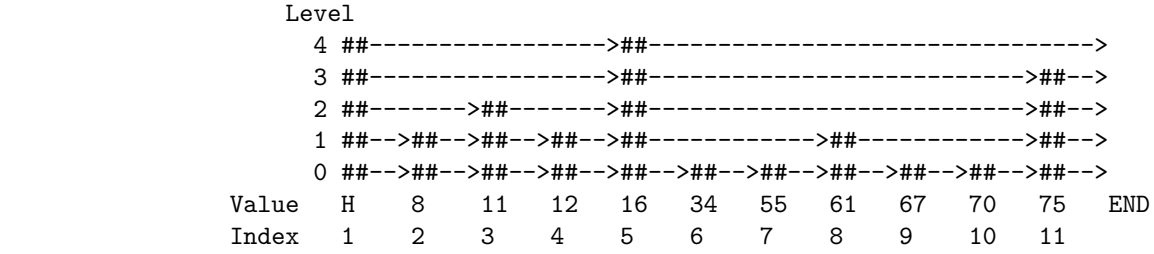
INFO-F-103 — Algorithmique 1
Devoir 1 – Skip list

1 Skip list[1][2]

Une skip list est une structure de données de type liste liée triée, c'est-à-dire une liste dont tous les éléments sont rangés de façon croissante, offrant des opérations de recherche, d'insertion et de suppression efficaces ($\mathcal{O}(\log n)$ en moyenne).

Contrairement aux listes liées classiques dont chaque noeud contient simplement une référence vers le noeud suivant (et le précédent si elle est bidirectionnelle), les noeuds de la skip list contiennent en plus un ensemble de références vers différents autres noeuds plus loin dans la liste, permettant de passer certains éléments pour effectuer une recherche plus rapide.

En pratique, voici la représentation d'une skip list:



Cette liste contient les valeurs triées $[8, 11, 12, 16, 43, 55, 61, 67, 70, 75]$ et est ainsi composée de 11 noeuds (un par valeur plus la tête de liste). Chaque flèche représente une référence, on peut donc voir que $5^{ème}$ noeud, contenant la valeur 16, dispose de 5 références permettant d'accéder à:

- `None` (niveau 4)
- 75 (niveau 3, noeud 11)
- 75 (niveau 2, noeud 11)
- 61 (niveau 1, noeud 8)
- 34 (niveau 0, noeud 6)

Il est à noter que pour que la skip list soit bien formée, il est impératif qu'un noeud ayant une référence au niveau n_t ait aussi une référence au niveau n pour tout n tel que $n_t > n$. On ne pourrait pas, par exemple, enlever la référence de niveau 3 du noeud 5 sans enlever sa référence de niveau 4. La tête de liste contient une référence vers chaque premier noeud de chaque niveau de la skip list.

1.1 Recherche

La recherche d'une valeur se fait de la manière suivante:

1. On démarre de la tête de liste, sur le niveau le plus haut
2. On avance tant que la valeur suivante est inférieure à celle cherchée et différente de `None`
3. Si la valeur suivante est la bonne, on a trouvé. Sinon, on descend d'un niveau et retour en 2

Par exemple, pour chercher 67 le chemin est le suivant:

```
Noeud 1 (Head) -> Noeud 5 (16) -> Descente niveau 3 -> Descente niveau 2
-> Descente niveau 1 -> Noeud 8 (61) -> Descente niveau 0 -> Noeud 9 (67)
```

1.2 Suppression

La suppression d'un élément se fait de manière intuitive. On retire le noeud et toutes ses références de la liste et on relie pour ne pas laisser de "trou".

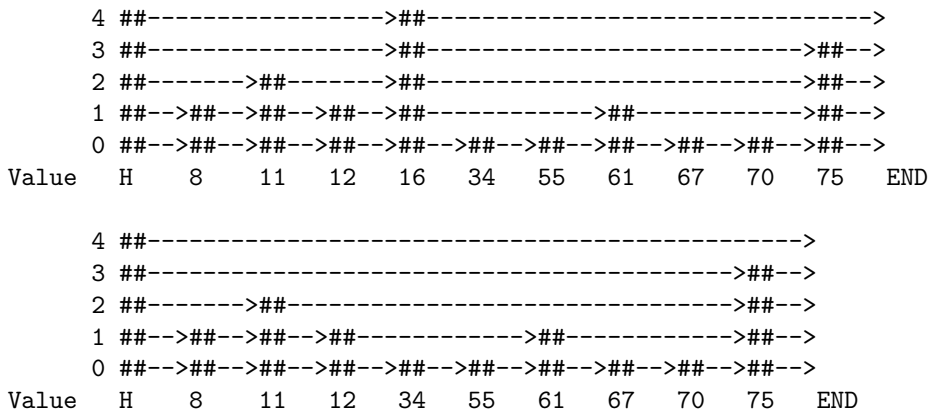


Figure 1: Suppression du noeud de valeur 16

1.3 Insertion

Pour insérer un élément, on commence par l'attacher au niveau 0 (on peut voir chaque niveau comme une liste liée classique, le plus bas contenant tous les éléments) et on applique une procédure stochastique pour l'insertion dans les niveaux supérieurs.

1. Insertion au niveau 0
2. Avec probabilité $\frac{1}{2}$, STOP
3. Passage au niveau supérieur, insertion, retour en 2

2 Énoncé

L'assistant en charge de ce devoir ayant besoin d'utiliser une structure de données aussi formidable, il a décidé de créer une classe la représentant en Python. Malheureusement, lors d'un lundi matin difficile, il a renversé par mégarde son e-café sur le fichier .py, ce qui a eu pour effet d'effacer deux méthodes indispensables.

Il vous est ainsi demandé de réparer le fichier Python en écrivant le corps des deux fonctions suivantes:

1. `previous_at_level(value, level=0)` : Cette méthode prend en paramètre une valeur et un niveau et renvoie le noeud de la liste précédant la valeur sur le niveau donné. Notons que la valeur ne doit pas nécessairement se trouver dans la liste, on veut juste le noeud contenant la valeur inférieure la plus grande. Naturellement, la fonction **doit** opérer de manière efficace, comme lors d'une recherche de valeur, ce n'est pas juste un parcours de liste liée en commençant directement sur le niveau.

   Exemples avec la liste présentée plus haut:

   - `previous_at_level(61, 1)` : Noeud 5 (valeur 16)
   - `previous_at_level(55)` : Noeud 6 (valeur 34)
   - `previous_at_level(14, 2)` : Noeud 3 (valeur 11)
   - `previous_at_level(72)` : Noeud 10 (valeur 70)

2. `search(value)`: Cette méthode recherche simplement une valeur dans la liste et renvoie le (premier) noeud la contenant ou `None` si elle n'est pas présente. La recherche se fait avec la technique expliquée dans la première partie de ce document: en partant de la tête de liste sur le niveau le plus haut, si on peut avancer, on le fait, sinon, on descend d'un niveau.

Nous vous demandons donc de compléter la classe définie dans le fichier `SortedList.py` que vous trouverez joint à cet énoncé sur l'Université Virtuelle. Vous trouverez aussi un fichier `tests.py` vous permettant de vérifier si votre structure de donnée est opérationnelle mais prêtez attention au fait que les tests ne prouvent pas que votre code est efficace et que les méthodes sont implémentées correctement.

3 Consignes

La version électronique est à remettre sur l'Université Virtuelle sous forme d'un fichier Python nommé netID.py (exemple: `fragerar.py`) contenant la classe `Node` et la classe `SortedList` complétée. Vous ne devez pas modifier les noms de méthodes, ni en rajouter.

La version papier sera à remettre à votre assistant lors de la séance d'exercices du mardi 16 février 2016.

Pour toute question → fragerar@ulb.ac.be