IMAC
INGÉNIEUR
ESIPE

Algorithmique Avancé

TP 5

Hachage et Tableau associatif

Les structures de données et les tris vu précédemment permettent de retrouver facilement des données spécifiques. Cependant jusqu'ici vous n'avez cherché que des nombres à l'aide d'autres nombres, ce qui a finalement peu d'intérêt. Il existe des structures de données qui permettent de stocker des éléments en leurs associant des "**clés**" facilement retrouvable, par exemple des nombres dont on connait déjà des méthodes de recherche.

1 Hachage

Le **hachage** est une technique qui permet de calculer un entier à partir d'un élément qui n'en est pas un, par exemple une chaîne de caractères. Le but étant de manipuler plus facilement ces éléments, notamment de les comparer plus facilement.
On peut notamment calculer un indice à partir d'un élément pour le stocker dans un tableau, ceci permettant de le retrouver facilement plus tard.

```c
#define ARRAY_SIZE 40
int hash(char* string)
{
  int i = 0;
  int hash_value = 0;
  while(string[i] != '\0')
  {
    hash_value += (int)string[i];
    i++;
  }
  return hash_value % ARRAY_SIZE;
}

int main()
{
  char* hash_table[ARRAY_SIZE];
  char* my_element = "Yolo\0";
  hash_table[hash(my_element)] = my_element;
  printf("Content of hash_table at %d is %s\n", hash(my_element), my_element);
  // Content of hash_table at 19 is Yolo
}
```

En ayant un hachage en $O(1)$, on obtient une insertion et une recherche instantané. Les tableaux basés sur le hachage sont des **tables de hachages**, chaque indice correspond à un code (une **valeur de hachage**). Dans l'exemple ci-dessus, on obtient facilement des **conflits**, des chaines de caractères qui ont la même valeur de hachage, qui doivent être stockés dans la même case du tableau. On se

retrouve devant deux solutions selon la stratégie de stockage, soit on écrase la valeur courante, soit on stocke tous les éléments au même endroit dans une liste.

2 Tableau associatif

Aussi appelé **Map**, il s'agit d'une structure qui pour une clé donnée retrouve un élément. La clé doit être un élément dont on peut calculer une valeur de hachage unique. En stockant un élément avec sa clé, on peut le retrouver en utilisant une recherche rapide ou un arbre binaire de recherche.

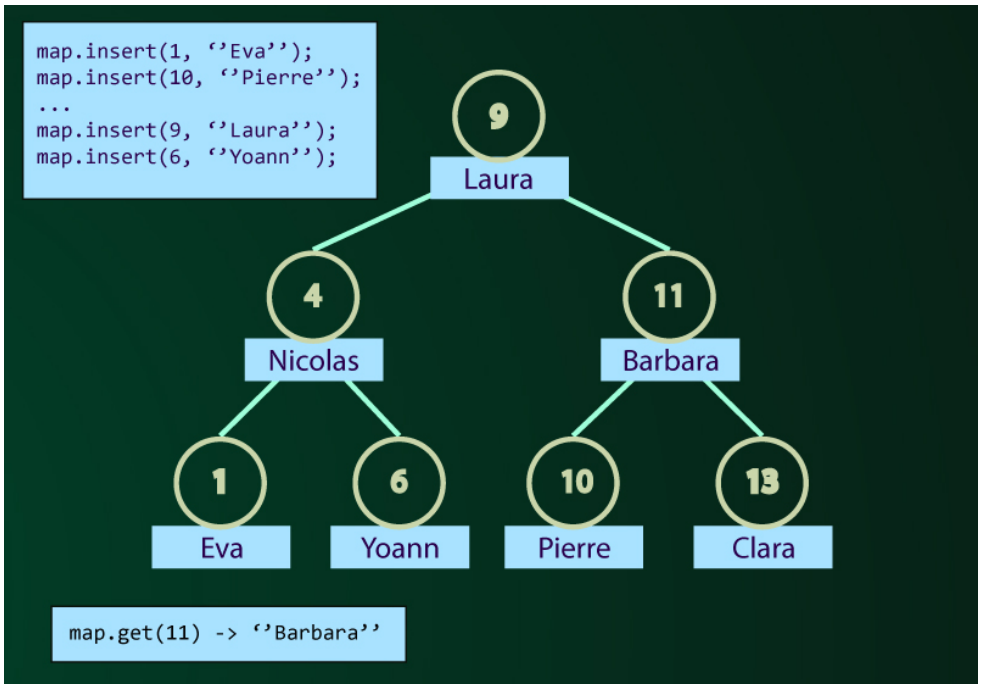

FIGURE 1 – Exemple de tableau associatif basé sur un arbre binaire de recherche

Les tableaux associatifs permettent de stocker et récupérer des éléments en $O(log(n))$. On peut les stocker avec des nombres entier, à virgules, des chaines de caractères ou tous types qui peuvent être comparés rapidement.

3 TP

Implémenter les fonctions d'une table de hachage stockant des chaines de caractères. La valeur de hachage d'une chaine est le code ascii de son premier caractère, si cette valeur dépasse la taille du tableau, prenez le reste de la division de cette valeur par la taille du tableau.

- **hash**(string $word$) : Retourne la valeur de hachage de $word$.
- **insert**(string $word$) : Insère l'élément $word$ dans la table à l'indice correspondant à sa valeur de hachage. S'il y a déjà un élément, écraser cet élément.
- **contains**(string $word$) : Retourne Vrai si l'élément $word$ se trouve dans la table, Faux autrement.

Réimplémenter ces fonctions mais en stockant des listes chainés pour stocker plusieurs chaines de caractères avec la même valeur de hash.

Implémenter les fonctions suivantes pour implémenter un tableau associatif stockant des entiers indexés par des chaines de caractères. Utilisez le hachage polynomial pour obtenir des clés uniques à partir de chaine de caractères. Considérons qu'on utilise que les 128 premiers caractères de l'encodage ascii.

- **hash**(string key) : Retourne la valeur de hachage de key.
- **insert**(string key, int $value$) : Insère $value$ en utilisant pour clé $value$.
- **get**(string key) : Retourne la valeur correspondant à key. Retourne 0 si aucune valeur de correspond à key.

Vous pouvez l'implémenter avec un arbre binaire ou un tableau dynamique.
Vous pouvez utiliser le langage que vous souhaitez.

3.1 C++

Le dossier *Algorithme_TP5/TP* contient un dossier *C++*. Vous trouverez dans ce dossier des fichiers *exo<i>.pro* à ouvrir avec *QtCreator*, chacun de ces fichiers projets sont associés à un fichier *exo<i>.cpp* à compléter pour implémenter les différentes fonctions ci-dessus.
L'exercice *exo1.cpp* implémente une structure *HashTable* possédant les différentes méthodes d'une table de hachage à implémenter. Cette structure est semblable à *Array*, il possède les mêmes fonctions d'accès que lui mais manipule des $strings$ plutôt que des int.

```
class HashTable {

  void print(); // declaration de la methode print de HashTable

}

void HashTable::print() // corps de la methode print de HashTable
{
  for (i=0; i < this->size(); ++i)
    printf("%d ", this->get(i));
}
```

Notes :

— Dans une fonction C_{++}, si le type d'un paramètre est accompagné d'un '&' alors il s'agit d'un paramètre entré/sortie. La modification du paramètre se répercute en dehors de la fonction.
— Lorsque vous faites appel à $this$ dans une méthode d'une structure (ou d'une classe), vous pouvez accéder aux attributs de la structure en question, comme dans l'exemple ci-dessus.