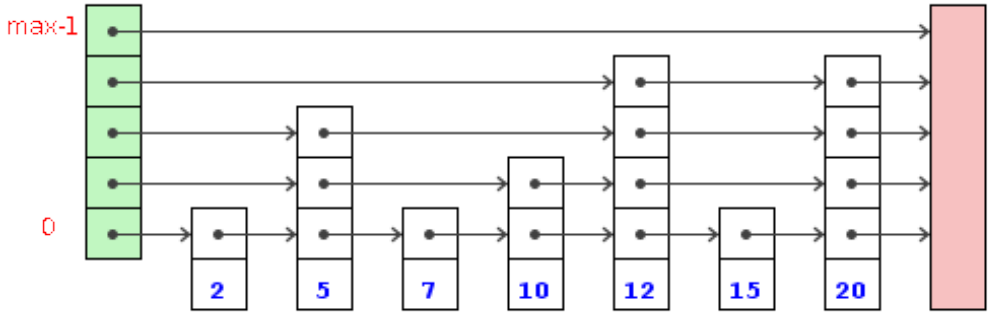
Université Grenoble Alpes
Master MIASHS - 2019
Documents autorisés
2 heures
barème indicatif

Examen de programmation 2
SkipList

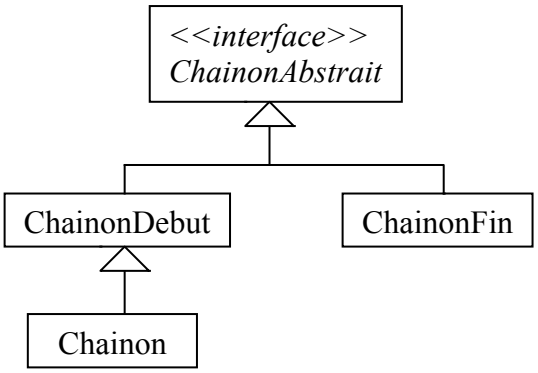
Une SkipList se présente comme une amélioration d'une liste chaînée triée. Elle contient des références supplémentaires vers l'avant, ajoutés de façon aléatoire, de sorte que la recherche dans la liste puisse « sauter » (skip) de nombreux éléments.
La SkipList est organisée en couches. La couche la plus basse est simplement une liste chaînée standard. Chaque couche supérieure est une voie plus rapide pour parcourir les couches inférieures.



Chaque chaînon de la liste a un nombre de couches calculé aléatoirement entre 1 et *max*.
La liste est représentée en utilisant deux chaînons particuliers, un chaînon de début, et un chaînon de fin.

1- Les chainons

Dans la première partie nous nous intéressons aux classes permettant de représenter les chaînons de la liste.



1.1- Les constructeurs (3 points)

Définir les constructeurs des classes `ChainonDebut`, et `Chainon`. Définir également la méthode `getInstance()` de la classe `ChainonFin`. Dans le constructeur de `ChainonDebut`, `n` est la taille du tableau `suivants`.

1.2- Méthodes compareTo (3 points)

Définir les méthodes `compareTo` des différentes classes.

- Une instance de `ChainonDebut` est inférieure à toute instance de `Chainon`, et à l'instance de `ChainonFin`. Une instance de `ChainonDebut` est égale à une autre instance de `ChainonDebut`.
- L'instance de `ChainonFin` est supérieure à toute instance de `Chainon`, et à toute instance de `ChainonDebut`.
- Deux instances de `Chainon` sont comparées en comparant leurs attributs `element`.

2- La classe SkipList

Pour calculer la taille du tableau `suivants` d'un `Chainon`, on utilise l'algorithme suivant : on tire aléatoirement un nombre entre 0 et 1 jusqu'à ce qu'il soit supérieur à 0.5. Ce nombre de tirages + 1 est la taille du tableau `suivants` s'il est inférieur à `max`, sinon la taille du tableau est `max`. Rappel : La méthode `Math.random()` retourne une valeur aléatoire plus grande ou égale à 0 et inférieure à 1.

2.1- Méthode tailleAlea (2 points)

Définir la méthode `tailleAlea` de la classe `SkipList`.

2.2- Les constructeurs (3 points)

Définir les constructeurs de la classe `SkipList`. Les constructeurs construisent une liste vide représentée de la façon ci-contre :
Le paramètre `max` (5 sur l'exemple) est la taille maximum des tableaux `suivants` pour les `Chainon`, et la taille du tableau `suivants` de `ChainonDebut`. Le paramètre `max` ne doit pas être inférieur à 1. Le constructeur sans paramètre initialise `max` à 5.

2.3- L'itérateur (3 points)

Compléter les méthodes `hasNext` et `next` de la classe anonyme déclarée dans la méthode `iterator`.
La méthode next devra lever une exception de type `NoSuchElementException` si l'appel à cette méthode est fait lorsque `current` est égal à l'instance de `ChainonFin`.

2.4- Méthode contains (3 points)

Définir la méthode `contains` qui retourne `true` si le paramètre `o` appartient à la liste et `false` sinon. Il faut que la méthode appartient utilise la structure particulière de la liste pour être le plus rapide possible. On devra s'inspirer de la programmation de la méthode `ajout`, qui ajoute un élément à sa place dans la liste, donnée en annexe.

2.5- Méthode last (3 points)

Définir la méthode `last` qui retourne le dernier élément de la liste s'il existe et `null` sinon. Il faut que la méthode `last` utilise la structure particulière de la liste pour être le plus rapide possible.

3- Questions de cours (5 points)

Parmi les instruction suivantes, donner les numéros de celles qui sont valides (i.e. qui compilent). Justifier brièvement la réponse. (3 points)

```
1. SkipList<Object> l1 = new SkipList<Object>();
2. SkipList<Integer> l2 = new SkipList<Integer>();
3. Set<Integer> l3 = new SkipList<Integer>();
4. List<Integer> l4 = new SkipList<Integer>();
5. Collection<Integer> l5 = new SkipList<Integer>();
6. SortedSet<Integer> l6 = new SkipList<Integer>();
```

Parmi les classes `ArrayList`, `LinkedList`, `TreeSet`, `HashSet`, quelle classe pourrait remplacer une implémentation complète de la SkipList ? Justifier la réponse. (2 points)

4- Annexes

```
public interface ChainonAbstrait<T extends Comparable<T>>
                              extends Comparable<ChainonAbstrait<T>> {
}
```

```
public class ChainonFin<T extends Comparable<T>>
                                    implements ChainonAbstrait<T> {

  private static final ChainonFin<?> inst = new ChainonFin<>();

  public static ChainonFin<?> getInstance(){
    ...
  }

  public int compareTo(ChainonAbstrait<T> o) {
    ...
  }
}
```

```
public class ChainonDebut<T extends Comparable<T>>
                                    implements ChainonAbstrait<T> {

  final ChainonAbstrait<T>[] suivants;

  public ChainonDebut(int n) {
    ...
  }

  public int compareTo(ChainonAbstrait<T> o) {
    ...
  }
}
```

```
public class Chainon<T extends Comparable<T>> extends ChainonDebut<T>{

  protected T element;

  public Chainon(int n, T e) {
    ...
  }

  public int compareTo(ChainonAbstrait<T> o) {
    ...
  }
}
```

```
public class SkipList<T extends Comparable<T>> extends AbstractSet<T> {

  private int max;
  private int taille;
  private ChainonDebut<T> debut;


  public SkipList() {
    ...
  }

  public SkipList(int max) {
    ...
  }
```

```
  private static int tailleAlea(int max) {
    ...
  }

  public int size() {
    return taille;
  }

  public boolean add(T e) {
    // création d'un nouveau chaînon
    Chainon<T> nouveau = new Chainon<>(tailleAlea(max), e);
    // tableau pour la mise à jour des références suivants
    ChainonDebut<T>[] maj = new ChainonDebut[max];
    // recherche de courant tel que courant.suivant >= nouveau
    ChainonDebut<T> courant = debut;
    for (int i = courant.suivants.length - 1; i >= 0; i--) {
      int r = courant.suivants[i].compareTo(nouveau);
      while (r < 0) {
        courant = (ChainonDebut<T>) courant.suivants[i];
        r = courant.suivants[i].compareTo(nouveau);
      }
      if (r == 0)
        return false;// la valeur a est déjà présente dans la liste

      maj[i] = courant;
    }

    // chaînage du nouveau chaînon
    for (int i = 0; i < nouveau.suivants.length; i++) {
      nouveau.suivants[i] = maj[i].suivants[i];
      maj[i].suivants[i] = nouveau;
    }
    taille += 1;
    return true;
  }

  public boolean contains(Object o) {
    ...
  }

  public Iterator<T> iterator() {
    return new Iterator<T>() {
      ChainonAbstrait<T> current=debut.suivants[0];

      public boolean hasNext() {
        ...
      }

      public T next() {
        ...
      }

      public void remove() {
        throw new UnsupportedOperationException();
      }
    };
  }

  public T last() {
    ...
  }
}
```