Algorithmes gloutons

Exercice 1 : Un problème d'emploi de temps

Les cours de l'UFR *Chaos appliqué* ont tous lieu le mercredi, à des horaires détermines sans la moindre concertation.

Chaque UE offre le même nombre d'ECTS, et ne nécessite aucun travail personnel pour être validée.

Pour maximiser les crédits obtenus, il suffit donc de s'inscrire au plus grand nombre possible d'UE… Malheureusement, le secrétariat refuse de procéder aux IP lorsque les créneaux horaires se chevauchent.

Parmi les algorithmes gloutons suivants, lesquels fournissent une solution optimale ? Justifier.

(1) Schéma général de l'algorithme : tant qu'il reste des UE disponibles, choisir une UE α et éliminer toutes les UE incompatibles avec α…

a. terminant le plus tard possible.

b. de durée minimale

c. terminant le plus tôt possible

d. commençant le plus tôt possible

e. commençant le plus tard possible

f. créant le moins de conflits possible.

Je vais donc construire mon emploi de temps selon un choix locale. Une fois qu'il est fait, on ne revient pas dessus.

Pour chaque stratégie : montrer qu'elle est optimale, ou montrer un contre-exemple.

Montrer qu'un algorithme est non-optimale c'est plus facile (trouver un contre-exemple).

Pour montrer qu'une solution est optimale, par contradiction : on suppose que l'algorithme ne trouve pas la solution optimale. Alors on prend une solution optimale et on prend la solution trouvée par l'algorithme (qui est considérer mauvaise – c.-à-d. : pas optimale) et on montre qu'on peut remplacer le résultat de notre algorithme par la solution optimale, or cela est une contradiction vu qu'on supposait que l'algo n'est pas optimal.

Notation

On va noter $[a, b]$ une UE qui commence à a et termine à b.

(1a) Schéma général de l'algorithme : tant qu'il reste des UE disponibles,
choisir une UE α **terminant le plus tard possible**
et éliminer toutes les UE incompatibles avec α.

Pas optimale.

Contre-exemple (par notation) : prendre un « a » tout petit et un « b » super grand :

[0,6] [1,2] [3,4]

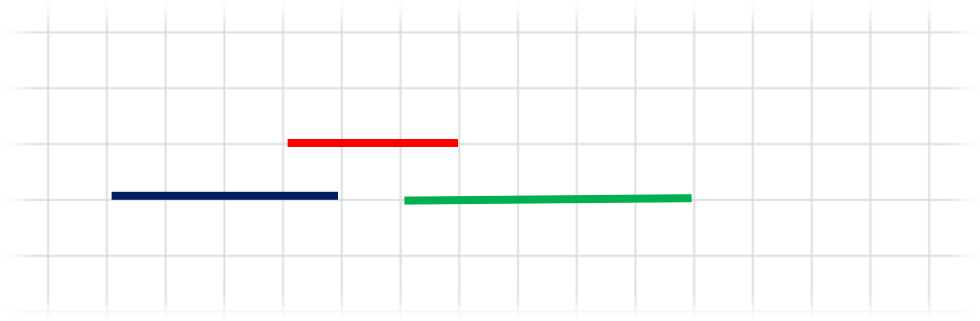
La stratégie « **terminant le plus tard possible** » fait qu'on choisie [0,6], alors que la bonne solution est : [1,2] [3,4].

En dessin :

(1**b**) Schéma général de l'algorithme : tant qu'il reste des UE disponibles, choisir une UE α **de durée minimale** et éliminer toutes les UE incompatibles avec α.

Pas optimale.

Contre-exemple : prendre un créneau qui est court mais qui chevauche 2 autres créneaux plus long :



En chiffres :

[0,4] [3,6] [5,10]

La stratégie « **de durée minimale** » fait qu'on choisit [3,6], donc cette stratégie n'est pas optimale.

(1**c**) Schéma général de l'algorithme : tant qu'il reste des UE disponibles,
choisir une UE α **terminant le plus tôt possible**
et éliminer toutes les UE incompatibles avec α.

Optimale. On peut voir ça comme le problème de distribution de ressources vu en CM9. Preuve : par la suite.

(1**d**) Schéma général de l'algorithme : tant qu'il reste des UE disponibles,
choisir une UE α **commençant le plus tôt possible**
et éliminer toutes les UE incompatibles avec α.

Pas optimale.

Contre-exemple : même contre-exemple que pour la stratégie **a**.

[0,6] [1,2] [3,4]

La stratégie « **commençant le plus tôt possible** » fait qu'on choisit [0,6], alors que la bonne solution est : [1,2] [3,4].

En dessin :

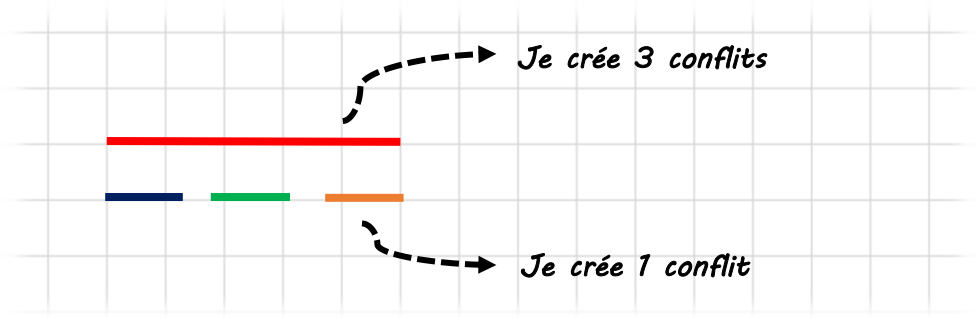
(1e) Schéma général de l'algorithme : tant qu'il reste des UE disponibles,
choisir une UE α **commençant le plus tard possible**
et éliminer toutes les UE incompatibles avec α.

Optimale.

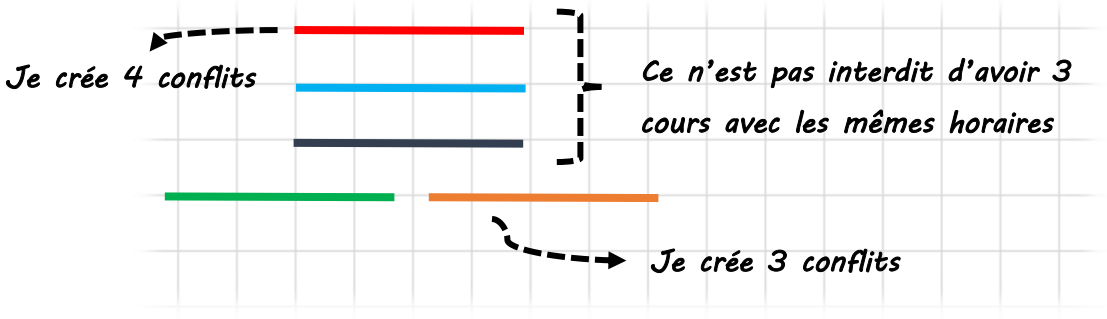
Symétrique a la stratégie **C** car on inverse juste les horaires, on multiplie tous par -1 et on inverse les intervalles, et ça revient au même. C'est la même stratégie mais dans l'autre sens : une fois de gauche -> droite, une fois de droite -> gauche.

(1f) Schéma général de l'algorithme : tant qu'il reste des UE disponibles,
choisir une UE α **créant le moins de conflits possible**
et éliminer toutes les UE incompatibles avec α.

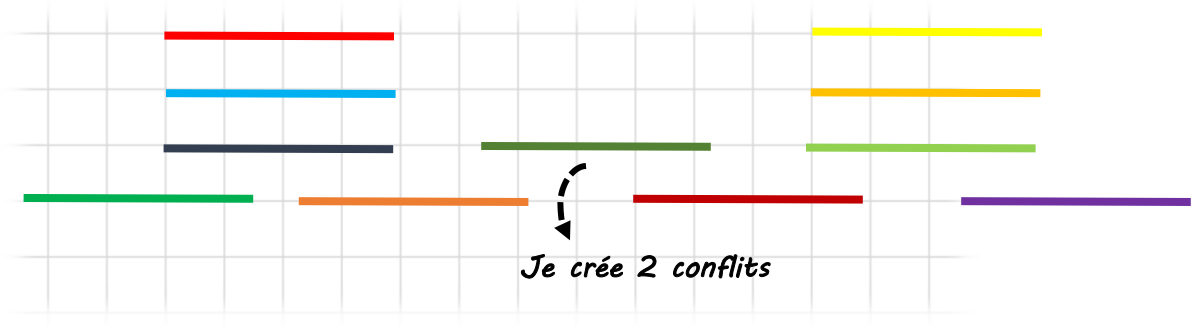
On a l'impression que c'est optimal :



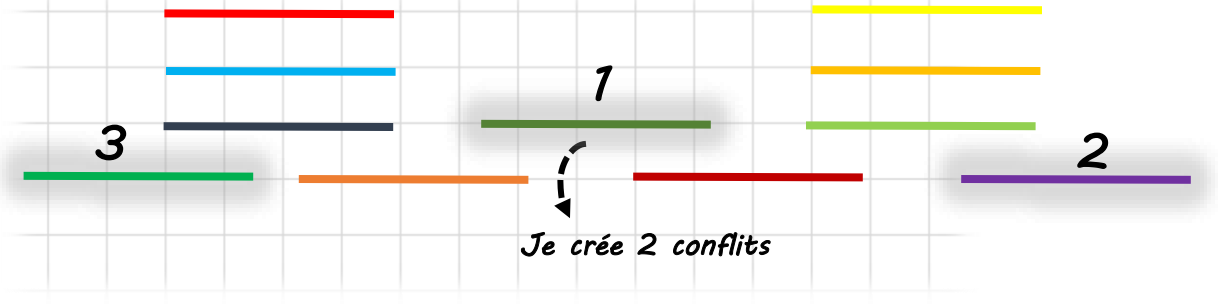
Maintenant, imaginons quelque chose comme ça :



Je peux étendre cet exemple pour arriver à un contre-exemple. On ajoute des cours :



La stratégie « **créant le moins de conflits possible** » fait qu'on choisit les cours suivants selon l'ordre suivant :



Alors que la bonne solution est : les 4 cours du bas. La stratégie commence pour choisir le cours « 1 », donc peut importe ce qu'on fait après : la solution n'est pas optimale.

Contre-exemple : l'idée est de construire un exemple qui me force à choisir le cours « 1 », alors que les 4 du bas sont meilleurs.

Donc : **Pas optimal.**

Comment montrer que la stratégie C est optimale ?

On ordonne les séquences par fin du cours.

Soit γ (Gamma) la séquence (séquence d'UEs) trouvée par la stratégie.
Soit σ (Sigma) une séquence optimale. (Peut y avoir plusieurs séquences optimales).

On regarde le premier endroit où les 2 séquences diffèrent (l'endroit ou γ et σ diffères).

Le début est pareil

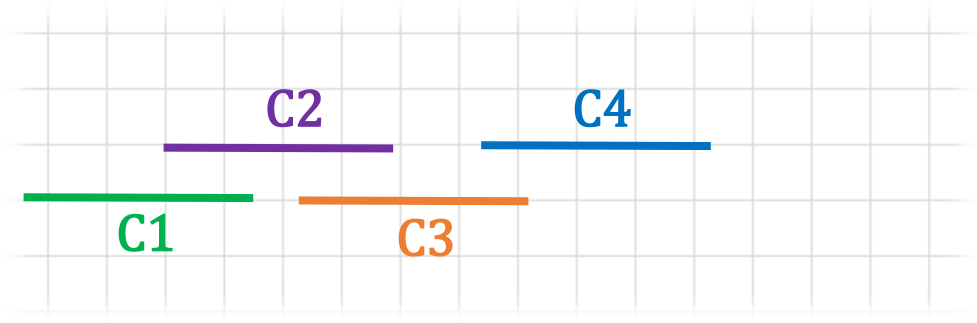
$$\gamma = \gamma_1 \underbrace{\alpha}_{alpha} \gamma_2$$

$$\sigma = \gamma_1 \underbrace{\beta}_{beta} \sigma_2$$

α, β : 2 UEs

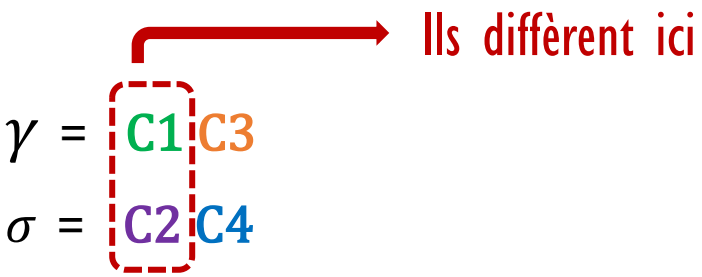
$\alpha \neq \beta$

Je peux remplacer le β dans mon sigma (σ) par α ?
Voici un exemple :



γ (Gamma) va trouver = C1 C3

Je prends maintenant une séquence optimale quelconque : σ = C2 C4



Dans σ, je peut remplacer C2 par C1 ?

La stratégie choisit le cours qui se termine le plus tôt, donc pas de problème de remplacer β par α, car α se termine plus tôt.

Donc on peut remplacer β par α dans σ.

Donc, quesque on obtient ?

On obtient une séquence qui est plus proche de Gamma qu'avant. Par induction, cette séquence optimale va devenir de plus en plus à Gamma et je finis par remplacer Sigma par Gama et ainsi j'ai montré que Gamma est une séquence optimale. **Donc, la séquence trouvée par la stratégie est optimale.**

- S'il n'y a pas une place ou Gamma et Sigma sont diffèrent : ce sont les même. On a fini la preuve.
- La propriété principale est que la stratégie choisit le cours qui se termine le plus tôt, donc je peux remplacer Beta par Alpha car Beta se termine après Alpha.

(La preuve n'est pas complète…)

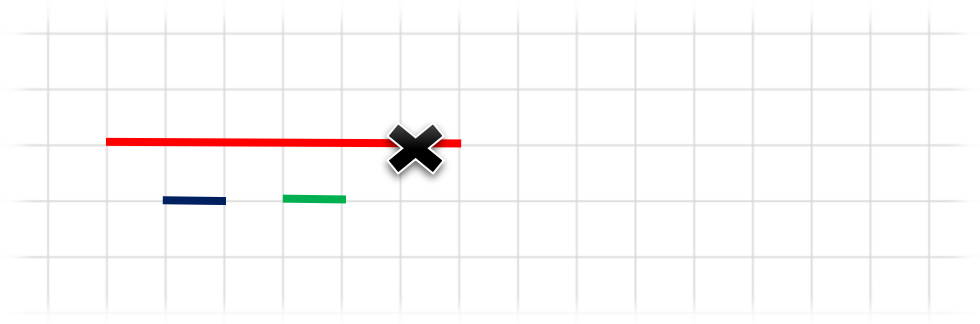
(2) Schéma général de l'algorithme : tant qu'il y a des conflits, éliminer une UE ω…

a. de durée maximale.

b. engendrant le plus de conflits possibles.

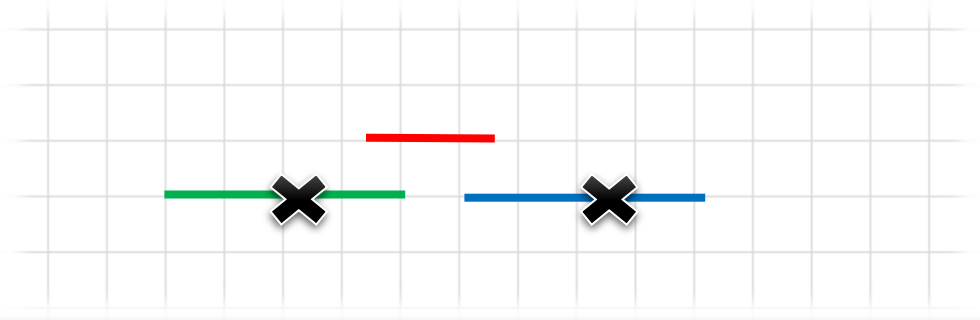
(1a) Schéma général de l'algorithme : tant qu'il y a des conflits, éliminer une UE ω **de durée maximale.**

On commence avec tous les cours et on enlève des cours, ainsi on trouve la solution.



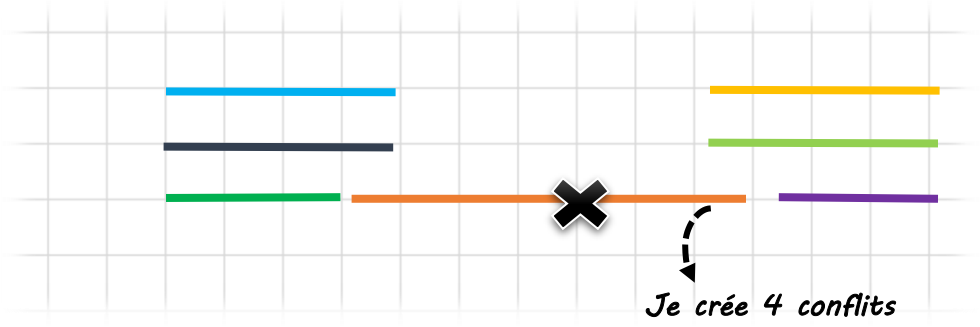
Là on a éliminé l'UE de durée maximal qui est aussi celle qui fait le plus de conflits.

Pas optimale. Contre-exemple :



(1b) Schéma général de l'algorithme : tant qu'il y a des conflits, éliminer une UE ω **engendrant le plus de conflits possibles.**

Pas optimale. Contre-exemple :



(3) **Tant qu'**il reste des UE disponibles,
soit α et β les deux UE commençant le plus tôt (α avant β)

- **Si** α et β sont disjointes, choisir α ;
- **Sinon, si** α recouvre β, éliminer α ;
- **Sinon**, éliminer β.

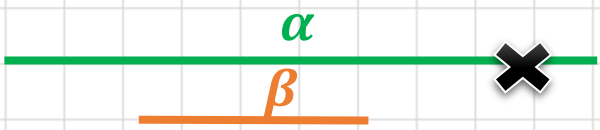
Optimal ?

Cas 1

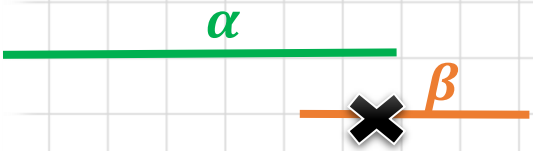
On commence par α

Cas 2

On vire α



Sinon



β termine plus tard
pas disjointes
α recouverte par β → β continue plus loin

⇒ *On élimine β*

Cet algo, on la pas déjà vu ?

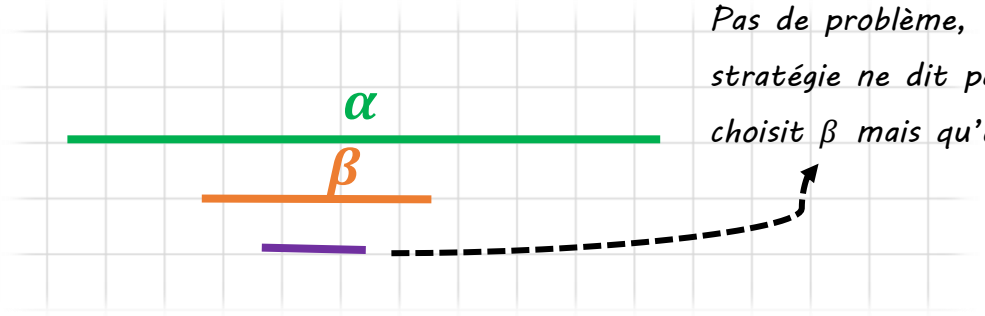
Oui. C'est la même stratégie que (IC) car :

- **Cas 1 :** α termine le plus tôt.
- **Cas 2 :** β termine le plus tôt.

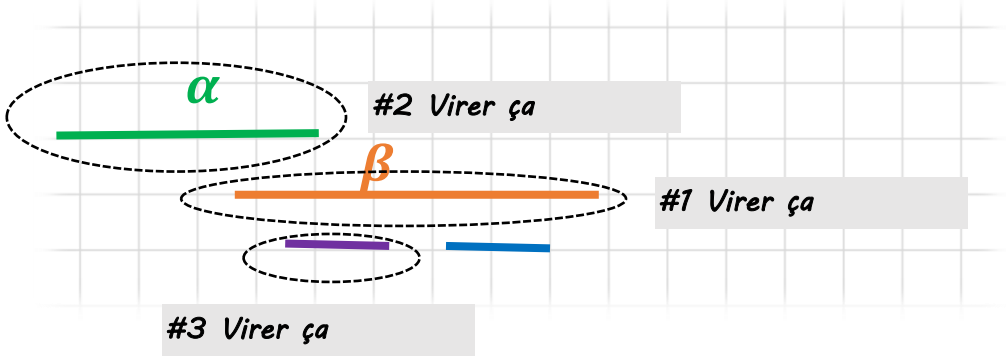
  Et si on a un autre cours qui termine le plus tôt ?

  On choisit ça.

  Pas de problème, car la stratégie ne dit pas qu'on choisit β mais qu'on élimine α



- **Cas 3 :** Avec la même raison que pour CAS 2, je ne choisis pas α, j'élimine β.



Pour tomber vers la stratégie Ic, faut faire plusieurs fois l'algo d'élimination pour arriver au même résultat que Ic.

Exercice 2 : Fractions égyptiennes

La façon égyptienne d'écrire des fractions $\frac{a}{b}$ avec $a,b \in \mathbb{N}$ et $a < b$ consiste à les écrire comme une somme de fractions unitaires (de la forme $\frac{1}{d}$) distinctes.

Par exemple : $\frac{2}{3} = \frac{1}{3} + \frac{1}{4} + \frac{1}{12}$

Le but de cet exercice est de minimiser le nombre de fractions utilisées dans la somme.

Comment peut-on écrire $\frac{5}{6}$?

On considère les deux algorithmes suivants, dus tous les deux à Fibonacci (13eme siècle).

Motivation : J'ai 5 personnes et j'ai 5 pizzas, comment faire pour distribuer équitablement les pizzas ? On peut s'en sortir en coupent certaines pizzas en 2, et certaines en 3…

On a $\frac{a}{b}$ $a < b$ et on veut trouver une representation avec des fraction unitaires $(a = 1)$ et que les dénominateurs soient différents.

Algorithme 1 :

Ecrire $\frac{a}{b}$ comme la somme de a termes $\frac{1}{b}$

Tant qu'il reste des termes identiques dans la somme,

En prendre un de plus petit dénominateur
et décomposer grâce à l'identité $\frac{1}{b} = \frac{1}{b+1} + \frac{1}{b(b+1)}$

Ainsi pour l'exemple $\frac{2}{3}$ on commence par écrire $\frac{1}{3} + \frac{1}{3}$ puis l'on décompose le deuxième $\frac{1}{3}$ en $\frac{1}{3+1} + \frac{1}{3(3+1)}$ ce qui donne la décomposition précédente :

$$\frac{2}{3} = \frac{1}{3} + \frac{1}{4} + \frac{1}{12}$$

(1) Appliquer l'algorithme sur les exemples $\mathbf{\frac{2}{5}}$ et $\mathbf{\frac{2}{11}}$

$$\frac{2}{5} = \frac{1}{5} + \mathbf{\frac{1}{5}}$$

$$\mathbf{\frac{1}{b}} = \frac{\mathbf{1}}{\mathbf{b+1}} + \frac{\mathbf{1}}{\mathbf{b(b+1)}} \Rightarrow \frac{1}{5} = \frac{1}{5+1} + \frac{1}{5(5+1)} = \frac{1}{6} + \frac{1}{30}$$

$$\frac{2}{5} = \frac{1}{5} + \mathbf{\frac{1}{6}} + \mathbf{\frac{1}{30}}$$

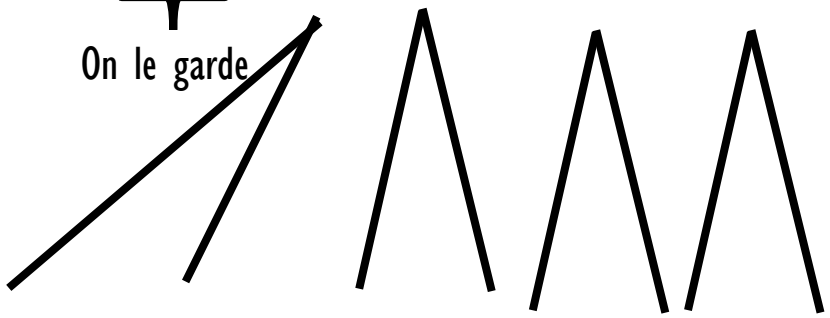
$$\frac{2}{11} = \frac{1}{11} + \mathbf{\frac{1}{11}}$$

$$\mathbf{\frac{1}{b}} = \frac{\mathbf{1}}{\mathbf{b+1}} + \frac{\mathbf{1}}{\mathbf{b(b+1)}} \Rightarrow \frac{1}{11} = \frac{1}{11+1} + \frac{1}{11(11+1)} = \frac{1}{12} + \frac{1}{132}$$

$$\frac{2}{11} = \frac{1}{11} + \mathbf{\frac{1}{12}} + \mathbf{\frac{1}{132}}$$

(2) (Commencer à) appliquer l'algorithme sur l'exemple $\mathbf{\frac{5}{121}}$

$$\frac{5}{121} = \underbrace{\frac{\mathbf{1}}{\mathbf{121}}}_{\text{On le garde}} + \frac{1}{121} + \frac{1}{121} + \frac{1}{121} + \frac{1}{121}$$



$$\frac{1}{122} + \underbrace{\frac{1}{121 \cdot 122}}_{14762} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots$$

(4) Si elle est optimale (en nombre de termes), le prouver ;

(5) Sinon, exhiber un contre-exemple ;

$\frac{2}{5}$ on peut le faire avec 2 fractions unitaires, a la place de $\frac{2}{5} = \frac{1}{5} + \frac{1}{6} + \frac{1}{30}$?

Oui, donc on n'a pas dans cet algo le plus petit nombre de fractions unitaires. Donc, l'algo n'est pas optimal.

Algorithme 2 (glouton) :

Partir avec la somme d'une fraction $\frac{a}{b}$ avec $a < b$.

Tant qu'elle n'est pas unitaire,

Appliquer à la dernière fraction de la somme l'identité

$$\frac{a}{b} = \frac{1}{\lfloor b/a \rfloor} + \frac{(-b) \bmod a}{b\lfloor b/a \rfloor}$$

Par exemple :

$$\frac{7}{15} = \frac{1}{3} + \frac{2}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120}$$

(1) Appliquer l'algorithme sur les exemples $\frac{\mathbf{2}}{\mathbf{5}}$ et $\frac{\mathbf{2}}{\mathbf{11}}$

L'idée est de dire : je vais choisir le plus grand dénominateur possible. Par exemple, si on a $\frac{2}{5}$:

$$\frac{\mathbf{a}}{\mathbf{b}} = \frac{\mathbf{1}}{\mathbf{[b/a]}} + \frac{\mathbf{(-b) \bmod a}}{\mathbf{b[b/a]}}$$

mod 2 *c'est soit 0, soit 1*

$$\frac{2}{5} = \frac{1}{[5/2]} + \frac{(-5) \bmod 2}{5 \cdot [5/2]} = \frac{1}{3} + \frac{1}{15}$$

Si ici j'ai une fraction pas unitaire, je continue.

$$\frac{2}{11} = \frac{1}{[11/2]} + \frac{(-11) \bmod 2}{11 \cdot [11/2]} = \frac{1}{6} + \frac{1}{11 \cdot 6} = \frac{1}{6} + \frac{1}{66}$$

(2) (Commencer à) appliquer l'algorithme sur l'exemple $\frac{\mathbf{5}}{\mathbf{121}}$

$$\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}$$

(4) Si elle est optimale (en nombre de termes), le prouver ;

(5) Sinon, exhiber un contre-exemple ;

Les 2 algo ne sont pas optimal.

C'est un problème très compliquer en réalité...

(3) Montrer que les 2 algo se termine en produisant une décomposition correcte

(6) Peut-on donner une borne de la longueur de la suite ainsi produite ?

D'abord admettent que ça termine. Pourquoi les algos sont corrects ?

Si l'algo termine on trouve à la fin une somme égal a la fraction de départ (égalité mathématique).

- Pas une preuve formelle.

Pourquoi l'algo 2 se termine ?

Pour l'algo 2 l'argument est très simple : je remplace $\frac{a}{b}$ par un truc unitaire + « autre chose ». L'« autre chose », qu'est-ce qu'on peut dire de lui ?

Strictement inferieur a a

On remplace $\frac{a}{b}$ par $\frac{1}{\dots} + \frac{(-b) \bmod a}{\dots}$

Le numérateur du dernier terme décroit, à chaque étape, au moins de 1 : car à chaque fois on fait $\mathbf{mod\ a}$ et on sait que $\mathbf{(-b) \bmod a < a}$

Donc, on termine au max après $(\mathbf{a-1})$ étapes, donc c'est linaire en $\mathbf{a}$, qu'importe le $\mathbf{b}$ (dans la complexité on ne s'intéresse pas au coup de $(-b) \bmod a$).

Pourquoi l'algo 1 se termine ?

On peut le voir aussi comme :

On remplace $\frac{a}{b}$ par $\frac{1}{b}+\frac{a-1}{b}$. Ça revient à : $\frac{1}{b}+\frac{a-1}{b+1}+\frac{a-1}{b(b+1)}$

Donc le a est remplacer par $a-1, a-1$ ... **récursivement** ensuite remplacer $a-2, a-2$ ... . **Donc le nominateur il diminue. On aura 2^a a terme.**

Mais y'a un petit problème. Ce que je fabrique à partir de $\frac{a-1}{b+1}$ et $\frac{a-1}{b(b+1)}$, ca risque pas de se chevaucher, risque qu'on obtient le même ? On peut argumenter que non mais c'est un peut compliquer…

Donc, on va juste dire que le numérateur diminue ⇒ l'algo se termine.

$$\frac{1}{b}+\overbrace{\frac{1}{b}+\cdots+\frac{1}{b}}^{(a-1)\ fois}$$

$$\frac{1}{b}+\left(\left(\frac{1}{b+1}+\frac{1}{b(b+1)}\right)+\cdots+\left(\frac{1}{b+1}+\frac{1}{b(b+1)}\right)\right)$$

Dans l'algo 2 au max j'ai a fractions unitaires. Si en entrée on lui donne $\frac{a}{b}$

Dans l'algo 1 2^a-1 fractions unitaires (pas au max, c'est toujours la même chose). Donc, si a est grand c'est beaucoup…

Algo 2 meilleur mais pas optimal.

```
def frac(a,b):
    if a == 1:
        return [(a,b)]
    else:
        return [(1,b)] + frac(a-1,b+1) + frac(a-1,b*(b+1))

def frac2(a,b):
    if a == 1:
        return [(a,b)]
    else:
        d = b//a+1
        return [(1,d)] + frac2((-b)%a,b*d)
```

Exercice 3 : Algorithme de Boruvka

Il s'agit d'un algorithme d'arbre couvrant minimal écrit en 1926 par Otakar Boruvka pour concevoir un réseau de distribution électrique pour la Moravie du Sud.

Il travaille sur des multi-graphes $\mathbf{G = (V, E)}$:
boucles autorisées, plusieurs arêtes possibles entre deux sommets donnés.

F l'ensemble des arêtes choisies (on le remplit peu à peu)

Procedure Boruvka(G)

- $F \leftarrow \emptyset$ — *F ← vide*
- **tant que** G a plusieurs sommets **faire** — *Tant que G n'est pas réduit à un sommet*
  - supprimer les boucles de G. — *Les arêtes qui partent et arrivent au même sommet*
  - **pour** tous sommets x et y **faire**
    - **si** plusieurs arêtes entre x et y **alors**
      - ne garder que la plus légére — *Remplacer les arêtes multiples entre deux sommets par une seule dont le poids est le minimum*
  - **pour** tout sommet x **faire**
    - $e_x \leftarrow \{x, y\}$ l'arete minimale parmi les arêtes touchant x (Pami les arêtes de poids minimal , celle avec y minimale). — *Trouver l'arête e_x de poids minimum adjacente à x*
    - $F \leftarrow F \cup \{x, y\}$ — *F ← F union e_x*
  - **pour** toute arête e_x choisie à l'étape précédente **faire**
    - contracter e_x (Ses deux extrémités deviennent un seul sommet) — *Fusionner les sommets aux extrémités de e_x*
  - **retourner** F — *F l'ensemble des arêtes choisies*

Problème de l'arbre couvrant minimal

En théorie des graphes, étant donné un graphe non orienté connexe dont les arêtes sont pondérées, un arbre couvrant de poids minimal de ce graphe est un arbre couvrant (sous-ensemble qui est un arbre et qui connecte tous les sommets ensemble) dont la somme des poids des arêtes est minimale.

Algorithme de Borůvka : Principe

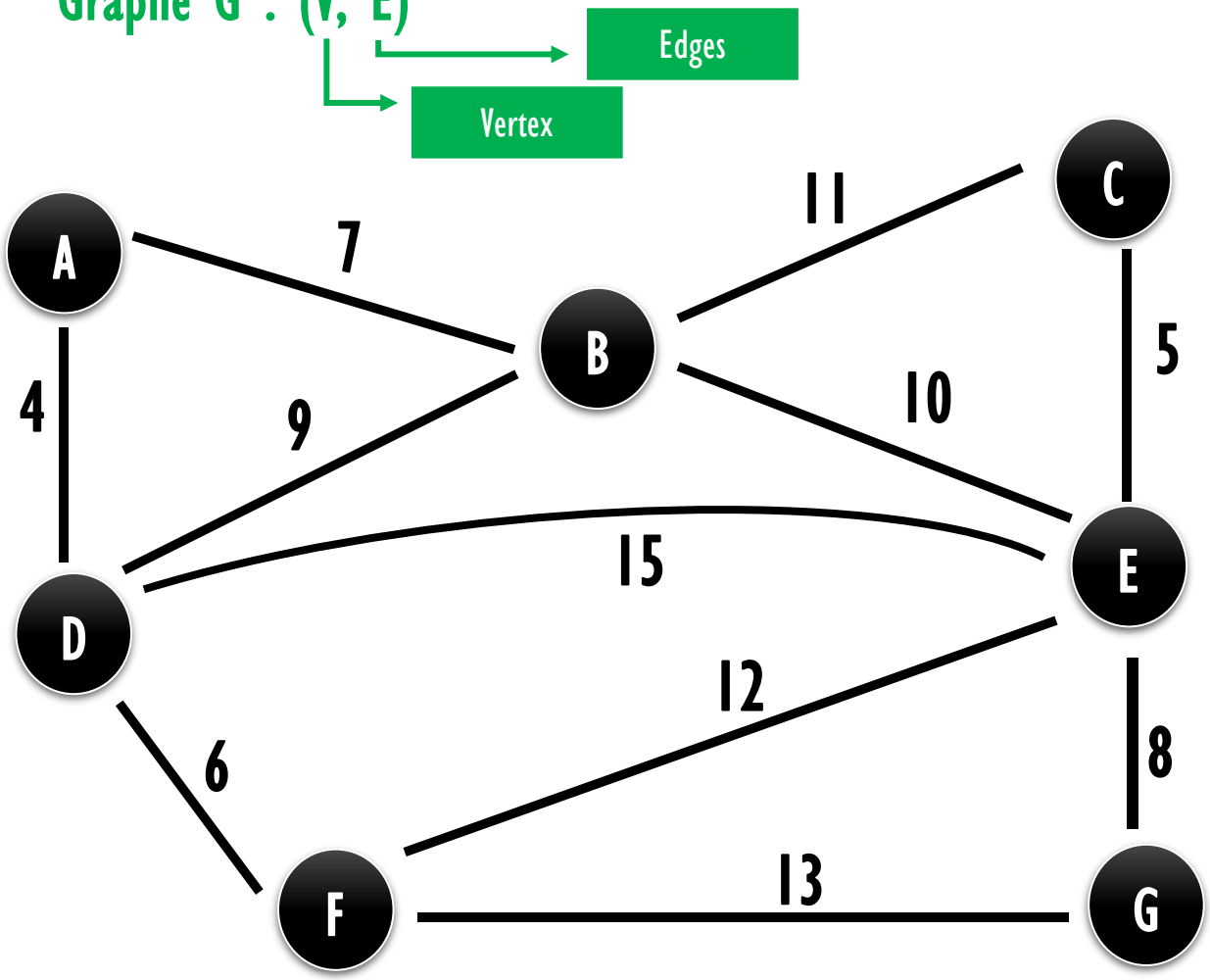
Le principe est de réduire G en contractant des arêtes : on choisit peu à peu les arêtes qui seront dans l'arbre, et à chaque fois que l'on en choisit une, on fusionne les nœuds que cette arête relie. Ainsi, il ne reste plus qu'un sommet à la fin.

On peut aussi décrire l'algorithme sans parler de contraction : on construit une forêt dont les arbres fusionnent peu à peu pour former un arbre couvrant de poids minimum.

Source : Wikipedia - CC BY-SA 3.0

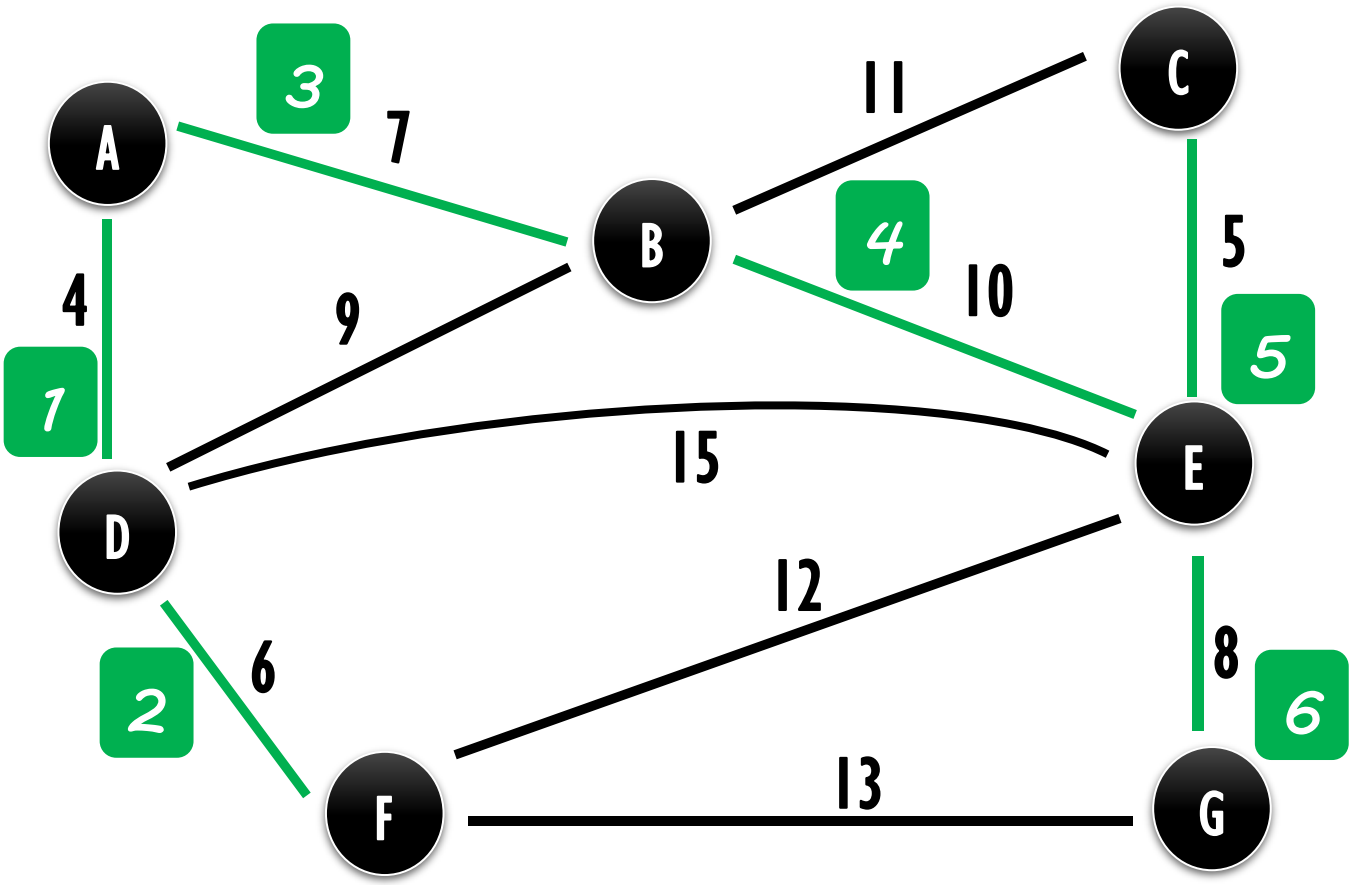
(1) Faire tourner l'algorithme sur un exemple

Graphe G : (V, E)



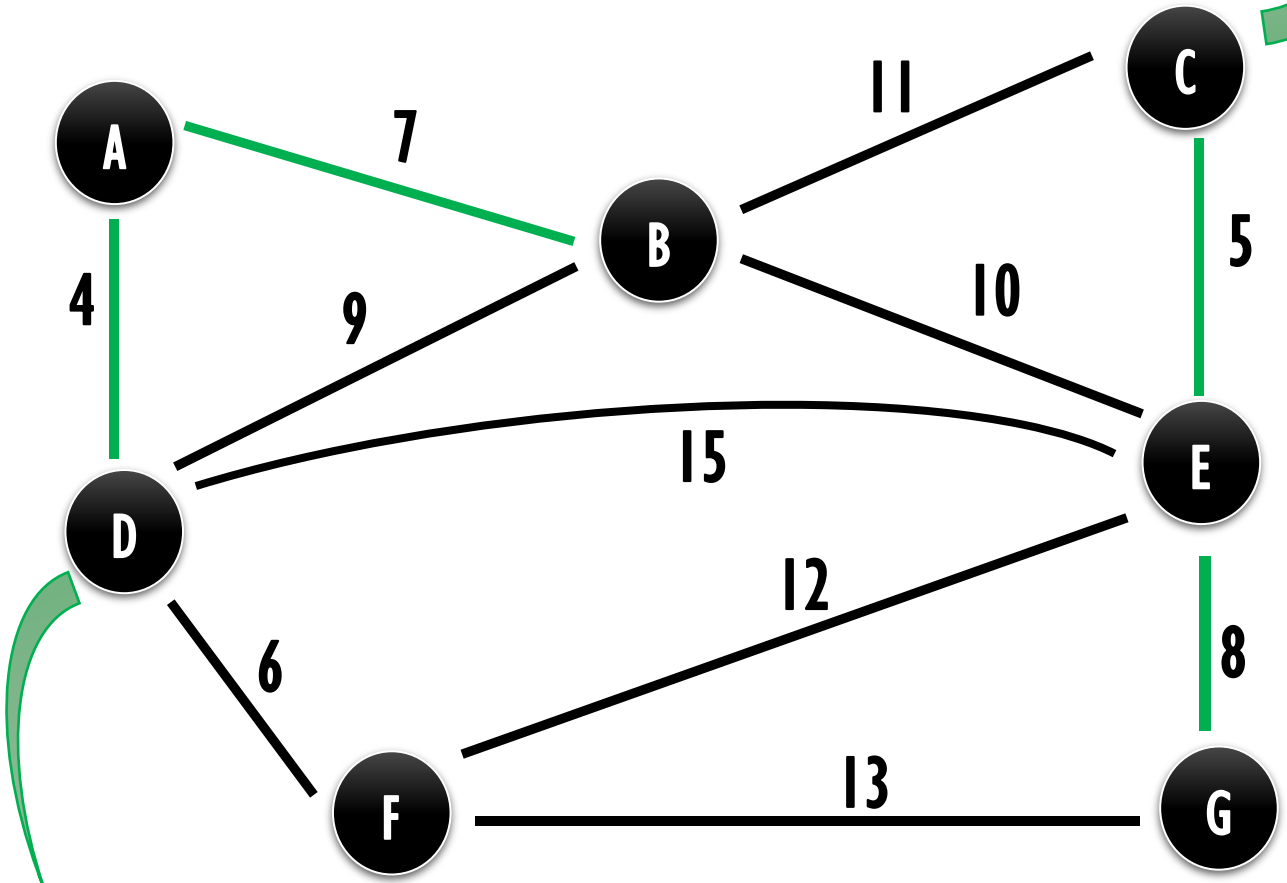
L'algorithme de Prim aurai prit n'importe qu'elle sommet (D, par exemple) et après il prend l'arrête le moins cher de sorte qu'il y aura pas de cycles (l'arrête de poids 4), ensuite on prend l'arrête de poids 6, ensuite l'arrête de poids 7 (on regarde tous les sommets pour choisir la plus petite)…

On prend les arrêtes dans l'ordre suivant :



Algorithme de Boruvka

Pour chaque sommet : choisir une arrête (le plus petite). En cas d'égalité : faire attention

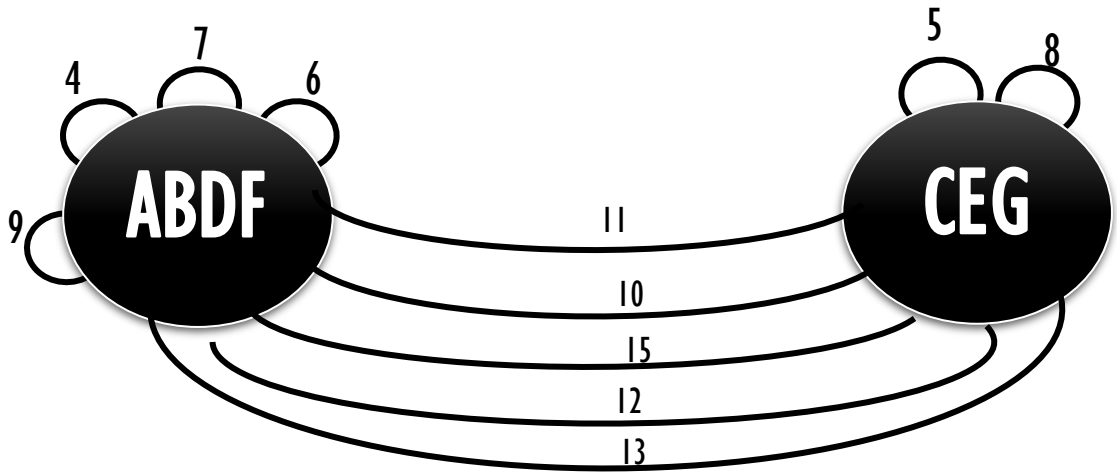


C'est toujours le choix AD même si il a déjà été choisi

Suite du déroulement de l'algorithme de Boruvka

(1) Mettre les arrêtes dans F.

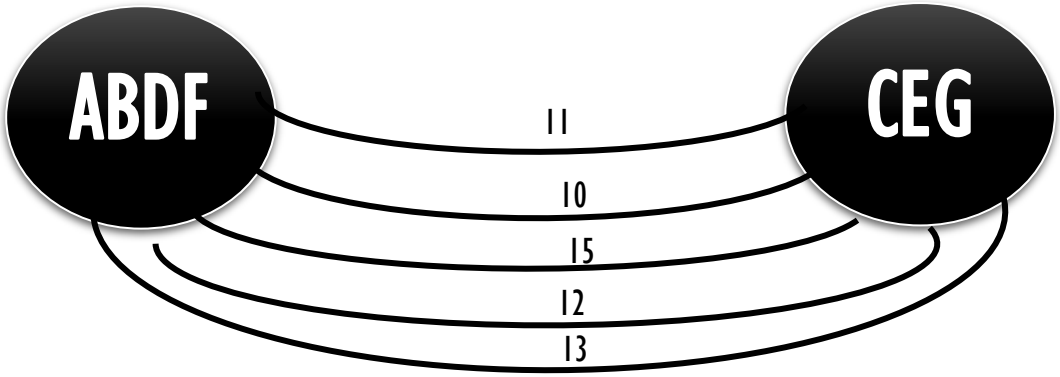
(2) Mettre les sommets ensemble :



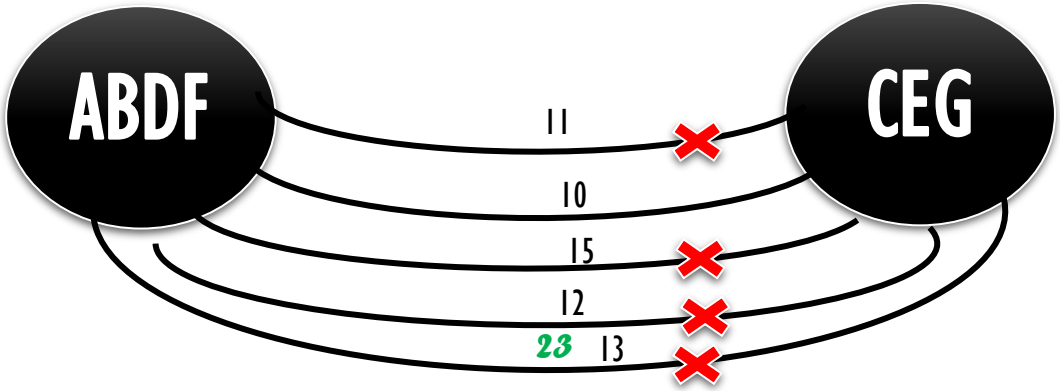
Voilà le nouveau graphe.

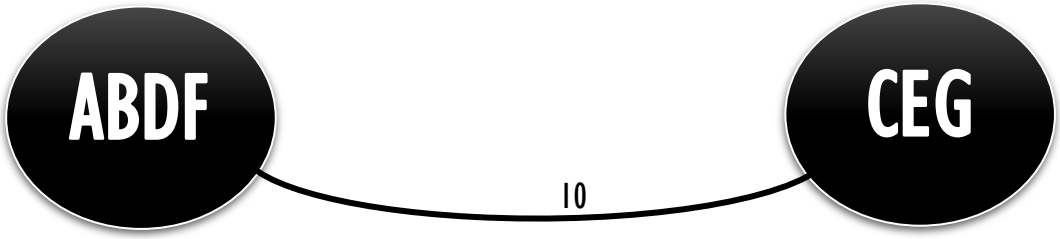
On a fait la 1er itération, maintenant on fait la 2eme :

(1) Suppression des boucles :



(2) Ensuite, on garde que le 10 :





(3) On contracte, donc a la fin il va me rester :

J'ai 1 seul nœud, donc j'ai fini.

(2) Quelle est sa complexité ? En particulier, donner une borne pour le nombre de la boucle externe, et réfléchir à l'implémentation des opérations des boucles ***pour***

Dans la question (1) on a exécuté 2 fois la boucle externe.

n sommets, m arrêtes $m \geq n$

Si on commence avec n sommets, après 1 tour de boucles, on a au maximum $\lfloor n/2 \rfloor$ sommets car chaque sommet va être contracté au moins avec un autre (on contrainte au moins 2 sommets d'une arrête adjacente à chaque sommet).

Donc, si à chaque fois on divise par 2, combien de boucles ?
On fait la boucle extérieure au maximum $\log_2(n)$ fois.

A l'intérieur ?

Le cout des intérieurs des boucles ? en tous $|m|$.

Donc,

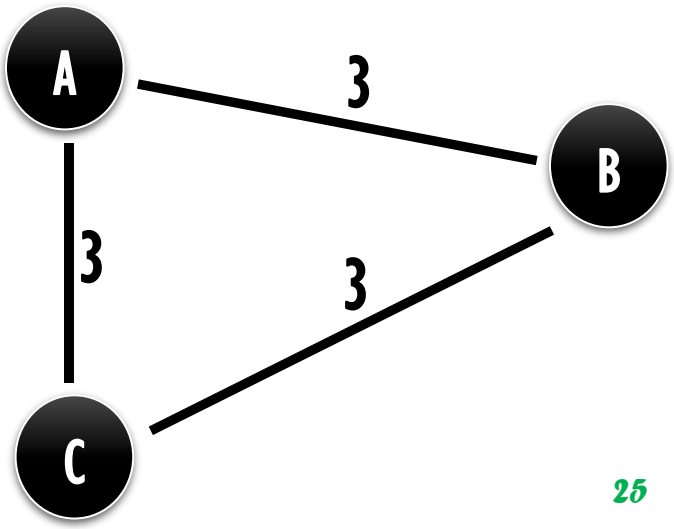
$$O(m \cdot \log_2(n))$$

Le nombre d'arrêtes : ce qui compte le plus dans la complexité.

(3) A quoi la règle « y minimal » sert-elle ? Qu'est-ce qui se passe, si on a par exemple un graphe 3 sommets tous connectés avec une arête de même valeur ?

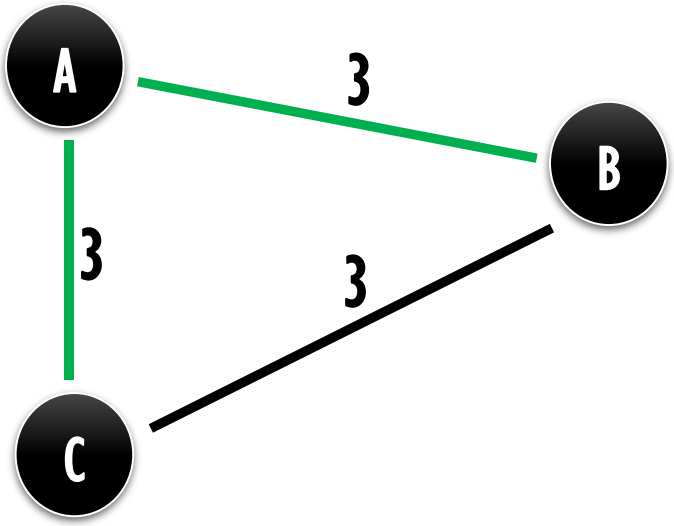
En cas d'égalité ?

Que fait l'algo dans un cas comme ça :



Si pour chaque sommet on aurai choisi au hasard, ça aurait posai un problème car A aurai pu choisir AC, B aurai pu choisir BC et B aurai pu choisir BA.

Donc, faut ajouter une règle : « parmi les arêtes de poids minimal, choisir celle avec y minimal » (y = la destination) : **ça sert à éviter des cycles.**



(4) Montrer la correction de cet algorithme.

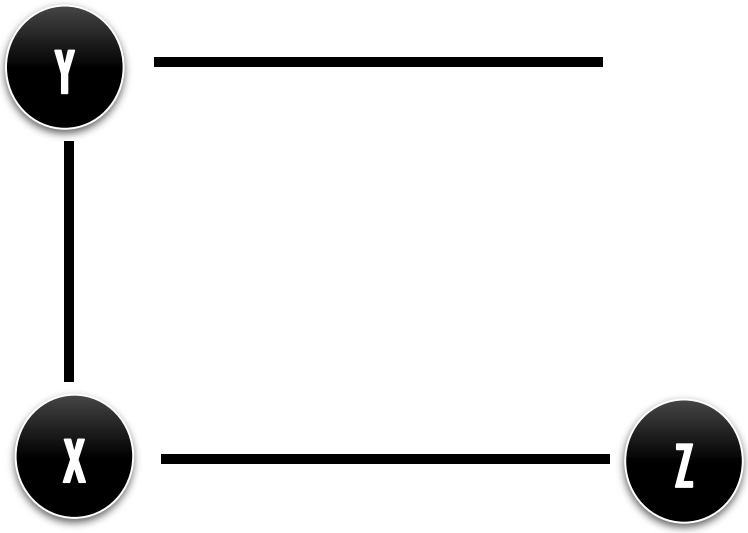
On va pas le montrer en détails, mais on va juste montrer une chose :

pourquoi l'arrête minimale qui sort d'un sommet, doit être de poids minimal dans l'arbre couvrant ?

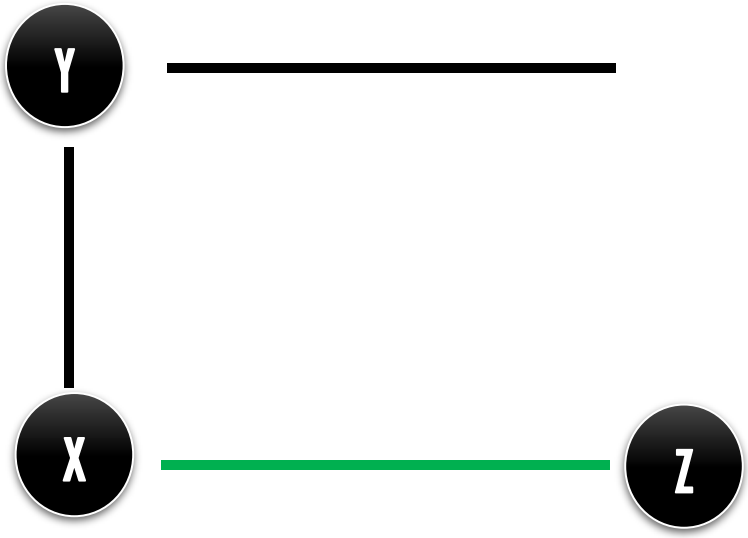
On montre que l'arrête minimale partant d'un sommet x, et qui va jusqu'à y :(x, y), doit être dans l'Arbre Couvrant Minimal (ACM) du graphe.

Preuve par contradiction :

Supposons que l'ACM ne contient pas l'arrête : (x, y) , il y a une autre arrête (x, z) dans l'ACM.



On choisit :



Maintenant, on enlève (x, z) et on ajoute (x, y) et on obtient un arbre couvrant meilleur. Contradiction !

(5) On dit qu'un algorithme est *parallélisable* si son temps d'exécution peut être divisé par **k** en utilisant **k** processeurs.

Cet algorithme est-il *parallélisable* ?

Cet algorithme, on peut le paralléliser ?

Si j'ai k processeurs → division du temp d'exécution par k, oui c'est possible, car pour chaque nœud on choisie une arrête, les choix sont complétement indépendant, donc pas de problème.

Si j'ai, par exemple, 1 million de sommets c'est plus facile de faire le calcul en parallèle, ce qui est pas le cas pur l'algorithme de Prim car en Prim le choix se fait un par un (dans notre algorithme : choix complètement indépendant des autres).