Université de Lille 1 SCIENCES ET TECHNOLOGIES

UFR IEEA

Master 1 d'Informatique ACT

2014–2015

Groupe:
NOM:
Prénom:

DS 3 novembre
Ecrire directement sur la feuille.

Exercice 1 : Le doublon *(6 pts)*

On a un tableau trié de $n+1$ entiers de 0 à $n-1$. On sait que chaque entier de 0 à $n-1$ est présent une et une seule fois dans le tableau sauf une valeur *doub* ($0 \leq doub \leq n-1$) qui est présente deux fois dans le tableau. On cherche à déterminer ce doublon *doub*.

Q 1. *(1 pt)* Que vaut donc T[0]? T[n]? De façon générale quelle valeur peut prendre T[i]?

Réponse:

T[0]=0, T[n]=n-1, T[i]=i ou T[i]=i-1.

Q 2. *(5 pts)* Proposer un algorithme en $O(\log n)$ qui détermine *doub*. Justifier la correction de votre algorithme.

Par exemple si $T =$

i	0	1	2	3	4	5	6	7	8	9
$T[i]$	0	1	2	3	3	4	5	6	7	8

, la valeur retournée sera 3.

Réponse:

Rechercher le doublon revient donc à rechercher le plus petit i tel que T[i]=T[i-1]=i-1, le doublon étant alors i-1.

Ou encore: Si T[i]=i, on sait que T[j]=j pour j<=i et donc le doublon est >=i. Si T[i]=i-1, on sait que T[j]=j-1 pour j>=i et donc le doublon est <=i-1.

Donc voici deux exemples de solutions (correction en commentaire du code)

```
//T trié, T[0]=0, T[T.length-1]=T.length -1; T contient un  seul doublon
int it_dicho(int T[]){
        int g=0;
        int d=T.length-1;
        while (g<d-1) {
            //invariant: g<d,  T trié croissant, T[g]=g, T[d]=d-1
            //  avec un et un seul doublon dans T[g..d])
            int m=(g+d)/2;
            if (T[m]==m) g=m;
            else d=m;}
        return T[g];
 }
Preuve:
arrêt : si g<d-1, alors  g<m<d, donc (d-g) décroît strictement
correction:
T reste trié.
si T[m]==m,  après g=m on a toujours  T[g]=g, T[d]=d-1, et l'unique doublon est entre g et d
si non, T[m]=m-1, et après d=m, on a toujours T[g]=g, T[d]=d-1, et l'unique doublon est entre g et d
de même on montre (voir preuve arrêt) que g<d.
l'invariant reste donc invariant et il est vrai au départ par hypothèse
quand g>=d-1, comme g<d, g=d-1: il reste deux éléments identiques, c'est le doublon.

ou en version récursive:
//g <=d, T de g à d contient les entiers triés de g à d-1 avec un et un seul doublon
int rec_doublon_dicho(int T[],int g, int d){
        if (g>=d-1) return T[g];
        //si il y a au plus deux éléments, c'est le doublon!
        int m=(g+d)/2;
        if (T[m]==m)
             //le doublon ne peut être une valeur de g à m-1, sinon on aurait T[m]=m-1
             return rec_doublon_dicho(T,m,d);
        else
             //T[m]=m-1, le doublon ne peut être une valeur >=m
             return rec_doublon_dicho(T,g,m);
 }
```

Exercice 2 : Le problème du sac à dos revisité *(14 pts)*

Soit la version suivante du problème du sac à dos:

Donnée du problème:

$p_1, \cdots, p_n$ // n entiers positifs

c // un entier positif

Sortie:

Vrai, si il existe $J \subset [1..n]$ -un choix des objets, tel que $\sum_{i \in J} p_i = c$ -qui remplisse exactement le sac -.

Faux, sinon.

Interprétation: il y a un sac de capacité maximale c et n objets de poids p_i. On veut choisir les objets à emporter pour remplir **exactement le sac**, i.e. pour avoir exactement c comme poids total.

On ne peut pas fractionner les objets.

Exemple: supposons que $n = 5, p_1 = 8, p_2 = 7, p_3 = 5, p_4 = 3, p_5 = 4$. Pour $c = 22$, on retournera `Vrai` car on pourra choisir de prendre les objets 1, 2, 4, 5. Pour $c = 26$, on retournera `Faux` car il n'y a pas de solution.

Q 1.*(1 pt)* Pour l'exemple précédent, y a-t-il une solution pour $c = 20$? pour $c = 21$? pour $c = 23$? pour $c = 28$? Justifier brièvement.

Réponse: 20: Oui car 8+7+5; 21 : Non car le poids total est 27, il faudrait laisser 6, ce qui est impossible. 23 :Oui car $8 + 7 + 5 + 3$. 28: Non car $28 >$ poids total.

Q 2.*(3 pts)* On définit $Remp(r, j)$ qui vaut Vrai si et seulement si on peut remplir exactement un sac dee capacité r avec les objets de numéro strictement supérieur à j, avec $0 \leq r \leq c$, $0 \leq j \leq n$. Le problème initial correspond donc à $Remp(c, 0)$.

Q 2.1. Que vaut $Remp(0, j)$?

Réponse: Vrai

Q 2.2. Que vaut $Remp(c, n)$ si $c > 0$?

Réponse: Faux

Q 2.3. Si $r > 0$ et $j < n$, exprimer $Remp(r, j)$ en fonction de $Remp(r, j + 1)$, $Remp(r - p_{j+1}, j + 1)$ (deux cas peuvent être envisagés).

Réponse:

$Remp(r, j) = Remp(r - p_j, j + 1) || Remp(r, j + 1)$, si $r \geq p_j$

$Remp(r, j) = Remp(r, j + 1)$, si $r < p_j$

Q 3.*(10 pts)* Déduire de ce qui précède un algorithme en $O(n * c)$ pour le problème. On retournera de plus un choix des objets qui remplit exactement le sac, si il en existe un. Vous pouvez répondre partiellement en proposant un algorithme qui ne retourne pas le choix des objets ou/et qui ne respecte pas la contrainte de complexité. Préciser si vous êtes dans ce dernier cas.

Réponse:

```
//p tableau de poids, ATTENTION p[i]=p_{i+1}
// c capacité
int n=p.length;
boolean[][] tRemp= new boolean[c+1][n+1];
boolean remp(){
   //remplissage table
   for (int  j= 0 ; j <=n; j++) tRemp[0][j]=true; //cas r=0
   for (int  r= 1 ; r <=c; r++) tRemp[r][n]=false; //cas j=n
   for (int  j= n-1 ; j>=0; j--)
       for (int  r= 1 ; r <=c; r++) //récurrence
           if  (r < p[j])  tRemp[r][j]=tRemp[r][j+1];
           else tRemp[r][j]=tRemp[r][j+1] || tRemp[r-p[j]][j+1];
          //rappel: p[j]=p_{j+1}
   // remontée
   if (tRemp[c][0]) {
      int r=c;
      for (int  j= 0; j<n; j++)
          if  (!tRemp[r][j+1]) { System.out.println(j+1);r=r-p[j];}
    }
    return  tRemp[c][0];
}
```