**Algorithmique / Travaux pratiques**

S03 – Type Abstrait, Liste





05.03.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

1. Voici ce que font les deux méthodes spécifiées dans la donnée.
   1. Première méthode :

Cette méthode va retourner un Integer se trouvant à la position « i » dans la liste « l », donnés en paramètre.

* 1. Deuxième méthode :

Cette méthode va regarder dans la liste « l » si l’Integer « e » s’y trouve. Elle retournera « true » si « e » y est et « false » dans le cas contraire.

1. Voici les différentes classes demandées
   1. List

package s03;

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;

}

}

* 1. ListItr

package s03;

public class ListItr {

List list;

ListNode pred, succ;

// ----------------------------------------------------------

public ListItr(List anyList) {

list = anyList;

goToFirst();

}

// ----------------------------------------------------------

public void insertAfter(int e) {

ListNode aux = new ListNode(e, pred, succ);

if (isFirst() && !isLast()) {

list.first = aux;

succ.prev = aux;

} else if (isFirst() && isLast()) {

list.first = aux;

list.last = aux;

} else if (!isFirst() && isLast()) {

list.last = aux;

pred.next = aux;

} else {

pred.next = aux;

succ.prev = aux;

}

list.size++;

succ = aux;

pred = aux.prev;

}

// ----------------------------------------------------------

public void removeAfter() {

if (isFirst() && !isLast() && succ != list.last) {

// 1er

succ.next.prev = null;

list.first = succ.next;

succ.next = null;

succ = list.first;

} else if (!isFirst() && !isLast() && succ != list.last) {

// Milieu

succ.next.prev = pred;

pred.next = succ.next;

succ.next = null;

succ.prev = null;

succ = pred.next;

} else if (!isFirst() && !isLast() && succ == list.last) {

// Dernier

pred.next = null;

list.last = pred;

succ.prev = null;

succ = null;

} else if (isFirst() && isLast()) {

// Pointe sur rien

} else if (isFirst() && !isLast() && succ == list.last) {

// Pointe sur 1er et dernier

succ = null;

list.first = null;

list.last = null;

}

list.size--;

}

// ----------------------------------------------------------

public int consultAfter() {

return succ.elt;

}

public void goToNext() {

pred = succ;

succ = succ.next;

}

public void goToPrev() {

succ = pred;

pred = pred.prev;

}

public void goToFirst() {

succ = list.first;

pred = null;

}

public void goToLast() {

pred = list.last;

succ = null;

}

public boolean isFirst() {

return pred == null;

}

public boolean isLast() {

return succ == null;

}

}

1. Voici le programme qui résout le problème AmStramGram

package s03;

public class AmStramGram {

public static int winnerAmStramGram(int n, int k) {

List l = new List();

ListItr li = new ListItr(l);

for (int i = n; i > 0; i--) {

li.insertAfter(i);

}

while (l.size() > 1) {

for (int i = 1; i < k; i++) {

if (li.isLast())

li.goToFirst();

li.goToNext();

}

if (li.isLast())

li.goToFirst();

li.removeAfter();

}

return l.first.elt;

}

public static void main(String[] args) {

System.*out*.println(*winnerAmStramGram*(5, 34));

System.*out*.println(*winnerAmStramGram*(14, 2));

}

}

1. Lorsque ces méthodes sont supprimées, on obtient le type abstrait de la pile (LIFO). Dans AmStramGram, ce type est inefficace sans les opérations goToXxx, isFirst et isLast.
2. Exercice 3 de la série 2 (Complexité)

Voici donc le tableau qui nous a aidés à résoudre cet exercice :

On peut remarquer que pour n2 les résultats sont plus au moins constants. Donc la complexité de l’algorithme Algo 1 est de O(n2).