Objectifs

- Gagner en aisance en programmation
- Savoir travailler en projet $\sim$ long
- Savoir utiliser des outils dédiés au projet
- Acquérir une certaine rigueur
- Voir certains aspects à l'interface maths/info

Présentation du projet

Python 3

Les environnements suivants :

- git (github, gitlab)
- 1. éditeur de texte + terminal ou
  2. IDE (Spyder3, PyCharm, ...)
  3. Google Colab / Cocalc ...

Technologies utilisées et attentes

Python 3

Les environnements suivants :

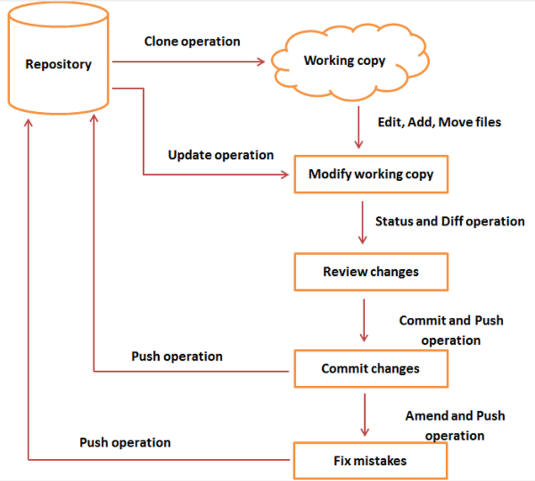
- git (github, gitlab)
- 1. éditeur de texte + terminal ou
  2. IDE (Spyder3, PyCharm, ...)
  3. Google Colab / Cocalc ...

Insister :

- code clair et lisible
  - noms de variables/fonctions sensés, fonctions (intermédiaires) courtes, commentaires, ...
- commentaires
  - paramètres et sorties de fonctions/méthodes
  - explication de code non-intuitif (imaginer que qqun qui n'a pas du tout participé au projet doive s'approprier le code)
- architecture et compartimentation

sur le github/lab

```
Projet-LDD2-S4
├─ worksheet.py
├─ modules
│   ├─ concept1.py
│   │     ⋮
│   └─ conceptn.py
└─ tests
    ├─ concept1_test.py
    │     ⋮
    └─ conceptn_test.py
```

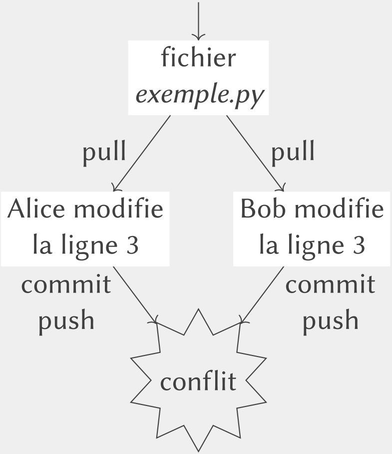


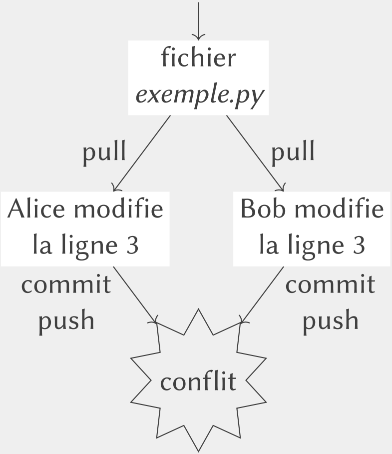
Créer un repo privé (par groupe) sur GitHub/GitLab. Inviter les autres membres du groupe.

```
git clone <url du repo git> # crée une version locale
                            # du repo
git pull # met à jour la version locale
git diff # affiche ce qui n'a pas encore été commité
git add <fichiers> # précise à git quels
                   # fichiers commiter
git commit -m "<message>" # commit les changements
git log # liste des commits
git amend  # change le dernier commit
git push origin master # applique les changements
                       # du commit
```

Git est plus complet que ça (voir branches).
Doc : `https://git-scm.com/docs`

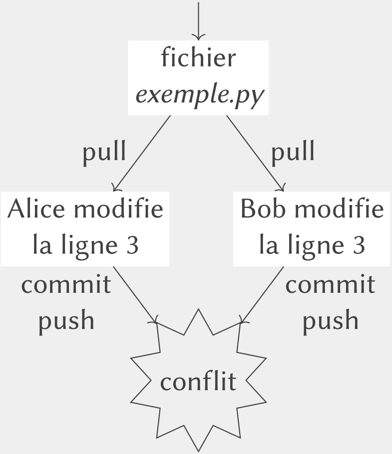

KEEP
CALM
AND
CRUSH
CONFLICTS


fichier
exemple.py
pull
pull
Alice modifie la ligne 3
Bob modifie la ligne 3
commit push
commit push
conflit



⋆ Un *conflit* intervient lorsque deux personnes ont modifié le même morceau de code "en même temps" $\Longrightarrow$ Git ne parvient pas à déterminer quelle version garder.

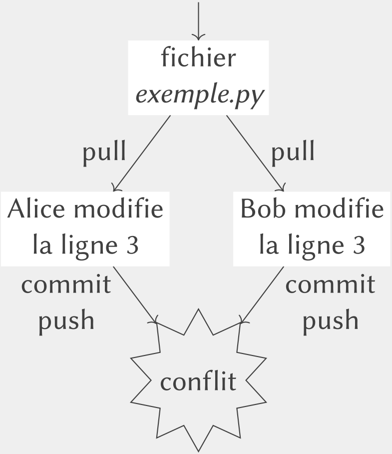
Conflits dans Git



⋆ Un *conflit* intervient lorsque deux personnes ont modifié le même morceau de code "en même temps" $\Longrightarrow$ Git ne parvient pas à déterminer quelle version garder.

⋆ Le message :

```
CONFLICT (content): merge conflict in example.py
```



⋆ Un *conflit* intervient lorsque deux personnes ont modifié le même morceau de code "en même temps" $\Longrightarrow$ Git ne parvient pas à déterminer quelle version garder.

⋆ Le message :

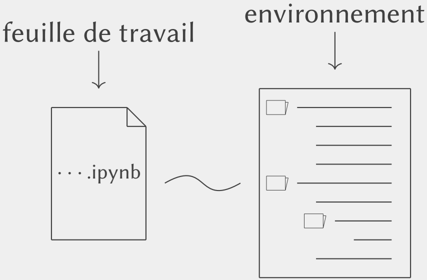
```
CONFLICT (content): merge conflict in example.py
```

⋆ les lignes conflictuelles dans le fichier :

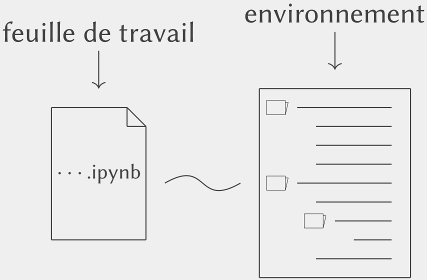
```
<<<<<<< HEAD
ligne(s) de la branche master
=======
ligne(s) de la branche conflictuelle
>>>>>>> branche_conflictuelle
```

Une fois les conflits résolus, il faut re-commit.

Google Colab (à utiliser avec parcimonie)

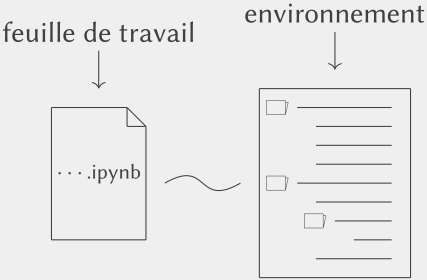


Google Colab (à utiliser avec parcimonie)



On peut partager sa feuille de travail et l'environnement courant avec d'autres (et travailler dessus en parallèle).

Google Colab (à utiliser avec parcimonie)



On peut partager sa feuille de travail et l'environnement courant avec d'autres (et travailler dessus en parallèle).

! Seul le fichier .ipynb est sauvegardé !
L'environnement est "renouvelé" toutes les 1h30/12h
⇒ sauvegarder ses fichiers ailleurs ! (avec Git par exemple)

Solution : Utiliser Git dans Colab

- Importer (cloner) le projet en début de séance

```
uname = "<nom d'utilisateur github>"
! git config --global user.email '$uname@gmail.com'
! git config --global user.name '$uname'

from getpass import getpass
password = getpass('Password:')
! git clone https://$uname:$password@github.com/<url du repo>
%reset_selective -f password
```

- Travailler dans Colab pendant la séance
- Pousser (push) les changements sur GitHub

```
! git add .
! git commit -m "<message du commit>"
! git push origin main
```

- **Logique** : permet de programmer

  ↓ peut être implémenté physiquement par

- **Circuits booléens**

  ↓ représentés par

- **Graphes** dirigés, ouverts et acycliques

Logique (propositionnelle)

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...
  - applications industrielles

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...
  - applications industrielles
    - vérification de logiciels embarqués (avions, certaines lignes de métro, ...)

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...
  - applications industrielles
    - vérification de logiciels embarqués (avions, certaines lignes de métro, ...)
    - vérification de compilateurs

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...
  - applications industrielles
    - vérification de logiciels embarqués (avions, certaines lignes de métro, ...)
    - vérification de compilateurs
    - ...

La logique en général

- domaine de recherche à part entière, à l'intersection entre maths et informatique
- existe plein de logiques
- permet d'exprimer problèmes, théorèmes et raisonnements mathématiques de manière très formelle $\Rightarrow$ "informatisation" des maths
  - on peut laisser vérifier un ordinateur qu'un raisonnement est vrai
  - on peut faire chercher à l'ordinateur un raisonnement mathématique qui permet de montrer P sachant Q
  - applications mathématiques
    - théorème du puissance 4
    - théorème des 4 couleurs
    - classification des pavages pentagonaux
    - ...
  - applications industrielles
    - vérification de logiciels embarqués (avions, certaines lignes de métro, ...)
    - vérification de compilateurs
    - ...
- fournit un bon modèle pour interface software/hardware

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\perp$ et $\top$, ...)

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\perp$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \ldots$

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\bot$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \ldots$
- des connecteurs logiques $\sim, \&, |$

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\perp$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \ldots$
- des connecteurs logiques $\sim, \&, |$
- les formules propositionnelles (FP) :
  - 0 et 1 sont des FP

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\perp$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \ldots$
- des connecteurs logiques $\sim, \&, |$
- les formules propositionnelles (FP) :
  - 0 et 1 sont des FP
  - les variables propositionnelles x_i sont des FP

Rappels (?) : formules propositionnelles

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\bot$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \ldots$
- des connecteurs logiques $\sim$, $\&$, $|$
- les formules propositionnelles (FP) :
  - 0 et 1 sont des FP
  - les variables propositionnelles x_i sont des FP
  - si P et Q sont des FPs, alors :
    - $\sim P$ est une FP (nommée “non P”)
    - $P \& Q$ est une FP (nommée “P et Q”)
    - $P \mid Q$ est une FP (nommée “P ou Q”)

- 2 constantes 0 et 1 (ou bien Faux et Vrai, $\perp$ et $\top$, ...)
- des variables propositionnelles $x_0, x_1, \dots$
- des connecteurs logiques $\sim$, &, $|$
- les formules propositionnelles (FP) :
  - 0 et 1 sont des FP
  - les variables propositionnelles x_i sont des FP
  - si P et Q sont des FPs, alors :
    - $\sim P$ est une FP (nommée “non P”)
    - $P\&Q$ est une FP (nommée “P et Q”)
    - $P \mid Q$ est une FP (nommée “P ou Q”)

Exemples : Formules propositionnelles

1	x_0	$\sim x_0$	$1\&(\sim x_0)$	$(x_0\&x_1)\mid(\sim(x_1 \mid x_2))$

on note $\mathrm{Var}(P)$ les variables propositionnelles dans la formule P

Exemple

$$\mathrm{Var}\left((x_0 \& x_1)|(\sim (x_1 \mid x_2))\right) = \{x_0, x_1, x_2\}$$

on note $\mathrm{Var}(P)$ les variables propositionnelles dans la formule P

Exemple

$$\mathrm{Var}\left((x_0\&x_1)|(\sim (x_1 \mid x_2))\right) = \{x_0, x_1, x_2\}$$

Une *évaluation* ρ de P est une fonction $\rho : \mathrm{Var}(P) \to \{0, 1\}$. Elle induit une évaluation de P.

Exemple

Avec $\rho : \begin{cases} x_0 \mapsto 1 \\ x_1 \mapsto 1 \\ x_2 \mapsto 0 \end{cases}$ la formule précédente devient $(1\&1)|(\sim (1 \mid 0))$.

on note $\mathrm{Var}(P)$ les variables propositionnelles dans la formule P

Exemple

$$\mathrm{Var}\left((x_0 \& x_1) | (\sim (x_1 \mid x_2))\right) = \{x_0, x_1, x_2\}$$

Une *évaluation* ρ de P est une fonction $\rho : \mathrm{Var}(P) \to \{0, 1\}$. Elle induit une évaluation de P.

Exemple

Avec $\rho : \begin{cases} x_0 \mapsto 1 \\ x_1 \mapsto 1 \\ x_2 \mapsto 0 \end{cases}$ la formule précédente devient $(1 \& 1) | (\sim (1 \mid 0))$.

Avec n variables propositionnelles, on a 2^n évaluations possibles.

P	$\sim P$
0	1
1	0

Rappels (?) : évaluation et tables de vérité

P	$\sim P$
0	1
1	0

P	Q	$P\&Q$
0	0	0
0	1	0
1	0	0
1	1	1

Rappels (?) : évaluation et tables de vérité

P	$\sim P$
0	1
1	0

P	Q	$P\&Q$
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	$P \mid Q$
0	0	0
0	1	1
1	0	1
1	1	1

P	$\sim P$
0	1
1	0

P	Q	$P \& Q$
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	$P \mid Q$
0	0	0
0	1	1
1	0	1
1	1	1

Exemple

$$(1\&1)|(\sim (1 \mid 0)) \;=\; 1|(\sim 1) \;=\; 1|0 \;=\; 1$$

Rappels (?) : évaluation et tables de vérité

P	$\sim P$
0	1
1	0

P	Q	$P\&Q$
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	$P \mid Q$
0	0	0
0	1	1
1	0	1
1	1	1

Exemple

$(1\&1)|(\sim (1 \mid 0)) \; = \; 1|(\sim 1) \; = \; 1|0 \; = \; 1$

Peut-on faire plus que "et", "ou", "non" ?

Rappels (?) : évaluation et tables de vérité

P	$\sim P$
0	1
1	0

P	Q	$P\&Q$
0	0	0
0	1	0
1	0	0
1	1	1

P	Q	$P \mid Q$
0	0	0
0	1	1
1	0	1
1	1	1

Exemple

$(1\&1)|(\sim (1 \mid 0)) \;=\; 1|(\sim 1) \;=\; 1|0 \;=\; 1$

Peut-on faire plus que "et", "ou", "non" ?

Exemple

$(\sim P) \mid Q$ et l'implication logique $P \rightarrow Q$ ont la même évaluation quelle que soient les évaluations de P et de Q.

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

Aparté : Satisfiabilité

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué

Aparté : Satisfiabilité

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué
- trouver un algorithme "efficace" (complexité polynomiale) pour le résoudre $\Rightarrow$ 1 million de $

Aparté : Satisfiabilité

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué
- trouver un algorithme "efficace" (complexité polynomiale) pour le résoudre $\Rightarrow$ 1 million de $
- permet de résoudre énormément de problèmes (ex : Sudoku, chemin Hamiltonien dans un graphe, ...)

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué
- trouver un algorithme "efficace" (complexité polynomiale) pour le résoudre $\Rightarrow$ 1 million de $
- permet de résoudre énormément de problèmes (ex : Sudoku, chemin Hamiltonien dans un graphe, ...)
- applications en automatisation de preuves

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué
- trouver un algorithme "efficace" (complexité polynomiale) pour le résoudre $\Rightarrow$ 1 million de $
- permet de résoudre énormément de problèmes (ex : Sudoku, chemin Hamiltonien dans un graphe, ...)
- applications en automatisation de preuves
- grosse recherche sur le sujet

Satisfiablilité (SAT)

Etant donnée P une FP, trouver une évaluation de $\mathrm{Var}(P)$ qui évalue P à 1.

- problème compliqué
- trouver un algorithme "efficace" (complexité polynomiale) pour le résoudre $\Rightarrow$ 1 million de $
- permet de résoudre énormément de problèmes (ex : Sudoku, chemin Hamiltonien dans un graphe, ...)
- applications en automatisation de preuves
- grosse recherche sur le sujet
- brute-force : algorithme exponentiel (teste toutes les évaluations possibles)
  ex : Sudoku : 324 variables propositionnelles $\Rightarrow 2^{324} \approx 3,4.10^{97}$ évaluations

Equivalence de formules propositionnelles

Equivalence

Deux FP P et Q qui ont les mêmes variables sont équivalentes ($P \equiv Q$) si toute évaluation des variables donne le même résultat chez P et Q, i.e. si les tables de vérité de P et Q sont les mêmes.

Equivalence

Deux FP P et Q qui ont les mêmes variables sont équivalentes ($P \equiv Q$) si toute évaluation des variables donne le même résultat chez P et Q, i.e. si les tables de vérité de P et Q sont les mêmes.

Exemples

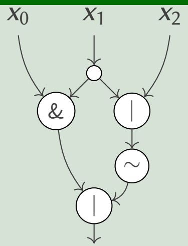
- $\sim (P\&Q) \equiv (\sim P) \mid (\sim Q)$
- $P\&(Q \mid R) \equiv (P\&Q) \mid (P\&R)$

Circuits booléens

Informellement : représentation d'une formule propositionnelle.

Exemple

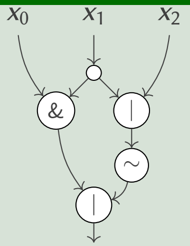
$(x_0 \& x_1) | (\sim (x_1 \mid x_2)) \rightsquigarrow$



Informellement : représentation d'une formule propositionnelle.

Exemple

$(x_0 \& x_1) | (\sim (x_1 \mid x_2)) \rightsquigarrow$

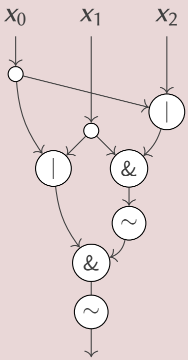


Les circuits booléens sont des graphes ouverts dirigés et acycliques.

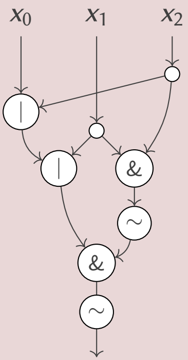
Circuit booléen

Exercice

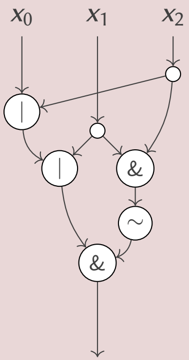
Quel circuit booléen obtient-on à partir de $((x_0 \mid x_2) \mid x_1)\&(\sim (x_1\&x_2))$?



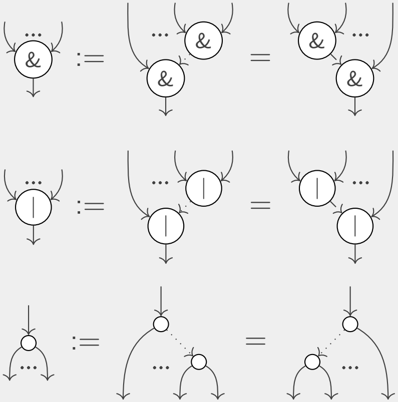
ou



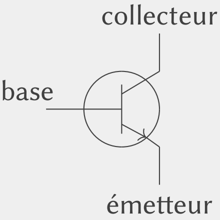
ou



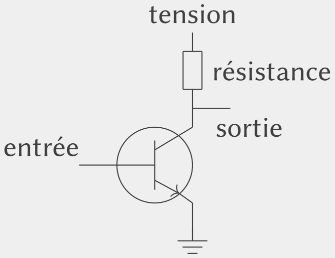
Puisque &, | et la copie sont tous associatifs, on se permet de représenter :



- Portes logiques implémentables grâce à des transistors :



Par exemple, la porte "non" ($\sim$) :



- Etude fine de la complexité des algorithmes

Graphes

Graphe

- Informellement : ensemble de noeuds (ou sommets) reliés par des arêtes

Graphe

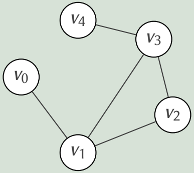
- Informellement : ensemble de noeuds (ou sommets) reliés par des arêtes
- Mathématiquement : $G = (V, E)$ avec V l'ensemble de sommets, $E \subseteq V \times V$ l'ensemble d'arêtes

Graphe

- Informellement : ensemble de noeuds (ou sommets) reliés par des arêtes
- Mathématiquement : $G = (V, E)$ avec V l'ensemble de sommets, $E \subseteq V \times V$ l'ensemble d'arêtes
  (dans un graphe non-orienté simple, au plus une arête entre deux sommets. Convention à choisir : $(u, v) \in E \implies (v, u) \notin E$ ou au contraire $(u, v) \in E \implies (v, u) \in E$)

- Informellement : ensemble de noeuds (ou sommets) reliés par des arêtes
- Mathématiquement : $G = (V, E)$ avec V l'ensemble de sommets, $E \subseteq V \times V$ l'ensemble d'arêtes
  (dans un graphe non-orienté simple, au plus une arête entre deux sommets. Convention à choisir : $(u, v) \in E \implies (v, u) \notin E$ ou au contraire $(u, v) \in E \implies (v, u) \in E$)

Exemple



représenté par

$$(\{v_0, v_1, v_2, v_3, v_4\}, \{(v_0, v_1), (v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\})$$

Exercice

Dessiner le graphe décrit par :

$$\Big(\{v_0, v_1, v_2, v_3, v_4, v_5\},\\ \{(v_0, v_3), (v_0, v_4), (v_0, v_5), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_3), (v_2, v_4), (v_2, v_5)\}\Big)$$

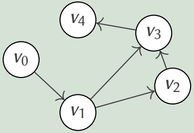
Dans un graphe non-dirigé, (u, v) et (v, u) sont interchangeables.

Dans un graphe non-dirigé, (u, v) et (v, u) sont interchangeables.
Pour avoir un graphe dirigé, il suffit de considérer l'arête (u, v) comme dirigée de u vers v.

Dans un graphe non-dirigé, (u, v) et (v, u) sont interchangeables.
Pour avoir un graphe dirigé, il suffit de considérer l'arête (u, v) comme dirigée de u vers v.

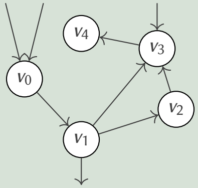
Exemple

$(\{v_0, v_1, v_2, v_3, v_4\}, \{(v_0, v_1), (v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\})$:



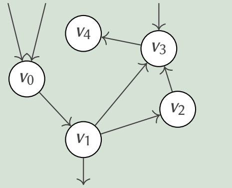
Graphe dirigé ouvert

Exemple



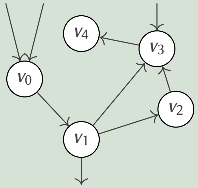
Graphe dirigé ouvert

Exemple



Mathématiquement, on rajoute deux listes (séquences) I et O qui contiennent les noeuds d'entrée (resp. de sortie)

Exemple

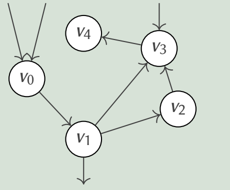


Mathématiquement, on rajoute deux listes (séquences) I et O qui contiennent les noeuds d'entrée (resp. de sortie)

Exemple

$G = (V, I, O, E)$ avec $I = [v_0, v_0, v_3]$ et $O = [v_1]$

Exemple



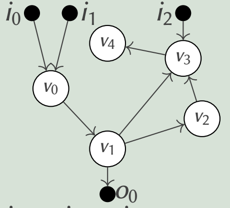
Mathématiquement, on rajoute deux listes (séquences) I et O qui contiennent les noeuds d'entrée (resp. de sortie)

Exemple

$G = (V, I, O, E)$ avec $I = [v_0, v_0, v_3]$ et $O = [v_1]$

Informatiquement, on peut faire exactement pareil, ou bien traiter les extrémités des "demi-arêtes" comme des noeuds à part entière, avec des contraintes (une seule connexion, entrante si c'est une sortie, sortante si c'est une entrée), et munis d'un ordre pour les entrées, et d'un autre pour les sorties.

Exemple



$i_0 \prec i_1 \prec i_2$

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.
Dans $G = (V, I, O, E)$, avec $u \in V$:

- $\deg^+(u) = \left|\{(x, u) \in E \mid x \in V\}\right| + \left|\{x \mid I[x] = u\}\right|$

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.
Dans $G = (V, I, O, E)$, avec $u \in V$:

- $\deg^+(u) = \big|\{(x, u) \in E \mid x \in V\}\big| + \big|\{x \mid I[x] = u\}\big|$
- $\deg^-(u) = \big|\{(u, x) \in E \mid x \in V\}\big| + \big|\{x \mid O[x] = u\}\big|$

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.
Dans $G = (V, I, O, E)$, avec $u \in V$:

- $\deg^+(u) = \left|\{(x, u) \in E \mid x \in V\}\right| + \left|\{x \mid I[x] = u\}\right|$
- $\deg^-(u) = \left|\{(u, x) \in E \mid x \in V\}\right| + \left|\{x \mid O[x] = u\}\right|$
- $\deg(u) = \deg^+(u) + \deg^-(u)$

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.
Dans $G = (V, I, O, E)$, avec $u \in V$:

- $\deg^+(u) = \left|\{(x, u) \in E \mid x \in V\}\right| + \left|\{x \mid I[x] = u\}\right|$
- $\deg^-(u) = \left|\{(u, x) \in E \mid x \in V\}\right| + \left|\{x \mid O[x] = u\}\right|$
- $\deg(u) = \deg^+(u) + \deg^-(u)$

Degrés maximal $\Delta(G)$ et minimal $\delta(G)$ d'un graphe G :

- $\Delta(G) = \max(\{\deg(u) \mid u \in V\})$
- $\delta(G) = \min(\{\deg(u) \mid u \in V\})$

Degrés des noeuds d'un graphe

Informellement, le degré entrant (resp. sortant) d'un noeud est le nombre d'arêtes qui arrivent sur le (resp. partent du) noeud.
Le degré d'un noeud est la somme de ses degrés entrant et sortant.
Dans $G = (V, I, O, E)$, avec $u \in V$:

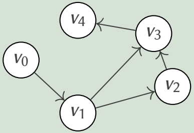
- $\mathsf{deg}^+(u) = \left|\{(x, u) \in E \mid x \in V\}\right| + \left|\{x \mid I[x] = u\}\right|$
- $\mathsf{deg}^-(u) = \left|\{(u, x) \in E \mid x \in V\}\right| + \left|\{x \mid O[x] = u\}\right|$
- $\mathsf{deg}(u) = \mathsf{deg}^+(u) + \mathsf{deg}^-(u)$

Degrés maximal $\Delta(G)$ et minimal $\delta(G)$ d'un graphe G :

- $\Delta(G) = \mathsf{max}(\{\mathsf{deg}(u) \mid u \in V\})$
- $\delta(G) = \mathsf{min}(\{\mathsf{deg}(u) \mid u \in V\})$

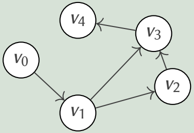
On peut également définir Δ^+, δ^+, Δ^- et δ^-.

Exemple

Dans 

, $[(v_0, v_1), (v_1, v_3), (v_3, v_4)]$ est un chemin de v_0 à v_4.

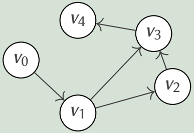
Exemple

Dans  , $[(v_0, v_1), (v_1, v_3), (v_3, v_4)]$ est un chemin de v_0 à v_4.

Soit $G = (V, I, O, E)$ un graphe dirigé (ouvert ou non). Un chemin de $u \in V$ à $v \in V$ est une séquence d'arêtes $[e_i]_{0 \leq i < n}$ telle que :

- $e_0[0] = u$ et $e_{n-1}[1] = v$
- $e_i[1] = e_{i+1}[0]$ (pour $0 \leq i < n - 1$)

Exemple

Dans  , $[(v_0, v_1), (v_1, v_3), (v_3, v_4)]$ est un chemin de v_0 à v_4.

Soit $G = (V, I, O, E)$ un graphe dirigé (ouvert ou non). Un chemin de $u \in V$ à $v \in V$ est une séquence d'arêtes $[e_i]_{0 \leq i < n}$ telle que :

- $e_0[0] = u$ et $e_{n-1}[1] = v$
- $e_i[1] = e_{i+1}[0]$ (pour $0 \leq i < n - 1$)

On appelle n la longueur du chemin.

Connexité

Informellement, deux noeuds dans un graphe sont connexes s'ils ne sont pas séparables.

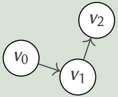
Connexité

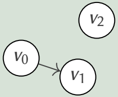
Informellement, deux noeuds dans un graphe sont connexes s'ils ne sont pas séparables. Formellement, u et v sont connexes s'il existe un chemin entre u et v dans le graphe non-dirigé induit.

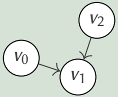
Informellement, deux noeuds dans un graphe sont connexes s'ils ne sont pas séparables. Formellement, u et v sont connexes s'il existe un chemin entre u et v dans le graphe non-dirigé induit.

Exemple

Dans les graphes suivants, v_0 et v_2 sont ...





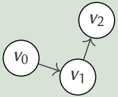


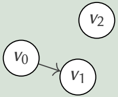
Connexité

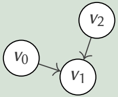
Informellement, deux noeuds dans un graphe sont connexes s'ils ne sont pas séparables. Formellement, u et v sont connexes s'il existe un chemin entre u et v dans le graphe non-dirigé induit.

Exemple

Dans les graphes suivants, v_0 et v_2 sont ...







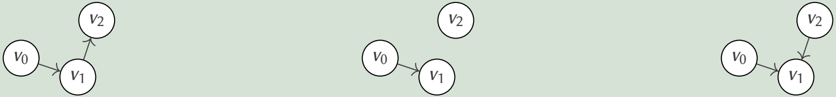
Si u_0 et u_1 sont connexes, alors $\text{connexe}(u_0, v) \iff \text{connexe}(u_1, v)$.

Connexité

Informellement, deux noeuds dans un graphe sont connexes s'ils ne sont pas séparables. Formellement, u et v sont connexes s'il existe un chemin entre u et v dans le graphe non-dirigé induit.

Exemple

Dans les graphes suivants, v_0 et v_2 sont ...



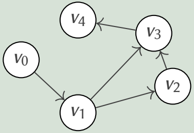
Si u_0 et u_1 sont connexes, alors $\text{connexe}(u_0, v) \iff \text{connexe}(u_1, v)$.
Partitionne les noeuds d'un graphe (en ce qu'on appelle des composantes connexes).

Cycles dans un graphe dirigé

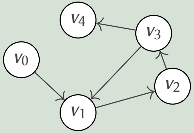
Un cycle est un chemin qui boucle sur lui même.

Un cycle est un chemin qui boucle sur lui même.

Exemple



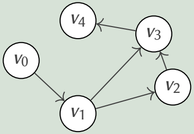
n'a pas de cycle, mais



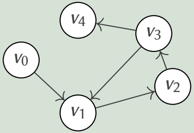
en a un : $[(v_1, v_2), (v_2, v_3), (v_3, v_1)]$

Un cycle est un chemin qui boucle sur lui même.

Exemple



n'a pas de cycle, mais



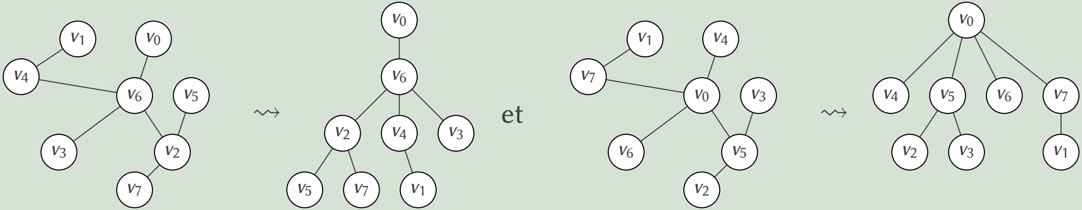
en a un : $[(v_1, v_2), (v_2, v_3), (v_3, v_1)]$

Leur étude est importante en théorie des graphes.
Exemples : 1) la longueur d'un chemin d'un graphe qui contient des cycles n'est pas forcément borné 2) infinité de chemins entre v_0 et v_4 ci-dessus.

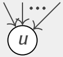
Un graphe qui n'a pas de cycle est qualifié d'*acyclique*.

Un graphe connexe non-dirigé acyclique est parfois appelé un arbre. (plus précisément, le choix d'un ordre sur les noeuds fixe un arbre)

Exemple

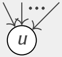


Test d'acyclicité

Soit $G = (V, E)$ un graphe dirigé. S'il est acyclique, il a forcément un noeud qui n'est la source d'aucune arête : 

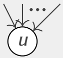
. On appelle un tel noeud une feuille.

Test d'acyclicité

Soit $G = (V, E)$ un graphe dirigé. S'il est acyclique, il a forcément un noeud qui n'est la source d'aucune arête :  . On appelle un tel noeud une feuille.

Un graphe privé d'une feuille est acyclique ssi le graphe de départ l'est aussi.

Test d'acyclicité

Soit $G = (V, E)$ un graphe dirigé. S'il est acyclique, il a forcément un noeud qui n'est la source d'aucune arête : 

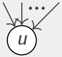
. On appelle un tel noeud une feuille.

Un graphe privé d'une feuille est acyclique ssi le graphe de départ l'est aussi.

Un algo simple :

- Si G n'a pas de noeud, il est acyclique $\square$
- On cherche une feuille de G
  - s'il n'en a pas, G est cyclique $\square$
  - sinon, on ôte la feuille, et on recommence du début

Test d'acyclicité

Soit $G = (V, E)$ un graphe dirigé. S'il est acyclique, il a forcément un noeud qui n'est la source d'aucune arête : 

. On appelle un tel noeud une feuille.

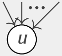
Un graphe privé d'une feuille est acyclique ssi le graphe de départ l'est aussi.

Un algo simple :

- Si G n'a pas de noeud, il est acyclique $\square$
- On cherche une feuille de G
  - s'il n'en a pas, G est cyclique $\square$
  - sinon, on ôte la feuille, et on recommence du début

Cet algorithme termine car à chaque appel, on diminue le nombre de noeuds.

Test d'acyclicité

Soit $G = (V, E)$ un graphe dirigé. S'il est acyclique, il a forcément un noeud qui n'est la source d'aucune arête : 

. On appelle un tel noeud une feuille.

Un graphe privé d'une feuille est acyclique ssi le graphe de départ l'est aussi.

Un algo simple :

- Si G n'a pas de noeud, il est acyclique $\square$
- On cherche une feuille de G
  - s'il n'en a pas, G est cyclique $\square$
  - sinon, on ôte la feuille, et on recommence du début

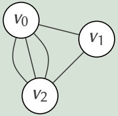
Cet algorithme termine car à chaque appel, on diminue le nombre de noeuds.

Exercice :

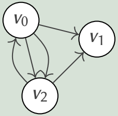
Peut-on faire plus efficace que de reparcourir le graphe à la recherche d'une feuille à chaque appel ?

Un *multigraphe* est un graphe qui peut avoir plusieurs arêtes entre deux mêmes noeuds.

Exemple

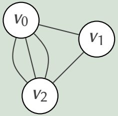


ou

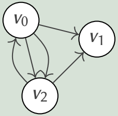


Un *multigraphe* est un graphe qui peut avoir plusieurs arêtes entre deux mêmes noeuds.

Exemple



ou



Problème : E est un ensemble $\Rightarrow$ pas de doublon. Solution : faire de E un *multiensemble* (simplement un ensemble où on peut avoir plusieurs fois un même élément).

Exemple (avec G ci-dessus)

$$G = \left(\{v_0, v_1, v_2\}, \{\{(v_0, v_1), (v_0, v_2), (v_0, v_2), (v_2, v_0), (v_2, v_1)\}\}\right)$$

Représentation informatique

Plein d'approches différentes selon ce que l'on veut optimiser.

Ici, utilisation d'ensembles / dictionnaires

Représentation informatique

Plein d'approches différentes selon ce que l'on veut optimiser.

Ici, utilisation d'ensembles / dictionnaires $\rightarrow$ temps d'accès en moyenne très bon, dans le pire des cas mauvais, et utilisation espace pas terrible.

Représentation informatique

Plein d'approches différentes selon ce que l'on veut optimiser.

Ici, utilisation d'ensembles / dictionnaires $\rightarrow$ temps d'accès en moyenne très bon, dans le pire des cas mauvais, et utilisation espace pas terrible.

Dans la suite, une approche objet, structure avec accès en temps constant à n'importe quel noeud, doublement chaînée.

Une classe pour les noeuds, et une pour les graphes (ouverts, dirigés).

```python
class node:
    def __init__(self, identity, label, parents, children):
        '''
        identity: int; its unique id in the graph
        label: string;
        parents: int->int dict; maps a parent node's id to its multiplicity
        children: int->int dict; maps a child node's id to its multiplicity
        '''
        self.id = identity
        self.label = label
        self.parents = parents
        self.children = children
```

```python
class open_digraph: # for open directed graph

    def __init__(self, inputs, outputs, nodes):
        '''
        inputs: int list; the ids of the input nodes
        outputs: int list; the ids of the output nodes
        nodes: node iter;
        '''
        self.inputs = inputs
        self.outputs = outputs
        self.nodes = {node.id:node for node in nodes}
        # self.nodes: <int,node> dict
```

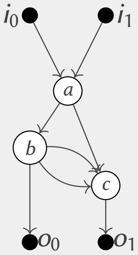
La classe du graphe (dirigé ouvert)

```python
class open_digraph: # for open directed graph

    def __init__(self, inputs, outputs, nodes):
        '''
        inputs: int list; the ids of the input nodes
        outputs: int list; the ids of the output nodes
        nodes: node iter;
        '''
        self.inputs = inputs
        self.outputs = outputs
        self.nodes = {node.id:node for node in nodes}
        # self.nodes: <int,node> dict
```

On utilise dans `self.nodes` :

- un dictionnaire "`dict`" : une structure de donnée dynamique, mutable, avec
  - accès/ajout/retrait en général en temps constant
  - utilisation d'espace assez médiocre
- une compréhension




```
n0 = node(0, 'a', {3:1, 4:1}, {1:1, 2:1})
n1 = node(1, 'b', {0:1}, {2:2, 5:1})
n2 = node(2, 'c', {0:1, 1:2}, {6:1})

i0 = node(3, 'i0', {}, {0:1})
i1 = node(4, 'i1', {}, {0:1})

o0 = node(5, 'o0', {1:1}, {})
o1 = node(6, 'o1', {2:1}, {})

G = open_digraph([3,4], [5,6], [n0,n1,n2,i0,i1,o0,o1])
```

⚠ Pour qu'un graphe soit bien formé, chaque arête doit se retrouver à la fois dans les `children` du noeud source, et dans les `parents` du noeud cible.

À faire rapidement

- Former un binôme ou un trinôme, et m'envoyer un mail pour me le signaler.
- M'envoyer aussi un mail si vous ne trouvez pas de binôme/trinôme.
- 1 personne par groupe : essayer de créer un repo sur GitHub/GitLab (et inviter les autres personnes du groupes).