TD No6: tables de hachage

Exercice 1 table de hachage: gestion des collisions par chaînage

Rappel de cours: dans une table de hachage, un élément e ayant une clef c est stocké à l'indice h(c). h est appelée fonction de hachage. Si la fonction h était injective (à deux éléments différents ont toujours des indices différents), l'accès à un élément se ferait en temps constant: le calcul de h(c).

En pratique, il n'est pas possible d'éviter que deux éléments aient le même indice. On parle alors de collision. L'une des méthode de gestion de collision appelée « gestion des collisions par chaînage » consiste à stocker dans le tableau des listes d'éléments plutôt que les éléments eux-même. Ainsi, dans la cellule numéro 4, on aura la liste des éléments dont l'indice (h(c)) vaut 4.

La complexité devient alors proportionnelle à la taille des listes.

Question 1 exemple

On considère la fonction de hachage: h(c) = m mod m avec m =11. Représentez l'état de la table de hachage après

- l'ajout des clefs: 22;4;39;88;15;17;59;16
- la suppression de 15

Simulez la recherche de 59 dans la table.

Question 2

Ecrire les algorithmes permettant d'ajouter, rechercher ou supprimer un élément dans une table de hachage en respectant les étapes suivantes :

- donner les prototypes des fonctions travaillant sur les tables de hachage
- donner les prototypes des fonctions utiles travaillant sur les listes chaînées
- écrire les fonctions travaillant sur les tables de hachage
- écrire les fonctions travaillant sur les listes chaînées

On utilisera les types suivants pour représenter les listes et les tables de hachage :

```
typedef struct telement {
  int clef;
  struct telement * suivant;
} element;
typedef element * liste;
typedef struct ttable {
  int taille;
  liste * tab;
}table;
```


Exercice 2 table de hachage: gestion des collisions par adresage ouvert

Une autre méthode de gestion des collisions utilisée lorsque l'on n'a pas la possibilité de gérer des listes chaînées consiste à stocker les éléments en collision ailleurs dans le tableau. On utilise alors une fonction h à deux arguments: h(c,i) avec i=nombre d'essais de placement déjà effectués. Ajouter un élément de clef c à la table revient alors à regarder sur la cellule d'indice h(c,0) est vide. Si c'est le cas, on y met l'élément. Dans le cas contraire, on essaie la cellule d'indice h(c,1), puis h(c,2), ...

Question 1 exemple

On considère la fonction de hachage: h(c,i) = (h'(c)+ i) mod m avec h'(c)= m mod m avec m =11. On vous demande de :

- Représentez l'état de la table de hachage après l'ajout des clefs: 22;4;39;88;16;15;17;59;
- Simuler la recherche de 59
- Représentez l'état de la table de hachage après la suppression de 15
- Simulez la recherche de 59
- représentez l'état de la table après l'ajout de 27

Question 2

Ecrire les algorithmes permettant d'ajouter, rechercher ou supprimer un élément dans une table de hachage.

On utilisera les types suivants pour représenter les tables de hachage :

```
#define VIDE 0
#define PLEIN 1
#define DEJAOCCUPE 2
typedef struct telem{
  int clef;
  int statut; // vide, plein ou libre mais déjà occupé
typedef struct ttable {
  int taille;
  liste * tab;
}table;
```

Exercice 3 B-arbres

Question 1

Insérez les éléments suivants dans un arbre 2-3-4 (un abre 2-3-4 est un B-arbre de degré minimal 2):

6, 30, 20; 15; 3; 35; 40; 42