Exercices d'algorithmique à programmer en TP

24 octobre 2017

1 AVERTISSEMENT

Il est contre-productif de vous lancer dans le projet si vous ne maîtrisez pas la programmation en Java. Il faut donc tenter de réaliser les exercices suivants, ou étudier les solutions quand elles sont fournies (il est inutile de "sécher" pendant des heures).

2 Pour augmenter la taille de la pile

Le compilateur Java gère assez mal la récusion ; par exemple, il ne détecte pas la récursion terminale. Il faut souvent agrandir la pile. La commane : "java -Xss100m monprogramme" permet d'augmenter la taille de la pile à 100 méga-octets.

3 Dessins de fractales

Pour Von Koch, dragon, Sierpinski, utiliser le squelette :
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/SQUELETTES_PROGRAMMES/DrawingLine`
Pour Mandelbrot, Julia : utiliser le squelette :
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/SQUELETTES_PROGRAMMES/squeletteMa`
Une solution en OCaml se trouve dans le répertoire :
`http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/MANDELJULIA/`
Il faut télécharger makefile et mandeljulia.ml, faire make, et lancer mandeljulia.opt

4 Ackermann

La définition est sur :
`https://en.wikipedia.org/wiki/Ackermann_function`.
Il est normal que cette fonction échoue.

5 Eratosthène

La solution est dans `https://fr.wikipedia.org/wiki/Crible_d%27%C3%89ratosth%C3%A8ne`

6 Tris de tableaux

Programmez le tri naïf, le tri rapide, le tri par fusion, le tri par tas.
Si nécessaire, le tri naïf est dans :
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/TRI_NAIF_TABLEAU/trin`
Programmer la recherche dichotomique (en O(log n)) dans un tableau trié :
Ecrire une procédure calculant l'indice dans le tableau trié de l'élément le plus grand inférieur à un élément donné (rendre -1 s'il n'en existe pas).
Une solution pour le tri rapide de tableaux est dans `http://ufrsciencestech.u-bourgogne.fr/licence`

7 Tris de listes

Pour les algorithmes de tri sur des listes de int :

```
public class L {
   public int elt;
   public L next;
   static public L cons( int x, L tail)
        { L cell=new L(); cell.elt=x; cell.next=tail; return cell;}
   static public L nil= null;
   static public boolean isempty( L liste) { return liste==nil ; }
   static public int head( L liste) { return liste.elt; }
   static public L tail(  L liste) { return liste.next; }
   static int fib( int x) { if (x < 2) return x; return fib(x-1) + fib(x-2); }
   static public int fact( int x) { if (x < 2) return 1; return x * fact(x-1); }

// concat de 2 listes
   static public L concat( L l1, L l2)
   { if (isempty( l1) ) return l2;
     if (isempty( l2) ) return l1;
     return cons( head( l1), concat( tail( l1), l2)); }

   static public int random( int x ) //{ return (311 * x) % 1001; }
   { return (int) (1000. * Math.random() ); }

   static public L lalea( int n)
        { if (n==0) return nil; return cons( random(n), lalea(n-1)); }

   static public L lalea2( int n) { L l=nil;
        for (int i=0; i<n; i++) l=cons( random(i), l);
        return l; }

static int min( int a, int b) { if (a<=b) return a; return b; }
   // minimm d'une liste :
   static public int minimum( L l, int sivide)
   { if ( isempty( l)) return sivide;
     return minimum( tail( l), min( head( l), sivide));
   }
   static public L remove( L l, int key)
   { if ( isempty( l)) return l;
     if (key == head( l)) return tail( l);
     return cons( head( l), remove( tail( l), key)); }
   static public L slowsort( L l)
```

```
   {
        if ( isempty( l) || isempty( tail( l))) return l ;
        int key = minimum( l, head( l));
        return cons( key, slowsort( remove( l, key)));
   }


   public static void main (String[] args)
   {
        L l=lalea2( 100);
        L ltriee= slowsort( l);
   }
}
```

La correction pour le tri de liste est dans :
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/TRIS_DE_LISTE/L.java`

8 Tri avec arbre binaire (non équilibré)

Ecrivez le tri avec un arbre binaire (à ne pas confondre avec le tas) non équilibré : les éléments du fils gauche sont inférieurs (ou égaux) à l'élement porté par la racine ; les éléments du fils droit sont supérieurs (ou égaux) à l'élement porté par la racine. Ecrivez la traversée Gauche-Racine-Droite de l'arbre, pour écrire les éléments de l'arbre dans un tableau trié.

Programmez la recherche dichotomique : écrire une procédure qui rend l'arbre (ou null) portant l'élément le plus grand et inférieur à un élément donné. De même, écrire une procédure qui rend l'arbre (ou null) portant l'élément le plus peit et supérieur (strictement) à un élément donné. Ecrivez une procédure calculant le noeud de l'élément le plus petit (le plus grand) d'un arbre donné.

Une solution du tri par arbre binaire est dans
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/TRI_PAR_ARBRE_BINAIRE/Arb.java`,
classe Arb.

9 Fibonacci

Programmer Fibonacci naïf. Programmez Fibonacci(n) en O(n) (avec une boucle par exemple). Programmez Fibonacci(n) rapide avec puissance rapide de matrices.

10 Hanoï

Programmer les tours de Hanoi :
`https://wikimonde.com/article/Tours_de_Hano%C3%AF`

En passant, il y a un lien entre les tours de Hanoi et le triangle de Sierpinski expliqué dans :
`https://www.youtube.com/watch?v=w_3hCjE6FWU`
`https://www.cut-the-knot.org/triangle/Hanoi.shtml`

11 Arithmétique

Programmez l'algorithme généralisé d'Euclide. Pour a et b deux entiers naturels, il calcule u et v deux entiers relatifs tels que $au + bv$ égale le PGCD (plus grand diviseur commun) de a et b.

12 Evaluation d'une expression postfixe

Ecrivez le programme d'évaluation d'une expression postfixe avec une pile (un tableau) :

```
java Calcul 1 2 3 "*" "+"
```

doit rendre 7 (Les guillemets sont utilisés pour éviter que le langage de script (un "shell") ne remplace l'étoile par les noms de fichier dans votre répertoire). Vous utiliserez un moins unaire (donc 100 - sera -100). Voici le début de calcul.java :

```
class Calcul {
static int npile= 0;
static int  pile[]= new int[100];
public static void main (String[] args)
{
        int n= args.length - 1 ;
        npile=0;;
        eval( args, pile);
}
public static void  eval( String[] args, int[] pile) { ... }
}
```

13 Plus courts chemins

Le programme suivant calcule les plus courts chemins et les affiche. Récupérez-le, compilez-le, exécutez-le, étudiez-le. Puis effacez les fonctions enchainer et PCC, avant de les ré-écrire.

```
import java.awt.*;
import java.math.*;

class LI { // gerer les listes d'entiers
   int elt; LI next;
   static  LI cons( int x, LI tail)
        { LI cell=new LI(); cell.elt=x; cell.next=tail; return cell;}
   static  boolean isempty( LI liste) { return liste==null ; }
   static  int head( LI liste) { return liste.elt; }
   static  LI tail(  LI liste) { return liste.next; }
   static  LI iaj( int i, int j) { if (i==j) return cons(i, null); return cons(i, iaj( i+1, j));}
}

class Pt {      double x, y;
      static Pt point( double u, double v) { Pt p=new Pt();  p.x=u; p.y=v; return p; }
}

class LP {   // GERER LES LISTES DE POINTS
   Pt elt; LP next;
   static  LP cons( Pt x, LP tail)
        { LP cell=new LP(); cell.elt=x; cell.next=tail; return cell;}
   static  boolean isempty( LP liste) { return liste==null ; }
   static  Pt head( LP liste) { return liste.elt; }
   static  LP tail(  LP liste) { return liste.next; }
}
```

```
class Arc { int v; double d;
      static Arc arc( int sommet, double distance)
      { Arc arc= new Arc(); arc.v=sommet; arc.d=distance; return arc;}
}
class LA  // GERER LES LISTES D'ARCS
{
   Arc elt;  LA next;
   static LA cons( Arc x, LA tail) { LA cell=new LA(); cell.elt=x; cell.next=tail; return cell;}
   static boolean isempty( LA liste) { return liste==null ; }
   static Arc head( LA liste) { return liste.elt; }
   static LA tail(  LA liste) { return liste.next; }
}


public class Dijkstra extends Frame {
      int width, height;
      Pt [] tabpts;
      LA [] tabarcs;
      double [] dist; // distance a la source
        int [] pred; // tableau des predecesseurs dans le PCC de la source au sommet

static double square( double d) { return d*d ; }


 // rend la liste des elements de rang pair (le premier est de rang 0 donc pair)
   static public LP pairs( LP l)
   { if (null==l) return null;
     if (LP.isempty( LP.tail( l))) return LP.cons( LP.head( l), null);
     return LP.cons( LP.head( l), pairs( LP.tail( LP.tail( l))));
   }
 // rend la liste des elements de rang impair
   static public LP impairs( LP l)
      { if (LP.isempty(l)) return null; return pairs( LP.tail(l)); }
 // fusion de 2 listes de points triees
   static public LP fusion( LP l1, LP l2)
   { if (LP.isempty( l1)) return l2;
     if (LP.isempty( l2)) return l1;
     Pt e1= LP.head( l1); Pt e2= LP.head( l2);
     if (e1.x <= e2.x || (e1.x==e2.x && e1.y <= e2.y))
      return LP.cons( e1, fusion( LP.tail( l1), l2));
     return LP.cons( e2, fusion( l1, LP.tail( l2)));
   }
   static public  LP mergesort( LP l)
   { if (LP.isempty( l) || LP.isempty( LP.tail( l))) return l;
     return fusion( mergesort( pairs( l)), mergesort( impairs( l)));
   }

   int etape( LI ls) // on met a jour les voisins du sommet a distance min de la source
   {       if( null == ls) return (-1);
      int s= LI.head( ls);
      for( LI li= ls; null != li; li = LI.tail( li))
      {      int i= LI.head( li);
            if ( dist[i] < dist[s]) s=i;
      }
```

```
   // ici s est le sommet  le plus proche de la source
   for( LA la= tabarcs[s]; null != la; la= LA.tail( la))
   {
         Arc arc = LA.head( la);
         int v= arc.v; double sv= arc.d;
         if( dist[s] + sv < dist[v]) { dist[v] = dist[s] + sv; pred[v]= s; }
   }
   return s;
}
static LI enlever( LI ls, int s)
{       LI l= null;
     for( ; null != ls; ls=LI.tail( ls))
          if( LI.head( ls) != s) l= LI.cons( LI.head( ls), l);
   return l;
}

void enchainer( LI ls)
   { for(;;)
       {  int s=etape( ls); if( -1==s) return; ls = enlever( ls, s);
       }
   }
void PCC( int src) // calcul des + courts chemins
{
   int n = tabarcs.length;
   dist = new double[n];
   for( int s=0 ; s<n; s++) dist[s]=1e10;
   dist[ src]= 0.;
   pred= new int[n];
   for( int s=0 ; s<n; s++) pred[s]= -1;
   pred[src]= src;
   LI ls = LI.iaj( 0, n-1);
     enchainer( LI.iaj( 0, n-1));
   // ici dist[] et pred[] sont corrects
}
void generer( int n, double rayon)
{
   System.out.println( "ENTREE DANS generer");

   LP l=LP.cons( Pt.point(0.,0.), null);
   for( int i=1 ; i<n;  )
   {      double x=  Math.random();
         double y=  Math.random();
         if ( 0.1 <= x && x <= 0.3 && 0.1 <= y && y <= 0.3) continue;
         if ( square( x - 0.6) + square( y - 0.6) < square (0.25)) continue;
         l= LP.cons( Pt.point( x, y), l); i++ ;
   }
   l= mergesort( l);
   tabpts = new Pt[ n];
   for (int k=0; l != null; l=LP.tail(l), k++) tabpts[k] =  LP.head(l);
   tabarcs= new LA[ n]; for( int i=0; i<n; i++) tabarcs[i]=null;
   for( int s=0; s<n; s++)
   {      double sx=tabpts[s].x; double sy= tabpts[s].y;
         for( int t=s+1; t<n && tabpts[t].x - tabpts[s].x <= rayon; t++)
         {      double tx= tabpts[t].x; double ty=tabpts[t].y;
```

```
                    double st = Math.sqrt( square( tx-sx) + square( ty-sy));
                if (st < rayon)
                {      tabarcs[t]= LA.cons( Arc.arc( s, st), tabarcs[t]);
                      tabarcs[s]= LA.cons( Arc.arc( t, st), tabarcs[s]);
                }
            }
      }

      // PCC( (int) (((double) (n-1) ) * Math.random()));
      System.out.println( "CALCUL PCC");
      PCC( 0 );
      System.out.println( "SORTIE DE  generer");
   }

   public void init() {
      System.out.println( "ENTREE DS INIT");
      width = getSize().width;
      height = getSize().height;
      setBackground( Color.white );
      System.out.println( "SORTIE DE INIT");
   }

   void tracer_segment( Graphics g, int i, int j)
   {
      double scale = (double) (getSize().width);
      int xi= (int) (scale * tabpts[i].x); int yi= 10 + (int) (scale * tabpts[i].y);
        int xj = (int) (scale * tabpts[j].x); int yj = 10 + (int) (scale * tabpts[j].y);
        g.drawLine( xi, yi, xj, yj);
   }
   public void paint( Graphics g )
   {

      g.setColor( Color.gray );
      for ( int i = 0; i < tabarcs.length - 1; ++i )
      {
        for( LA l= tabarcs[i]; null != l; l= LA.tail( l))
      { Arc arc= LA.head( l);
        int j = arc.v;
          tracer_segment( g, i, j);
      }
      }
      g.setColor( Color.green);
      g.setColor( Color.red);
      for ( int i = 0; i < tabarcs.length - 1; ++i )
      { int j= pred[i]; if (j != -1) tracer_segment( g, i, j); }

   }

public static void main (String arg[]) {
  Dijkstra f = new Dijkstra();
  f.init();
  f.generer( 8000, 0.03);
  f.setBounds(0, 0, 900, 900);
  f.setVisible(true);
```

```
 }
}
```

14 "Backtrack", ou recherche arborescente avec retour arrière

14.1 Programmer la résolution du problème des reines

Une première solution est ici
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/REINE/Reine.java`
Une deuxième solution utilise la classe Search et la class Backtrack :

```
abstract class Backtrack implements Search
{
      public L backtrack( L pile, L solutions)
      {
        while( L.nil != pile)
        {
                Object etat= L.hd( pile); pile= L.tl( pile);
                if (estsolution( etat))
                        solutions=L.cons( etat, solutions);
                else pile=empilefils( etat, pile);
        }
        return solutions;
      }
}

interface Search
{
      public boolean estsolution( Object o);
      public L empilefils( Object state, L pile);
      public L backtrack( L pile, L solutions);
}
```

Pbm.java cherche un chemin entre 1 et 100 ; les fils de x sont $2x$ et $2x+1$ (si x est assez petit). Voici Pbm.java :

```
public class Pbm implements Search
{
      public boolean estsolution( Object o) { int i= ((Integer)o).intValue(); return 100==i; }
      public L empilefils( Object s, L pile)
      { int n=((Integer) s).intValue();
        if (n < 100) { Object x1= new Integer(2*n);
                    Object x2 = new Integer(2*n +1);
                    System.out.println( "etat n, empile n1, empile n2 = " + n + ", " + x1 + ", " + x2
                    return L.cons( x1, L.cons( x2, pile));
                   }
       else return pile;
      }
      public L backtrack( L pile, L solutions)
      {       if (L.nil==pile) return solutions;
            Object etat= L.hd( pile); pile= L.tl( pile);
            if (estsolution( etat)) return       backtrack( pile, L.cons( etat, solutions));
            else return backtrack( empilefils( etat, pile), solutions);
```

```
    }
    public static void main (String[] args)
    {
          Pbm pb= new Pbm();
          L sol= pb.backtrack( L.cons( new Integer( 1), L.nil), L.nil);
          for( L l=sol; null!=l; l=L.tl(l))
          { int x=((Integer) L.hd(l)).intValue(); System.out.println( "Solution: "+ x); }
    }
}
```

Par analogie, écrivez Reine.java.
Une solution utilisant Backtrack.java et Search.java est dans
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/SEARCH/Reine.java`

14.2 Programmer la résolution du "compte est bon"

Une solution est dans le répertoire
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/CORRECTIONS/COMPTEBON`

15 Votre projet

Après avoir réalisé les exercices précédents, vous pouvez vous consacrer au projet.
Pour le projet du lancer de rayons, un fichier Tr.java est disponible dans le répertoire
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/2017/PROJET_LANCER_DE_RAYONS/`
ainsi que des images que vous devez produire quand votre projet sera réalisé, dans le répertoire
`http://ufrsciencestech.u-bourgogne.fr/licence2/Info31/2017/PROJET_LANCER_DE_RAYONS/IMAGES`
Le film devrait être affichable avec la commande : `gnome-open roman_000.jpeg` ou avec un navigateur (firefox, mozilla, konqueror, etc) en ouvrant une des images.