

# N.Richard ALG3

NRI

October 8, 2021

## Contents

<b>1</b>	<b>ex 1.1</b>	<b>1</b>
<b>2</b>	<b>ex 1.2</b>	<b>1</b>
<b>3</b>	<b>ex 1.3</b>	<b>2</b>
<b>4</b>	<b>1.4</b>	<b>2</b>
	4.1 version 1 . . . . .	2
	4.2 version 2 . . . . .	2
<b>5</b>	<b>1.5</b>	<b>3</b>
	5.1 v1 . . . . .	3
	5.2 v2 . . . . .	3
<b>6</b>	<b>exo 3</b>	<b>4</b>
	6.1 3.1 . . . . .	4
	6.2 3.2 . . . . .	4
	6.3 3.3 . . . . .	4
	6.4 3.4 . . . . .	4
	6.5 3.5 . . . . .	5

## 1 ex 1.1

données :

- ListeChainée<T> liste
- T val

résultat : (néant) il faut introduire "val" dans "liste"

void insérerValeur(ListeChainée<T> liste, T val) liste.insérerTête(val) fin algo

## 2 ex 1.2

données :

- ListeChainée<T> liste
- T val

```
1 résultat : ElementListe<T>
2
3 ElementListe<T> recherche(ListeChainée<T> liste, T val)
4     ElementListe<T> cur = liste.getPremier();
5     while (cur != null && cur.getValeur() != val)
6         cur = cur.getSuivant()
7     fin-while
8     // 2 possibilités:
9     // - cur est null
10    // - cur.getValeur() == val
```

```

11         return cur
12     fin algo

```

### 3 ex 1.3

```

1     boolean contient(ListeChainée<T> liste, T val)
2         return recherche(liste, val) != null
3     fin algo

```

## 4 1.4

### 4.1 version 1

```

1     boolean supprimerValeur(ListeChainée<T> liste, T val)
2         /*
3          * Attention ceci ne convient pas car parcours inutile :
4          * if (!contient(liste, val))
5          *     return false
6          *
7          * else
8          *     // supprimer la valeur
9          *
10         */
11         // idée générale :
12         // SI la valeur cherchée est en début de liste,
13         // on supprime et on retourne vrai
14         // SINON on cherche un ElementListe qui précède la valeur cherché
15         ElementListe<T> prec = liste.getPremier()
16         while (prec != null
17             && prec.getSuivant() != null
18             && prec.getSuivant().getValeur() != val) {
19             prec = prec.getSuivant()
20         }
21         // REMARQUE: si prec == null, c'est que la liste est vide
22         // 3 possibilités:
23         // 1. prec == null, càd la liste vide (si prec==null)
24         // 2. prec.getSuivant() == null càd la valeur n'est pas trouvée
25         // 3. sinon, c'est qu'on a trouvé la valeur
26         if (prec == null)
27             return false
28         else if (liste.getPremier().getValeur() == val)
29             liste.supprimerTête()
30             return true
31         else if (prec.getSuivant() != null)
32             liste.supprimerAprès(prec)
33             return true
34         else
35             return false
36         fin if
37     fin algo

```

### 4.2 version 2

```

1     boolean supprimerValeur(ListeChainée<T> liste, T val)
2         ElementListe<T> prec = liste.getPremier()
3         if (prec == null)
4             return false
5         else if (prec.getValeur() == val)

```

```

6         liste.supprimerTête()
7         return true
8     fin if
9
10    while (prec.getSuivant() != null
11           && prec.getSuivant().getValeur() != val)
12        prec = prec.getSuivant()
13    fin while
14
15    if (prec.getSuivant() != null)
16        liste.supprimerAprès(prec)
17        return true
18    else
19        return false
20    fin if
21
22 fin algo

```

## 5 1.5

### 5.1 v1

```

1 int supprimerToutes(ListeChainée<T> liste, T val) {
2     int cnt = 0;
3     while (supprimerValeur(liste, val)){
4         cnt++;
5     }
6     return cnt;
7 }

```

très lisible mais ça reparcourt la liste du début à chaque itération

### 5.2 v2

```

1 int supprimerToutes(ListeChainée<T> liste, T val) {
2     ElementListe<T> prec = null;
3     ElementListe<T> cur = liste.getPremier();
4     int cnt = 0;
5     // contrainte: "prec.getSuivant() == cur" (sauf en tête de liste, prec = null)
6     while (cur != null) {
7         if (cur.getValeur() == val) {
8             cnt++;
9             if (prec == null) {
10                liste.supprimerTête();
11                cur = liste.getPremier();
12            } else {
13                liste.supprimerAprès(prec);
14                cur = prec.getSuivant();
15            }
16        } else {
17            prec = cur;
18            cur = cur.getSuivant();
19        }
20    }
21    return cnt;
22 }

```

## 6 exo 3

### 6.1 3.1

```
1 void ajouterTrié(ListeChaînée<T> l, T val) {
2     // trouver le bon endroit
3     ElementListe<T> prec = null;
4     ElementListe<T> cur = l.getPremier();
5     while (cur != null && cur.getValeur() < val) {
6         prec = cur;
7         cur = cur.getSuivant();
8     }
9     // insérer : insérerTête ou insérerAprès(elt, val)
10    if (prec == null) {
11        l.insérerTête(val);
12    } else {
13        l.insérerAprès(prec, val);
14    }
15 }
```

### 6.2 3.2

```
1 ElementListe<T> rechercheTriée(ListeChaînée<T> l, T val) {
2     ElementListe<T> cur = l.getPremier();
3     while (cur != null && cur.getValeur() < val) {
4         cur = cur.getSuivant();
5     }
6     //3 possibilités:
7     // cur == null, c'est-à-dire val ne se trouve pas là
8     // cur.getValeur() > val : val ne s'y trouve pas non plus
9     // cur.getValeur() == val : val s'y trouve (dans cur)
10
11    if (cur == null || cur.getValeur() > val)
12        return null;
13    else
14        return cur;
15 }
```

### 6.3 3.3

```
1 boolean contientTriée(ListeChaînée<T> l, T val) {
2     return rechercheTriée(l, val) != null;
3 }
```

### 6.4 3.4

```
1 boolean supprimerTriée(ListeChaînée<T> l, T val) {
2     ElementListe<T> prec = null;
3     ElementListe<T> cur = l.getPremier();
4     while (cur != null && cur.getValeur() < val) {
5         prec = cur;
6         cur = cur.getSuivant();
7     }
8     if (cur == null || cur.getValeur() > val) {
9         return false;
10    }
11    // ici on sait cur.getValeur() == val
```

```

12     if (prec == null) {
13         l.supprimerTête();
14     } else {
15         l.supprimerAprès(prec);
16     }
17     return true;
18 }

```

## 6.5 3.5

```

1  int supprimerToutesTriées_v1(ListeChainée<T> l, T val) {
2      ElementListe<T> prec = null;
3      ElementListe<T> cur = l.getPremier();
4      int cnt = 0;
5      while (cur != null && cur.getValeur() == val) {
6          l.supprimerTête();
7          cur = l.getPremier();
8          cnt++;
9      }
10     while (cur != null && cur.getValeur() < val) {
11         prec = cur;
12         cur = cur.getSuivant();
13     }
14     while (cur != null && cur.getValeur() == val) {
15         l.supprimerAprès(prec);
16         cur = prec.getSuivant();
17         cnt++;
18     }
19     return cnt;
20 }
21
22 //Autre version
23
24 int supprimerToutesTriées_v2(ListeChainée<T> l, T val) {
25     ElementListe<T> prec = null;
26     ElementListe<T> cur = l.getPremier();
27     int cnt = 0;
28     while (cur != null && cur.getValeur() < val) {
29         prec = cur;
30         cur = cur.getSuivant();
31     }
32     while (cur != null && cur.getValeur() == val) {
33         cur = cur.getSuivant();
34         if (prec == null) {
35             l.supprimerTête()
36         }
37         else {
38             l.supprimerAprès(prec);
39         }
40         cnt++;
41     }
42     return cnt;
43 }

```