Les calculatrices sont interdites.

N.B. : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction.
Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

PRÉAMBULE : Les deux parties qui composent ce sujet sont indépendantes et peuvent être traitées par les candidats dans un ordre quelconque.

Partie I : Automates et langages

Le but de cet exercice est l'étude des propriétés de l'opération de déterminisation d'un automate. Cette opération a été étudiée dans le cadre du cours. Malgré cela, les réponses aux questions doivent être complètes. Une simple référence au contenu du cours ne suffit pas.

1 Automate fini

Pour simplifier les preuves, nous nous limiterons au cas des automates finis semi-indéterministes, c'est-à-dire les automates finis non déterministes qui ne contiennent pas de transitions instantanées (ou ϵ-transitions). Les résultats étudiés s'étendent au cadre des automates finis quelconques.

1.1 Représentation d'un automate fini

Déf. I.1 (Automate fini semi-indéterministe) *Soit l'alphabet X (un ensemble de symboles), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble contenant Λ et les mots composés de séquences de symboles de X (donc $\Lambda \in X^\star$), un automate fini semi-indéterministe sur X est un quintuplet $A = (Q, X, I, T, \gamma)$ composé de :*

- *Un ensemble fini d'états : Q ;*
- *Un ensemble d'états initiaux : $I \subseteq Q$;*
- *Un ensemble d'états terminaux : $T \subseteq Q$;*
- *Une relation de transition confondue avec son graphe : $\gamma \subseteq Q \times X \times Q$.*

Pour une transition (o, e, d) donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.

Remarquons que γ est le graphe d'une application de transition $\delta : Q \times X \to \mathcal{P}(Q)$ dont les valeurs sont définies par :

$$\forall o \in Q,\ \forall e \in X,\ \delta(o, e) = \{d \in Q \mid (o, e, d) \in \gamma\}$$

La notation γ est plus adaptée que δ à la formalisation et la construction des preuves dans le cadre des automates indéterministes.

Déf. I.2 (Automate fini déterministe) *Soit $\mathcal{A} = (Q, X, I, T, \gamma)$ un automate fini semi-indéterministe, $\mathcal{A}$ est déterministe si et seulement si :*

$$card(I) = 1$$
$$\forall o \in Q, \forall x \in X, card(\{d \in Q \mid (o, x, d) \in \gamma\}) \leq 1$$

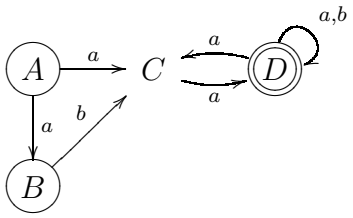
1.2 Représentation graphique d'un automate

Les automates peuvent être représentés par un schéma suivant les conventions :
- les valeurs de la relation de transition γ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- un état initial est entouré d'un cercle (i) ;
- un état terminal est entouré d'un double cercle $((t))$;
- un état qui est à la fois initial et terminal est entouré d'un triple cercle $(((it)))$;
- une arête étiquetée par le symbole $e \in X$ va de l'état o à l'état d si et seulement si $(o, e, d) \in \gamma$.

Exemple I.1 *L'automate $\mathcal{E} = (Q, X, I, T, \gamma)$ avec :*

$$Q = \{A, B, C, D\}$$
$$X = \{a, b\}$$
$$I = \{A, B\}$$
$$T = \{D\}$$
$$\gamma = \{(A, a, B), (A, a, C), (B, b, C), (C, a, D), (D, a, C), (D, a, D), (D, b, D)\}$$

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini

Soit $\gamma^\star$ l'extension de γ à $Q \times X^\star \times Q$ définie par :

$$\forall q \in Q,\ (q, \Lambda, q) \in \gamma^\star$$

$$\left\{ \begin{array}{l} \forall e \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{array} \right. \quad (o, e.m, d) \in \gamma^\star \Leftrightarrow \exists q \in Q,\ ((o, e, q) \in \gamma) \wedge ((q, m, d) \in \gamma^\star)$$

Soit X un alphabet, un langage sur X est un sous-ensemble de $X^\star$.
Le langage sur X reconnu par un automate fini est :

$$L(A) = \{m \in X^\star \mid \exists o \in I,\ \exists d \in T,\ (o, m, d) \in \gamma^\star\}$$

Question I.1 *Donner sans la justifier une expression régulière ou ensembliste représentant le langage sur $X = \{a, b\}$ reconnu par l'automate $\mathcal{E}$ de l'exemple I.1.*

2 Déterminisation d'un automate

2.1 Relation de transition étendue

La relation de transition γ définie sur des états Q peut être étendue à des ensembles d'états de $\mathcal{P}(Q)$ de la manière suivante :

Déf. I.3 (Relation de transition étendue) *Soit $\gamma \subseteq Q \times X \times Q$ une relation de transition sur l'ensemble d'états Q et l'alphabet X, la relation étendue à $\mathcal{P}(Q)$, $[\gamma] \subseteq \mathcal{P}(Q) \times X \times \mathcal{P}(Q)$ est définie par :*

$$\left\{ \begin{array}{l} \forall O \in \mathcal{P}(Q), \\ \forall x \in X, \\ \forall D \in \mathcal{P}(Q), \end{array} \right. (O, x, D) \in [\gamma] \Leftrightarrow \left\{ \begin{array}{l} \forall o \in O, \exists d \in Q, (o, x, d) \in \gamma \Rightarrow d \in D \\ \forall d \in D, \exists o \in O, (o, x, d) \in \gamma \end{array} \right.$$

2.2 Définition

Soit l'opération interne sur les automates finis semi-indéterministes définie par :

Déf. I.4 (Déterminisation d'un automate) *Soit $\mathcal{A} = (Q, X, I, T, \gamma)$ un automate fini semi-indéterministe, l'automate $[\mathcal{A}] = ([Q], X, [I], [T], [\gamma])$ qui résulte de la déterminisation de $\mathcal{A}$ est défini par :*

$$\begin{array}{l} [Q] \subseteq \mathcal{P}(Q) \\ [Q] = \{q \in \mathcal{P}(Q) \mid m \in X^\star, (I, m, q) \in [\gamma]^\star\} \\ [I] = \{I\} \\ [T] = \{t \in [Q] \mid t \cap T \neq \emptyset\} \end{array}$$

Question I.2 *En considérant l'exemple I.1, construire l'automate $[\mathcal{E}]$ (seuls les états et les transitions utiles, c'est-à-dire accessibles depuis les états initiaux, devront être construits).*

Question I.3 *Donner sans la justifier une expression régulière ou ensembliste représentant le langage sur $X = \{a, b\}$ reconnu par l'automate $[\mathcal{E}]$.*

2.3 Propriétés

Question I.4 *Montrer que : si $\mathcal{A}$ est un automate fini semi-indéterministe alors $[\mathcal{A}]$ est un automate fini déterministe.*

Question I.5 *Montrer que :*

$$\left\{ \begin{array}{l} \forall O \in [Q], \\ \forall m \in X^\star, \\ \forall D \in [Q], \end{array} \right. (O, m, D) \in [\gamma]^\star \Leftrightarrow \left\{ \begin{array}{l} \forall o \in O, \exists d \in Q, (o, m, d) \in \gamma^\star \Rightarrow d \in D \\ \forall d \in D, \exists o \in O, (o, m, d) \in \gamma^\star \end{array} \right.$$

Question I.6 *Soit $\mathcal{A}$ un automate fini semi-indéterministe, montrer que :*

$$\forall m \in X^\star, m \in L([\mathcal{A}]) \Leftrightarrow m \in L(\mathcal{A})$$

Question I.7 *Quelle relation peut-on en déduire entre $L([\mathcal{A}])$ et $L(\mathcal{A})$?*

Partie II : Algorithmique et programmation en CaML

Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, ...) ni de références.

Une image numérique est constituée de points disjoints discrets (abscisse et ordonnée dans $\mathbb{N}$). De nombreuses techniques de traitements d'images reposent sur la construction du contour de l'enveloppe convexe d'un ensemble de points distincts. Les techniques les plus courantes reposent sur un parcours de tous les points de l'ensemble pour construire le contour. Ceci est coûteux en temps d'exécution. Nous allons étudier une approche qui permet de réduire le nombre de points parcourus lors de la reconstruction d'un contour après l'ajout ou le retrait d'un point ainsi que lors de la fusion de deux contours. Cette approche est dérivée de la structure d'arbre binaire de recherche adaptée à un espace à deux dimensions. Elle consiste à construire les arbres enveloppants inférieurs et supérieurs pour les points de l'ensemble puis à extraire la partie convexe de leurs contours.

Nous considérerons dans le sujet des ensembles de points dont les abscisses et les ordonnées sont toutes différentes. L'approche étudiée peut être généralisée à des ensembles quelconques. Cette contrainte de séparabilité se définit de la manière suivante :

Déf. II.1 (Points séparables) *Soit un ensemble de points $\mathcal{P}$, les points de $\mathcal{P}$ sont séparables si et seulement si :*

$$\forall (x_1, y_1) \in \mathcal{P}, \forall (x_2, y_2) \in \mathcal{P}, x_1 = x_2 \Leftrightarrow y_1 = y_2$$

Tous les ensembles de points considérés par la suite sont séparables.

1 Représentation du problème

1.1 Notations

Nous nous plaçons dans un espace euclidien orienté de dimension 2 muni d'un repère $(O, \vec{\imath}, \vec{\jmath})$.

- (p, q) représente la droite passant par les points p et q,
- $[p, q]$ représente le segment d'extrémités p et q et $]p, q[= [p, q] \setminus \{p, q\}$,
- $\widehat{pqr}$ est l'angle orienté entre les vecteurs $\overrightarrow{qp}$ et $\overrightarrow{qr}$, également noté $\widehat{\overrightarrow{qp}, \overrightarrow{qr}}$.

1.2 Rappels

Déf. II.2 (Polygone simple) *Un polygone simple $\mathcal{P}$ est une suite $(p_1, \ldots, p_n)$ d'au moins trois points appelés sommets. Ces points forment des segments $[p_i, p_{i+1}]$ (avec $p_{n+1} = p_1$) appelés arêtes tels que :*

- *l'intersection de deux arêtes qui ne sont pas adjacentes est vide,*
- *l'intersection de deux arêtes adjacentes est réduite à leur extrémité commune,*
- *les extrémités de deux arêtes adjacentes ne sont pas alignées.*

La réunion des arêtes du polygone $\mathcal{P}$ s'appelle le contour de $\mathcal{P}$.

Déf. II.3 (Intérieur d'un polygone) *Un polygone simple $\mathcal{P}$ sépare le plan en trois régions :*
- *le contour de $\mathcal{P}$,*
- *une région bornée, l'intérieur de $\mathcal{P}$,*
- *une région non bornée, l'extérieur de $\mathcal{P}$,*

Le contour constitue la frontière entre l'intérieur et l'extérieur de $\mathcal{P}$.

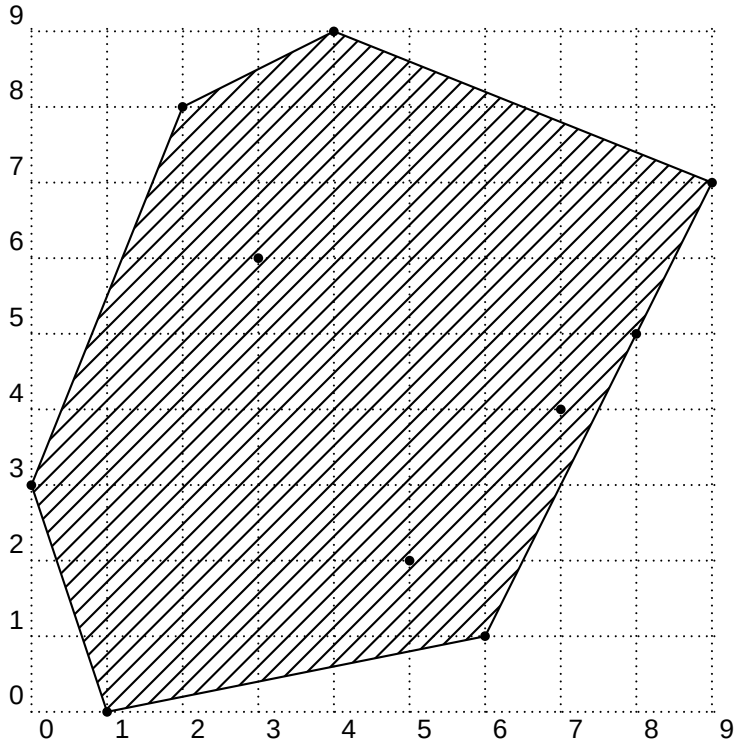
Théorème II.1 *Soit $\mathcal{P}$ un polygone simple, soient p et q deux points distincts du plan qui n'appartiennent pas au contour de $\mathcal{P}$ et tels que le nombre d'intersections entre le segment $[p, q]$ et le contour de $\mathcal{P}$ est fini, alors le nombre d'intersections entre $[p, q]$ et le contour de $\mathcal{P}$ est pair si et seulement si p et q sont tous les deux dans la même région du plan délimitée par le contour de $\mathcal{P}$.*

Déf. II.4 (Enveloppe convexe) *Soit $\mathcal{P}$ un ensemble de points; l'enveloppe convexe de $\mathcal{P}$, notée $\mathcal{E}_\mathcal{C}(\mathcal{P})$, est le plus petit des ensembles convexes contenant $\mathcal{P}$. Dans le cas où $\mathcal{P}$ est un ensemble fini, ce que nous supposons dans la suite, nous admettrons que $\mathcal{E}_\mathcal{C}(\mathcal{P})$ est la réunion du contour d'un polygone convexe et de son intérieur; ce polygone convexe est caractérisé par ses sommets : une suite $\mathcal{S} = (s_1, \ldots, s_n)$ composée de points de $\mathcal{P}$, appelés par extension les sommets de $\mathcal{E}_\mathcal{C}(\mathcal{P})$. Les points de $\mathcal{E}_\mathcal{C}(\mathcal{P})$ sont alors des barycentres à coefficients positifs des points de $\mathcal{S}$, c'est-à-dire :*

$$p \in \mathcal{E}_\mathcal{C}(\mathcal{P}) \Leftrightarrow \exists \{m_i\}_{i\in[1,n]}, m_i \geq 0, \sum_{i=1}^{n} m_i \neq 0, \sum_{i=1}^{n} m_i \overrightarrow{ps_i} = \overrightarrow{0}$$

Les points de la suite $\mathcal{S}$ sont ordonnés de manière à ce que :
- *les segments $[s_i, s_{i+1}]$ sont les arêtes du polygone convexe (avec $s_{n+1} = s_1$), ils forment une ligne polygonale : le contour du polygone,*
- *le contour du polygone est positif, c'est-à-dire orienté dans le sens direct.*

Exemple II.1 *Le schéma suivant représente en hachuré l'enveloppe convexe $\mathcal{E}_\mathcal{C}(\mathcal{P})$ d'un ensemble de points $\mathcal{P} = \{(0,3), (1,0), (2,8), (3,6), (4,9), (5,2), (6,1), (7,4), (8,5), (9,7)\}$. Les sommets de $\mathcal{E}_\mathcal{C}(\mathcal{P})$ sont $\mathcal{S} = ((1,0), (6,1), (9,7), (4,9), (2,8), (0,3))$.*



1.3 Codage en CaML

Un point est représenté par un couple d'entiers :

```
type point == int * int;;
```

Un ensemble, une liste, une suite de points sont représentés par une liste de points :

```
type suite = point list;;
```

Exemple II.2 *L'ensemble de points $\mathcal{P}$ de l'exemple II.1 est représenté par la liste :*

```
[ (0,3);(1,0);(2,8);(3,6);(4,9);(5,2);(6,1);(7,4);(8,5);(9,7) ]
```

La suite des sommets de l'enveloppe convexe de $\mathcal{P}$ est représentée par la liste :

```
[ (1,0);(6,1);(9,7);(4,9);(2,8);(0,3) ]
```

Question II.1 *Écrire en CaML une fonction* `comparer` *de type* `point -> point -> bool` *telle que l'appel* `(comparer p1 p2)` *renvoie la valeur vraie si* `p1` *et* `p2` *ont les mêmes composantes et faux sinon.*

Question II.2 *Écrire en CaML une fonction* `taille` *de type* `suite -> int` *telle que l'appel* `(taille e)` *renvoie le nombre de points contenus dans la suite* `e`*.*

1.4 Produit croisé et fonctions trigonométriques

Déterminer la convexité d'une figure repose sur des propriétés trigonométriques dont le calcul informatique est en général coûteux et source de nombreuses approximations. Pour éviter leurs utilisations, nous allons exploiter le produit croisé qui permet la comparaison d'angles entre vecteurs sans calculer effectivement ces angles.

Déf. II.5 (Produit croisé) *Le produit croisé de trois points p, q et r est égal au déterminant dans une base orthonormale directe des vecteurs $\overrightarrow{qp}$ et $\overrightarrow{qr}$, c'est-à-dire :* $[p,q,r] = Det(\overrightarrow{qp}, \overrightarrow{qr})$.

Question II.3 *Soient p, q et r trois points distincts; montrer que :*

1. $[p,q,r] > 0 \Leftrightarrow \widehat{pqr} \in]0,\pi[$,
2. $[p,q,r] < 0 \Leftrightarrow \widehat{pqr} \in]\pi,2\pi[$,
3. $[p,q,r] = 0 \Leftrightarrow$ *p, q et r sont alignés,*

Question II.4 *Écrire en CaML une fonction* `croise` *de type* `point -> point -> point -> int` *telle que l'appel* `(croise p q r)` *renvoie la valeur de* [`p`, `q`, `r`].

1.5 Séparation du plan par une droite

Déf. II.6 *Soit D une droite qui n'est pas parallèle à l'axe des ordonnées, soient g et d deux points de D tels que l'abscisse de g est strictement inférieure à l'abscisse de d. L'ensemble des points au-dessus de la droite D est $\{p \mid \widehat{dgp} \in]0, \pi[\}$, l'ensemble des points au-dessous de la droite D est $\{p \mid \widehat{dgp} \in]\pi, 2\pi[\}$.*

Théorème II.2 *Un polygone $\mathcal{S}$ de sommets $(s_1, \ldots, s_n)$ est convexe si et seulement si, pour toute arête $[s_i, s_{i+1}]$, les autres sommets de $\mathcal{S}$ sont, soit tous au-dessus, soit tous au-dessous par rapport à la droite (s_i, s_{i+1}).*

Question II.5 *Écrire en CaML une fonction* `separation` *de type* `point -> point -> suite -> (suite * suite * suite)` *telle que l'appel* `(separation p q e)` *renvoie un triplet de listes* `( s, i, a)` *composées de tous les points de l'ensemble* `e` *qui :*

- *sont distincts de* `p` *et* `q`*,*
- *se trouvent au-dessus de la droite* `(p,q)` *pour la liste* `s`*,*
- *se trouvent au-dessous de la droite* `(p,q)` *pour la liste* `i`*,*
- *appartiennent à la droite* `(p,q)` *pour la liste* `a`*.*

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.6 *Calculer une estimation de la complexité de la fonction* `separation` *en fonction de la taille de l'ensemble* `e`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Décomposition en sous-problèmes disjoints

Pour permettre la représentation d'un ensemble de points $\mathcal{P}$ par un arbre pour construire efficacement le contour de l'enveloppe convexe $\mathcal{E}_\mathcal{C}(\mathcal{P})$, nous allons décomposer cet ensemble en deux sous-ensembles disjoints séparés par la droite reliant le point le plus à gauche et le point le plus à droite de l'ensemble. Nous construirons ensuite l'arbre enveloppant inférieur qui correspond aux points au-dessous de la droite et l'arbre enveloppant supérieur qui correspond aux points au-dessus de la droite.

Déf. II.7 (Point le plus à gauche, le plus à droite) *Dans les hypothèses de cette étude, le point le plus à gauche, respectivement le plus à droite, est celui dont l'abscisse est la plus petite, respectivement la plus grande.*

Question II.7 *Écrire en CaML une fonction* `extremes` *de type* `suite -> (point * point)` *telle que l'appel* `(extremes e)`*, sur un ensemble non vide de points séparables* `e`*, renvoie un couple de points* `(g,d)` *de l'ensemble* `e`*. Le point* `g`*, respectivement* `d`*, doit être le point le plus à gauche, respectivement à droite, des points de* `e`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.8 *Soit $\mathcal{P}$ un ensemble de points, soit $G \in \mathcal{P}$, respectivement $D \in \mathcal{P}$, le point le plus à gauche, respectivement à droite, de $\mathcal{P}$, montrer que G et D sont des sommets de l'enveloppe convexe $\mathcal{E}_\mathcal{C}(\mathcal{P})$.*

Théorème II.3 *Soit $\mathcal{P}$ un ensemble de points; soient :*
- *G, respectivement D, le point le plus à gauche, respectivement le plus à droite, de $\mathcal{P}$,*
- *$\mathcal{P}_\mathcal{S}$ les points au-dessus de la droite (G, D),*
- *$\mathcal{P}_\mathcal{I}$ les points au-dessous de la droite (G, D),*
- *$\mathcal{P}_{\mathcal{GD}}$ les points de $]G, D[$,*

alors $\{G\} \cup \mathcal{P}_\mathcal{I} \cup \{D\} \cup \mathcal{P}_\mathcal{S} \cup \mathcal{P}_{\mathcal{GD}}$ forme une partition de $\mathcal{P}$
et $\mathcal{E}_\mathcal{C}(\mathcal{P}) = \mathcal{E}_\mathcal{C}(\{G, D\} \cup \mathcal{P}_\mathcal{S}) \cup \mathcal{E}_\mathcal{C}(\{G, D\} \cup \mathcal{P}_\mathcal{I})$.

Exemple II.3 *L'ensemble de points $\mathcal{P}$ de l'exemple II.1 est partitionné en :*
- $G = (0, 3)$
- $D = (9, 7)$
- $\mathcal{P}_\mathcal{I} = \{(1, 0), (5, 2), (6, 1), (7, 4), (8, 5)\}$
- $\mathcal{P}_\mathcal{S} = \{(2, 8); (3, 6); (4, 9)\}$
- $\mathcal{P}_{\mathcal{BH}} = \emptyset$

3 Structure d'arbres enveloppants

L'utilisation de la structure d'arbre enveloppant permet de réduire la complexité en temps de calcul de l'opération de construction du contour de l'enveloppe convexe d'un ensemble de points. Dans ce but, $\mathcal{P}_\mathcal{S} \cup \{G, D\}$, respectivement $\mathcal{P}_\mathcal{I} \cup \{G, D\}$, doit être organisé sous la forme d'un arbre enveloppant supérieur, respectivement inférieur. Nous ne traiterons par la suite que le cas inférieur car le cas supérieur s'obtient ensuite par symétrie.

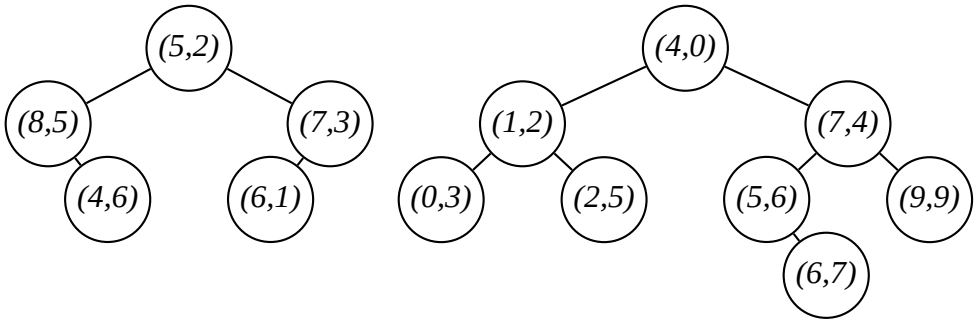
3.1 Arbres binaires de points

Un ensemble de points peut être implanté par un arbre binaire en utilisant les étiquettes des nœuds pour représenter les points contenus dans l'ensemble.

3.1.1 Définition

Déf. II.8 (Arbre binaire de points) *Un arbre binaire de points a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient un point comme étiquette (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires de points. L'ensemble de tous les nœuds de l'arbre a est noté $\mathcal{N}(a)$.*

Exemple II.4 (Arbres binaires de points) *Voici deux exemples d'arbres binaires étiquetés par des points (les sous-arbres vides fils des nœuds ne sont pas représentés) :*



3.1.2 Profondeur d'un arbre

Déf. II.9 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

Exemple II.5 (Profondeurs) *Les profondeurs des deux arbres binaires de l'exemple II.4 sont respectivement 3 et 4.*

Question II.9 *Donner une définition de la profondeur d'un arbre a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

Question II.10 *Considérons un arbre binaire de points représentant un ensemble contenant n points distincts. Quelle est la forme de l'arbre dont la profondeur est maximale ? Quelle est la forme de l'arbre dont la profondeur est minimale ? Calculer la profondeur de l'arbre en fonction de n dans ces deux cas. Vous justifierez vos réponses.*

3.1.3 Représentation des arbres binaires en CaML

Un arbre binaire de points est représenté par le type CaML :

```
type arbre = Vide | Noeud of arbre * point * arbre;;
```

Dans l'appel `Noeud( fg, p, fd)`, les paramètres `fg`, `p` et `fd` sont respectivement le fils gauche, l'étiquette et le fils droit de la racine de l'arbre créé.

Exemple II.6 *Le terme suivant*

```
Noeud(
   Noeud(
      Vide,
      (8,5),
      Noeud( Vide, (4,6), Vide)),
   (5,2),
   Noeud(
      Noeud( Vide, (6,1), Vide),
      (7,3),
      Vide))
```

est alors associé au premier arbre binaire représenté graphiquement dans l'exemple II.4.

3.1.4 Profondeur d'un arbre binaire de points

Question II.11 *Écrire en CaML une fonction `profondeur` de type `arbre -> int` telle que l'appel `(profondeur a)` renvoie la profondeur de l'arbre binaire de points `a`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.2 Arbres enveloppants inférieurs

Déf. II.10 (Arbre enveloppant inférieur) *Un arbre enveloppant inférieur est un arbre binaire de points tel que :*

- *son fils gauche, s'il existe, est un arbre enveloppant inférieur,*
- *son fils droit, s'il existe, est un arbre enveloppant inférieur,*
- *les étiquettes de ses deux fils ont une ordonnée supérieure à celle de son étiquette,*
- *les étiquettes de tous les nœuds de son fils gauche, s'il existe, ont une abscisse inférieure à celle de son étiquette,*
- *les étiquettes de tous les nœuds de son fils droit, s'il existe, ont une abscisse supérieure à celle de son étiquette.*

Exemple II.7 (Arbre enveloppant inférieur) *Le deuxième arbre de l'exemple II.4 est un arbre enveloppant inférieur.*

Question II.12 *Construire l'arbre enveloppant inférieur pour les points de l'ensemble $\mathcal{P}_{\mathcal{I}} \cup \{G, D\}$ correspondant à l'exemple II.3*

Question II.13 *Exprimer la définition II.10 sous la forme d'une propriété $AEI(a)$ qui indique que a est un arbre enveloppant inférieur. Vous pouvez utiliser pour cela $\emptyset$ et les fonctions définies sur les arbres binaires de points $\mathcal{G}$ et $\mathcal{D}$ et $\mathcal{N}$.*

4 Application au calcul d'une enveloppe convexe

Déf. II.11 (Contour direct d'un arbre enveloppant) *Le contour direct d'un arbre enveloppant est une suite de points composée des points de la branche la plus à gauche et des points de la branche la plus à droite. Ces points sont numérotés dans le sens direct. Pour un arbre enveloppant inférieur, le premier point est le point le plus à gauche, le dernier point est le point le plus à droite. La branche la plus à gauche est numérotée de haut en bas. La branche la plus à droite est numérotée de bas en haut.*

Exemple II.8 *La suite de points suivante est le contour direct du second arbre de l'exemple II.4 :*

$$((0,3),(1,2),(4,0),(7,4),(9,9))$$

Soient G et D deux points, soit $\mathcal{P}$ un ensemble de points tel que G, respectivement D, soit le point le plus à gauche, respectivement le plus à droite, de $\mathcal{P}$, et que tous les points de $\mathcal{P} \setminus \{G, D\}$ soient au-dessous de la droite (G, D). Soit $\mathcal{C}$ le contour direct de l'arbre enveloppant inférieur construit à partir de $\mathcal{P}$.

Question II.14 *Montrer que $\mathcal{C}$ est un polygone tel que tous les points de $\mathcal{P}$ soient à l'intérieur ou sur le contour de ce polygone.*

Question II.15 *Écrire en CaML une fonction `developper` de type `arbre -> suite` telle que l'appel `(developper a)` renvoie le contour direct de l'arbre enveloppant inférieur `a`. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.16 *Montrer que les sommets de l'enveloppe convexe de $\mathcal{P}$ appartiennent au contour direct $\mathcal{C}$.*

Question II.17 *Soient p_n, p_{n+1} et p_{n+2} 3 points consécutifs du contour $\mathcal{C}$, montrer que si $[p_n, p_{n+1}, p_{n+2}] \geq 0$ alors p_{n+1} n'est pas un sommet de $\mathcal{E}_{\mathcal{C}}(\mathcal{P})$.*

Question II.18 *Soit la sous-suite maximale de $\mathcal{C}$ telle que pour tout triplet (p_n, p_{n+1}, p_{n+2}) de points consécutifs de cette sous-suite, $[p_n, p_{n+1}, p_{n+2}] < 0$. Montrer que cette sous-suite est égale à la suite des sommets de $\mathcal{E}_{\mathcal{C}}(\mathcal{P})$.*

Question II.19 *Écrire en CaML une fonction `filtrer` de type `suite -> suite` telle que l'appel `(filtrer s)` renvoie la suite des sommets de l'enveloppe convexe de $\mathcal{P}$, si `s` désigne la suite des points constituant le contour direct $\mathcal{C}$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

5 Opérations sur un arbre enveloppant inférieur

5.1 Recherche dans un arbre enveloppant inférieur

Une première opération consiste à déterminer si un arbre enveloppant inférieur contient un point particulier.

Question II.20 *Écrire en CaML une fonction `contenir` de type `point -> arbre -> bool` telle que l'appel `(contenir p a)` renvoie la valeur vraie si l'arbre enveloppant inférieur `a` contient le point `p` et la valeur faux sinon. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.21 *Donner des exemples de valeurs des paramètres `p` et `a` de la fonction `contenir` qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Donner une estimation de la complexité dans le meilleur et pire cas de la fonction `contenir` en fonction du nombre de points dans `a`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

5.2 Ajout d'un point dans un arbre enveloppant

5.2.1 Scission d'un arbre enveloppant inférieur

Une deuxième opération consiste à scinder un arbre enveloppant inférieur en deux selon une abscisse tout en préservant cette structure. Elle produit deux arbres enveloppants inférieurs : celui dont les points ont des abscisses inférieures, celui dont les points ont des abscisses supérieures.

Question II.22 *Écrire en CaML une fonction `scinder` de type `int -> arbre -> (arbre * arbre)` telle que l'appel `(scinder x a)` renvoie une paire d'arbres enveloppants inférieurs dont la première composante est la partie de l'arbre enveloppant inférieur `a` contenant les points dont les abscisses sont supérieures à `x` et la seconde composante est la partie de l'arbre enveloppant inférieur `a` contenant les points dont les abscisses sont supérieures à `x`. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

5.2.2 Insertion d'un point

Une troisième opération consiste à insérer un élément dans un arbre enveloppant inférieur en préservant cette structure.

Question II.23 *Écrire en CaML une fonction* `ajouter` *de type* `point -> arbre -> arbre` *telle que l'appel* `(ajouter p a)` *renvoie un arbre enveloppant inférieur contenant les mêmes points que l'arbre enveloppant inférieur* `a` *ainsi que le point* `p`*. Nous supposons que* `a` *ne contient pas déjà* `p` *et qu'aucun point de* `a` *n'a la même abscisse ou la même ordonnée que* `p`*. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.24 *Donner une estimation de la complexité dans le meilleur cas de la fonction* `ajouter` *en fonction du nombre de points contenus dans* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Partie II : Algorithmique et programmation en PASCAL

Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, `repeat`, ...).

Une image numérique est constituée de points disjoints discrets (abscisse et ordonnée dans $\mathbb{N}$). De nombreuses techniques de traitements d'images reposent sur la construction du contour de l'enveloppe convexe d'un ensemble de points distincts. Les techniques les plus courantes reposent sur un parcours de tous les points de l'ensemble pour construire le contour. Ceci est coûteux en temps d'exécution. Nous allons étudier une approche qui permet de réduire le nombre de points parcourus lors de la reconstruction d'un contour après l'ajout ou le retrait d'un point ainsi que lors de la fusion de deux contours. Cette approche est dérivée de la structure d'arbre binaire de recherche adaptée à un espace à deux dimensions. Elle consiste à construire les arbres enveloppants inférieurs et supérieurs pour les points de l'ensemble puis à extraire la partie convexe de leurs contours.

Nous considérerons dans le sujet des ensembles de points dont les abscisses et les ordonnées sont toutes différentes. L'approche étudiée peut être généralisée à des ensembles quelconques. Cette contrainte de séparabilité se définit de la manière suivante :

Déf. II.1 (Points séparables) *Soit un ensemble de points $\mathcal{P}$, les points de $\mathcal{P}$ sont séparables si et seulement si :*

$$\forall (x_1, y_1) \in \mathcal{P}, \forall (x_2, y_2) \in \mathcal{P}, x_1 = x_2 \Leftrightarrow y_1 = y_2$$

Tous les ensembles de points considérés par la suite sont séparables.

1 Représentation du problème

1.1 Notations

Nous nous plaçons dans un espace euclidien orienté de dimension 2 muni d'un repère $(O, \vec{\imath}, \vec{\jmath})$.

- (p, q) représente la droite passant par les points p et q,
- $[p, q]$ représente le segment d'extrémités p et q et $]p, q[= [p, q] \setminus \{p, q\}$,
- $\widehat{pqr}$ est l'angle orienté entre les vecteurs $\overrightarrow{qp}$ et $\overrightarrow{qr}$, également noté $\widehat{\overrightarrow{qp}, \overrightarrow{qr}}$.

1.2 Rappels

Déf. II.2 (Polygone simple) *Un polygone simple $\mathcal{P}$ est une suite $(p_1, \ldots, p_n)$ d'au moins trois points appelés sommets. Ces points forment des segments $[p_i, p_{i+1}]$ (avec $p_{n+1} = p_1$) appelés arêtes tels que :*

- *l'intersection de deux arêtes qui ne sont pas adjacentes est vide,*
- *l'intersection de deux arêtes adjacentes est réduite à leur extrémité commune,*
- *les extrémités de deux arêtes adjacentes ne sont pas alignées.*

La réunion des arêtes du polygone $\mathcal{P}$ s'appelle le contour de $\mathcal{P}$.

Déf. II.3 (Intérieur d'un polygone) *Un polygone simple $\mathcal{P}$ sépare le plan en trois régions :*
- *le contour de $\mathcal{P}$,*
- *une région bornée, l'intérieur de $\mathcal{P}$,*
- *une région non bornée, l'extérieur de $\mathcal{P}$,*

Le contour constitue la frontière entre l'intérieur et l'extérieur de $\mathcal{P}$.

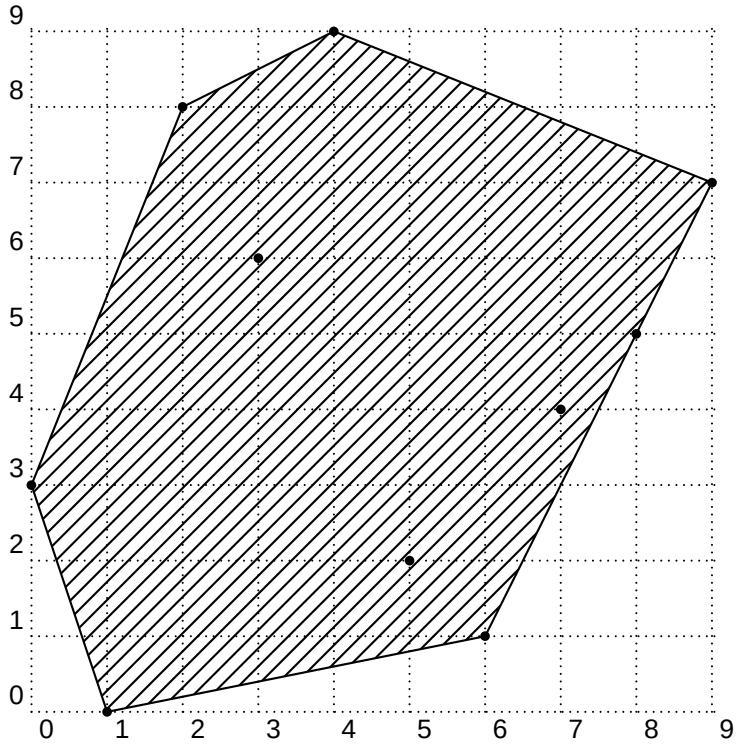
Théorème II.1 *Soit $\mathcal{P}$ un polygone simple, soient p et q deux points distincts du plan qui n'appartiennent pas au contour de $\mathcal{P}$ et tels que le nombre d'intersections entre le segment $[p, q]$ et le contour de $\mathcal{P}$ est fini, alors le nombre d'intersections entre $[p, q]$ et le contour de $\mathcal{P}$ est pair si et seulement si p et q sont tous les deux dans la même région du plan délimitée par le contour de $\mathcal{P}$.*

Déf. II.4 (Enveloppe convexe) *Soit $\mathcal{P}$ un ensemble de points; l'enveloppe convexe de $\mathcal{P}$, notée $\mathcal{E}_\mathcal{C}(\mathcal{P})$, est le plus petit des ensembles convexes contenant $\mathcal{P}$. Dans le cas où $\mathcal{P}$ est un ensemble fini, ce que nous supposons dans la suite, nous admettrons que $\mathcal{E}_\mathcal{C}(\mathcal{P})$ est la réunion du contour d'un polygone convexe et de son intérieur; ce polygone convexe est caractérisé par ses sommets : une suite $\mathcal{S} = (s_1, \ldots, s_n)$ composée de points de $\mathcal{P}$, appelés par extension les sommets de $\mathcal{E}_\mathcal{C}(\mathcal{P})$. Les points de $\mathcal{E}_\mathcal{C}(\mathcal{P})$ sont alors des barycentres à coefficients positifs des points de $\mathcal{S}$, c'est-à-dire :*

$$p \in \mathcal{E}_\mathcal{C}(\mathcal{P}) \Leftrightarrow \exists \{m_i\}_{i\in[1,n]}, m_i \geq 0, \sum_{i=1}^{n} m_i \neq 0, \sum_{i=1}^{n} m_i \overrightarrow{ps_i} = \overrightarrow{0}$$

Les points de la suite $\mathcal{S}$ sont ordonnés de manière à ce que :
- *les segments $[s_i, s_{i+1}]$ sont les arêtes du polygone convexe (avec $s_{n+1} = s_1$), ils forment une ligne polygonale : le contour du polygone,*
- *le contour du polygone est positif, c'est-à-dire orienté dans le sens direct.*

Exemple II.1 *Le schéma suivant représente en hachuré l'enveloppe convexe $\mathcal{E}_\mathcal{C}(\mathcal{P})$ d'un ensemble de points $\mathcal{P} = \{(0,3), (1,0), (2,8), (3,6), (4,9), (5,2), (6,1), (7,4), (8,5), (9,7)\}$. Les sommets de $\mathcal{E}_\mathcal{C}(\mathcal{P})$ sont $\mathcal{S} = ((1,0), (6,1), (9,7), (4,9), (2,8), (0,3))$.*



1.3 Codage en PASCAL

Un point est représenté par le type de base `POINT`.

Un ensemble, une liste, une suite de points sont représentés par le type de base `SUITE`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION P_Creer(a,o:INTEGER):POINT;` renvoie un point dont l'abscisse est `a` et l'ordonnée `o`,
- `FUNCTION P_Abs(p:POINT):INTEGER;` renvoie l'abscisse de `p`,
- `FUNCTION P_Ord(p:POINT):INTEGER;` renvoie l'ordonnée de `p`,
- `NIL` représente la suite vide de points,
- `FUNCTION S_Ajouter(p:POINT;s:SUITE):SUITE;` renvoie une suite de points composée d'un premier point `p` et du reste de la suite contenu dans `s`,
- `FUNCTION S_Premier(s:SUITE):POINT;` renvoie le premier point de la suite `s`,
- `FUNCTION S_Reste(s:SUITE):SUITE;` renvoie le reste de la suite `s` privée de son premier point,
- `FUNCTION S_Juxtaposer(s1,s2:SUITE):SUITE;` renvoie la suite composée de la suite `s1` suivie de la suite `s2`.

Exemple II.2 *L'ensemble de points $\mathcal{P}$ de l'exemple II.1 est représenté par la liste :*

```
S_Ajouter( P_Creer(0,3), S_Ajouter( P_Creer(1,0),
S_Ajouter( P_Creer(2,8), S_Ajouter( P_Creer(3,6),
S_Ajouter( P_Creer(4,9), S_Ajouter( P_Creer(5,2),
S_Ajouter( P_Creer(6,1), S_Ajouter( P_Creer(7,4),
S_Ajouter( P_Creer(8,5), S_Ajouter( P_Creer(9,7), NIL))))))))))
```

La suite des sommets de l'enveloppe convexe de $\mathcal{P}$ est représentée par la liste :

```
S_Ajouter( P_Creer(1,0), S_Ajouter( P_Creer(6,1),
S_Ajouter( P_Creer(9,7), S_Ajouter( P_Creer(4,9),
S_Ajouter( P_Creer(2,8), S_Ajouter( P_Creer(0,3), NIL))))))
```

Question II.1 *Écrire en PASCAL une fonction `comparer(p1,p2:POINT):BOOLEAN;` telle que l'appel `comparer(p1,p2)` renvoie la valeur vraie si `p1` et `p2` ont les mêmes composantes et faux sinon.*

Question II.2 *Écrire en PASCAL une fonction `taille(e:SUITE):INTEGER;` telle que l'appel `taille(e)` renvoie le nombre de points contenus dans la suite `e`.*

1.4 Produit croisé et fonctions trigonométriques

Déterminer la convexité d'une figure repose sur des propriétés trigonométriques dont le calcul informatique est en général coûteux et source de nombreuses approximations. Pour éviter leurs utilisations, nous allons exploiter le produit croisé qui permet la comparaison d'angles entre vecteurs sans calculer effectivement ces angles.

Déf. II.5 (Produit croisé) *Le produit croisé de trois points p, q et r est égal au déterminant dans une base orthonormale directe des vecteurs $\overrightarrow{qp}$ et $\overrightarrow{qr}$, c'est-à-dire :* $[p,q,r] = Det(\overrightarrow{qp}, \overrightarrow{qr})$.

Question II.3 *Soient p, q et r trois points distincts; montrer que :*

1. $[p,q,r] > 0 \Leftrightarrow \widehat{pqr} \in]0,\pi[$,
2. $[p,q,r] < 0 \Leftrightarrow \widehat{pqr} \in]\pi,2\pi[$,
3. $[p,q,r] = 0 \Leftrightarrow p$, q *et* r *sont alignés,*

Question II.4 *Écrire en PASCAL une fonction* `croise(p,q,r:POINT):INTEGER;` *telle que l'appel* `croise(p,q,r)` *renvoie la valeur de* [`p`, `q`, `r`].

1.5 Séparation du plan par une droite

Déf. II.6 *Soit D une droite qui n'est pas parallèle à l'axe des ordonnées, soient g et d deux points de D tels que l'abscisse de g est strictement inférieure à l'abscisse de d. L'ensemble des points au-dessus de la droite D est $\{p \mid \widehat{dgp} \in]0,\pi[\}$, l'ensemble des points au-dessous de la droite D est $\{p \mid \widehat{dgp} \in]\pi,2\pi[\}$.*

Théorème II.2 *Un polygone $\mathcal{S}$ de sommets $(s_1,\ldots,s_n)$ est convexe si et seulement si, pour toute arête $[s_i,s_{i+1}]$, les autres sommets de $\mathcal{S}$ sont, soit tous au-dessus, soit tous au-dessous par rapport à la droite (s_i,s_{i+1}).*

Un triplet de suites de points est représenté par le type de base `TRIPLET`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION T_Creer(s1,s2,s3:SUITE):TRIPLET;` renvoie un triplet dont les éléments sont `s1`, `s2` et `s3`,
- `FUNCTION T_Un(t:TRIPLET):SUITE;` renvoie le premier élément de `t`,
- `FUNCTION T_Deux(t:TRIPLET):SUITE;` renvoie le deuxième élément de `t`,
- `FUNCTION T_Trois(t:TRIPLET):SUITE;` renvoie le troisième élément de `t`.

Question II.5 *Écrire en PASCAL une fonction* `separation(p,q:POINT;e:SUITE):TRIPLET;` *telle que l'appel* `separation(p,q,e)` *renvoie un triplet* `T_Creer(s,i,a)` *de listes composées de tous les points de l'ensemble* `e` *qui :*

- *sont distincts de* `p` *et* `q`,
- *se trouvent au-dessus de la droite* `(p,q)` *pour la liste* `s`,
- *se trouvent au-dessous de la droite* `(p,q)` *pour la liste* `i`,
- *appartiennent à la droite* `(p,q)` *pour la liste* `a`.

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.6 *Calculer une estimation de la complexité de la fonction* `separation` *en fonction de la taille de l'ensemble* `e`. *Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Décomposition en sous-problèmes disjoints

Pour permettre la représentation d'un ensemble de points $\mathcal{P}$ par un arbre pour construire efficacement le contour de l'enveloppe convexe $\mathcal{E}_\mathcal{C}(\mathcal{P})$, nous allons décomposer cet ensemble en deux sous-ensembles disjoints séparés par la droite reliant le point le plus à gauche et le point le plus à droite de l'ensemble. Nous construirons ensuite l'arbre enveloppant inférieur qui correspond aux

points au-dessous de la droite et l'arbre enveloppant supérieur qui correspond aux points au-dessus de la droite.

Déf. II.7 (Point le plus à gauche, le plus à droite) *Dans les hypothèses de cette étude, le point le plus à gauche, respectivement le plus à droite, est celui dont l'abscisse est la plus petite, respectivement la plus grande.*

Un couple de points est représenté par le type de base `COUPLE`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION C_Creer(p1,p2:POINT):COUPLE;` renvoie un couple dont les éléments sont `p1` et `p2`,
- `FUNCTION C_Un(c:COUPLE):POINT;` renvoie le premier élément de `c`,
- `FUNCTION C_Deux(c:COUPLE):POINT;` renvoie le second élément de `c`.

Question II.7 *Écrire en PASCAL une fonction `extremes(e:SUITE):COUPLE;` telle que l'appel `extremes(e)`, sur un ensemble non vide de points séparables `e`, renvoie un couple de points `C_Creer(g,d)` de l'ensemble `e`. Le point `g`, respectivement `d`, est le point le plus à gauche, respectivement à droite, des points de `e`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.8 *Soit $\mathcal{P}$ un ensemble de points, soit $G \in \mathcal{P}$, respectivement $D \in \mathcal{P}$, le point le plus à gauche, respectivement à droite, de $\mathcal{P}$, montrer que G et D sont des sommets de l'enveloppe convexe $\mathcal{E}_{\mathcal{C}}(\mathcal{P})$.*

Théorème II.3 *Soit $\mathcal{P}$ un ensemble de points; soient :*

- *G, respectivement D, le point le plus à gauche, respectivement le plus à droite, de $\mathcal{P}$,*
- *$\mathcal{P}_{\mathcal{S}}$ les points au-dessus de la droite (G, D),*
- *$\mathcal{P}_{\mathcal{I}}$ les points au-dessous de la droite (G, D),*
- *$\mathcal{P}_{\mathcal{GD}}$ les points de $]G, D[$,*

alors $\{G\} \cup \mathcal{P}_{\mathcal{I}} \cup \{D\} \cup \mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{GD}}$ forme une partition de $\mathcal{P}$
et $\mathcal{E}_{\mathcal{C}}(\mathcal{P}) = \mathcal{E}_{\mathcal{C}}(\{G, D\} \cup \mathcal{P}_{\mathcal{S}}) \cup \mathcal{E}_{\mathcal{C}}(\{G, D\} \cup \mathcal{P}_{\mathcal{I}})$.

Exemple II.3 *L'ensemble de points $\mathcal{P}$ de l'exemple II.1 est partitionné en :*

- $G = (0, 3)$
- $D = (9, 7)$
- $\mathcal{P}_{\mathcal{I}} = \{(1, 0), (5, 2), (6, 1), (7, 4), (8, 5)\}$
- $\mathcal{P}_{\mathcal{S}} = \{(2, 8); (3, 6); (4, 9)\}$
- $\mathcal{P}_{\mathcal{BH}} = \emptyset$

3 Structure d'arbres enveloppants

L'utilisation de la structure d'arbre enveloppant permet de réduire la complexité en temps de calcul de l'opération de construction du contour de l'enveloppe convexe d'un ensemble de points. Dans ce but, $\mathcal{P}_{\mathcal{S}} \cup \{G, D\}$, respectivement $\mathcal{P}_{\mathcal{I}} \cup \{G, D\}$, doit être organisé sous la forme d'un arbre enveloppant supérieur, respectivement inférieur. Nous ne traiterons par la suite que le cas inférieur car le cas supérieur s'obtient ensuite par symétrie.

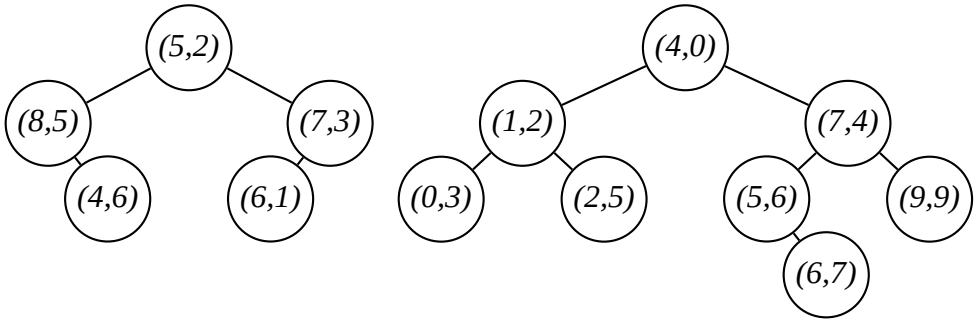
3.1 Arbres binaires de points

Un ensemble de points peut être implanté par un arbre binaire en utilisant les étiquettes des nœuds pour représenter les points contenus dans l'ensemble.

3.1.1 Définition

Déf. II.8 (Arbre binaire de points) *Un arbre binaire de points a est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient un point comme étiquette (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires de points. L'ensemble de tous les nœuds de l'arbre a est noté $\mathcal{N}(a)$.*

Exemple II.4 (Arbres binaires de points) *Voici deux exemples d'arbres binaires étiquetés par des points (les sous-arbres vides fils des nœuds ne sont pas représentés) :*



3.1.2 Profondeur d'un arbre

Déf. II.9 (Profondeur d'un arbre) *Les branches d'un arbre relient la racine aux feuilles. La profondeur d'un arbre A est égale au nombre de nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$.*

Exemple II.5 (Profondeurs) *Les profondeurs des deux arbres binaires de l'exemple II.4 sont respectivement 3 et 4.*

Question II.9 *Donner une définition de la profondeur d'un arbre a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

Question II.10 *Considérons un arbre binaire de points représentant un ensemble contenant n points distincts. Quelle est la forme de l'arbre dont la profondeur est maximale ? Quelle est la forme de l'arbre dont la profondeur est minimale ? Calculer la profondeur de l'arbre en fonction de n dans ces deux cas. Vous justifierez vos réponses.*

3.1.3 Représentation des arbres binaires en PASCAL

Un arbre binaire de points est représenté par le type de base `ARBRE`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `Vide` est une constante de valeur `NIL` qui représente un arbre ou une liste vide ;
- `FUNCTION Noeud(fg:ARBRE;p:POINT;fd:ARBRE):ARBRE;` est une fonction qui renvoie un arbre dont le fils gauche, l'étiquette et le fils droit de la racine sont respectivement `fg`, `p` et `fd`,

- `FUNCTION Gauche(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a` ;
- `FUNCTION Etiquette(a:ARBRE):POINT;` est une fonction qui renvoie l'étiquette de la racine de l'arbre `a` ;
- `FUNCTION Droit(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils droit de la racine de l'arbre `a`.

Exemple II.6 *L'appel suivant*

```
Noeud(
   Noeud(
      Vide,
      P_Creer(8,5),
      Noeud( Vide, P_Creer(4,6), Vide)),
   P_Creer(5,2),
   Noeud(
      Noeud( Vide, P_Creer(6,1), Vide),
      P_Creer(7,3),
      Vide))
```

renvoie le premier arbre binaire de points représenté graphiquement dans l'exemple II.4.

3.1.4 Profondeur d'un arbre binaire de points

Question II.11 *Écrire en PASCAL une fonction* `profondeur(a:ARBRE):INTEGER;` *telle que l'appel* `profondeur(a)` *renvoie la profondeur de l'arbre binaire de points* `a`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.2 Arbres enveloppants inférieurs

Déf. II.10 (Arbre enveloppant inférieur) *Un arbre enveloppant inférieur est un arbre binaire de points tel que :*

- *son fils gauche, s'il existe, est un arbre enveloppant inférieur,*
- *son fils droit, s'il existe, est un arbre enveloppant inférieur,*
- *les étiquettes de ses deux fils ont une ordonnée supérieure à celle de son étiquette,*
- *les étiquettes de tous les nœuds de son fils gauche, s'il existe, ont une abscisse inférieure à celle de son étiquette,*
- *les étiquettes de tous les nœuds de son fils droit, s'il existe, ont une abscisse supérieure à celle de son étiquette.*

Exemple II.7 (Arbre enveloppant inférieur) *Le deuxième arbre de l'exemple II.4 est un arbre enveloppant inférieur.*

Question II.12 *Construire l'arbre enveloppant inférieur pour les points de l'ensemble $\mathcal{P}_\mathcal{I} \cup \{G, D\}$ correspondant à l'exemple II.3*

Question II.13 *Exprimer la définition II.10 sous la forme d'une propriété $AEI(a)$ qui indique que a est un arbre enveloppant inférieur. Vous pouvez utiliser pour cela $\emptyset$ et les fonctions définies sur les arbres binaires de points $\mathcal{G}$ et $\mathcal{D}$ et $\mathcal{N}$.*

4 Application au calcul d'une enveloppe convexe

Déf. II.11 (Contour direct d'un arbre enveloppant) *Le contour direct d'un arbre enveloppant est une suite de points composée des points de la branche la plus à gauche et des points de la branche la plus à droite. Ces points sont numérotés dans le sens direct. Pour un arbre enveloppant inférieur, le premier point est le point le plus à gauche, le dernier point est le point le plus à droite. La branche la plus à gauche est numérotée de haut en bas. La branche la plus à droite est numérotée de bas en haut.*

Exemple II.8 *La suite de points suivante est le contour direct du second arbre de l'exemple II.4 :*

$$((0,3),(1,2),(4,0),(7,4),(9,9))$$

Soient G et D deux points, soit $\mathcal{P}$ un ensemble de points tel que G, respectivement D, soit le point le plus à gauche, respectivement le plus à droite, de $\mathcal{P}$, et que tous les points de $\mathcal{P} \setminus \{G, D\}$ soient au-dessous de la droite (G, D). Soit $\mathcal{C}$ le contour direct de l'arbre enveloppant inférieur construit à partir de $\mathcal{P}$.

Question II.14 *Montrer que $\mathcal{C}$ est un polygone tel que tous les points de $\mathcal{P}$ soient à l'intérieur ou sur le contour de ce polygone.*

Question II.15 *Écrire en PASCAL une fonction* `developper(a:ARBRE):SUITE;` *telle que l'appel* `developper(a)` *renvoie le contour direct de l'arbre enveloppant inférieur* `a`*. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.16 *Montrer que les sommets de l'enveloppe convexe de $\mathcal{P}$ appartiennent au contour direct $\mathcal{C}$.*

Question II.17 *Soient p_n, p_{n+1} et p_{n+2} 3 points consécutifs du contour $\mathcal{C}$, montrer que si $[p_n, p_{n+1}, p_{n+2}] \geq 0$ alors p_{n+1} n'est pas un sommet de $\mathcal{E}_{\mathcal{C}}(\mathcal{P})$.*

Question II.18 *Soit la sous-suite maximale de $\mathcal{C}$ telle que pour tout triplet (p_n, p_{n+1}, p_{n+2}) de points consécutifs de cette sous-suite, $[p_n, p_{n+1}, p_{n+2}] < 0$. Montrer que cette sous-suite est égale à la suite des sommets de $\mathcal{E}_{\mathcal{C}}(\mathcal{P})$.*

Question II.19 *Écrire en PASCAL une fonction* `filtrer(s:SUITE):SUITE;` *telle que l'appel* `filtrer(s)` *renvoie la suite des sommets de l'enveloppe convexe de $\mathcal{P}$, si* `s` *désigne la suite des points constituant le contour direct $\mathcal{C}$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

5 Opérations sur un arbre enveloppant inférieur

5.1 Recherche dans un arbre enveloppant inférieur

Une première opération consiste à déterminer si un arbre enveloppant inférieur contient un point particulier.

Question II.20 *Écrire en PASCAL une fonction* `contenir(p:POINT;a:ARBRE):ARBRE;` *telle que l'appel* `contenir(p,a)` *renvoie la valeur vraie si l'arbre enveloppant inférieur* `a` *contient le point* `p` *et la valeur faux sinon. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.21 *Donner des exemples de valeurs des paramètres* `p` *et* `a` *de la fonction* `contenir` *qui correspondent au meilleur et pire cas en nombre d'appels récursifs effectués.*

Donner une estimation de la complexité dans le meilleur et pire cas de la fonction `contenir` *en fonction du nombre de points dans* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

5.2 Ajout d'un point dans un arbre enveloppant

5.2.1 Scission d'un arbre enveloppant inférieur

Une deuxième opération consiste à scinder un arbre enveloppant inférieur en deux selon une abscisse tout en préservant cette structure. Elle produit deux arbres enveloppants inférieurs : celui dont les points ont des abscisses inférieures, celui dont les points ont des abscisses supérieures.

Un couple d'arbre est représentée par le type de base `PAIRE`.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION PA_Creer(a1,a2:ARBRE):PAIRE;` renvoie un couple dont les éléments sont `a1` et `a2`,
- `FUNCTION P_Un(p:PAIRE):ARBRE;` renvoie le premier élément de `p`,
- `FUNCTION C_Deux(p:ARBRE):ARBRE;` renvoie le second élément de `p`.

Question II.22 *Écrire en PASCAL une fonction* `scinder(x:INTEGER;a:ARBRE):PAIRE;` *telle que l'appel* `scinder(x,a)` *renvoie une paire d'arbres enveloppants inférieurs dont la première composante est la partie de l'arbre enveloppant inférieur* `a` *contenant les points dont les abscisses sont supérieures à* `x` *et la seconde composante est la partie de l'arbre enveloppant inférieur* `a` *contenant les points dont les abscisses sont supérieures à* `x`*. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

5.2.2 Insertion d'un point

Une troisième opération consiste à insérer un élément dans un arbre enveloppant inférieur en préservant cette structure.

Question II.23 *Écrire en PASCAL une fonction* `ajouter(p:POINT;a:ARBRE):ARBRE;` *telle que l'appel* `ajouter(p,a)` *renvoie un arbre enveloppant inférieur contenant les mêmes points que l'arbre enveloppant inférieur* `a` *ainsi que le point* `p`*. Nous supposons que* `a` *ne contient pas déjà* `p` *et qu'aucun point de* `a` *n'a la même abscisse ou la même ordonnée que* `p`*. L'algorithme utilisé ne devra parcourir au plus qu'une fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.24 *Donner une estimation de la complexité dans le meilleur cas de la fonction* `ajouter` *en fonction du nombre de points contenus dans* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Fin de l'énoncé