# Complexité des problèmes

## IV. Notion de réduction

Cyril Nicaud

M1 Informatique, 2014–2015

# DISTINCTNESS

- L'objectif est de donner des indications sur la difficulté d'un problème en se servant d'un autre problème

**DISTINCTNESS**

Soit $T$ un tableau de taille $n$. Le problème DISTINCTNESS consiste à déterminer s'il n'y a pas d'élément qui apparaît plusieurs fois dans $T$.

```
def distinctness(T):
  T.sort()
  for i in range(len(T)-1)
    if T[i] == T[i+1]:
      return False
  return True
```

Complexité : $\mathcal{O}(n)$ + Tri

# DISTINCTNESS et SORT

**DISTINCTNESS**

Soit $T$ un tableau de taille $n$. Le problème DISTINCTNESS consiste à déterminer s'il n'y a pas d'élément qui apparaît plusieurs fois dans $T$.

- On peut résoudre DISTINCTNESS grâce à SORT, avec un surcoût en $\mathcal{O}(n)$
- On va le noter :

$$\texttt{DISTINCTNESS} \ll_{\mathcal{O}(n)} \texttt{SORT}$$

- En particulier, on a un algorithme en $\mathcal{O}(n \log n)$ pour DISTINCTNESS.

# Définition

**Définition**

Soient $P$ et $Q$ deux problèmes. On dit que $P$ se réduit à $Q$ en temps $f_n$, noté $P \lll_{\mathcal{O}(f_n)} Q$, quand on peut résoudre $P$ en utilisant un nombre borné d'appels à $Q$ avec un surcoût en $\mathcal{O}(f_n)$.

Plus précisément, il faut :

1. transformer l'entrée $x$ de $P$ en une entrée $\varphi(x)$ de $Q$;
2. exécuter un algorithme $\mathcal{A}$ qui résoud $Q$ sur $\varphi(x)$;
3. trouver la solution de $P$ à partir de la solution de $Q$.

- La complexité totale de cette solution dépend de la complexité de $\mathcal{A}$ sur les entrées de taille $\varphi(x)$ et de $f_n$.

# Définition

**Définition**

Soient $P$ et $Q$ deux problèmes. On dit que $P$ se réduit à $Q$ en temps $f_n$, noté $P \lll_{\mathcal{O}(f_n)} Q$, quand on peut résoudre $P$ en utilisant un nombre borné d'appels à $Q$ avec un surcoût en $\mathcal{O}(f_n)$.

- Si $f_n$ est petite par rapport à la complexité de $Q$, cela signifie que $P$ est plus simple que $Q$.
- Si on veut préciser la taille de $\varphi(x)$ par rapport à $x$ on notera

$$P(n) \lll_{\mathcal{O}(f_n)} Q(g_n),$$

quand les $x$ de taille $n$ sont transformés en des $\varphi(x)$ de taille au plus $g_n$

# Exemples

$$\texttt{DISTINCTNESS} \lll_{\mathcal{O}(n)} \texttt{SORT}$$

$$\texttt{2SUM} \lll_{\mathcal{O}(n \log n)} \texttt{2SUMSorted}$$

$$\texttt{MIN} \lll_{\mathcal{O}(1)} \texttt{SORT}$$

$$\texttt{MEDIAN} \lll_{\mathcal{O}(1)} \texttt{SORT}$$

- `MEDIAN` consiste à trouver la valeur médiane du tableau (même nombre d'éléments plus grands que de plus petits)
- `MIN` consiste à trouver le plus petit élément dans un tableau

# Attention

```
def 3SUM(T):
  T.sort()
  for z in T:
    if 2SUM(T,-z):
      return True
  return False
```

On n'a pas réduit `3SUM` à `2SUM` car on appelle `2SUM` un nombre non borné de fois.

# Deux utilisations (important !)

$$P \lll_{\mathcal{O}(f_n)} Q$$

On suppose $\mathcal{O}(f_n)$ petit par rapport aux temps d'exécution de $P$ et $Q$

- L'utilisation classique : trouver une solution efficace à $P$ en utilisant la connaissance d'une solution efficace à $Q$: l'inconnu est $P$
- Nouvelle utilisation : montrer que $Q$ est difficile à résoudre car on sait prouver que $P$ est difficile: l'inconnu est $Q$

- Par exemple, si on sait que $P$ est en $\Omega(n^3)$, cela implique que $Q$ est en $\Omega(n^3)$
- **Rappel** : on parle bien des problèmes, pas des algorithmes

# Réduction et 3SUM

- On a vu que 3SUM peut être résolu en temps $\mathcal{O}(n^2)$
- Supposons qu'il soit aussi en $\Omega(n^2)$, alors si

$$\texttt{3SUM} \lll_{o(n^2)} P,$$

on a que $P$ est en $\Omega(n^2)$

- On rappelle que $u_n \in o(n^2)$ signifie que $u_n$ est négligeable devant $n^2$ :

$$\frac{u_n}{n^2} \xrightarrow[n\to\infty]{} 0.$$

- Donc sous l'hypothèse que 3SUM est quadratique on a une borne inférieure quadratique pour $P$

# 3SUM et 3SUM′

**3SUM′**

Soient $A$, $B$ et $C$ trois ensemble de nombres tels que $|A|+|B|+|C| = n$ points. Le problème 3SUM′ consiste à indiquer s'il existe $a \in A$, $b \in B$ et $c \in C$ tels que $a + b = c$.

- On a facilement que

$$\texttt{3SUM}(n) \ll_{\mathcal{O}(n)} \texttt{3SUM}'(3n)$$

- Il suffit de calculer $-S = \{-x \mid x \in S\}$ et d'appeler $\texttt{3SUM}'(S, S, -S)$
- A-t'on une réduction linéaire dans l'autre sens ?

# 3SUM et 3SUM'

- On veut montrer que $\texttt{3SUM}' \lll_{\mathcal{O}(n)} \texttt{3SUM}$
- A partir de $A$, $B$ et $C$ on doit créer un ensemble $S$ tel que :
  - $S$ contient un nombre linéaire de nombres
  - $x + y + z = 0$ dans $S$ ssi il existe $a \in A$, $b \in B$ et $c \in C$ tels que $a + b = c$.
- On peut supposer que $A$ et $B$ et $C$ ne contiennent que des éléments strictement positifs
- On pose $m = 2\max(A, B, C)$
- On met dans $S$ les $a' = a + m$, les $b' = b$ et les $c' = -c - m$ : $a + b = c \Leftrightarrow a' + b' + c' = 0$
- Il reste à montrer que si 3 éléments de $S$ somment à $0$, ils viennent de trois ensembles différents :
  - $m < a' \leq \frac{3}{2}m$, $0 < b' \leq \frac{1}{2}m$ et $-\frac{3}{2}m \leq c' < -m$
  - On en déduit facilement qu'au plus un élément vient de $A$ (de $C$)
  - Il y a forcément un élément négatif qui vient donc de $C$
  - Si on prend deux éléments de $B$ et un de $C$ la somme est $< 0$

# 3SUM et 3SUM′

**Théorème**

3SUM et 3SUM′ sont aussi difficiles l'un que l'autre :

$$\texttt{3SUM} \lll_{\mathcal{O}(n)} \texttt{3SUM}' \quad \text{et} \quad \texttt{3SUM}' \lll_{\mathcal{O}(n)} \texttt{3SUM}$$

On le notera

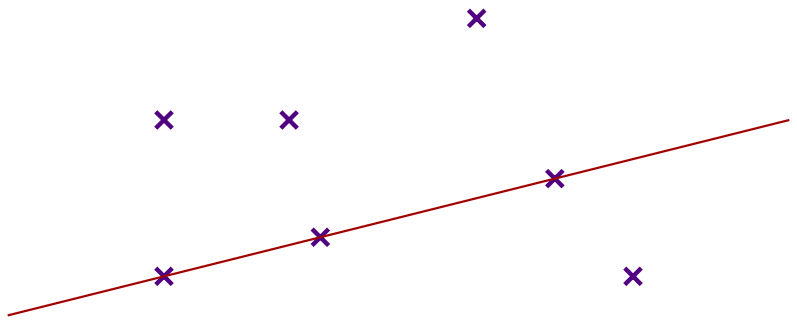
$$\texttt{3SUM} =\!=_{\mathcal{O}(n)} \texttt{3SUM}'$$

# Alignements de points

**3ALIGN**

Soit $P$ un ensemble de $n$ points. Le problème 3ALIGN consiste à indiquer s'il existe trois points (différents) de $P$ qui sont alignés.

- Les points sont donnés par leurs coordonnées, ce sont donc des couples de réels
- Dans notre modèle, on considère qu'on peut faire les opérations $+$, $\times$, $-$ et les comparaisons en temps constant



# Test d'alignement de 3 points

- Il nous faut un moyen de tester si 3 points $M$, $N$ et $P$ sont alignés
- **Solution élémentaire :** calculer l'équation de la droite passant par $M$ et $N$, puis regarder si $P$ vérifie l'équation.
- Si $M = \binom{x_M}{y_M}$ et $N = \binom{x_N}{y_N}$ ne sont pas alignés verticalement, la droite $\Delta$ a pour équation

$$y = a\,x + b, \qquad \text{avec } \begin{cases} y_M = a\,x_M + b \\ y_N = a\,x_N + b \end{cases}$$

  On trouve $a$ et $b$ comme solution d'un système de deux équations à deux inconnues
- Dans le cas où $x_M = x_N$, il suffit de tester si $x_M = x_P$

# Test d'alignement de 3 points

- Il nous faut un moyen de tester si 3 points $M$, $N$ et $P$ sont alignés
- **Produit vectoriel :** si $\vec{u}$ et $\vec{v}$ sont deux vecteurs, alors

$$\vec{u} \wedge \vec{v} = \vec{0} \Leftrightarrow \vec{u} \text{ et } \vec{v} \text{ sont colinéaires}$$

- Cela dit, de façon équivalente, $\overrightarrow{MN} \wedge \overrightarrow{MP} = \vec{0}$ ssi les trois points sont alignés.
- On trouve :

$$\overrightarrow{MN} \wedge \overrightarrow{MP} = \begin{pmatrix} 0 \\ 0 \\ (x_N - x_M)(y_P - y_M) - (x_P - x_M)(y_N - y_M) \end{pmatrix}$$

- Les points sont donc alignés ssi

$$(x_N - x_M)(y_P - y_M) = (x_P - x_M)(y_N - y_M)$$

# 3ALIGN est 3SUM difficile

- Nous allons montrer le résultat suivant :

**Théorème**

3SUM se réduit à 3ALIGN en temps linéaire :

$$\texttt{3SUM} \ll_{\mathcal{O}(n)} \texttt{3ALIGN}$$

- Cela signifie, sous l'hypothèse que 3SUM est en $\Omega(n^2)$, 3ALIGN est aussi en $\Omega(n^2)$

# La réduction de 3SUM à 3ALIGN

- On part d'un tableau $T$ avec $n$ contenant des réels
- On crée un tableau de points $P$, contenant les éléments $(x, x^3)$ pour tous les $x$ dans $T$

**Lemme**

Les trois points distincts $(a, a^3)$, $(b, b^3)$ et $(c, c^3)$ sont alignés si et seulement si $a + b + c = 0$.

- **Preuve :** au tableau

# Le résultat

**Théorème**

3SUM se réduit à 3ALIGN en temps linéaire :

$$\texttt{3SUM} \ll_{\mathcal{O}(n)} \texttt{3ALIGN}$$

```
def 3SUM(T):
  P = []
  for x in T:
    P.append( (x,x**3) )
  return 3ALIGN(P)
```

Si 3SUM est en $\Omega(u_n)$, alors 3ALIGN aussi.

# Borne inférieure pour 3SUM

- Au fait, connaît-on une borne inférieure pour 3SUM ?

**Erickson 1999**

Dans un modèle de calcul assez faible (arbre de décision avec des combinaisons linéaires d'au plus trois variables), 3SUM est en $\Omega(n^2)$.

- On a longtemps conjecturé que 3SUM est en $\Omega(n^2)$ pour des modèles classiques d'ordinateurs, mais ...

# Borne inférieure pour 3SUM

**Grønlund, Pettie 2014**

Il existe un algorithme pour 3SUM de complexité

$$\mathcal{O}\left(\frac{n^2}{u_n}\right), \text{ avec } u_n = \left(\frac{\log n}{\log\log n}\right)^{2/3}.$$

- C'est un résultat de cette année qui prouve que la conjecture sur la borne inférieure (dans un modèle plus géneral) est fausse !

# Conclusion sur 3ALIGN

**3ALIGN**

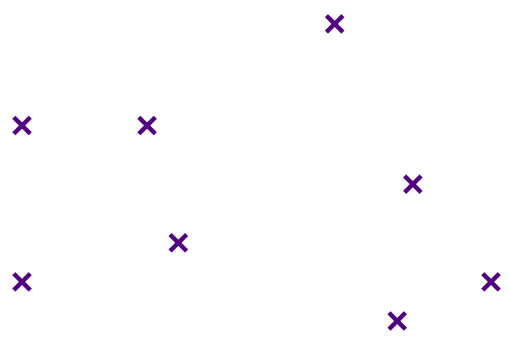
Concernant le problème 3ALIGN :

- On sait le résoudre en temps $\mathcal{O}(n^2)$, c'est un peu difficile à obtenir et hors programme pour ce cours
- On sait qu'on ne peut pas le résoudre en temps meilleur que 3SUM

# Enveloppe convexe

**DISTINCTNESS**

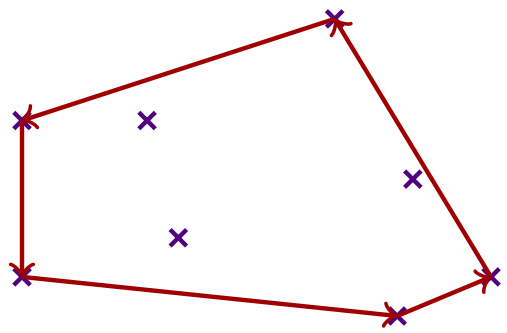
Soit $P$ un ensemble de $n$ points. Le problème CONVEX consiste à déterminer une liste des points obtenus en parcourant l'enveloppe convexe de $P$ dans le sens trigonométrique.



# Enveloppe convexe

**DISTINCTNESS**

Soit $P$ un ensemble de $n$ points. Le problème CONVEX consiste à déterminer une liste des points obtenus en parcourant l'enveloppe convexe de $P$ dans le sens trigonométrique.



# La réduction de SORT à CONVEX

- On part d'un tableau $T$ contenant $n$ réels
- On crée un tableau $P$ de $n$ points de coordonnées $(x, x^2)$, pour les $x \in T$
- On appelle CONVEX sur $P$ pour récupérer la solution $S$
- On a le tableau trié en parcourant les abscisses de $S$ depuis la plus petite abscisse

**Théorème**

On a

$$\texttt{SORT} \lll_{\mathcal{O}(n)} \texttt{CONVEX}$$

- On en déduira que dans un modèle raisonnable, CONVEX est en $\Omega(n \log n)$