Implémentation des skip-listes

Objectifs pédagogiques

- Découvrir une nouvelle structure de données
- Travailler les listes
- Mettre en oeuvre une structure de données plus performante que les simples listes, de manière à bien préparer la notion de complexité abordée dans le cours « complexité et algorithmique avancée » au prochain semestre
- Compilation séparée, manipulation de pointeurs, allocation dynamique, gestion de la ligne de commande
- Si possible, découvrir les interfaces graphiques, en C, découvrir comment utiliser une librairie inconnue : se renseigner sur internet, explorer les possibilités...

Introduction

Vous avez appris dans les années passées et dans ce cours à manipuler les listes chainées.

Manipuler des éléments triés dans une liste chainée n'est pas très performant. En effet, rechercher un élément vous demande potentiellement de parcourir toute la liste car celle-ci n'est pas indexée, contrairement à un tableau trié. De même l'insertion et la suppression sont coûteuses en temps de calcul.

Ce projet vous propose d'implémenter une liste très particulière : la skip-liste, ou liste à enjambement. Ce type spécial de listes a été inventé par William Pugh. Vous trouverez une description de cette nouvelle structure de données à l'adresse suivante : http://fr.wikipedia.org/wiki/Skip-list. La skip-liste permet une insertion, supression et recherche en un temps logarithmique **en moyenne** par rapport au nombre d'éléments de la liste.

Objectifs

Le projet consiste à implémenter les skip-listes, à tester cette implémentation pour s'assurer de sa correction, et surtout à évaluer les performances de cette structure de données en terme de temps de recherche.

Il faudra donc aussi appliquer un protocole d'expérimentation pour tester cette structure. Voici le protocole proposé :

1. Lister les 1000 premiers entiers dans un ordre aléatoire
2. Insérer dans cet ordre ces 1000 premiers entiers dans une skip-liste vide au départ
3. Une fois les 1000 entiers insérés, calculer la hauteur de la skip-liste, les longueurs des listes pour chaque hauteur
4. Rechercher chacun des 1000 entiers, et calculer pour chaque recherche le nombre de pas nécessaires pour le trouver. On définira un pas par le suivi d'un pointeur : `e = e->suivant`, ou `e = e->dessous` (ces champs `suivant` et `dessous` seront plus clairs pour vous quand vous aurez pris connaissance des documents sur les skip-listes).
5. Recommencer ces étapes de nombreuses fois afin de calculer des moyennes.

Le programme devra pouvoir faire une sortie en mode texte du contenu de la structure (voir ci-dessous). Si vous vous en sentez capables, vous pouvez aussi proposer une sortie graphique. Une telle possibilité sera considérée comme un bonus pour l'évaluation. Mais attention, le temps consacré à son développement ne

devra pas être au détriment du reste du projet (implémentation de la skip-liste, tests, évaluation des performances, rapport).

En savoir plus sur les skip-listes

Vous pourrez vous référer aux sources suivantes pour comprendre le principe des skip-listes. Vous aurez aussi accès aux algorithmes d'insertion, suppression et recherche d'un élément dans une skip-liste :

- La page wikipedia pour une première découverte : http://fr.wikipedia.org/wiki/Skip-list
- L'article de l'inventeur des skip-listes, William Pugh : http://www.cs.uwaterloo.ca/research/tr/1993/28/root2side.pdf
- Une applet java permettant de manipuler une skip-liste : http://iamwww.unibe.ch/~wenger/DA/SkipList/

Convention sur les skip-listes pour ce travail

Vous implémenterez une skip-liste d'entiers. Contrairement à certains travaux sur le web, la hauteur d'une skip-liste ne devra pas être limitée.

Pour s'entendre sur un vocabulaire commun, dans la skip-liste ci-dessous :

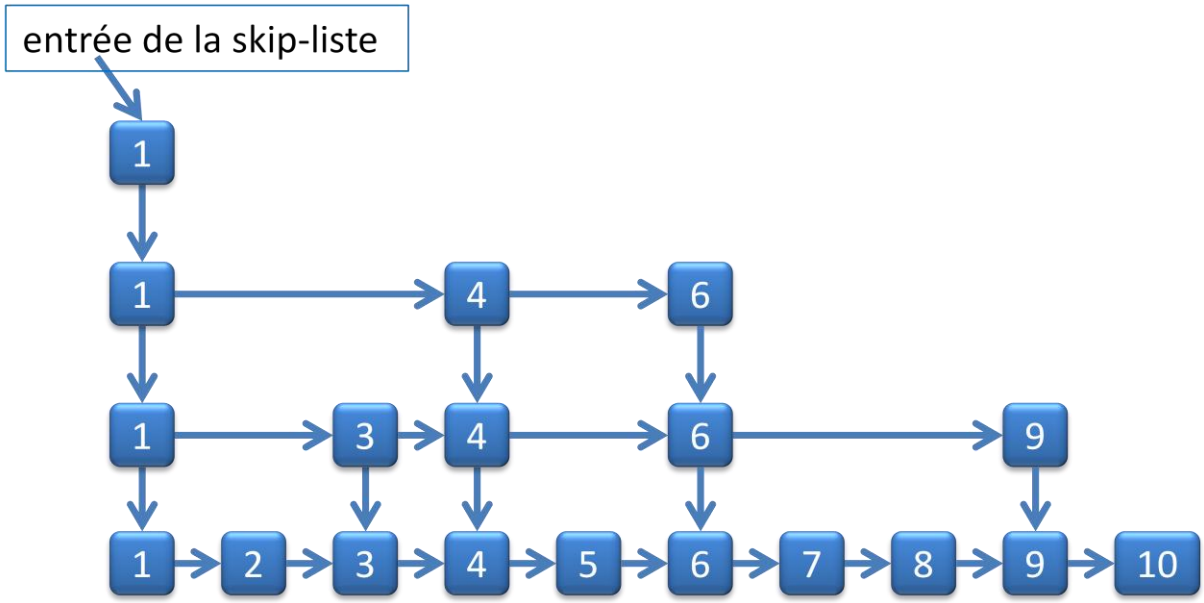
```
1
1-----4---6
1---3-4---6-----9
1-2-3-4-5-6-7-8-9-10
```

Les tirets sont juste présents pour montrer la correspondance verticale entre les éléments.

On dira que :

- La hauteur est 4
- La liste de hauteur 1 est : `1-2-3-4-5-6-7-8-9-10`
- La liste de hauteur 4 est `1`
- la longueur de la liste de hauteur 2 est 5

Sur le web, vous trouverez des présentations particulières des skip-listes. Ces présentations vous dirigent vers une implémentation plus compliquée que nécessaire. Voici donc, une représentation sour forme de pointeur de la liste ci-dessus (pour la compréhension, les liens horizontaux peuvent être appelés `suivant`, et les liens verticaux `dessous` ; les noeuds sont alignés sur le plan vertical par souci de lisibilité uniquement) :



Exemple d'utilisation (voir les sources pour comprendre) :

- pour rechercher 7 :
- j'entre par le 1. 1 est plus petit que 7, et est en fin de liste. Je passe en dessous.
- 1 est plus petit que 7, je passe au suivant
- 4 est plus petit que 7, je passe au suivant
- 6 est plus petit que 7, et est en fin de liste. Je passe en dessous.
- 6 est plus petit que 7 mais 9 est plus grand que 7 : je passe en dessous
- 6 est plus petit que 7, je passe au suivant
- Je tombe sur 7 : 7 est bien présent dans la liste.
- Nombre de pas pour rechercher : 6
- (7 pas auraient suffi pour rechercher 10).

Voici pour vous aider un scénario d'utilisation d'une skip-liste (ce scénario ne permet pas de comprendre l'ensemble du principe et des algorithmes des skip-listes. Voir les références données pour cela). Dans ce scénario la probabilité d'insérer un élément dans la liste supérieure est 0.5 (`face` : ne pas insérer, `pile` : insérer) :

- Initialisation : NIL
- Insertion de 1 :
  - Insertion dans la liste de hauteur 1
  - tirages aléatoires tombent sur `face`
  - On reste sur une hauteur de 1.
  - Résultat :
    - `1`
- Insertion de 2
  - Insertion dans la liste de hauteur 1
  - tirages aléatoires tombent sur `face`
  - On reste sur une hauteur de 1
  - Résultat
    - `1-2`
- Insertion de 3
  - Insertion dans la liste de hauteur 1
  - tirages aléatoires tombent sur `pile,face`
  - On insére aussi dans la liste de hauteur 2
  - Résultat
    - `----3`
    - `1-2-3`
- Insertion de 10
  - Insertion dans la liste de hauteur 1
  - tirages aléatoires tombent sur `face`
  - On n'insère pas dans la liste de hauteur 2
  - Résultat
    - `----3`
    - `1-2-3-10`
- Insertion de 7
  - Insertion dans la liste de hauteur 1

    - tirages aléatoires tombent sur `pile,pile,face`
    - On insère aussi dans les listes de hauteur 2 et 3
    - Résultat
        - `------7`
        - `----3-7`
        - `1-2-3-7-10`
- Insertion de 6
    - Insertion dans la liste de hauteur 1
    - tirages aléatoires tombent sur `pile,pile,pile,face`
    - On insère aussi dans les listes de hauteur 2, 3 et 4
    - Résultat
        - `------6`
        - `------6-7`
        - `----3-6-7`
        - `1-2-3-6-7-10`
- Suppression de 3
    - Résultat
        - `----6`
        - `----6-7`
        - `----6-7`
        - `1-2-6-7-10`
- Suppression de 6
    - La liste vide de hauteur 4 est supprimée
    - Résultat
        - `----7`
        - `----7`
        - `1-2-7-10`

Contraintes sur le travail demandé :

- Votre type `SkipListe` doit s'inspirer du type `Liste` que vous avez développé pendant les séances de TP
- Il faut utiliser la programmation modulaire : un couple .h/.c pour le type `SkipListe`…
- Votre programme doit intégrer un `Makefile` avec les cibles `test`, `prog` et `all`. `test` permet de générer un exécutable qui fait les tests de base (quelques insertions, suppressions, recherche d'un éléments présent, d'un élément absent…) ; `prog` crée l'exécutable pour le programme tel que demandé dans ce projet (gestion de la ligne de commande,…) ; `all` permet de générer ces deux exécutables. Enfin, il faut une cible `clean` pour nettoyer les .o, les exécutables et les fichiers temporaires.
- Votre programme généré par la cible `prog` doit utiliser la ligne de commande avec les options suivantes :
    - `-h` : affiche les options possibles sous un format proche de celui des pages man de Linux
    - `-p` : probabilité d'insérer un élément sur la liste supérieure. Par défaut, vaut 0.5
    - `-n` : nombre d'éléments à insérer. Les valeurs de 1 à la valeur donnée sont insérées dans la skip-liste. La valeur par défaut est 1000. Les entiers sont insérés dans un ordre aléatoire. Option incompatible avec `-f`
    - `-f` : option incompatible avec `-n`. Lit les entiers à insérer dans le fichier donné par `-f`.
    - `-o` : précise les affichages demandés. L'option est composée d'une suite de caractères collés les uns aux autres. Voici les caractères possibles et l'affichage associé :
        - `h` : demande l'affichage de la hauteur de la skip-liste après l'insertion des éléments.
        - `l` : demande l'affichage de la longueur de chaque liste en fonction de leur hauteur
        - `t` : affiche la skip-liste en mode texte une fois qu'elle est remplie.
        - `r` : lance la recherche de chaque élément de la liste, et affiche le nombre de pas total nécessaires pour retrouver les éléments.
        - `i` : le programme affiche graphiquement le contenu de la skip-liste une fois qu'elle est remplie. Ce mode est optionnel car vous n'allez pas tous implémenter une visualisation graphique. Mais l'option doit exister. Elle affiche un message d'erreur sur la sortie standard d'erreur si le module « interface graphique » n'est pas développé.

Conseils

- Les algorithmes d'insertion, de supression, et de recherche, ainsi que des implémentations de la skip-liste sont déjà présentes sur le web. Vous pouvez vous en inspirer, évidemment, MAIS PAS PLAGIER. Citer vos sources est évidemment une obligation.
- Investissez donc du temps sur l'interface utilisateur (via la ligne de commande, via les sorties texte, et via l'interface graphique si vous en développez une) et sur les tests de vos structures de données.
- Pour l'interface graphique, vous pouvez vous inspirer de http://iamwww.unibe.ch/~wenger/DA/SkipList/, même si la hauteur de la skip-liste y est limitée, ce qui me gène. C'est donc surtout une source d'inspiration pour votre propre visualisation.

Le rapport

Il doit contenir :

- Un mode d'emploi pour la compilation (1/2 page)
- Un copier-coller de la sortie due à l'option –h (1/2 page)
- Une description de vos structures de données, de leur implémentation (2 pages)
- Une description des algorithmes d'insertion, supression et recherche (3 pages)
- Des tests utilisant les deux programmes des cibles `test` et `prog`. Les tests doivent montrer que l'insertion de 10 éléments fonctionne, que la suppression fonctionne (d'un élément présent, un non présent), que la recherche d'un élément présent et non présent fonctionne, que les diverses options fonctionnent... (3 pages)
- Une partie montrant les statistiques obtenues : (4 pages)
    - Graphique de la hauteur moyenne en fonction du nombre d'éléments à insérer. Ajoutez sur ce graphique la courbe log en base 2
    - Graphique des longueurs moyennes des listes en fonction de leur hauteur, pour une insertion de 1000 entiers (faire de nombreux tests pour calculer ces moyennes). Ajoutez sur ce graphique la courbe $f(x) = 1000/(2^{x-1})$ où x est la hauteur de la liste.
    - Un graphique montrant le temps moyen de recherche d'un élément en fonction du nombre d'éléments dans le liste de hauteur 1.
    - Un paragraphe doit analyser ces courbes, et conclure sur les performances en recherche et en place mémoire de cette structure, et donner une comparaison par rapport aux listes simplement chainées.

Evaluation

L'évaluation insistera sur la qualité de programmation : utilisation de fonctions, de struct, allocation dynamique, gestion de la ligne de commande, compilation séparée, robustesse

Date de retour et organisation

Le 07 janvier 2013. La travail est à faire pas groupe de 4.