Lecture et rédaction scientifiques: Skip-lists

Steve Zaretti

1er août 2016

Table des matières

1 Introduction

En informatique, il est courant de vouloir enregistrer des données quelconques. Une structure de données est une représentation logique de ces données. La manière de représenter les données permet de résoudre différents problèmes. Une structure précise peut être plus performante pour un cas précis ou, au contraire, être très peu performante. D'autre part, une meilleur organisation de l'information peut aussi réduire de façon significative la complexité d'un algorithme.

Il existe un grand nombre de structures de données, toutes proposent aux concepteurs de logiciel deux fonctionnalités standards : enregistrer et récupérer un renseignement. Selon l'approche employée pour la représentation logique, les performances seront différentes. Certaines structures de données offrent la possibilité de récupérer le premier élément en temps constant, au détriment du temps d'accès aux autres éléments. Alors que d'autres permettent un accès en temps constant à chacun des éléments en dépit d'une durée plus longue pour l'enregistrement d'une information.

1.1 Liste chainée

Une liste chainée ~~*(figure a)*~~ fig. 1 est une structure de données de taille ~~arbitraire~~variable. Le principe est que chacun des éléments de cette liste contient une référence vers l'élément suivant. Cette pratique facilite l'insertion et la suppression des éléments, au détriment de la rapidité de la recherche.

Une liste chainée peut aussi être triée, ce qui inverse les conséquences sur ses performances. Une insertion devient plus compliquée, car il faut rechercher, avant tout, ou placer l'élément dans la liste afin de garder la structure cohérente.

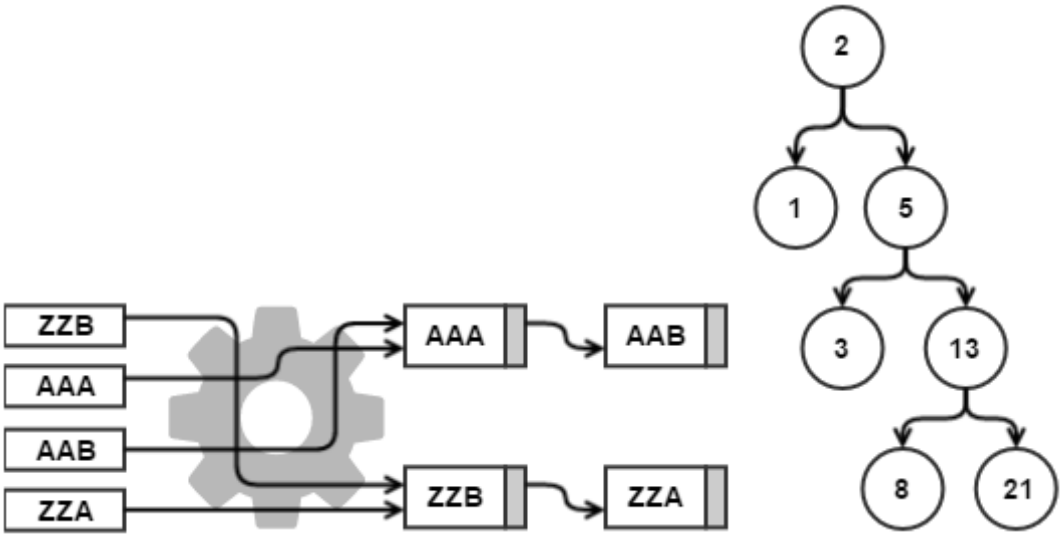
La recherche dans une liste chainée s'effectue élément par élément : si ce n'est pas le premier élément de la liste, il faut regarder le second. Si ce n'est pas celui-ci, il faut alors regarder celui d'après jusqu'à ce que l'élément recherché soit trouvé ou qu'il n'y ait pas d'élément suivant. Dans le cas d'une liste chainée croissante, la recherche peut s'arrêter plus tôt : si l'élément que l'on regarde est plus grand que celui qui est recherché.

FIGURE 1 – Liste chainée ordonnées

Les alternatives

Il existe de nombreuses alternatives aux listes chainées pour trouver un élément plus rapidement : les tables de ~~hashages *(figure b)*~~hashage fig. 2a, les arbres binaires ~~*(figure c)*~~fig. 2b, etc. Mais certaines séquences d'insertions peuvent être catastrophiques. Par exemple, insérer des éléments croissants dans un arbre binaire provoque un rééquilibrage de l'arbre. En revanche, si les éléments sont insérés de façon aléatoire, l'équilibrage devient plus rare.

Grâce aux probabilités, les « skip-list » ~~*(figure d)*~~ fig. 3 bénéficient d'un atout majeur : aucune séquence ne peut provoquer systématiquement le pire scénario. Mieux encore, les skip-lists utilisent des algorithmes plus simples ~~et rapides~~ que ses concurrents. ~~Les skip-lists sont aussi très légères et peuvent être configurées pour n'utiliser que $1\frac{1}{3}$ pointeur par élément.~~



(a) Table de hashage (b) Arbre binaire

FIGURE 2 – Différentes structures de données

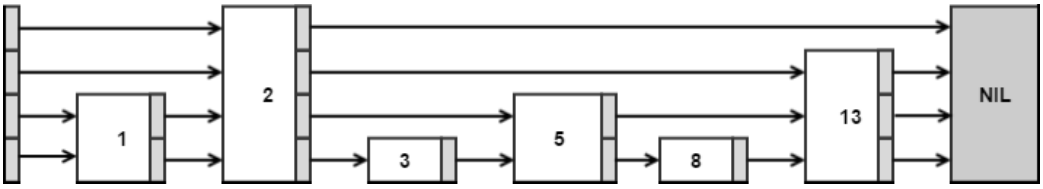


FIGURE 3 – Skip-list

1.2 Skip-List

Tout comme un dictionnaire, une skip-list permet de récupérer la définition d'un terme recherché *(dit clé)*. Cette recherche est rapide, car les éléments *(dit nœud)* présents dans un dictionnaire sont triés ~~, et qu'~~et il est possible de commencer une exploration plus approfondie à un endroit précis. Une skip-list bénéficie des avantages de la liste chainée, en limitant fortement ses inconvénients. Cette structure de données utilise les chaines de façon parallèle. Une fonction basée sur des probabilités permet de déterminer ~~si une nouvelle chainedoit figurer en parallèle ou non. Cette couche supérieure est un moyen plus rapide~~ la hauteur de la chaine. Plus cette hauteur est élevée, moins la liste contient d'éléments. Une couche haute est donc un accès plus rapide vers des éléments plus loin dans la liste.

Afin d'illustrer la skip-liste, il faut l'imaginer comme un réseau de transport en commun idéal où le temps d'attente des correspondances est nul. Un nœud dans la skip-liste est un arrêt obligatoire dans lequel un passager peut changer de moyen de locomotion. La couche la plus basse dans la skip-liste peut-être interprétée comme un bus. Il s'arrête à chacun de ses arrêts prévus, son temps de parcours est très lent. La couche n°2, plus rapide, est un métro. La couche n°3, un train. Si un passager souhaite aller voir un concert à Bruxelles alors qu'il est à la gare de Charleroi, il ne va pas faire le trajet en bus. Il est tout naturel et plus rapide dans ce cas de prendre dans un premier temps le train, puis le métro afin qu'il se rapproche le plus près possible du concert. Si le dernier arrêt de métro n'a pas permis d'accéder à ~~cet élément.~~

~~**Définition**~~

la salle du concert, seulement dans ce cas le passager doit emprunter le bus.

Définition

La hauteur maximale d'une skip-list est définie lors de sa création. Bien qu'il n'y ait pas de taille maximale d'une skip-list, il est conseillé d'utiliser

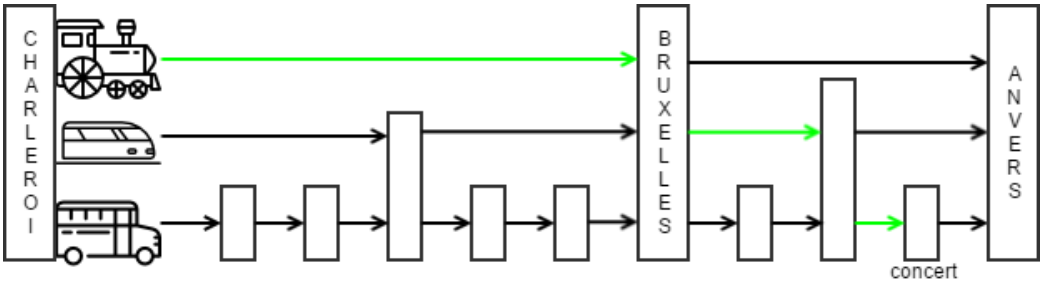


FIGURE 4 – Métaphore d'une skip-liste par un réseau de transport en commun

$log_2(n)$ où n est le nombre d'éléments théoriques présents dans la liste. ~~Le niveau le plus bas de la liste mène~~

L'insertion dans la liste utilise une fonction basée sur la probabilité p afin de définir sa hauteur maximale. Si p vaut $\frac{1}{2}$, ça signifie que le nœud a une probabilité de $\frac{1}{2}$ d'être de hauteur 2, et a une probabilité $\frac{1}{4}$ d'être de hauteur 3. Un nœuds de hauteur h possède h pointeur. Un nœud de hauteur 3 a donc un accès au nœud suivant ~~. Les~~ de hauteur 3, ainsi que celui de hauteur 2 et 1.

En raison des probabilités, les nœuds supérieurs ~~pointent~~ possèdent un pointeur vers un nœud plus avancé dans la liste. Une couche supérieure est ~~une voie rapide vers la couche inférieure~~donc une voie plus rapide. Ainsi, la couche la plus haute contient les plus grands sauts~~.~~, et la couche la plus basse permet d'accéder à chacun des éléments.

Le parcours d'une skip-list se fait de haut en bas, et de gauche à droite~~à gauche~~. Si la clé de l'élément suivant sur une couche est plus grande que celle qu'on recherche, celle-ci continue sur une voie inférieure. ~~L'insertion dans~~

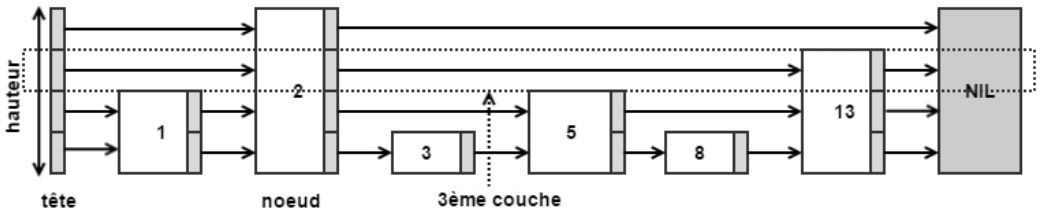


FIGURE 5 – Structure d'une skip-list

1.3 La hauteur

La notion de hauteur est très importante dans une skip-liste. En effet, celle-ci permet de définir à combien de nœuds un élément est attaché. Si trop d'éléments possèdent la même hauteur, la recherche devient trop lente. Cette

méthode est définie à l'aide d'une fonction aléatoire. La hauteur ne dépend ni de la ~~liste utilise une fonction~~ clé de l'élément ni de sa valeur. La fonction aléatoire doit être définie pour qu'un élément de hauteur h possède le double de probabilités qu'un élément de hauteur $h + 1$. Ainsi, si un nœud de hauteur 1 a une probabilité p de 0.5 alors le niveau 2 a une probabilité ~~afin de définir sa hauteur maximale.~~ de 0.25.

2 Les algorithmes

Une skip-list a besoin d'avoir un pointeur vers sa tête ($node * head$). La liste doit connaitre le nombre de ~~voies actuellement utilisé ($size$)~~niveaux actuellement utilisés ($level$). Selon les implémentations, il est possible de définir un nombre de ~~voies~~ couches maximal ($levelMAX$). Il est aussi possible de changer le paramètre (p) de probabilité de création d'une nouvelle ~~voie~~couche. La variation de ~~ses~~ ces deux derniers paramètres est étudiée dans le chapitre Analyse des performances. Un nœud contient une clé key , une valeur $value$ ainsi qu'un tableau de pointeur vers les nœuds suivants $forward$. Il n'est pas nécessaire d'enregistrer la hauteur d'un nœud dans celui-ci.

```
typedef struct SkipList {
    int levelMAX, level;
    float p;
    struct node* head;
} SkipList;
typedef struct node {
    int key, value;
    struct node** forward;
} node;
```

L'initiation se fait de la manière suivante :

- La hauteur maximale est déterminée par le nombre d'éléments totaux (n) ~~présent~~ présents dans la liste. Pour des questions de rapidité, la valeur $log_2(n)$ est recommandée.
- Création d'un élément NIL possédant une clé plus grande que le maximum autorisé. Cette élément est aussi appelé *élément bidon.*
- Définitions ~~dès~~ de ses pointeurs $forward$ vers lui-même.
- Le nombre de voies actuelles est fixé à 1.
- Le pointeur de tête est défini par cet élément NIL.

~~L~~*La création d'~~algorithme écrit~~ une skip-liste en langage C figure en annexe A.1.*

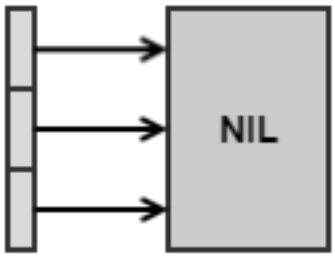


FIGURE 6 – Initialisation d'une skip-liste

2.1 La recherche

De manière générale, pour rechercher dans une skip-list il suffit de commencer par le niveau le plus haut et d'effectuer une recherche élément par élément. Si l'élément suivant devient plus grand que celui recherché, il suffit alors de descendre d'un niveau dans la liste.

Algorithmiquement parlant, La recherche d'un élément s'exécute de la façon suivante :

1. Commencer l'exploration de la liste par l'élément en tête, et se positionner sur la plus haute voie.
2. Sur la voie actuelle, regarder la clé de l'élément suivant.
3. ~~S'~~Si elle est plus grande et qu'il existe une voie inférieure, descendre d'une voie.
4. Sinon aller en 2.
5. Si la clé de l'élément suivant correspond à celle qui est recherchée : l'élément a été trouvé.
6. Sinon, l'élément n'a pas été trouvé.

L'algorithme écrit en langage C figure en annexe A.3.

2.2 ~~La hauteur~~

~~La notion de hauteur est très importante dans une skip-liste. En effet, celle-ci permet de définir à combien de nœuds un élément est attaché. Si trop d'éléments possèdent la même hauteur~~

Exemples

Comment rechercher l'élément 8 dans fig. 7 ?

— Se positionner sur la couche la plus haute : La 4ème voie.
— 8 est-il plus grand que 2 ? Oui, se positionner sur le nœud 2.

— 8 est-il plus grand que $+\infty$ *(NIL)* ? Non, descendre sur la 3ème voie.
— 8 est-il plus grand que 13 ? Non, descendre sur la 2ème voie.
— 8 est-il plus grand que 5 ? Oui, se positionner sur le nœud 5.
— 8 est-il plus grand que 13 ? Non, descendre sur la 1ère voie.
— 8 est-il plus grand que 8 ? Non, Il n'existe pas de voie plus basse.
— 5 est le dernier nœud parcouru. L'élément suivant sur la voie 1 est 8. 8 est-il l'élément recherché ? Oui, la recherche ~~devient trop lente. Cette méthode est définie à l'aide~~ est concluante.

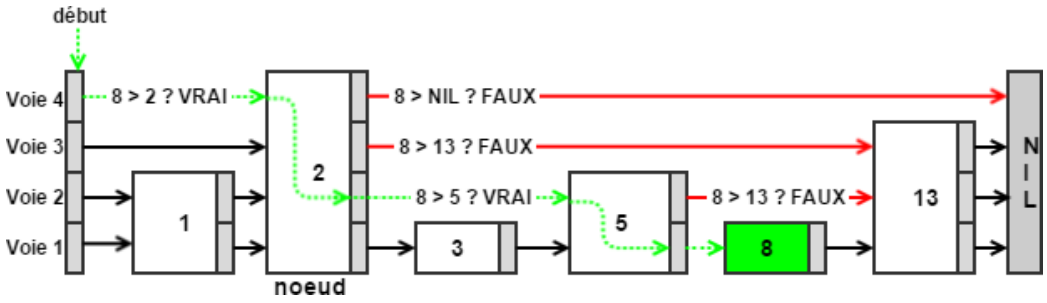


FIGURE 7 – Recherche de l'élément 8 dans une skip-liste

Comment rechercher l'élément 9 dans fig. 8 ?

- Se positionner sur la couche la plus haute : La 4ème voie.
- 8 est-il plus grand que 2 ? Oui, se positionner sur le nœud 2.
- 8 est-il plus grand que $+\infty$ *(NIL)* ? Non, descendre sur la 3ème voie.
- 8 est-il plus grand que 13 ? Non, descendre sur la 2ème voie.
- 8 est-il plus grand que 5 ? Oui, se positionner sur le nœud 5.
- 8 est-il plus grand que 13 ? Non, descendre sur la 1ère voie.
- 8 est-il plus grand que 8 ? Non, Il n'existe pas de voie plus basse.
- 5 est le dernier nœud parcouru. L'élément suivant sur la voie 1 est 9. 9 est-il l'élément recherché ? Non, la recherche est infructueuse.

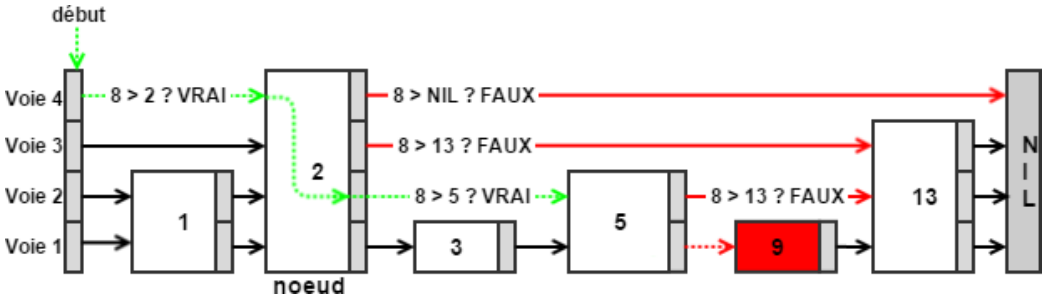


FIGURE 8 – Recherche de l'élément 9 dans une skip-liste

2.2 L'insertion

L'insertion d'une ~~fonction aléatoire. La hauteur ne dépend ni de la clé de l'élément ni de sa valeur. La fonction aléatoire doit être définie pour~~ entrée dans une skip-list est essentiellement la même chose que la recherche. A l'exception qu'un ~~élément de hauteur h possède le double de probabilité qu'un élément de hauteur $h + 1$. Ainsi, si un nœud de hauteur 1 a une probabilité p de 0.5 alors le niveau2 a une probabilité de 0.25~~tableau *update*, de taille identique à la hauteur de la liste, est utilisé pour mémoriser les pointeurs de chaque élément avant de descendre d'un niveau. Ce tableau est utilisé pour mettre à jour les liens nécessaires pour que la structure de donnée reste cohérente.

2.3 ~~L'insertion~~

~~L~~Algorithmiquement parlant, l'insertion d'un élément dans une skip-liste s'effectue de la façon suivante :

1. Commencer l'exploration de la liste par l'élément en tête, et se positionner sur la plus haute voie.
2. Sur la voie actuelle, regarder la clé de l'élément suivant.
3. Si elle est plus grande et qu'il existe une voie inférieure, marquer ce ~~nœud~~pointeur. Puis, descendre d'une voie.
4. Sinon aller en 2.
5. Si la clé de l'élément suivant correspond à celle qui est recherchée : modifier la valeur de l'élément par celle de la valeur à insérer. **FIN**.
6. Sinon, appeler la fonction de probabilité pour récupérer sa valeur n.
7. Si n est plus grand que la hauteur h de la liste, mettre à jour les pointeurs supérieurs à h vers NIL. Ensuite, définir la hauteur maximum actuelle h à n.
8. Créer le nœud x de hauteur n.
9. Pour chaque voie de 1 à n.
10. Mettre à jour les pointeurs de x vers l'élément suivant de l'élément marqué de cette voie.
11. Mettre à jour les pointeurs de l'élément marqué de cette voie vers x.
12. **FIN**.

L'algorithme écrit en langage C est présenté en annexe A.4.

Exemples

Comment insérer l'élément 8 dans fig. 9 ?

- Se positionner sur la couche la plus haute : La 4ème voie.
- 8 est-il plus grand que 2 ? Oui, se positionner sur le nœud 2.
- 8 est-il plus grand que $+\infty$ *(NIL)* ? Enregistrer ce lien en mémoire, puis descendre sur la 3ème voie.
- 8 est-il plus grand que 13 ? Enregistrer ce lien en mémoire, puis descendre sur la 2ème voie.
- 8 est-il plus grand que 5 ? Oui, se positionner sur le nœud 5.
- 8 est-il plus grand que 13 ? Enregistrer ce lien en mémoire, puis descendre sur la 1ère voie.
- 8 est-il plus grand que 8 ? Enregistrer ce lien en mémoire, Il n'existe pas de voie plus basse.
- 5 est le dernier nœud parcouru. L'élément suivant sur la voie 1 est 8. 8 est-il l'élément recherché ? Non, on crée un nouvel élément. Générer une hauteur aléatoire : hauteur 1.
- Ajouter un lien du nouvel élément 8 en voie 1. Ce lien doit pointer vers l'élément du lien enregistré à cette même voie. Il s'agit de l'élément 13.
- Modifier ce lien enregistré en voie 1 (de l'élément 5 vers l'élément 13) pour qu'il pointe vers ce nouvel élément 8.
- FIN.

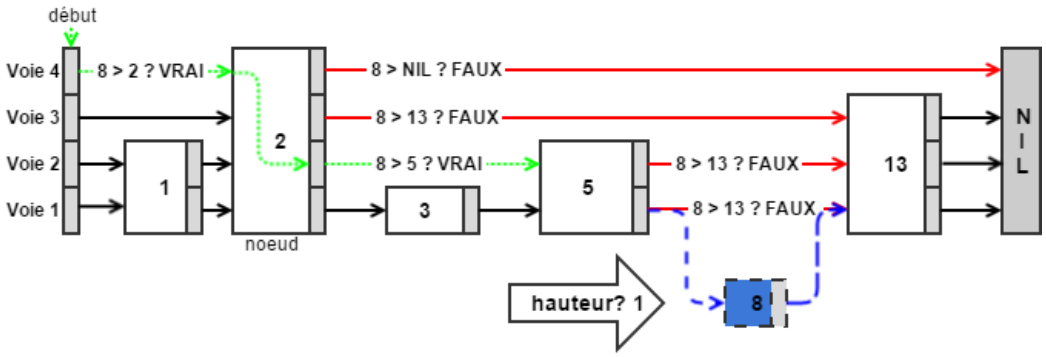


FIGURE 9 – Insertion dans une skip-liste

Comment insérer l'élément 5 dans fig. 9 ?

- Se positionner sur la couche la plus haute : La 4ème voie.
- 5 est-il plus grand que 2 ? Oui, se positionner sur le nœud 2.
- 5 est-il plus grand que $+\infty$ *(NIL)* ? Enregistrer ce lien en mémoire, puis descendre sur la 3ème voie.
- 5 est-il plus grand que 13 ? Enregistrer ce lien en mémoire, puis descendre sur la 2ème voie.
- 5 est-il plus grand que 13 ? Enregistrer ce lien en mémoire, puis descendre sur la 1ère voie.
- 5 est-il plus grand que 3 ? Oui, se positionner sur le nœud 3.
- 5 est-il plus grand que 8 ? Enregistrer ce lien en mémoire. Il n'existe a pas de voie plus basse.
- 5 est le dernier nœud parcouru. L'élément suivant sur la voie 1 est 8. 8 est-il l'élément recherché ? Non, on crée un nouvel élément. Générer une hauteur aléatoire : hauteur 2.
- Ajouter un lien du nouvel élément 5 en voie 1. Ce lien doit pointer vers l'élément du lien enregistré à cette même voie. Il s'agit de l'élément 8.
- Modifier ce lien enregistré en voie 1 (de l'élément 3 vers l'élément 8) pour qu'il pointe vers ce nouvel élément 5.
- Ajouter un lien du nouvel élément 5 en voie 2. Ce lien doit pointer vers l'élément du lien enregistré à cette même voie. Il s'agit de l'élément 13.
- Modifier ce lien enregistré en voie 2 (de l'élément 2 vers l'élément 13) pour qu'il pointe vers ce nouvel élément 5.
- FIN.

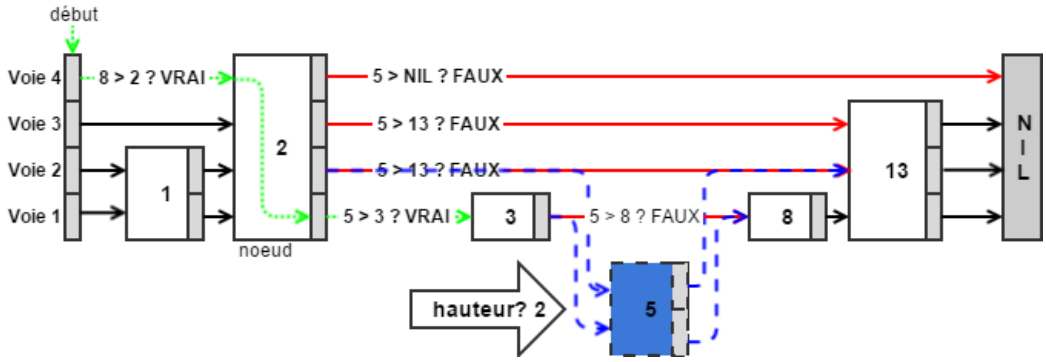


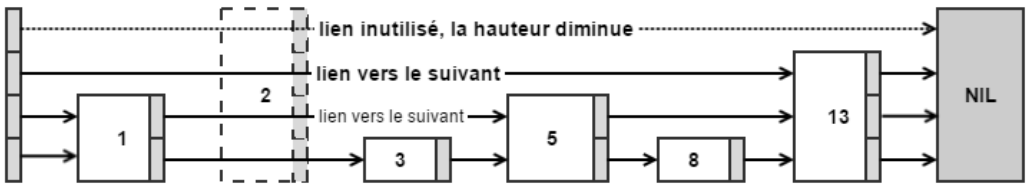
FIGURE 10 – Insertion dans une skip-liste

2.3 La suppression

La suppression d'un élément dans une skip-liste s'effectue de la façon suivante :

1. Commencer l'exploration de la liste par l'élément en tête, et se positionner sur la plus haute voie.
2. Sur la voie actuelle, regarder la clé de l'élément suivant.
3. Si elle est plus grande et qu'il existe une voie inférieure, marquer ce nœud. Puis, descendre d'une voie.
4. Sinon aller en 2.
5. Si la clé de l'élément suivant ne correspond pas à celle qui est recherchée, **FIN**.
6. Sinon pour chaque niveau de l'élément trouvé x, mettre à jour les pointeurs de l'élément marqué de cette voie vers l'élément suivant de x.
7. Supprimer x.
8. Si l'élément de tête de la hauteur actuelle de la liste pointe vers NIL, diminuer la hauteur actuelle de la liste de 1.
9. Sinon, **FIN**.
10. Si la hauteur est plus grande que 1, aller en 8.

L'algorithme écrit en langage C figure en annexe A.5.



3 Analyse des performances

La caractéristique majeure de la skip-list est l'aléatoire. L'insertion d'un élément nouveau utilise une hauteur qui est déterminée par le hasard. De ce fait, une même séquence produira une liste différente. Cela a comme conséquence qu'aucune séquence prédéterminée de nombres ne peut produire une dégradation des performances, contrairement aux arbres binaires qui eux doivent être constamment rééquilibrés.

De par sa nature probabiliste, on suppose que l'utilisateur n'aura pas un comportement à vouloir dégénérer volontairement la liste. C'est-à-dire que les performances peuvent être dégradées en retirant consciemment les éléments après avoir inspecté leur hauteur. Si l'utilisateur supprime tous les nœuds de hauteur $h > 1$, le pire scénario de la skip-list équivaut à une liste chainée classique.

Le temps requis pour exécuter l'insertion et la suppression dans une skip-list est prédominé par le temps de la recherche d'un élément. Ces deux derniers n'ont qu'un cout supplémentaire constant par rapport à la hauteur de la liste ; alors que le cout de la recherche est proportionnel à la longueur du chemin à parcourir.

3.1 Analyse pour $p = \frac{1}{2}$

Pour une variable aléatoire $p = \frac{1}{2}$ et une skip-list de taille infinie, celle-ci tend à ce que chacun des niveaux possèdent $\frac{1}{2}$ moins d'éléments. Ainsi, le niveau 1 contient n élément, le niveau 2 $\frac{n}{2}$, le niveau 3 $\frac{n}{4}$, ..., un élément à hauteur h : $\frac{n}{2^{h-1}}$. Par conséquent, le nombre total de liens dans une skip-list équivaut à

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots = \sum_{h=1}^{+\infty} \frac{n}{2^{h-1}} = 2n$$

Ce qui signifie qu'en moyenne un élément dans une skip-list possède 2 liens. Vu que la recherche se fait de haut en bas et de gauche à droite, chaque clé examinée à un niveau précis (i) ne peut pas appartenir à un niveau supérieur $(i+1)$. De ce fait, on peut déduire qu'en moyenne le nombre de fois que l'on avance dans la liste à un niveau précis équivaut au nombre de liens divisé par le nombre d'éléments, soit $\frac{2n}{n} = 2$.

La probabilité qu'un élément soit de hauteur i est de $p^i = \frac{1}{2^i}$. Le facteur p est constant ($\frac{1}{2}$), la probabilité de la hauteur des clés est équiprobable. Si l'on additionne la probabilité de hauteur chacune des clés de la skip-list, et que l'on prend h comme hauteur de la liste, on déduit la formule suivante.

$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots + \frac{1}{2^h} = \sum_{i=1}^{h} (\frac{1}{2})^i$$

Cette hauteur, il est possible de l'estimer. Si h tend vers l'infini, la série converge vers 1. Ce qui a comme conséquence :

$$n(\frac{1}{2})^h = 1 \iff (\frac{1}{2})^h = \frac{1}{n} \iff h = \log_{\frac{1}{2}}(\frac{1}{n}) \iff h = \log_2(n)$$

L'algorithme de recherche effectue possède 2 boucles imbriquées. La boucle la plus profonde permet d'avancer de gauche à droite dans la liste. Pour $p = \frac{1}{2}$, en moyenne celle-ci avance de 2. La boucle supérieure quant à elle varie selon la hauteur de la liste. La hauteur est estimée, avec une forte probabilité, à $\log_2 n$. Si l'on combine les 2 estimations, nous avons donc une forte probabilité que la recherche s'effectue en $\log_2(2n)$ opérations, soit $\mathcal{O}(log(n))$.

3.2 Généralisation

Le raisonnement utilisé précédemment pour $p = \frac{1}{2}$ reste valable quelle que soit la valeur de p, voir table 1. Le raisonnement qui effectue une sommation servant au calcul de lien dans une liste convergera vers $\frac{n}{p}$. Par contre, la somme des probabilités $(p^1 + p^2 + p^3 + ... + p^k)$, converge toujours vers $\frac{p}{1-p}$.

Scénario	attendu	pire
Nombre d'élément	n	n
Hauteur	$log_{\frac{1}{p}} n$	∞
Nombre de lien	n/p	$n * h$
Recherche	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Insertion	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Suppression	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$

TABLE 1 – Tableau récapitulatif

Le pire cas

Les ordinateurs ne possèdent pas une mémoire infinie. Par conséquent, il est impossible d'ajouter une infinité d'éléments. C'est pourquoi à la création d'une skip-list, la hauteur maximale est bornée par la formule du cas attendu avec une forte probabilité. Le pire scénario de hauteur infinie est donc impossible. Dans le cas très improbable que chacun des éléments de la skip-list soit à cette hauteur maximale, nous avons une recherche en $n * h$: soit $\mathcal{O}(n)$.

De façon générale, avec une constance $c > 1$ h est plus grande que $clogn$ avec une probabilité d'au plus $\frac{1}{n^c - 1}$. En d'autres mots, la probabilité que h soit plus petit que $clogn$ est d'au moins $1 - \frac{1}{n^c - 1}$. Par conséquent, avec une forte probabilité, la hauteur d'une skip-liste est bien d'ordre $\mathcal{O}(logn)$. Par exemple : si $n = 1000$, la probabilité est d'une sur un million.

3.3 Expérimentation

Les expérimentations suivantes ont été réalisées sur un ordinateur Windows 10 64 bits possédant un AMD FX 8350 et 16Gb de RAM. Les tests ont été lancés 10^5 fois, les résultats présentés sont la moyenne de tous les tests. Les très légères différences avec les résultats attendu peuvent s'expliquer par la façon dont un ordinateur génère des nombres aléatoires. On parle d'ailleurs de nombre "pseudo" aléatoire.

L'algorithme utilisé pour générer les nombres aléatoires dans ces tests est *xoroshiro128+*. Il a été développé par David Blackman et Sebastiano Vigna. Il s'agit de l'algorithme le plus rapide aujourd'hui. Il est intéressant d'utiliser un générateur rapide afin de déterminer la hauteur d'un élément de la skip-list. L'utilisation de la fonction *rand* de *stdlib* permet d'insérer 10^7 éléments en 5.792 secondes, alors que l'algorithme *xoroshiro128+* permet d'en insérer autant en 4.014 secondes. Cette différence de 30% n'est pas négligeable.

Variation de p

Afin de vérifier que le p à bien l'effet voulu, il suffit de compter le nombre d'éléments existants à chaque niveau. S'il y a n éléments à la hauteur 1, on s'attend à obtenir np éléments de hauteur 2 avec une forte probabilité.

Dans les faits, en essayant différentes valeurs pour une skip-list de 1000 éléments, on peut obtenir les résultats présents dans la table 2. Le résultat obtenu, correspond à notre espérance : p influence correctement la hauteur de la liste et par conséquent son poids.

Une question plus intéressante à se poser serait : Quel est la meilleure

p \ h	1	2	3	4	5	6	7	8	9	10
4/5	1000.00	800.05	640.02	512.01	409.64	327.74	262.16	209.74	167.81	134.23
3/4	1000.00	749.97	562.52	421.83	316.34	237.25	177.98	133.45	100.10	75.07
2/3	1000.00	666.76	444.52	296.35	197.60	131.70	87.79	58.49	38.98	25.99
3/5	1000.00	600.00	360.00	216.02	129.61	77.77	46.66	28.00	16.79	10.06
1/2	1000.00	500.01	250.06	125.04	62.56	31.27	15.63	7.82	3.91	1.96
2/5	1000.00	399.96	160.02	64.01	25.60	10.23	4.09	1.63	0.00	0.00
1/3	1000.00	333.33	111.14	37.03	12.34	4.11	0.00	0.00	0.00	0.00
1/4	1000.00	249.97	62.48	15.63	3.90	0.00	0.00	0.00	0.00	0.00
1/5	1000.00	200.04	40.00	8.01	0.00	0.00	0.00	0.00	0.00	0.00
1/10	1000.00	100.01	10.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1/20	1000.00	50.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

TABLE 2 – Le nombre d'éléments par niveau en fonction de p

valeur pour p ? Choisir un petit nombre réduit le nombre de liens dans la liste. Cela a comme conséquence de pouvoir faire de plus grand saut, mais moins souvent. Par exemple avec $p = 0.1$, il sera fort probable de faire des sauts d'une dizaine d'éléments. Mais la recherche du 9ème élément se fera sur la couche la plus basse qui elle, est 10 fois lente. En revanche, choisir un grand nombre pour p, augmentera fortement le nombre de liens dans la skip-list. Celle-ci possède donc un grand nombre de hauteurs, et donc plus de saut. Cela a comme conséquence de faire plus souvent des sauts, mais ils seront plus petits.

4 ~~Comparaison~~ Comparaisons avec d'autres structures~~de données~~

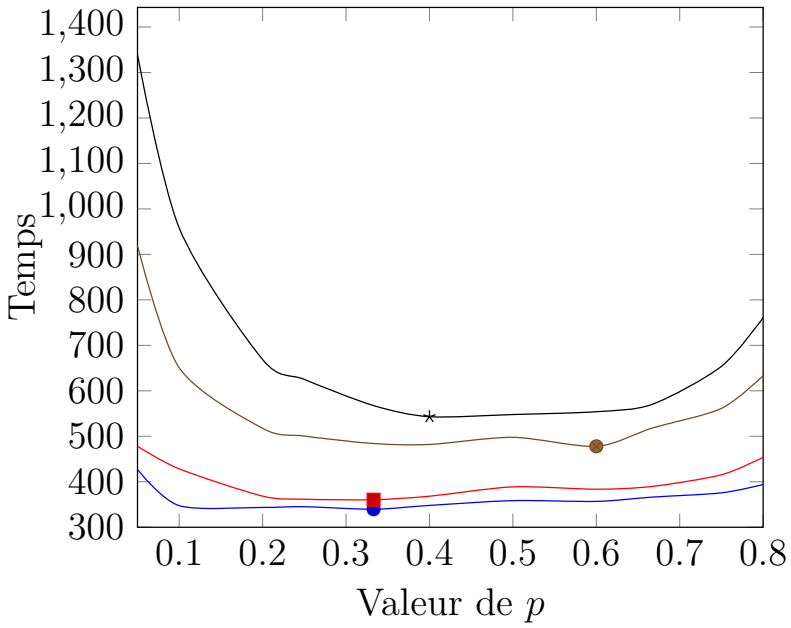


FIGURE 11 – Vitesse d'exécution en fonction de p

c

d

Annexe A Représentation d'une Skip-List en langage C

A.1 Initiation

```
SkipList* SK_init(int maxElem, float p) {
    SkipList* list  = (SkipList*)malloc(sizeof(SkipList));


    list->levelMAX = (int)round(log10(maxElem) / log10(1.0 /(double)p));
    if( list->levelMAX <= 0)
        list->levelMAX = 1;

    list->level = 1;
    list->head = createNode(list, INT_MAX, INT_MAX);
    for (int i = 0; i < list->levelMAX; i++)
        list->head->forward[i] = list->head;
    list->p = p;

    return list;
}
```

A.2 Fonction aléatoire

```
int getRandomLevel(SkipList* list) {
    int level  = 1;

        level++;
    }
    return MIN(level, list->levelMAX);
}
```

A.3 Recherche

```
node* SK_Search(SkipList* list, int key) {

    node* x = list->head;
```

```
    for (int i  = list->level - 1; i  >= 0; i--) {

        while (x->forward[i]->key < key) {
            x = x->forward[i];

        }
    }

    x = x->forward[0];
    if  (x->key == key) {

        return x;
    }

    return NULL;
}
```

A.4 Insertion

```
int SK_Insert(SkipList* list,  int key,  int value)  {

    node** update = (node**)malloc(sizeof(node*)*list->levelMAX);
    node* x = list->head;

    for (int i  = list->level - 1; i  >= 0; i--) {

        while (x->forward[i]->key < key) {

            x = x->forward[i];
        }
        update[i]  = x;
    }

    x = x->forward[0];
    if  (x->key == key) {

        x->value = value;
    }
    else {
        int level  = getRandomLevel(list);
```

```
        if (level  > list->level) {

            for (int i  = list->level; i  < level;  i++) {
                update[i]  = list->head;

            }
            list->level = level;
        }

        x = createNode(list, key, value);
        for (int i  = 0; i  < level;  i++) {
            // ???
            x->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = x;

        }
    }

    free(update);
    return 0;
}
```

A.5 Suppression

```
int SK_Delete(SkipList* list, int key) {

    node** update = (node**)malloc(sizeof(node*)*list->level);
    node* x = list->head;

    for (int i  = list->level - 1; i  >= 0; i--) {

        while (x->forward[i]->key < key) {

            x = x->forward[i];
        }
        update[i]  = x;
    }


    x = x->forward[0];
```

```
    if (x->key == key) {
        for (int i = 0; i < list->level; i++) {
            if( update[i]->forward[i] != x)
                break;
            update[i]->forward[i] = x->forward[i];

        }

        removeNode(x);

        while(list->level > 1 &&
            list->head->forward[list->level-1] == list->head ) {
            list->level--;
        }
    }

    free(update);
    return 0;
}
```

@online1, author = Patrice Roy, title = Skip Lists, date = 27/02/2016, url = http ://h-deb.clg.qc.ca/Sujets/Structures-donnees/SkipLists.html, @online2, author = Sylvie Hamel, title = Dictionnaires ordonnés et “Skip List”, date = 25/02/2016, url = http ://www.iro.umontreal.ca/ hamelsyl/SkipList.pdf,