

Sujet E

□ Exercice : *type A*

On s'intéresse dans cette partie à un site Internet d'échange de supports de cours entre enseignants de MP2I/MPI. Chaque personne désirant proposer ou récupérer du contenu doit commencer par se créer un compte sur ce site et peut ensuite accéder à du contenu ou en proposer.

Ce site repose sur une base de données contenant en particulier une table, nommée **ressources**. Elle possède un enregistrement par document téléversé sur le site. Ses attributs sont :

- **id**, un identifiant numérique, unique pour chaque ressource ;
- **owner**, le pseudo de la personne ayant créé la ressource ;
- **annee**, l'année de publication de la ressource ;
- **titre**, une chaîne de caractères décrivant la ressource ;
- **type**, chaîne de caractères pouvant être cours, ds, tp ou td.

Voici un extrait de cette table :

id	owner	annee	titre	type
4	dknuth	2020	Machine à décalage	cours
13	alovelace	2022	Intelligence artificielle	td
...

1. Écrire une requête SQL permettant de connaître tous les titres des ressources déposées par « jclarke » classées par année de publication croissante.

```
SELECT titre
FROM ressources
WHERE owner="jclarke"
ORDER BY annee ASC;
```

2. Écrire une requête SQL permettant de connaître le nombre total de ressources de type cours présentes sur le site.

```
SELECT COUNT(*)
FROM ressources
WHERE type = "cours";
```

3. Que fait la requête suivante : ?

```
SELECT R.owner
FROM Ressources AS R
WHERE R.type = 'td'
GROUP BY R.owner
ORDER BY COUNT(*) DESC
LIMIT 3
```

Cette requête permet d'afficher les trois premiers propriétaires de ressources ayant posté le plus de ressources de type td

Cette base de données contient également une table **utilisateurs** qui contient les informations sur les utilisateurs du site. Elle possède un enregistrement par utilisateur. Ses attributs sont :

- **nom**, le nom de l'utilisateur (clé primaire) ;
- **mdp**, le mot de passe de l'utilisateur.
- **email**, l'adresse email de l'utilisateur.
- **naissance**, l'année de naissance de l'utilisateur.

Voici un extrait de cette table :

nom	mdp	email	naissance
dknuth	chepas123	dknuth@bigboss.com	1938
...

L'attribut `owner` de la table `ressources` est une clé étrangère qui référence l'attribut `nom` de la table `utilisateurs`.

4. Ecrire une requête SQL permettant de lister toutes les ressources déposées par des utilisateurs nés après 2000.

```
SELECT * FROM ressources
JOIN utilisateurs
ON ressources.owner = utilisateurs.nom
WHERE utilisateurs.naissance>2000
```

5. Ecrire une requête SQL permettant de lister tous les utilisateurs n'ayant déposé aucune ressource.

```
SELECT nom FROM utilisateurs
EXCEPT
SELECT owner FROM ressources
```

□ Exercice : type B

On considère un tableau d'entiers t de taille $n \neq 0$ et on s'intéresse au problème de la recherche de la somme maximale d'une tranche de t (c'est à dire d'éléments contigus de t). Par exemple, si $t = \{2, -7, -5, 4, -1, 10, -4, 9, -2\}$ alors la somme maximale d'une tranche est 18 (obtenue en prenant la tranche $\{4, -1, 10, -4, 9\}$). On notera $(t_i)_{0 \leq i \leq n-1}$ les éléments de t et $S_{i,j} = t_i + \dots + t_j$ ($0 \leq i \leq j \leq n-1$) la somme de la tranche comprises entre les indices i (inclus) et j (inclus).

Les fonctions demandées dans cet exercice doivent être écrites en langage C. Un fichier contenant le code compagnon de cet exercice est à télécharger à l'adresse <https://fabricenativel.github.io/cpge-info/oraux/>, il contient une fonction de prototype `int max3(int a, int b, int c)` qui renvoie le maximum des trois entiers `a`, `b` et `c` ainsi qu'une fonction `main` dans laquelle est définie le tableau donné en exemple ci-dessus.

1. Ecrire une fonction de signature `int tranchemax_naif(int tab[], int taille)` qui résoud ce problème en calculant de proche en proche les $S_{i,j}$ grâce aux relations $S_{i,i} = t_i$ et $S_{i,j} = S_{i,j-1} + t_j$ et en renvoyant leur maximum.

```
1  int tranchemax_naif(int tab[], int n)
2  {
3      int tmax = tab[0];
4      int sij;
5      for (int i = 0; i < n; i++)
6      {
7          sij = 0;
8          for (int j = i; j < n; j++)
9          {
10             sij += tab[j];
11             if (sij > tmax)
12             {
13                 tmax = sij;
14             }
15         }
16     }
17     return tmax;
18 }
```

2. Quelle est la complexité de cette fonction ?

On utilise donc deux boucles imbriquées dans laquelle on effectue uniquement des opérations élémentaires, la complexité est donc $\mathcal{O}(n^2)$.

On veut maintenant implémenter une méthode *diviser pour régner*, pour cela, on note $k = \left\lfloor \frac{n}{2} \right\rfloor$ et :

- On sépare t en deux sous tableaux $t_g = \{t_0, \dots, t_{k-1}\}$ et $t_d = \{t_{k+1}, \dots, t_{n-1}\}$ (on remarquera bien que t_k n'appartient à aucun de ces deux sous tableaux.)

- On recherche récursivement les tranches maximales de ces deux sous tableaux ainsi que la valeur maximale d'une tranche contenant t_k (qu'on pourra obtenir en considérant le maximum des tranches se terminant sur t_k et celui des tranches commençant en t_k).
 - on prend le maximum des trois valeurs obtenues.
3. Ecrire une fonction de signature `int max_tranche(int tab[], int start, int k, int end)` qui renvoie dans le tableau `tab` entre les indices `start` (inclus) et `end` (exclu) le maximum d'une tranche contenant l'élément `tab[k]`.

```

1 // Maximum des tranches qui contiennent l'élément d'indice k
2 int max_tranche(int tab[], int start, int k, int end)
3 {
4     int sk_max_left = tab[k];
5     int sk_max_right = tab[k];
6     int sk;
7     // Tranche qui se termine en k
8     sk = tab[k];
9     for (int i = k - 1; i >= start; i--)
10    {
11        sk = sk + tab[i];
12        if (sk > sk_max_left)
13        {
14            sk_max_left = sk;
15        }
16    }
17    // Tranche qui commence en k
18    sk = tab[k];
19    for (int i = k + 1; i < end; i++)
20    {
21        sk = sk + tab[i];
22        if (sk > sk_max_right)
23        {
24            sk_max_right = sk;
25        }
26    }
27    return max3(sk_max_left, sk_max_right, sk_max_left + sk_max_right - tab[k]);
28 }

```

4. Ecrire une fonction de signature `int tranchemax_dpr(int tab[], int size, int start, int end)` qui implémente la méthode diviser pour régner décrite ci-dessus.

```

1  int somme_maxi_aux(int tab[], int size, int start, int end)
2  {
3      if (end == start)
4      {
5          return tab[start];
6      }
7      if (end == start + 1)
8      {
9          return max3(tab[start], tab[end], tab[start] + tab[end]);
10     }
11     int s1, s2, s3;
12     int mid = (start + end) / 2;
13     s1 = somme_maxi_aux(tab, size, start, mid - 1);
14     s2 = max_tranche(tab, start, mid, end);
15     s3 = somme_maxi_aux(tab, size, mid + 1, end);
16     return max3(s1, s2, s3);
17 }
18
19 int tranchemax_dpr(int tab[], int size)
20 {
21     return somme_maxi_aux(tab, size, 0, size - 1);
22 }

```

5. Quelle est la complexité de la méthode diviser pour régner ?

Pour résoudre un problème de taille n , on doit en résoudre deux de tailles $\lfloor \frac{n}{2} \rfloor$ et rechercher le maximum des tranches contenant l'élément t_k . Cette opération a une complexité linéaire, on a donc :

$$\begin{cases} C(0) & \in O(1) \\ C(2n) & = 2C(n) + O(n) \end{cases}$$

La résolution de cette équation de complexité donne une complexité en $O(n \log n)$.