Corrigé E.D. Algorithmes et Structures de Données n° 2

Thème : Les Listes

Exercice II.1 Calcul en notation polonaise postfixée

Un algorithme est le suivant :

Soit n la longueur de l'expression

répéter n fois
Lire la chaîne suivante c ;
si c est un opérateu r op alors
dépiler(entier i1) ; erreur si pile vide
dépiler(entier i2)) ; erreur si pile vide
calculer r=i1 op i2 ;
empiler® ;
sinon si c est un entier alors empiler© ;
sinon erreur ;
finsi ;
finsi ;
fin répéter ;
si pile non vide alors erreur ; fin si ;

Programme java :

```
import java.io.*;
public class Postfixe {
    static BufferedReader in =
                                                    new  BufferedReader(new
InputStreamReader(System.in));
    public static void main (String[] args) {
      Pile P=new Pile ();
      String s;
      try {
                for (int i=0; i< args.length; i++) {
           s = args[i];
          if (s.equals("+")) {
             int i1= P.sommet();
             P.depiler();
             int i2= P.sommet();
            P.depiler();
            P.empiler(i1+i2);
           }
          else if (s.equals("-")){
                   int i1= P.sommet();
                  P.depiler();
                  int i2= P.sommet();
                  P.depiler();
                  P.empiler(i2-i1);
                 }
                  else if (args[i].equals("x")) {
                     int i1= P.sommet();
                    P.depiler();
                    int i2= P.sommet();
                    P.depiler();
                   P.empiler(i1*i2);
                   }
                     else if (s.equals("/")){
                     int i1= P.sommet();
```

```
                    P.depiler();
                    int i2= P.sommet();
                   P.depiler();
                   P.empiler(i2/i1);
                   }
                    else P.empiler (Integer.parseInt(s));

}


if (P.estVide())
System.out.print("Expression mal ecrite");
      else System.out.print("Resultat du calcul : "+ P.sommet());
      }
      catch (Exception e) {
                  System.out.print("Probleme "+ e);
            }
    }
}
```

Exercice II.2 Inversion d'une liste chaînée

Question 1 On veut écrire une nouvelle méthode `renverser` qui inverse la liste courante**.**

- l pointera au fur et à mesure sur la partie non encore inversée de la liste. Nous utilisons 2 pointeurs supplémentaires :
- r qui pointe sur la tête de la sous-liste déjà inversée de la liste. Initialisé à null.
- p est simplement un pointeur auxiliaire qui permet d'effectuer le transfert d'un élément de la tête de l vers la tête de r.

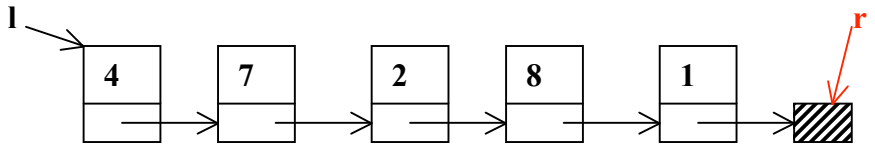
```
Liste renverser
     Liste l=this ;
     Liste r= null;
     Liste p ;
début
     tant que l /= null faire
          p = l;      -- on sauvegarde dans p la tête de la liste l
          l = l.suivant; -- on avance l (on enlève la tête de l)
          -- on insère p en tête de r
          p.suivant = r;
          r = p;
     fait;
     -- ici l==null et r contient le résultat de l'inversion.
     retourner r;
fin
```

Implantation en Java, en utilisant la classe Liste et les méthodes vues en cours :

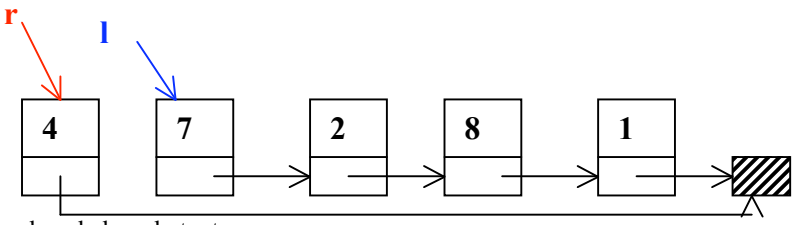
```
Liste renverser(Liste l){
        Liste l1= l ;
        Liste r= null;
```

```
        Liste p ;
        while ( l1 != null){
            p = l1;//on sauvegarde dans p la tête de la liste l
            l1 = l1.queue();// on avance l1 (on enlève la tête de l1)
            p = new Liste(p.tete(),r);      // on insère p en tête de r
            r = p;
        }
       //ici l1==null et r contient le résultat de l'inversion.
        return (r);
    }
```

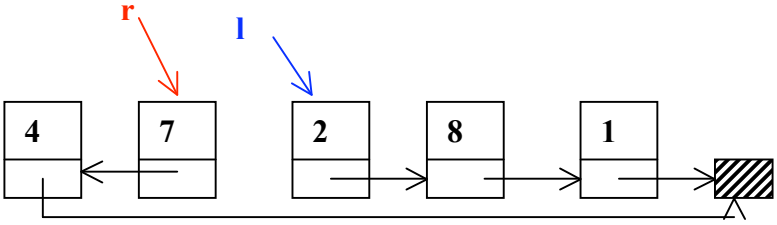
Sur l'exemple, au début :



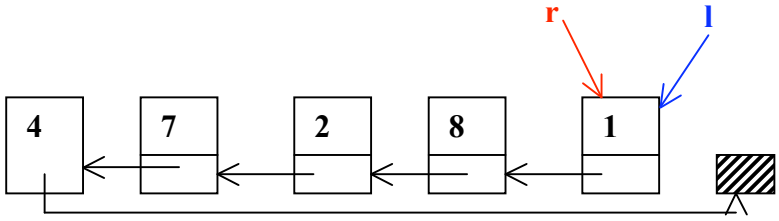
Premier passage dans la boucle tant que :



Deuxième passage dans la boucle tant que :



Etc …à la fin :

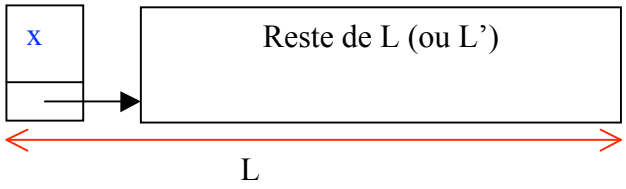


Question 2 Calculer la complexité de cette procédure.
Si n est le nombre d'éléments (ou longueur) de la liste, alors la boucle s'exécute n fois. Un passage par la boucle correspond à 4 opérations. Donc la complexité de cette procédure est O(n).

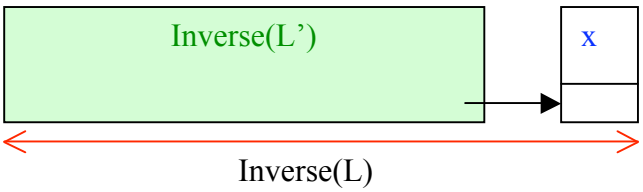
Exercice II.3 Inversion récursive d'une liste chaînée

Illustration de l'idée de la récursion :

Une liste L non vide peut toujours être considérée comme la juxtaposition de son premier élément (ou de son en_tête), que nous notons x, avec une autre liste L' (qui est en fait L privée de x).



Si on sait inverser L' alors on sait inverser L puisque :



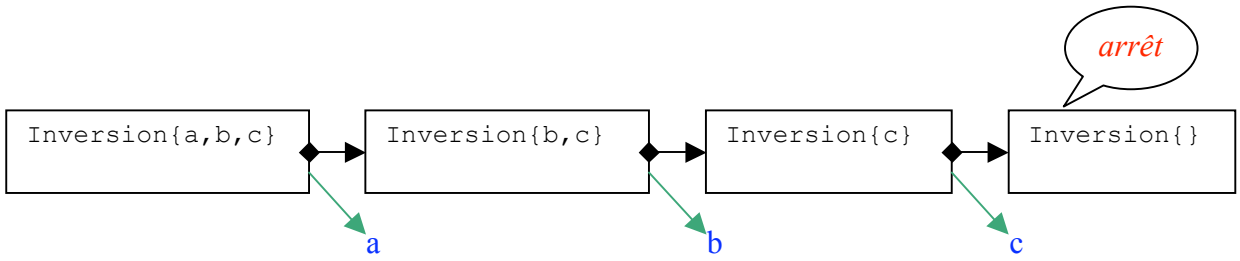
Question 1

```
Liste inverserRec (Liste l){  //inversion recursive
       Liste l1 = l ;
       if ( l1.queue() == null) return l1 ;
       int x = l1.tete();
       l1=l1.queue();   //l est alors tronquée de son premier element
       l1=inverserRec(l1) ;     //appel récursif
       l1.insererenqueue (x);
       return (l1);
   }
```

Question 2 Calculer la complexité de cette procédure.

Exemple d'exécution :

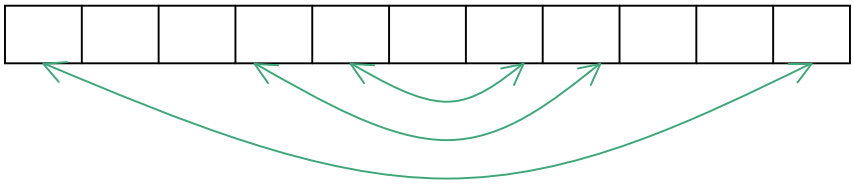
Si n est le nombre d'éléments de la liste, alors la procédure inversion est invoquée n fois.



A chaque fois, il faut au pire n opérations pour effectuer l'insertion en queue. Cette procédure est donc en $O(n^2)$. A comparer avec la procédure d'inversion de la question précédente.

Exercice II.4 Inversion d'une liste contiguë

Question 1 Définir un principe efficace d'inversion d'une liste représentée par un tableau**.**



On peut permuter les contenus des éléments extrêmes, par paires. On permute 1 et n, puis 2 et (n-1), puis 3 et (n-2), …. Si n est pair, tous les éléments sont permutés 2 à deux et cela fait n/2 permutations. Si n est impair, l'élément central ne change pas et cela fait (n-1)/2 permutations.

Question 2 Écrire la méthode inverser et calculer sa complexité.

```
void  inverseTab(int tab[]){
        int n, milieu, tamp ;
        n=tab.length - 1;
        milieu = n / 2;  // division entière
        for ( int i=0;i <=milieu;i++) {
        //permutation des éléments d'indice i et n-i
        tamp = tab[n-i];
        tab [n-i]= tab[i];
        tab[i ]=tamp;
    }
}
```

De l'ordre de n/2 passages dans la boucle pour (3 opérations élémentaires à chaque fois) donc la complexité est O(n).