Université Montpellier
Mountaz Hascoët

Examen du 29/05/2017 - Session 1

Documents autorisés :

Une feuille recto-verso de notes personnelles

Intersection de deux intervalles

1) Expliquez de la manière la plus simple et concise possible, le principe de l'algorithme utilisé par le programme ci-dessous.
2) Résumer les hypothèses implicites d'utilisation de ce programme et dire pour quels cas de la figure 2 le programme ci-dessous donne une réponse satisfaisante et pour quels cas la réponse donnée par le programme n'est pas satisfaisante. Justifiez vos réponses.
3) Écrivez un algorithme permettant de traiter tous les exemples de la figure 2 en annexe et donnez pour cet algorithme le nombre de branchements.
4) On souhaite distinguer des catégories de cas d'intersections par rapport d'une part au du nombre d'opérations (x,+,-,/) utiles et d'autre part à la longueur du chemin à parcourir dans l'arbre de contrôle de l'algorithme. Donnez pour chaque catégorie le nombre d'opérations, la longueur du chemin, ainsi que les exemples de la figure 2 qui correspondent à cette catégorie.
5) On vous demande de rendre votre algorithme *plus lisible,* c'est à dire, plus facile à comprendre par autrui.
   (a) Résumez les éléments qui contribuent à une meilleure lisibilité selon vous.
   (b) Résumez le principe des modifications à apporter à l'écriture de votre algorithme pour le rendre plus lisible. Si cette simplification est significativement différente de l'écriture donnée à la question 3, donnez la nouvelle écriture de votre algorithme.

```
public class Segment extends Line2D.Double{
	Point2D.Double[] p;
	public Segment(Point2D.Double p0, Point2D.Double p1){
		super(p0,p1);
		p = new Point2D.Double[2];
		p[0] = new Point2D.Double(p0.x,p0.y);
		p[1] = new Point2D.Double(p1.x,p1.y);
	}

	public Point2D.Double intersection(Segment s2){
		if(this.intersectsLine(s2)){
			double m1 = (p[1].y - p[0].y) / (p[1].x - p[0].x);
			double m2 = (s2.p[1].y - s2.p[0].y) / (s2.p[1].x - s2.p[0].x);
			double p1 = p[0].y - m1 * p[0].x;
			double p2 = s2.p[0].y - m2 * s2.p[0].x;
			double x = (p2 - p1) / (m1 - m2);
			double y = m1 * x + p1;

			return new Point2D.Double(x, y);
		}

		return null;
	}
}
```

Figure 1 : extrait de programme, la méthode `intersectsLine` renvoie vrai si les droites portant les segments s'intersectent et faux sinon.

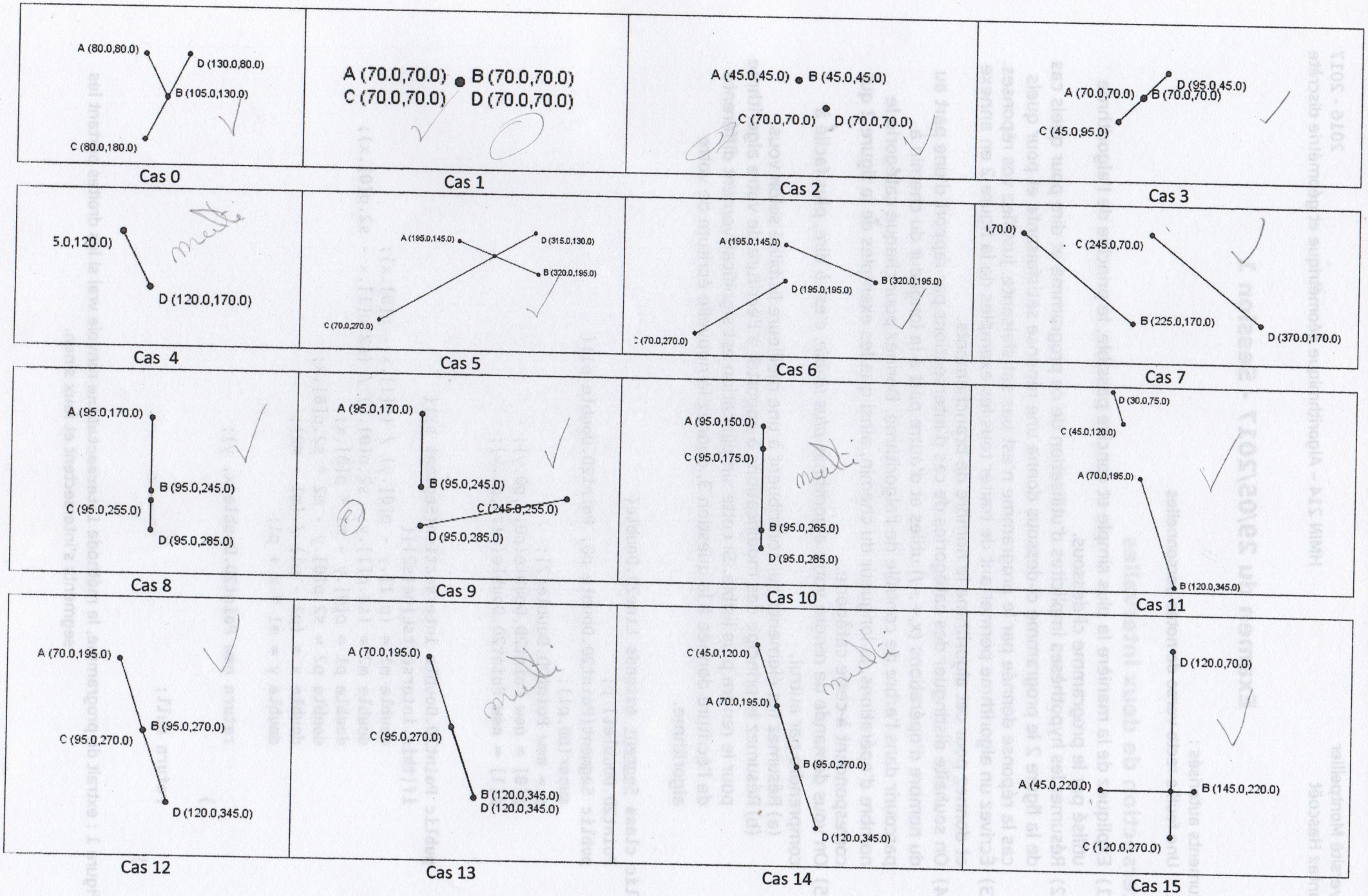

Figure 2 : quelques cas de configurations de deux segments [AB] et [CD] dans le plan