Devoir d'entraînement

Informatique MP2I

Durée : **2 heures**

Tous documents, calculatrices, machines connectées interdits.

Ce sujet est composé de 4 exercices indépendants. Ce sujet comporte 5 pages.

— Les programmes doivent être écrits en C.
— Une grande importance sera apportée à la clarté et à la lisibilité.
— On pourra toujours librement utiliser une fonction démandée dans une question précédente, même si cette question n'a pas été traitée.
— On suppose que les bibliothèques `stdbool.h`, `stdio.h`, `stdlib.h` et `assert.h` ont déjà été importées et que vous n'avez donc pas à le faire.

Exercice 1 Implémentation d'ensemble par représentation binaire

On rappelle que le type C `unsigned int` représente les entiers naturels sur 4 octets selon la représentation vue en cours.
Dans cet exercice, on utilise la représentation binaire des entiers naturels pour coder des ensembles : à chaque entier écrit en base 2 on associe l'ensemble dont les éléments sont les positions des bits égaux à 1.
Par exemple, $\overline{1101}^2$ a trois bits à 1, en positions 0, 2 et 3. Donc l'entier $\overline{1101}^2$, représente l'ensemble $\{0, 2, 3\}$.
Comme un `unsigned int` est stocké sur 32 bits, cette méthode permet donc de coder n'importe quel sous-ensemble de $\{0, ..., 31\}$ sous forme d'un `unsigned int`.

Q1a. Quelle est la représentation décimale de l'entier qui code $\{0, 5, 6\}$?

Q1b. Quel est l'ensemble codé par l'entier 43 ?

Dans un soucis de concision, pour toutes les questions suivantes, on renomme en `bitfield` le type `unsigned int`. Vous pouvez librement utiliser le nom `bitfield` sur votre copie en lieu et place du nom `unsigned int` :


```c
typedef unsigned int bitfield;
```

Q2. Écrire une fonction `bitfield singleton(int elem)` qui renvoie le `bitfield` qui encode l'ensemble à un élément $\{elem\}$. On pourra supposer, sans avoir à le vérifier, que `elem` appartient bien à $[0, 31]$.

Q3. Écrire une fonction `bool empty(bitfield b)` qui renvoie `true` si et seulement si `b` encode l'ensemble vide.

On rappelle la définition de l'opérateur vu en TP "et bit-à-bit" noté `&`. Soient `b1` et `b2` deux `bitfield`. Pour tout $i \in [0, 31]$, le bit en position i de `b1 & b2` est à 1 si et seulement si les bits en position i de `b1` **et** de `b2` sont à 1.

Q4. Écrire une fonction `bitfield inter(bitfield b1, bitfield b2)` qui renvoie le `bitfield` encodant l'intersection des ensembles codés par `b1` et `b2`.

Q5. Écrire une fonction `bool mem(bitfield b, int elem)` qui renvoie `true` si et seulement si l'élément `elem` est dans l'ensemble codé par `b`. On pourra supposer, sans avoir à le vérifier, que `elem` appartient bien à $[0, 31]$.

Q6a. Soit `b` un `bitfield` qui encode un ensemble B non-vide de minimum m. Montrer que `b & (b - 1)` encode $B \backslash \{m\}$. On pourra raisonner séparément sur les bits de `b` aux positions supérieures, inférieures ou égale à m.

Q6b. En déduire une fonction `int card(bitfield b)` qui renvoie le cardinal de l'ensemble encodé par `b`.

Exercice 2 Liste chaînée déroulée

La mémoire des ordinateurs modernes est optimisée pour accéder de façon successive à des adresses contiguës. Pour cette raison les listes chaînées, dont les maillons sont alloués de façon non-nécessairement contiguë en mémoire, ont souvent des performances pratiques dégradées comparées à celles des tableaux.

Il existe une structure à mi-chemin entre le tableau et la liste chaînée qu'on nomme **liste chaînée déroulée**. Le concept est similaire à celui de la liste chaînée, mais chaque maillon stocke jusqu'à k éléments de façon contiguë plutôt que 1.

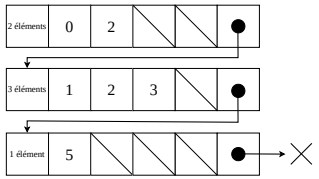


FIGURE 1 – Exemple de maillons d'une liste chaînée déroulée pour les éléments $0, 2, 1, 2, 3, 5$ avec $k = 4$.

On n'impose pas spécialement que tous les maillons soient remplis au maximum. En Figure 1, un 1er maillon contient les deux premiers éléments, un 2ème contient les trois suivants et le 3ème maillon en contient un dernier.

Dans cet exercice, on pose $k = 8$. On se propose d'implémenter cette structure.

```c
struct urmaillon {
  int nb_elem;            //nombre d'éléments réellement stockés ici
  int content[8];         //tableau pouvant accueillir au plus 8 éléments
  struct urmaillon *next; //pointeur vers éventuel maillon suivant (sinon NULL)
};
typedef struct urmaillon urmaillon;

struct urlist {
  urmaillon *first; //pointeur vers éventuel premier maillon (sinon NULL)
  urmaillon *last;  //pointeur vers éventuel dernier maillon (sinon NULL)
};
typedef struct urlist urlist;
```

Vous devez implémenter les différentes fonctions de l'interface de cette structure. **Vos fonctions doivent maintenir vraie la propriété sur les `urlist` (P) suivante** :

(P) : tous les `urmaillon` d'une `urlist` non-vide contiennent entre 4 et 8 éléments, à l'exception éventuellement du dernier qui en contient entre 1 et 8.

Q1. Écrire une fonction `urlist *empty()` qui renvoie un pointeur vers une nouvelle `urlist` qui ne contient aucun élément.

Q2. Écrire une fonction `int length(urlist *l)` qui renvoie le nombre d'éléments contenu dans la `urlist` pointée par `l`.

Q3. Écrire une fonction `void append(urlist *l, int elem)` qui ajoute l'élément `elem` à la fin de la liste pointée par `l`. *Indication :* plusieurs stratégies sont possibles, mais faites attention à maintenir vraie la propriété (P).

Q4. Écrire une fonction `int get(urlist *l, int i)` qui renvoie l'élément en position i dans la `urlist` pointée par `l`. Vous pourrez supposer, sans avoir besoin de le vérifier dans le programme, que la liste pointée par `l` contient strictement plus de `i` éléments.

Q5. Écrire une fonction `void delete(urlist *l)` qui libère l'espace mémoire occupé par la `urlist` pointée par `l` et ses maillons.

Q6. Écrire une fonction `void insert(urlist *l, int i, int elem)` qui insère l'élément `elem` en position `i` dans la `urlist` pointée par `l`. L'élément `elem` devient le nouvel élément en position `i` et la position de tous les éventuels éléments qui étaient en positions supérieures ou égales à `i` sont augmentées d'un. On pourra supposer, sans le vérifier, que la liste pointée par `l` est non-vide et que $0 \leq i \leq \mathit{length}(l)$.

Exercice 3 L'élément solitaire

Soit `t` un tableau contenant `n` entiers naturels. Tout au long de cet exercice, on suppose que tous les éléments de `t` y apparaissent en deux exemplaires, à l'exception d'un unique élément `x` qui n'y apparaît qu'une seule fois. On souhaite trouver cet élément `x`.
Par exemple, dans le tableau dont les éléments sont `1,2,4,5,6,5,6,4,1`, l'élément à trouver est `2`.

Q1. Écrire une fonction `unsigned int lonely(unsigned int t[], unsigned int n)` qui renvoie l'élément solitaire dans le tableau `t` de taille `n`. On attend une complexité dans le pire cas en $O(n^2)$ et une brève justification de cette complexité.

Il existe en C un opérateur dénoté `^` qu'on appelle le "*XOR* bit-à-bit", ou "*ou exclusif* bit-à-bit". Soient `a` et `b` deux `unsigned int`. Alors `a ^ b` est également un `unsigned int` qu'on définit ainsi :

pour tout $i \in [0, 31]$, le bit en position i de `a ^ b` est à 1 si et seulement si les bits en position i de `a` et de `b` sont différents (l'un des deux est à 0, l'autre est à 1).

Calculer le *XOR* bit-à-bit de deux `unsigned int` compte comme 1 opération pour vos analyses de complexité.

Q2. Montrer que pour tout `unsigned int a`, la valeur de `a ^ a` est `0`. *Indication :* on pourra raisonner sur chaque bit séparément.

Q3. Montrer que pour tout `unsigned int a`, la valeur de `a ^ 0` est `a`.

Q4. Montrer que pour tous `unsigned int a` et `b`, la valeur de `a ^ b` est la même que celle de `b ^ a`.

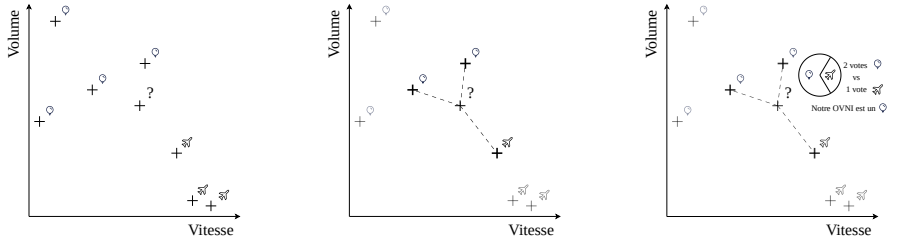
Q5. Montrer que pour tous `unsigned int a`, `b` et `c` la valeur de `(a ^ b) ^ c` est la même que celle de `a ^ (b ^ c)`

Q6. En déduire une fonction `unsigned int lonely_optimized(unsigned int t[], unsigned int n)` qui renvoie l'élément solitaire dans le tableau `t` de taille `n`. On attend une complexité dans le pire cas en $O(n)$ et une brève justification de cette complexité.

Exercice 4 Méthode des k plus proches voisins

Le radar embarqué de votre avion a détecté un objet volant non-identifié. Vous pensez qu'il s'agit soit d'un ballon météo, soit d'un avion civil. Vous connaissez la vitesse et le volume de l'OVNI. Dans cet exercice on souhaite écrire un programme qui détermine si l'OVNI est un avion civil ou un ballon météo.

Comme on ne connaît pas d'algorithme qui résolve ce problème (très concret) précis, il va falloir utiliser un méthode d'intelligence artificielle basique. On suppose qu'on dispose d'un **jeu de données** de tous les précédents OVNIs détectés par le passé. Chacun de ces OVNIs a été **étiqueté** a posteriori en fonction de s'il s'agissait en réalité d'un ballon ou d'un avion.
Pour identifier notre OVNI o, on va regarder les k profils du jeu de données les plus proches de o en terme de vitesse et de volume. Si une majorité de ces k profils correspondent à des ballons, on décrète que o est un ballon météo. Sinon on décrète que o est un avion civil. C'est la méthode des k **plus proches voisins**.



Jeu de données étiquetées avion ou ballon et OVNI ? à identifier.

On trouve les k plus proches voisins du profil ?.

Majorité de ballons parmi les voisins, l'OVNI est classifié comme un ballon.

FIGURE 2 – Méthode des k plus proches voisins (avec $k = 3$).

La Figure 2 représente un exemple de la méthode des 3 plus proches voisins. On cherche à classifier le profil marqué d'un (?). On cherche les trois profils les plus proches parmi le jeu de données. Parmi ces profils, deux correspondent à des ballons météos et un à un avion, on propose donc de classifier l'OVNI comme un ballon météo.

On propose la distance d suivante entre deux profils d'OVNI A, B de vitesses respectives c_A, c_B et volumes respectifs V_A, V_B :

$$d(A, B) = (c_A - c_B)^2 + (V_A - V_B)^2$$

On représente les profils d'OVNIs du jeu de données par le type structuré suivant :

```
struct ovni {
  float speed;  // vitesse de l'OVNI (m/s)
  float volume; // volume  de l'OVNI (m cube)
  bool plane;   // true si avion, false si ballon météo
};
typedef struct ovni ovni;
```

Q1. Écrire une fonction `float distance(float c, float V, ovni p)` qui calcule la distance entre un profil `p` et celui de l'OVNI actuel de vitesse `c` et de volume `V`.

On suppose que le jeu de données contient $n > 1$ profils. On représente le jeu de données par un tableau `database` contenant `n` éléments de type `ovni`.
Dans cet exercice nous allons considérer $k = 1$ (Q2) puis $k = n$ (Q3) puis k quelconque dans $[1, n]$ (Q4).

Q2. Écrire une fonction `bool one-nn(float c, float V, ovni database[], int n)` qui renvoie `true` si et seulement si l'OVNI de vitesse `c` et de volume `V` est un avion selon la méthode des k plus proches voisins avec $k = 1$.

Q3a. Dans cette question, on pose $k = n$. Soient deux profils quelconque d'OVNIs o_1 et o_2. Est-il possible que la méthode des k plus proches voisin classifie o_1 comme un ballon météo et o_2 comme un avion ? Pourquoi ?

Q3b. Écrire une fonction `bool all-nn(float c, float V, ovni database[], int n)` qui renvoie `true` si et seulement si l'OVNI de vitesse `c` et de volume `V` est un avion selon la méthode des k plus proches voisins avec $k = n$.

Q4. Écrire une fonction `bool k-nn(float c, float V, int k, ovni database[], int n)` qui renvoie `true` si et seulement si l'OVNI de vitesse `c` et de volume `V` est un avion selon la méthode des `k` plus proches voisins. On pourra supposer, sans avoir à le vérifier, que `k` est compris entre 1 et n.

Nous discuterons en deuxième année du choix de la valeur k et de ses conséquences.