Examen Algorithmique avancée
UPMC — Master d'Informatique —
17 décembre 2008 – durée 2h

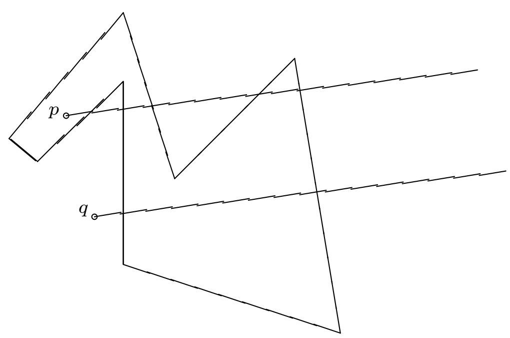
Les seuls documents autorisés sont les polys de cours et de TDs, ainsi que les documents manuscrits.

1 Intérieur d'un polygone simple [8 points]

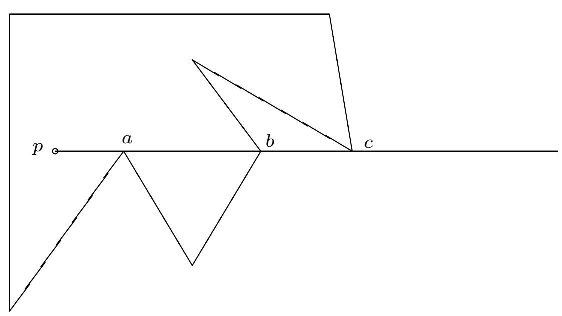
Soit $\mathbf{P}$ un polygone simple dont on connaît un contour $(s_0, \ldots, s_{n-1})$. Le but de cet exercice est de calculer si un point p du plan, est intérieur (au sens strict) à $\mathbf{P}$.

Principe On teste d'abord si le point p est sur le contour du polygone $\mathbf{P}$. S'il n'est pas sur le contour, on choisit une direction $\overrightarrow{u}$, puis on trace une demi-droite Δ d'origine p (p exclu) et de direction $\overrightarrow{u}$, et on compte le nombre k de fois où la demi-droite Δ coupe le contour du polygone $\mathbf{P}$: le point p est intérieur à $\mathbf{P}$ si et seulement si k est impair. Pour simplifier le problème, on suppose dans un premier temps que le polygone $\mathbf{P}$ n'a pas de côté parallèle à Δ.

Par exemple, dans la figure ci-dessous, p est intérieur à $\mathbf{P}$ ($k = 5$) et q n'est pas intérieur à $\mathbf{P}$ ($k = 2$).



Il faut faire attention à dénombrer correctement les intersections, en tenant compte des cas où Δ passe par l'un des sommets de $\mathbf{P}$, comme illustré ci-dessous.



Combien faut-il compter d'intersections en a ? en b ? en c ?

On suppose que l'on dispose des fonctions :
- `succ(s)` qui renvoie le successeur de s dans le contour $(s_0, \ldots, s_{n-1})$ (le successeur de s_{n-1} est s_0)
- `appartientSegment(p,a,b)` qui renvoie vrai ssi le point p appartient au segment $[a, b]$.

La direction $\overrightarrow{u}$ étant fixée, on suppose que l'on dispose des fonctions :
- `intersecte(p,u,a,b)` qui renvoie vrai ssi la demi-droite Δ d'origine p (p exclu) et de direction $\overrightarrow{u}$ coupe le segment $]a, b[$
- `appartientDemiDroite(p,u,a)` qui renvoie vrai ssi le point a appartient à la demi-droite Δ d'origine p (p exclu) et de direction $\overrightarrow{u}$
- `dePartEtDautre(p,u,a,b)` qui renvoie vrai ssi les points a et b sont situés de part et d'autre (au sens strict) de la droite passant par p et de direction $\overrightarrow{u}$.

1. Écrire une fonction `estSurContour` qui, étant donnés un point p et un polygone simple $\mathbf{P}$ dont on connaît un contour $(s_0, \ldots, s_{n-1})$, renvoie vrai si et seulement si p est sur le contour de $\mathbf{P}$.
2. (a) Écrire un algorithme qui détermine si un point p du plan est intérieur (au sens strict) à un polygone simple $\mathbf{P}$ dont on connaît un contour $(s_0, \ldots, s_{n-1})$.
   (b) Quelle est la complexité de cet algorithme, sachant que la complexité de chacune des fonctions `succ`, `appartientSegment`, `intersecte`, `appartientDemiDroite` et `dePartEtDautre` est en $O(1)$?

3. On suppose que le plan est muni d'un repère orthonormé direct $(O, \overrightarrow{i}, \overrightarrow{j})$. Écrire une définition de la fonction `dePartEtDautre`.
4. Étudier le problème dans le cas général où le polygone **P** peut avoir des côtés parallèles à Δ (comment doit-on traiter les extrémités des côtés confondus avec Δ ?).

2 Arbres équilibrés [4 points]

On rappelle la définition de la hauteur h d'un arbre binaire T : si T est réduit à 1 nœud alors $h(T) = 0$, et sinon $h(T) = 1+ \max \{h(T_1), h(T_2)\}$, où T_1 et T_2 sont les sous-arbres à la racine de T.

Soit $\mathcal{P}_c$ la propriété suivante définie sur les arbres binaires : un arbre T vérifie la propriété $\mathcal{P}_c$ si et seulement si *il existe une constante c telle que pour tout nœud de T, les hauteurs de ses sous-arbres diffèrent au plus de c.*

1. Caractériser la famille des arbres binaires qui vérifient $\mathcal{P}_0$ et celle des arbres binaires qui vérifient $\mathcal{P}_1$. Donner un exemple d'arbre vérifiant $\mathcal{P}_2$ et un exemple d'arbre ne vérifiant pas $\mathcal{P}_2$.
2. Montrer (ou admettre) que pour tout $c \in \mathbb{N}^*$, il existe $\alpha \in]0, 1]$, tel que $\alpha(1 + \alpha)^c \leq 1$.
3. Montrer, par récurrence sur la hauteur, que pour tout arbre binaire de taille n et de hauteur h, qui vérifie $\mathcal{P}_c$, on a $n \geq (1 + \alpha)^h - 1$, pour un certain α tel que $0 < \alpha \leq 1$.
4. En déduire que tout arbre qui vérifie la propriété $\mathcal{P}_c$ a une hauteur en $O(\log n)$.

3 Plus courts chemins [8 points]

Soit $G = (S, A, w)$ un graphe connexe orienté valué, avec n sommets et m arcs, et soit $r \in S$ un sommet source. Cet exercice travaille sur l'algorithme de Dijkstra, vu en cours et rappelé ci-dessous :
Les variables globales `pere` et `dist` sont des tables de taille n, dont les éléments sot initialisés respectivement à 0 et ∞ ; l'ensemble Q est initialisé à S, et géré en file de priorité.

Procedure DijkstraPCC(r)
  dist[r]=0
  Repeter Tant que non estVide(Q) **Faire**
    u= extractionMIN(Q,dist)
    Pour chaque sommet v successeur de u **Faire**
      Si estDans(v,Q) **et** dist[u]+w(u,v)< dist[v] **Alors**
        pere[v]=u ; dist[v]=dist[u]+w(u,v)
      Fin Si
    Fin Pour
  Fin Repeter
Fin Procedure

1. On suppose que tous les arcs ont valuation 1 : $\forall(x, y) \in A, w(x, y) = 1$.
   (a) Montrer que l'algorithme de Dijkstra calcule alors l'arborescence de parcours en largeur du graphe G à partir de r.
   (b) Discuter la complexité en temps de cet algorithme, selon la représentation (liste, tas, file binômiale, file de Fibonacci) de la file de priorité Q.
   (c) Pour calculer l'arborescence de parcours en largeur par l'algorithme précédent, il suffit en réalité de gérer Q comme une file FIFO. Expliquer pourquoi et écrire l'algorithme de Dijkstra modifié en conséquence.
   Montrer que la complexité du calcul de l'arborescence de parcours en largeur est alors en temps $O(n + m)$.
2. On suppose maintenant que la valuation des arcs est un nombre entier borné : $\exists k \in \mathbb{N}$ tel que $\forall(x, y) \in A$, $w(x, y) \in \{0, 1, ..., k\}$.
   (a) Montrer que pour tout sommet x de G, on a toujours : si $dist[x] \neq \infty$, alors $dist[x] \leq k(n - 1)$.
   (b) On suppose que si une file de priorité ne contient que des clés à valeurs entières et bornées par K, et si les minimums successifs forment une suite croissante, alors il existe une structuration des clés telle que on peut réaliser une suite de p opérations `insertion`, `extractionMIN`, `diminueCle` en temps $O(p + K)$. En déduire que lorsque la valuation des arcs est un nombre entier borné, on peut calculer les plus courts chemins par l'algorithme de Dijkstra en temps $O(kn + m)$.
   (c) Étudier une structuration des données qui réalise les performances décrites à la question précédente pour une file de priorité ne contenant que des clés à valeurs entières et bornées, et telle que la suite des minimums successifs est une suite croissante.