Informatique *TP*
Logique Ipesup

Exercice

Écrire une fonction qui utilise une pile pour évaluer une expression arithmétique en notation postfixée. Ainsi, l'évaluation de l'expression `4 4 3 1 - -` doit renvoyer la valeur `42`. On pourra supposer que l'expression est donnée sous la forme d'une liste de chaînes de caractères.

Files

Exercice

Réaliser en C une structure de file mutable à l'aide de deux piles.

Exercice

Écrire une implémentation C d'une file réalisée par un tableau circulaire.

Exercice

Discuter la réalisation d'une file par un tableau circulaire en utilisant un *tableau redimensionnable* plutôt qu'un tableau, afin que la capacité ne soit pas limitée.

Tables de hachage

Exercice

Expliquer pourquoi `let i = abs (hash k) % n` n'est pas une solution acceptable pour la fonction `bucket`.

Exercice

Montrer qu'il existe deux chaînes de 14 caractères, pris parmi les vingt-six lettres de l'alphabet, qui donnent la même valeur par la fonction `hash` qu'on a decrit.

Exercice

On poursuit l'exercice précédent en cherchant encore plus de collisions pour la fonction `hash`.

1. Montrer qu'il existe deux chaînes de longueur deux, distinctes, formées uniquement de caractères alphabétiques dans `A-Z` et `a-z` (52 caractères au total), qui ont la même valeur pour la fonction `hash`.
2. En déduire une façon simple de construire un nombre arbitraire de chaînes de caractères ayant la même valeur pour la fonction `hash`.

Exercice

Ajouter au programme 3Tables de hachage en OCaml3 une fonction `remove` qui supprime de la table de hachage l'entrée correspondant à une clé donnée, si elle apparaît dans la table, et ne fait rien sinon. De même, ajouter au programme "Tables de hachage en C" une fonction `hashtbl_remove` avec la même spécification.

Exercice

Utiliser des seaux comme dans les programmes "Tables de hachage en C" et "Tables de hachage en OCaml" n'est pas la seule façon de résoudre les collisions dans une table de hachage. Cet exercice explore une autre solution, appelée *adressage ouvert*, qui consiste à se servir uniquement de deux tableaux, un pour les clés et l'autre pour les valeurs associées. Pour une clé k, on calcule l'indice $i = h(k) \pmod m$ donné par la fonction de hachage. Si la case i est libre, on y stocke cette entrée. Sinon, on examine les cases suivantes $i + 1, i + 2, \ldots$ à la recherche d'une case libre, et on prend la première que l'on trouve. Cette stratégie s'appelle le *sondage linéaire* (en anglais *linear probing*).

1. Identifier une condition nécessaire, portant sur le nombre d'entrées n et la taille m du tableau, pour que cette stratégie puisse fonctionner.
2. Proposer un type C pour une telle table de hachage, où les clés sont des chaînes (de type `char*`) et les valeurs des entiers (de type `int`).
3. Comparer l'espace occupé par cette table de hachage et celle du programme 7.20, en fonction du nombre d'entrées n et de la taille m des tableaux. Il peut être utile de faire intervenir la quantité $\alpha = n/m$ que l'on appelle la *charge* de la table de hachage.
4. Donner le code d'une fonction C qui détermine à quel indice doit être associée une clé k, selon la stratégie ci-dessus.
5. En déduire le code de trois fonctions C qui ajoute une entrée dans la table, détermine si une clé apparaît dans la table et renvoie la valeur associée à une clé, le cas échéant, et la valeur -1 sinon.
6. Discuter le problème de la suppression d'une entrée dans la table.

Exercice

(*filtre de Bloom*) Un *filtre de Bloom* est une structure de données qui réalise un ensemble et fournit deux opérations : ajouter un élément et tester la présence d'un élément. Cette dernière opération doit donner un résultat correct pour les éléments qui ont été ajoutés à l'ensemble mais elle peut donner un résultat incorrect pour les autres éléments. Un filtre de Bloom utilise un tableau de m booléens et k fonctions de hachage $h_1, \ldots, h_k$ qui envoient les éléments sur $0, \ldots, m-1$. Quand on ajoute un élément x, on met à `true` les booléens aux indices $h_1(x), \ldots, h_k(x)$. Quand on teste la présence de x, on renvoie `true` si et seulement si tous les booléens aux indices $h_1(x), \ldots, h_k(x)$ sont à `true`.

1. Proposer une implémentation (en OCaml ou en C) d'un filtre de Bloom pour des chaînes de caractères, où les paramètres k et m sont passés en arguments au constructeur. Pour la fonction de hachage h_i, on pourra reprendre la fonction `hash` qu'on a defini, où la constante 31 est remplacée par un entier tiré au hasard au moment de la construction.
2. Tester empiriquement l'efficacité d'un tel filtre, pour différentes valeurs des paramètres k et m. Par exemple, ajouter tous les mots du dictionnaire dans un filtre,

puis, pour chaque mot w du dictionnaire, tester la présence d'un mot wc où c est un caractère qui n'apparaît dans aucun mot (par exemple un caractère non alphabétique comme `\n`). Compter les faux positifs et commenter le résultat.