

TC Informatique

PC N° 2
16 Novembre 2000

François Sillion

<http://w3.edu.polytechnique.fr/informatique>

Points en suspens

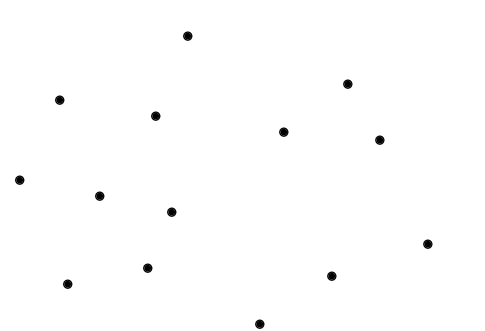
- Délégués ?
- Accès à la documentation, aux transparents
- Questions ?

Un programme ultra simple : Crible d'Eratosthène

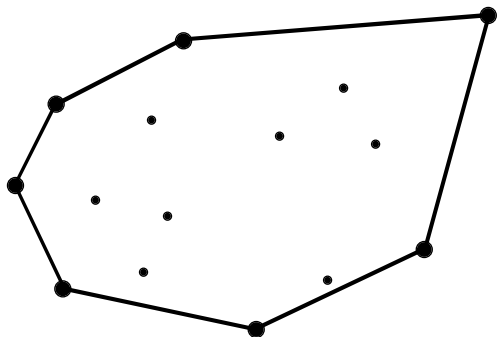
```
public class crible
{
    final int max = 100;
    boolean EstPremier[] = new boolean [max];

    static void rempli ( int p )
    {
        int k = 2;
        while ( k*p < max ) {
            EstPremier[k*p] = false;
            k++;
        }
    }
    static void Initialise()
    {
        for ( int i = 0 ; i < max ; ++i )
            EstPremier[i] = true;
    }
    public static void main ( String[] arg )
    {
        int i;
        Initialise();
        for ( i = 2 ; i < max ; ++i )
            rempli(i);
        for ( i = 1 ; i < max ; ++i )
            if ( EstPremier[i] )
                writeln(i + "est premier");
    }
}
```

Enveloppe convexe



Enveloppe convexe

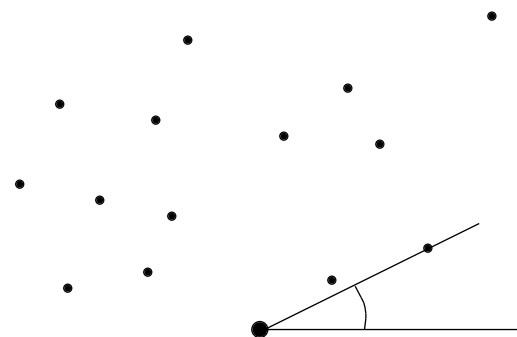


PC 2

François Sillion

5

Algorithme du cordeau

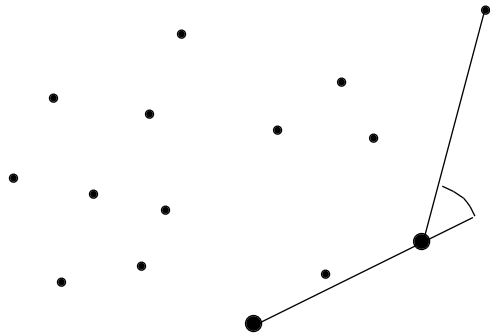


PC 2

François Sillion

6

Algorithme du cordeau



Enveloppe convexe : Algorithme du cordeau

- Remarques

- La fonction Angle retourne une mesure de l'angle entre $P[i]P[j]$ et l'horizontale. Pas forcément l'angle lui-même...
- Échange de deux éléments du tableau :

```
static void Echange(int a, int b)
```

```
{  
    Point p = P[a];  
    P[a] = P[b];  
    P[b] = p;  
}
```

- Nouvel objet

```
class Point  
{  
    double x,y;  
}
```

```
Point Pt = new Point();  
Pt.x = 2.0;  
Point[] P = new Point[5];
```

on verra qu'on peut définir des **constructeurs...**

Enveloppe convexe : Algorithme du cordeau

```
public class cordeau
{
    final static int max = 100;
    static Point[] P = new Point[max+1];

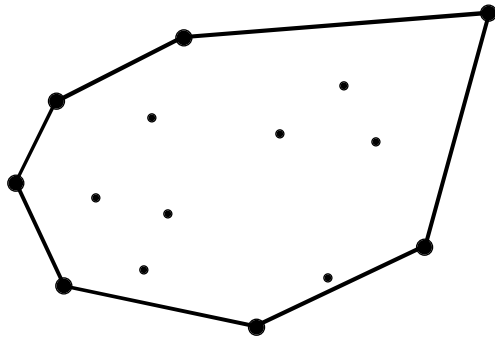
    static int wrap ( Point[] P, int max )
    {
        int min = TrouveOrdonnéeMinimum();
        // place une sentinelle
        P[max] = P[min];
        double angle = 0;
        for ( j=0 ; j < max ; ++j )
        {
            Echange(j,min);
            min = max; angle = 360;
            for ( int k=j+1; k <= max; ++k )
                if ( Angle ( P[j], P[k] ) < angle )
                {
                    min = k; angle = Angle(P[j],P[k]);
                }
            if ( min == max ) return j;
        }
    }
    public static void main ( String[] arg )
    {
        Initialise();
        int Nb = wrap ( P , max );
    }
}
```

Notions de complexité

- Complexité des opérations demandées par un algorithme en fonction de la taille des données d'entrée
- Temps ou espace
- Comportement asymptotique
- Cas le pire, comportement en moyenne
- Notations
 - $O(n)$, $O(n \log(n))$...
- Complexité dépendant de la sortie

Complexité de l'algorithme du cordeau ?

- $O(n^2)$ dans le cas le pire (tous les points sur l'enveloppe convexe)
- $O(n \cdot p)$ si p est le nombre de points sur l'enveloppe
- Peut-on faire mieux ?



Tris simples

- Organisation en modules fonctionnels
- Structures de données
 - Objets Java
 - Constructeurs
 - Classes
- Algorithmes simples
- Complexité

Tri (avec variables globales)

```
class Tri {
    // Tableau à a trier
    static String[] noms;
    // Fonctions de tri
    static void trier() { ... }
    // Fonction d'impression
    static void imprimer() { ... }
    // Fonction principale
    public static void main(String[] args) {
        // Initialisation du tableau de noms
        noms = new String[3];
        noms[0] = "Annie";
        noms[1] = "Julie";
        noms[2] = "Armand";

        trier();
        imprimer();
    }
}
```

Tri (avec variables locales)

```
class Tri {
    static void trier(String[] t)
    { ... }
    static void imprimer(String[] t)
    { ... }

    public static void main(String[] args) {
        String[] noms = new String[] {
            "Annie",
            "Julie",
            "Armand"
        };

        trier(noms);
        imprimer(noms);
    }
}
```

Tri (ligne de commande)

```
% java Tri Annie Julie Armand

class Tri {
    static void trier(String[] t)
    { ... }
    static void imprimer(String[] t)
    { ... }

    public static void main(String[] args) {
        trier(args);
        imprimer(args);
    }
}
```

Utilisation d'un objet spécifique

```
class Eleve {
    String nom;
    float note;
}
class Tri {
    static void trier(Eleve[] eleves) { ... }
    static void imprimer(Eleve[] eleves) { ... }

    public static void main(String[] args) {
        Eleve[] eleves = new Eleve[3];
        eleves[0] = new Eleve();
        eleves[0].nom = "Annie"; eleves[0].note = 13.0;
        eleves[1] = new Eleve();
        eleves[1].nom = "Julie"; eleves[1].note = 7.0;
        eleves[2] = new Eleve();
        eleves[2].nom = "Armand"; eleves[2].note = 18.0;
        trier(eleves);
        imprimer(eleves);
    }
}
```

Utilisation d'un constructeur

```
class Eleve {
    String nom;
    float note;
    Eleve(String s, float x) {
        nom = s;
        note = x;
    }
}
class Tri {
    static void trier(Eleve[] eleves) { ... }
    static void imprimer(Eleve[] eleves) { ... }
    public static void main(String[] args) {
        Eleve[] eleves = new Eleve[] {
            new Eleve("Annie", 13.0),
            new Eleve("Julie", 7.0),
            new Eleve("Armand", 18.0)
        };
        trier(eleves);
        imprimer(eleves);
    }
}
```

De quoi a-t-on besoin pour le tri ?

- **Comparaison**

```
static boolean meilleur(Eleve e1, Eleve e2)
{
    return (e1.note > e2.note);
}
```

- **Echange**

```
static void echanger(Eleve[] eleves, int i1, int i2)
{
    Eleve e = eleves[i1];
    eleves[i1] = eleves[i2];
    eleves[i2] = e;
}
```

Tri par sélection

```
void trier(Eleve[] eleves) {
    final int N = eleves.length;
    for (int i = 0 ; i < N - 1 ; ++i) {
        int m = i;
        for (int j = i + 1 ; j < N ; ++j) {
            if (meilleur(eleves[m], eleves[j]))
                m = j;
        }
        echanger(eleves, i, m);
    }
}
```

- Complexité ?

Tri par insertion

```
static void trier(Eleve[] eleves) {
    final int N = eleves.length;
    for (int i = 1 ; i < N ; ++i) {
        Eleve e = eleves[i];
        int j = i;
        while (j > 0 && meilleur(eleves[j - 1], e) )
        {
            eleves[j] = eleves[j - 1];
            --j;
        }
        eleves[j] = e;
    }
}
```

- Sentinelle (plus mauvais élève)
- Complexité ? Cas le pire, en moyenne ?

Tri à bulles

```
static void trier(Eleve[] eleves) {
    final int N = eleves.length;
    for (int i = N - 1 ; i >= 0 ; --i) {
        for (int j = 1 ; j <= i ; ++j) {
            if (meilleur(eleves[j - 1],
                eleves[j])) {
                    echanger(eleves, j - 1, j);
            }
        }
    }
}
```

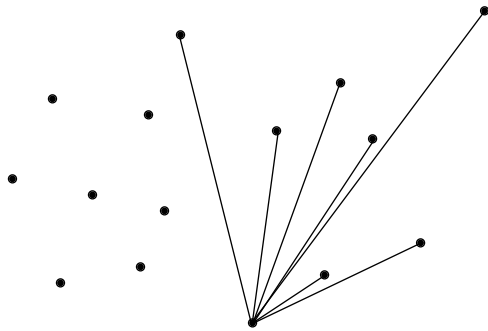
- Complexité ?

Complexité du tri

- Comparaisons, échanges
- Borne inférieure en $N \log N$
- Importance pratique de la boucle la plus interne (*inner loop*)
- Cas pire, complexité moyenne
- Exemples

Enveloppe convexe : Balayage de Graham

- Suppose que l'on sache trier le tableau
- Une fois le tri réalisé, parcours **linéaire** des sommets (non trivial).



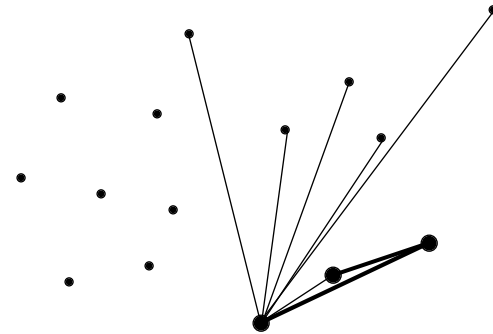
PC 2

François Sillion

23

Enveloppe convexe : Balayage de Graham

- On considère chaque point dans l'ordre comme prochain point de l'enveloppe.



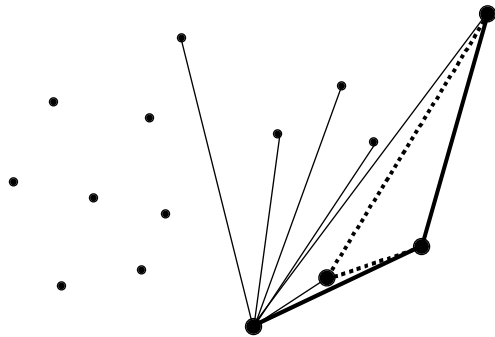
PC 2

François Sillion

24

Enveloppe convexe : Balayage de Graham

- **Back-tracking:** on revient en arrière pour assurer qu'on ne prend que des virages à gauche.



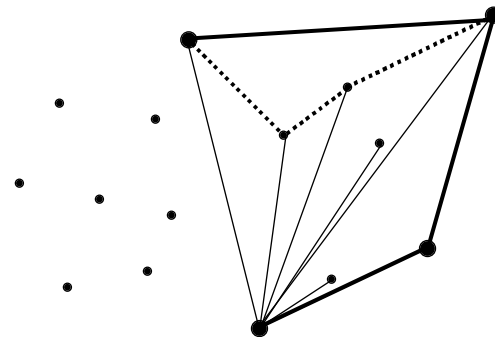
PC 2

François Sillion

25

Enveloppe convexe : Balayage de Graham

- Eventuellement on revient de plusieurs crans en arrière (boucle `while`)
- Complexité: $N \log N$



PC 2

François Sillion

26