TERRA NUMERICA

P=NP ? Une question à 1 million de dollars

Qu'est-ce qu'un problème difficile ? qu'un algorithme efficace ?

Stage maths olympiques 2023

Centre International de Valbonne

Nicolas Nisse

cnrs Inria UNIVERSITÉ CÔTE D'AZUR

22 août 2023

Qui suis-je ?

Chercheur Inria

Qui suis-je ?

Chercheur Inria

« Je suis chercheur en informatique »

« Non ! J'y connais rien en ordinateur »

Qui suis-je ?

Chercheur Inria

« Je suis chercheur en informatique »

« Je suis chercheur en mathématique »

« Que vaut 145693 * 68034 + 12 $\sqrt{34697}$?»

« Non ! J'y connais rien en ordinateur »

« Aucune idée, je suis nul en calcul »

Qui suis-je ?

Chercheur Inria

« Je suis chercheur en informatique »

« Non ! J'y connais rien en ordinateur »

« Je suis chercheur en mathématique »

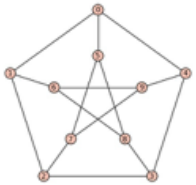
« Que vaut 145693 * 68034 + 12 $\sqrt{34697}$?»

« Aucune idée, je suis nul en calcul »

« Je suis chercheur en informatique théorique » « … »

Qui suis-je ?

Chercheur Inria



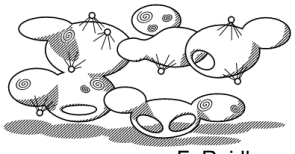
$$\min \sum_{e \in \mathcal{E}} y_e$$

$$\text{s.t.} \sum_{a \in \mathcal{A}^+(u)} f_a^i - \sum_{a \in \mathcal{A}^-(u)} f_a^i = \begin{cases} |V_i| - 1 & \text{if } u = s_i \\ -1 & \text{if } u \neq s_i \end{cases} \quad \forall u \in V_i,\ V_i \in C$$

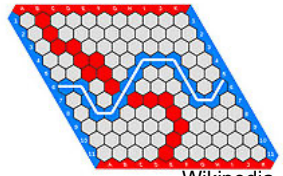
$$f_a^i \leq |V_i| \cdot x_a, \quad \forall V_i \in C, a \in A$$

$$x_{(u,v)} \leq y_{uv}, \quad \forall uv \in \mathcal{E}$$

$$x_{(v,u)} \leq y_{uv}, \quad \forall uv \in \mathcal{E}$$



F. Reidl



Wikipedia

<u>Théorie des graphes, algorithmique</u>, combinatoire, topologie, jeux…

« Je suis chercheur en informatique »

« Non ! J'y connais rien en ordinateur »

« Je suis chercheur en mathématique »

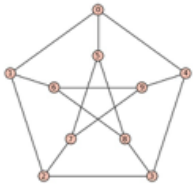
« Que vaut 145693 * 68034 + 12 $\sqrt{34697}$?»

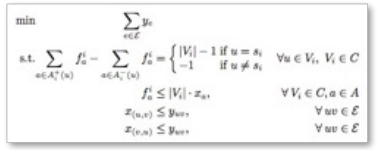
« Aucune idée, je suis nul en calcul »

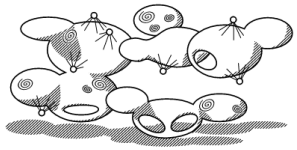
« Je suis chercheur en informatique théorique » « … »

Qui suis-je ?

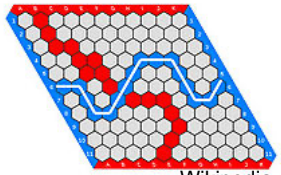
Chercheur Inria







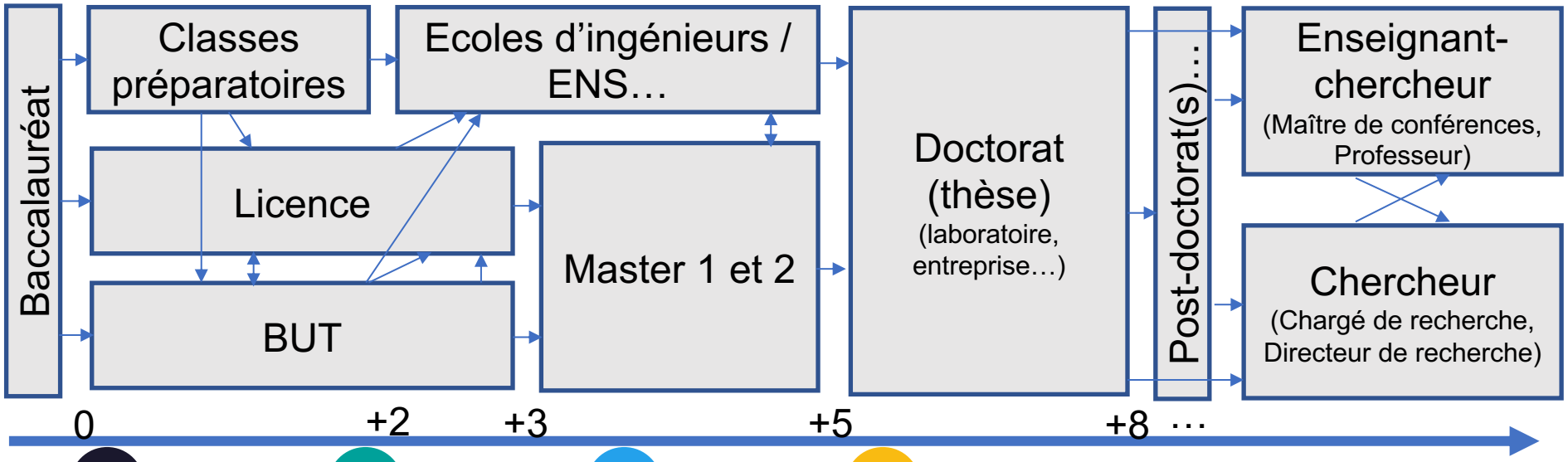
F. Reidl



Wikipedia

<u>Théorie des graphes, algorithmique</u>, combinatoire, topologie, jeux…

Comment devenir (enseignant-)chercheur ? (non exhaustif)



Que fait un chercheur ?

Apprendre

- lire des articles, des livres, …
- écouter des conférences d'autres chercheurs,
- discuter avec des collègues, des industriels, ...
- se poser des questions
- (essayer d') y répondre

Transmettre son savoir

- écrire des articles, des livres, ...
- présenter ses travaux à des conférences,
- enseigner (Université, ...), vulgariser (Fête de la Science, intervention dans les lycées, les collèges, les écoles,...)
- former des chercheurs débutants (étudiants à l'université…).

Chercher, mais quoi ?

Chercher, mais quoi ?

1900 : 2e congrès international des mathématiciens. 23 problèmes de D. Hilbert

Chercher, mais quoi ?

1900 : 2e congrès international des mathématiciens. 23 problèmes de D. Hilbert
2000 : 7 problèmes du millénaire. Clay Mathematics Institute (prix : 1 million de dollars)

- Hypothèse de Riemann (1859)
- Conjecture de Poincaré (1904)
- Problème P = NP ? (1971)
- Conjecture de Hodge (1930)
- Conjecture de Birch et Swinnerton-Dyer (1960)
- Equations de Navier-Stokes (19e siècle)
- Equations de Yang-Mills (années 1950)

Chercher, mais quoi ?

1900 : 2e congrès international des mathématiciens. 23 problèmes de D. Hilbert
2000 : 7 problèmes du millénaire. Clay Mathematics Institute (prix : 1 million de dollars)

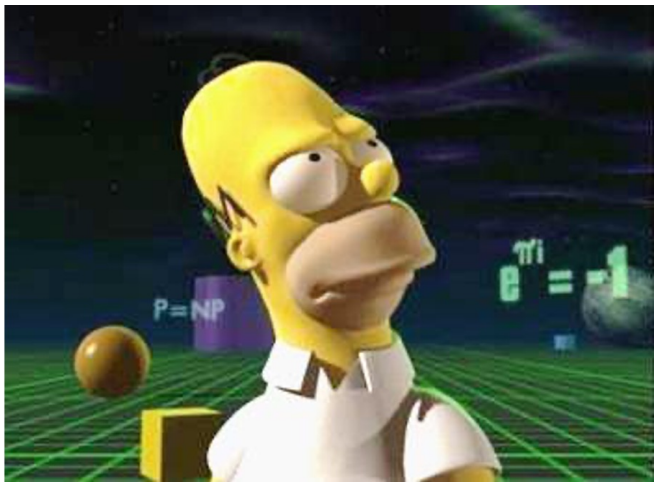
- Hypothèse de Riemann (1859)
- ~~Conjecture de Poincaré (1904)~~ Théorème de Perelman, 2003
- Problème P = NP ? (1971)
- Conjecture de Hodge (1930)
- Conjecture de Birch et Swinnerton-Dyer (1960)
- Equations de Navier-Stokes (19e siècle)
- Equations de Yang-Mills (années 1950)

Chercher, mais quoi ?

1900 : 2e congrès international des mathématiciens. 23 problèmes de D. Hilbert
2000 : 7 problèmes du millénaire. Clay Mathematics Institute (prix : 1 million de dollars)

- Hypothèse de Riemann (1859)
- ~~Conjecture de Poincaré (1904)~~
- Problème P = NP ? (1971)

  « Est-ce que pouvoir vérifier **rapidement** une solution à un problème implique de pouvoir la trouver **rapidement** ? »
- Conjecture de Hodge (1930)
- Conjecture de Birch et Swinnerton-Dyer (1960)
- Equations de Navier-Stokes (19e siècle)
- Equations de Yang-Mills (années 1950)



1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

1er problème : recherche de minimum

Quel est le minimum de cette liste ? (12043, 54677, 32455, 67889, 12102)

1er problème : recherche de minimum

Quel est le minimum de cette liste ? (12043, 54677, 32455, 67889, 12102)

Est-ce un problème « facile » ?

1er problème : recherche de minimum

Quel est le minimum de cette liste ? (12043, 54677, 32455, 67889, 12102)

Est-ce un problème « facile » ?

Et là ? (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

1er problème : recherche de minimum

Quel est le minimum de cette liste ? (12043, 54677, 32455, 67889, 12102)

Est-ce un problème « facile » ?

Et là ? (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

1re remarque : le « temps » de résolution augmente avec la « taille » de l'entrée du problème.

Le « temps » ? Trop subjectif : vous ? Votre ordinateur ? Un super-calculateur ?

Temps = nombre d'opérations/d'instructions

1er problème : recherche de minimum

Quel est le minimum de cette liste ? (12043, 54677, 32455, 67889, 12102)

Est-ce un problème « facile » ?

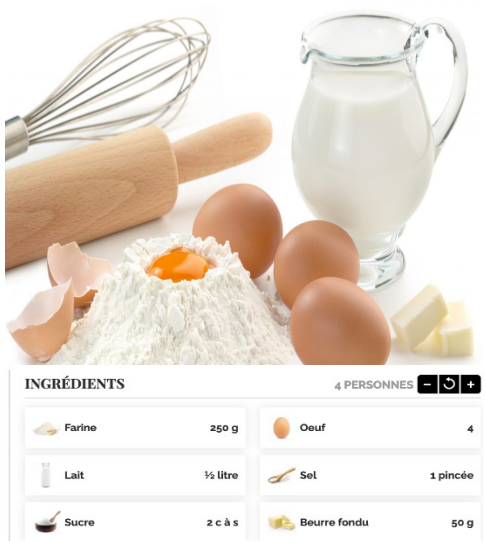
Et là ? (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

1re remarque : le « temps » de résolution augmente avec la « taille » de l'entrée du problème.

Temps = nombre d'opérations/d'instructions

Comment avez-vous fait ? Donnez un **ALGORITHME**

ALGORITHME : suite finie d'instructions non ambiguës qui, étant donné un problème et une entrée, donne/renvoie/calcule une sortie/solution correcte.



Algorithme

ALGORITHME : suite finie d'instructions non ambiguës qui, étant donné un problème et une entrée, donne/renvoie/calcule une sortie/solution correcte.

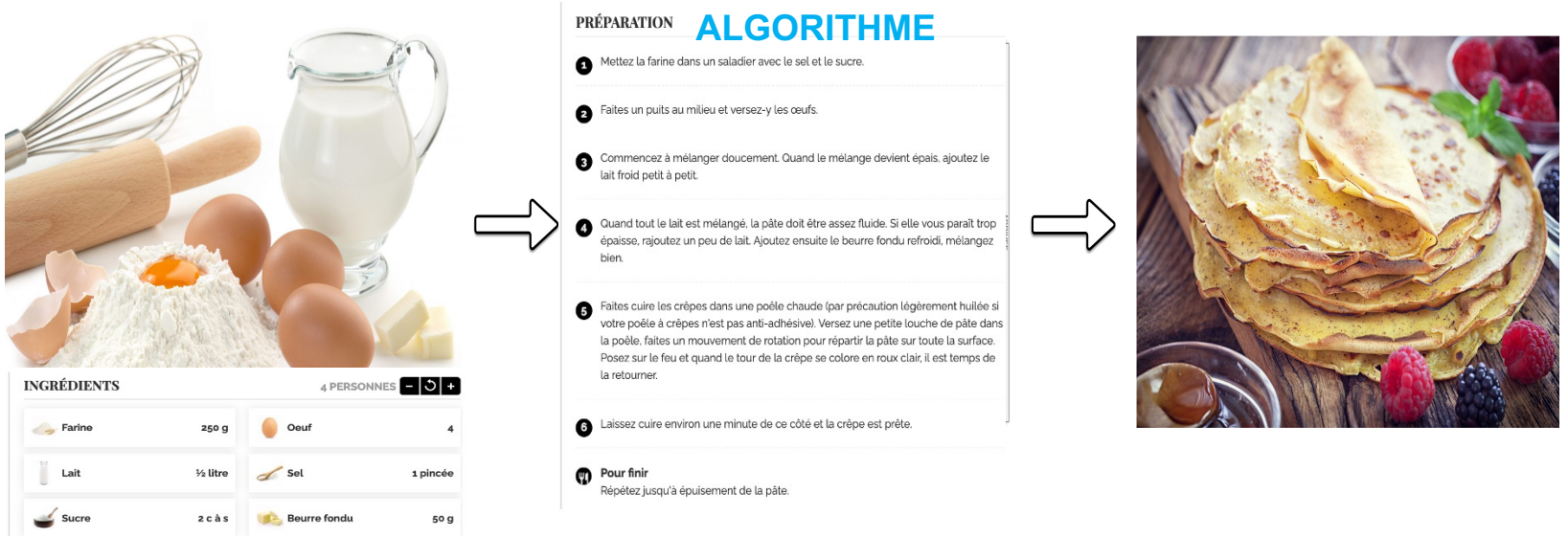
-Mélanger ces éléments du bout des doigts au centre de la fontaine  ??

- Abaisser la pâte à 3mn d'épaisseur → Fleurez généreusement le plan de travail, puis placez votre boule de pâte au centre. Exercez une pression sur la boule, en faisant rouler le pâton ou le rouleau à pâtisserie de haut en bas. Tournez la pâte d'un quart de tour, puis recommencez le même mouvement.
- Foncer les cercles
- Chiqueter les bords
- Passer l'appareil au chinois étamine  ??
- Verser l'appareil sur la garniture

« Instruction non ambiguë »
« Nombre d'instructions »

=> dépend du contexte !!

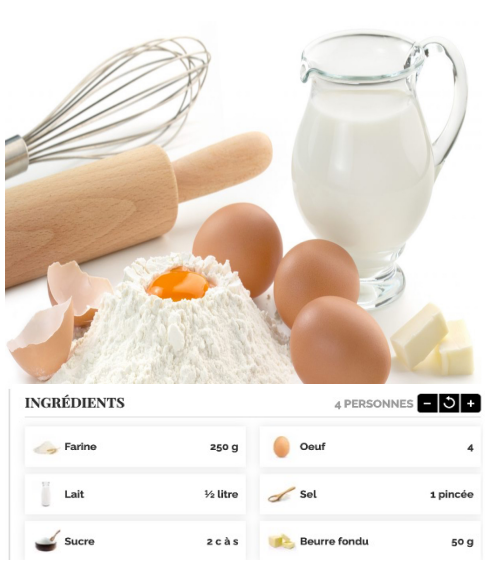
ALGORITHME : suite finie d'instructions non ambiguës qui, étant donné un problème et une entrée, donne/renvoie/calcule une sortie/solution correcte.



Un algorithme doit être **correct** : terminer et donner le résultat attendu quelle que soit l'entrée du problème.

Algorithme

ALGORITHME : suite finie d'instructions non ambiguës qui, étant donné un problème et une entrée, donne/renvoie/calcule une sortie/solution correcte.



Programme : traduction d'un algorithme dans un langage informatique (interprétable par un ordinateur). Ex: Java, C, C++, Python, Caml, SQL, Html, XML…

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

Donnez un **ALGORITHME**

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 02043

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00909

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00345

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00345

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00345

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Min = 00299

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Résultat = 00299

N « opérations » pour déterminer le minimum d'une liste de N éléments !

1er problème : recherche de minimum

Quel est le minimum de cette liste ?

L= (02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : on regarde les éléments les uns après les autres, on retient le plus petit élément qu'on a vu jusque là.

```
Min ← L[1]
Pour i = 2 à taille(L)
        Si L[i]<Min alors Min←L[i]
Renvoyer Min
```

Résultat = 00299

N « opérations » pour déterminer le minimum d'une liste de N éléments !

Il faut au moins lire les N éléments => OPTIMAL

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

2e problème : trier une liste

Trier cette liste dans l'ordre croissant : (12043, 54677, 32455, 67889, 12102)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant : (12043, 54677, 32455, 67889, 12102)

(12043, 12102, 32455, 54677, 67889)

Est-ce un problème « facile » ? Plus/moins facile que le précédent ?

2e problème : trier une liste

Trier cette liste dans l'ordre croissant : (12043, 54677, 32455, 67889, 12102)

(12043, 12102, 32455, 54677, 67889)

Est-ce un problème « facile » ? Plus/moins facile que le précédent ?

Et là ?

(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

« J'ai bon ? »

(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03400, 03450, 03458, 04003, 04003, 04005,
04567, 07889, 10056, 12102, 23000, 03005, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

« J'ai bon ? »

NON !

(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03400, 03450, 03458, 04003, 04003, 04005,
04567, 07889, 10056, 12102, 23000, 03005, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

Remarque 1 : « facile » de vérifier : N opérations pour une liste de taille N

Mais vérifier chacune de vos réponses serait fastidieux

Remarque 2 : un algorithme serait plus convainquant ! **Donnez m'en un !!**

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

Proposez un **ALGORITHME**

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567, 30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003, 36451, 45674, 23647, 03450, 00701, 01001, 03458, 23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

ALGORITHME : On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

```
Res ← ()
Tant que L n'est pas vide
        m = minimum(L)
        ajouter m à la fin de Res
        supprimer m de L
Renvoyer Res
```

Res=(00299, 00321)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 70056, 03005, 00389, 02001)

ALGORITHME : On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 04003,
36451, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 70056, 03005, 00389, 02001)

ALGORITHME : On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345, 00389)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=()

ALGORITHME :

On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001, 02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003, 04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001, 32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=()

ALGORITHME :

On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001, 02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003, 04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001, 32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

Nombre d'opérations (comparaisons) réalisées :

N + (N-1) + (N-2) + (N-3) + … + 3 + 2 +1 = ?

N = taille de la liste

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=()

ALGORITHME :

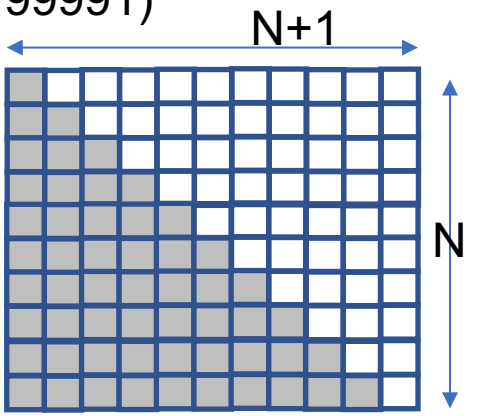
On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001, 02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003, 04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001, 32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

Nombre d'opérations (comparaisons) réalisées :

$$N + (N-1) + (N-2) + (N-3) + \ldots + 3 + 2 + 1 = \frac{N(N+1)}{2} \approx N^2$$



Ici N=32, donc 528 comparaisons

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=()

ALGORITHME :

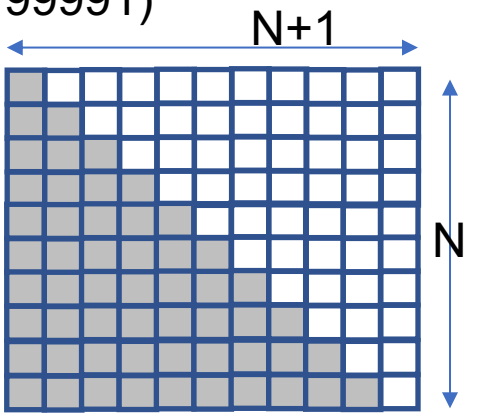
On cherche le minimum, puis le minimum dans ce qui reste, puis le minimum dans ce qui reste...

Res ← ()
Tant que **L** n'est pas vide
    m = minimum(**L**)
    ajouter **m** à la fin de **Res**
    supprimer **m** de **L**
Renvoyer **Res**

Res=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001, 02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003, 04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001, 32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

Nombre d'opérations (comparaisons) réalisées :

$$N + (N-1) + (N-2) + (N-3) + \ldots + 3 + 2 + 1 = \frac{N(N+1)}{2} \approx N^2$$



PEUT ON FAIRE MIEUX ?

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 32455, 07889, 12102, 10056, 00909, 04567,
30001, 34567, 04003, 99991, 43567, 04005, 00345, 04003,
36451, 00299, 45674, 23647, 03450, 00701, 01001, 03458,
23000, 03400, 00408, 00321, 70056, 03005, 00389, 02001)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 07889, 32455, 10056, 12102, 00909, 04567,
30001, 34567, 04003, 99991, 04005, 43567, 00345, 04003,
00299, 36451, 23647, 45674, 00701, 03450, 01001, 03458,
03400, 23000, 00321, 00408, 03005, 70056, 00389, 02001)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 54677, 07889, 32455, 10056, 12102, 00909, 04567,
30001, 34567, 04003, 99991, 04005, 43567, 00345, 04003,
00299, 36451, 23647, 45674, 00701, 03450, 01001, 03458,
03400, 23000, 00321, 00408, 03005, 70056, 00389, 02001)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 07889, 32455, 54677, 00909, 04567, 10056, 12102,
04003, 30001, 34567, 99991, 00345, 04003, 04005, 43567,
00299, 23647, 36451, 45674, 00701, 01001, 03450, 03458,
00321, 00408, 03400, 23000, 00389, 02001, 03005, 70056)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(02043, 07889, 32455, 54677, 00909, 04567, 10056, 12102,
04003, 30001, 34567, 99991, 00345, 04003, 04005, 43567,
00299, 23647, 36451, 45674, 00701, 01001, 03450, 03458,
00321, 00408, 03400, 23000, 00389, 02001, 03005, 70056)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00909, 02043, 04567, 07889, 10056, 12102, 32455, 54677,
00345, 04003, 04003, 30001, 04005, 34567, 43567, 99991,
00299, 00701, 01001, 03450, 03458, 23647, 36451, 45674,
00321, 00389, 00408, 02001, 03005, 03400, 23000, 70056)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00909, 02043, 04567, 07889, 10056, 12102, 32455, 54677,
00345, 04003, 04003, 30001, 04005, 34567, 43567, 99991,
00299, 00701, 01001, 03450, 03458, 23647, 36451, 45674,
00321, 00389, 00408, 02001, 03005, 03400, 23000, 70056)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00345, 00909, 02043, 04003, 04003, 04005, 04567, 07889,
10056, 12102, 30001, 32455, 34567, 43567, 54677, 99991,
00299, 00321, 00389, 00408, 00701, 01001, 02001, 03005,
03400, 03450, 03458, 23000, 23647, 36451, 45674, 70056)

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00345, 00909, 02043, 04003, 04003, 04005, 04567, 07889,
10056, 12102, 30001, 32455, 34567, 43567, 54677, 99991,
00299, 00321, 00389, 00408, 00701, 01001, 02001, 03005,
03400, 03450, 03458, 23000, 23647, 36451, 45674, 70056)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001, 02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003, 04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001, 32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L

Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003,
04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

Nombre d'opérations (comparaisons) réalisées pour une liste de N éléments :

$u_1 = u_2 = 1$ et $u_N = 2 \times u_{\frac{N}{2}} + N$

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003,
04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

Nombre d'opérations (comparaisons) réalisées pour une liste de N éléments :

$u_1 = u_2 = 1$ et $u_N = 2 \times u_{\frac{N}{2}} + N$

Solution : $u_N \approx N \times log_2(N)$

Rappel : $log_2(N)$ ≈ nombre de fois qu'il faut diviser N par 2 pour être ≤ 1

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003,
04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

Nombre d'opérations (comparaisons) réalisées pour une liste de N éléments :

$u_1 = u_2 = 1$ et $u_N = 2 \times u_{\frac{N}{2}} + N$

Solution : $u_N \approx N \times log_2(N)$

Ici N=32, donc 32*5 = 160 comparaisons

Rappel : log_2(N) ≈ nombre de fois qu'il faut diviser N par 2 pour être ≤ 1

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003,
04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

Nombre d'opérations (comparaisons) réalisées pour une liste de N éléments :

$u_1 = u_2 = 1$ et $u_N = 2 \times u_{\frac{N}{2}} + N$

Solution : $u_N \approx N \times log_2(N)$

On peut montrer que N x log_2(N) est optimal

2e problème : trier une liste

Trier cette liste dans l'ordre croissant :

L=(00299, 00321, 00345, 00389, 00408, 00701, 00909, 01001,
02001, 02043, 03005, 03400, 03450, 03458, 04003, 04003,
04005, 04567, 07889, 10056, 12102, 23000, 23647, 30001,
32455, 34567, 36451, 43567, 45674, 54677, 70056, 99991)

ALGORITHME de Tri-fusion:

Si $|L| \leq 2$, trier L
Sinon, diviser L en deux parties

- trier chacune des parties *récursivement*
- *« fusionner »* les deux listes triées.

Nombre d'opérations (comparaisons) réalisées pour une liste de N éléments :

$u_1 = u_2 = 1$ et $u_N = 2 \times u_{\frac{N}{2}} + N$ Solution : $u_N \approx N \times log_2(N)$

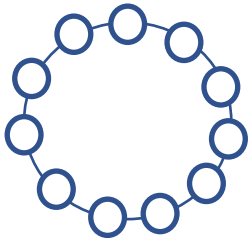
On peut montrer que N x $\log_2$(N) est optimal

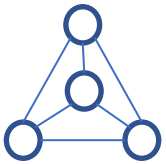
Donc : chercher le minimum (temps N) est « plus facile » que trier $N \leq N \log_2(N)$

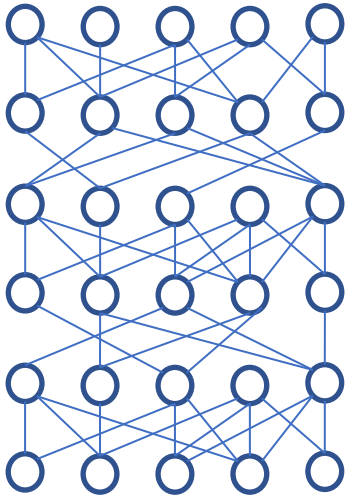
1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

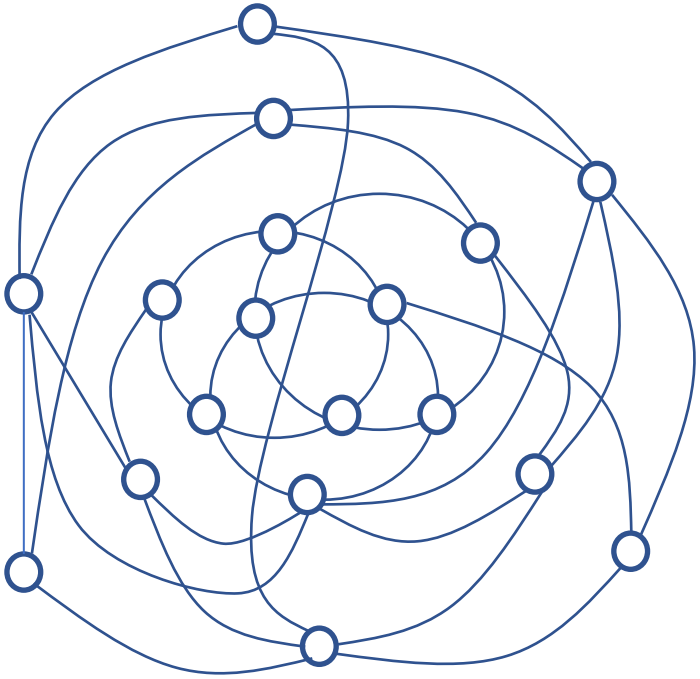
3me problème : 3-coloration

Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?



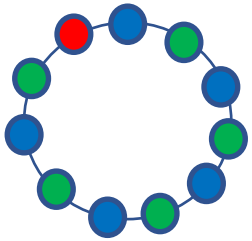




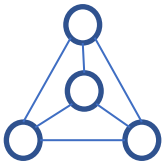


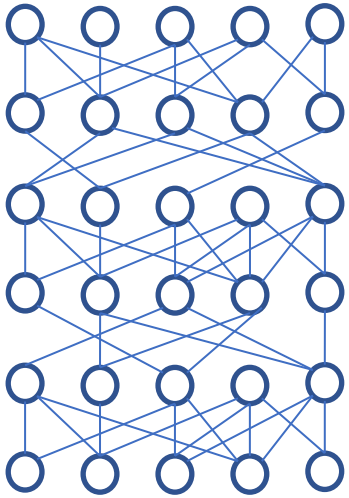
3me problème : 3-coloration

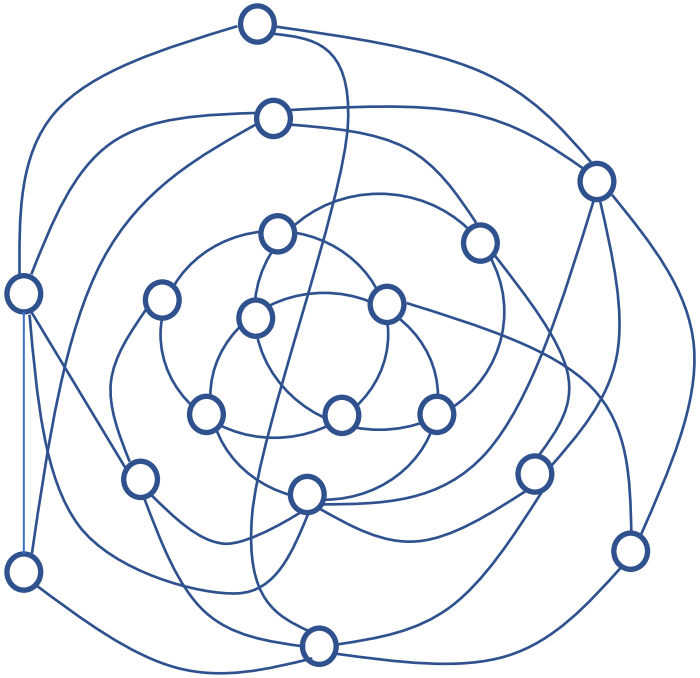
Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?



OK

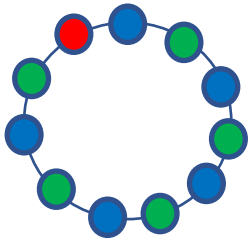




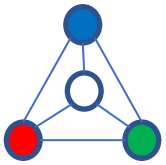


3me problème : 3-coloration

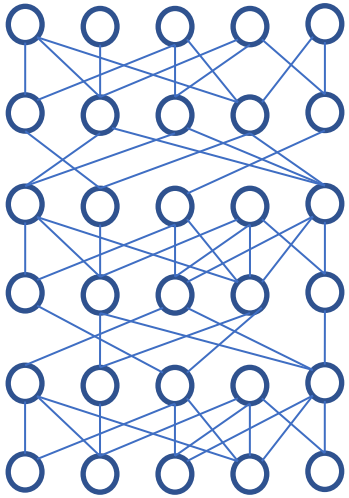
Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

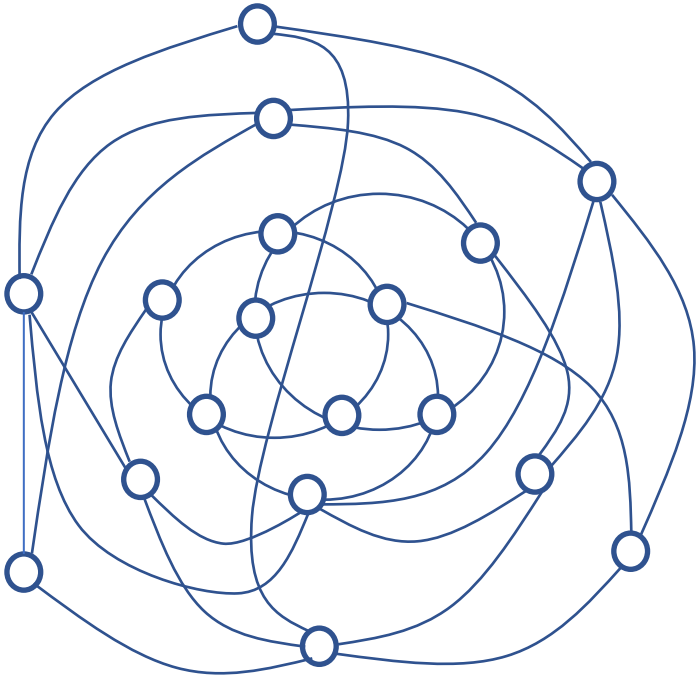


OK



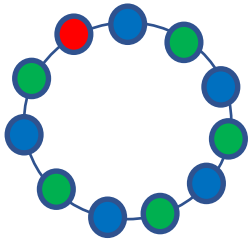
NOK



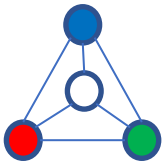


3me problème : 3-coloration

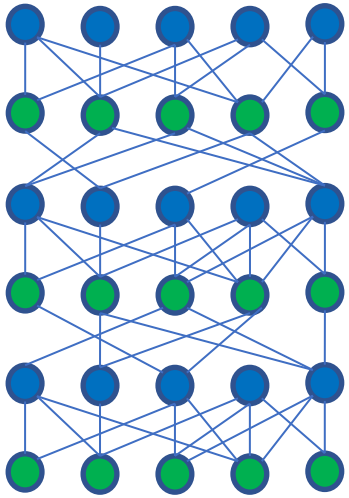
Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?



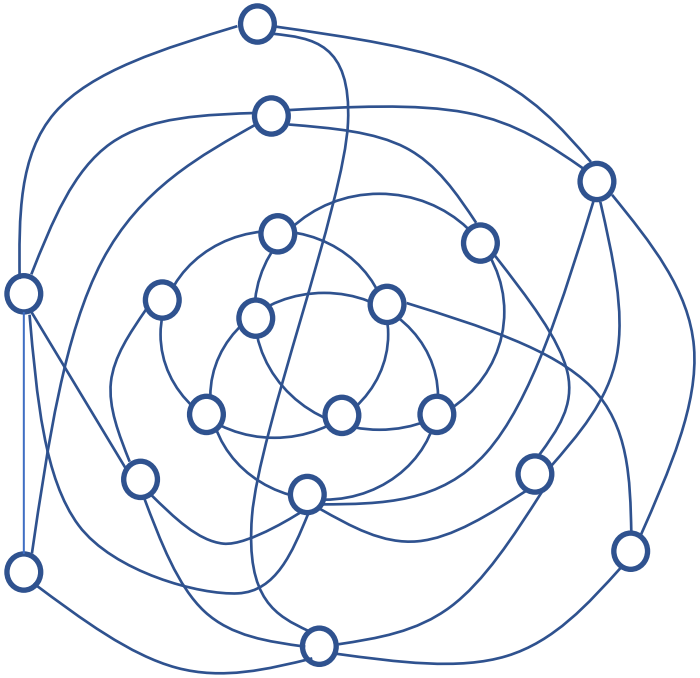
OK



NOK



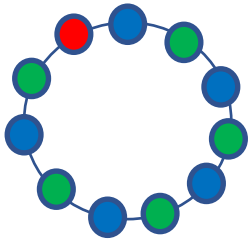
OK



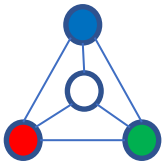
?

3me problème : 3-coloration

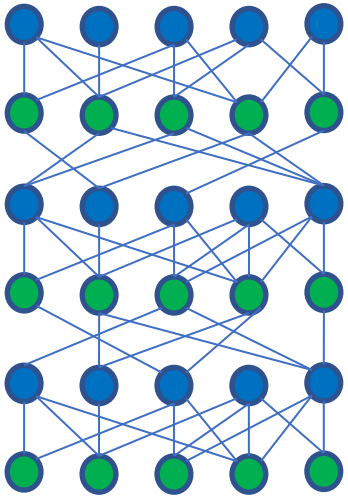
Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?



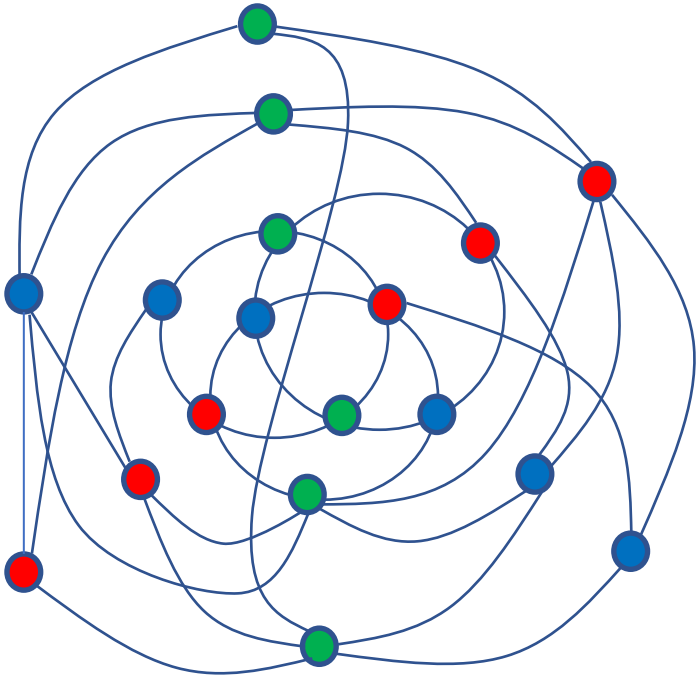
OK



NOK



OK



3me problème : 3-coloration

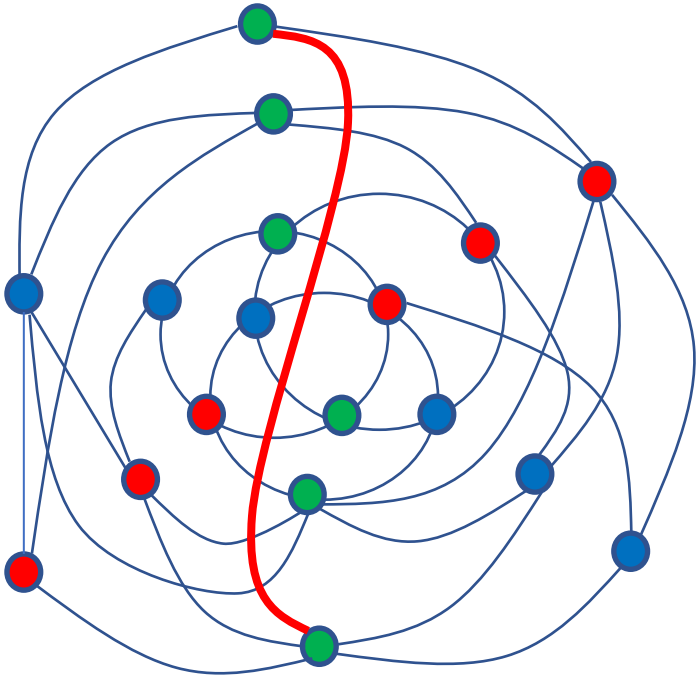
Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

Cette coloration n'est pas propre

(facile à vérifier : vérifier les M arêtes)

Mais… en existe-t-il une ?

Donnez un algorithme qui résout le problème **pour n'importe quel graphe**.



NOK ?

3me problème : 3-coloration

Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

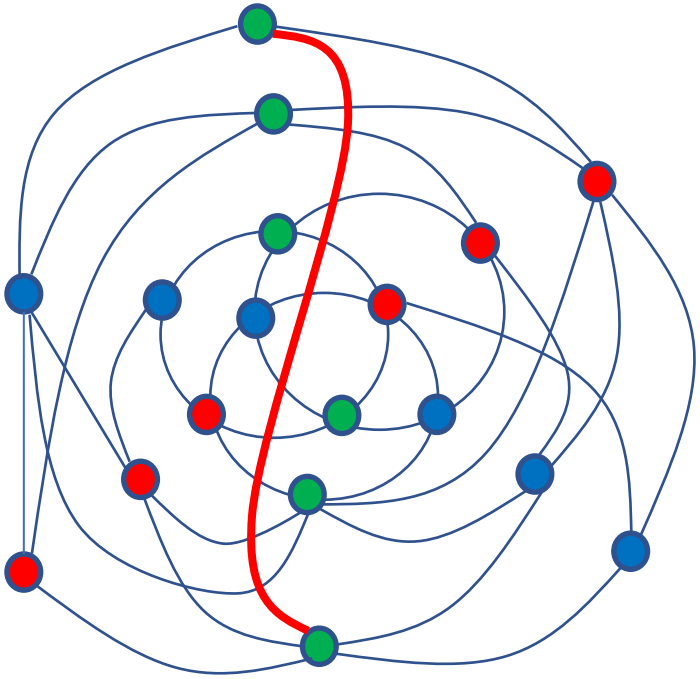
Cette coloration n'est pas propre

(facile à vérifier : vérifier les M arêtes)

Mais… en existe-t-il une ?

Donnez un algorithme qui résout le problème **pour n'importe quel graphe**.

Pour chaque coloration **C**
Si **C** est propre
Alors renvoyer **C**
Renvoyer *« impossible »*



NOK ?

3me problème : 3-coloration

Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

Cette coloration n'est pas propre

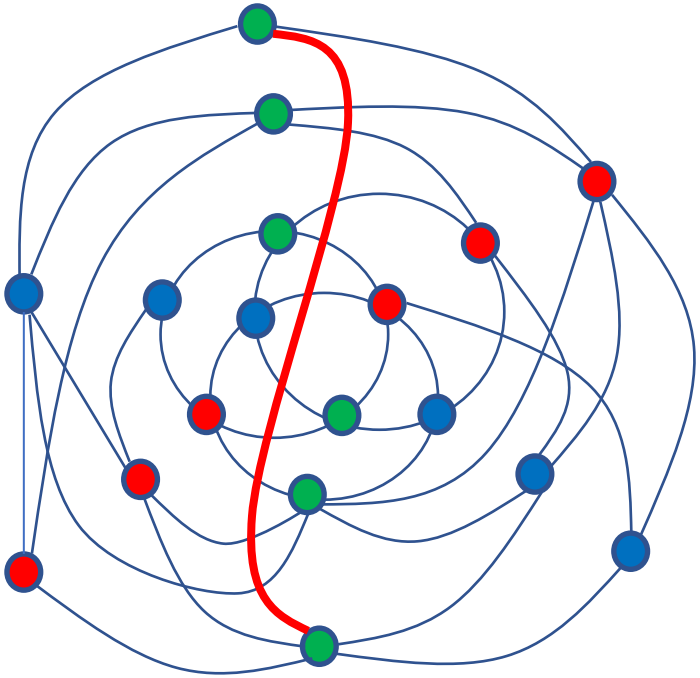
(facile à vérifier : vérifier les M arêtes)

Mais… en existe-t-il une ?

Donnez un algorithme qui résout le problème **pour n'importe quel graphe**.

Pour chaque coloration **C**
Si **C** est propre
Alors renvoyer **C**
Renvoyer *« impossible »*

3^N colorations à vérifier. **(dans le « pire cas »)**



NOK ?

3me problème : 3-coloration

Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

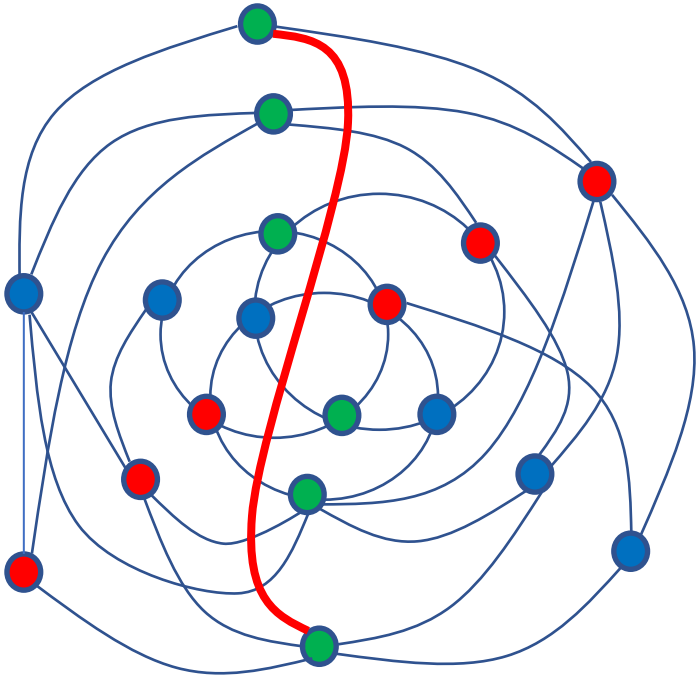
Cette coloration n'est pas propre

(facile à vérifier : vérifier les M arêtes)

Mais… en existe-t-il une ?

Donnez un algorithme qui résout le problème **pour n'importe quel graphe**.

Pour chaque coloration **C**
Si **C** est propre
Alors renvoyer **C**
Renvoyer *« impossible »*



NOK ?

3^N colorations à vérifier. **Peut-on faire mieux ?**

ON NE SAIT PAS FAIRE (« beaucoup ») MIEUX !!!!

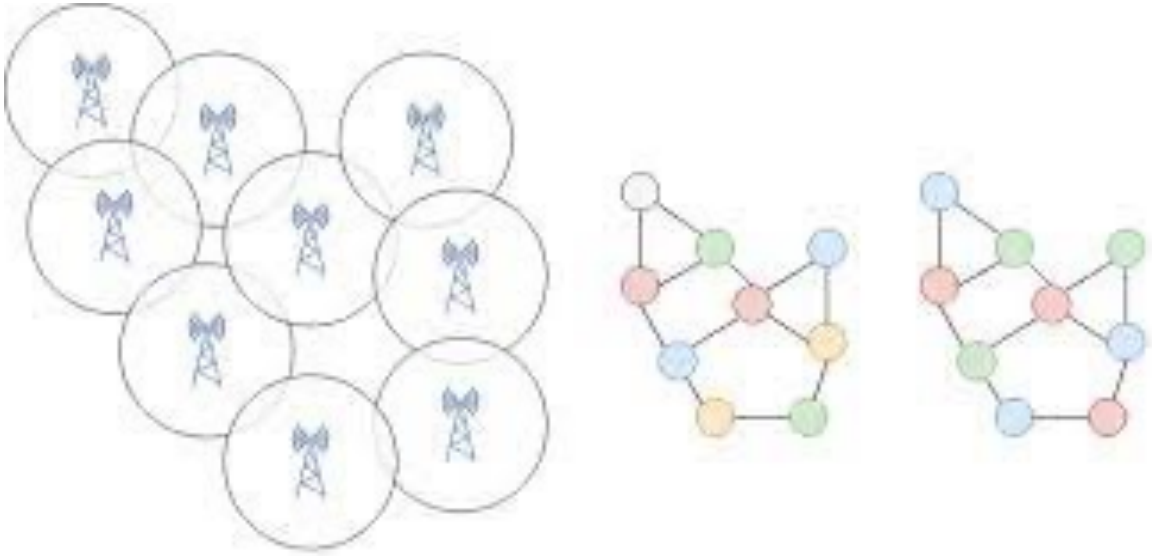
3me problème : 3-coloration

Etant donné un graphe (**N** sommets, **M** arêtes), peut-on colorer ses sommets avec au plus 3 couleurs, tel que 2 sommets adjacents ont toujours des couleurs différentes ?

3^N colorations à vérifier. **Peut-on faire « beaucoup » mieux ?**
ON NE SAIT PAS !!!! Et c'est embêtant…

Application :
Affectation de fréquences à des antennes radio.

2 antennes proches (adjacentes) interfèrent (doivent avoir des fréquences différentes)



Une fréquence, ça coûte cher…

Où va-t-on ?

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Récapitulons un peu

- Un problème est donné avec une entrée de « taille » **N**
- Le temps de résolution augmente avec **N**
- La complexité (temporelle) d'un algorithme est le nombre maximum (en « pire cas ») d' « opérations » réalisées en fonction de **N**

- Recherche du minimum d'une liste de **N** éléments : **N** comparaisons (OPTIMAL)
- Tri d'une liste de **N** éléments :
  Algo 1 : $\mathbf{N^2}$ comparaisons
  Tri-fusion : $\mathbf{N \log_2(N)}$ comparaisons (OPT)
- Décider si un graphe de **N** sommets est 3-colorable : $\mathbf{3^N}$ opérations (OPT ???)

Récapitulons un peu

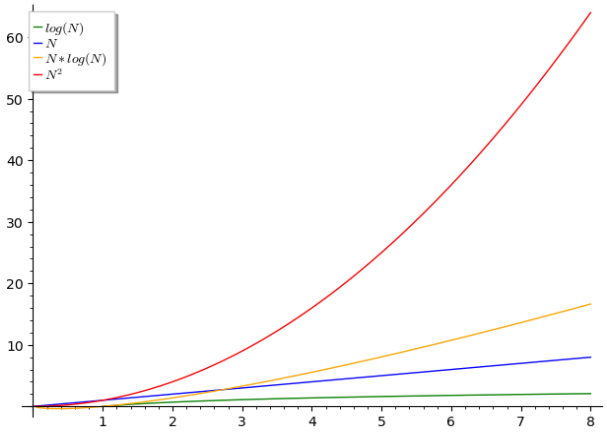
- Un problème est donné avec une entrée de « taille » **N**
- Le temps de résolution augmente avec **N**
- La complexité (temporelle) d'un algorithme est le nombre maximum (en « pire cas ») d' « opérations » réalisées en fonction de **N**

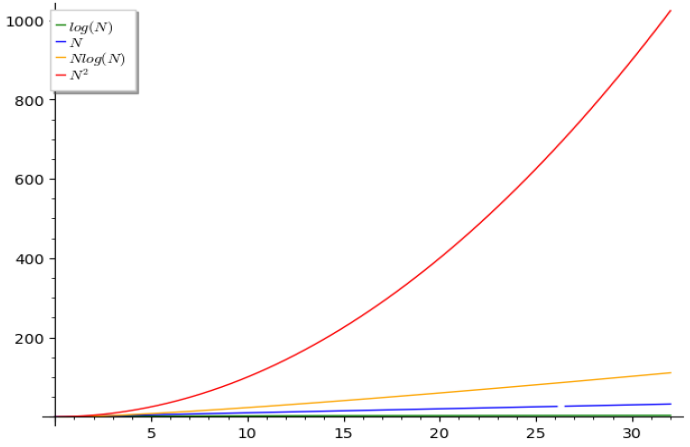
- Recherche du minimum d'une liste de **N** éléments : **N** comparaisons (OPTIMAL)
- Tri d'une liste de **N** éléments :
  Algo 1 : $\mathbf{N^2}$ comparaisons
  Tri-fusion : $\mathbf{N \log_2(N)}$ comparaisons (OPT)
- Décider si un graphe de **N** sommets est 3-colorable : $\mathbf{3^N}$ opérations (OPT ???)

- Pour résoudre un problème « efficacement », on cherche un algorithme de plus faible complexité
- La complexité d'un problème correspond à la plus petite complexité connue d'un algorithme pour résoudre ce problème

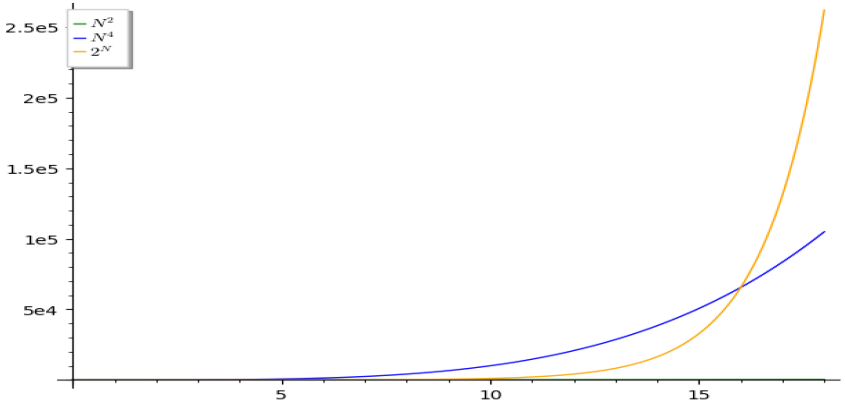
Rappels (ou pas) de maths.

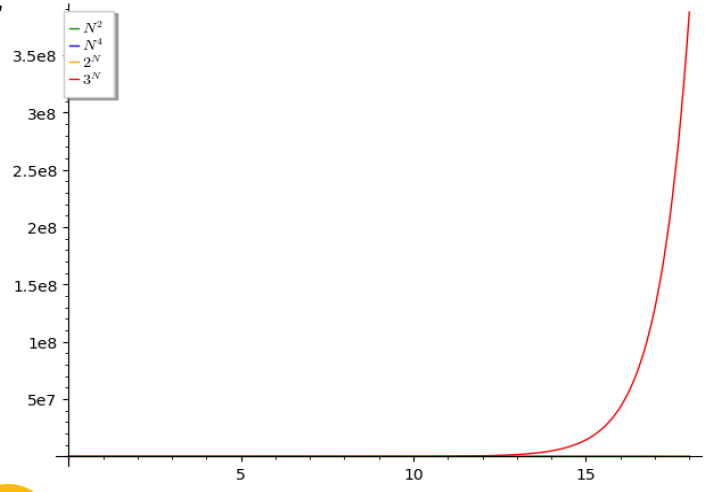
Polynôme : f(N)= $\sum_{k=0}^{d} c_k N^k$ (ex : $f(N) = N^7 + 3N^4 - 7N^2 + 4$)





Exponentielle : $f(N) = c^N = c * c * \ldots * c$





TERRA NUMERICA

Rappels (ou pas) de maths.

log_2(N)	N	N×log_2(N)	N^2	N^4	2^N	3^N	N!	N^N
1	2	2	4	16	4	9	2	4
2	4	8	16	256	16	81	24	256
3	8	24	64	4096	256	6561	40320	$16{,}7.\ 10^6$
3,321…	10	33,21…	100	10000	1024	59049	$\approx 3{,}6.10^6$	10^{10}
4	16	64	256	65536	65536	$\approx 4.\ 10^7$	$\approx 21.\ 10^{12}$	$\approx 1{,}8.\ 10^{19}$
5	32	160	1024	$\approx 10^6$	$\approx 4{,}3.\ 10^9$	$\approx 1{,}85.\ 10^{15}$	$\approx 2{,}63.\ 10^{35}$	$\approx 1{,}46.\ 10^{48}$
7	128	896	16384	$\approx 26.\ 10^7$	$\approx 3{,}4.\ 10^{38}$	$\approx 1{,}18.\ 10^{61}$	$\approx 3{,}86.\ 10^{215}$	$\approx 5{,}3.\ 10^{269}$
10	1024	10240	$\approx 10^6$	$\approx 10^{12}$	« ∞ »	« ∞ »	« ∞ »	« ∞ »

10^6 = un million
10^9 secondes = 31 ans et 251 jours
10^{12} = 1000 milliards
10^{80} atomes dans l'univers observable
10^{120} parties d'échecs possibles

un ordinateur (basic) fait 10^{10} opérations élémentaires par seconde

Rappels (ou pas) de maths.

polynomial | **exponentiel**

$log_2(N)$	N	$N * log_2(N)$	N^2	N^4	2^N	3^N	N!	N^N
1	2	2	4	16	4	9	2	4
2	4	8	16	256	16	81	24	256
3	8	24	64	4096	256	6561	40320	$16{,}7.\ 10^6$
3,321…	10	33,21…	100	10000	1024	59049	$\approx 3{,}6.10^6$	10^{10}
4	16	64	256	65536	65536	$\approx 4.\ 10^7$	$\approx 21.10^{12}$	$\approx 1{,}8.\ 10^{19}$
5	32	160	1024	$\approx 10^6$	$\approx 4{,}3.\ 10^9$	$\approx 1{,}85.\ 10^{15}$	$\approx 2{,}63.\ 10^{35}$	$\approx 1{,}46.\ 10^{48}$
7	128	896	16384	$\approx 26.\ 10^7$	$\approx 3{,}4.\ 10^{38}$	$\approx 1{,}18.\ 10^{61}$	$\approx 3{,}86.\ 10^{215}$	$\approx 5{,}3.\ 10^{269}$
10	1024	10240	$\approx 10^6$	$\approx 10^{12}$	∞	∞	∞	∞

Rappels (ou pas) de maths.

polynomial **exponentiel**

$log_2(N)$	N	$N * log_2(N)$	N^2	N^4	2^N	3^N	N!	N^N
1	2	2	4	16	4	9	2	4
2	4	8	16	256	16	81	24	256
3	8	24	64	4096	256	6561	40320	$16,7.\ 10^6$
3,321…	10	33,21…	100	10000	1024	59049	$\approx 3,6.10^6$	10^{10}
4	16	64	256	65536	65536	$\approx 4.\ 10^7$	$\approx 21.10^{12}$	$\approx 1,8.\ 10^{19}$
5	32	160	1024	$\approx 10^6$	$\approx 4,3.\ 10^9$	$\approx 1,85.\ 10^{15}$	$\approx 2,63.\ 10^{35}$	$\approx 1,46.\ 10^{48}$
7	128	896	16384	$\approx 26.\ 10^7$	$\approx 3,4.\ 10^{38}$	$\approx 1,18.\ 10^{61}$	$\approx 3,86.\ 10^{215}$	$\approx 5,3.\ 10^{269}$
10	1024	10240	$\approx 10^6$	$\approx 10^{12}$	∞	∞	∞	∞

Un problème que l'on sait résoudre en temps polynomial (par un algorithme de complexité polynomiale) est un problème de la **classe P** (Polynomial)

Où va-t-on ?

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Sudoku

N=9

	C1	C2	C3	C4	C5	C6	C7	C8	C9
L1	8	9					2	6	
L2	2							5	7
L3	5		1	2					
L4	1		5		3	7	6		
L5				8		6			
L6				1	2		7		
L7						4	5		
L8	9	5	4						2
L9		1						4	6

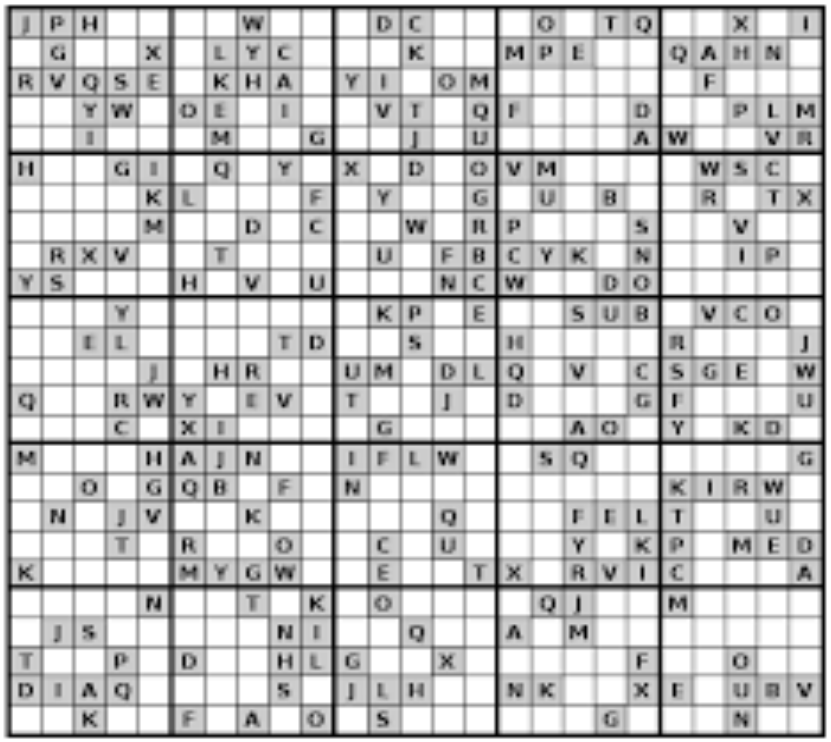
Sudoku

N=9

	C1	C2	C3	C4	C5	C6	C7	C8	C9
L1	8	9					2	6	
L2	2							5	7
L3	5		1	2					
L4	1		5		3	7	6		
L5				8		6			
L6				1	2		7		
L7						4	5		
L8	9	5	4						2
L9		1						4	6

C’est compliqué (pour moi)…
Mais une grille 9*9 est un jeu d’enfant pour un ordinateur

Rappel : on cherche une méthode pour tout **N** N^{N^2}



Sudoku

N=9

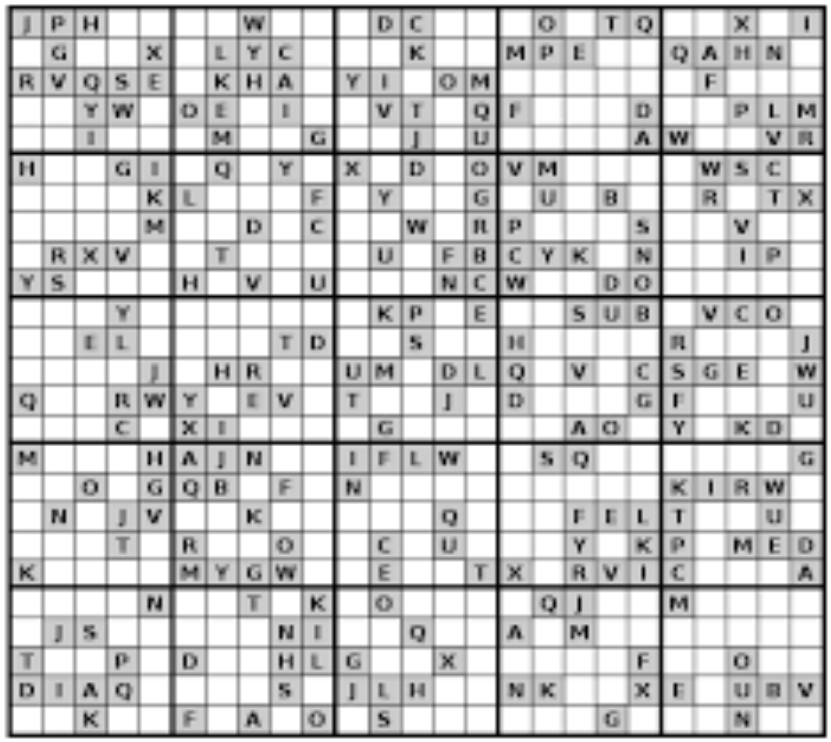
	C1	C2	C3	C4	C5	C6	C7	C8	C9
L1	8	9					2	6	
L2	2							5	7
L3	5		1	2					
L4	1		5		3	7	6		
L5				8		6			
L6				1	2		7		
L7						4	5		
L8	9	5	4						2
L9		1						4	6

	C1	C2	C3	C4	C5	C6	C7	C8	C9
L1	8	9	7	4	5	3	2	6	1
L2	2	4	3	6	1	8	9	5	7
L3	5	6	1	2	7	9	4	3	8
L4	1	2	5	9	3	7	6	8	4
L5	3	7	9	8	4	6	1	2	5
L6	4	8	6	1	2	5	7	9	3
L7	6	3	2	7	8	4	5	1	9
L8	9	5	4	3	6	1	8	7	2
L9	7	1	8	5	9	2	3	4	6

Vérifier une solution est « facile »

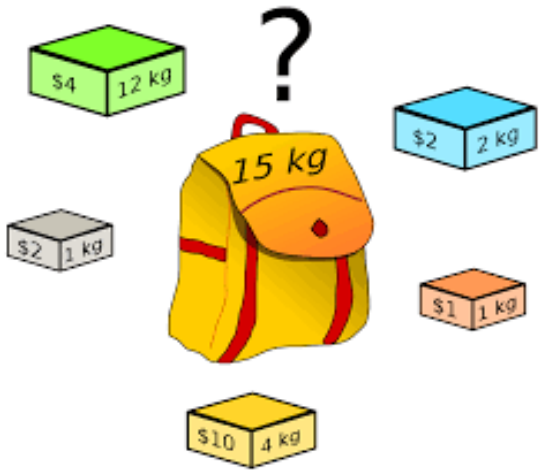
C’est compliqué (pour moi)…
Mais une grille 9*9 est un jeu d’enfant pour un ordinateur

Rappel : on cherche une méthode pour tout **N**



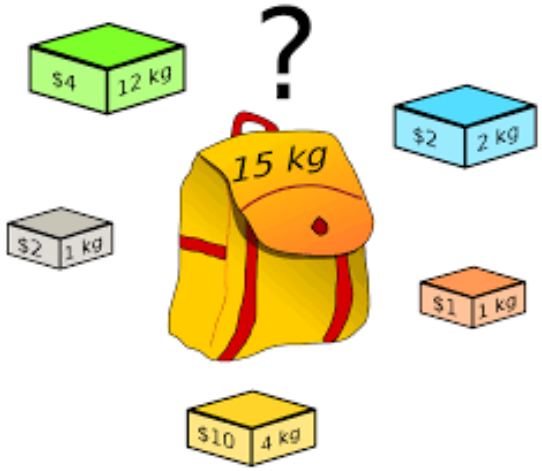
1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Sac-à-dos



Etant donné **N** objets (**poids/prix**), un sac-à-dos de contenance **Q** et un prix objectif **P**, peut-on choisir des objets de poids total $\leq \boldsymbol{Q}$ *(qui rentrent dans le sac)* et de prix total au moins **P** *?*

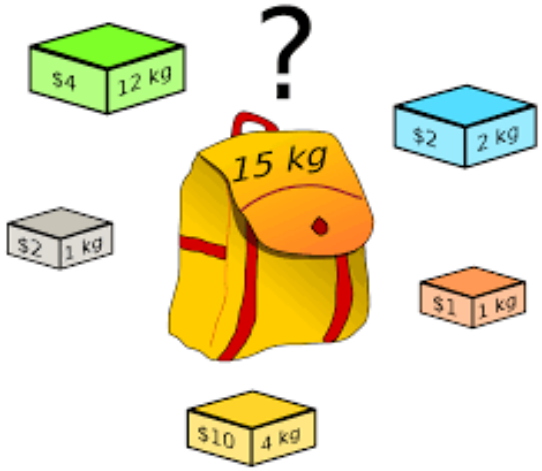
Sac-à-dos



Etant donnés **N** objets (**poids/prix**), un sac-à-dos de contenance **Q** et un prix objectif **P**, peut-on choisir des objets de poids total $\leq Q$ *(qui rentrent dans le sac)* et de prix total au moins **P** *?*

Ici, je choisirais les objets vert et jaune => je gagne 14 dollars !

Sac-à-dos



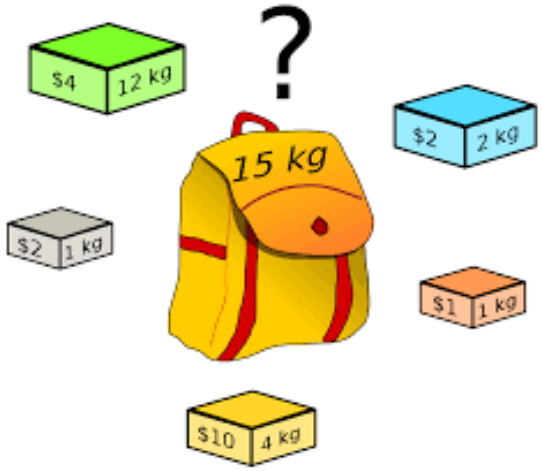
Etant donné **N** objets (**poids/prix**), un sac-à-dos de contenance **Q** et un prix objectif **P**, peut-on choisir des objets de poids total $\leq Q$ *(qui rentrent dans le sac)* et de prix total au moins **P** *?*

Ici, je choisirais les objets vert et jaune => je gagne 14 dollars !

Poids vert + poids jaune = 16kg > **Q**=15 kg

Vérifier une solution est « facile »

Sac-à-dos



Etant donnés **N** objets (**poids/prix**), un sac-à-dos de contenance **Q** et un prix objectif **P**, peut-on choisir des objets de poids total $\leq Q$ *(qui rentrent dans le sac)* et de prix total au moins **P** *?*

Ici, je choisirais les objets vert et jaune => je gagne 14 dollars !

Poids vert + poids jaune = 16kg > **Q**=15 kg

Vérifier une solution est « facile »

Algorithme : tester les 2^N possibilités

mais peut-on faire mieux ????

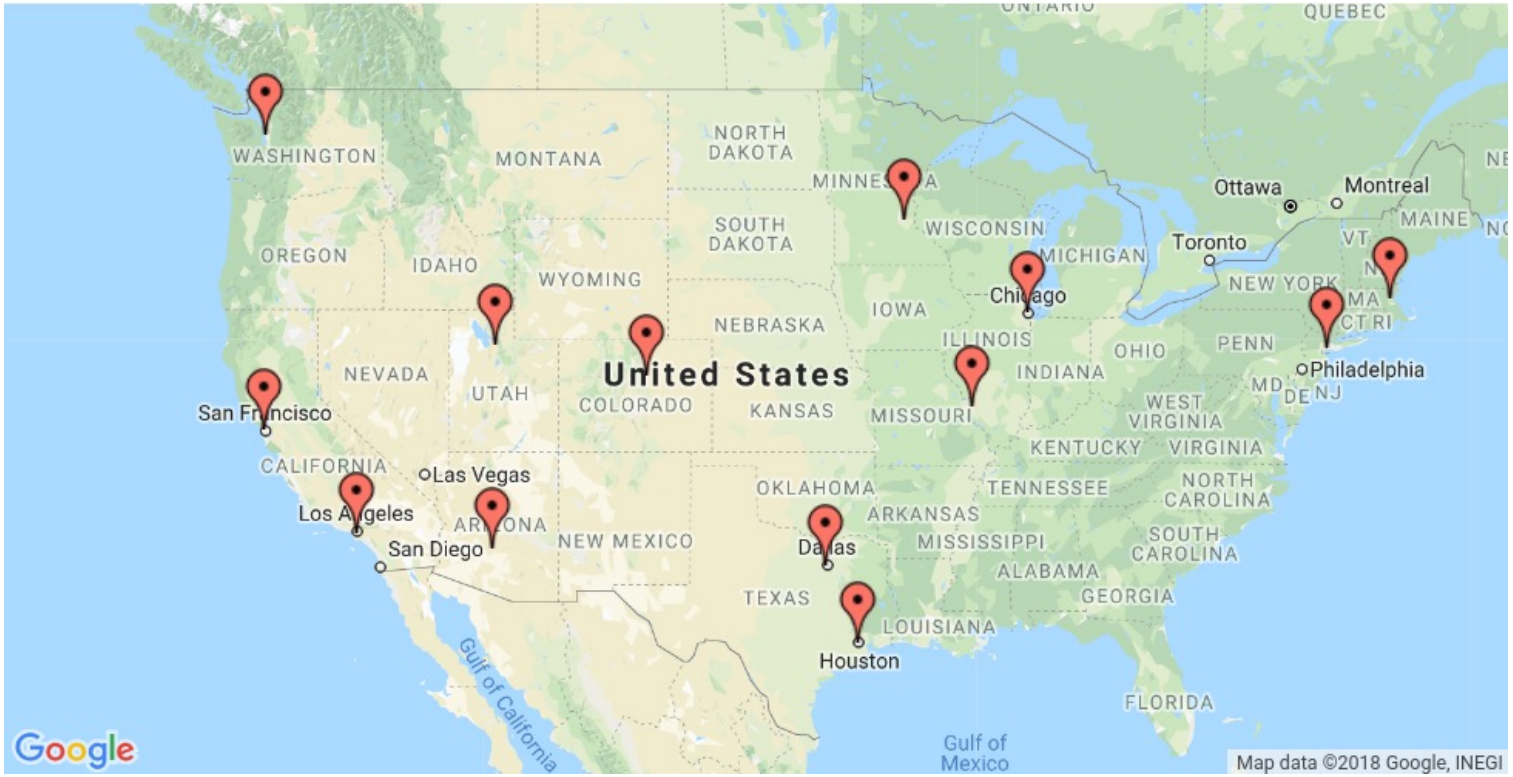
Où va-t-on ?

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Voyageur de commerce

Etant donné **N** villes (**N** sommets d'un graphe avec des distances) et une borne **B**, existe-t-il un circuit de longueur au plus **B** qui relie ces villes ?

Voyageur de commerce **Traveling Salesman Problem (TSP)**



Voyageur de commerce

Etant données **N** villes (**N** sommets d'un graphe avec des distances) et une borne **B**, existe-t-il un circuit de longueur au plus **B** qui relie ces villes ?

Voyageur de commerce

Etant donné **N** villes (**N** sommets d'un graphe avec des distances) et une borne **B**, existe-t-il un circuit de longueur au plus **B** qui relie ces villes ?

Pour toute permutation des sommets
  Si le circuit correspondant a longueur $\leq B$
  Alors le renvoyer
Renvoyer *« impossible »*

N! permutations

On connait de meilleurs algorithmes, de complexité 2^N,
mais peut on faire mieux ????

Voyageur de commerce

Etant données **N** villes (**N** sommets d'un graphe avec des distances) et une borne **B**, existe-t-il un circuit de longueur au plus **B** qui relie ces villes ?

Il est (encore) facile de vérifier qu'une solution est valide

Où va-t-on ?

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Récapitulons

- La complexité (temporelle) d'un algorithme est le nombre maximum (en « pire cas ») d'« opérations » réalisées en fonction de la taille **N** du problème.
- La complexité d'un problème est la meilleure complexité d'un algorithme connu pour résoudre ce problème.
- **Classe P** (Polynomial) : ensemble des pbms qu'on sait résoudre en temps polynomial.
- **Vérifier la solution d'un problème $\neq?$ Résoudre ce problème.**

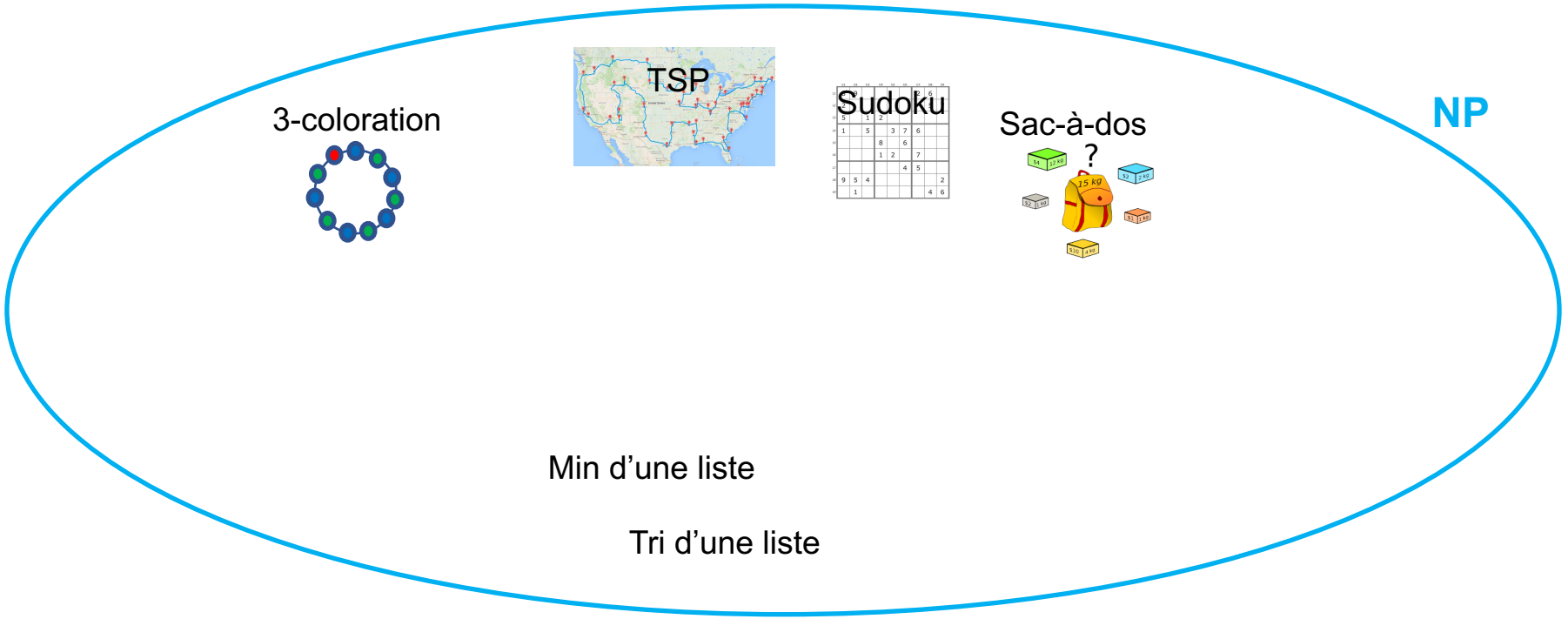
Récapitulons

- La complexité (temporelle) d'un algorithme est le nombre maximum (en « pire cas ») d'« opérations » réalisée en fonction de la taille **N** du problème.
- La complexité d'un problème est la meilleure complexité d'un algorithme connu pour résoudre ce problème.
- **Classe P** (Polynomial) : ensemble des pbms qu'on sait résoudre en temps polynomial.
- **Vérifier la solution d'un problème ≠? Résoudre ce problème.**

	Vérification d'une solution en temps polynomial ?	Calcul d'une solution en temps polynomial ?	« Meilleure » complexité connue pour résoudre
Minimum d'une liste de N éléments	OUI	OUI **Classe P**	N
Tri d'une liste de N éléments	OUI	OUI **Classe P**	N*log(N)
3-coloration d'un graphe de N sommets	OUI	?	1.3287^N
Sac-à-dos avec N objets	OUI	?	2^N
TSP avec N villes	OUI	?	2^N
Sudoku d'une grille N*N	OUI	?	N^{N*N}

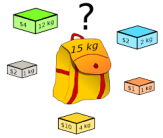
NP : Non-deterministic Polynomial

- **Classe NP** (Non-deterministic Polynomial) : ensemble des problèmes dont on sait vérifier une solution en temps polynomial.



NP : Non-deterministic Polynomial

- **Classe NP** (Non-deterministic Polynomial) : ensemble des problèmes dont on sait vérifier une solution en temps polynomial.



3-coloration

NP

Min d'une liste

Tri d'une liste

NP : Non-deterministic Polynomial

- **Classe NP** (Non-deterministic Polynomial) : ensemble des problèmes dont on sait vérifier une solution en temps polynomial.

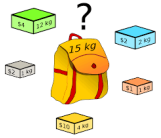
NP

3-coloration

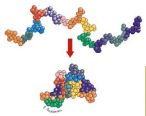
TSP

Sudoku

Sac-à-dos



Repliement de protéines



Tetris

SAT

$$(a_1 \vee a_2 \vee u_1) \wedge (a_3 \vee \neg u_1 \vee u_2) \wedge (a_4 \vee \neg u_2 \vee u_3) \wedge \ldots$$
$$\ldots (a_{p-2} \vee \neg u_{p-4} \vee u_{p-3}) \wedge (a_{p-1} \vee a_p \vee \neg u_{p-3}).$$

Mario Bross

Facteurs premiers

Multiplication de matrices

Test de primalité

Min d'une liste

Plus court chemin

Tri d'une liste

P vs. NP

- **Classe NP** (Non-deterministic Polynomial) : ensemble des problèmes dont on sait vérifier une solution en temps polynomial.
  - **Classe P** (Polynomial) : ensemble des problèmes que l'on sait résoudre en temps polynomial.

$P \subseteq NP$

NP

TSP

Sudoku

3-coloration

Sac-à-dos

?

Repliement de protéines

Tetris

SAT

$(a_1 \vee a_2 \vee u_1) \wedge (a_3 \vee \neg u_1 \vee u_2) \wedge (a_4 \vee \neg u_2 \vee u_3) \wedge \ldots$
$\ldots (a_{p-2} \vee \neg u_{p-4} \vee u_{p-3}) \wedge (a_{p-1} \vee a_p \vee \neg u_{p-3}).$

Mario Bross

Facteurs premiers

Multiplication de matrices

Test de primalité

Min d'une liste

Plus court chemin

P

Tri d'une liste

P vs. NP

- **Classe NP** (Non-deterministic Polynomial) : ensemble des problèmes dont on sait vérifier une solution en temps polynomial.
  - **Classe P** (Polynomial) : ensemble des problèmes que l'on sait résoudre en temps polynomial.

P⊆NP

NP

TSP

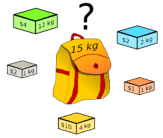
3-coloration

Tetris

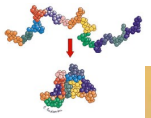
SAT

$$(a_1 \vee a_2 \vee u_1) \wedge (a_3 \vee \neg u_1 \vee u_2) \wedge (a_4 \vee \neg u_2 \vee u_3) \wedge \ldots$$
$$\ldots (a_{p-2} \vee \neg u_{p-4} \vee u_{p-3}) \wedge (a_{p-1} \vee a_p \vee \neg u_{p-3}).$$

Sac-à-dos ?



Repliement de protéines



Mario Bross

Facteurs premiers

Multiplication de matrices

Test de primalité
[Agrawal-Kayal-Saxena, 2002]

Min d'une liste

Plus court chemin

P

Tri d'une liste

Classification évolutive !

Le problème « P = NP ? »

NP

TSP

3-coloration

Sudoku

Sac-à-dos ?

Repliement de protéines

Tetris

SAT

$$(a_1 \vee a_2 \vee u_1) \wedge (a_3 \vee \neg u_1 \vee u_2) \wedge (a_4 \vee \neg u_2 \vee u_3) \wedge \dots$$
$$\dots (a_{p-2} \vee \neg u_{p-4} \vee u_{p-3}) \wedge (a_{p-1} \vee a_p \vee \neg u_{p-3}).$$

Mario Bross

Facteurs premiers

Multiplication de matrices

Test de primalité
[Agrawal-Kayal-Saxena, 2002]

Min d'une liste

Plus court chemin

P

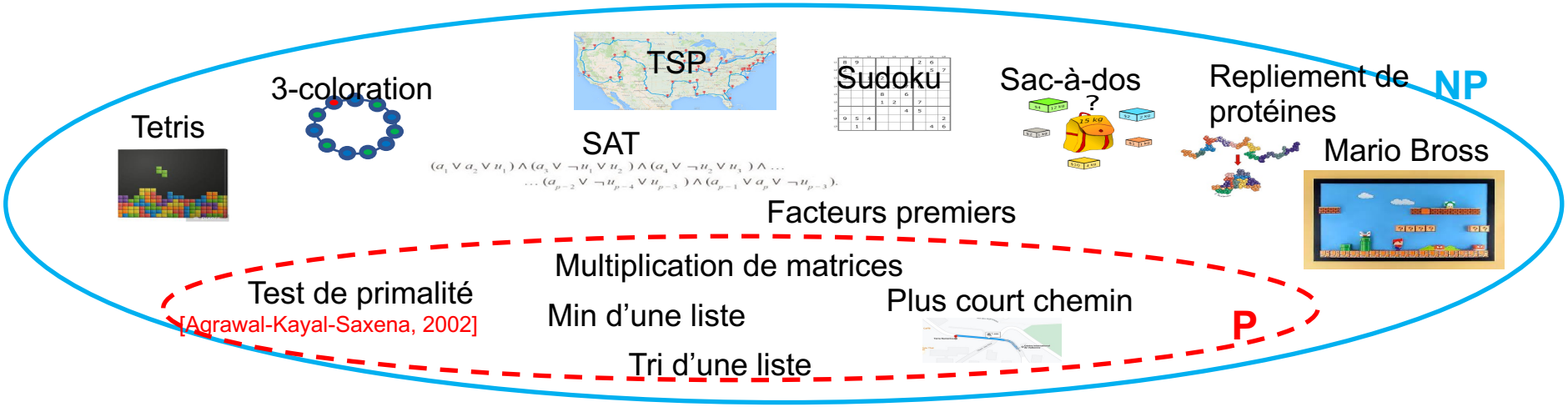
Tri d'une liste

Classification évolutive !

Est-ce que « P = NP » ?
Est-il possible de résoudre tous les problèmes de NP en temps polynomial ?
Est-ce que pouvoir vérifier rapidement une solution à un problème implique de pouvoir la trouver rapidement ?

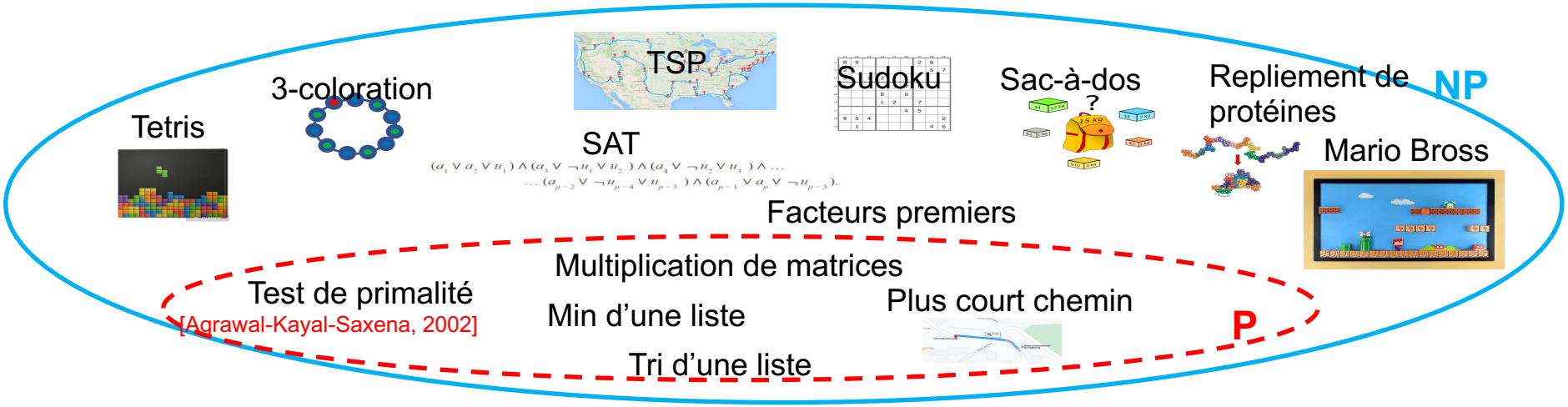
1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

« P = NP » ou « P ≠ NP » ?



1re approche : prouver que **P ≠ NP** : trouver un problème X∈NP tel qu'on ne peut pas le résoudre en temps polynomial… **borne inférieure** (difficile ?)

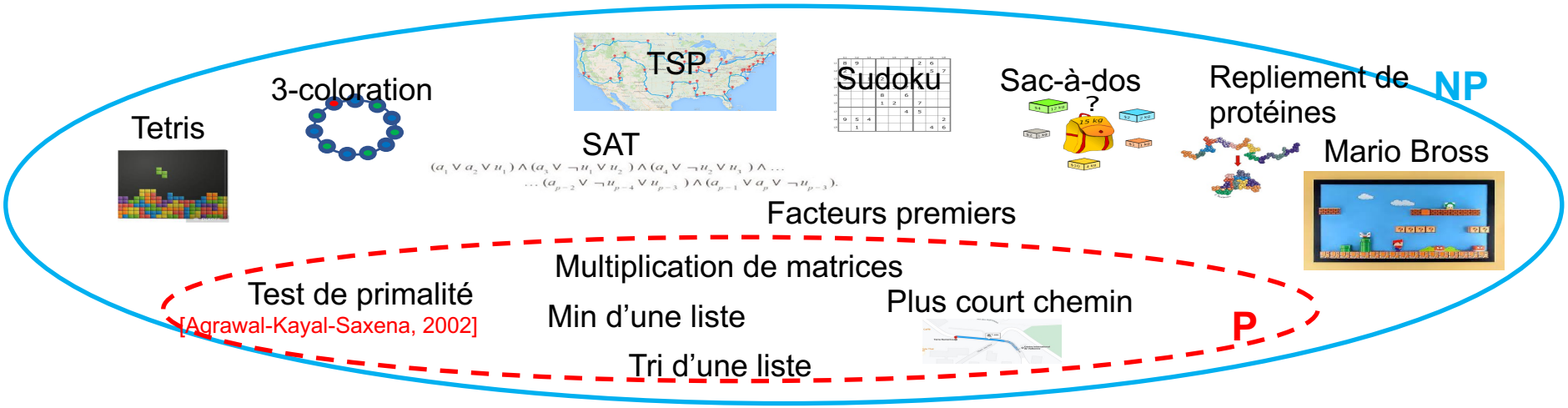
« P = NP » ou « P ≠ NP » ?



1re approche : prouver que **P ≠ NP** : trouver un problème X∈NP tel qu'on ne peut pas le résoudre en temps polynomial… **borne inférieure** (difficile ?)

2me approche : prouver que **P = NP** : prouver que tout problème de NP admet un algorithme polynomial **ça fait beaucoup de problèmes…**

« P = NP » ou « P ≠ NP » ?

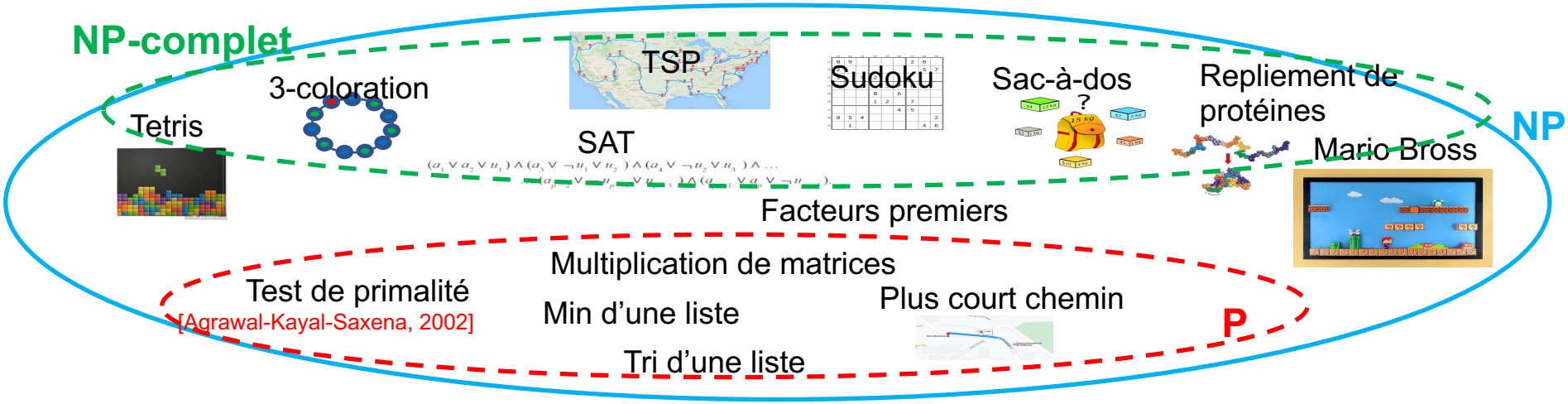


1re approche : prouver que **P ≠ NP** : trouver un problème X∈NP tel qu'on ne peut pas le résoudre en temps polynomial… **borne inférieure** (difficile ?)

2me approche : prouver que **P = NP** : prouver que tout problème de NP admet un algorithme polynomial **ça fait beaucoup de problèmes…**

3me approche : prouver que « **P = NP** » est indécidable... (Th. de Gödel)

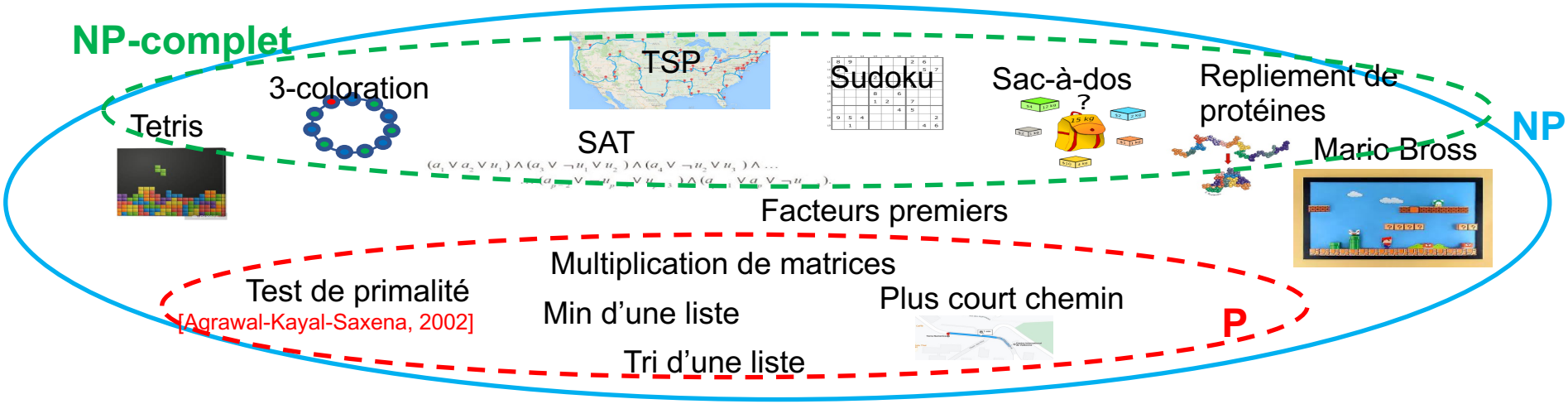
« P = NP » ou « P ≠ NP » ?



2me approche (bis) :

NP-complet : problèmes de NP « au moins aussi difficiles que tous les autres problèmes de la classe NP » [Cook-Levin, 1971]

« P = NP » ou « P ≠ NP » ?

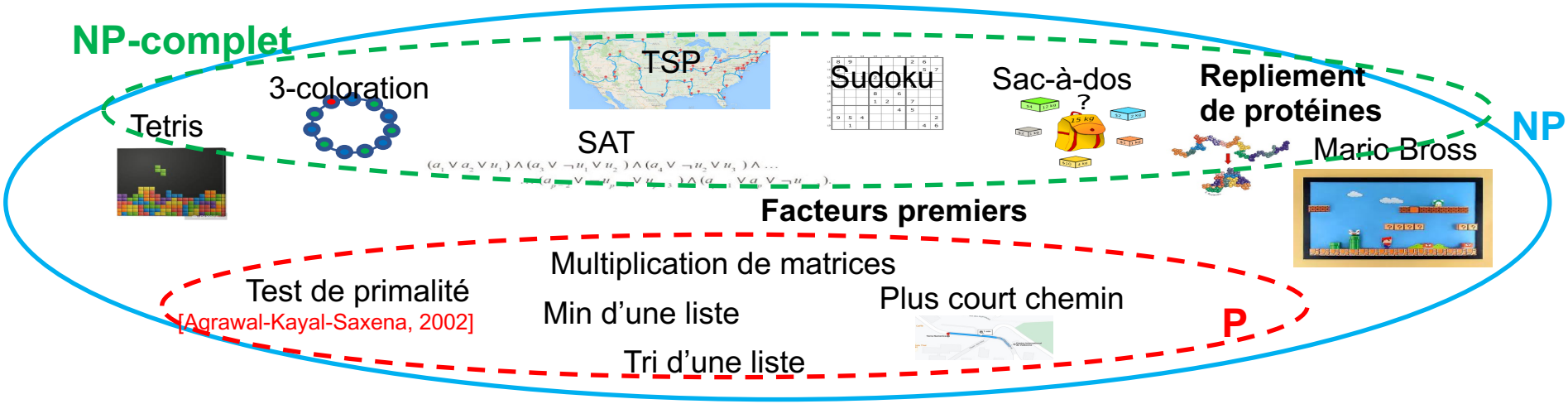


2me approche (bis) :

NP-complet : problèmes de NP « au moins aussi difficiles que tous les autres problèmes de la classe NP » [Cook-Levin, 1971]

prouver que **P** = **NP** : il « suffit » de prouver qu'un (n'importe lequel) problème NP-complet admet un algorithme polynomial

« P = NP » ou « P ≠ NP » ?



Et si P = NP ?

Décomposition en facteurs premiers

15 = 3*5 ; 561 = 3*11*17 ; 9841583561=79687*123503

applications en cryptologie (sécurisation des communications sur Internet)

Repliement de protéines : synthèse de médicaments, compréhension de maladies…

Mais : polynomial n'est pas nécessairement « facile » ex: N^{2023}

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion

Ce que, j'espère, vous retiendrez

1. **Complexité temporelle** (nombres d'opérations) :
   mesure de l'efficacité d'un algorithme
2. **P** : classe des problèmes que l'on sait résoudre en temps polynomial
   ex: recherche, tri…
3. **NP** : classe des problèmes dont on peut vérifier une solution en temps polynomial
4. **« P = NP ? »** : « Est-ce que pouvoir vérifier **rapidement** une solution à un problème implique de pouvoir la trouver **rapidement** ? »
5. Nombreux problèmes importants que l'on ne sait « que » résoudre en temps exponentiel. Peut-on faire mieux ?
   ex : TSP, SAT, sac-à-dos, repliement de protéines, coloration…

Ce que, j'espère, vous retiendrez

1. **Complexité temporelle** (nombres d'opérations) : mesure de l'efficacité d'un algorithme
2. **P** : classe des problèmes que l'on sait résoudre en temps polynomial ex: recherche, tri…
3. **NP** : classe des problèmes dont on peut vérifier une solution en temps polynomial
4. **« P = NP ? »** : « Est-ce que pouvoir vérifier **rapidement** une solution à un problème implique de pouvoir la trouver **rapidement** ? »
5. Nombreux problèmes importants que l'on ne sait « que » résoudre en temps exponentiel. Peut-on faire mieux ?
   ex : TSP, SAT, sac-à-dos, repliement de protéines, coloration…

MERCI POUR VOTRE ATTENTION ! **QUESTIONS ?**

1. Introduction
2. Trois problèmes
   - Recherche de minimum dans une liste
   - Tri d'une liste
   - 3-coloration d'un graphe
3. Complexité temporelle d'un algorithme et d'un problème
4. Exemples de problèmes « difficiles »
   - Sudoku
   - Sac-à-dos
   - Voyageur de commerce
5. Le problème « P = NP ? »
6. Pistes pour le résoudre et applications
7. Conclusion
8. BONUS : et en pratique ?

Retour sur le Voyageur de commerce

« quel doit être mon parcours optimal ? »

N=45 (44 villes + Pôle Nord)

Algorithme de complexité : $\mathbf{2^N}$

$2^{45}=3{,}52*10^{13}$

Pas de cadeaux cette année ?

Retour sur le Voyageur de commerce

« quel doit être mon parcours optimal ? »

N=45 (44 villes + Pôle Nord)

Algorithme de complexité : $\mathbf{2^N}$

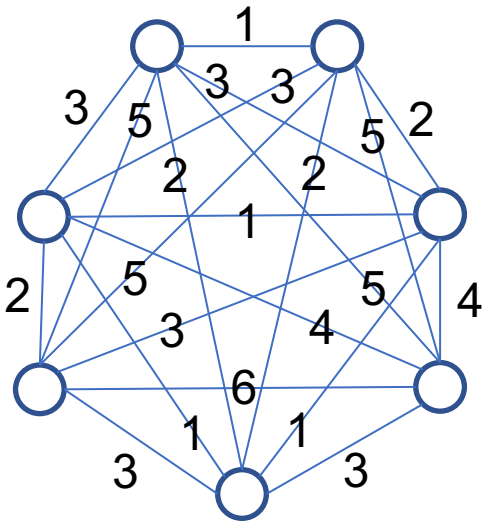
$2^{45}=3{,}52*10^{13}$

Pas de cadeaux cette année ?

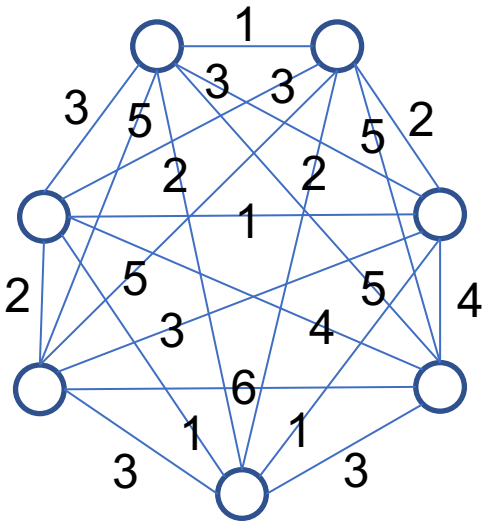
Algorithme non optimal ?

- **Heuristique** : calcule rapidement une solution valide sans garantie de sa qualité.
- **Approximation** : calcule rapidement une solution valide dont la qualité est « proche » de l'optimal

Voyageur de commerce : 2-approximation

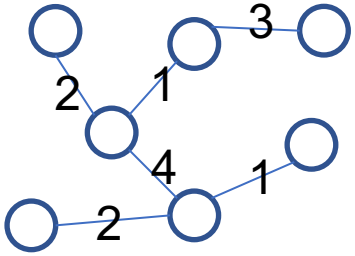


Voyageur de commerce : 2-approximation



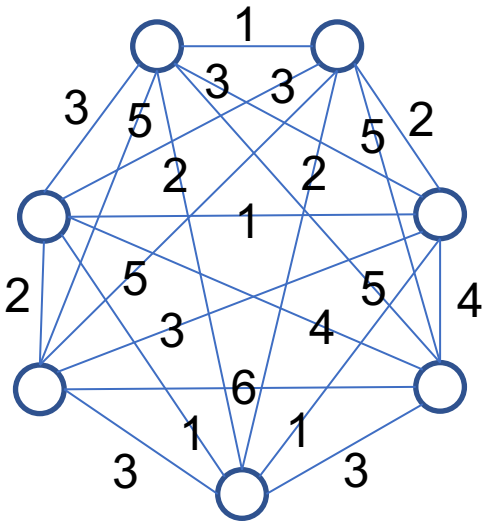
Détour : Arbre Couvrant Minimum

??



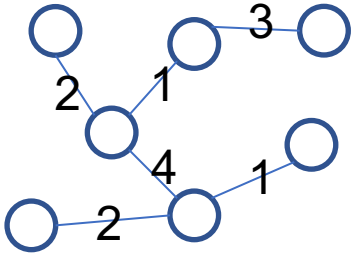
Arbre = graphe connexe sans cycle
(ici, de poids 13)

Voyageur de commerce : 2-approximation



Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

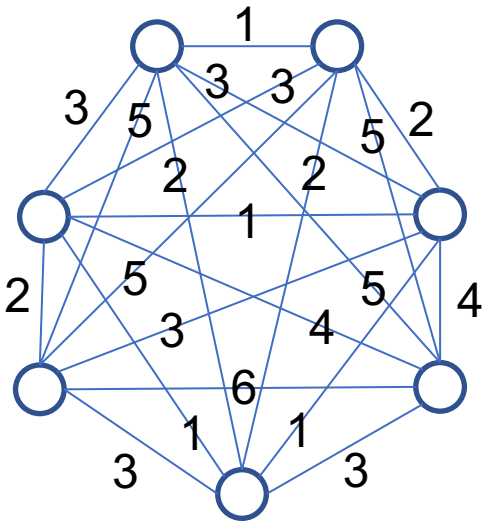
Algorithme optimal pour ACM

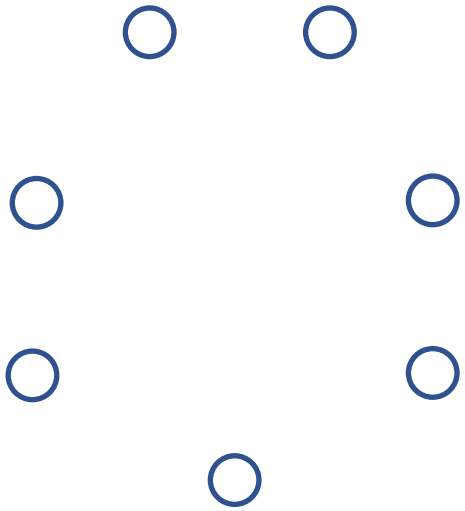
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

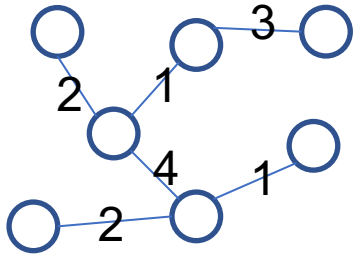
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

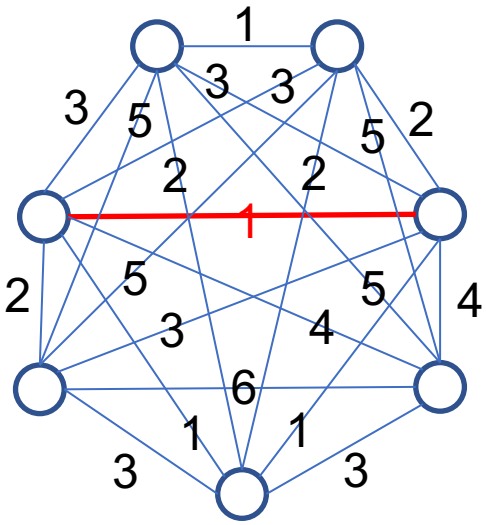
Algorithme optimal pour ACM

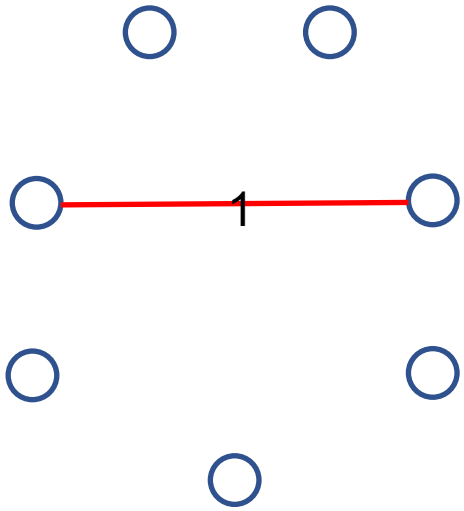
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

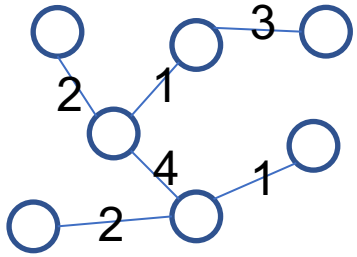
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

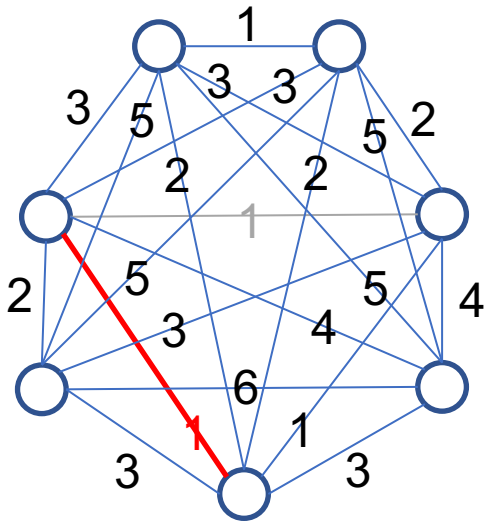
Algorithme optimal pour ACM

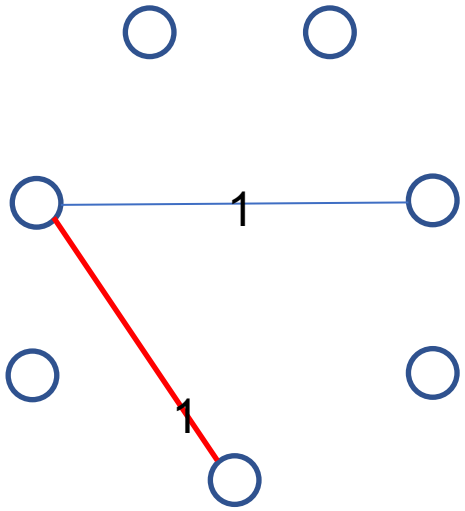
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

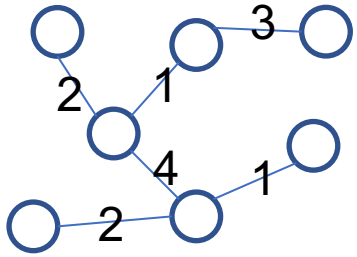
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

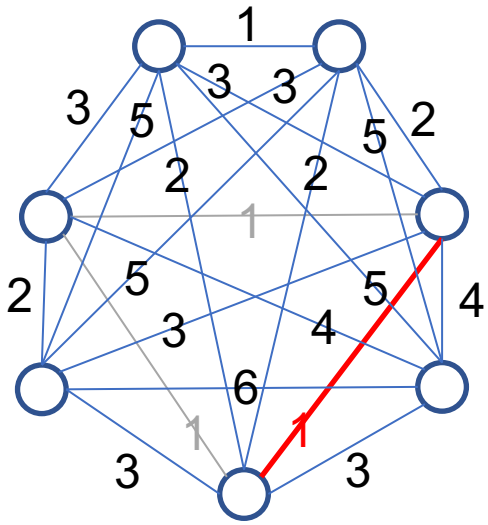
Algorithme optimal pour ACM

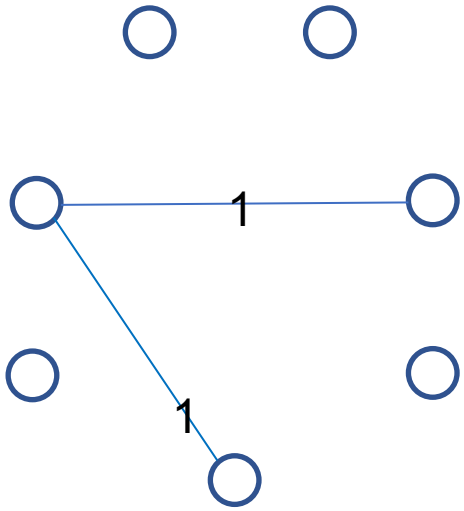
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

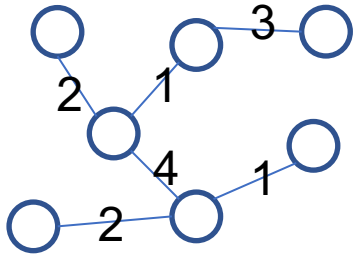
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

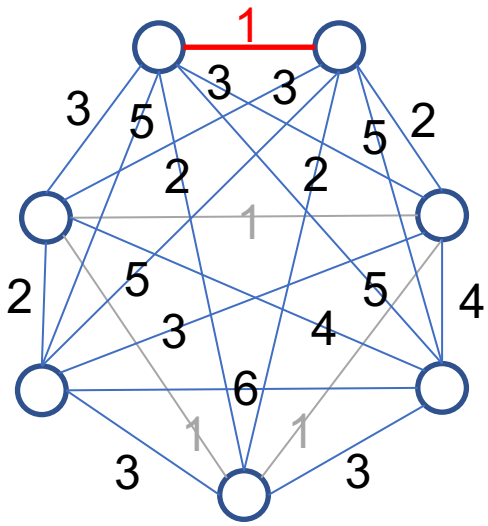
Algorithme optimal pour ACM

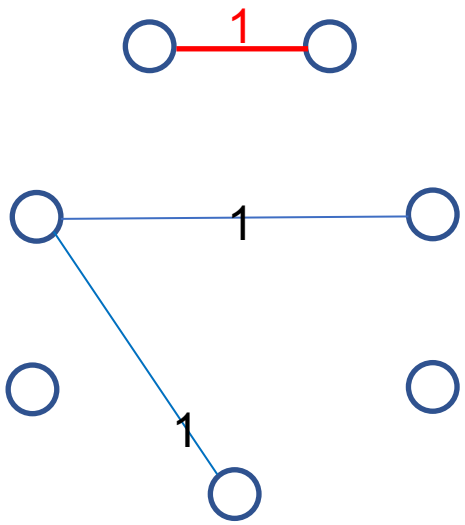
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

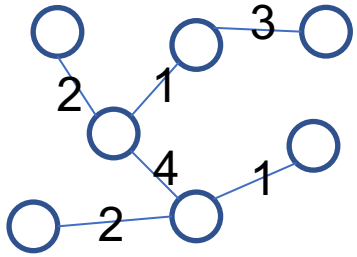
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

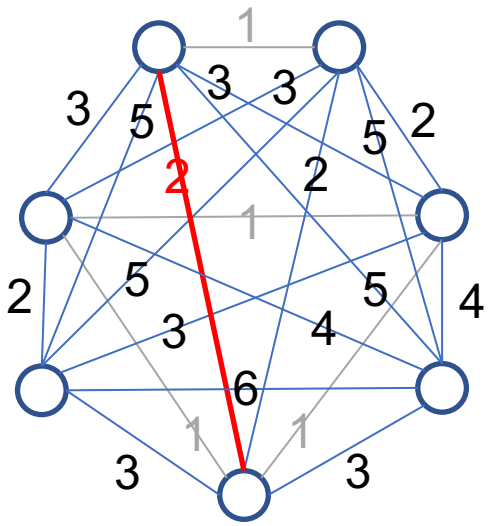
Algorithme optimal pour ACM

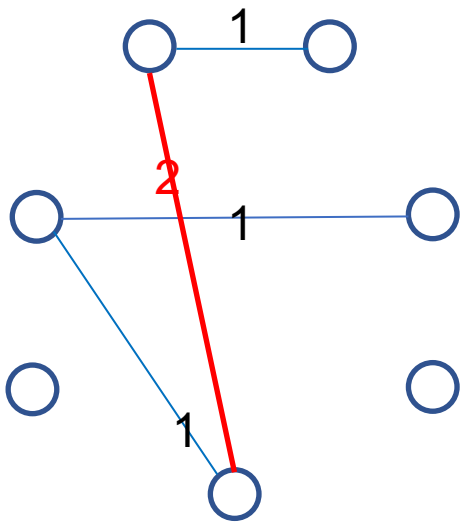
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

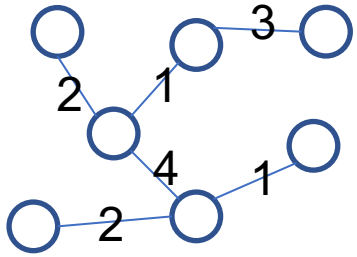
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

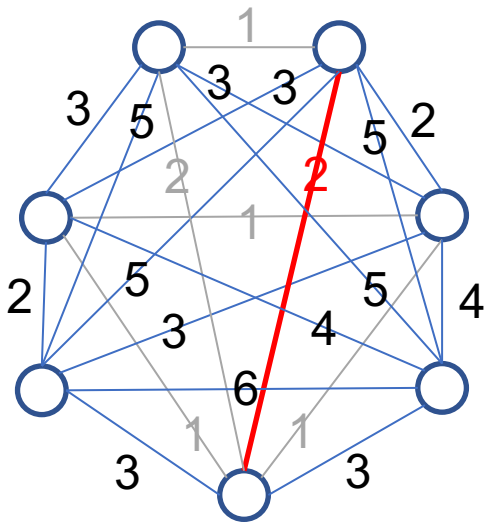
Algorithme optimal pour ACM

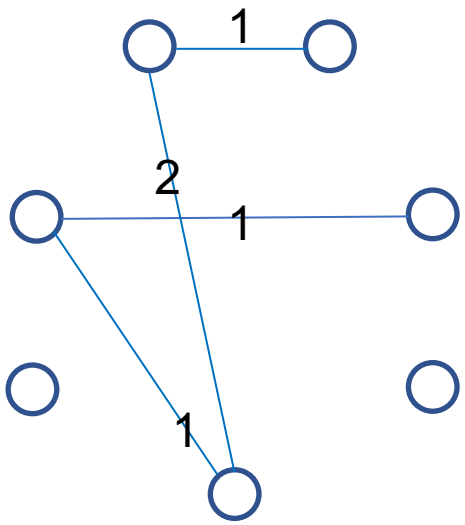
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

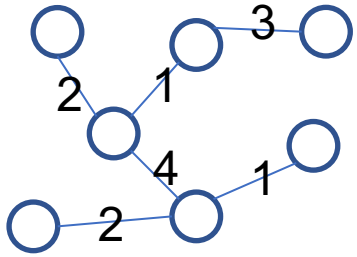
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

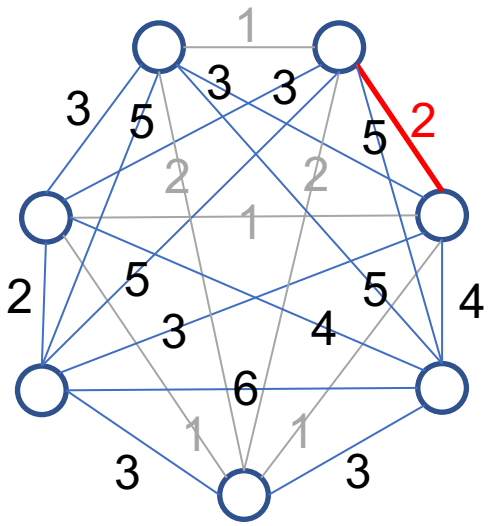
Algorithme optimal pour ACM

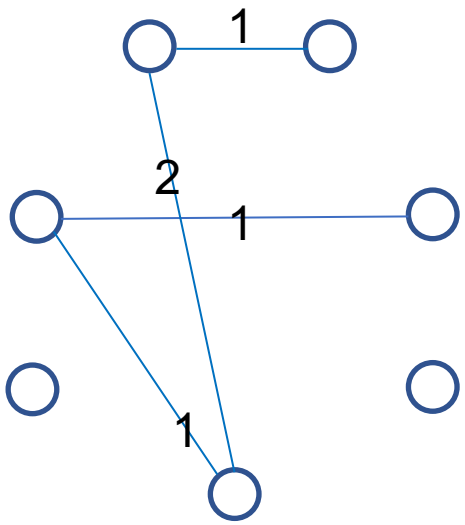
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

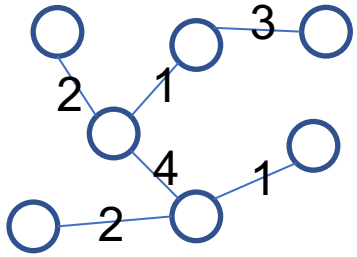
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

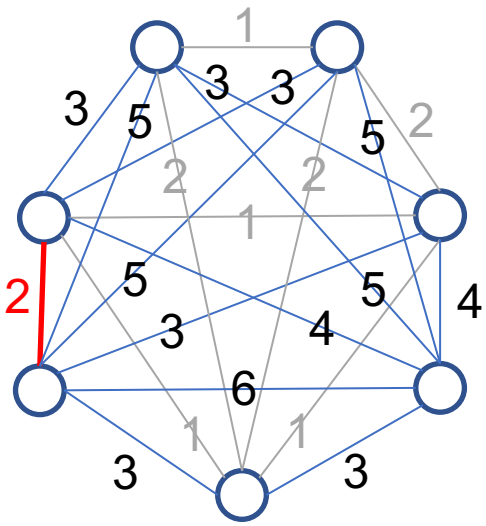
Algorithme optimal pour ACM

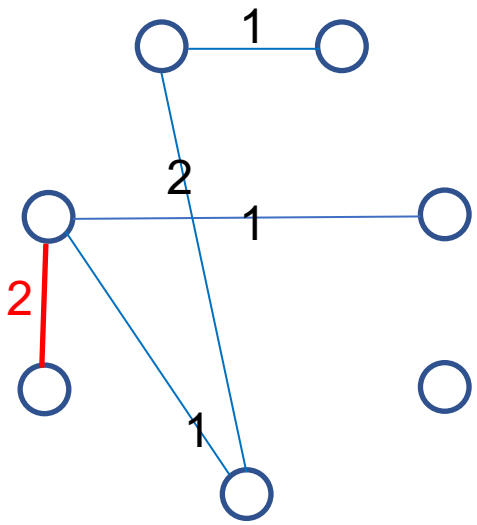
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

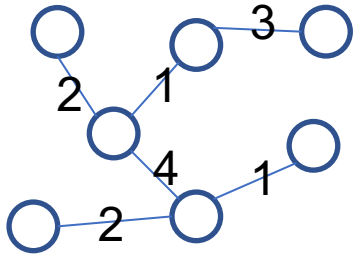
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

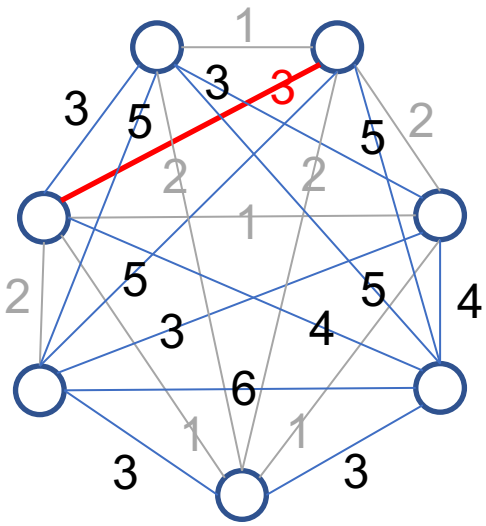
Algorithme optimal pour ACM

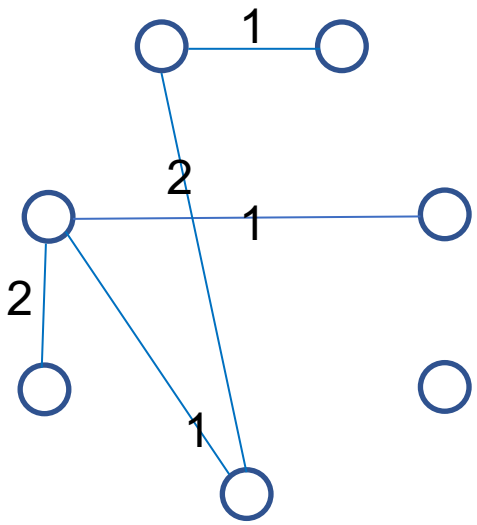
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

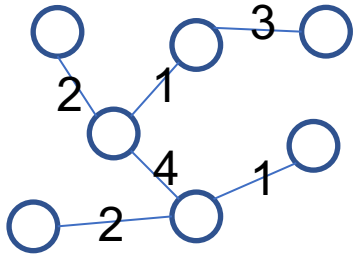
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

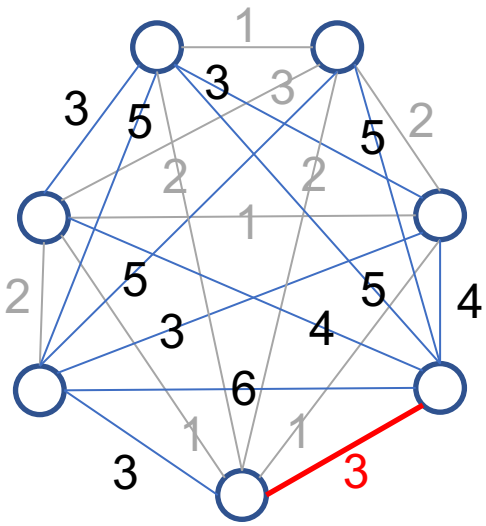
Algorithme optimal pour ACM

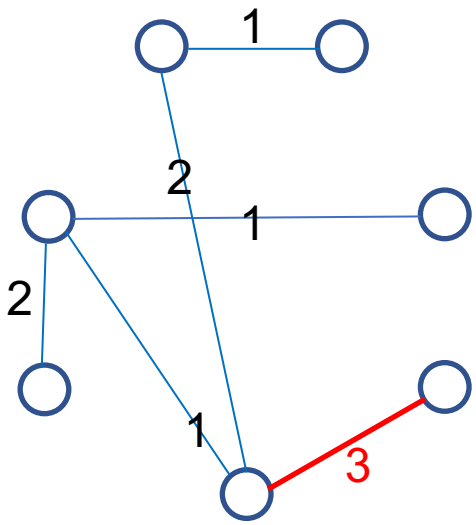
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

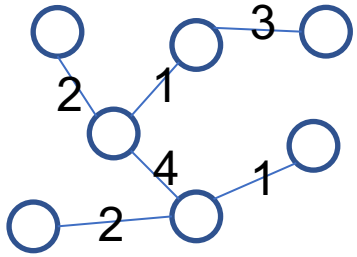
Voyageur de commerce : 2-approximation





Détour : Arbre Couvrant Minimum

??



Arbre = graphe connexe sans cycle
(ici, de poids 13)

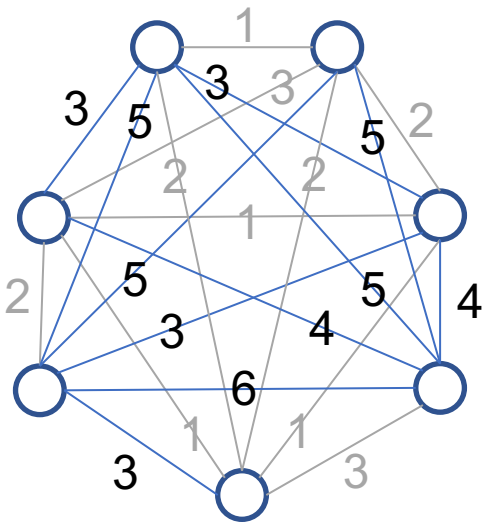
Algorithme optimal pour ACM

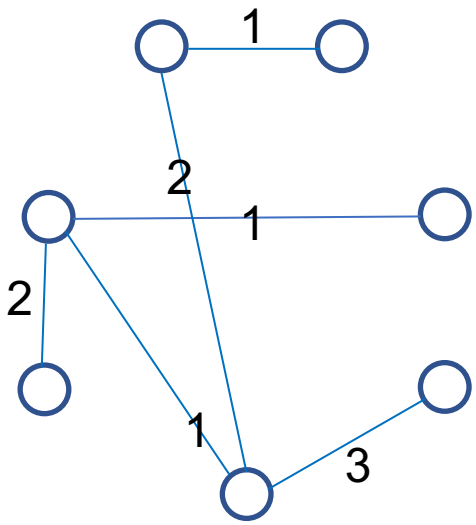
[Borůvska 1926, Kruskal 1956, Prim 1959…]

Ordonner les arêtes par poids croissants
Dans cet ordre, ne conserver une arête que si elle ne crée pas de cycle avec celles déjà sélectionnées.

(1,1,1,1,2,2,2,2,3,3,3,3,3,3,4,4,5,5,5,5,6)

Voyageur de commerce : 2-approximation





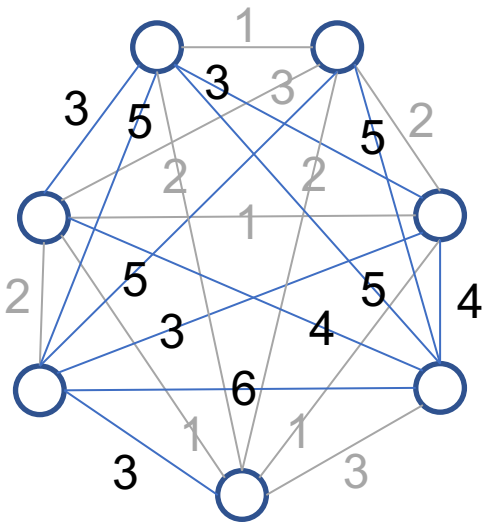
Trouver un Arbre Couvrant Minimum (ACM) dans un graphe de N sommets et M arêtes.

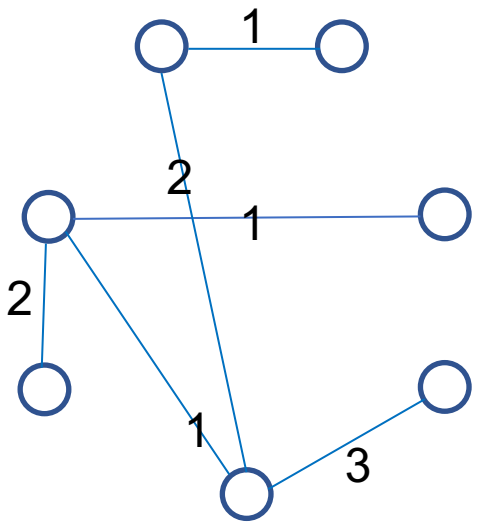
Trier les arêtes : M log(M)

ici : $M = \frac{N(N+1)}{2}$

Dans l'exemple : poids min AC =10

Voyageur de commerce : 2-approximation





Trouver un Arbre Couvrant Minimum (ACM) dans un graphe de N sommets et M arêtes.

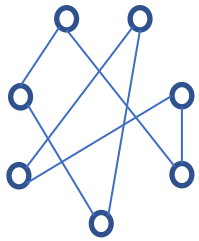
Trier les arêtes : M log(M)

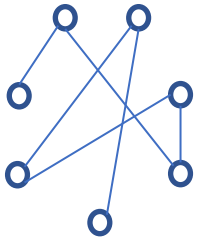
ici : $M = \frac{N(N+1)}{2}$

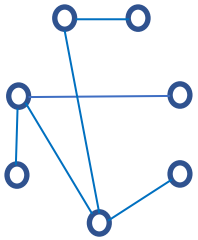
Dans l'exemple : poids min AC =10

Théorème : pour tout graphe : poids min AC ≤ longueur min. d'un tour.

Preuve :

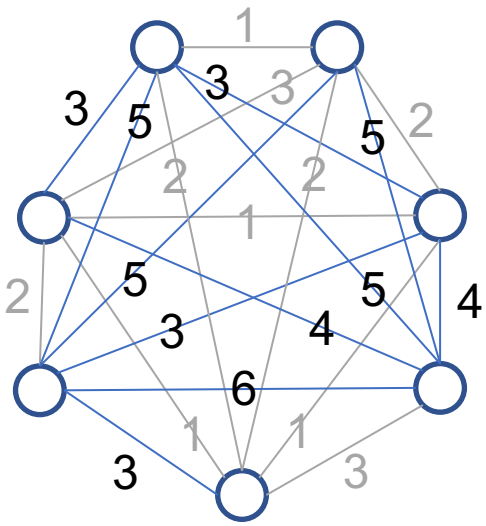


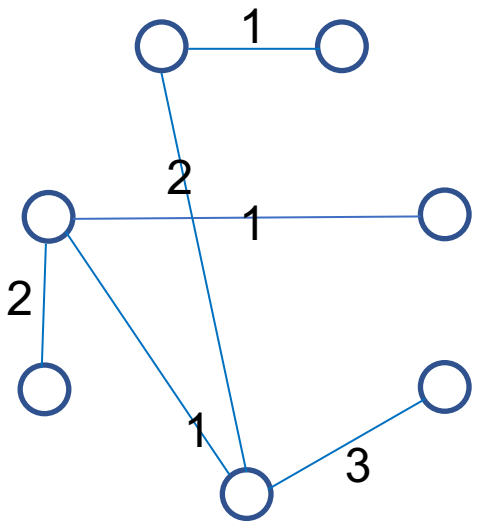




Tour de poids min. ≥ arbre couvrant ≥ poids min. d'un AC

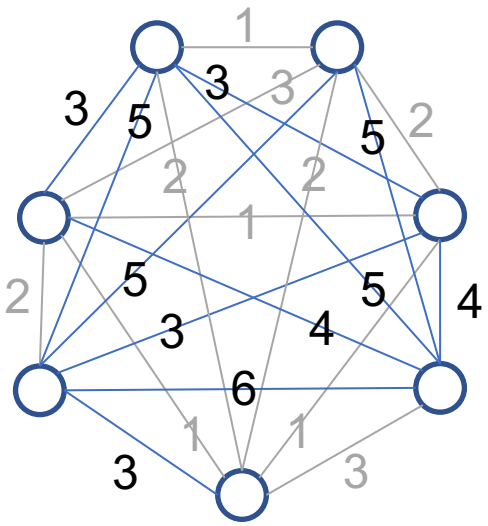
Voyageur de commerce : 2-approximation

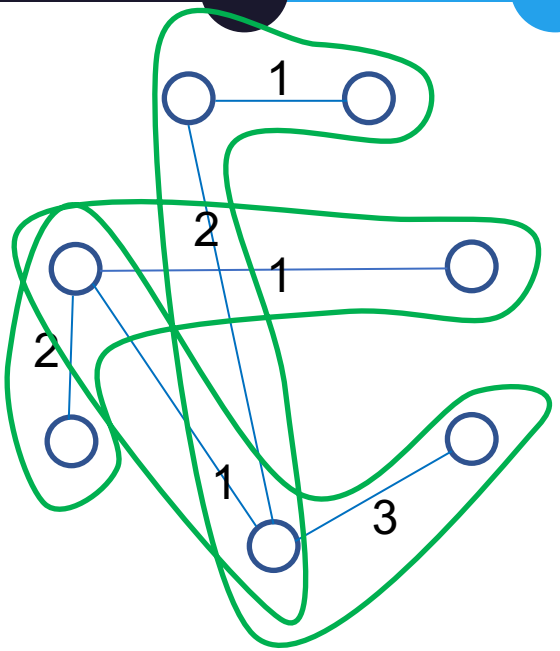




poids min AC =10 ≤ longueur min. d'un tour

Voyageur de commerce : 2-approximation

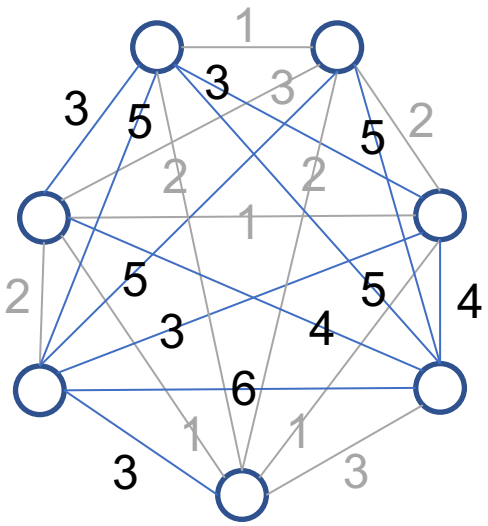


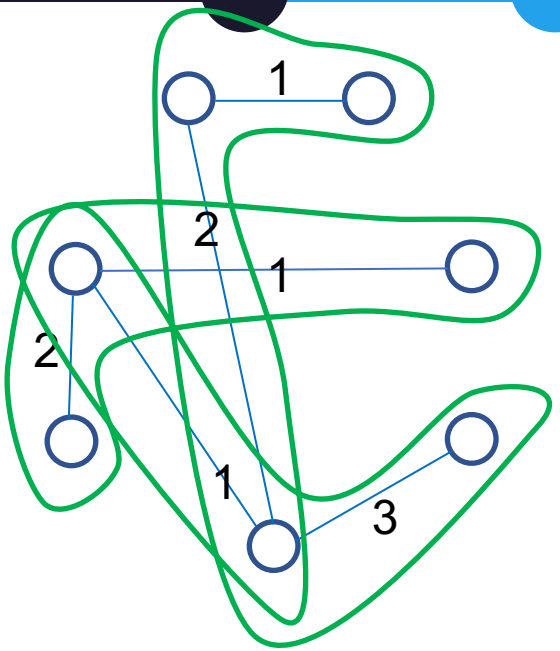


poids min AC =10 ≤ longueur min. d'un tour

Poids(C) =20 ≤ 2* longueur min. d'un tour

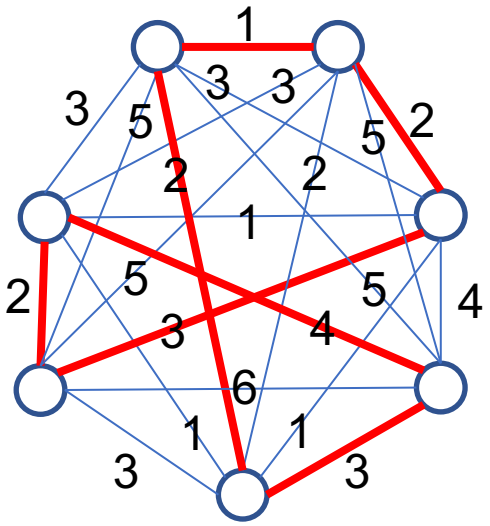
Voyageur de commerce : 2-approximation



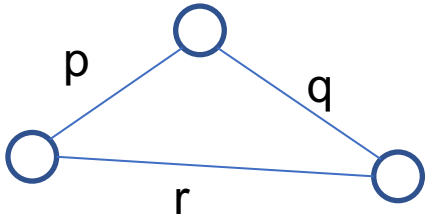


poids min AC =10 ≤ longueur min. d'un tour

Poids(C) =20 ≤ 2* longueur min. d'un tour

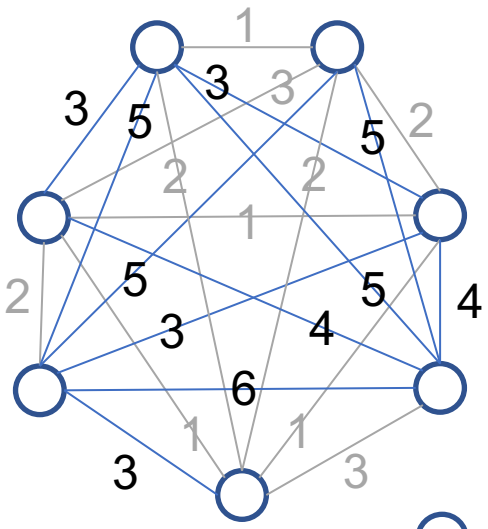


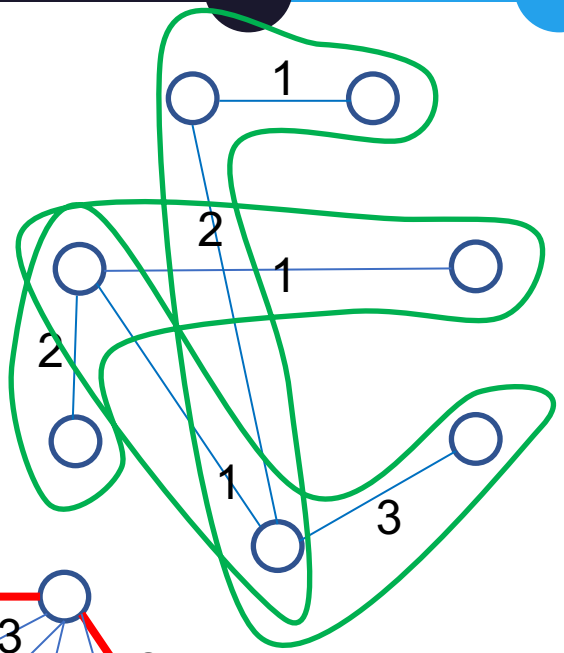
Si inégalité triangulaire :



r ≤ p+q

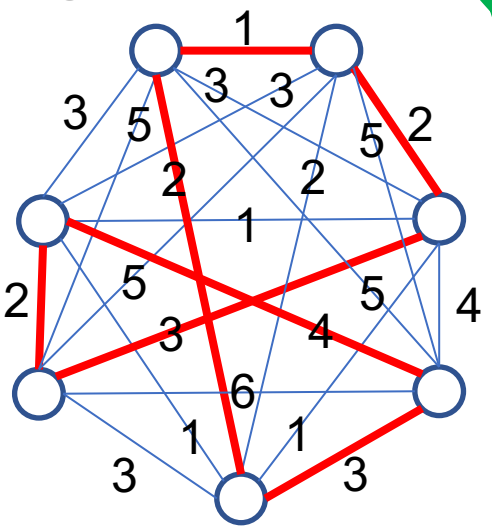
Voyageur de commerce : 2-approximation





poids min AC =10 ≤ longueur min. d'un tour

Poids(C) =20 ≤ 2* longueur min. d'un tour



Si inégalité triangulaire :

Algorithme de complexité $O(N^2 \log N)$ qui calcule un cycle Hamiltonien de longueur au plus 2*OPT

Slides complémentaires

7 problèmes du millénaire

$$\zeta(z) = \sum_{k=1}^{\infty} \frac{1}{k^z} = \prod_{k\ premier} \frac{1}{1-k^{-z}}$$

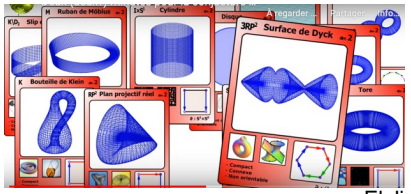
- **Hypothèse de Riemann** (1859) Analyse ↔ Arithmétique

« les zéros non triviaux de la fonction zêta de Riemann ont tous pour partie réelle ½ »

« If I were to awaken after having slept for a thousand years, my first question would be: Has the Riemann hypothesis been proven? »
D. Hilbert

- ~~Conjecture de Poincaré (1904)~~ Théorème de Perelman, 2003

« Toute 3-variété compacte sans bord et simplement connexe est homéomorphe à la 3-sphère »



El Jj

- **Problème P = NP ?** (1971) Informatique ↔ Mathématique

« Est-ce que pouvoir vérifier rapidement une solution à un problème implique de pouvoir la trouver rapidement ? »

- **Conjecture de Hodge** (1930)

« il est possible de calculer la cohomologie d'une variété algébrique projective complexe à partir de ses sous-variétés»

- **Conjecture de Birch et Swinnerton-Dyer** (1960)

« pour toute courbe elliptique sur le corps des rationnels, l'ordre d'annulation au centre de la bande critique de la fonction L associée est égal au rang de la courbe »

- **Equations de Navier-Stokes** (19e siècle)

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p + \nu \Delta \mathbf{v} + \mathbf{f}(\boldsymbol{x}, t)$$

« existence et régularité des solutions des équations de Navier-Stokes ? »

- **Equations de Yang-Mills** (années 1950)

géométrie, physique des particules…

(prix : 1 million de dollars)

Rappels (ou pas) de maths.

polynomial — exponentiel

log_2(N)	N	N∗ log_2(N)	N^2	N^4	2^N	3^N	N!	N^N
1	2	2	4	16	4	9	2	4
2	4	8	16	256	16	81	24	256
3	8	24	64	4096	256	6561	40320	$16{,}7.\ 10^6$
3,321…	10	33,21…	100	10000	1024	59049	$\geq 3{,}6.10^6$	10^{10}
4	16	64	256	65536	65536	$\geq 4.\ 10^7$	$\approx 21.\ 10^{12}$	$\geq 1{,}8.\ 10^{19}$
5	32	160	1024	$\geq 10^6$	$\approx 4{,}3.\ 10^9$	$\approx 1{,}85.\ 10^{15}$	$\approx 2{,}63.\ 10^{35}$	$\approx 1{,}46.\ 10^{48}$
7	128	896	16384	$\geq 26.\ 10^7$	$\approx 3{,}4.\ 10^{38}$	$\approx 1{,}18.\ 10^{61}$	$\approx 3{,}86.\ 10^{215}$	$\approx 5{,}3.\ 10^{269}$
10	1024	10240	$\geq 10^6$	$\geq 10^{12}$	∞	∞	∞	∞

Polynôme : $\sum_{k=0}^{d} c_k\ N^k$; ex: $N^7 + 3N^4 - 7N^2 + 4N$

Exponentielle : $c^N = c * c * \ldots * c$

Factorielle : $N! = N * (N-1) * \ldots * 2 * 1 \sim \sqrt{2\pi N}\left(\frac{N}{e}\right)^N$

10^6 = un million
10^{12} = 1000 milliards
10^{15} millionièmes de s. = 31 ans et 251 jours
10^{80} atomes dans l'univers observable
10^{120} parties d'échecs possibles