Algorithmes gloutons

Le principe de l'algorithme glouton : faire toujours un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale.

1 Égypte

On appelle fraction égyptienne une fraction de la forme $\frac{1}{n}$ avec $n \in \mathbb{N}^*$.

1. Soient a et b deux entiers (premiers entre eux) tels que $a < b$. Donner l'expression de la fraction égyptienne la plus grande parmi les fractions égyptiennes strictement plus petites que $\frac{a}{b}$ et l'expression de leur différence.
2. On considère l'algorithme suivant :

Xcas

```
egypte(a,b):={
local c,d;
c:=a;
a:=numer(c/b);
b:=denom(c/b);
si a==1 alors return   b   ;
sinon
c:=a-irem(b,a);
d:=iquo(b,a)+1;
return (d,egypte(c,b*d));
fsi;
}:;
```

   dans lequel l'entrée $(a;b)$ est un couple d'entiers avec $a < b$.
   Écrire une version itérative de l'algorithme.
3. Quelle est la sortie de l'algorithme avec l'entrée $(a,b) = (2,2n+1)$ où n est un entier naturel non nul ?
4. Quel est le rôle de cet algorithme ? Démontrer.
5. L'algorithme glouton proposé donne-t-il une décomposition en somme de fractions égyptiennes avec le minimum de termes possibles ?

Une résolution

1. $$\frac{a}{b} = \frac{1}{\lfloor \frac{b}{a} \rfloor + 1} + \frac{a - \text{irem}(b,a)}{b \times \left(\lfloor \frac{b}{a} \rfloor + 1 \right)}$$

   Il n'y a pas d'entier n tel que $\frac{a}{b} > \frac{1}{n} > \frac{1}{\lfloor \frac{b}{a} \rfloor + 1}$ puisque cela s'écrit aussi $\frac{b}{a} < n < \lfloor \frac{b}{a} \rfloor + 1$.

2. Une version itérative :

Xcas

```
egypt(a,b):={
local k,tmp;
// k initialisé à sequence vide  :
 k:=seq[];
// on réduit la fraction a/b :
tmp:=numer(a/b);
b:=denom(a/b);
a:=tmp;
// boucle principale :
tantque a<>1 faire
        // ajout d'un élément à la séquence k :
        k:=k,(iquo(b,a)+1);
        // fraction restante et simplification de cette fraction :
        tmp:=a-irem(b,a);
        b:=b*(iquo(b,a)+1);
        a:=numer(tmp/b);
        b:=denom(tmp/b);
ftantque;
// ajout du dernier dénominateur à la séquence :
k:=k,(b);
return k;
}:;
```

3.
$$\frac{2}{2n+1} = \frac{1}{n+1} + \frac{1}{(n+1)(2n+1)}$$

4. L'algorithme écrit le quotient $\frac{a}{b}$ sous la forme d'une somme de fractions égyptiennes à dénominateurs strictement croissants.

   (a) L'algorithme se termine.
   a, b désignent ci-dessous le numérateur et le dénominateur après réduction de la fraction $\frac{a}{b}$.
   A chaque étape,
      i. Soit $a = 1$ (c'est à dire $\text{irem}(b,a) = 0$) et l'algorithme se termine.
      ii. Soit $a \neq 1$. Dans ce cas, on a $\text{irem}(b,a) \neq 0$ (puisque, la fraction étant irréductible, on a b non multiple de a). On a donc $0 < \text{irem}(b,a) < a$ et $0 < a - \text{irem}(b,a) < a$. Les fractions $\frac{a-\text{irem}(b,a)}{b\times\left(\lfloor\frac{b}{a}\rfloor+1\right)}$ successives ont (après réduction) des numérateurs strictement décroissants compris entre 1 et a : l'algorithme se termine.

   (b) La liste des dénominateurs renvoyée par l'algorithme est strictement croissante.
   On a $2\frac{b}{a} > 2\lfloor\frac{b}{a}\rfloor$ pour $a \neq 1$ puisque $\frac{a}{b}$ est réduite et $2\lfloor\frac{b}{a}\rfloor \geqslant \left\lfloor\frac{b}{a}\right\rfloor + 1$ puisque $\lfloor\frac{b}{a}\rfloor \geqslant 1$. On a donc $2\frac{b}{a} > \lfloor\frac{b}{a}\rfloor + 1$ lorsque $a \neq 1$ (et aussi, de façon claire, lorsque $a = 1$ puisque $b > 1$ à toute étape).
   On a donc :
$$\frac{a}{b} - \frac{1}{\lfloor\frac{b}{a}\rfloor+1} < \frac{1}{\lfloor\frac{b}{a}\rfloor+1}$$

En d'autres termes, la fraction $\frac{a-\text{irem}(b,a)}{b\times\left(\lfloor\frac{b}{a}\rfloor+1\right)}$ est strictement plus petite que la fraction égyptienne précédente. Les fractions égyptiennes obtenues dans la suite de la décomposition seront donc strictement plus petites que la fraction égyptienne $\frac{1}{\lfloor\frac{b}{a}\rfloor+1}$ et donc à dénominateurs strictement plus grands.

5. Le nombre de termes n'est pas minimal.
   L'algorithme renvoie par exemple la décomposition suivante :

$$\frac{19}{20}=\frac{1}{2}+\frac{1}{3}+\frac{1}{9}+\frac{1}{180}$$

   alors que l'on a :

$$\frac{19}{20}=\frac{10+5+4}{20}=\frac{1}{2}+\frac{1}{4}+\frac{1}{5}$$

   Ou encore :

$$\frac{5}{121}=\frac{1}{25}+\frac{1}{757}+\frac{1}{763\,309}+\frac{1}{873\,960\,180\,913}+\frac{1}{1\,527\,612\,795\,642\,093\,418\,846\,225}$$

   alors que :

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

2 Les épreuves dans le gymnase

Dans un gymnase doivent se dérouler une série d'épreuves. Les épreuves ne sont pas seulement caractérisées par leurs durées : chaque épreuve est caractérisée par une date de début d_i et une date de fin f_i.
On souhaite "caser" le plus possible d'épreuves, deux épreuves ne pouvant avoir lieu en même temps (leurs intervalles de temps doivent être disjoints).

Glouton 1 – On trie les épreuves par durée croissante, on choisit la plus courte, puis la plus courte parmi celles qui lui sont compatibles, puis …Ce choix mène-t-il au déroulement d'un nombre d'épreuves maximal ?

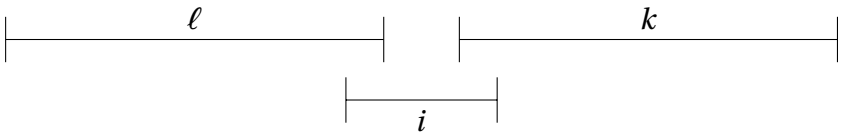
Glouton 2 – On trie les événements par dates de commencement croissantes et on gloutonne : on choisit l'événement commençant le plus tôt, puis le plus tôt parmi les événements compatibles …Même question.

Glouton 3 – On trie cette fois les événements par nombre d'intersections croissant : on choisit d'abord celui qui intersecte le moins d'événements, puis …Même question.

Glouton 4 – On trie les événements par dates de fin croissantes et on gloutonne : on choisit l'épreuve se terminant au plus tôt, puis l'épreuve se terminant au plus tôt parmi celles qui sont compatibles à la première…Même question.

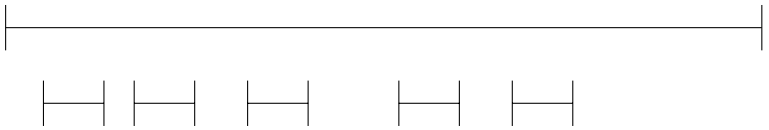
Une résolution

Glouton 1 – Non optimal. Contre-exemple :



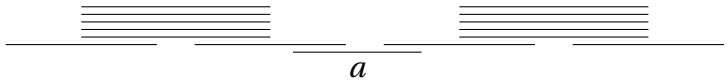
L'algorithme choisit i : une épreuve alors que deux sont possibles.

Glouton 2 – Non optimal. Contre-exemple :



L'algorithme choisit une épreuve au lieu de plusieurs.

Glouton 3 – Non optimal. Contre-exemple :



L'algorithme choisit a et trois événements seront choisis au total. Alors que l'on peut clairement en choisir 4.

Glouton 4 – Optimal.

On note f la date de fin la plus petite.

Soit $\Gamma = \{f_1, f_2, \ldots, f_k\}$ un ensemble de dates de fin d'épreuves définissant une solution optimale avec $f_1 < f_2 < \cdots < f_k$.

Si $f \neq f_1$, on remplace une épreuve correspondant à f_1 par une épreuve correspondant à f, cela est possible puisque $f \leqslant f_1 < f_2 < \ldots$. L'ensemble $\Gamma' = \{f, f_2, \ldots, f_k\}$ est donc également optimal (même nombre d'épreuves).

On note ensuite f' la date de fin la plus petite parmi les dates de fin d'épreuves compatibles avec f (c'est à dire les épreuves de date de début $> f$).

Si $f_2 \neq f'$, on remplace une épreuve correspondant à f_2 par une épreuve correspondant à f', cela est possible puisque f' compatible avec f et $f' \leqslant f_2 < \ldots$. L'ensemble $\Gamma'' = \{f, f', \ldots, f_k\}$ est donc également optimal (même nombre d'épreuves).

...

3 Monnaie

1. On dispose de pièces de monnaie (sans limite d'effectifs) de 18 chloubis , 7 chloubis et 1 chloubi. On cherche la façon de payer n chloubis (n entier naturel) en utilisant le moins possibles de pièces. On gloutonne ainsi : on utilise des pièces de 18 chloubis tant qu'on peut, puis des pièces de 7 tant qu'on peut, puis des pièces de 1. Cette gloutonnerie donnera-t-elle une réponse optimale ?
2. La même gloutonnerie conduit-elle à un nombre minimal de pièces lorsque les pièces sont des pièces 10, 5, 2 et 1 ?

3. Soit $p \in \mathbb{N}$, $p \geqslant 2$. La même gloutonnerie conduit-elle à un nombre minimal de pièces lorsque les pièces sont des pièces de 1, p, p^2, p^3, …, p^k ?

Une résolution

1. L'existence de pièces de 1 chloubi assure que l'algorithme donne une façon de payer. Mais l'algorithme ne renvoie pas une solution optimale.

   Avec Xcas :

   Xcas
   ```
   monnaie(n,a,b,c):={
   local na,nb,nc;
   na:=iquo(n,a);n:=irem(n,a);
   nb:=iquo(n,b);n:irem(n,b);
   nc:=iquo(n,c);n:=irem(n,c);
   return(na,nb,nc,n);
   }:;
   ```

   monnaie(21,18,7,1) renvoie (1,0,3,0). Le dernier 0 signifie que tout est payé : avec 1 pièce de 18 chloubis et 3 de 1 chloubi. Alors qu'on peut payer avec 3 pièces seulement (3 pièces de 7).

2. Avec xcas :

   Xcas
   ```
   monnaie(n,a,b,c,d):={
   local na,nb,nc,nd;
   na:=iquo(n,a);n:=irem(n,a);
   nb:=iquo(n,b);n:irem(n,b);
   nc:=iquo(n,c);n:=irem(n,c);
   nd:=iquo(n,d);n:=irem(n,d);
   return(na,nb,nc,nd,n);
   }:;
   ```

   La réponse de l'algorithme est optimale dans ce cas.

   Une solution optimale :

   (a) utilise au plus une pièce de 5 (sinon on remplace 2 pièces de 5 par une pièce de 10),

   (b) au plus 2 pièces de 2 (sinon on remplace 3 pièces de 2 par une de 5 + une de 1).

   (c) au plus 1 pièce de 1 (sinon on remplace 2 pièces de 1 par une pièce de 2).

   La somme payée avec les pièces de 1, 2, 5 est donc d'au plus $1+2\times 2+5=10$. Elle ne vaut pas 10 (sinon on remplace par une pièce de 10). Elle vaut donc au plus 9. Le nombre de pièces de 10 utilisées est donc égal à $\lfloor \frac{n}{10} \rfloor$, c'est à dire le nombre calculé par l'algorithme glouton.

   Pour payer le reste, c'est à dire $n := \mathrm{mod}(n,10)$, une solution optimale payera en pièces de 1 et 2 au plus $1+2\times 2=5$ chloubis. Cette somme est en fait < 5 (sinon …) et le nombre de pièces de 5 utilisés est donc $\lfloor \frac{n}{5} \rfloor$.

   De même pour le nombre pièces de 2.

3. L'algorithme est optimal dans ce cas également.

Xcas

```
money(n,p,k):={
local j,liste;
liste:=seq[];
pour j de k jusque 0 pas -1 faire
liste:=liste,(iquo(n,p^j));
n:=irem(n,p^j);
fpour;
return liste;
}:;
```

Une solution optimale utilise :

(a) au plus $p-1$ pièces valant p^{k-1} (sinon on en remplace p par une pièce de p^k).

(b) au plus $p-1$ pièces valant p^{k-2} (sinon on en remplace p par une pièce de p^{k-1}).

(c) ...

La somme payée avec les pièces de 1, p, ..., p^{k-1} est donc d'au plus :

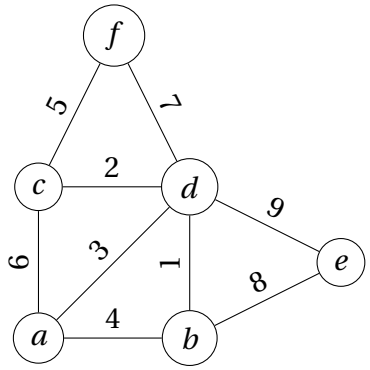
$$(p-1)(1+p+p^2+\cdots+p^{k-1}) = (p-1)\times\frac{p^k-1}{p-1} = p^k-1$$

le nombre de pièces de p^k chloubis utilisées dans une solution optimale pour payer n chloubis est donc $\lfloor\frac{n}{p^k}\rfloor$, c'est à dire le nombre déterminé par l'algorithme ...et ainsi de suite (lien avec algorithme des divisions en cascade pour l'écriture d'un nombre en base p).

4 Deux algorithmes sur des graphes

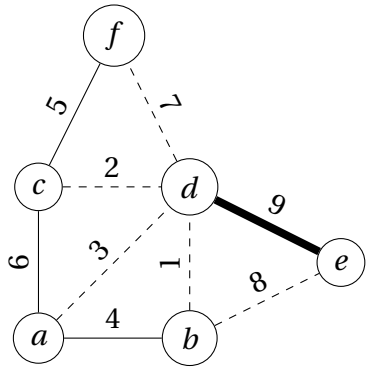
4.1 Couplage

Soit G un graphe, on appelle couplage tout ensemble d'arêtes tel que deux arêtes quelconques de cet ensemble ne sont jamais incidentes à un sommet commun. Les arêtes du graphe ci-dessous sont pondérées. On aimerait déterminer un couplage de poids maximal. Appliquer le principe glouton (c'est à dire : choisir l'arête de poids maximal, puis l'arête de poids maximal parmi les arêtes que l'on peut encore choisir ...). Aboutit-on à un couplage de poids maximal ?

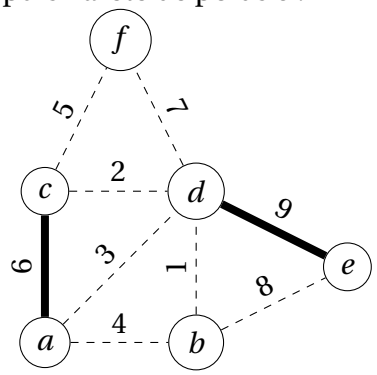


Une résolution

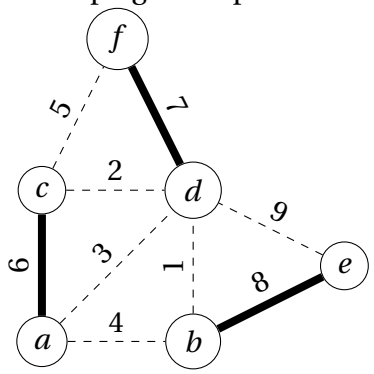
Le choix glouton mène à choisir d'abord l'arête de poids 9 :



puis l'arête de poids 6 :



Ce couplage a un poids de $9+6=15$ et n'est pas maximal :



4.2 Arbre de poids maximal

Algorithme de Kruskal.

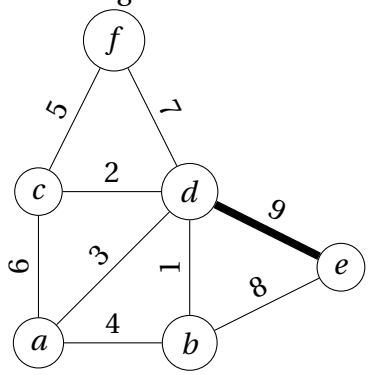
Un arbre dans un graphe G est un sous-graphe qui ne possède aucun cycle. Les arêtes de l' arbre ci-dessous sont pondérées. Le choix glouton (on choisit l'arête de poids maximal, puis l'arête de poids maximal parmi celles qui peuvent encore être choisies…) mène-t-il à un arbre de poids maximal ?

On admettra : « Tous les arbres couvrants d'un graphe connexe G ont le même nombre d'arêtes (à savoir $n-1$ où n est le nombre de sommets). »

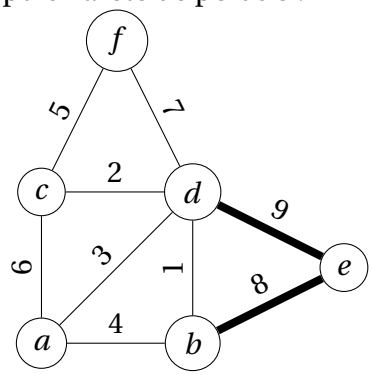
Une résolution

On choisit l'arête de poids maximal, puis l'arête de poids maximal parmi celles qui peuvent encore être choisies… : le principe est le même que pour le couplage mais la contrainte n'est plus la même…

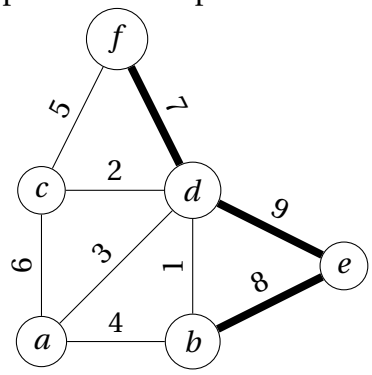
Le choix glouton mène à choisir d'abord l'arête de poids 9 :



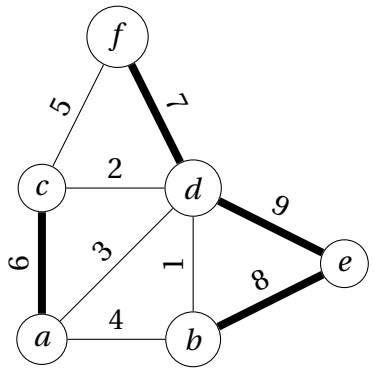
puis l'arête de poids 8 :



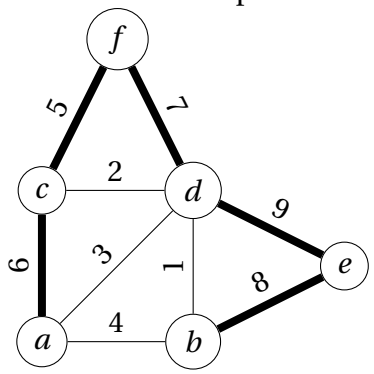
puis l'arête de poids 7 :



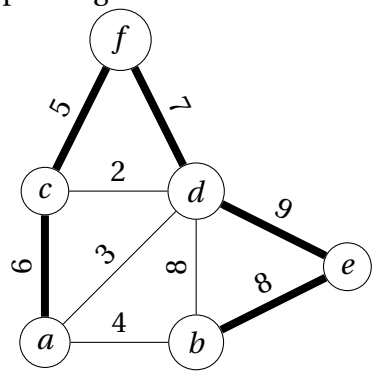
l'arête de poids 6 :



et enfin l'arête de poids 5 :



Cet algorithme porte le nom d'algorithme de Kruskal. Il mène toujours à un arbre couvrant de poids maximal (cf paragraphe suivant).
Ici le caractère maximal est évident puisque les poids des arêtes non choisies sont tous strictement inférieurs au min des poids des arêtes choisies, tout échange diminuerait donc le poids total.
On pourrait avoir une situation a priori plus ambiguë avec par exemple (le deuxième 8 n'est pas choisi par l'algorithme car il fermerait un cycle) :



4.3 Matroïde

1. Les couplages d'un graphe constituent une famille d'ensembles « héréditaire » : si un ensemble H d'arêtes est un couplage du graphe G alors tout ensemble H' d'arêtes contenu dans H est aussi un couplage du graphe G.

Les ensembles d'arêtes d'un graphe G engendrant un graphe sans cycle vérifient également cette propriété d'hérédité : si un ensemble H d'arêtes engendre un graphe sans cycle alors tout ensemble H' d'arêtes contenu dans H engendre aussi un graphe sans cycle.

2. Formulation de l'algorithme glouton pour un couple $(E, \mathscr{I})$ où E est un ensemble fini d'éléments (par exemple : l'ensemble des arêtes d'un graphe) et $\mathscr{I}$ une famille de parties de E héréditaire (par exemple, la famille des couplages ou la famille des graphes sans cycles d'un graphe). Les éléments de $\mathscr{I}$ seront nommés parties indépendantes.

Entrée	le couple $(E, \mathscr{I})$ et une fonction poids w (à valeurs positives) définie sur E.
Traitement	$J := \varnothing$, $A := E$ TantQue $A \neq \varnothing$ $\quad e :=$ élément de A de poids maximal $\quad A := A - e$ $\quad$ Si $J + e$ est une partie indépendante alors $J := J + e$ FinTantQue
Sortie	J

L'algorithme se termine toujours (puisque E est fini). Sous certaines conditions (voir ci-dessous), il donnera une partie indépendante de poids maximal.

3. La famille des ensembles d'arêtes d'un graphe engendrant un graphe sans cycle vérifie la propriété d'échange (on ne le démontrera pas ici) : « si H et H' sont des ensembles d'arêtes du graphe G engendrant un graphe sans cycle et si $\left|H'\right| < |H|$ alors il existe un élément e de H tel que $H' + e$ engendre un graphe sans cycle. »

   De façon plus générale, un système $(E, \mathscr{I})$ héréditaire sera appelé matroïde s'il vérifie la propriété d'échange : « si H et H' sont des parties indépendantes et si $\left|H'\right| < |H|$ alors il existe un élément e de H tel que $H' + e$ est une partie indépendante. »

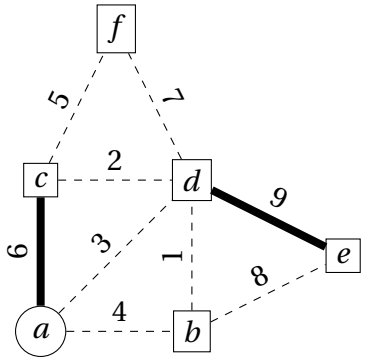
   (a) Vérifier que la famille des couplages d'un graphe ne possède pas la propriété d'échange.

   (b) Vérifier que pour un système héréditaire qui n'est pas un matroïde l'algorithme glouton énoncé ci-dessus peut ne pas mener à une partie indépendante de poids maximal.

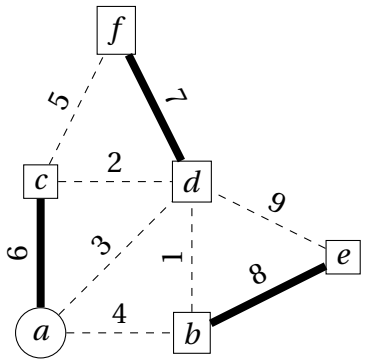
   (c) Vérifier que, pour un matroïde, l'algorithme glouton ci-dessus mène à une partie indépendante de poids maximal (et en particulier, l'algorithme de Kruskal donne un arbre couvrant de poids maximal).

Une résolution

a – Le couplage suivant



ne peut pas être « complété » en un couplage par une arête du couplage :



b – On a vu un exemple plus haut avec les couplages.
De façon plus générale :
Soit $(E,\mathscr{I})$ un système héréditaire ne vérifiant pas la propriété d'échange. Soient I et J deux parties indépendantes avec $|I|<|J|$ telles que pour tout $e\in J-I$, $I+e$ n'est pas une partie indépendante.
Notons $p=|I|$. Soit w définie sur E par :

$$w(e):=\begin{cases}p+2 \text{ pour } e\in I\\ p+1 \text{ pour } e\in J-I\\ 0 \text{ sinon}\end{cases}$$

L'algorithme va d'abord épuiser les éléments de I, toute arête éventuellement ajoutée ensuite sera de poids nul. L'algorithme construit donc une partie de poids $p(p+2)=p^2+2p$, ce qui n'est pas maximal puisque J pèse au moins $(p+1)^2=p^2+2p+1$.

c – Soit $I=\{e_1;e_2;\ldots;e_n\}$ une partie indépendante obtenue par l'application de l'algorithme (les éléments e_i étant numérotés dans l'ordre de leur choix dans l'application de l'algorithme, on a en particulier : $w(e_1)\geqslant w(e_2)\geqslant\cdots\geqslant w(e_n)$). Soit $J=\{f_1;f_2;\ldots;f_\ell\}$ une partie indépendante quelconque (on peut supposer $w(f_1)\geqslant w(f_2)\geqslant\cdots\geqslant w(f_\ell)$).
On a $\ell\leqslant n$ sinon on peut ajouter un f_j à I et l'algorithme ne se serait pas arrêté sur l'ensemble I obtenu.
- On a $w(e_1)\geqslant w(f_1)$ puisque l'algorithme a débuté en choisissant une arête de poids maximal ($\{e_1\}$ et $\{f_1\}$ sont des parties indépendantes par hérédité).
- Supposons que $w(e_1)+w(e_2)+\cdots+w(e_k)\geqslant w(f_1)+w(f_2)+\cdots+w(f_k)$. Il existe $v\in\{f_1;f_2;\ldots;f_k;f_{k+1}\}-\{e_1;e_2;\ldots;e_k\}$ tel que $\{e_1;e_2;\ldots;e_k\}\cup\{v\}$ est une partie indépendante.
  On a $w(e_{k+1})\geqslant w(v)$ puisque l'algorithme choisit e_{k+1} et $w(v)\geqslant w(f_{k+1})$.
  On a donc $w(e_1)+w(e_2)+\cdots+w(e_k)+w(e_{k+1})\geqslant w(f_1)+w(f_2)+\cdots+w(f_k)+w(f_{k+1})$.
- On a donc $w(I)\geqslant w(J)$.

5 Le sac à dos

5.1 Le sac à dos du cambrioleur

1. Un voleur dévalisant un magasin trouve n objets. L'objet numéro i vaut v_i euros et pèse w_i kg. Le voleur veut que son butin ait la plus grande valeur (en euros) possible mais ne peut pas emporter

plus de W kg dans son sac à dos.

Glouton 1 – Le résultat est-il optimal en choisissant l'objet le plus cher parmi ceux qui peuvent tenir dans le sac, puis le plus cher parmi ceux qui peuvent encore tenir …

Glouton 2 – Le résultat est-il optimal en choisissant d'abord les objets de plus grand prix au kg ?

2. Les objets sont maintenant des quantités fractionnables (par exemple 10 kg d'une certaine poudre). L'algorithme glouton consistant à charger dans le sac à dos la plus grande quantité du produit le plus cher au kg, puis la plus grande quantité du produit le plus cher au kg parmi ceux qui restent …donne-t-il une solution optimale ?

Une résolution

1. Glouton 1 – Par exemple, $W = 15$ kg. Avec 3 objets, un de poids 15 et de prix 100. Deux autres de poids 8 et 8 et de prix 60 et 60, on voit que cette première stratégie ne fonctionne pas.

   Glouton 2 – La stratégie n'est pas non plus nécessairement optimale. Avec $W = 5$:

Objet	1	2	3
prix en euro	5	8	9
masse en kg	1	2	3
prix par kg	5	4	3

   L'algorithme glouton mène à choisir objet 1 et objet 2 : valeur 13 euros

   tandis que le choix objet 2 et objet 3 donne la valeur 17 euros.

2. L'algorithme donne une solution optimale (la complexité de l'algorithme est donc environ celle d'un tri).

poudre n^o	1	2	3	…	n
quantité en kg dans une solution optimale	Q_1	Q_2	Q_3	…	Q_n
quantité en kg choisie par l'algorithme	q_1	q_2	q_3	…	q_n

   Le choix fait dans l'algorithme permet d'affirmer $q_1 \geqslant Q_1$.

   Si $q_1 > Q_1$, on remplace dans la solution supposée optimale, un poids $q_1 - Q_1$ pris dans les poudres 2, 3, …, n par le poids $q_1 - Q_1$ de poudre 1 (c'est possible puisque le poids q_1 de poudre était possible) et on obtient une valeur totale strictement plus grande. Contradiction. Donc $q_1 = Q_1$.

   Il nous reste à remplir $W - q_1$ kg avec les poudres 2,3, …, n. On procède par récurrence avec les mêmes arguments.

5.2 Le sac à dos de l'espion

La présentation du sac à dos dans cette section a été utilisée dans un algorithme de chiffrement (Merkle-Hellman) pour lequel on pourra trouver le principe détaillé ici :
`http://www.apprendre-en-ligne.net/crypto/menu/index.html`
On dispose d'une suite finie d'entiers s_1, s_2, …, s_k super-croissante, c'est à dire telle que $s_j > \sum_{\ell=1}^{j-1} s_\ell$ ($2 \leqslant j \leqslant k$). On se donne un entier $C > 0$ et on cherche si la somme de certains éléments s_j est égale à C, c'est à dire si l'on peut trouver $(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k) \in \{0;1\}^k$ tel que $\sum_{j=1}^{k} \varepsilon_j s_j = C$.
Montrer que l'algorithme glouton ci-dessous résout le problème :

Xcas

```
sacados(liste,C):={
// liste : séquence super croissante d'entiers
// C : somme à atteindre
local j, longueur_liste,solu,s;
longueur_liste:=size(liste);
solu:=seq[];
pour j de longueur_liste-1 jusque 0 pas -1 faire
s:=liste[j];
si s<=C alors solu:=solu,s; C:=C-s;fsi;
fpour;
si C==0 alors return solu;
sinon return "pas de solution";
fsi;}:;
```

sacados([13,45,183,315,801],511) renvoie par exemple 315,183,13.

Une résolution

1. Unicité de la solution lorsqu'elle existe.

   Soit $(x_1, x_2, \ldots, x_k)$ une solution du pb de sac à dos, c'est à dire un élément de $\{0;1\}^k$ tel que $x_1\,s_1 + x_2\,s_2 + \cdots + x_k\,s_k = C$. Supposons que l'on dispose d'une autre solution $(y_1, y_2, \ldots, y_k)$.

   On a

   $$x_1\,s_1 + x_2\,s_2 + \cdots + x_k\,s_k = y_1\,s_1 + y_2\,s_2 + \cdots + y_k\,s_k$$

   (a) Supposons que $x_k = 1$ et $y_k = 0$. Comme s est supercroissante, on aurait alors

   $$x_1\,s_1 + x_2\,s_2 + \cdots + x_k\,s_k \geqslant s_k > s_1 + s_2 + \cdots + s_{k-1} \geqslant y_1\,s_1 + y_2\,s_2 + \cdots + y_{k-1}\,s_{k-1} = y_1\,s_1 + y_2\,s_2 + \cdots + y_k\,s_k$$

   d'où une contradiction.

   On établit de même que $x_k = 0$ et $y_k = 1$ est impossible. On a donc $x_k = y_k$.

   (b) On a donc l'égalité suivante :

   $$x_1\,s_1 + x_2\,s_2 + \cdots + x_{k-1}\,s_{k-1} = y_1\,s_1 + y_2\,s_2 + \cdots + y_{k-1}\,s_{k-1}$$

   Le même raisonnement montre que l'on a $x_{k-1} = y_{k-1}$.

   (c) ...

   La solution est donc unique.

2. Dans le cas où le sac à dos proposé n'a pas de solution, l'algorithme renverra "pas de solution". Dans le cas où le sac à dos proposé a une solution (nécessairement unique puisque la séquence est supercroissante), il s'agit d'établir que l'algorithme donnera toujours cette solution. Notons $(x_1, x_2, \ldots, x_k) \in \{0;1\}^k$ cette solution. On pose pour la preuve $\varepsilon_i = 0$ lorsque s_i n'est pas dans la liste solu construite par l'algorithme et $\varepsilon_i = 1$ lorsque s_i est dans la liste solu construite par l'algorithme.

   (a) L'algorithme commence par comparer s_k et C. Si $s_k > C$, l'algorithme renvoie $\varepsilon_k = 0$ (c'est à dire n'ajoute pas l'élément s_k dans la liste solu) et il est clair que l'on a également $x_k = 0$ (sinon la somme $\sum_j x_j\,s_j$ est strictement plus grande que C).

Si $s_k \leqslant C$, l'algorithme renvoie $\varepsilon_k = 1$ (c'est à dire ajoute l'élément s_k à la liste solu). Et on a également $x_k = 1$, sinon avec la supercroissance :

$$\sum_{j=1}^{k} x_j\, s_j = \sum_{j=1}^{k-1} x_j\, s_j < s_k \leqslant C.$$

Dans tous les cas, nous voyons que $\varepsilon_k = x_k$.

(b) L'algorithme transforme C en $C - \varepsilon_k\, s_k$ et va donc maintenant résoudre le problème du sac à dos avec la séquence supercroissante $(x_1, x_2, \ldots, x_{k-1})$ et la somme $C - \varepsilon_k\, s_k$. $(x_1, x_2, \ldots, x_{k-1})$ est évidemment l'unique solution de ce sac à dos. Et le même raisonnement qu'à l'étape précédente montre que $\varepsilon_{k-1} = x_{k-1}$.

(c) ...

Ainsi l'algorithme trouve l'unique solution lorsqu'il y en a une.

6 Coloration des sommets d'un graphe

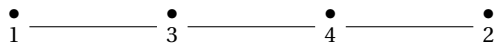
On cherche à obtenir une coloration des sommets d'un graphe qui satisfasse à la contrainte suivante : deux sommets voisins n'ont jamais la même couleur. Une question se pose : quel est le plus petit nombre de couleurs permettant de colorier les sommets d'un graphe sous cette contrainte ? (ce plus petit nombre est appelé nombre chromatique du graphe).
On considère l'algorithme suivant :

Donnée – un graphe G et des couleurs 1,2,3,4...Les sommets de G sont numérotés de 1 à n $(s_1, s_2, \ldots, s_n)$.
Processus – pour i allant de 1 à n, affecter au sommet s_i la plus petite couleur non déjà affectée à ses voisins déjà coloriés (c'est-à-dire la plus petite couleur non déjà affectée à ceux des sommets $s_1, s_2, \ldots, s_{i-1}$ qui lui sont adjacents). En d'autres termes, on gloutonne : on prend localement le plus petit nombre possible.
Sortie – Une coloration valide du graphe G. Mais le nombre de couleurs utilisées est-il minimal ?

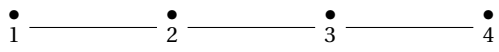
6.1 L'algorithme ne fournit pas nécessairement une coloration optimale

1. Appliquer cet algorithme au graphe ci-dessous avec les deux numérotations des sommets proposées :

   (a)

   (b)

2. Cet algorithme donne-t-il le nombre chromatique du graphe ?

Une résolution

1. (a)

(b)

2. L'algorithme ne donne donc pas toujours le nombre chromatique, mais on peut démontrer qu'il existe toujours une numérotation des sommets telle que la coloration donnée par l'algorithme soit optimale et donne donc le nombre chromatique (voir plus loin).

6.2 Nombre maximal de couleurs utilisées par l'algorithme

Montrer que cet algorithme donnera toujours une coloration utilisant au plus $\Delta(G)+1$ couleurs (où Δ désigne le degré maximal des sommets).

Une résolution

Lorsqu'on colorie les sommets suivant cet algorithme, le numéro de la couleur attribuée au sommet s_i est d'au plus

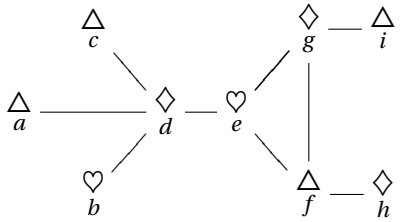
$$1+\left|\{s_1,s_2,\ldots,s_{i-1}\}\cap \text{Voisinage}(s_i)\right|$$

et

$$1+\left|\{s_1,s_2,\ldots,s_{i-1}\}\cap \text{Voisinage}(s_i)\right| \leqslant 1+\Delta$$

6.3 L'algorithme peut donner une coloration optimale

1. Montrer que la coloration du graphe ci-dessous est optimale mais qu'elle ne peut pas être obtenue par l'algorithme.



2. Établir qu'il existe toujours une numérotation initiale de G telle que l'application de l'algorithme donne une coloration optimale (c'est à dire avec $\chi(G)$ couleurs).

Une résolution

1. La présence d'un triangle assure que $\chi \geqslant 3$ et la coloration proposée utilisant 3 couleurs, elle est optimale. Mais cette coloration ne peut être obtenue par l'algorithme décrit ci-dessus, car dans toute application de cet algorithme les sommets b et c ont la même couleur. En effet si l'un des deux est coloré avant d alors il prend la couleur numéro 1 et l'autre qu'il soit coloré avant ou après d la prendra aussi (d prenant nécessairement une autre couleur puisqu'adjacent à un sommet déjà coloré en couleur 1). Si d est coloré avant les sommets b et c, soit d a la couleur numéro 1 et alors b et c prennent la couleur numéro 2, soit d n'a pas la couleur numéro 1 et alors b et c la prendront.

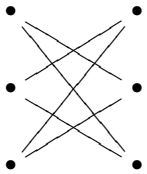
2. Supposons disposer d'une coloration optimale de G par les couleurs c_1, c_2, …, $c_{\chi(G)}$. On numérote d'abord les sommets de couleur c_1, puis les sommets de couleur c_2, …L'application de l'algorithme aura alors pour effet de colorier de la couleur 1 les sommets qui étaient de couleur c_1, de la couleur 2 ou de la couleur 1 les sommets qui étaient de couleur c_2, de la couleur 3 ou 2 ou 1 ceux qui étaient de couleur c_3 …On utilisera donc au plus $\chi(G)$ couleurs.

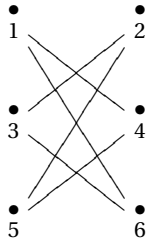
   Cette remarque ne rend pas plus opérationnel l'algorithme présenté : il y a en effet $n!$ numérotations initiales des sommets et les essayer toutes donnent rapidement des temps de calcul beaucoup trop longs.

6.4 Une très mauvaise coloration ?

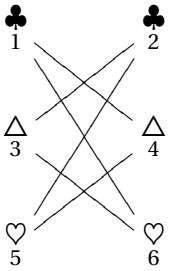
L'algorithme présenté ne donne pas nécessairement une coloration optimale. Mais une très mauvaise coloration (du point de vue du nombre de couleurs utilisées) est-elle possible ?
Soit $k \geqslant 2$ un entier. Construire un graphe G de nombre chromatique 2 et une numérotation des sommets de ce graphe G telle que l'application de l'algorithme précédent donne une coloration avec k couleurs.

Une résolution
Avec le graphe suivant :

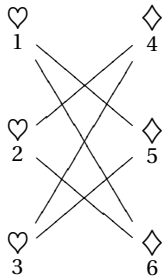




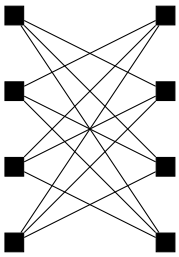
donne la coloration suivante en trois couleurs :



alors que deux couleurs suffisent et peuvent s'obtenir avec le même algorithme en numérotant ainsi :



Sur le même modèle, une numérotation "ligne par ligne" du graphe suivant :



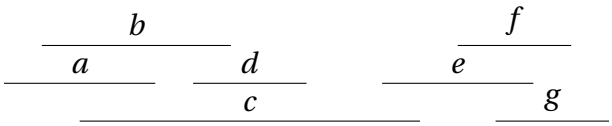
donnera une coloration avec 4 couleurs tandis qu'une numérotation par colonne donnera une coloration optimale en deux couleurs.
Le lecteur généralisera aisément.

6.5 Graphe d'intervalles

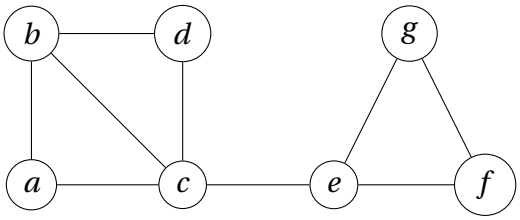
On reprend le problème du gymnase. On cherche maintenant le nombre minimal de gymnases permettant le déroulement de tous les événements.

Glouton 1 – Les dates de fin au plus tôt ayant permis le déroulement d'un nombre optimal d'épreuves avec un seul gymnase, on essaie de "remplir" le gymnase 1 au maximum avec ce principe, puis on passe à un gymnase 2, puis . . .Le résultat sera-t-il optimal ?

Glouton 2 – On représente par exemple les intervalles de temps :



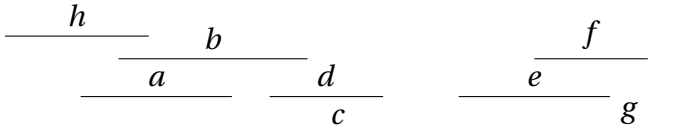
par le graphe :



Montrer qu'une coloration du graphe par l'algorithme glouton décrit plus haut utilise un nombre de couleurs égal au nombre chromatique avec une numérotation des sommets correspondant à l'ordre des extrémités gauche des intervalles.

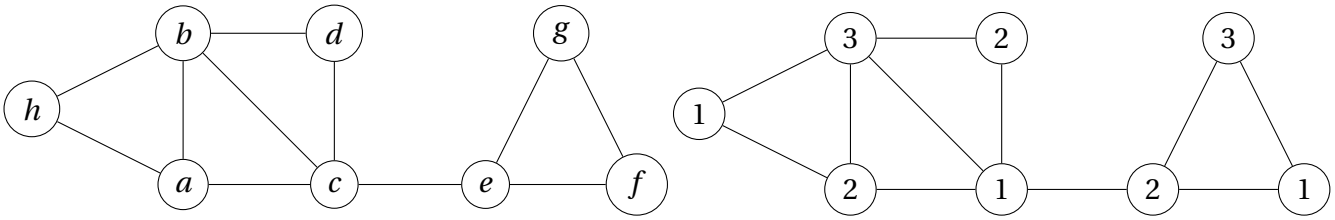
Une résolution

1. Le résultat n'est pas optimal. Contre-exemple :

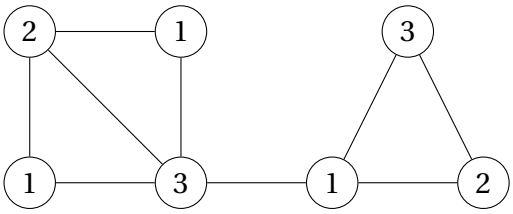


Le gymnase 1 verra se dérouler les épreuves : h, d, e,
le gymnase 2 : a, f
le gymnase 3 : b, g
le gymnase 4 : c.

Alors que 3 gymnases suffisent : h, c, f ; a, d, e ; b, g. Cette deuxième répartition est celle que l'on obtient avec le glouton 2 :



2. Sur l'exemple, ce choix donne la coloration :



Lorsqu'un sommet v reçoit une couleur k, c'est que l'extrémité gauche de l'intervalle v appartient à $k-1$ intervalles (sinon on pourrait choisir une couleur $\leqslant k-1$ pour v). Tous ces intervalles s'intersectent donc (l'extrémité gauche de v est commun à tous) et le graphe présente donc une clique d'ordre $\geqslant k$, donc le nombre chromatique est d'au moins k.

Lors du déroulement des JO, remplir au mieux le gymnase "principal" n'est donc pas compatible a priori avec l'utilisation d'un nombre minimal de gymnases.

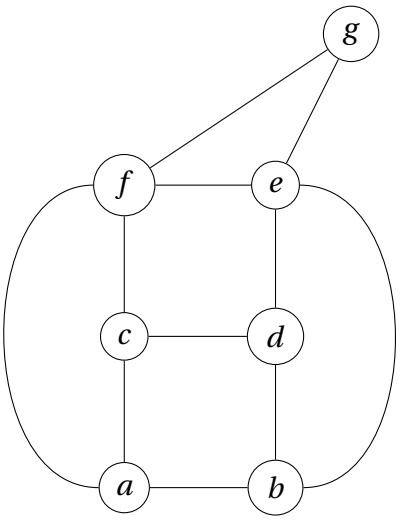
6.6 Algorithme de Brelaz

On définit, à toute étape de l'algorithme, le degré-couleur d'un sommet comme le nombre de couleurs déjà utilisées pour ses voisins.

Donnée – un graphe G et des couleurs 1,2,3,4...(les degrés-couleur sont initialisés à 0).
Processus – Prendre parmi les sommets de degré-couleur maximal un sommet de degré maximal, lui attribuer la plus petite couleur possible. Mettre à jour les degrés-couleur.
Sortie – Une coloration valide du graphe G. Mais le nombre de couleurs utilisées est-il minimal ?

Exercice

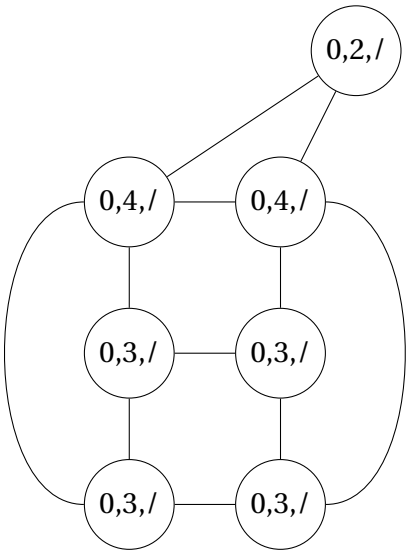
1. Appliquer l'algorithme au graphe ci-dessous :



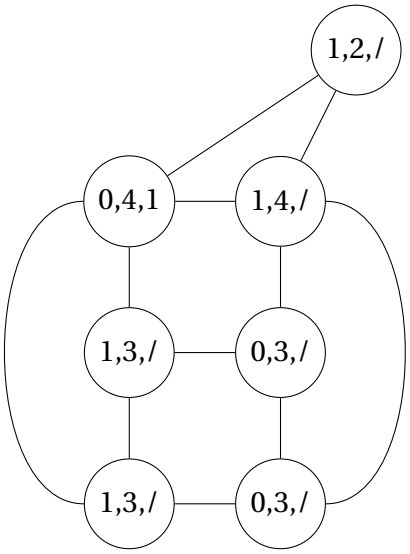
2. Montrer que la coloration obtenue n'est pas nécessairement optimale (c'est à dire : peut demander un nombre de couleurs strictement supérieur au nombre chromatique).
3. Montrer que l'algorithme de Brelaz colore les graphes bipartis en deux couleurs.

Une résolution

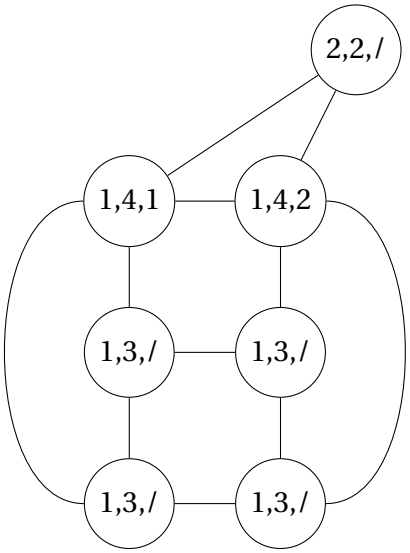
1. On considère le graphe ci-dessous.
   A chaque étape, on indique le triplet degré-couleur, degré, couleur :



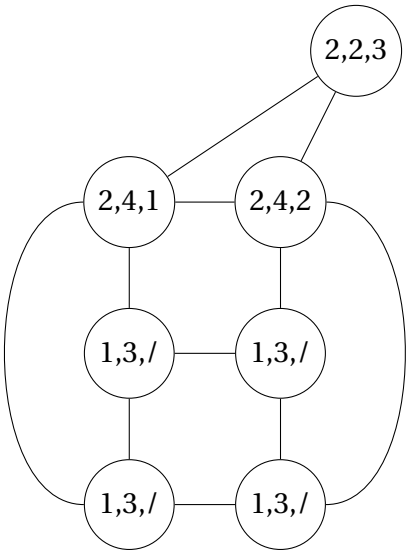
   On choisit un sommet de degré maximal, par exemple f :



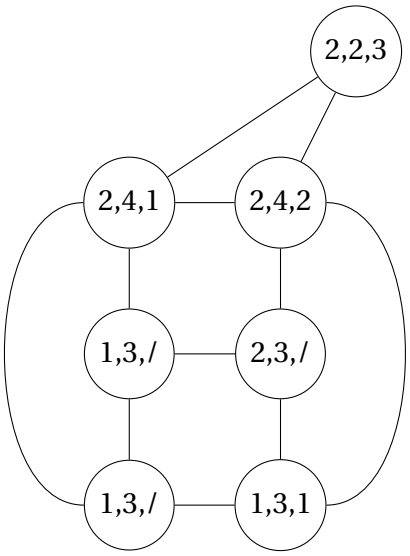
On doit choisir ensuite e :



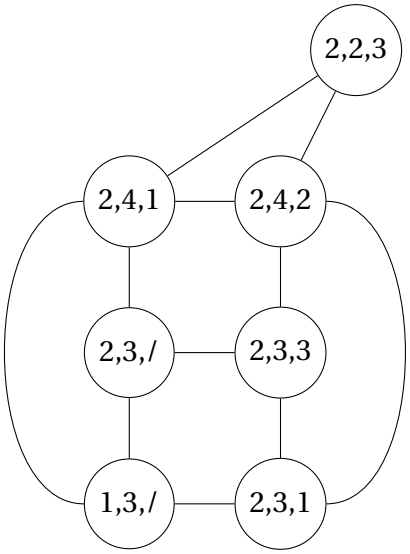
On doit ensuite choisir g :



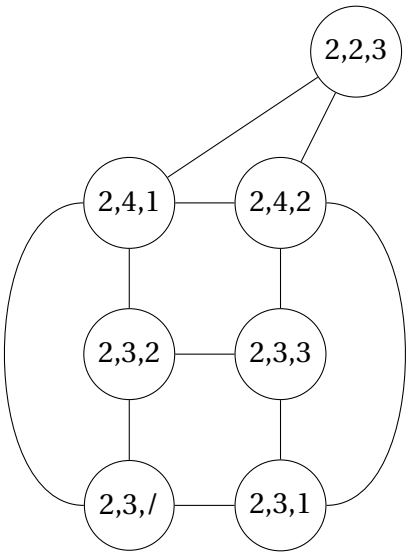
On peut choisir le suivant au hasard parmi les quatre non colorés, par exemple b :



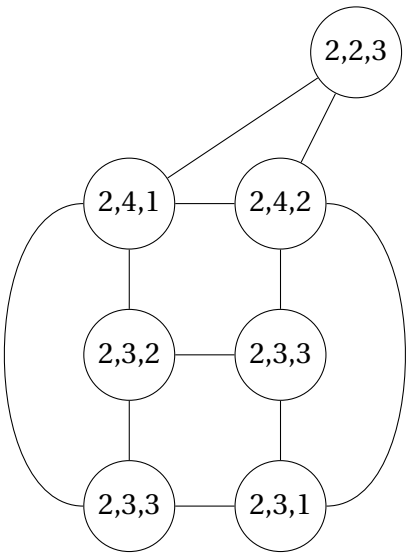
On doit ensuite choisir d :



puis c :

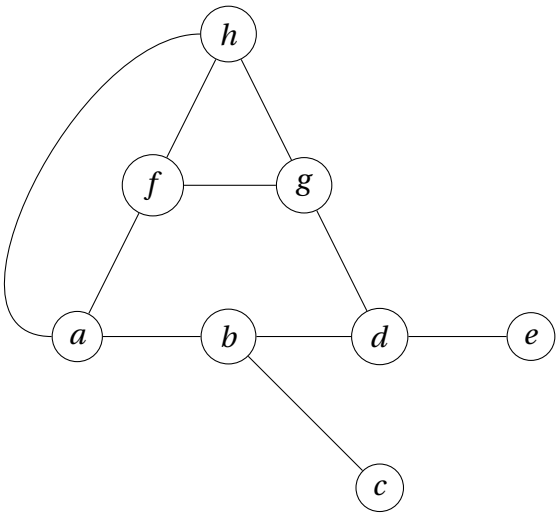


puis a :

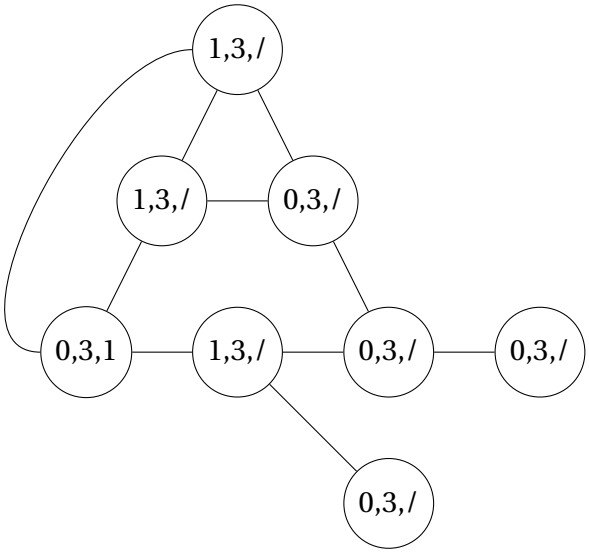


La coloration est évidemment optimale (présence d'un triangle).

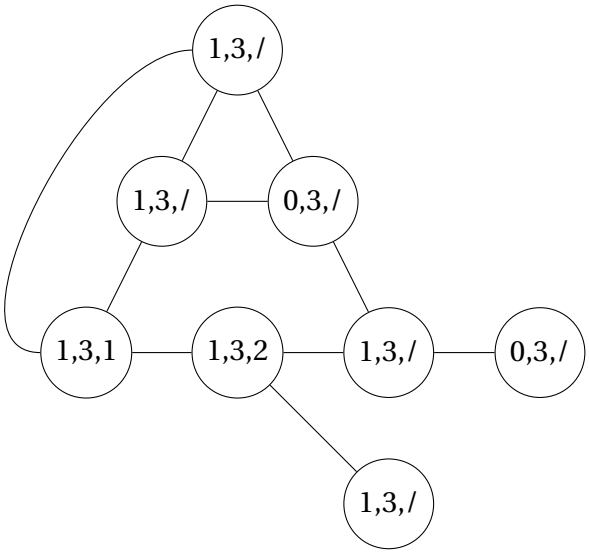
2. On considère le graphe ci-dessous :



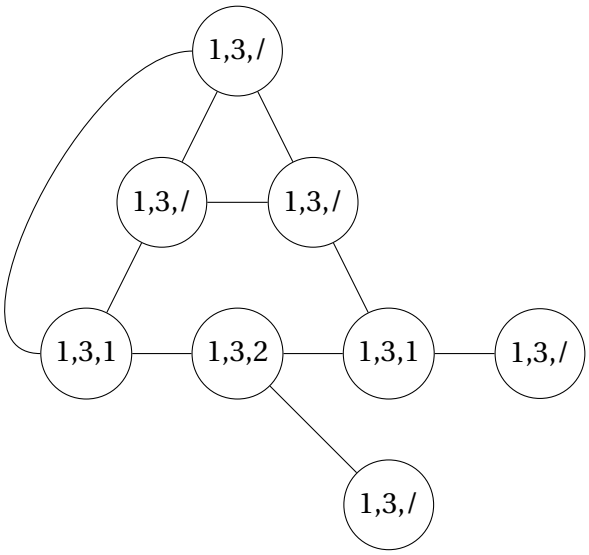
Choisissons a en premier :



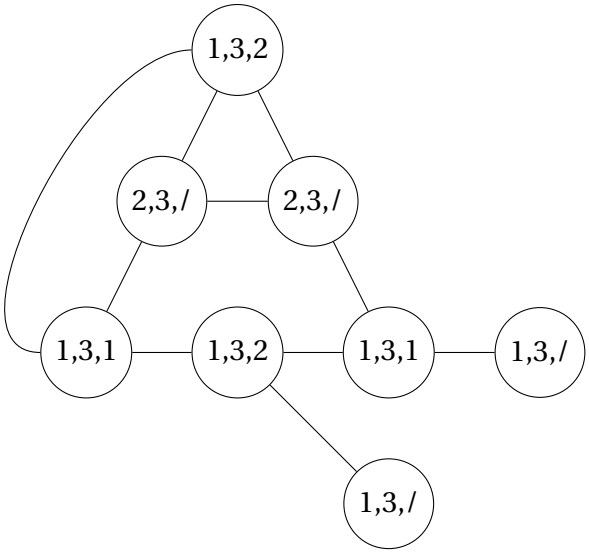
Puis b :



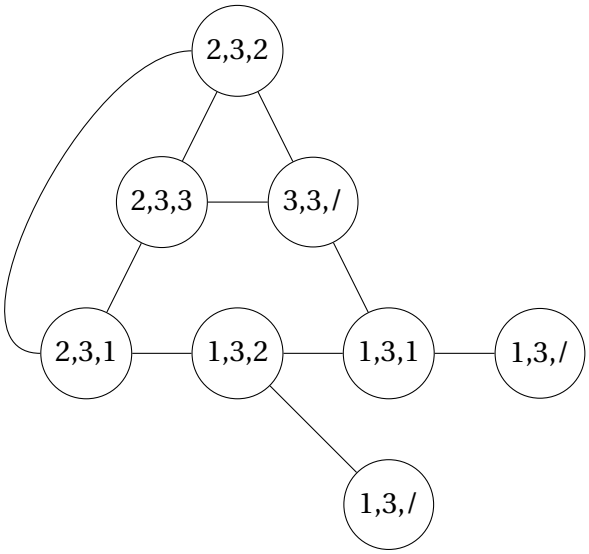
puis d :


1,3,/
1,3,/
1,3,/
1,3,1
1,3,2
1,3,1
1,3,/
1,3,/

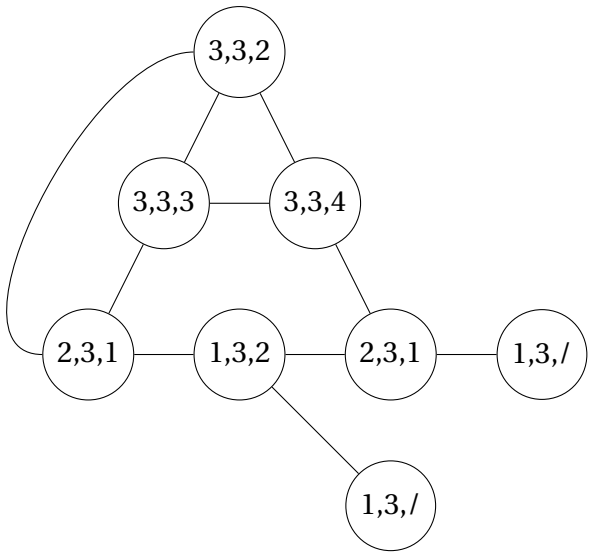
Puis h :


1,3,2
2,3,/
2,3,/
1,3,1
1,3,2
1,3,1
1,3,/
1,3,/

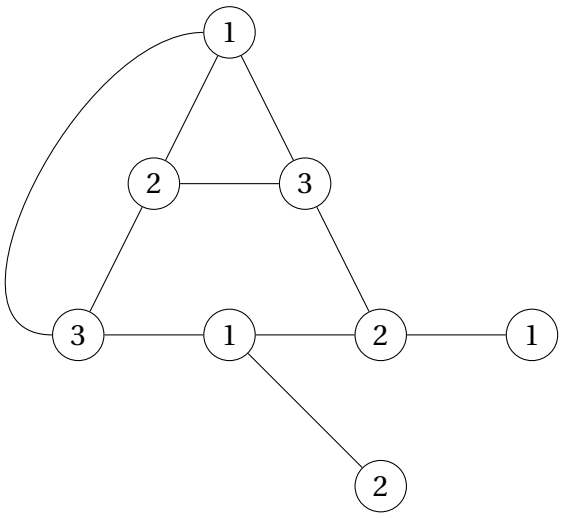
puis f :


2,3,2
2,3,3
3,3,/
2,3,1
1,3,2
1,3,1
1,3,/
1,3,/

puis g :



On a donc utilisé 4 couleurs, alors que 3 suffisent :



3. Soit G un graphe biparti (connexe, sinon on raisonne composante par composante). On colorie, en couleur 1, un sommet de degré maximal (parti A). Puis on colorie nécessairement un sommet voisin (degré-couleur 1), du parti B, en une couleur 2.

   A une étape k, supposons que tous les sommets coloriés de A sont en couleur 1 et tous les sommets coloriés de B sont en couleur 2. Les degrés-couleur sont donc tous $\leqslant 1$. Parmi les sommets non coloriés, certains sont voisins d'un sommet colorié (connexité) et sont de degré-couleur 1. L'algorithme choisit un non colorié de degré-couleur 1, il sera colorié en couleur 1 s'il est dans A (car il est voisin de sommets de B mais pas de A) et en couleur 2 s'il est dans B (car il est voisin de sommets de A mais pas de B) .