Exercices de révision

Jean-Pierre Becirspahic
Lycée Louis-Le-Grand

Exercice 1

Rédiger une fonction `is_prime(n)` qui prend en argument un entier naturel n et retourne un booléen traduisant le primalité de n.

Exercice 1

Rédiger une fonction `is_prime(n)` qui prend en argument un entier naturel n et retourne un booléen traduisant le primalité de n.

```
def is_prime(n):
    if n < 2:
        return False
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d += 1
    return True
```

Exercice 1

Rédiger une fonction `is_prime(n)` qui prend en argument un entier naturel n et retourne un booléen traduisant le primalité de n.

```
def is_prime(n):
    if n < 2:
        return False
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d += 1
    return True
```

Rédiger une fonction `next_prime(n)` qui retourne le plus petit des nombres premiers strictement supérieur à l'entier n.

Exercice 1

Rédiger une fonction `is_prime(n)` qui prend en argument un entier naturel n et retourne un booléen traduisant le primalité de n.

```
def is_prime(n):
    if n < 2:
        return False
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d += 1
    return True
```

Rédiger une fonction `next_prime(n)` qui retourne le plus petit des nombres premiers strictement supérieur à l'entier n.

```
def next_prime(n):
    p = n + 1
    while not is_prime(p):
        p += 1
    return p
```

Exercice 1

Rédiger un script construisant un tableau `prime` contenant les 100000 plus petits nombres premiers.

Exercice 1

Rédiger un script construisant un tableau `prime` contenant les 100000 plus petits nombres premiers.

```
prime = [2]
for _ in range(99999):
    prime.append(next_prime(prime[-1]))
```

Exercice 1

Rédiger un script construisant un tableau `prime` contenant les 100000 plus petits nombres premiers.

```
prime = [2]
for _ in range(99999):
    prime.append(next_prime(prime[-1]))
```

Peut-on faire mieux ?

Exercice 1

Rédiger un script construisant un tableau `prime` contenant les 100000 plus petits nombres premiers.

```
prime = [2]
for _ in range(99999):
    prime.append(next_prime(prime[-1]))
```

Peut-on faire mieux ?

$\longrightarrow$ oui, en remplaçant la fonction `is_prime` par :

```
def is_prime(n):
    if n < 2:
        return False
    for p in prime:
        if p * p > n:
            return True
        if n % p == 0:
            return False
```

Coût : $O\left(\frac{\sqrt{n}}{\log\sqrt{n}}\right) = O\left(\frac{\sqrt{n}}{\log n}\right)$ au lieu de $O(\sqrt{n})$.

Exercice 2

Rédiger une fonction `factorisation(n)` qui décompose un entier naturel n en facteurs premiers.
Par exemple, `factorisation(90)` retournera le tableau :

```
[(2, 1), (3, 2), (5, 1)]
```

Exercice 2

Rédiger une fonction `factorisation(n)` qui décompose un entier naturel n en facteurs premiers.
Par exemple, `factorisation(90)` retournera le tableau :

```
[(2, 1), (3, 2), (5, 1)]
```

```
def factorisation(n):
    facteurs = []
    p = 2
    while n > 1:
        s = 0
        while n % p == 0:
            s += 1
            n //= p
        if s > 0:
            facteurs.append((p, s))
        p += 1
    return facteurs
```

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

```
def nb_diviseurs(n):
    s = 1
    for (_, k) in factorisation(n):
        s *= k + 1
    return s
```

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

```
def nb_diviseurs(n):
    s = 1
    for (_, k) in factorisation(n):
        s *= k + 1
    return s
```

Une fonction `sum_diviseurs(n)` qui retourne la somme des diviseurs de n :

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

```
def nb_diviseurs(n):
    s = 1
    for (_, k) in factorisation(n):
        s *= k + 1
    return s
```

Une fonction `sum_diviseurs(n)` qui retourne la somme des diviseurs de n :

```
def sum_diviseurs(n):
    s = 1
    for (p, k) in factorisation(n):
        s *= (p**(k+1) - 1) // (p - 1)
    return s
```

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

```
def nb_diviseurs(n):
    s = 1
    for (_, k) in factorisation(n):
        s *= k + 1
    return s
```

Une fonction `sum_diviseurs(n)` qui retourne la somme des diviseurs de n :

```
def sum_diviseurs(n):
    s = 1
    for (p, k) in factorisation(n):
        s *= (p**(k+1) - 1) // (p - 1)
    return s
```

Une fonction `phi(n)` qui calcule l'indicatrice d'Euler $\varphi(n)$ de n :

Exercice 2

En déduire une fonction `nb_diviseurs(n)` qui retourne le nombre de diviseurs d'un entier naturel n :

```
def nb_diviseurs(n):
    s = 1
    for (_, k) in factorisation(n):
        s *= k + 1
    return s
```

Une fonction `sum_diviseurs(n)` qui retourne la somme des diviseurs de n :

```
def sum_diviseurs(n):
    s = 1
    for (p, k) in factorisation(n):
        s *= (p**(k+1) - 1) // (p - 1)
    return s
```

Une fonction `phi(n)` qui calcule l'indicatrice d'Euler $\varphi(n)$ de n :

```
def phi(n):
    s = n
    for (p, _) in factorisation(n):
        s = s // p * (p - 1)
    return s
```

Exercice 3

Recherche dichotomique

`t` est un tableau trié de valeurs distinctes et on suppose $t[0] \leqslant x < t[n-1]$.
Rédiger une fonction `dicho(x, t)` qui retourne l'unique entier i tel que :

$$t[i] \leqslant x < t[i+1].$$

Exercice 3

Recherche dichotomique

`t` est un tableau trié de valeurs distinctes et on suppose $t[0] \leqslant x < t[n-1]$.
Rédiger une fonction `dicho(x, t)` qui retourne l'unique entier i tel que :

$$t[i] \leqslant x < t[i+1].$$

```
def dicho(x, t):
    i, j = 0, len(t) - 1
    while i + 1 < j:
        k = (i + j) // 2
        if t[k] <= x:
            i = k
        else:
            j = k
    return i
```

Exercice 3

Recherche dichotomique

`t` est un tableau trié de valeurs distinctes et on suppose $t[0] \leqslant x < t[n-1]$.
Rédiger une fonction `dicho(x, t)` qui retourne l'unique entier i tel que :

$$t[i] \leqslant x < t[i+1].$$

```
def dicho(x, t):
    i, j = 0, len(t) - 1
    while i + 1 < j:
        k = (i + j) // 2
        if t[k] <= x:
            i = k
        else:
            j = k
    return i
```

- Comment prouver la validité de cet algorithme ?

Exercice 3

Recherche dichotomique

`t` est un tableau trié de valeurs distinctes et on suppose $t[0] \leqslant x < t[n-1]$.
Rédiger une fonction `dicho(x, t)` qui retourne l'unique entier i tel que :

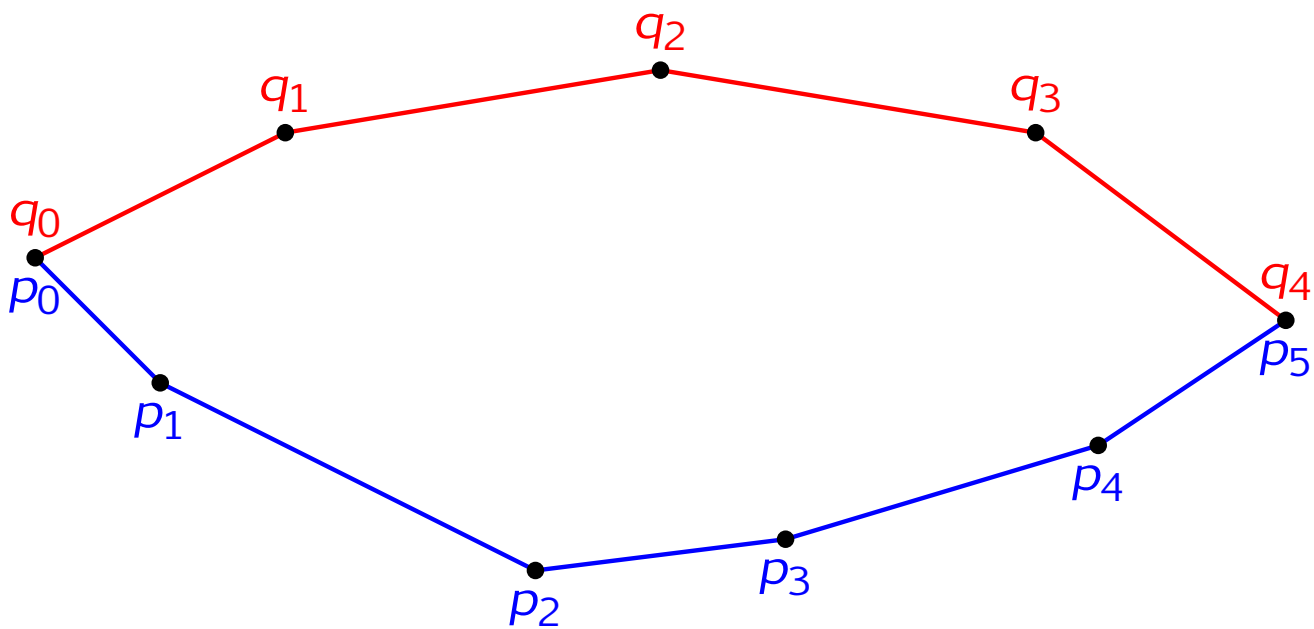
$$t[i] \leqslant x < t[i+1].$$

```
def dicho(x, t):
    i, j = 0, len(t) - 1
    while i + 1 < j:
        k = (i + j) // 2
        if t[k] <= x:
            i = k
        else:
            j = k
    return i
```

- Comment prouver la validité de cet algorithme ?
- Calculer la complexité de cet algorithme.

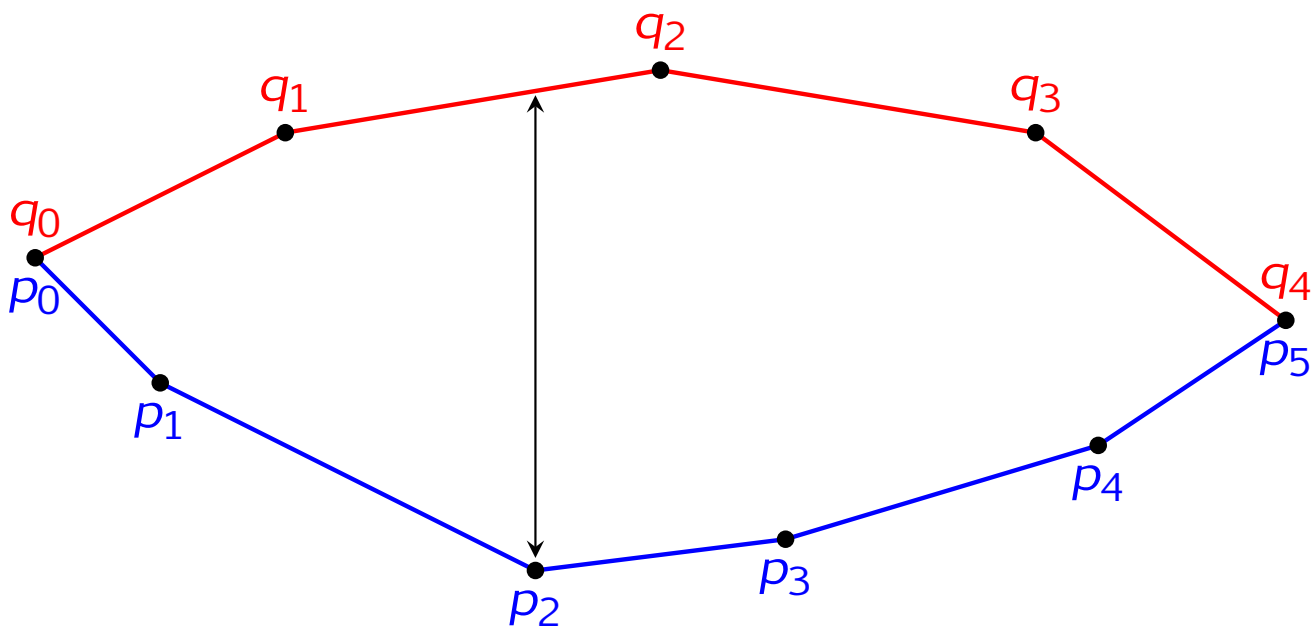
Calcul de l'épaisseur verticale maximale

Nous étudions l'épaisseur maximale d'un convexe bi-dimensionnel. Le polygone est représenté en mémoire par deux listes de sommets p et q triés par abscisses croissantes correspondant respectivement à la bordure inférieure et supérieure de ce convexe.
On suppose que ces deux listes ont même point de départ et d'arrivée.



Calcul de l'épaisseur verticale maximale

Nous étudions l'épaisseur maximale d'un convexe bi-dimensionnel. Le polygone est représenté en mémoire par deux listes de sommets p et q triés par abscisses croissantes correspondant respectivement à la bordure inférieure et supérieure de ce convexe.
On suppose que ces deux listes ont même point de départ et d'arrivée.



Justifier que l'épaisseur maximale est localisée sur un sommet du polygone. On suppose désormais celui-ci situé sur la bordure inférieure.

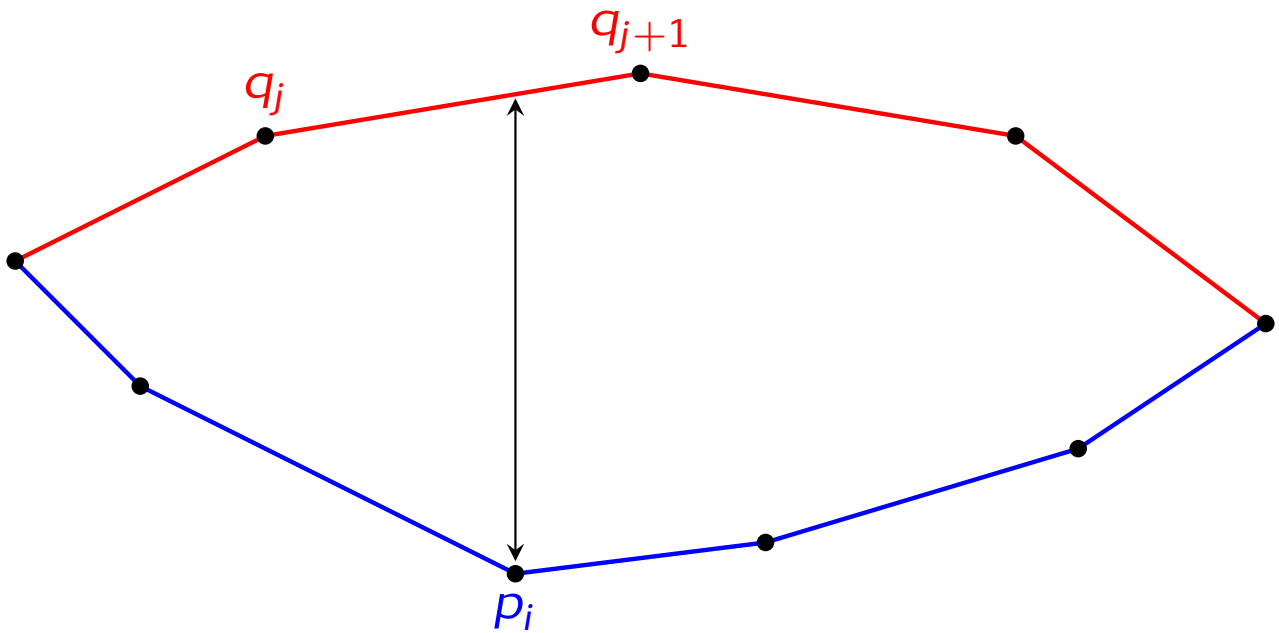
Calcul de l'épaisseur maximale verticale

Calcul de l'épaisseur en un sommet

Les deux bordures représentées par des listes de tuples p et q :

`p[i]` est le couple (p_{ix}, p_{iy}) formé des coordonnées du point p_i.

Définir une fonction `dist(i, j)` qui calcule la distance verticale de p_i à la droite (q_j, q_{j+1}). Quel est son coût ?



Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 1 : double parcours.

```
ep = 0
for i in range(1, len(p)-1):
    for j in range(len(q)-1):
        if q[j][0] <= p[i][0] and p[i][0] < q[j+1][0]:
            ep = max(ep, dist(i, j))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 1 : double parcours.

```
ep = 0
for i in range(1, len(p)-1):
    for j in range(len(q)-1):
        if q[j][0] <= p[i][0] and p[i][0] < q[j+1][0]:
            ep = max(ep, dist(i, j))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

$$O(mn)$$

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 2 : parcours et recherche dichotomique.

```
ep = 0
for i in range(1, len(p)-1):
    j = dicho(p[i][0], q)
    ep = max(ep, dist(i, j))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 2 : parcours et recherche dichotomique.

```
ep = 0
for i in range(1, len(p)-1):
    j = dicho(p[i][0], q)
    ep = max(ep, dist(i, j))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

$$O(m \log n)$$

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 3 : parcours unique en parallèle.

```
ep = 0
j = 0
for i in range(1, len(p)-1):
    while q[j][0] <= p[i][0]:
        j += 1
    ep = max(ep, dist(i, j-1))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

Calcul de l'épaisseur maximale verticale

Nous allons maintenant présenter trois méthodes de calcul de l'épaisseur maximale.

Méthode 3 : parcours unique en parallèle.

```
ep = 0
j = 0
for i in range(1, len(p)-1):
    while q[j][0] <= p[i][0]:
        j += 1
    ep = max(ep, dist(i, j-1))
```

Exprimer son coût en fonction des entiers $m = \text{len}(p)$ et $n = \text{len}(q)$.

$$O(m+n)$$

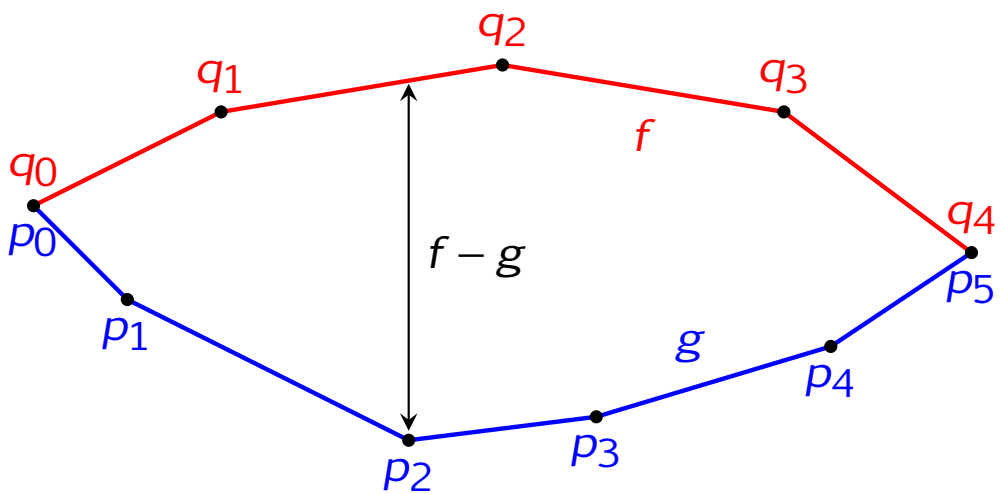
Calcul de l'épaisseur verticale maximale

Peut-on faire mieux ?

Calcul de l'épaisseur verticale maximale

Peut-on faire mieux ?

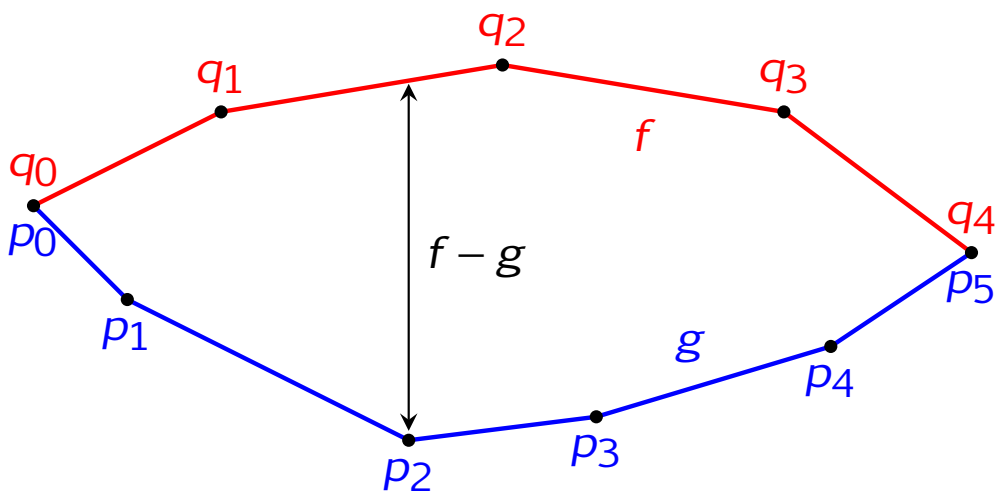
Si f est la fonction dessinant la bordure supérieure et g celle dessinant la bordure inférieure, f est concave et g convexe donc $f - g$ est concave et son maximum est atteint à la position d'épaisseur maximale.



Calcul de l'épaisseur verticale maximale

Peut-on faire mieux ?

Si f est la fonction dessinant la bordure supérieure et g celle dessinant la bordure inférieure, f est concave et g convexe donc $f - g$ est concave et son maximum est atteint à la position d'épaisseur maximale.



Définir une fonction `gradient(i)` qui renvoie une valeur :

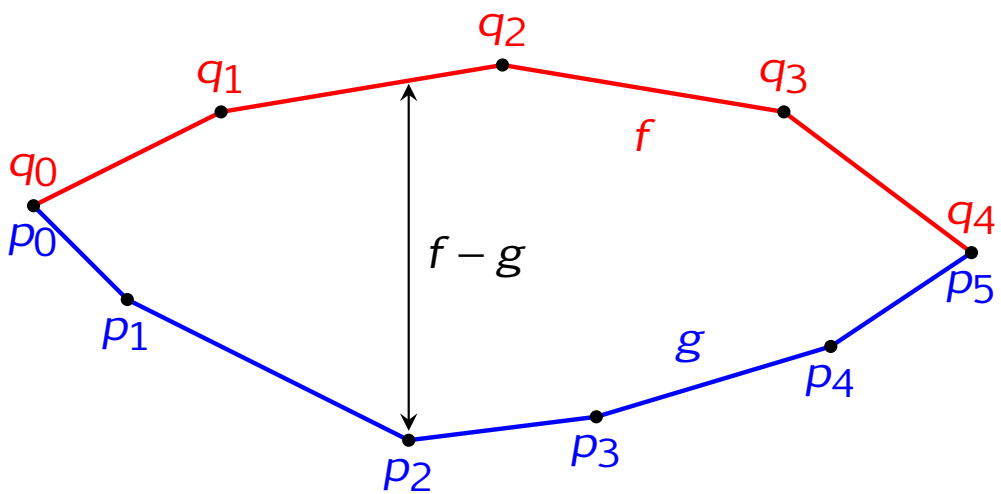
- négative si $p[i]$ est située à gauche du maximum ;
- nulle si $p[i]$ est le maximum ;
- positive si $p[i]$ est située à droite du maximum.

Quel est son coût ?

Calcul de l'épaisseur verticale maximale

Peut-on faire mieux ?

Si f est la fonction dessinant la bordure supérieure et g celle dessinant la bordure inférieure, f est concave et g convexe donc $f - g$ est concave et son maximum est atteint à la position d'épaisseur maximale.



Définir une fonction `gradient(i)` qui renvoie une valeur :

- négative si $p[i]$ est située à gauche du maximum ;
- nulle si $p[i]$ est le maximum ;
- positive si $p[i]$ est située à droite du maximum.

Quel est son coût ? $O(\log n)$

Calcul de l'épaisseur verticale maximale

Méthode 4 : double recherche dichotomique.

Le script suivant calcule l'indice du sommet correspondant à l'épaisseur maximale :

```
i, j = 0, len(p)-1
while i + 1 < j:
    k = (i + j) // 2
    g = gradient(k)
    if g == 0:
        return k
    elif g > 0
        j = k
    else:
        i = k
return i
```

Quel est son coût ?

Calcul de l'épaisseur verticale maximale

Méthode 4 : double recherche dichotomique.

Le script suivant calcule l'indice du sommet correspondant à l'épaisseur maximale :

```
i, j = 0, len(p)-1
while i + 1 < j:
    k = (i + j) // 2
    g = gradient(k)
    if g == 0:
        return k
    elif g > 0
        j = k
    else:
        i = k
return i
```

Quel est son coût ?

$$O(\log m \times \log n)$$