La classe NP

AAC

Sophie Tison
Université Lille 1
Master1 Informatique

Bilan du dernier cours

- Définition de la complexité d'un problème. ?

Bilan du dernier cours

- Définition de la complexité d'un problème.
  La complexité d'un problème est complexité minimale (en temps) dans le pire des cas d'un algorithme le résolvant.

Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème. Lesquelles?

Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
  - Méthode de l'adversaire ou de l'oracle
  - Arbres de décision
  - Réductions

Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
- La notion de propriété ?

Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
- La notion de propriété
  Propriété=problème de décision $\equiv$ langage

Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
- La notion de propriété
- Classes de problèmes, la classe P Définition de P?

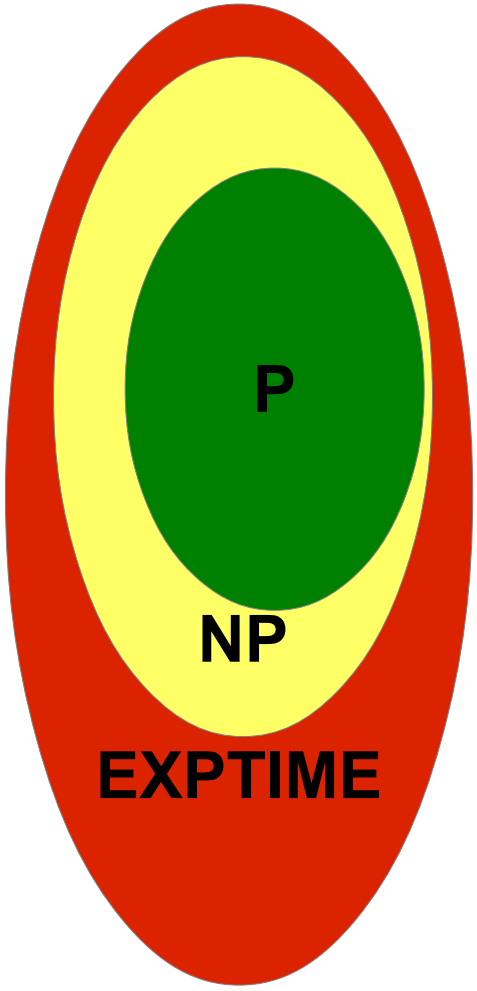
Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
- La notion de propriété
- Classes de problèmes, la classe P
  P(Time)=La classe des priorités qui peuvent être décidées en temps polynomial.

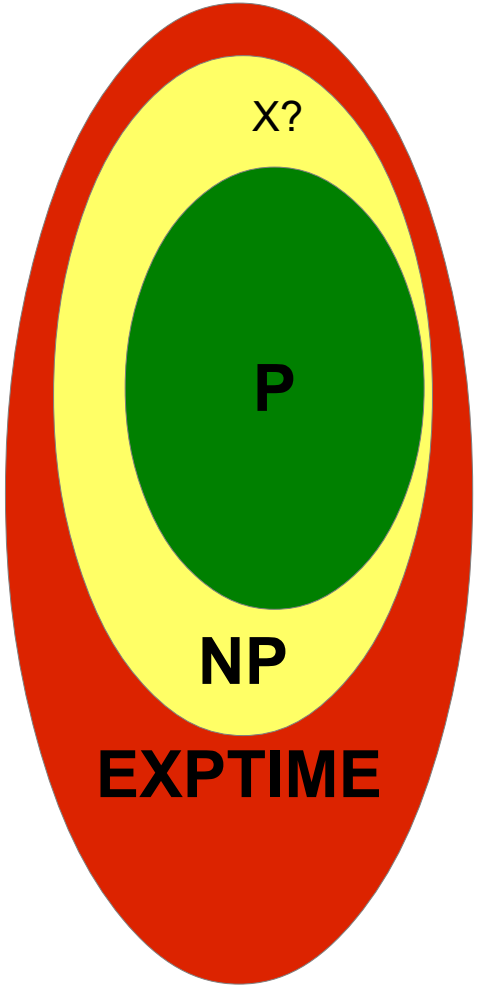
Bilan du dernier cours

- Définition de la complexité d'un problème.
- Quelques méthodes pour trouver une borne inf à la complexité d'un problème.
- La notion de propriété
- Classes de problèmes, la classe P

Aujourd'hui: la classe NP



Aujourd'hui: la classe NP



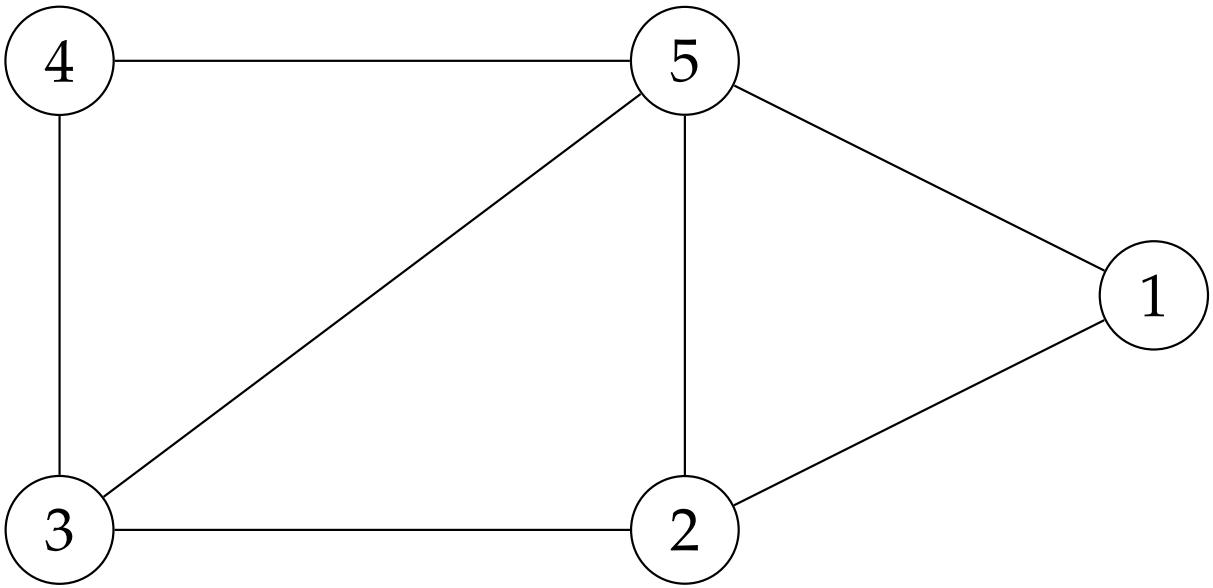
Exemple 1: Le 3-coloriage de graphes

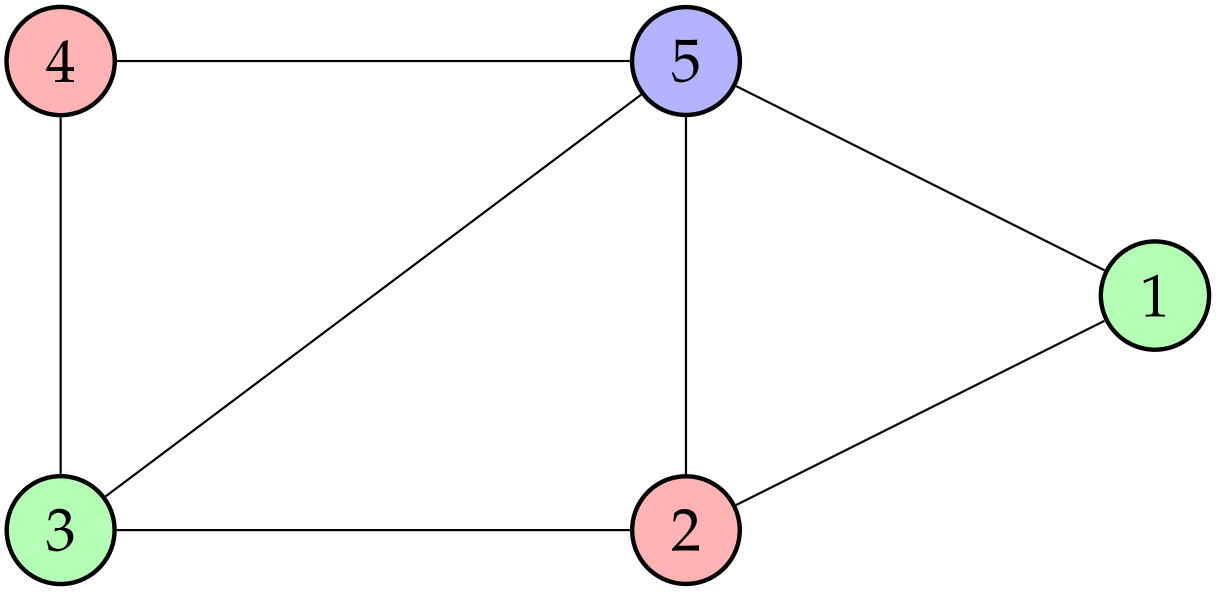
Donnée: $G = (S, A)$ un graphe

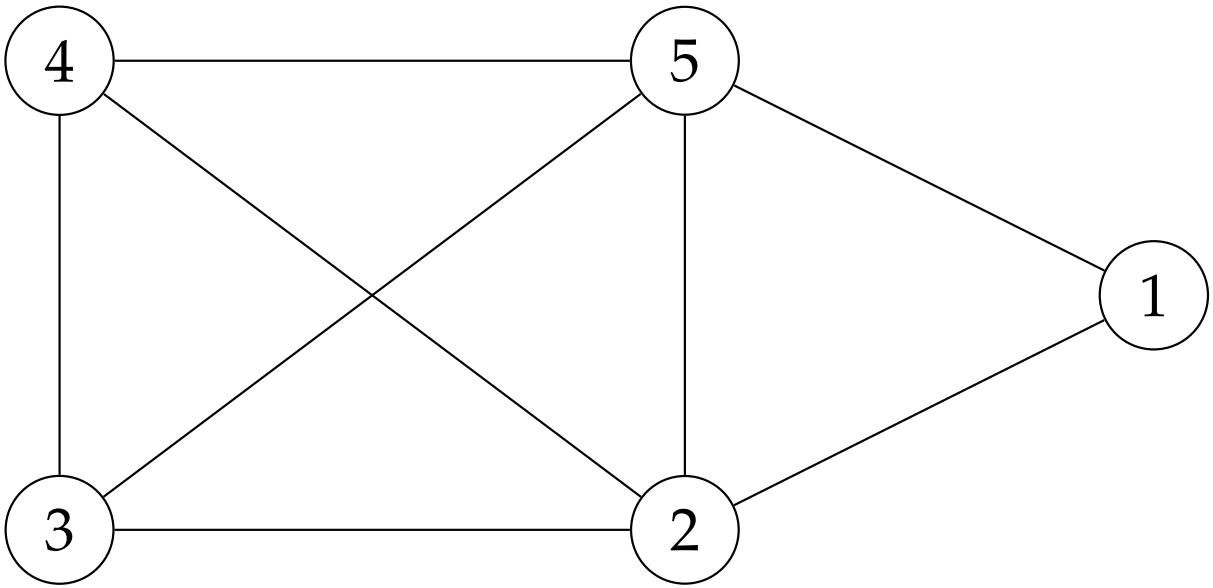
Sortie: oui, si le graphe peut être coloré en 3 couleurs i.e. on peut trouver:

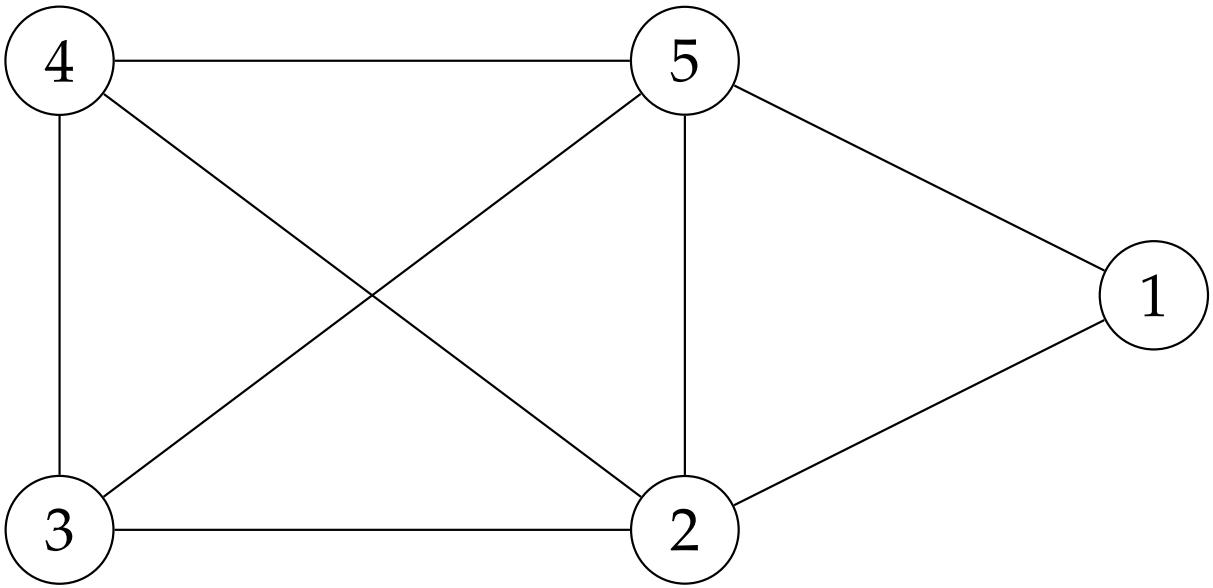
une application col de S dans $\{1, 2, 3\}$ telle que

$col(s) \neq col(s')$ si s, s' sont reliés par un arc ($(s, s') \in A$ ou $(s', s) \in A$)


4
5
1
3
2


4
5
1
3
2


4
5
1
3
2



N’est pas 3-coloriable!

Exemple 2: Le circuit Hamiltonien

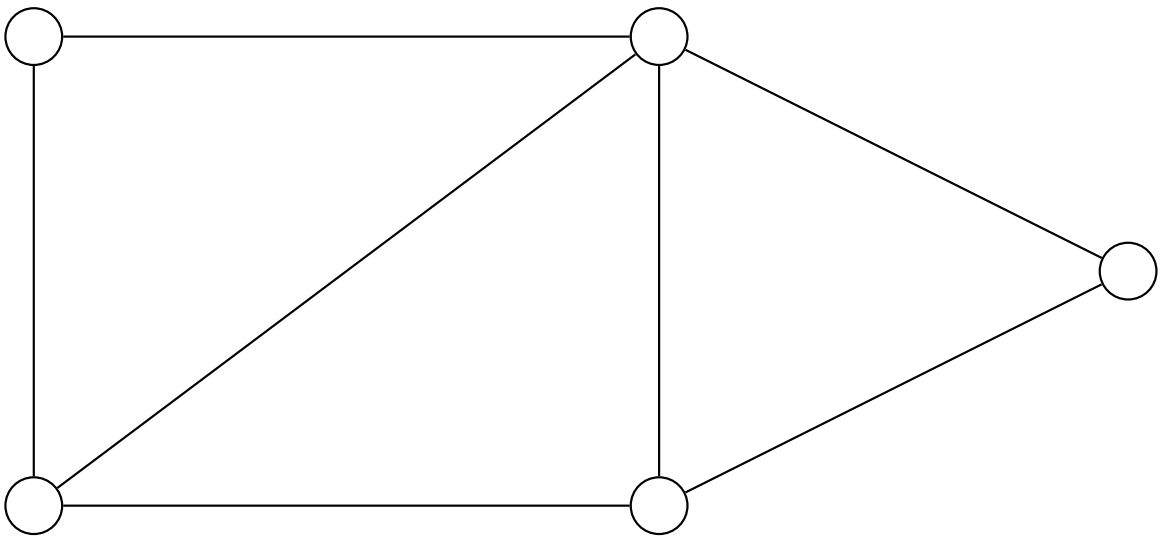
Donnée: $G = (S, A)$ un graphe, n un entier Sortie: oui, si le graphe contient un circuit hamiltonien i.e. on peut trouver:

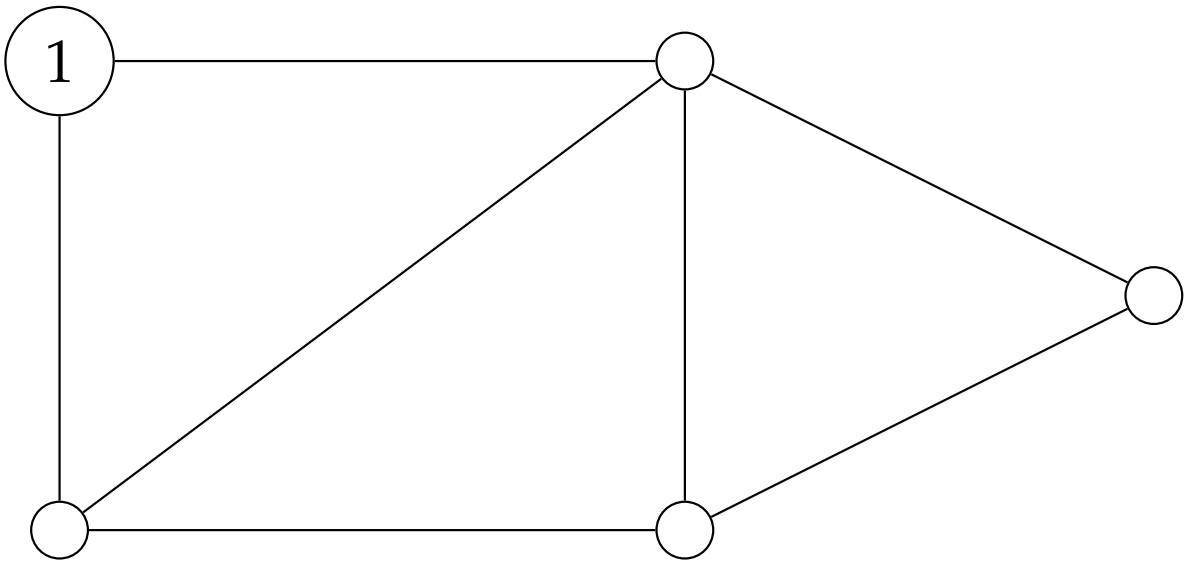
une application injective *sommet* de $\{1, 2, ..., n\}$ dans S telle que:

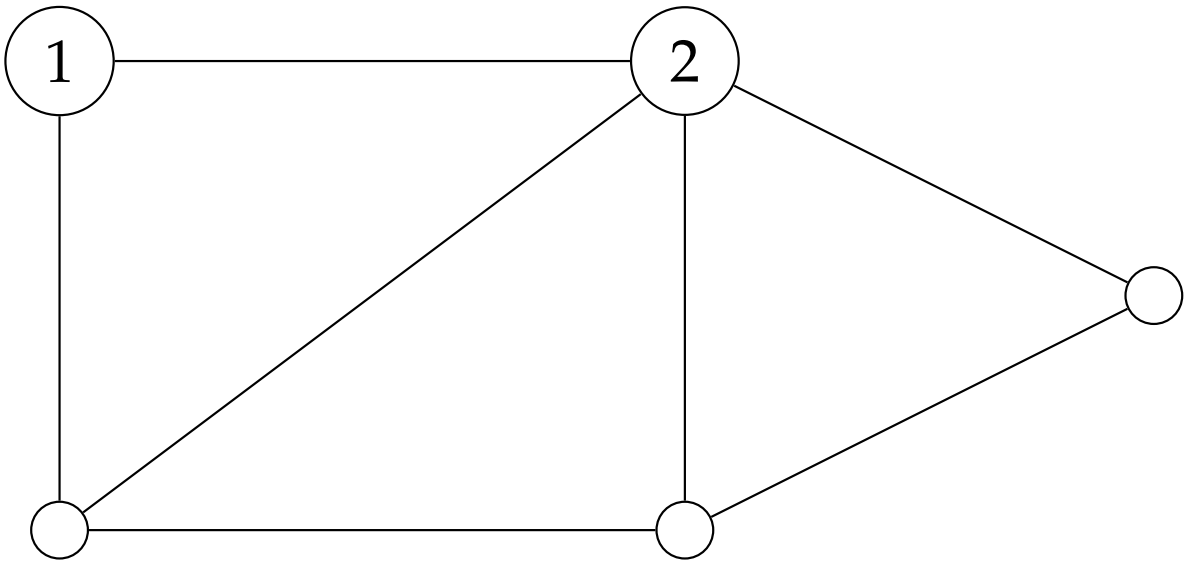
elle soit injective (on ne passe pas deux fois par le même sommet)

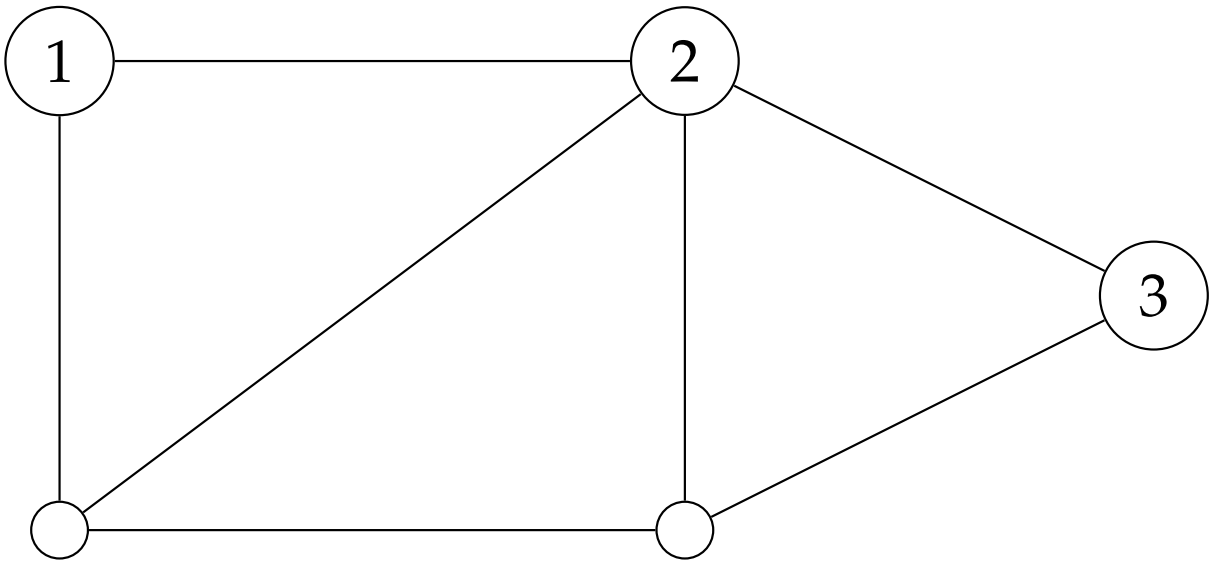
$(sommet(i), sommet(i+1)) \in A$ pour $1 <= i < n$

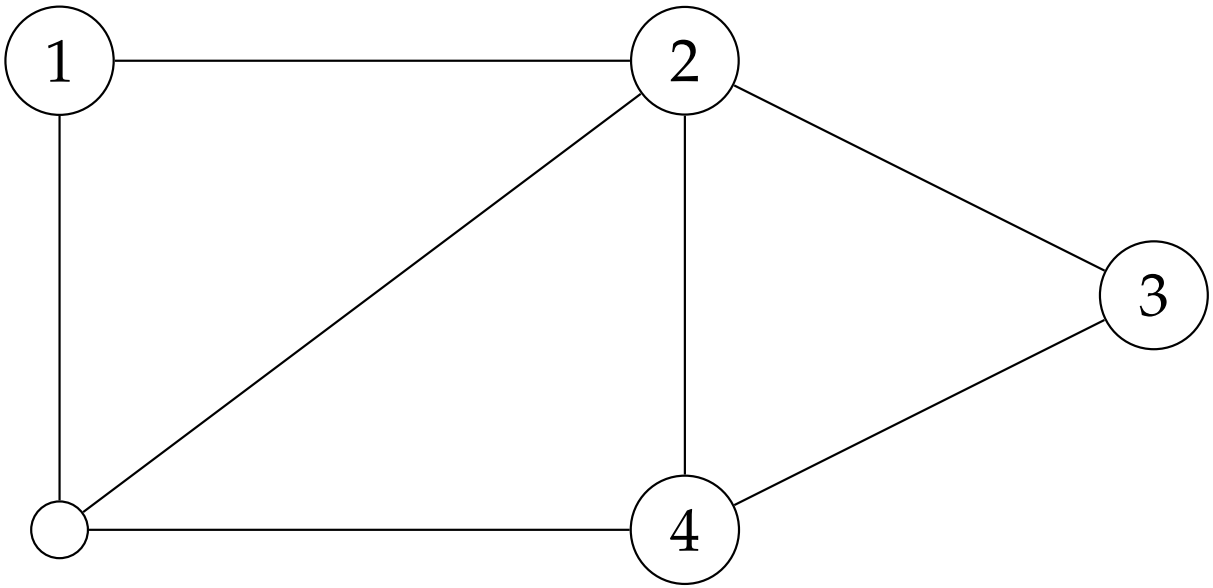
$(sommet(n), sommet(1)) \in A$

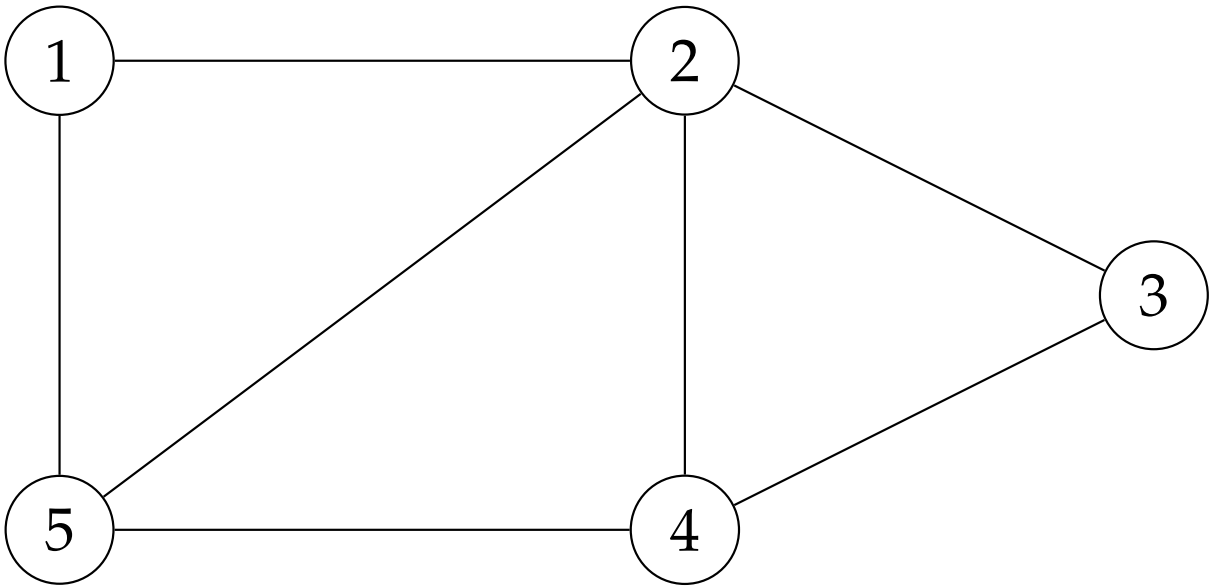


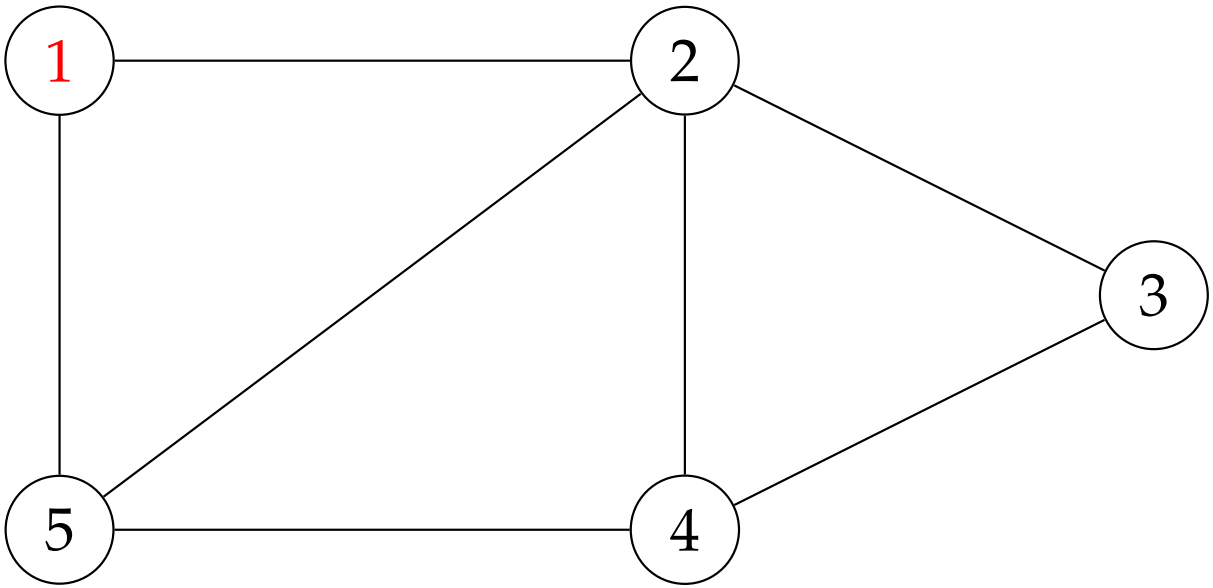

1


1
2

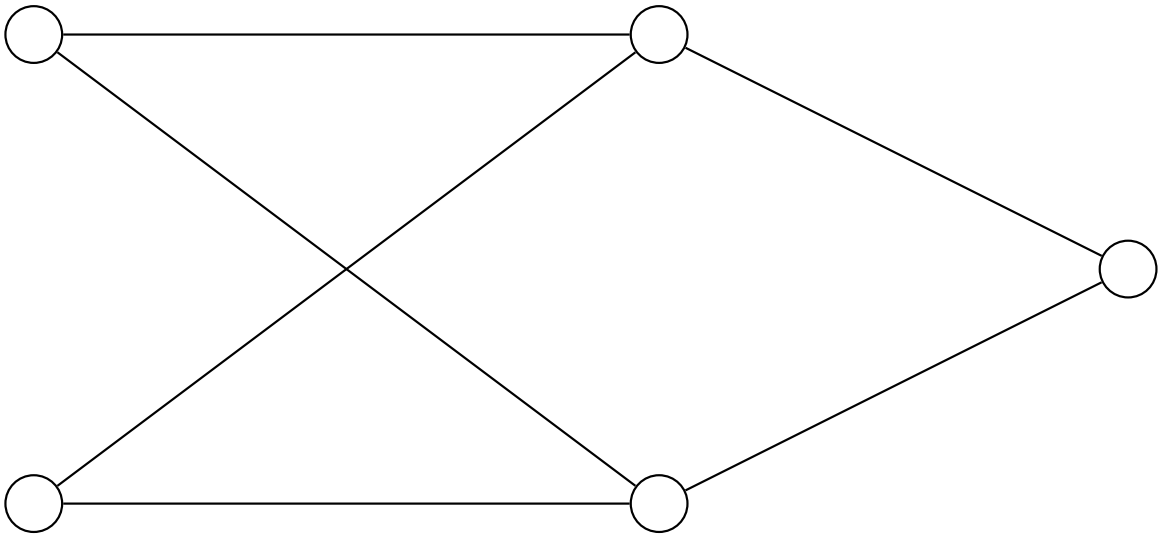

1
2
3


1
2
3
4


1
2
3
5
4



Il y a un circuit hamiltonien!



N’a pas de circuit hamiltonien.

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i

i.e. on peut trouver

$J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Exemple 3: Atteindre la cible

Instance:
$6, 2, 8, 6, 9, 12$

cible: 25

Exemple 3: Atteindre la cible

Instance:
$6, 2, 8, 6, 9, 12$

cible: 25=6+2+8+9

Exemple 3: Atteindre la cible

Instance:
$6, 2, 8, 6, 9, 12$

cible: 13

Exemple 3: Atteindre la cible

Instance:
$6, 2, 8, 6, 9, 12$

cible: 13

N'a pas de solution!

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Φ est satisfiable: $v(x_1) = Vrai, v(x_3) = Faux, ...$

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Φ est satisfiable: $v(x_1) = Vrai, v(x_3) = Faux, ...$

si $\Phi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_3)$

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Φ est satisfiable: $v(x_1) = Vrai, v(x_3) = Faux, ...$

si $\Phi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_3)$

Φ n'est pas satisfiable (Comment le prouver?)

QUEL EST LE LIEN?

Pour chacune des propriétés précédentes, on cherche si il existe une solution qui vérifie une certaine contrainte.

.les ”candidats-solutions” sont des objets pas trop ”gros”: un coloriage, une valuation..

.vérifier si une solution vérifie la contrainte est ”facile”

C’est la spécificité des *NP* ...

Remarque: ça ne nous donne pas pour autant d’algo efficace ... Enumérer les candidats solutions nous amène à une solution exponentielle! Par exemple il y a plus de 3-coloriages d’un graphe de 200 sommets que d’atomes dans l’univers.

La définition de NP via les certificats

Définition

L est dit NP si il existe un polynôme Q, et un algorithme polynomial à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$$

A est appelé algorithme de vérification,
c est appelé certificat (ou preuve, ou témoin..).

La définition de NP via les certificats

Définition

L est dit NP si il existe un polynôme Q, et un algorithme polynomial à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$$

A est appelé algorithme de vérification,
c est appelé certificat (ou preuve, ou témoin..).

$|c|$ représente la taille de c.

La définition de *NP* via les certificats

Définition

L est dit NP si il existe un polynôme Q, et un algorithme polynomial à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$$

A est appelé algorithme de vérification,
c est appelé certificat (ou preuve, ou témoin..).

$|c|$ représente la taille de c.
$|c| \leq Q(|u|)$: la taille des certificats est bornée polynomialement par rapport à la talle de l'entrée.

La définition de NP via les certificats

Définition

L est dit NP si il existe un polynôme Q, et un algorithme polynomial à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$$

A est appelé algorithme de vérification,
c est appelé certificat (ou preuve, ou témoin..).

$|c|$ représente la taille de c.
$|c| \leq Q(|u|)$: la taille des certificats est bornée polynomialement par rapport à la talle de l'entrée.
On dit que A vérifie L en temps polynomial.

L'intuition?

On peut par exemple voir c comme une preuve, A comme un algorithme qui vérifie la preuve; vous pouvez être capable de vérifier facilement la preuve courte qu'un gentil génie, (un prof, par exemple:-)) vous donne mais cela n'implique pas forcément qu'elle soit facile à trouver...

L'intuition?

On peut par exemple voir c comme une preuve, A comme un algorithme qui vérifie la preuve; vous pouvez être capable de vérifier facilement la preuve courte qu'un gentil génie, (un prof, par exemple:-)) vous donne mais cela n'implique pas forcément qu'elle soit facile à trouver...

Une propriété NP sera donc une propriété pour laquelle les instances positives ont une preuve "courte" et "facile" à vérifier.

Comment montrer qu'une propriété est NP?

Pour montrer qu'une propriété est NP, il faut:

- définir la notion de certificat et montrer que la taille d'un certificat est bornée polynomialement par la taille du problème

Comment montrer qu'une propriété est NP?

Pour montrer qu'une propriété est NP, il faut:

- définir la notion de certificat et montrer que la taille d'un certificat est bornée polynomialement par la taille du problème

- définir l'algorithme de Vérification et montrer qu'il est polynomial (et correct...)

Etre 3-coloriable: les certificats

Etre 3-coloriable: les certificats

Un certificat pour G est juste un coloriage des noeuds.

Etre 3-coloriable: les certificats

Un certificat pour G est juste un coloriage des noeuds.
On peut par exemple le représenter par un tableau de couleurs indexé par les sommets.
On a donc :

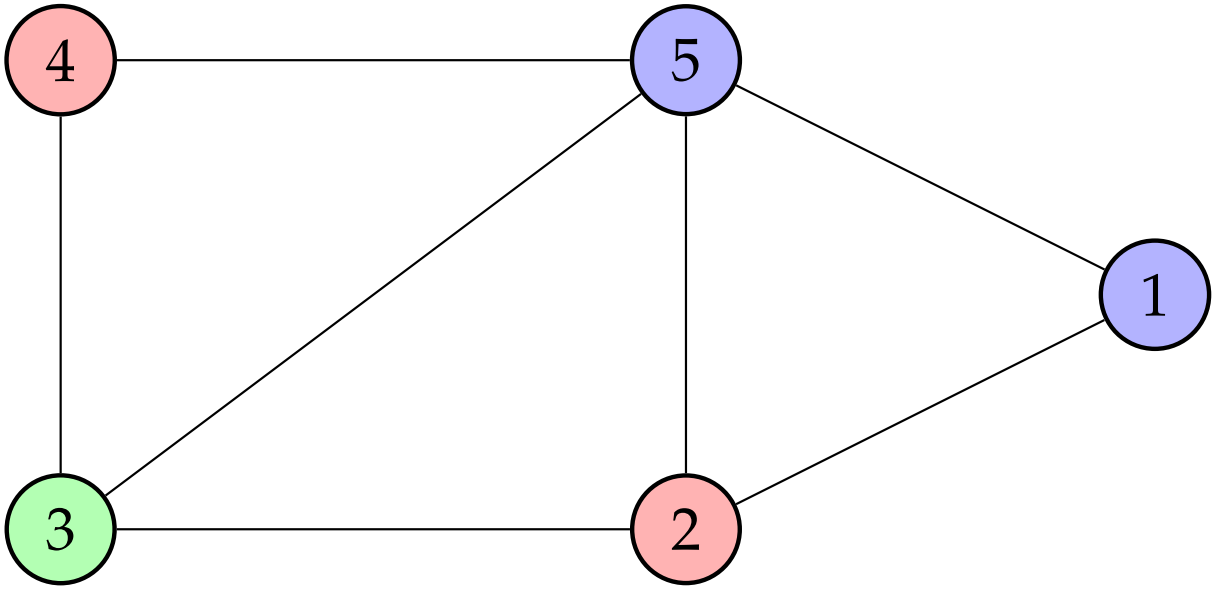
ETRE 3-COLORIABLE: LES CERTIFICATS

Un certificat pour G est juste un coloriage des noeuds.
On peut par exemple le représenter par un tableau de couleurs indexé par les sommets.
On a donc :

$$\text{taille du certificat} <= \text{taille du graphe}$$

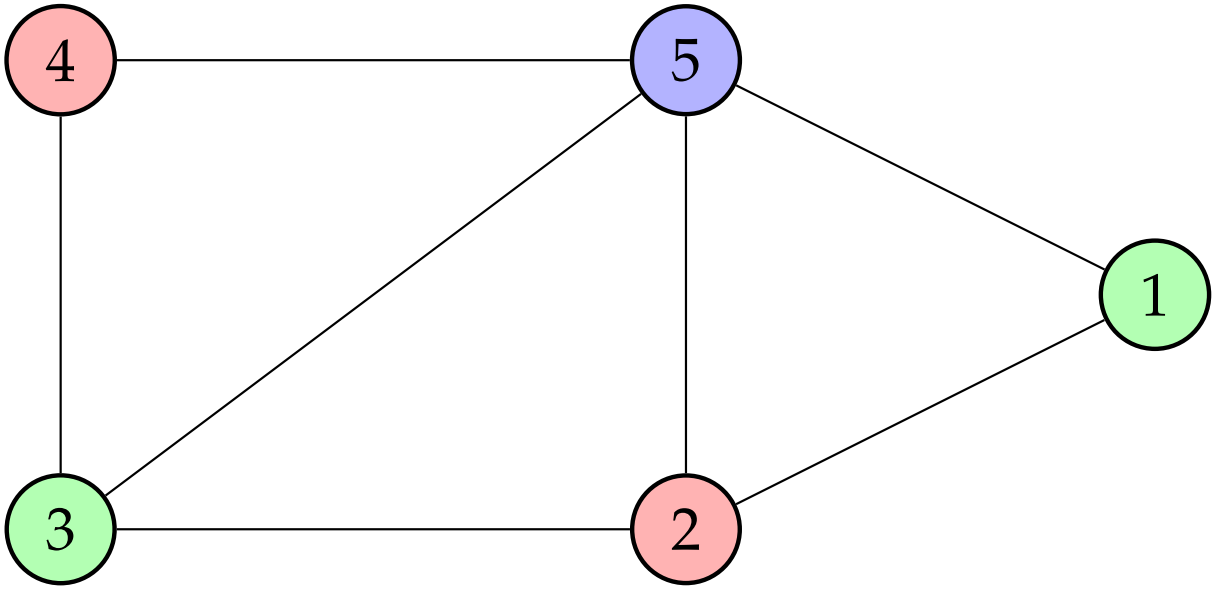
(on suppose que la taille d'un graphe est au moins le nombre de sommets plus le nombre d'arcs)
La taille d'un certificat est alors bien linéaire, donc polynomialement bornée, par rapport à celle de la donnée.

Etre 3-coloriable: l'algorithme de vérification



Doit retourner Faux!

Etre 3-coloriable: l'algorithme de vérification



Doit retourner Vrai!

ETRE 3-COLORIABLE: L'ALGORITHME DE VÉRIFICATION

Un certificat est valide Ssi aucun arc ne relie deux noeuds de même couleur: le vérifier est bien polynomial:

```
boolean A(col, G){
  Pour chaque arc (s,d) de G
          si col(s)=col(d) retourner Faux;
  retourner Vrai;
}
```

La complexité de l'algorithme est de l'ordre de $card(A)$ donc bien polynomiale.
'Etre 3-coloriable' est donc bien une propriété NP.

Le circuit Hamiltonien: les certificats

Donnée: $G = (S, A)$ un graphe, n un entier, $n <= card(S)$

Le circuit Hamiltonien: les certificats

Donnée: $G = (S, A)$ un graphe, n un entier, $n <= card(S)$

Certificat: une suite de n sommets, soit par exemple un tableau de n sommets.

Le circuit Hamiltonien: les certificats

Donnée: $G = (S, A)$ un graphe, n un entier, $n <= card(S)$

Certificat: une suite de n sommets, soit par exemple un tableau de n sommets.
Donc la taille d'un certificat est au plus n (ou $n * \log(card(S))$, si on prend en compte la taille du codage d'un sommet) , donc inférieure à la taille du graphe:
$|c| \leq |u|$, soit $|c| \leq Q(|u|)$ avec $Q(x) = x$

Vérification: if faut vérifier que c'est bien un circuit : tous les sommets sont différents, un sommet et son suivant sont bien reliés par un arc.

Le circuit Hamiltonien: la vérification

```
//cert: tableau de n sommets
A(cert,G){
boolean dejapasse=new passe[nbsommets];
//pour vérifier on ne passe pas 2 fois par le meme
pour i de 1 à n
  si dejapassé[cert[i]] retourner Faux;
   //on passe deux fois par  ce sommet
  si (cert[i],cert[i+1]) n'est pas un arc de G
   retourner Faux;
  dejapasse[cert(i)]=true;
fin pour;
  si (cert[n],cert[1]) n'est pas un arc)
  alors retourner Faux;
  sinon retourner Vrai;
```

Le circuit Hamiltonien

La complexité de l'algorithme est de l'ordre de n donc bien polynomiale.
Donc, la propriété est bien NP!

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Certificat:

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Certificat: $J \subset \{1, ..., n\}$

On peut le représenter par

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Certificat: $J \subset \{1, ..., n\}$

On peut le représenter par un tableau de n booléens: la taille d'un certificat est donc

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Certificat: $J \subset \{1, ..., n\}$

On peut le représenter par un tableau de n booléens: la taille d'un certificat est donc n soit inférieure à la taille du problème.

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Certificat: $J \subset \{1, ..., n\}$

On peut le représenter par un tableau de n booléens: la taille d'un certificat est donc n soit inférieure à la taille du problème.

A vérifier: $c = \sum_{i \in J} x_i$:

Exemple 3: Atteindre la cible

A vérifier: $c = \sum_{i \in J} x_i$:

```
boolean A(cert, x_1,...,x_n,s){
    int s==0;
    Pour i de 1 à n
          si  cert(i) alors s=s+x_i
    fin pour;
    retourner (s==c);
 }
```

Algo en $O(n)$

La propriété est bien NP!

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: Φ une expression booléenne avec n variables booléennes, $x_1, \ldots, x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Certificat:

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Certificat: v , par exemple un tableau de n booléens

Algo de vérification:

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Certificat: v , par exemple un tableau de n booléens

Algo de vérification: évaluer $v(\Phi)$

Sat est bien une propriété NP.

Récapitulatif: Comment et pourquoi montrer qu'une propriété est NP?

Pour montrer qu'une propriété est NP, il faut:

- définir la notion de certificat et montrer que la taille d'un certificat est bornée polynomialement par la taille du problème

- définir l'algorithme de Vérification et montrer qu'il est polynomial (et correct...). On montre qu'une propriété est *NP* pour situer la complexité du problème, rarement pour résoudre le problème: l'objectif n'est donc pas en général d'optimiser l'algorithme.

La définition via le non-déterminisme

NP=Non-Déterministe Polynomial

Définition

Alternative: une propriété Pr est NP si il existe un algorithme non déterministe polynomial qui décide Pr.

Remarque: NP=Non-Déterministe Polynomial et non pas Non Polynomial!

Algorithmes non déterministes

Un algorithme non-déterministe peut être vu comme un algorithme avec des instructions de type "choix(i,1..n)": on choisit aléatoirement un entier dans l'intervalle $[1..n]$. (On peut se restreindre à $n = 2$.)
On peut prendre comme modèle de calcul non-déterministe, les Machines de Turing non déterministes: on étudiera ce modèle dans les derniers cours.

Algorithmes non déterministes polynomiaux

Un algorithme non-déterministe A est dit polynomial si il existe un polynôme Q tel que pour toute entrée u, **tous** les calculs de A sur u ont une longueur d'exécution bornée par $Q(|u|)$.

Algos non déterministes à valeurs booléennes

Soit A un algorithme non déterministe à valeurs booléennes et dont tous les calculs s'arrêtent; il décide la propriété Pr suivante: "u vérifie Pr Ssi il existe un calcul de A sur u qui retourne Vrai." (penser à un automate non déterministe: un mot est accepté si et seulement si il existe au moins un chemin acceptant.)

Algos non déterministes à valeurs booléennes : exemple

```
Cherchercible(x_1,....x_n, c){
s=0;
Pour  i in 1..n
 Choisir(onleprend?,1..2);
  Si (onleprend?==1) s=s+x_i;
 finPour;
 retourner (s==c);}
```

La définition via le non-déterminisme versus La définition via les certificats

Soit un algorithme non-déterministe polynomial pour vérifier P. Comment définir la notion de certificat?

La définition via le non-déterminisme versus La définition via les certificats

Soit un algorithme non-déterministe polynomial pour vérifier P. Comment définir la notion de certificat?

Un certificat correspond à une suite de choix dans l'algorithme. Il est bien de taille bornée polynomialement car l'algorithme non-déterministe est polynomial.

La définition via le non-déterminisme versus La définition via les certificats

Soit un algorithme non-déterministe polynomial pour vérifier P. Comment définir la notion de certificat?

Un certificat correspond à une suite de choix dans l'algorithme. Il est bien de taille bornée polynomialement car l'algorithme non-déterministe est polynomial.

L'algorithme de vérification

La définition via le non-déterminisme versus La définition via les certificats

Soit un algorithme non-déterministe polynomial pour vérifier P. Comment définir la notion de certificat?

Un certificat correspond à une suite de choix dans l'algorithme. Il est bien de taille bornée polynomialement car l'algorithme non-déterministe est polynomial.

L'algorithme de vérification consiste à vérifier que l'exécution de l'algorithme non déterministe correspondant à la suite de choix donnée par le certificat retourne Vrai. Il est bien déterministe polynomial.

La définition via le non-déterminisme versus La définition via les certificats

Supposons qu'on ait prouvé que la propriété est NP en utilisant la notion de certificat. Comment définir un algorithme non-déterministe polynomial pour le problème?

L'algorithme non-déterministe consiste à

La définition via le non-déterminisme versus La définition via les certificats

Supposons qu'on ait prouvé que la propriété est NP en utilisant la notion de certificat. Comment définir un algorithme non-déterministe polynomial pour le problème?

L'algorithme non-déterministe consiste à

- d'abord générer aléatoirement un certificat -la partie non déterministe- ; elle est bien polynomiale car le certificat est d longueur polynomialement bornée.

La définition via le non-déterminisme versus La définition via les certificats

Supposons qu'on ait prouvé que la propriété est *NP* en utilisant la notion de certificat. Comment définir un algorithme non-déterministe polynomial pour le problème?

L'algorithme non-déterministe consiste à

- d'abord générer aléatoirement un certificat -la partie non déterministe- ; elle est bien polynomiale car le certificat est d longueur polynomialement bornée.
- à vérifier le certificat en utilisant l'algorithme -déterministe polynomial- de vérification.

L'algorithme est bien non-déterministe polynomial.

NP ET LES AUTRES: NP PAR RAPPORT À P?

Bien sûr, P est inclus dans NP: toute propriété P est une propriété NP;

NP ET LES AUTRES: NP PAR RAPPORT À P?

Bien sûr, P est inclus dans NP: toute propriété P est une propriété NP;

L'algorithme de vérification est l'algorithme de décision et n'a pas besoin de certificat: on peut prendre pour certificat le mot vide.

La conjecture $NP \neq P$?

On conjecture -en général- que $P \neq NP$, mais personne n'a su le prouver!

La conjecture $NP \neq P$?

On conjecture -en général- que $P \neq NP$, mais personne n'a su le prouver!

Aucune propriété NP n'a été prouvée à ce jour ne pas être P.

La conjecture $NP \neq P$?

On conjecture -en général- que $P \neq NP$, mais personne n'a su le prouver!

Aucune propriété NP n'a été prouvée à ce jour ne pas être P.

D'un autre côté, pour beaucoup de propriétés NP, aucun algorithme polynomial n'a été trouvé (ou prouvé exister) malgré les efforts de milliers de personnes!

La conjecture $NP \neq P$?

On conjecture -en général- que $P \neq NP$, mais personne n'a su le prouver!

Aucune propriété NP n'a été prouvée à ce jour ne pas être P.

D'un autre côté, pour beaucoup de propriétés NP, aucun algorithme polynomial n'a été trouvé (ou prouvé exister) malgré les efforts de milliers de personnes!

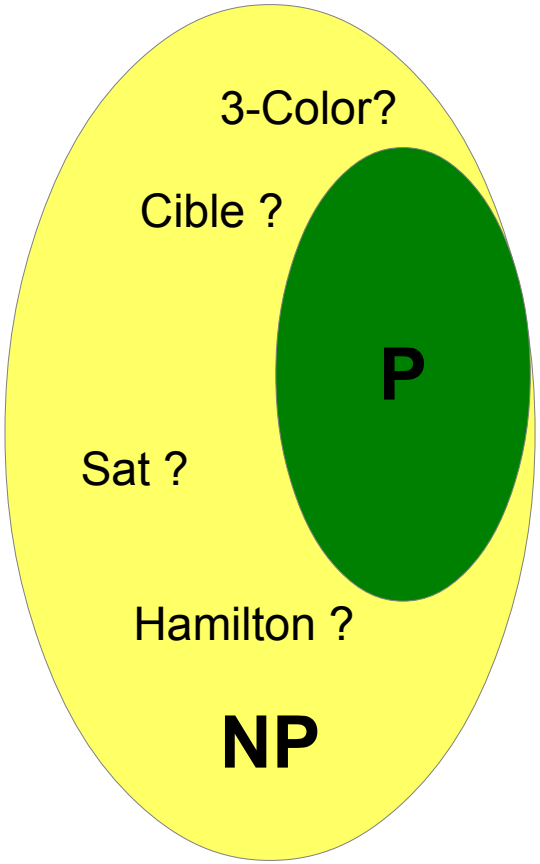
On verra dans le prochain cours qu'il existe des propriétés NP telles que si on trouvait un algorithme polynomial pour l'une d'entre elles, il y aurait un algorithme polynomial pour n'importe quelle propriété NP.

La conjecture $NP \neq P$?

La conjecture a été émise par S. Cook en 1971 et le ”Clay Mathematics Institute” offre 1 million de dollars à celui qui trouve la réponse à la question!

La conjecture $NP \neq P$?

La conjecture $NP \neq P$?



NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est inclus dans *EXPTIME*:

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est inclus dans *EXPTIME*: l'algorithme énumère et teste tous les certificats possibles.

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est inclus dans *EXPTIME*: l'algorithme énumère et teste tous les certificats possibles.

```
AUneSolution(Instance u)
pour tout certificat c de taille <= Q(|u|)
  si A(c,u) alors retourne Vrai;
retourne Faux
```

L'algorithme est bien exponentiel en temps.

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est inclus dans *EXPTIME*: l'algorithme énumère et teste tous les certificats possibles.

```
AUneSolution(Instance u)
pour tout certificat c de taille <= Q(|u|)
  si A(c,u) alors retourne Vrai;
retourne Faux
```

L'algorithme est bien exponentiel en temps.

Q? en mémoire?

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

NP est inclus dans *EXPTIME*: l'algorithme énumère et teste tous les certificats possibles.

```
AUneSolution(Instance u)
pour tout certificat c de taille <= Q(|u|)
  si A(c,u) alors retourne Vrai;
retourne Faux
```

L'algorithme est bien exponentiel en temps.

Q? en mémoire? Il est polynomial

NP ET LES AUTRES: *NP* PAR RAPPORT À *EXPTIME*

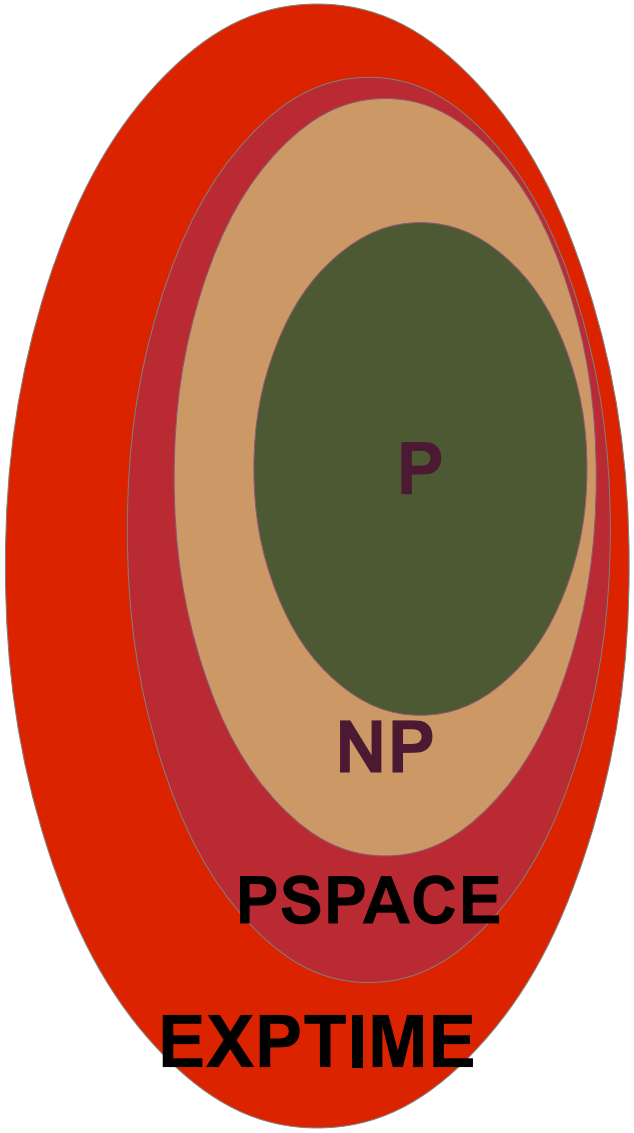
NP est inclus dans *EXPTIME*: l'algorithme énumère et teste tous les certificats possibles.

```
AUneSolution(Instance u)
pour tout certificat c de taille <= Q(|u|)
  si A(c,u) alors retourne Vrai;
retourne Faux
```

L'algorithme est bien exponentiel en temps.

Q? en mémoire? Il est polynomial :
NP est inclus dans *PSPACE*.

La Hiérarchie



NP ET LES PROPRIÉTÉS DE CLÔTURE BOOLÉENNE...

La classe *NP* est close par union, intersection

NP ET LES PROPRIÉTÉS DE CLÔTURE BOOLÉENNE...

La classe *NP* est close par union, intersection

Par contre, on ne sait pas si *NP* est close par complémentaire:

NP ET LES PROPRIÉTÉS DE CLÔTURE BOOLÉENNE...

La classe *NP* est close par union, intersection

Par contre, on ne sait pas si *NP* est close par complémentaire:

Disposer de la notion de certificat et d'algorithme de vérification pour une propriété Q, n'implique à priori pas que *non* Q soit *NP*;
Exemple: comment vérifier qu'il n'existe pas de 3−coloriage?

NP et co-*NP*

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

NP et co-*NP*

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

Bien sûr, P

NP et co-*NP*

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

Bien sûr, $P \subset$ co-*NP*

NP et co-NP

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

Bien sûr, $P \subset$ co-NP

On conjecture aussi que $NP \neq co - NP$ mais, là encore ce n'est qu'une conjecture!

Bien sûr, si $NP = P$, on aurait

NP et co-NP

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

Bien sûr, $P \subset$ co-NP

On conjecture aussi que $NP \neq co - NP$ mais, là encore ce n'est qu'une conjecture!

Bien sûr, si $NP = P$, on aurait NP=co-NP.

NP et co-NP

Définition

Une propriété Q telle que non Q soit NP est dite co-NP.

Bien sûr, $P \subset$ co-NP

On conjecture aussi que $NP \neq co - NP$ mais, là encore ce n'est qu'une conjecture!

Bien sûr, si $NP = P$, on aurait NP=co-NP.

Mais on pourrait avoir $NP = co - NP$ sans que NP soit égal à P!

NP et les propriétés de clôture, suite

Exercice

La classe NP est close par concaténation et étoile de Kleene

Le bilan

- Une propriété *NP* est une propriété pour la quelle trouver une ”solution” (une preuve ..) est peut-être difficile, mais vérifier une solution (une preuve...) est facile.

Le bilan

- Une propriété *NP* est une propriété pour la quelle trouver une ”solution” (une preuve ..) est peut-être difficile, mais vérifier une solution (une preuve...) est facile.
- P: easy to find

Le bilan

- Une propriété *NP* est une propriété pour la quelle trouver une ”solution” (une preuve ..) est peut-être difficile, mais vérifier une solution (une preuve...) est facile.
- P: easy to find
- NP: easy to check
- Attention: Montrer qu'une propriété est *NP* n'est pas montrer qu'elle n'est pas *P*!!!

Le bilan

- Une propriété *NP* est une propriété pour la quelle trouver une "solution" (une preuve ..) est peut-être difficile, mais vérifier une solution (une preuve...) est facile.
- P: easy to find
- NP: easy to check
- Attention: Montrer qu'une propriété est *NP* n'est pas montrer qu'elle n'est pas *P*!!!
- Montrer qu'une propriété est *NP* n'est en général qu'une étape... On cherche en général ensuite à montrer qu'elle est aussi *NP*-dure: voir le prochain cours.