Skip lists

1 Type abstrait ensemble (set)

Un ensemble est une structure de données où l'on souhaite réaliser les opérations suivantes :
- ajouter un élément à l'ensemble ;
- déterminer si un élément est présent dans l'ensemble ;
- retirer un élément de l'ensemble.

Lorsqu'un même élément est ajouté plusieurs fois à un ensemble, un seul exemplaire est conservé, les ajouts suivants sont ignorés.

Il est possible d'envisager plusieurs implémentations naïves pour le type abstrait ensemble :

tableau dynamique : les éléments sont ajoutés un par un dans un tableau. Dans ce cas, pour toutes les opérations, il est nécessaire de commencer par chercher l'élément dans le tableau (pour ne pas créer de doublons et savoir d'où le retirer). L'absence de structure du tableau fait que cette opération est réalisable en $O(n)$. Si l'on sait d'où retirer l'élément, la suppression se fait en temps constant $O(1)$ car il suffit de remplacer l'élément supprimé par le dernier du tableau.

tableau dynamique trié : le tableau est maintenu trié au fur et à mesure des insertions. Dans ce cas la recherche peut se faire par dichotomie en $O(\log n)$. Par contre l'insertion et la suppression doivent maintenir la structure triée, ce qui nécessite de décaler des éléments dans le tableau. Ces deux opérations sont donc en $O(n)$

liste chaînée : lorsque l'on sait où insérer ou supprimer dans une liste chaînée, l'opération peut se faire en temps constant $O(1)$. Par contre, pour localiser un élément, même lorsque la liste est triée, il est nécessaire de parcourir linéairement toute la liste, en $O(n)$.

Les skip-lists se basent sur les listes chaînées, mais visent à réaliser toutes ces opérations en temps logarithmique $O(\log n)$. Ce ne sont pas les seules structures qui le permettent, les arbres binaires équilibrés y parviennent aussi, et les tables de hashage réalisent toutes ces opérations en temps constant amorti $O(1)$. Pour comparer ces structures entre elles, il est donc nécessaire de regarder les autres opérations souhaitées sur l'ensemble. Par exemple, les tables de hashage basiques ne permettent pas de lister les éléments insérés. Les skip-lists sont réputées intéressantes pour l'accès concurrent : lorsque plusieurs programmes veulent consulter ou modifier les données en même temps [1]. Elles sont utilisées dans des projets à grande échelle, et parfois combinées à d'autres structures de données [2].

2 Skip-lists pour le type ensemble

Le principe des skip-lists est d'améliorer le parcours linéaire d'une liste chaînée en ajoutant des *raccourcis*. Ces raccourcis permettent en gros d'utiliser la dichotomie sur une liste chaînée. Lorsque les éléments de la skip-list sont triés, il est donc possible d'exploiter cette propriété pour chercher les éléments en temps logarithmique $O(\log n)$.

2.1 Raccourcis

Pour ajouter des raccourcis, les skip-lists rajoutent la notion de *niveau* par rapport aux liste chaînées classiques. Chaque niveau est une liste chaînée, et le niveau 0 est équivalent à la liste chaînée contenant tous les éléments de la skip-list. Le niveau 1 contient en moyenne un élément du niveau 0 sur deux, dans le même ordre. Le niveau 2 contient en moyenne un élément sur deux du niveau 1, dans le même ordre. Et ainsi de suite jusqu'à ce qu'il n'y ait plus d'élément. Chacun de ces niveau correspond aux raccourcis : en se déplaçant sur la liste de

1. `https://www.cs.tau.ac.il/~shanir/nir-pubs-web/Papers/OPODIS2006-BA.pdf`
2. `https://redis.io/topics/data-types-intro`

niveau 1, on se déplace deux fois plus vite que sur la liste de niveau 0. De même le niveau 2 va deux fois plus vite que le niveau 1. Le parcours d'une skip-list permet donc de faire deux choses :

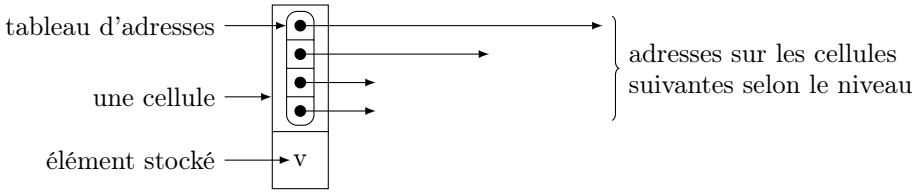
— accéder à l'élément suivant du niveau courant (on garde le niveau, on change l'élément) ;
— accéder à l'élément courant au niveau du dessous (on change le niveau, on garde l'élément).

2.2 Skip list parfaite

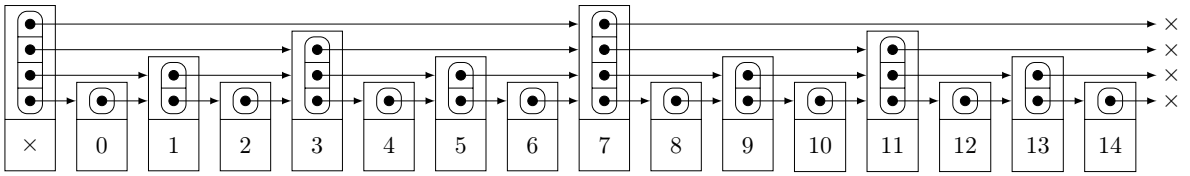
Il existe plusieurs manières d'implémenter les skip lists. Nous vous proposons la méthode suivante à base de tableaux, et utilisant une cellule « bidon » :

— chaque élément d'une skip list est associé à une cellule ;
— chaque cellule contient sa valeur, et un tableau aussi grand que le nombre de niveaux où elle apparaît ;
— pour chaque niveau, le tableau stocke l'adresse de la cellule suivante sur ce niveau ;
— une cellule bidon possède les adresses des têtes de tous les niveaux ;
— la skip list possède l'adresse de la cellule bidon.

Les adresses de chaque cellule sont stockées dans des tableaux : la cellule d'un élément apparaissant sur quatre niveaux contient un tableau de quatre adresses. Une cellule de la skip list sera ainsi représentée ainsi :



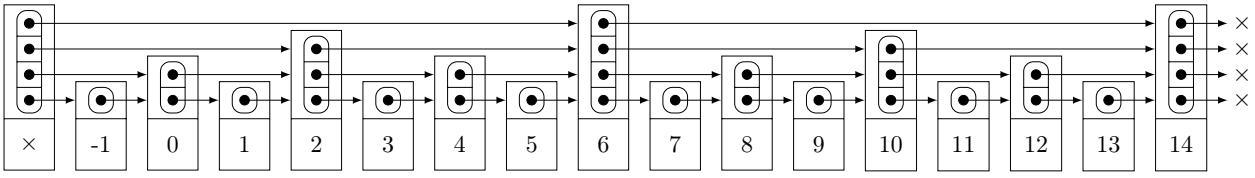
Muni de cette représentation, une skip list parfaite permettant de faire de la dichotomie et d'accéder à tout élément en $O(\log n)$ aurait la forme suivante :



Notez bien que les cases des tableaux contiennent les adresses des *cellules* et non des cases de tableaux d'autres cellules. Dans cette liste parfaite, le niveau 1 contient exactement une cellule sur deux du niveau 0, le niveau 2 contient exactement une cellule sur deux du niveau 1, ...

2.3 Maintenir une structure parfaite ?

Tout l'intérêt des listes chaînées par rapport aux tableaux est de pouvoir insérer et supprimer des éléments n'importe où dans la liste sans avoir à déplacer tout le reste des éléments. Maintenir une skip list parfaite compliquerait énormément cette tâche. Pour conserver l'équilibre parfait d'un élément sur deux entre les niveau, une simple insertion en tête de liste changerait les niveaux de tous les éléments de la liste, et forcerait donc à reparcourir toute la liste pour mettre à jour toutes les adresses. Par exemple en insérant l'élément -1 en tête de la skip list précédente, nous obtiendrions une skip list dont le niveau 1 aura complètement changé, ainsi que les suivants :



Maintenir en place une skip list parfaite est donc trop coûteux. C'est pourquoi les skip lists utilisent un processus de *randomisation* pour s'assurer de bonnes propriétés en moyennes sans avoir à préserver une structure parfaite.

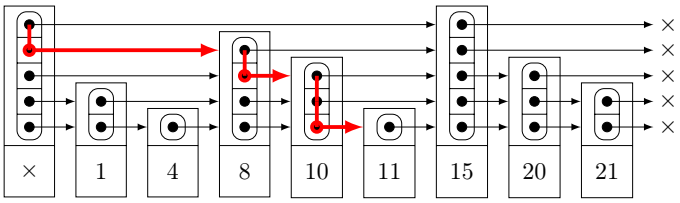
2.4 Randomisation

Dans une skip list parfaite, le nombre de niveaux d'un élément dépend de sa position dans la liste. Dans une skip list randomisée, le nombre de niveaux d'un élément est tiré aléatoirement via un mécanisme de pile ou face, indépendamment de la taille de la liste ou des niveaux des autres éléments. Tout élément apparaît au niveau 0. Pour déterminer si un élément apparaît sur le niveau 1, nous tirons à pile ou face. Si nous obtenons pile, l'élément est conservé pour le niveau 1, sinon il reste de niveau 0. Pour savoir si un élément présent au niveau 1 est présent au niveau 2, nous tirons à nouveau à pile ou face, et ainsi de suite. Le nombre de niveaux auquel apparaît un élément est donc un plus le nombre de pile successifs que nous avons obtenus. Si le nombre de niveaux d'un élément dépasse le nombre de niveaux actuellement présents dans la liste, des niveaux supplémentaires sont ajoutés à la cellule bidon, et chacun de ces niveaux pointe sur la cellule de l'élément.

L'insertion d'un nouvel élément consiste donc à :
- tirer à pile ou face le nombre de niveaux sur lequel il est présent ;
- ajouter éventuellement des niveaux à la cellule bidon ;
- déterminer les cellules (éventuellement bidon) précédentes sur chaque niveau où il apparaît ;
- insérer la nouvelle cellule à sa place dans chacun des niveaux.

2.5 Parcours

Comme mentionné auparavant, le parcours d'une skip-list comporte deux éléments : avancer sur le niveau courant, et descendre d'un niveau. Pour utiliser au maximum les raccourcis, le parcours commence toujours au niveau le plus élevé, car c'est à ce niveau qu'on avance le plus vite sur la liste. Pour ne rater aucun élément, le parcours commence sur la cellule bidon qui est en tête et appartient à tous les niveaux. Pour localiser 11 dans la skip-list triée suivante, l'algorithme part de la cellule bidon au niveau 4 (la liste comporte 5 niveaux numérotés de 0 à 4). Le parcours suit ensuite le principe suivant :
- si la valeur de l'élément est celle de l'élément recherché, retourner vrai ;
- sinon si la cellule suivante existe et sa valeur est plus petit que l'élément cherché, on avance ;
- sinon si on peut descendre on descend d'un niveau ;
- sinon l'élément n'est pas présent dans la liste.



Votre parcours ressemblera toujours à un escalier de ce type.

3 Votre travail : skip list triée

Votre travail consiste à implémenter une structure de skip list randomisée permettant de stocker des entiers. Lorsque vous insérerez un élément dans la skip list, vous vous assurerez que les éléments restent triés. Vous programmerez les opérations d'insertion, de suppression et de recherche d'élément dans la skip list.

3.1 Structure de donnée

Reprenez votre travail sur les listes chaînées, et créez une structure cellule pour contenir les éléments nécessaires au fonctionnement d'une skip list :
- la valeur contenue dans la cellule ;
- un tableau de suivants.

Il s'agit en gros de remplacer l'adresse de la cellule suivante par un tableau d'adresses des cellules suivantes à chaque niveau. Ainsi si dans une liste chaînée on accédait à la cellule suivante d'une cellule `c` via `c->suivante`, dans une skip list on accède au suivant au niveau `k` via `c->suivante[k]`. Pour ce tableau, vous pouvez utiliser les `std::vector`[3] de la librairie standard, qui ont la possibilité de grandir, ce qui sera pratique pour la cellule bidon en tête.

3. `http://www.cplusplus.com/reference/vector/vector/`

Pour la liste en elle-même, prévoyez initialement le constructeur pour une liste vide, le destructeur, l'insertion et la recherche. Vous ajouterez ensuite l'opérateur d'affectation et le constructeur par copie lorsque le reste sera fonctionnel.

3.2 Insertion

Pour réaliser l'insertion, il est nécessaire de bien comprendre comment parcourir une skip list. Pour se placer sur une skip list, il est nécessaire de savoir sur quelle cellule on se place et à quel niveau. Si `c` correspond à l'adresse de la cellule courante, et `k` correspond au niveau courant, il est alors possible de

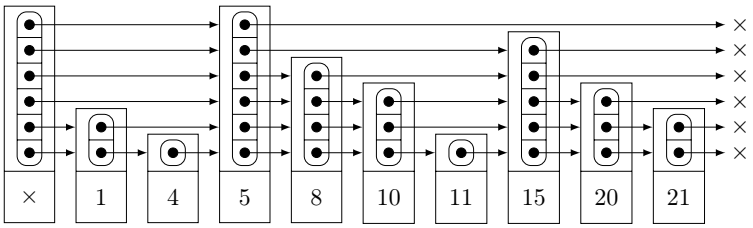
— passer à la cellule suivante sur le même niveau via `c = c->suivante[k]` ;

— rester sur la même cellule en descendant d'un niveau via `k = k-1` ;

On sortira de la liste si `c == nullptr` ou si `k < 0`.

Dans un premier temps, ne vous souciez pas des doublons. Lorsque le reste fonctionnera, vous ferez en sorte que l'insertion d'une valeur déjà présente dans la skip list n'ait aucun effet.

L'insertion de 5 dans la liste précédente (section 2.5), en supposant que le jeu de pile ou face nous indique que l'élément doit être présent sur six niveaux donnerait :



Ici, le niveau 5 a été créé. Sur le niveau 4, 5 a été inséré entre la cellule bidon et 15. L'adresse stockée précédemment dans la cellule bidon pour le niveau 4 a été recopiée dans 5, et la cellule bidon possède désormais au niveau 4 l'adresse de 5. Au niveau 3, 5 a été inséré entre la cellule bidon et 8. Une nouvelle fois l'adresse convenue précédemment dans la cellule bidon a été recopiée dans 5, et la cellule bidon a pris l'adresse de 5. Au niveau 2, idem qu'au niveau 3. Au niveau 1, 5 a été inséré entre 1 et 8. L'adresse précédemment stockée au niveau 1 dans 1 est désormais recopiée dans 5, et 1 stocke maintenant l'adresse de 5 au niveau 1. Au niveau 0, 5 est inséré entre 4 et 8, l'adresse précédemment stockée dans 4 est maintenant recopiée dans 5, et 4 a l'adresse de 5. À chaque niveau, l'insertion se fait donc comme dans une liste chaînée : il faut connaître la cellule de la liste située juste avant l'insertion, recopier l'adresse de sa suivante dans la cellule insérée, puis remplacer l'adresse de sa suivante par l'adresse de la cellule insérée.

3.3 Recherche

La recherche d'un élément de la skip list est très similaire à l'insertion, le parcours sera réalisé exactement de la même façon. Il suffira de vérifier lors du parcours si la cellule courante contient la valeur cherchée ou non.

3.4 Constructeur par copie et affectation

Un constructeur par copie et un opérateur d'affectation seront automatiquement générés si vous n'en implémentez pas. Leur comportement par défaut consiste à recopier les données bit à bit, ce qui n'est pas souhaitable pour les mêmes raisons que la liste chaînée : les cellules sont alors partagées, et la première liste supprimée détruit les cellules que l'autre utilise. Il vous faudra donc recopier toutes les cellules. Vous n'êtes par contre pas obligés de conserver les mêmes hauteurs pour les cellules, tant que les propriétés statistiques des hauteurs sont préservées. Vous pouvez donc faire appel à la fonction d'insertion pour réaliser la copie.