Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Algorythmique et structure de données

T1A

Réseau et Sécurité

S03 Type Abstrait, liste

Roten Marc

2017/2018

Table des matières

1 Ex 1

```
public static int
g
(List l, int i) {
 ListItr li = new ListItr(l);
 for(int k=0; k<i; k++)
  li.goToNext();
 return li.consultAfter();
}
```

retourne l'élément à l'indice donné en paramètre

```
public static boolean
f
(List l, int e) {
 ListItr li = new ListItr(l);
 while(!li.isLast()) {
  if (li.consultAfter()==e) return true;
  li.goToNext();
 }
 return false;
}
```

return true si on se trouve sur l'avant dernier élément de notre liste

2 Ex 2

classe ListItr

```
public class ListItr {
  final List list;
  ListNode pred, succ;
  // ----------------------------------------------------------
  public ListItr( List anyList ) {
    list = anyList;
    goToFirst();
  }
  // ----------------------------------------------------------
  public void insertAfter(int e) {
    ListNode myNode = new ListNode(e, pred, succ);
    if(isFirst()) {
      list.first=myNode;
    } else {
      pred.next= myNode;
    }
    if(isLast()){
```

```
      list.last = myNode;
    } else {
      succ.prev=myNode;
    }
    succ = myNode;
    list.size++;
  }
  // ------------------------------------------------------
  public void removeAfter() {
    if(isLast()) {
      System.out.println("there is no element after ");
      return;
    }
    if(isFirst()&&succ.next==null) {
      System.out.println("there is only 1 element in your
         list");
      /*
       * we delete the only element if just one elt
       */
      succ = null;
      list.first=null;
      list.last=null;
    }
    /*
     * on the forst elt, we do removeAfter
     */
    else if(isFirst()) {
      succ=succ.next;
      succ.prev=null;
      list.first=succ;
    }
    else if(succ.next==null) {
      /*
       * if we are on the last elt and not the only elt of my
          list
       */
      succ = null;
      pred.next=null;
      list.last=pred;
    }else {
      succ = succ.next;
      pred.next=succ;
      succ.prev=pred;
    }
    list.size--;
  }
  // ------------------------------------------------------
```

---

ensuite ma classe List

```
public class List {
  ListNode first, last;
  int size;
  // --------------------
  public     List() {
    first = null;
    last = null;
    size = 0;
  }
  public boolean isEmpty() {
    return size() == 0;
  }
  public int size() {
    return size;
  }
```

3 Ex 3

```
  public static int winnerAmStramGram(int n, int k) {
    List   l = new List();
    ListItr li = new ListItr(l);
    for(int j=0;j<n;j++) {
      /*
       * let's build the list
       */
      li.insertAfter(j);
    }
    li.goToFirst();
    while(l.size()>1)
    {
      /*
       * we advance by k times with goToNext()
       * if isLast is true, go back to the first elt
       */
      for(int i=1;i<k;i++) {
        li.goToNext();
        if(li.isLast()) {
          li.goToFirst();
        }
      }
      System.out.println("deleted "+li.consultAfter());
      li.removeAfter();
      li.goToFirst();
    }
    li.goToFirst();
    return li.consultAfter();
```

```
}
```

résultat à la console

```
Console  Problems  Javadoc  Declaration  Debug
<terminated> AmStramGram [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (8 mars 2018 à 14:33:40)
deleted 2
deleted 4
deleted 3
Winner is 1
```

4 Ex4

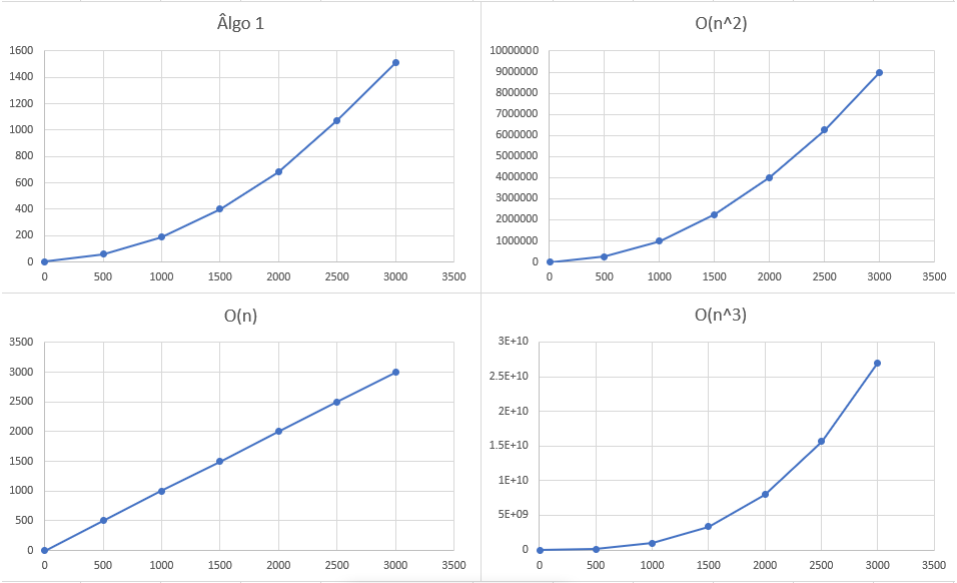
On obtient le type abstrait de pile, c'est un type abstrait de même efficacité que le type Abstrait liste. Mais dans ce cas précis, un itérateur est plus optimisé.

5 What Complexity

voici les mesures de n pour l'algorithme.

n	t (ms)	n	O(n)	O(n^2)	O(n^3)
1	1	1	1	1	1
500	58	500	500	250000	125000000
1000	189	1000	1000	1000000	1000000000
1500	403	1500	1500	2250000	3375000000
2000	684	2000	2000	4000000	8000000000
2500	1074	2500	2500	6250000	1.5625E+10
3000	1511	3000	3000	9000000	2.7E+10

voici les différentes complexité connue comparée à notre algorithme



notre algorithme Algo1 est donc de complexité O(n)