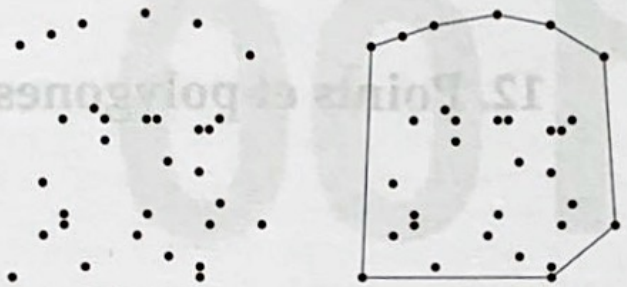
12.1 Enveloppe convexe



Entrée Sortie

Définition Étant donné un ensemble de n points, il faut produire un polygone convexe formé à partir d'un sous-ensemble de ces points tel que les points restants soient contenus dans le polygone. La solution est aussi le polygone de plus petit périmètre contenant tous les points.

Borne inférieure sur la complexité Il n'est pas possible dans le cas général de calculer l'enveloppe convexe en temps $o(n \log n)$ [1]. Pour s'en convaincre, soit une séquence de n nombres $a_1, \ldots, a_n$. Le calcul de l'enveloppe convexe des points $(a_1, a_1^2), \ldots, (a_n, a_n^2)$ les renverra dans un ordre correspondant au tri des nombres $a_1, \ldots, a_n$. Donc si on pouvait faire ce calcul en $o(n \log n)$ opérations, cela donnerait un algorithme de tri de même complexité.

Algorithme en $O(n \log n)$ L'algorithme généralement présenté pour ce problème est le balayage de Graham. Mais nous préférons présenter une variante, *l'algorithme d'Andrew*, qui ne traite pas les points suivant leur angle autour d'un point de repère, mais en fonction de leur coordonnée x. Il a l'avantage de ne pas faire intervenir des calculs d'angles qui pourraient introduire des erreurs de précision.

Nous décrivons seulement comment obtenir la partie du haut de l'enveloppe convexe. Les points sont parcourus dans l'ordre croissant de leur coordonnée x, et dans un tableau *top* on maintient l'enveloppe convexe des points déjà traités. Chaque nouveau point p est ajouté à *top*, puis tant que l'avant-dernier point de *top* rend la séquence non convexe, il est enlevé.

Détails d'implémentation La partie basse de l'enveloppe *bot* est obtenue de la même manière. Le résultat est la concaténation des deux listes, en inversant l'ordre de la liste *top* pour obtenir les points de l'enveloppe dans l'ordre normal, le sens contraire des aiguilles d'une montre. Notez que les premiers et derniers éléments des listes sont identiques, et seraient alors en double dans le résultat de la concaténation. Il est alors important d'enlever ces éléments en trop.

Pour faciliter l'écriture dans le code présenté, le point p n'est ajouté aux listes *top* et *bot* qu'après la suppression des éléments ayant rendu la séquence non convexe.

1. Il s'agit bien du petit o de la notation de Landau.

```
def andrew(S):
    S.sort()
    top = []
    bot = []
    for p in S:
        while len(top) >= 2 and not left_turn(p, top[-1], top[-2]):
            top.pop()
        top.append(p)
        while len(bot) >= 2 and not left_turn(bot[-2], bot[-1], p):
            bot.pop()
        bot.append(p)
    return bot[:-1] + top[:0:-1]
```

12.2 Mesures d'un polygone

Étant donné un polygone simple[1] p sous la forme d'une liste de n points en orientation normale[2], on peut être amené à effectuer plusieurs mesures, voir figure 12.2.

Calcul de la surface en temps linéaire On peut calculer la surface A via la formule

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i),$$

où l'indice $i+1$ est pris modulo n. Le i-ième terme désigne la surface signée du triangle $(0, p_i, p_{i+1})$, dont le signe dépend de l'orientation du triangle. Chaque terme permet de se ramener au calcul de l'aire d'un polygone ayant un point de moins, ainsi l'aire du polygone peut être exprimée comme une séquence d'additions et de soustractions d'aires de triangles.

```
def area(p):
    A = 0
    for i in range(len(p)):
        A += p[i - 1][0] * p[i][1] - p[i][0] * p[i - 1][1]
    return A / 2.
```

Calcul du nombre de points entiers sur le contour Pour simplifier, nous supposons que les points du polygone ont des coordonnées entières. Dans ce cas, la réponse se détermine en sommant pour chaque segment $[a, b]$ de p, le nombre de points entiers entre a et b, excluant a pour ne pas compter de point en double. Si x est la valeur absolue de la différence des abscisses de a et b et y l'équivalent pour les ordonnées, alors ce nombre vaut

$$\begin{cases} y & \text{si } x = 0 \\ x & \text{si } y = 0 \\ \text{le PGCD de } x \text{ et } y & \text{sinon.} \end{cases}$$