IFT339 Hiver 2022 Laboratoire 6 (en équipe)

Un multi-ensemble sans ordre

Objectifs de ce laboratoire

Implanter un multi-ensemble dont la fonctionnalité est proche du multiset de la SL, avec les fonctions génératrices O(1) amorti par l'utilisation d'un tableau d'adressage dispersé.

Description de la tâche à réaliser

On vous fournit le code de la classe générique unordered_multiset basée sur un tableau d'adressage dispersé. Nous utiliserons un tableau d'alvéoles. Aux fins de cet exercice, une alvéole sera implantée avec une liste de la SL.

Au départ, on n'a qu'une seule alvéole vide. À mesure que l'on ajoute des éléments, quand le temps de recherche devient trop long, on ajoute des alvéoles. Une stratégie simple sera de toujours avoir un nombre d'alvéoles égal à une puissance de 2 moins 1, pour assurer une meilleure dispersion si jamais la fonction de hachage n'est pas très bonne pour faire cette dispersion. En effet, la fonction de dispersion nous retourne unsize_t sur lequel nous n'avons pas beaucoup de contrôle. En prenant cette valeur modulo une puissance de 2 moins 1, on contribue à améliorer la dispersion. La représentation sera donc constituée des éléments suivants:

```
size_t SIZE;
vector<list<TYPE>*> REP;
double facteur_min,facteur_max;
classe_de_dispersion disperseur;
```

En plus du tableau d'adressage dispersé et du nombre total d'éléments dans le multi-ensemble, la partie privée de cette classe contient deux facteurs de charge. Le facteur maximal représente le nombre moyen maximal d'éléments par alvéole. Lorsque cette moyenne est dépassée on doit procéder au doublement du nombre d'alvéoles. Ce doublement implique de passer de N à 2N pour conserver la séquence 1 3 7 15 31 etc.d'alvéoles utiles pour l'ensemble. De la même façon on réduit la dimension du tableau s'il devient trop éparpillé en passant de 64 à 32 16 8. etc. Il est éparpillé si le nombre moyen d'éléments par alvéole est inférieur à facteur_min.

La représentation contient aussi un “disperseur” soit un objet disposant d'une méthode operator()(TYPE) qui retourne un size_t servant à la dispersion, comme vu plus en détail en classe. On peut prendre soit le disperseur fourni par C++ dans la portée std (`std::hash<TYPE>`) ou donner une classe de dispersion comme second paramètre de la déclaration de classe. Le disperseur est utilisé pour l'insertion, l'élimination et la redispersion (rehash). L'option par défaut est le disperseur de la portée std.

Et l'itération? L'itérateur doit être autonome pour avancer au prochain élément du tableau d'adressage dispersé. Il doit donc pouvoir avancer dans une alvéole pour passer à l'élément suivant, reconnaitre quand il est rendu à la fin de cette alvéole pour passer à l'alvéole suivante non vide, et il doit faire l'équivalent pour reculer dans l'ensemble. Pour cela, il doit disposer de deux éléments d'information : un itérateur identifiant la position de l'alvéole dans le tableau (ALV) et un itérateur identifiant la position à l'intérieur de l'alvéole courante (POS) :

```
typename vector<list<TYPE>*>::iterator ALV;
typename forward_list<TYPE>::iterator POS;
```

Notez que le tableau contient des pointeurs vers des listes dynamiques, et non des listes. Avec cette information, on peut avancer ou reculer. Vous avez déjà une partie de la classe iterator de codée, il ne manque en fait que les fonctions privées avancer() et reculer(). Tout le reste est codé. Les fonctions que vous devez coder sont donc : avancer, reculer, insert, erase (les deux versions), count, find et rehash. Une description succincte de ces fontions vous est donnée au verso. Vous pouvez aussi aller voir la description formelle de ces fonctions sur le site cplusplus :

https://www.cplusplus.com/reference/unordered_set/unordered_multiset/

Remise du travail

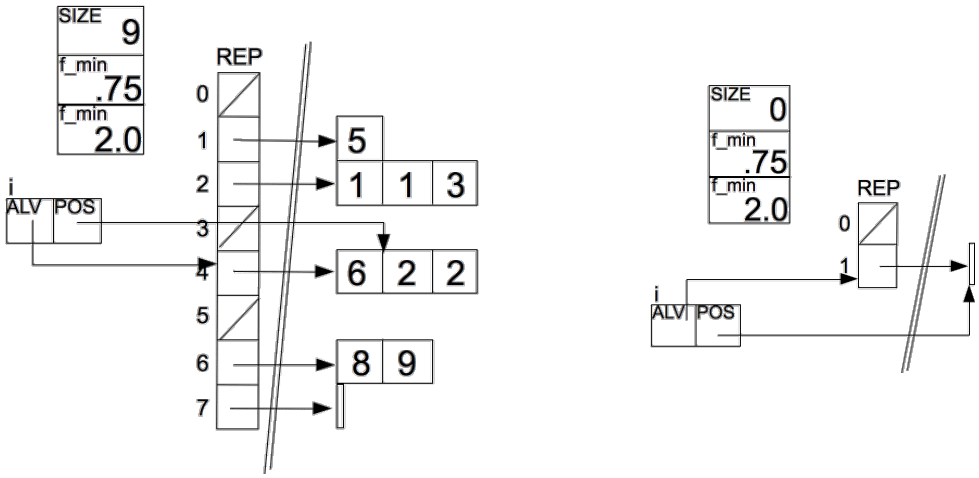
Ce travail doit être complété à 23 h 59 le mercredi 13 avril, dernier jour de cours de la session. Soumettez-le à `turnin`, après vous être assurés qu'il fonctionne bien sur tarin. Une seule soumission par équipe, dans le compte de votre choix! Ne soumettez que le fichier unordered_set_2.h, qui contiendra les définitions des fonctions que vous devez coder. Vous n'avez pas à modifier la définition de la classe ni les autres fonctions. J'utiliserai mon propre programme principal. Comme pour les labos précédents, je vous fournis une fonction afficher() que vous pouvez utiliser pour tester en partie votre code.

Jean Goulet

Voici quelques illustrations d'un tel multiensemble. L'objectif ici est de manipuler une hiérarchie complexe d'objets. Il y a quatre éléments dans la représentation. SIZE nous donne le nombre d'éléments dans le multiensemble. facteur_min et facteur_max nous donnent les ratios minimums et maximums acceptables, pourvu bien sûr qu'il y ait au moins un élément dans le multiensemble!. Pour les fins de ce labo, on a fixé ces valeurs dans le constructeur, sans possibilité pour l'usager de modifier ces éléments. Ce serait facile à adapter, comme pour le multiset de la SL. L'élément principal de la représentation est un vector. Ce vecteur contient soit des pointeurs nuls pour les alvéoles vides, soit des pointeurs vers des listes non vides. Le dernier élément de ce vecteur est une liste vide, qui sert à représenter la position de la fin. Il est important de ne jamais aller ajouter d'éléments dans cette liste, car alors, on ne pourrait plus reconnaître la fin du multiensemble.

Vous avez d'abord l'illustration du cas général d'un ensemble qui contient 9 éléments dans 7 alvéoles. On utilise une huitième alvéole pour représenter la position de la fin de façon unifiée avec les autres positions. Les alvéoles vides sont représentées par un pointeur nul. La dernière alvéole n'est pas vide, mais la liste sur laquelle elle pointe est une liste vide. En effet, on ne veut pas aller déréférencer un itérateur qui nous donnerait cette position. On illustre aussi un itérateur de multiset. C'est un couple d'itérateurs : le premier (ALV) donne accès à l'alvéole (c'est un itérateur de vecteur de pointeurs de liste). Le second (POS) donne la position de l'élément à l'intérieur de l'alvéole. Dans le premier cas, l'itérateur nous donne la position du premier "2".

Vous avez aussi une illustration d'un multiensemble vide. Dans ce cas, il n'y a qu'une seule alvéole utilisable, plus celle servant à représenter la fin. On y voit un itérateur vers la fin bien sûr, puisque c'est la seule qui existe!



Les fonctions que vous devez coder sont (comme d'habitude, ne vous occupez pas de robustesse) :

rehash(n): modifie la dimension du tableau d'adressage dispersé et replace tous les éléments à un nouvel endroit en conséquence. Ce nombre INCLUT l'alvéole supplémentaire qui ne sert que pour identifier la fin du tableau d'adressage dispersé.

avancer: une fonction qui avance l'itérateur implicite à la prochaine position

reculer: une fonction qui recule l'itérateur implicite à la prochaine position.

insert(val): ajoute un exemplaire de la valeur val à la bonne place dans la bonne alvéole (tous les éléments égaux doivent être regroupés ensembles dans l'alvéole (retour: itérateur vers l'élément inséré)

erase(val): enlève tous les exemplaires de la valeur val dans l'ensemble (retour: nombre d'éléments enlevés)

erase(i): enlève l'objet en position i dans l'ensemble (retour: itérateur vers la prochaine position)

count(val): retourne le nombre d'occurrences de VAL dans l'ensemble

find(val):retourne un itérateur vers la première occurrence de val dans l'ensemble, ou la fin.