans écraser des valeurs qui seront nécessaires aux calculs suivants ?

Dans le fichier `tp05.py` écrire la fonction `ligneTrianglePascal(n)` qui crée et renvoie un tableau de taille `n+1` contenant la ligne numéro `n` du triangle de Pascal.