UNIVERSITÉ
CAEN
NORMANDIE

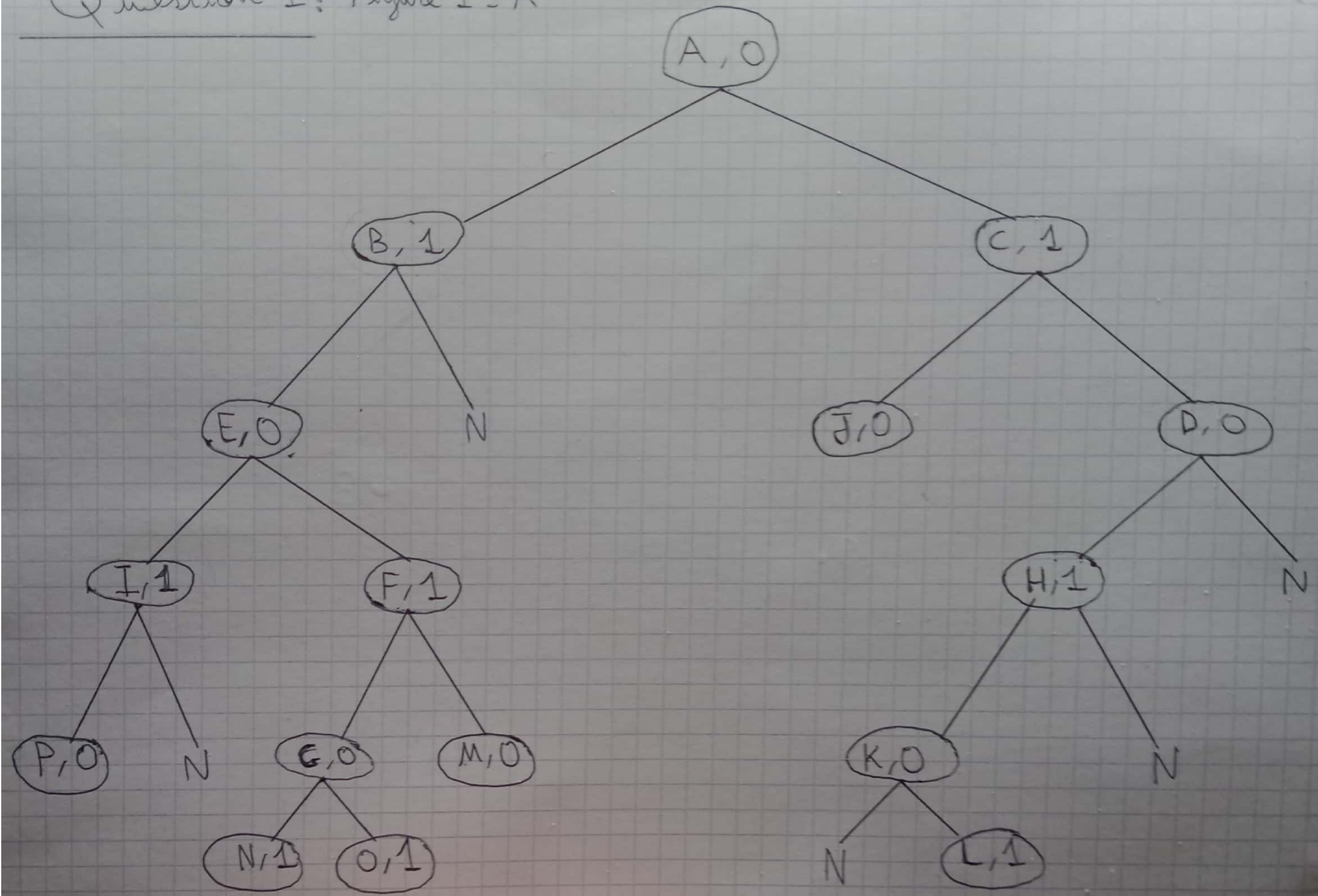
Devoir maison : Algorithme

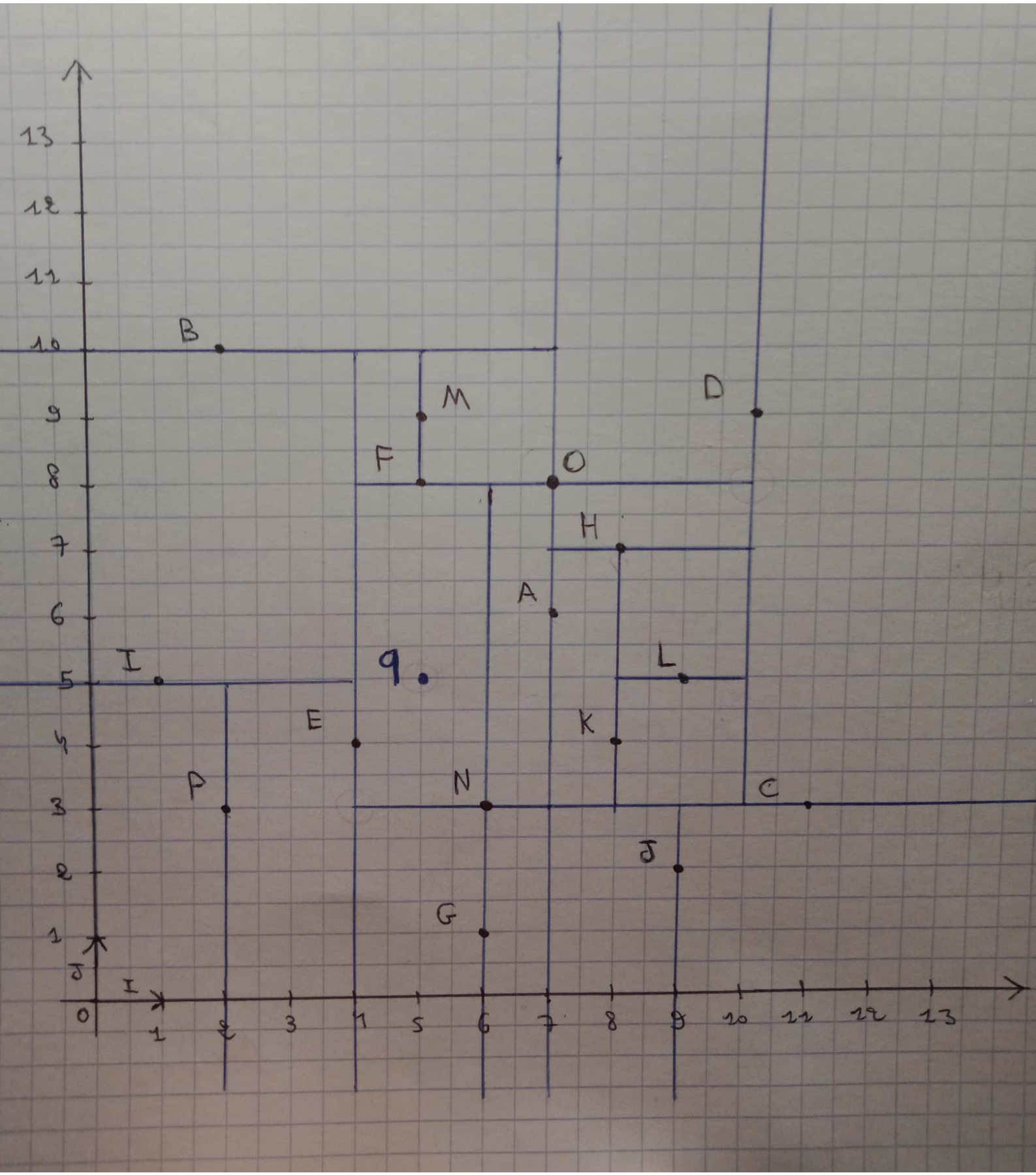
(la partie théorique)

Fait par :

- **Faical Sid Ahmed Rahmani (22010400)**
- **Brice Le Meur (21810250)**
- **Num binôme : 2447**

Question 1 : Figure 1 - A




13
12
11
10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
I
J
B
M
D
F
O
H
A
I
9.
L
E
K
P
N
C
J
G

Question 2 :

```
creerArbre(p : Point, g : ABRZ, d : ABRZ; cs :entier) : ABRZ
        tmp:ABRZ; tmp = nouveau(NoeudZ)
        tmp->valeur = p ; tmp->cs = cs
        tmp->gauche = g ; tmp->droit = d
        retourner tmp

*/

inserer(arb: ABRZ, p: Point): ABRZ
        si (arb = None) alors retourner insererAux(arb, p, 0)
        sinon retourner insererAux(arb, p, arb->cs)


insererAux(arb: ABRZ, p: Point, cs: entier): ABRZ
    ptr : Point //point à la racine
    si (arb = None) alors retourner creerArbre(p,None,None,cs)
    sinon:
        ptr = arb->valeur
        si (ptr = p) alors retourner arb
        si (cs = 0) alors:
            si (p.x <= ptr.x) alors arb->gauche = insererAux(arb->gauche,p,1)
            sinon arb->droit = insererAux(arb->droit,p,1)
        sinon: // cs=1
            si (p.y <= ptr.y) alors arb->gauche = insererAux(arb->gauche,p,0)
            sinon arb->droit = insererAux(arb->droit,p,0)
        retourner arb
```

Question 3 :

```
recherche(arb : ABRZ, p : Point) : ABRZ
    si arb = None alors retourner None
    si arb->valeur = p alors retourner arb
    ptr:Point ; ptr = arb->valeur
    si arb->cs = 0 alors
        si p.x <= ptr.x alors retourner recherche(arb->gauche,p)
        sinon retourner recherche(arb->droit,p)
    sinon: // arb->cs = 1
        si p.y <= ptr.y alors retourner recherche(arb->gauche,p)
        sinon retourner recherche(arb->droit,p)
```

complexité : au pire des cas on a :

- **Le point recherché n'existe pas.**
- **Le point apparait dans une feuille.**

Dans les deux cas on suppose qu'on va faire un parcours de la plus longue branche de l'arbre, donc le nombre maximum des nœuds visités

= hauteur(arb) + 1

Question 4 :

1/ question.option ;

// cette fonction compare entre deux noeuds et retourne le noeud dont l'abscisse du point est minimum.
// si les deux noeuds = None on retourne None.

```
minAbc(a:ABRZ, b:ABRZ):ABRZ
    si a = None et b = None alors retourner None
    si a = None et b <> None alors retourner b
    si b = None et a <> None alors retourner a
    si a->valeur.x < b->valeur.x alors retourner a
    retourner b // a->valeur.x >= b->valeur.x
```

// cette fonction compare entre deux noeuds et retourne le nœud dont l'ordonnée du point est minimum
// si les deux noeuds = None on retourne None

```
minOrd(a:ABRZ, b:ABRZ):ABRZ
    si a = None et b = None alors retourner None
    si a = None et b <> None alors retourner b
    si b = None et a <> None alors retourner a
    si a->valeur.y < b->valeur.y alors retourner a
    retourner b // a->valeur.y >= b->valeur.y
```

```
min(arb : ABRZ, dim : entier):ABRZ

    // le cas d'une feuille
    si arb->gauche = None et arb->droit = None alors retourner arb
    si dim = 0 alors // on cherche le point dont l'abscisse est minimum
        si arb->cs = 0 alors // on cherche à gauche
            si arb->gauche <> None alors retourner min(arb->gauche,dim)
            retourner arb
        sinon: // si arb->cs = 1 on cherche à gauche et à droite
            min1:ABRZ ; min2:ABRZ ; min1 = None ; min2 = None
            si arb->gauche <> None alors min1 = min(arb->gauche,dim)
            si arb->droit <> None alors min2 = min(arb->droit,dim)
            min1 = minAbc(min1,min2)
            // on compare le minimum des deux sous-arbre avec la racine
            si min1->valeur.x < arb->valeur.x alors retourner min1
            retourner arb
    sinon: // on cherche le point dont l'ordonnée est minimum
        si arb->cs = 1 alors // on cherche à gauche
            si arb->gauche <> None alors retourner min(arb->gauche,dim)
            retourner arb
        sinon: // si arb->cs = 0 on cherche à gauche et à droite
            min1:ABRZ ; min2:ABRZ ; min1 = None ; min2 = None
            si arb->gauche <> None alors min1 = min(arb->gauche,dim)
            si arb->droit <> None alors min2 = min(arb->droit,dim)
            min1 = minOrd(min1,min2)
            // on compare le minimum des deux sous-arbre avec la racine
            si min1->valeur.y < arb->valeur.y alors retourner min1
            retourner arb
```

2/ la liste des points visités (dim=0) : A , B , E , I , P , I , E , B , A

Question5 :

1/ question option :

```
dansDroite(arb : ABRZ, a : entier, dim : entier)
   si arb <> None alors // si arb = None on fait rien
        si dim = 0 alors // x = a
             si arb->valeur.x = a alors
                    // on affiche
                  afficher arb->valeur
             si arb->cs = 0 alors // on part soit à droite soit à gauche
                    si arb->valeur.x < a alors dansDroite(arb->droit,a,dim)
                    sinon dansDroite(arb->gauche,a,dim)
             sinon // arb->cs = 1  on part à gauche et à droite
                      dansDroite(arb->gauche,a,dim)
                      dansDroite(arb->droit,a,dim)
        sinon // dim = 1   y = a
             si arb->valeur.y = a alors
                  //on affiche
                  afficher arb->valeur

             si arb->cs = 1 alors // on part soit à droite soit à gauche
                  si arb->valeur.y < a alors dansDroite(arb->droit,a,dim)
                  sinon dansDroite(arb->gauche,a,dim)
             sinon // arb->cs = 0  on part à gauche et à droite
                      dansDroite(arb->gauche,a,dim)
                      dansDroite(arb->droit,a,dim)
```

2/ la liste des points visités (dim=0 , a=9) : A , C , J , J , C , D , H , K , L , L , K , H , H , D , C , A

Question 6 :

```
// cette fonction permet de déterminer si un point appartient à
z=[pig,psd]

est_dans_zone(p,pig,psd:Point):booleen
   retourner (pig.x < p.x) et (pig.y < p.y) et (p.x <= psd.x)
                              et (p.y <= psd.y)

// cette fonction permet de déterminer si la droite verticale
// passant par p coupe la zone z=[pig,psd]
// sans traiter le cas ou p.x = psd.x (traité dans intersection)

coupe_vert_zone(p,pig,psd:Point):booleen
   retourner (pig.x < p.x) et (p.x < psd.x)

// cette fonction permet de déterminer si la droite horizontale
// passant par p coupe la zone z=[pig,psd]
// sans traiter le cas ou p.y = psd.y (traité dans intersection)

coupe_horz_zone(p,pig,psd:Point):booleen
   retourner (pig.y < p.y) et (p.y < psd.y)
```

```
intersection(arb : ABRZ, v : Point, w : Point)

    si arb <> None alors

        si est_dans_zone(arb->valeur,v,w) = vraie
                // le point est dans la zone
                afficher arb->valeur
                intersection(arb->gauche,v,w)
                intersection(arb->droit,v,w)

        sinon si arb->cs = 0 alors
                    si coupe_vert_zone(arb->valeur,v,w) = vraie
                        // on part à gauche et à droite
                        intersection(arb->gauche,v,w)
                        intersection(arb->droit,v,w)
                    sinon si arb->valeur.x >= w.x  alors
                                    // le point à droite donc on part à gauche
                                    intersection(arb->gauche,v,w)
                              sinon intersection(arb->droit,v,w) // à gauche
                sinon // arb->cs = 1
                    si coupe_horz_zone(arb->valeur,v,w) = vraie
                        // on part à gauche et à droite
                        intersection(arb->gauche,v,w)
                        intersection(arb->droit,v,w)
                    sinon si arb->valeur.y >= w.y alors
                                    // le point en haut donc on part à gauche
                                    intersection(arb->gauche,v,w)
                              sinon intersection(arb->droit,v,w) // en bas
```

Question 7 :

```
// cette fonction calcule la distance entre deux point
dist(m,n:Point):double
   retourner sqrt((n.x-m.x)**2 + (n.y-m.y)**2)

// cette fonction teste si la droite passant par p
// verticale si cs = 0 et horizontale si cs = 1
// coupe ou non le cercle de centre cen et de rayon ray.
// retourne 0 si la droite coupe le cercle
// -1 si elle est en bas ou à gauche du cercle et
//  1 si elle est au dessus ou à droite du cercle.

position_droite_cercle(p:Point,cs:entier,cen:Point,ray:double):entier
   si cs = 0 alors
      si cen.x - ray <= p.x et p.x <= cen.x + ray alors retourner 0
      si cen.x - ray > p.x  alors retourner -1
      si cen.x + ray < p.x alors retourner 1
   si cs = 1 alors
      si cen.y - ray <= p.y et p.y <= cen.y + ray alors retourner 0
      si cen.y - ray > p.y  alors retourner -1
      si cen.y + ray < p.y alors retourner 1


plusproche(arb:ABRZ, q:Point):ABRZ
   si arb = None retourner None
// initialisation : ppc = racine , dminc = entre la racine et q
   retourner plusprocheAux(arb,q,arb,dist(arb->valeur,q))
```

```
plusprocheAux(arb:ABRZ, q:Point, ppc:ABRZ, dminc:double):ABRZ

    si arb = None alors retourner ppc
    si dist(arb->valeur,q) <  dminc alors
          // le point courant est plus proche
          dminc = dist(arb->valeur,q)
          ppc = arb
    si position_droite_cercle(arb->valeur,arb->cs,q,dminc) = 0 alors
          // la droite (h ou v) coupe le cercle
          a,b:ABRZ // on cherche le plus proche pour chaque sous arbre
          a = plusprocheAux(arb->gauche,q,ppc,dminc)
          b = plusprocheAux(arb->droit,q,ppc,dminc)
          // on retourne le plus proche entre le plus proche courant
          // et le plus proche de chaque sous arbre
          si dist(a->valeur,q) <= dist(b->valeur,q) alors
              si dist(a->valeur,q) < dminc alors retourner a
          si dist(b->valeur,q) <= dist(a->valeur,q) alors
              si dist(b->valeur,q) < dminc alors retourner b
          retourner ppc
    si position_droite_cercle(arb->valeur,arb->cs,q,dminc) = -1 alors
      // la droite (h ou v) est (en bas ou à gauche) du cercle
      // il suffit de chercher dans le sous arbre droit
          retourner plusprocheAux(arb->droit,q,ppc,dminc)
    // la droite (h ou v) est (en haut ou à droite) du cercle
    // il suffit de chercher dans le sous arbre gauche
    retourner plusprocheAux(arb->gauche,q,ppc,dminc)
```

2 / la liste des points visités :

A,B,E,I,P,I,I,E,F,G,N,G,O,G,F,E,B,A,C,J,C,D,H,K,H,H,D,C,A

Le point plus proche pour q=(5,5) est E