Devoir Surveillé

François Delbot Laurent Pierre

1er décembre 2015

1. Durée : toute la séance de TD
2. Aucun document, ni appareil électronique (même vos téléphones et vos écouteurs). En prticulier, vous n'avez pas le droit de consulter vos anciennes sources ou un site internet.
3. Barème indicatif, qui sera modifié, à votre avantage.
4. Vous devez rendre sur cours en ligne un fichier .c, dont le nom doit respecter le format suivant : NOM_Prenom.c
5. Votre fichier doit être encodé en ascii, ou utf8.
6. Votre fichier doit compiler. Testez le avant de le rendre, sinon aucune évaluation ne sera possible.
7. Le sujet est long, ne perdez pas de temps. Nous le savons, et nous en tiendrons compte pour la notation.

Attention : ce sujet nous sert à vous évaluer. Recopier des choses sur la copie/l'ordinateur d'à côté est une fraude et ne nous permet pas de vous venir en aide si nous constatons des difficultés. De plus, tricher ne peut pas vous permettre de vous épanouir, ni d'être fiers de vous. Alors, s'il vous plait : jouez le jeu !

1 Premier ? Pas premier ?

1.1 Les nombres premiers (1 points)

`Prototype de la fonction : int est_premier(int nb);`

Un nombre entier positif est premier si il n'est divisible que par 1 ET par lui-même. Les nombres 2, 3, 5, 7 et 11 sont premiers, 4, 6 et 15 ne sont pas premiers. Le nombre 1 n'est pas premier car il n'est divisible que par ... lui-même. Ecrire une fonction qui retourne 1 si le nombre passé en argument est premier, 0 sinon.

1.2 Premier facteur premier (1 points)

`Prototype de la fonction : int premier_facteur_premier(int nb);`

Soit $M = p \cdot q$, avec p et q deux nombres premiers. De nombreux cryptosystèmes reposent sur la complexité de retrouver les deux nombres p et q en un temps raisonnable. Ecrire une fonction qui retourne le plus petit facteur premier composant le nombre passé en argument. Par exemple, pour l'entier $15 = 3 \cdot 5$ votre fonction doit retourner 3.

2 Création de tableaux

2.1 Allocation dynamique de tableau (1 points)

`Prototype de la fonction : int * allocation_tableau(int taille);`

Ecrire une fonction qui alloue dynamiquement un tableau d'entiers dont la taille est passée en argument. Cette fonction doit retourner l'adresse du premier élément du tableau.

2.2 Remplissage d'un tableau (1 points)

`Prototype de la fonction : void remplir_tableau_aleatoire(int *tab, int taille, int max);`

Ecrire une fonction qui va remplir un tableau d'entiers (passé en argument avec sa taille) avec des valeurs aléatoires comprises entre 0 inclus et max (un entier passé en argument) inclus.

3 Trier un tableau

3.1 Partition d'une partie d'un tableau (2 points)

`Prototype de la fonction : int partition(int *tab, int debut, int fin);`

Ecrire une fonction qui partitionne une sous partie des éléments d'un tableau (délimitée par les indices *debut* et fin) de telle sorte que toutes les valeurs de cette sous partie qui sont inferieures à celle du pivot se trouvent à gauche du pivot, les autres se trouvant à droite du pivot. Par défaut, la première valeur de la sous partie à partitionner servira de pivot (c'est à dire la valeur d'indice *debut*).
Cette fonction doit retourner l'indice du pivot car il risque d'être déplacé.
Exemple :

- tableau en entrée : 1|2|-1|7|4|8|3|9|1|0|2|1|2|3
- debut : 4
- fin : 10
- tableau après partitionnement : 1|2|-1|7|1|0|2|3|4|8|9|1|2|3
- retour de la fonction : 8

3.2 Tri rapide (1 points)

`Prototype de la fonction : void tri_rapide(int *tab,int debut, int pivot, int fin);`

implémentez l'algorithme du tri rapide (Quick Sort) de manière récursive : Cet algorithme consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Il suffit ensuite d'appliquer récursivement sur ces deux parties, droite et gauche. On suppose que l'on possède déjà une fonction permettant de partitionner les valeurs d'une partie d'un tableau et qui retourne l'indice du pivot. Le prototype de cette fonction de partition est le suivant :

```
int partition(int *tab, int debut, int fin);
```

3.3 Recherche par dichotomie (version recursive) (1 points)

`Prototype de la fonction : int dichotomie_recursive(int *tab,int debut,int fin,int valeur);`

Lorsqu'un tableau est trié (par ordre croissant par exemple), il est possible de retrouver efficacement une valeur recherchée en utilisant une recherche par dichotomie. Le principe est de comparer m, la valeur stockée au milieu du tableau, à *valeur*, la valeur recherchée. Si elles sont égale, c'est que la valeur est présente dans le tableau. Si $m > valeur$, il faut chercher la valeur dans la moitié inférieur du tableau, sinon dans la moitié supérieure. On recommence alors sur la portion restante du tableau et ainsi de suite jusqu'à trouver la valeur recherchée ou que l'espace de recherche restant soit vide.

Écrire une fonction (récursive) qui implémente cette méthode. Elle devra retourner l'indice de la case dans laquelle se trouve la valeur recherchée si elle est présente dans le tableau, -1 sinon.

4 Les points-cols

4.1 Minimum de la colone (1 points)

`Prototype de la fonction : int min_colonne(int **mat,int taille, int indice_col);`

Ecrire une fonction qui, étant donné une matrice carrée et l'indice d'une colonne, retourne la valeur minimum contenu dans cette colonne. On considère que la première dimension de la matrice représente l'abscisse et la seconde l'ordonnée. Ainsi, dans la matrice suivante
$1, 2, 3$
$4, 5, 6$
$7, 8, 9$
Le point de coordonnée $(0, 2)$ vaut 7.

4.2 Maximum d'une ligne (1 points)

`Prototype de la fonction : int maximum_ligne(int **mat, int taille, indice_ligne);`

Ecrire une fonction qui, étant donné une matrice carrée et l'indice d'une ligne, retourne la valeur maximum contenu dans cette ligne. On considère que la première dimension de la matrice représente l'abscisse et la seconde l'ordonnée. Ainsi, dans la matrice suivante
$1, 2, 3$
$4, 5, 6$
$7, 8, 9$
Le point de coordonnée $(0, 2)$ vaut 7.

4.3 Recherche de point col (1 points)

`Prototype de la fonction : int point_col(int **mat, int taille, int x, int y);`

On souhaite rechercher dans une matrice carrée les éléments qui sont à la fois un maximum sur leur ligne et un minimum sur leur colonne. Ces éléments sont appelés des points-cols. Ecrire une fonction qui retourne 1 si la valeur contenue aux coordonnées (x, y) est un point col, 0 sinon.

On considère que la première dimension de la matrice représente l'abscisse et la seconde l'ordonnée. Ainsi, dans la matrice suivante
$1, 2, 3$

$4, 5, 6$
$7, 8, 9$
Le point de coordonnée $(0, 2)$ vaut 7.

5 Le problème de la somme de sous-ensembles

5.1 Décomposition en base 2 (1 points)

`Prototype de la fonction : void decomposition_base_2_tableau(int nb, int *tab, int taille);`

Ecrire une fonction qui décompose un entier nb en base 2. Plus précisemment, vous devez remplir le tableau passé en argument avec cette décomposition, chaque case du tableau correspondant à un chiffre (0 ou 1) de la décomposition. Le nombre de chiffres de la décomposition est donné par la taille du tableau. Le bit de poids faible (l'unité) se trouve dans la case d'indice le plus élevé du tableau. Par exemple avec $nb = 5$ et $taille = 5$ on obtient le tableau suivant : 0|0|1|0|1

5.2 Le problème de la somme de sous-ensembles (3 points)

`Prototype de la fonction : int subsetsum(int *ensemble, int taille, int objectif);`

On considère le problème de la somme de sous ensembles. Etant donné un ensemble de *taille* entiers contenus dans *ensemble* le tableau passé en argument, existe-t-il un sous-ensemble X de ces entiers dont la somme (des entiers de X) vaut $objectif$? Pour cela, vous devez tester tous les sous ensembles possibles en utilisant la fonction de décomposition en base 2 de l'exercice précédant. Si la décomposition en base 2 place un 1 à l'indice k cela signifie que l'entier situé à l'indice k dans le tableau *ensemble* fait partie de X. Si un tel sous ensemble existe, vous devez retourner l'entier correspondant à la décomposition sinon, retourner -1.

6 Les piles

6.1 Définition de la structure pile (1 points)

`Prototype de la fonction :`

Définissez une structure permettant de manipuler une pile au moyen d'un tableau de taille 10.

6.2 La pile est-elle vide ? (1 points)

`Prototype de la fonction : int Pile_tableau_est_vide(pile p);`

Ecrire une fonction qui retourne 1 si la pile passée en argument est vide, 0 sinon.

6.3 La pile est-elle pleine ? (1 points)

`Prototype de la fonction : int Pile_tableau_est_pleine(pile p);`

Ecrire une fonction qui retourne 1 si la pile passée en argument est pleine, 0 sinon.

6.4 Dépiler (1 points)

`Prototype de la fonction : int Pile_tableau_depiler(pile *p);`

Ecrire une fonction qui enleve l'élément situé au sommet de la pile et le retourne. Si cela n'est pas possible, retourner l'entier le plus petit possible (limits.h).

6.5 Empiler (1 points)

`Prototype de la fonction : int Pile_tableau_empiler(pile *p, int element);`

Ecrire une fonction qui ajoute un élément au sommet de la pile dont le pointeur est passé en argument. Si l'opération se déroule correctement, la fonction retourne 0, 1 sinon.