Laboratoire : Les tables de hachage

1 Objectifs

Le but de ce laboratoire est de découvrir les tables de hachage en utilisant deux techniques de résolution de collisions par adressage ouvert : la redistribution quadratique et le double hachage.

2 Travail à faire

Vous devez implémenter les deux classes de table de hachage que nous vous fournissons avec cet énoncé. La première classe utilise la redistribution quadratique pour la résolution des collisions, et l'autre le double hachage. Les classes utilisent des *foncteurs* pour les fonctions de hachage. De plus, la taille de la table doit toujours être un nombre premier. Nous vous fournissons deux méthodes privées nécessaires pour trouver le prochain nombre premier suivant un certain entier.

Vous devez également tester la performance des tables de hachage. Pour ce faire, vous devez utiliser la méthode `statistiques()` et sa méthode utilitaire privée `_statistiques`. Vous n'avez à évaluer les statistiques des collisions que pour la méthode d'insertion. De plus, nous vous offrons deux foncteurs de hachage pour le type `int`, et deux autres pour le type `string`. Vous devez implémenter deux nouveaux foncteurs de `int` et deux nouveaux pour `string`. Vous aurez alors en main 8 foncteurs.

Vous devez faire les deux tests suivants pour chacune des deux classe :

- Utilisez le *Google Test* intitulé `fluxEnleverAjouterOk`, qui teste l'ajout aléatoire de clés de type `int`, avec les 4 foncteurs de `int` et afficher les statistiques. Que remarquez-vous ?
- Basez-vous sur `fluxEnleverAjouterOk` pour écrire un test qui fait l'ajout aléatoire de clés de type `string`. Il y a un générateur de chaînes aléatoires plus bas dans cet énoncé. Utilisez les 4 foncteurs de `string` et affichez les statistiques. Que remarquez-vous ?

3 Astuce

Pour vous aider, nous vous proposons d'implémenter les deux méthodes suivantes. Si vous implémentez ces deux méthodes de la manière décrite, vous pourrez alors les réutiliser pour résoudre la plupart des autres méthodes à implémenter.

```
// Cette méthode recherche dans la table une position libre, c'est à
// dire une position vacante ou une position effacée. Elle retourne la
// première position libre trouvée en utilisant la redispersion choisie.
size_t _trouverPositionLibre(const TypeClef &) const;

// Cette méthode recherche dans la table la position d'une clé, c'est à
// dire une position qui contient la clé, qu'elle soit effacée ou
// occupée, ou une position vacante. Elle retourne la première position
// clé trouvée en utilisant la redistribution choisie.
size_t _trouverPositionClef(const TypeClef &) const;
```

À noter que de cette manière, la redistribution est assurée de ne pas boucler à l'infini si lia table contient des positions vacantes qui sont accessibles par la redistribution.

4 Double hachage

Le double hachage effectue ceci : $\Big(h_1(\text{clé}) + i \times h_2(\text{clé})\Big) \bmod \text{m\_tab.taille}()$.

Ici, il y a un risque de boucle infinie si $h_1(\text{clé})$ est occupée et si $h_2(\text{clé}) \bmod \text{m\_tab.taille}() = 0$. Pourquoi ? Vous pouvez gérer ce cas en utilisant $h_2(\text{clé}) = 1$.

5 Générateur de chaînes aléatoires

Vous pouvez utiliser ce code pour générer des chaînes de caractères aléatoirement :

```
string alphanum = "1234567890abcdefghijklmnopqrstuvwxyz"
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#%&*()[]{}.,;:^";

string clef;
int N = 10;
srand(time(NULL));

for (int i = 0; i < N; ++i) {
    clef.push_back(alphanum[rand() % alphanum.size()]);
}
```

6 Documentation

Voir la section Documentation/Normes sur le site Web du cours. Vous y trouverez la description des commentaires attendus dans un programme ainsi que des normes de programmation en vigueur dans notre cours.

7 Important

1. Nous vous fournissons des tests unitaires *Google Test* que vous pouvez utiliser pour vérifier votre implémentation.
2. Vous êtes tenu de faire la gestion des exceptions dans les méthodes que vous avez à implémenter. Référez-vous à la documentation *Doxygen* fournie avec l'énoncé pour connaître les types d'exception que vous avez à gérer pour chacune d'elles. Vous devez utiliser le cadre de la théorie du contrat que nous avons préparé dans les fichiers `ContratException.cpp` et `ContratException.h` disponibles sur le site du cours.
3. Vous devez documenter chaque méthode, que vous avez à implémenter, avec les commentaires de *Doxygen*. Référez-vous à section de Doxygen sur le site Web du cours ainsi qu'aux exemples de cette semaine pour découvrir les commentaires que vous avez à écrire. Assurez-vous de respecter les normes de programmation en vigueur disponibles également sur le site du cours.
4. Vous devez générer la documentation en format HTML. À cette fin, nous vous fournissons un fichier de configuration `sdd.doxyfile` que vous pouvez utiliser tel quel.
5. Nous vous encourageons à utiliser au maximum la partie privée de la classe afin d'y ajouter des fonctions utilitaires privées. Elles permettent d'augmenter la lisibilité du code et de réduire la duplication de code identique.

Bon travail !