Université Savoie Mont Blanc, UFR SceM
INFO505, L3 INFO

24 octobre 2023

TD 4, Pointeurs, manipulation de bits et tableau de bits, ensembles par fonction caractéristique

---

On veut faire des tableaux de taille arbitraire de booléens, mais en minimisant la mémoire nécessaire. On propose donc de faire un tableau de bits, où l'utilisateur pourra accéder et modifier chaque booléen du tableau de façon transparente.

Question 1. Type de données `VectorBool`

On pourrait réutiliser le type `Vector` adapté aux entier mais on demande ici de redéfinir un type spécialisé. L'utilisateur pourra spécifier à la création le nombre de booléens qu'il va utiliser.

Question 2. Initialisation

Ecrire l'action `VectorBool_init` qui crée un tableau de booléens de taille spécifiée par l'utilisateur. On écrira aussi une fonction `VectorBool_clear` qui met à faux tous les booléens. Celle-ci sera appelée à l'initialisation.

Question 3. Mettre vrai ou faux dans un booléen

Ecrire l'action `VectorBool_set` qui met à vrai le booléen d'indice spécifié. Ecrire l'action `VectorBool_reset` qui met à faux le booléen d'indice spécifié.

Question 4. Tester la valeur d'un booléen

Ecrire la fonction `VectorBool_get` qui retourne la valeur du booléen d'indice spécifié.

Question 5. Compter le nombre de vrai dans le tableau

Ecrire la fonction `VectorBool_nbTrue` qui retourne le nombre de valeurs à vrai dans le tableau. Doit-on regarder toutes les cases bit à bit ?

Question 6. Utilisation comme représentation d'ensemble par fonction caractéristique

Supposons que l'on veuille représenter un ensemble d'entiers pris parmi $\{0, 1, \ldots, n-1\}$, où n ne dépasse pas les 30 millions (environ). Il peut être alors intéressant d'utiliser un tableau de bits pour représenter cet ensemble. Soit S un ensemble et T le tableau de bits correspondant, on définit :

$$\forall x \in \{0, 1, \ldots, n-1\}, x \in S \Leftrightarrow T[x] = 1$$

Le coût mémoire est de l'ordre de $n/8$ octets. Tester la présence d'un élément est très rapide. Si on compare à une structure de type arbre binaire, le coût mémoire est de l'ordre de $20m$ octets, où m est le nombre d'éléments de l'ensemble. Le tableau de bits est donc intéressant dans le cas où m n'est pas trop petit et devient meilleur dès lors que $m \geq n/160$. On utilise notamment ces représentations par ensemble caractéristique pour représenter une région dans une image, afin de savoir très rapidement si un pixel appartient ou non à la région.

1. Définissez un type `EnsembleC` en utilisant le type `VectorBool`.
2. Ecrivez les actions/fonctions

```
void EnsembleC_init( EnsembleC* E, int n );
void EnsembleC_insere( EnsembleC* E, int v );
void EnsembleC_supprime( EnsembleC* E, int v );
int EnsembleC_appartient( EnsembleC* E, int v );
void EnsembleC_termine( EnsembleC* E, int n );
int EnsembleC_nbElements( EnsembleC* E );
int EnsembleC_estVide( EnsembleC* E );
```

3. De façon plus générale, on voudrait faire l'intersection, l'union ou la différence d'ensembles.

```
/* E = E1 inter E2, en sortie. */
void intersection( EnsembleC* E, EnsembleC* E1, EnsembleC* E2 );
...
```

Que doit-on rajouter à `VectorBool` pour faire les choses efficacement ?