IFT339 Automne 2014 Laboratoire 5

Les conteneurs de recherche sans ordre

Objectifs de ce laboratoire

Dans ce laboratoire, nous allons implanter un ensemble sans ordre avec les fonctionnalités de base du `unordered_set` de la bibliothèque normalisée.

Description de la tâche à réaliser

Je vous donne le code de base, déjà fonctionnel, et je vous demande de le compléter. Les fonctions qui manquent sont:

erase reçoit un itérateur d'`ensemble_sans_ordre` et doit retirer l'élément à cette position

reculer une fonction privée de la classe des itérateurs d'`ensemble_sans_ordre` qui passe à la position précédente (utilisée dans les opérateurs `--`) pour pouvoir itérer à reculons

rehash la fonction qui fait une copie complète de l'ensemble quand le facteur de charge est dépassé. Cette fonction est appelée au début de la fonction d'insertion. Pour qu'elle ne fasse pas dégénérer la complexité de la classe, une approche simple est de doubler la dimension à chaque fois que cette opération est nécessaire. Pour unifier avec la situation initiale où la dimension est 0, le plus simple est de prendre deux fois la dimension courante plus 1.

Bien sûr, on aurait pu faire plus complexe un peu. Par exemple, cette classe ne donne pas accès à l'usager aux alvéoles individuelles, ni au nombre d'alvéoles, comme la classe `unordered_set` le fait. Tout est encapsulé dans la classe.

La représentation de l'ensemble

Il y a trois membres dans la représentation de l'ensemble. `SIZE` et `FACTEUR_DE_CHARGE` donnent le nombre d'éléments dans l'ensemble et le nombre maximal moyen par alvéole acceptable. Par exemple, si le facteur est 1.2 et qu'il y a 17 alvéoles, on accepte qu'il y ait 20 éléments dans l'ensemble. Si on a 21 éléments au début du processus d'insertion, on procède à la réorganisation de l'ensemble AVANT de faire l'insertion. Pourquoi avant plutôt qu'après? Pour unifier avec le cas initial, où on aura 0 éléments et 0 alvéoles. Avec un facteur de 1.0 par exemple, on aura que 1.0*0 n'est pas inférieur à 0 éléments, ce qui amènera à réorganiser pour avoir une alvéole, donc suffisamment pour faire notre insertion. Encore un exemple d'unification du cas limite avec le cas général. Il n'y a aucun endroit où on doit se préoccuper des cas limites de 0 éléments ou de 0 alvéoles.

Le tableau lui-même est un `vector` de la SL. Les éléments du `vector` sont des `list` de la SL aussi. Les éléments de ces listes sont des pointeurs vers les objets insérés dans l'ensemble. On a donc encore une représentation par handle. Pourquoi? Parce que les normes de la SL réclament qu'un pointeur ou une référence à un objet de l'ensemble ne puisse pas être invalidé par une réorganisation. On ne doit donc pas bouger un élément une fois qu'il a été inséré. L'ensemble est donc représenté de la façon suivante (voir les dessins au verso)

```
std::vector<std::list<TYPE*>> REP;
size_t SIZE;
float FACTEUR_DE_CHARGE;
```

Quant aux itérateurs d'ensemble, on a besoin de trois information pour les représenter : à quel ensemble ils sont associés, dans quelle alvéole, et à quelle position dans l'alvéole. Pour la position de la fin, `ALV` est égal à `REP.size()`.

```
ensemble_sans_ordre<TYPE>* ENS;
size_t ALV;
typename std::list<TYPE*>::iterator POS;
```

Remise du travail

Ce travail doit être complété à 23 h 59 jeudi le 4 décembre au plus tard. Soumettez-le à `turnin`, après vous être assurés qu'il fonctionne bien sur tarin. Ne soumettez pas un répertoire, et attention à la syntaxe! Une seule soumission par équipe! Ne soumettez que votre fichier ensemble.h. J'utiliserai mon propre programme principal. Ne changez pas le nom!

```
turnin -c ift339 -p labo5 ensemble.h
```

Jean Goulet

Dans l'exemple illustré ci-dessous:

E est un `ensemble_sans_ordre` qui contient 8 éléments dans 5 alvéoles (le nombre d'éléments du vecteur). Il admet un facteur de charge de 2.0. Tant que le nombre d'éléments demeure strictement inférieur à 5*2.0 (donc jusqu'à 9 éléments) on ne modifiera pas le nombre d'alvéoles. Si le nombre est à 10 au moment où on s'apprête à faire la prochaine insertion, on réorganise. À ce moment, le nombre d'alvéoles passe à 11 (5*2+1). Aucun élément ne doit alors être déplacé. On peut par exemple créer un nouveau `vector`, et y insérer les pointeurs vers les éléments existants en les enlevant des anciennes listes. Si on faisait une "vraie" application, on s'occuperait aussi de réduire le nombre d'alvéoles quand on a enlevé beaucoup d'éléments.

L'itérateur `i` représente la position de l'élément 5. Il y a trois éléments dans cette représentation : un pointeur vers l'ensemble, le numéro de l'alvéole qui contient cet élément (3) et un itérateur de liste qui pointe sur l'élément de la liste qui contient le pointeur vers 5. On pourrait penser que seul le troisième élément est utile. Les deux autres le sont quand vient le temps d'avancer à la prochaine position dans l'ensemble. Tant qu'on reste dans la même alvéole, il n'y a pas de problème. Dès qu'on arrive à la fin de l'alvéole, il faut pouvoir le détecter, et il faut qu'on sache dans quelle alvéole aller ensuite. Comme on doit sauter les alvéoles vides quand on avance dans l'ensemble, il faut savoir combien il y a d'alvéoles au total dans l'ensemble. La meilleure façon de le savoir est de pouvoir le demander directement à l'ensemble auquel est associé cet itérateur. C'est à cela que sert le pointeur vers l'ensemble.

L'itérateur `j` représente la position de la fin. Dans cette représentation, on a encore un pointeur vers l'ensemble concerné, et le numéro de l'alvéole correspond au nombre total d'alvéoles dans cet ensemble, donc à la position qui suit la dernière.