Cours d'ALGOrithmique AVancée

Tiffany Grenier

septembre-décembre 2013
(dernière révision : 28 décembre 2013)

Merci à Salah Eddine Alaoui, Maxim Vasilishin et Cyrille Pressat dont les notes de cours et de travaux dirigés m'ont servi à compléter les miennes pour la création de ce fichier.

Sommaire

0..0..1 Rappel : Notations mathématiques courantes utilisées

- Si S est un ensemble, |S| désigne le nombre d'éléments de S
- En logique booléenne, $\wedge$ signifie "ET", tandis que $\vee$ signifie "OU"

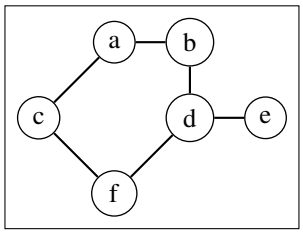
Chapitre 1 : Graphes

1.I Définitions de graphes

1.I.1 Introduction

1.I.1.a. **Graphe (définition générale)** *(Définition) :*
Un graphe G est un couple (V, E) où V (ou V(G) en cas de confusion possible) est l'ensemble des sommets et E (ou E(G) en cas de confusion possible) $\subset V \times V$ est l'ensemble des arêtes reliant des sommets.

Exemple 1.I.1.b



V = {a, b, c, d, e, f}
E = {(a; b), (a; c), (c; f), (b; d), (d; e), (d; f)}

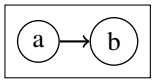
Il existe plusieurs variantes de graphes, présentées ci-après.

1.I.2 Graphes (non-)dirigés/(non-)orientés

1.I.2.a. **Graphe non-dirigé/non-orienté** *(Définition) :*
Dans un graphe non orienté, les arêtes sont symétriques. (a; b) et (b; a) désignent la même arête.

1.I.2.b. **Graphe dirigé/orienté** *(Définition) :*
Dans un graphe orienté (également appelé graphe dirigé), les arêtes ont chacune une direction.

Exemple 1.I.2.c



est l'arête (a; b). a *en est le sommet source et* b *le sommet cible.*
Le graphe orienté ci-dessus ne contient pas l'arête (b; a).

1.I.3 Graphes avec ou sans boucles/auto-arêtes

1.I.3.a. **Graphe avec boucles/auto-arêtes et graphe sans boucle/auto-arête** *(Définition) :*
Un graphe avec boucles (ou auto-arêtes) est un graphe dans lequel sont permises les arêtes dont les deux extrémités sont identiques. On peut donc trouver des arêtes (a; a) dans un tel graphe.
On appelle graphe sans boucle un graphe dans lequel il ne peut y avoir d'auto-arêtes.

1.I.4 Graphes multiples ou graphes simples

1.I.4.a. **Graphe multiple et graphe simple** *(Définition) :*
Un graphe multiple est un graphe pour lequel il peut exister plusieurs arêtes entre deux sommets donnés. Pour de tels graphes, la désignation d'une arête ne peut se faire uniquement par les noms des sommets qu'elle relie. A contrario, un graphe pour lequel il ne peut y avoir plus d'une arête entre deux sommets donnés est appelé un graphe simple.

1.I.5 Graphes valués et graphes colorés

1.I.5.a. **Graphe sommets-valué et graphe sommets-coloré** *(Définition) :*
Un graphe sommets-valué est un graphe dont les sommets possèdent un vecteur de poids quantitatif.
Un graphe sommets-coloré est un graphe dont les sommets possèdent un vecteur de poids qualitatif.

1.I.5.b. **Graphe arêtes-valué et graphe arêtes-coloré** *(Définition) :*
Un graphe arêtes-valué est un graphe dont les arêtes possèdent un vecteur de poids quantitatif.
Un graphe arêtes-coloré est un graphe dont les arêtes possèdent un vecteur de poids qualitatif.

1.I.5.c. **Graphe non-valué et graphe non-coloré** *(Définition) :*
Un graphe non-valué est un graphe dont ni les sommets ni les arêtes ne possèdent de vecteur de poids quantitatif.
Un graphe non-coloré est un graphe dont ni les sommets ni les arêtes ne possèdent de vecteur de poids qualitatif.

1.I.6 Graphes finaux

1.I.6.a. **Graphe final** *(Définition) :*
Un graphe final est un graphe qui évolue dans le temps.

1.I.7 Exemples de types de graphes

Exemple 1.I.7.a (*de l'étude des communications dans un réseau de serveurs interconnectés physiquement*)

- *serveurs = sommets*
- *câbles = arêtes*
- *pas de boucle possible*
- *pas de direction dans les connexions*
- *pas plus d'un câble entre deux serveurs*
- *capacité de communication des câbles = poids des arêtes*

$\Longrightarrow$ *graphe simple non-dirigé arête-valué sans boucles.*

Exemple 1.I.7.b (*de l'éude des interactions entre protéines*)

- *protéines = sommets*
- *capacité de liaison = arêtes*
- *pas de boucle possible, ni de duplication de l'information*
- *pas de direction*

$\Longrightarrow$ *graphe simple non-dirigé non-valué non-coloré sans boucles.*

Exemple 1.I.7.c (*de l'étude de l'ordonnancement de tâches*)

- *tâches à effectuer = sommets*
- *succession de tâches = arêtes*
- *répétitions possibles*
- *ordre = direction des arêtes*
- *durée des tâches = poids quantitatif sur les sommets*
- *possibilité plusieurs options coexistantes*

$\Longrightarrow$ *graphe multiple dirigé sommet-valué avec boucles.*

1.I.8 Sous-graphes

1.I.8.a. **Sous-graphe** *(Définition) :*
Un sous-graphe H d'un graphe G est un graphe tel que V(H) ⊂ V(G) et E(H) ⊂ E(G).

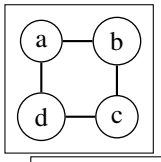
1.I.8.b. **Sous-graphe induit** *(Définition) :*
Soit H un sous-graphe de G. H est induit par un sous-ensemble A de sommets de G si et seulement s'il est défini par l'ensemble de sommets A et l'ensemble de toutes les arêtes de G reliant des sommets éléments de A. On note H = G[A].

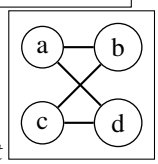
1.II Représentations de graphes

1.II.1 Graphiquement

Les sommets d'un graphe sont représentés graphiquement par des points, et ses arêtes par des traits. Dans un graphe non-orienté, les arêtes sont des segments d'extrémités les sommets qu'elles relient, tandis que pour un graphe non-orienté, les arêtes sont des flèches allant de leur source à leur cible.
La représentation graphique d'un graphe n'est pas unique.

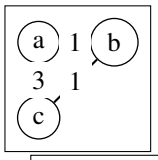
Exemple 1.II.1.a

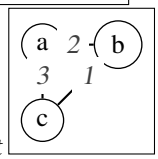




et *représentent le même graphe.*

Exemple 1.II.1.b



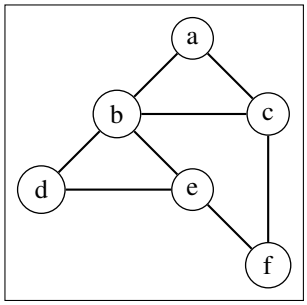


et *ne sont pas isomorphes.*

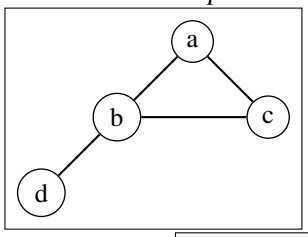
1.II.1.c. **Graphes isomorphes** *(Définition) :*
Deux graphes G et H sont isomorphes si et seulement s'il existe une bijection $\varphi : V(G) \to V(H)$ telle que $\forall (u, v) \in V(G)^2, (u, v) \in E(G) \iff \varphi(U, v) \in E(H)$ et que le poids de (u, v) est le même que celui de $\varphi(u, v)$ si le graphe est arête-valué.
Concrètement, deux graphes isomorphes ont la même structure.

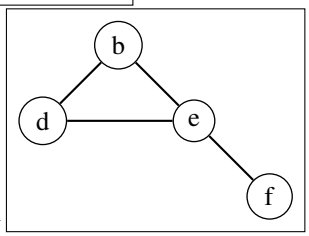
Exemple 1.II.1.d

Soit le graphe suivant G *où les sommets représentent des individus présents à une soirée et les arêtes représentent les liens d'amitié (supposés réciproques) existant entre ces personnes.*



*(*b *est probablement l'organisateur de la soirée)*
On s'intéresse aux présentations d'amis possibles.





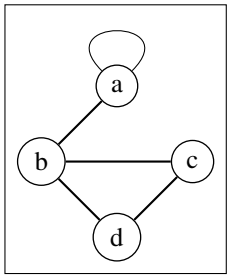
*(*G[a, b, c, d]*) et (*G[b, d, e, f]*) sont isomorphes mais représentent des réalités différentes.*

1.II.2 Informatiquement (mathématiquement)

1.II.3 Matrice d'adjacence

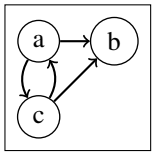
1.II.3.a. **Matrice d'adjacence** *(Définition) :*
La matrice d'adjacence A d'un graphe G est une matrice de taille $|V(G)| \times |V(G)|$ telle que pour tout couple $\{u; v\}$ de sommets, $m_{uv} = 1$ si et seulement s'il y a dans G une arête de u vers v ; et $m_{uv} = 0$ sinon.

Exemple 1.II.3.b



$$M = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Exemple 1.II.3.c



$$M = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

1.II.3.c.1 Remarques :

– Dans le cas d'un graphe non-orienté, la matrice est symétrique car l'arête (u, v) est identique à l'arête (v, u)
– Si G est simple et arête-valué, m_{uv} vaut le poids de l'arête (u, v).
– Si G est multiple et non-arête-valué, on assimile le nombre d'arête entre u et v à un poids attribué au couple $\{u; v\}$ afin de traiter le graphe comme un graphe simple arête-valué.

1.II.4 Liste d'arêtes

Le graphe est codé par une matrice de taille $|E(G)| \times 2$ où chaque ligne correspond à une arête. Dans le cas d'un graphe arête-valué, on peut aussi ajouter une colonne à cette matrice afin qu'elle contienne aussi les valeurs attribuées aux arêtes.
Avec ce système, on ne garde que l'information nécessaire, ce qui représente un fort gain d'efficacité par rapport à la matrice d'adjacence pour les graphes creux (avec peu d'arêtes relativement au nombre de sommets).

1.II.4..2 Remarque : Il s'agit bien d'une liste et non d'un ensemble. La redondance est par conséquence permise.

1.II.5 Liste de voisinage

On code le graphe par une liste de $|V(G)|$ vecteur associant à chaque sommet la liste de ses sommets voisins, ou la liste des couples (voisins ; poids des arêtes).

1.III Densité, voisinage, degré

1.III.1 Densité d'un graphe

1.III.1.a. **Densité** *(Définition) :*
La densité d'un graphe est le rapport entre le nombre d'arêtes qu'il contient et le nombre d'arête qu'il pourrait contenir. Elle a toujours une valeur située entre 0 et 1[1].

Dans le cas d'un graphe G simple, non-orienté, non-valué, non-coloré et sans boucles, on a :
$d(G) = \frac{e}{\frac{n(n-1)}{2}} = \frac{2e}{n(n-1)}$ où $n = |V(G)|$ et $e = |E(G)|$, car il y a $C_n^2 = \binom{n}{2} = \frac{n(n-1)}{2}$ arêtes possibles dans un tel graphe.

1.III.1.b. **Clique (ou graphe complet)** *(Définition) :*
Une clique est un graphe de densité 1 : il existe une arête entre chaque paire de sommets (tous ses sommets sont reliés deux à deux par une arête).

1.III.2 Voisinage et degré

1.III.2.a. **Voisinage d'un sommet** *(Définition) :*
Dans le graphe G quelconque, le voisinage N(v) du sommet v est l'ensemble des sommets voisins de v.

1.III.2.b. **Degré d'un sommet** *(Définition) :*
Dans le graphe G quelconque, le degré d(v) du sommet v est le nombre des voisins de v, c'est-à-dire que $d(v) = |N(v)|$.

1.III.2.c Cas des graphes dirigés

1.III.2.d. **Voisinage externe** *(Définition) :*
Dans le graphe G orienté, on appelle le voisinage externe du sommet v, noté $N^+(v)$, l'ensemble des sommets u vérifiant $(v;u) \in E(G)$, c'est-à-dire l'ensemble des voisins cibles d'une arête dont v la source.

1.III.2.e. **Voisinage interne** *(Définition) :*
Dans le graphe G orienté, on appelle le voisinage externe du sommet v, noté $N^-(v)$, l'ensemble des sommets u vérifiant $(u;v) \in E(G)$, c'est-à-dire l'ensemble des voisins sources d'une arête dont v la cible.

1.III.2.f. **Degré sortant** *(Définition) :*
On appelle degré sortant de v le nombre de ses voisins externes : $d^+(v) = \left|N^+(v)\right|$.

1.III.2.g. **Degré entrant** *(Définition) :*
On appelle degré sortant de v le nombre de ses voisins internes : $d^-(v) = |N^-(v)|$

Propriété #1.III.2.g.1 :
$N(v) = N^+(v) \cup N^-(v)$ et $d(v) = d^+(v) + d^-(v)$.

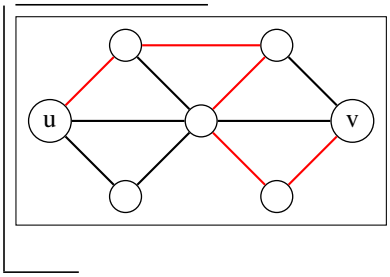
1.IV Chemin, connexité, distance

1.IV.1 Chemins, marches et cycles

1.IV.1.a. **Chemin** *(Définition) :*
Soient G un graphe et u et v deux sommets de ce graphe. Un chemin de u à v de longueur $k \in \mathbb{N}$ est une suite de k + 1 sommets distincts deux à deux tels que le premier soit u, le dernier soit v, il existe une arête entre chaque sommet et son successeur. On note ce chemin $(u_0, ;u_{k+1})$ avec :

1. $u_0 = u$, $u_{k+1} = v$
2. $\forall i \in [0;k] \cap \mathbb{N}, (u_i;u_{i+1}) \in E(G)$
3. $\forall i \in [0;k] \cap \mathbb{N}$, les u_i sont tous distincts deux à deux

Exemple 1.IV.1.b

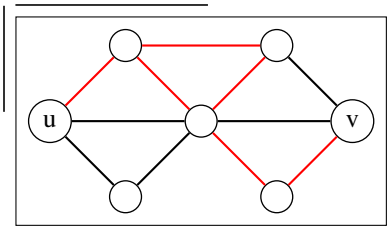


1.IV.1.b.1 Remarques :

- La longueur d'un chemin est son **nombre d'arêtes**
- Il peut y avoir plusieurs chemins entre u et v
- u est appelé source du chemin et v cible du chemin
- Un chemin orienté est un chemin d'un graphe orienté pour lequel chaque arête ne peut être parcourue que de la source vers la cible (elles doivent être orientée d'un sommet de la liste vers son successeur dans cette même liste)

1.IV.1.c. **Marche** *(Définition) :*
Si l'on omet la 3e condition de la définition d'un chemin, on parle d'une marche entre u et v.

Exemple 1.IV.1.d



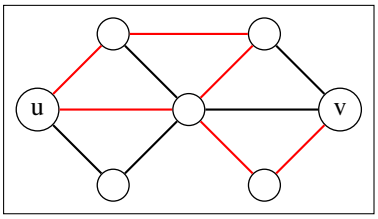
1.IV.1.d.1 Remarque :

– Tout chemin est une marche
– S'il existe une marche entre u et v,on peut la réduire en un chemin en éliminant récursivement les boucles

1.IV.1.e. **Cycle** *(Définition) :*
Un cycle de longueur k est une liste de sommets $(u_0, ;u_{k+1})$ telle que

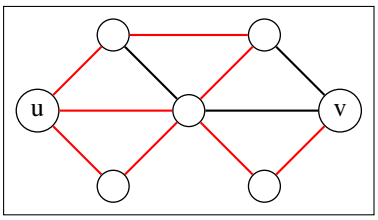
1. $u_0 = u_{k+1}$ et c'est la seule répétition parmi les u_i
2. $\forall i \in [0; k] \cap \mathbb{N}, (u_i; u_{i+1}) \in E(G)$

Exemple 1.IV.1.f (*de cycle*)



le tracé rouge est un cycle

Exemple 1.IV.1.g (*de non-cycle*)



le tracé rouge n'est pas un cycle

1.IV.1.g.1 Remarques :

– La longueur correspond aussi au nombre de sommets
– Un cycle orienté est un cycle d'un graphe orienté pour lequel chaque arête doit être orientée d'un sommet de la liste vers son successeur dans cette même liste

Exemple 1.IV.1.h

//TODO : exemple de chemin orienté manquant

1.IV.1.i. **Circuit** *(Définition) :*
Circuit est tantôt utilisé comme synonyme de cycle, tantôt de le pendant pour un cycle d'une marche pour un chemin.

1.IV.2 Graphes connexes

1.IV.2.a. **Connexité, graphe connexe** *(Définition) :*
Un graphe est connexe si,pour toute paire de sommets u et v de G, il existe un chemin entre u et v

1.IV.2.b. **Sous-graphe connexe maximal** *(Définition) :*
Un sous-graphe connexe est dit maximal si l'ajout d'un sommet du graphe au sous-graphe transforme le sous-graphe en un graphe non-connexe.

Propriété #1.IV.2.b.1 :
Tout graphe peut être décomposé de façon unique en un ensemble de sous-graphes connexes maximaux appelés composantes connexes.

Exemple 1.IV.2.c

//TODO : exemples et contre-exemples manquants

1.IV.2.d. **Composante fortement connexe** *(Définition) :*
Une composante fortement connexe est un sous-graphe maximal d'un graphe orienté vérifiant la propriété suivante :
$\forall (u, v) \in V(G)^2$, il existe un chemin orienté de u vers v ET un chemin orienté de v vers u

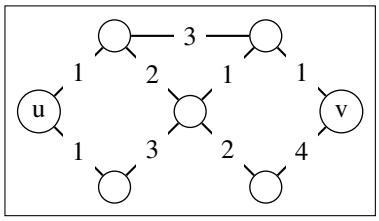
Exemple 1.IV.2.e

//TODO : exemples manquants

1. *Le graphe non-orienté sous-jacent d'une composante fortement connexe est connexe*
2. *Le graphe orienté n'est pas fortement connexe*

1.IV.2.f. **Distance** *(Définition) :*
Soit G un graphe arête-valué, la distance entre deux sommets u et v de G est le poids total minimal des chemins entre u et v, le poids d'un chemin étant la somme des poids de toutes ses arêtes. Si G n'est pas arête valué, on attribue le poids 1 à chaque arête pour l'assimiler à un graphe arête-valué.

Exemple 1.IV.2.g



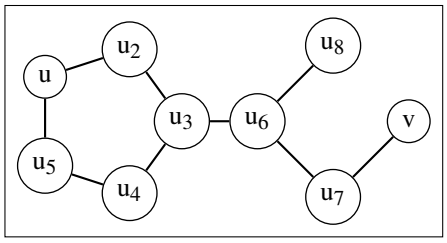
d(u, v) = 5

1.IV.2.g.1 Remarque : Dans le cas non-arête-valué, cela revient à chercher le chemin le plus court entre deux sommets.

1.IV.2.h. **Diamètre** *(Définition) :*
Le diamètre d'un graphe est la distance maximale qui soit entre toutes paires de sommets d'un graphe.

1.IV.2.h.1 Remarque : Dans le cas non-arête-valué, c'est le plus grand chemin le plus court entre tous les sommets d'un graphe.

Exemple 1.IV.2.i



diam(G) = 5 *car* d(u, v) = 5 *et toutes les autres paires de sommets ont une distance inférieure ou égale à* 5.

1.V Algorithme de Dijkstra

1.V.1 Objectif

Trouver le plus court chemin entre deux sommets donnés d'un graphe.

1.V.2 Problématique

Calcul de la distance entre le sommet u et tous les autres sommets du graphe

1.V.3 Principe

Soient G graphe et u le sommet départ.
On introduit le sous-graphe H qui contient l'ensemble des sommets dont on connaît la distance à u (on initialise donc H à u et on associe à u la distance 0)
Tant que H ≠ G, on considère les voisins de H dans G : soit v l'un de ces sommets et $w_1, \ldots, w_n$ ses voisins dans H. On calcule f(v) = min{(}d(u, w_i) + poids(w_i, v)) et on ajoute à H le sommet pour lequel f(v) est minimal ;la distance à u vaut f(u). //TODO : fin manquante ?

Chapitre 2 : Parcours dans les graphes

2.I Arbres

2.I.1 Définitions

2.I.1.a. **Arbre** *(Définition) :*
Un graphe non-orienté acyclique connexe minimal est appelé un arbre.
En d'autres termes, il n'a pas de cycles et n'est plus connexe si on lui retire une arête, quelqu'elle soit.

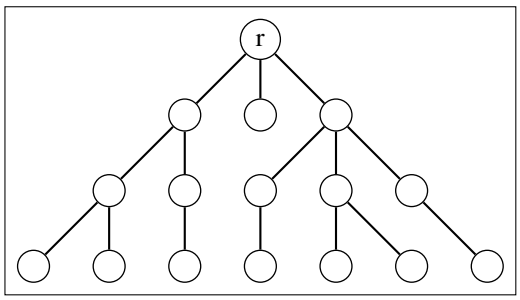
Exemple 2.I.1.b

//TODO : exemple manquant

2.I.1.c. **Arbre enraciné** *(Définition) :*
Soient n un entier et G un graphe, dont les sommets sont étiquetés par un niveau, construit de la façon suivante :

1. Initialisation V(G) = r et E(G) = ∅, r est le sommet-racine de G et son niveau est 0
2. Pour tout i, entier entre 0 et n,
   (a) Pour tout sommet v de niveau i, on ajoute un nombre fini de sommets $w_1,\ldots,w_{k_v}$ (appelés fils de v) chacun de niveau i + 1 et on ajoute à E(G) l'ensemble des arêtes $\left(v; w_j\right)$ ($j \in [\![1 ; k_v]\!]$). v est dit le père des nouveaux sommets
3. Le graphe final G est appelé "arbre enraciné en r"

Exemple 2.I.1.d



Graphe enraciné en r *avec des sommets de niveaux* 0 *à* 3.

Théorème #2.I.1.d.1 :

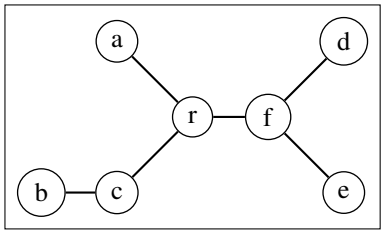
– Un arbre enraciné est un arbre
– Tout arbre peut s'écrire comme un arbre enraciné en chacun de ses sommets

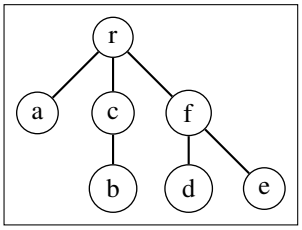
Démonstration du théorème

– Si G est un arbre enraciné, il est connexe puisque tous ses sommets ont un chemin qui remonte à r. De plus, si e = v, w est supprimé, on brise tout chemin entre v et w.
– Soit G un arbre et r ∈ V(G) un sommet quelconque de G. On affecte le niveau 0 à r, et à tout sommet (r inclus), on affecte pour fils ses voisins on explorés (c'est-à-dire tous ses voisins sauf son père car le graphe est acyclique). Comme le graphe est connexe, aucun sommet ne peut être ignoré par cette méthode.

□

Exemple 2.I.1.e



est identique à


Propriété #2.I.1.e.1 :
Un arbre enraciné à n sommets a n – 1 arêtes.

DÉMONSTRATION DE LA PROPRIÉTÉ

Dans la construction de l'arbre enraciné, on ajoute une arête à chaque ajout de sommet, sauf pour la racine. □

2.I.2 Intérêt des arbres

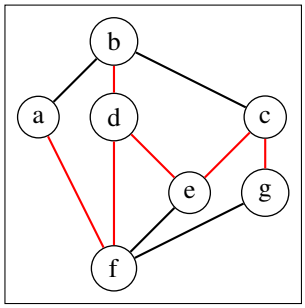
1. Dans certains cas, les arbres modélisent bien le problème (évolution, famille sans consanguinité, ...). Or de nombreux problèmes non-polynômiaux en général sont polynômiaux pour les arbres.
2. Les structures d'arbres permettent d'énumérer efficacement.

   Exemple 2.I.2.a (*Énumération de tous les mots de 4 lettres formés avec les lettres* A; B; C *et au plus 2 fois la lettre* A)
3. L'arbre est la structure la moins lourde assurant la connexité

2.I.2.b. **ARBRE COUVRANT** *(Définition) :*
On appelle arbre couvrant d'un graphe G tout arbre Ttel que V(T) = V(G) et E(T) ⊂ E(G)

Propriété #2.I.2.b.1 :
Un graphe G est connexe si et seulement si G admet un arbre couvrant.

Exemple 2.I.2.c



Rechercher un arbre couvrant est la solution optimale en terme de nombre d'arêtes parcourues pour visiter tous les sommets d'un graphe connexe.

2.I.3 Parcours en largeur

2.I.3.a Algorithme du parcours en largeur

Soient G un graphe non-orienté et v un sommet de G.
On peut construire le parcours en largeur de G partant de v selon l'algorithme suivant.

ALGORITHME 1 : Parcours en largeur de G partant de v

Input: (G,v) un graphe et l'un de ses sommets

Output: T l'arbre enraciné du parcours en largeur de G

Complexity: O(|E(G)|)

```
1:  function ParcoursEnLargeur(G,v)
2:      F = v                  # une file
3:      T = v                  # un arbre
4:      for u from u_1 to u_n do
5:          if u = v then
6:              visite(v) = TRUE
7:          else
8:              visite(u) = FALSE
9:          end if
10:     end for
11:     while F ≠ ∅ do
12:         u = head(F)              # on récupère dans u l'élément en tête de F
13:         F = F \ {u}
14:         while w ∈ V(G) ∧ (u; w) ∈ E ∧ visite(w) = FALSE do
15:             visite(w) = TRUE
16:             tail(F) = w          # on ajoute w à la fin de F
17:             T = T ∪ (u; w)           # ou T ∪ (v; w) ? ? mais ça n'a pas de sens !
18:         end while
19:     end while
20:     return T
21: end function
```

2.I.3.b Propriétés des arbres construits par un parcours en largeur

Propriété #2.I.3.b.1 :
L'arbre enraciné construit par le parcours en largeur de G partant de v est un arbre couvrant d'un graphe connexe G. De plus, le niveau de w ∈ V(G) dans cet arbre correspond à la longueur d'un plus court chemin entre v et w dans G

Propriété #2.I.3.b.2 :
Si G est non-arête-valué, le niveau de chaque sommet de l'arbre enraciné construit par le parcours en largeur de G partant de v est la distance entre ce sommet et v.

DÉMONSTRATION DE LA PROPRIÉTÉ

Soient G un graphe connexe et C un arbre enraciné de G. Supposant que C ne soient pas couvrant. Alors il existe au moins un sommet a de G qui ne soit pas un sommet de C. G étant connexe et il existe au moins une arête entre un sommet b de C et a. ce qui implique que lorsque l'algorithme est passé en b, a n'a été ajouté à C. Cela est impossible car les seuls sommets voisins de b non-ajoutés à C sont ceux qui étaient déjà dans C.
De plus, s'il existait un chemin pour court, le sommet aurait un niveau plus petit. □

Propriété #2.I.3.b.3 :
Le parcours en largeur est de complexité O(e) où e = |E(G)|.

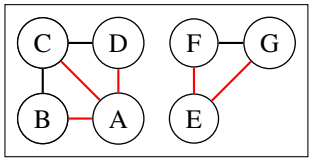
Démonstration de la propriété

Chaque sommet n'entre qu'une fois dans la file, et pour chacun des sommets, il faut regarder ses voisins au moment où il sort de la file. Par conséquent, d(v) opérations sont associées au sommet v.
On réalise donc
– $\sum_{v} d(v) = 2e$ opérations liées à une vérification des voisins.
– n opérations consistant à mettre à jour visite(u) pour chaque sommet u
Le sommet d'opérations total est de l'ordre de $2e + n = O(e)$ car $n \leq m + 1$ dans un graphe connexe.

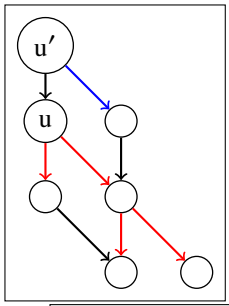
2.I.3.b.1 Remarque : Cette complexité est optimale car il faut e pas pour parcourir un chemin de longueur e. On ne peut donc pas faire mieux qu'une complexité linéaire. □

2.I.3.c Cas d'un graphe non connexe

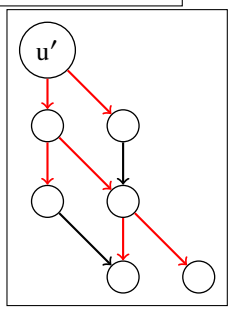
Il suffit d'exécuter un parcours en largeur sur chacune des composantes connexes. L'algorithme s'arrête quand la composante connexe du sommet de départ a été parcourue. On le relance ensuite à partir d'un sommet non parcouru du graphe.
On obtient finalement un ensemble (appelé forêt) d'arbres disjoints, chacun couvrant une composante connexe.



2.I.3.d Cas d’un graphe orienté



OU



Deux choix se présentent : soit on oublie l'orientation et on applique le cas non-orienté, soit on autorise le parcours des arêtes uniquement dans le sens de leur orientation et on recommence avec les sommets pour lesquels il n'existe pas de chemin orienté partant du sommet de départ. Dans le premier cas, on obtient un arbre couvrant, mais dans lequel les flèches peuvent être parcourues dans les deux sens ; tandis que dans le deuxième cas, on peut obtenir une forêt d'arbres couvrants et des résultats différents selon le sommet de départ.

2.I.4 Parcours en profondeur

2.I.4.a Algorithme du parcours en profondeur

L'algorithme du parcours en profondeur est le suivant :
Soient G un graphe non-orienté et v un sommet de G.
On peut construire le parcours en profondeur de G partant de v selon l'algorithme suivant.

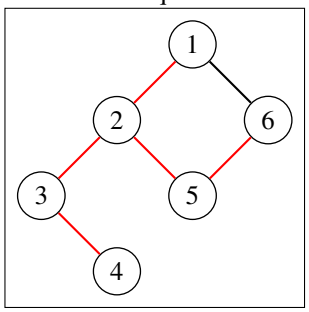
ALGORITHME 2 : Parcours en profondeur de G partant de v

Input: (G,v) un graphe et l'un de ses sommets

Output: T l'arbre enraciné du parcours en largeur de G

```
1:  function ParcoursEnProfondeur(G,v)
2:      F = v                    # une file
3:      T = v                    # un arbre
4:      for u from u_1 to u_n do
5:          if u = v then
6:              visite(v) = TRUE
7:          else
8:              visite(u) = FALSE
9:          end if
10:     end for
11:     while w ∈ V(G) ∧ (u; w) ∈ E ∧ visite(w) = FALSE do        # pour tous les voisins de u non visités
12:         T = T∪ParcoursEnProfondeur(G[visite = FALSE],w)      # où G[visite = FALSE] est le sous-graphe
    de G produit par les sommets non visités
13:     end while
14:     return T
15: end function
```

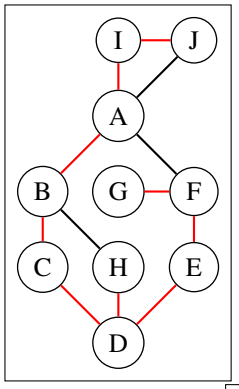
Cet algorithme revient à parcourir le graphe en avançant tant que possible et en remontant lorsque ce n'est plus possible, c'est-à-dire au père du sommet courant si ce dernier n'a pas de voisin non visités.
Il s'arrête lorsque le sommet courant n'a plus de voisin non visité et que c'est la racine.

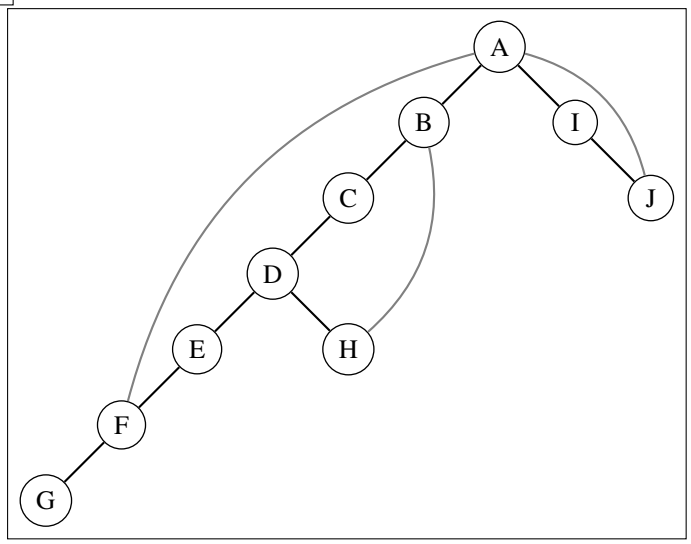


2.I.4.b Propriétés des arbres construits par un parcours en profondeur

Propriété #2.I.4.b.1 :
Si G est un graphe connexe non orienté, un parcours en profondeur lancé sur l'un de ses sommets renvoie un arbre couvrant de G.

Propriété #2.I.4.b.2 :
Si G est un graphe connexe non orienté, toute arête de G qui n'est pas dans un arbre couvrant T résultat de son parcours en profondeur à partir de l'un de ses sommets relie deux sommets qui sont descendants l'un de l'autre dans T.





correspond à l'arbre

Il n'y a pas d'arête entre la branche gauche et la branche droite. Cela permet de dégager des séparateurs dans le graphe.

DÉMONSTRATION DE LA PROPRIÉTÉ

Montrons que tout sommet u est visité.
Soit r la racine et $u \neq r$ un sommet quelconque de G. Comme G est connexe, il existe un chemin $r = u_0, u_1, u_2, \ldots, u_k = u$.
Démontrons par récurrence sur i qe u_i est visité.
Pour $i = 0$,c'est évident car $u_0 = r$ est la racine.
Supposons que u_i est visité,avec $i \in [\![0\,;k-1]\!]$.
Alors le parcours en profondeur est lancé sur u_i, ce qui implique qu'à un moment, on vérifie si u_{i+1} est visité. □

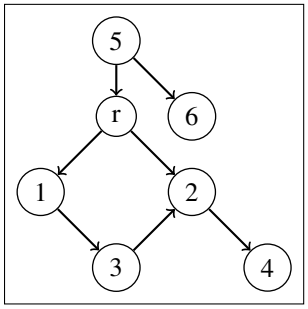
//TODO : morceau manquant ?

2.I.4.c Cas orienté

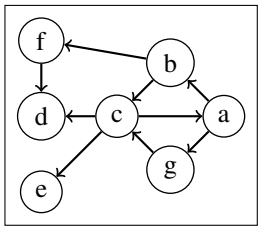
On remplace "voisin" par "successeur" (voisin externe) dans la boucle de l'algorithme.s Dans ce cas :
- L'ensemble des sommets parcourus au sein de l'arbre enraciné en u est l'ensemble des sommets v tel qu'il existe un chemin entre u et v. Si on choisit de parcourir le graphe en entier, le nombre d'arbres nécessaires dépend des choix des racines.
- La propriété topologique des arêtes n'appartenant pas à l'arbre change : si on dessine l'arbre enraciné en mettant les branches parcourues en premier à gauche, toute arête de G qui n'est pas dans l'arbre soit relie des descendants soit va de la droite vers la gauche

Exemple 2.I.4.d

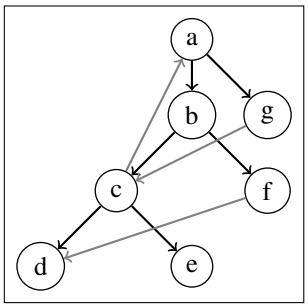
//TODO exemple à corriger



Exemple 2.I.4.e



a pour arbre de parcours en profondeur



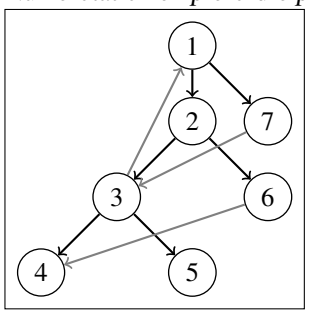
2.I.4.f Pré-ordre et post-ordre

2.I.4.g. **Numérotation en pré-ordre** *(Définition) :*
Une numérotation en pré-ordre correspond à numéroter les sommets dans l'ordre dans lequel ils sont parcourus pour la première fois.

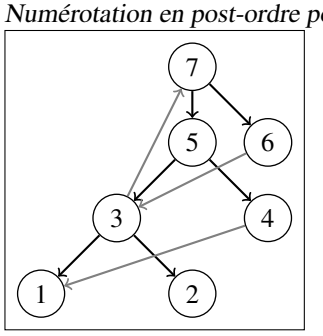
2.I.4.h. **Numérotation en post-ordre** *(Définition) :*
Une numérotation en post-ordre correspond à numéroter les sommets dans l'ordre dans lequel ils sont parcourus pour la dernière fois.

Exemple 2.I.4.i

Numérotation en pré-ordre pour l'exemple précédent : a(1)-b(2)-c(3)-d(4)-e(5)-f(6)-g(7)



Numérotation en post-ordre pour l'exemple précédent : d(1)-e(2)-c(3)-f(4)-b(5)-g(6)-a(7)



2.II Arbre couvrant de poids minimal

Dans le cas d'un graphe valué, on est intéressé par la recherche d'un arbre couvrant de poids minimal.

2.II.1 Algorithme de Kruskal

ALGORITHME 3 : Algorithme de Kruskal

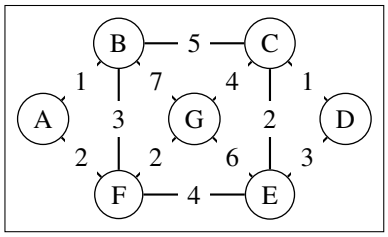
Input: (G) un graphe

Output: F l'arbre couvrant de G de poids minimal

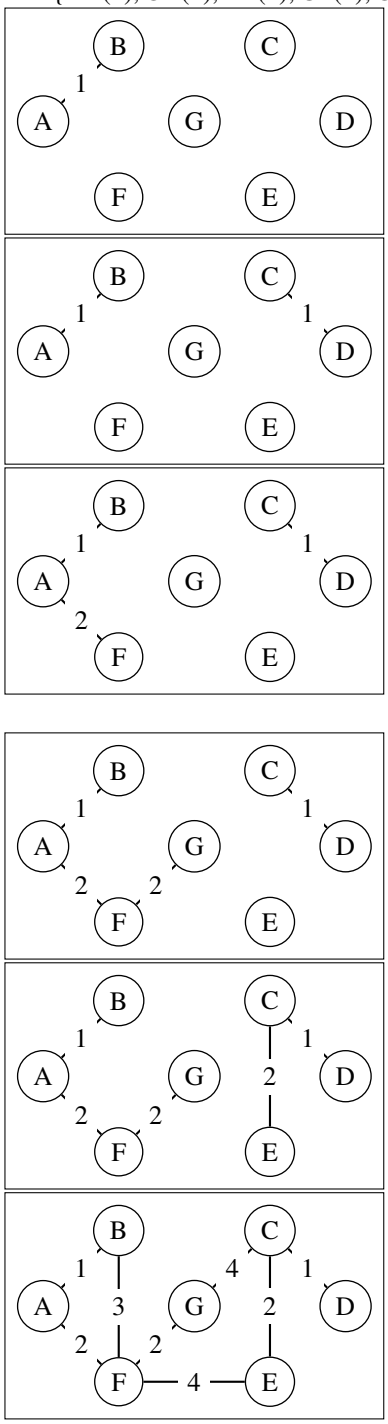
Complexity: $O\left(m^2 \log(()m)\right)$

```
1:  function KRUSKAL(G)
2:      L = E(G)                  # une liste des arêtes ordonnées par poids croissant
3:      F = ∅                     # l'arbre résultant
4:      while F n'est pas un arbre do
5:          e = head(L)
6:          L = L \ {e}
7:          if F ∪ {e} n'a pas de cycle then
8:              F = F ∪ {e}
9:          end if
10:     end while
11:     return F
12: end function
```

Exemple 2.II.1.a



L = {AB(1), CD(1), AF(2), GF(2), CE(2), BF(3), ED(3), EF(4), CG(4), BC(5), EG(6), BG(7)}



L = $\{4, 5, 6, 7\}$ *on a un arbre donc on arrête. Le poids de cet arbre est* 1 + 1 + 2 + 2 + 2 + 4 = 12.

On admet que l'arbre obtenu soit de poids minimal.

2.II.2 Algorithme de Primm

L'idée est de suivre le même raisonnement, mais en partant du graphe tout entier plutôt que d'un arbre vide.

ALGORITHME 4 : Algorithme de Primm

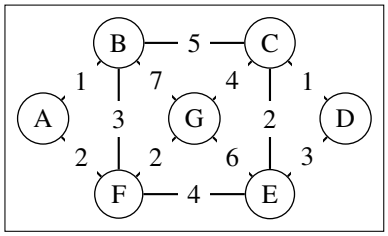
Input: (G) un graphe

Output: F l'arbre couvrant de G de poids minimal

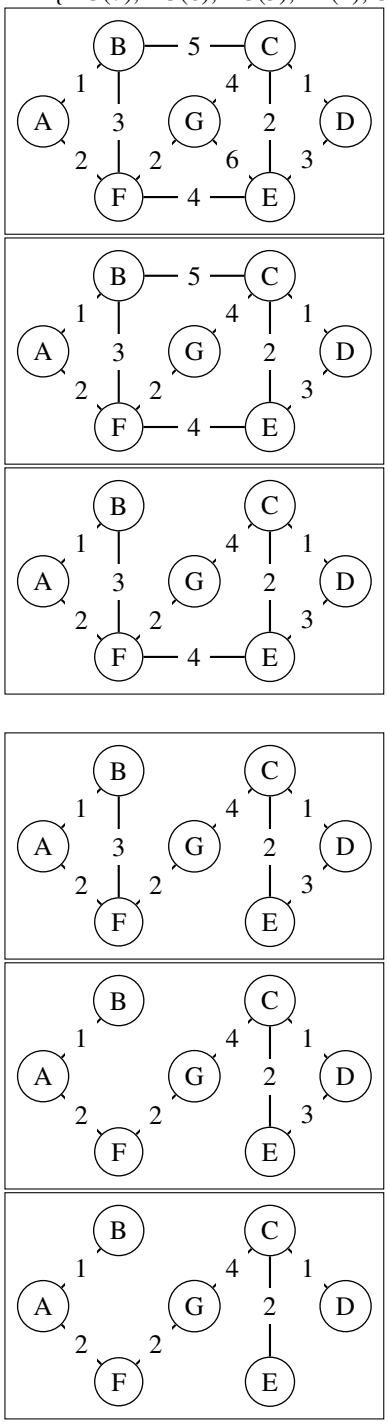
Complexity: $O\left(m^2 \log(()m)\right)$

```
1:  function PRIMM(G)
2:      L = E(G)                # une liste des arêtes ordonnées par poids décroissant
3:      F = G
4:      while F n'est pas un arbre do
5:          e = head(L)
6:          L = L \ {e}
7:          if F \ {e} est connexe then
8:              F = F \ {e}
9:          end if
10:     end while
11:     return F
12: end function
```

Exemple 2.II.2.a



L = {BG(7), EG(6), BC(5), EF(4), CG(4), BF(3), DE(3), CE(2), GF(2), AF(2), AB(1), CD(1)}



L = {2, 2, 2, 1, 1} *on a un arbre donc on arrête. Le poids de cet arbre est* 1 + 1 + 2 + 2 + 2 + 4 = 12

On admet que l'arbre obtenu soit de poids minimal.

2.III Circuits couvrants

Considérons les deux problèmes suivants :

Chinese Postman Problem : Un "postier chinois" cherche l'itinéraire le plus court pour sa tournée sachant qu'il doit parcourir chaque rue au moins une fois. Les rues sont modélisées par des arêtes valuées par leur longueur, les intersections par des

sommets/noeuds et l'un des sommets est marqué comme le point de départ et d'arrivée du postier. **Il faut trouver le circuit de poids minimal couvrant au moins une fois chaque *arête* du graphe de modélisation.**

Traveling Salesman Problem : Un "voyageur de commerce" doit faire le tour de ses clients (modélisés par des sommets) en minimisant un certain coût (durée, distance, prix,etc...) lié à ses déplacements. Les routes possibles allant d'un client à l'autre sont indiquées comme des arêtes valuées par ce qu'il coûte de les emprunter. **Il faut trouver le circuit de poids minimal couvrant au moins une fois chaque *sommet* du graphe de modélisation.**

2.III.1 Graphes eulériens

2.III.1.a. **Circuit eulérien** *(Définition) :*
Un circuit eulérien est une marche empruntant exactement une fois chaque arête d'un graphe et finissant en son point de départ.

2.III.1.b. **Graphe eulérien** *(Définition) :*
Un graphe est eulérien s'il admet un circuit eulérien

2.III.1.b.1 Remarques :

– Dans le cas où toutes les rues sont de même longueur, un circuit eulérien est la réponse optimale au Chinese Postman Problem.

Théorème #2.III.1.b.1 :

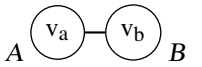
Un graphe connexe est eulérien si et seulement si tous ses sommets sont de degrés pairs.

Corollaire (du thérème précédent) #2.III.1.b.1.1 :
Le problème de décision "un graphe donné est-il eulérien ?" est linéaire en O(|E(G)|).

Pou démontrer le théorème, il faut introduire les ponts.

2.III.1.c. **Pont** *(Définition) :*
Dans un graphe, un pont est une arête dont la suppression augmente le nombre de composants connexes du graphe.

Exemple 2.III.1.d



G[A] *est un graphe connexe,* G[B] *est un graphe connexe et* E *est la seule arête de* G *entre* A *et* B *:* $V(G) = V(G[A]) \cup V(G[B])$ *et* $E(G) = E(G[A]) \cup E(G[B]) \cup \{e\}$

Proposition #2.III.1.d.1 :
Si e est un pont, il y a un nombre impair de sommets de degré impair de chaque côté de e (on "coupe" l'arête e en deux).

Démonstration de la proposition

Deux cas se présentent pour chaque côté de e. Concentrons-nous uniquement sur G[A], c'est-à-dire à G[A] : $\sum_{u \in G[A]} d(u) = 2|E(G[A])|$, donc $\sum_{u \in G[A]} d(u)$ est pair. Il y a par conséquent un nombre pair de sommets de degré impair dans G[A].

– Soit v_a est de degré impair dans G auquel cas il y a un nombre impair de sommets de degré impair du côté gauche de e.
– Soit v_a est de degré pair dans G. Donc v_a est de degré impair dans G[A]. Ces autres sommets ont le même degré dans G quand dans G[A] car e est un pont, ce qui fait de v_a le seul sommet de G[A] à ne pas avoir le même degré que dans G.

□

Démonstration du théorème

– S'il existe un circuit eulérien, un sommet parcouru k fois est forcément de degré 2k (car à chaque parcours, on utilise une arête pour arriver et une pour repartir).
– Supposons que G a tous ses sommets de degré pair et notons n = |V(G)| et m = |E(G)|
  Appliquons l'algorithme de Fleury suivant :
  1. On choisit v_0 un point de départ, et définit $W_0 = \emptyset$ comme l'ensemble des arêtes parcourues.
  2. On suppose que $W_i = \{e_1, \dots, e_i\}$ et note $v_0, \dots, v_i$ les arêtes parcourues, avec répétitions.
  3. S'il existe des arêtes $\{e_{i+1}, \dots, e_{m+1}\}$ adjacences à v_i non encore parcourues,
     – Si l'une d'entre elles e_k n'est pas un pont dans le graphe $H = (V(G), E(G) \setminus W_i)$ (le graphe où les arêtes de W_i ont été supprimées), on l'ajoute à W_i. (on peut lancer un parcours en profondeur à chaque fois pour connaître les arêtes qui sont des ponts).
     – Sinon, toutes ces arêtes sont des ponts et on en ajoute une quelconque e_k à W_i.
  4. On repart au point 2 avec $w_{i+1} = W_i \cup \{e_k\}$ et on note v_{i+1} l'autre extrémité de e_k.

  Cet algorithme est valide car
  – Chaque pas parcours une arête, donc l'algorithme est certain de se terminer
  – Au cours de l'algorithme, seul v_0 a un nombre d'arêtes adjacentes libres impair. Pour tout autre sommet, chaque parcours utilise 2 arêtes (une entrante et une sortante). Or, pour s'arrêter, il faut arriver sur un sommet n'ayant plus qu'un arête de libre : c'est forcément v_0.
  – Si toutes les arêtes n'étaient pas parcourues, cela voudrait dire qu'il resterait au moins un pont f dans le graphe, mais que ce serait aussi le cas de la première arête du chemin entre v_0 et l'extrémité v' de f incluse dans le parcours de l'algorithme. Ce sommet d'intersection v', différent de v_0, aurait donc un nombre impair non parcourues, et par conséquent un nombre impair d'arêtes, ce qui est impossible.

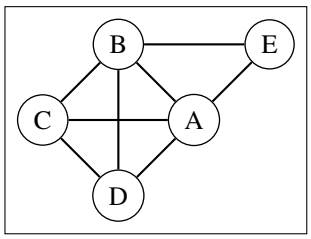
□

Corollaire (du thérème précédent) #2.III.1.d.1.1 :
Avec l'algorithme de Fleury, on peut construire en temps polynômial un circuit eulérien, s'il existe.

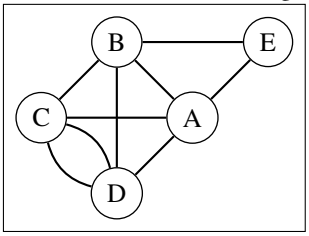
2.III.2 Problème du postier chinois

On cherche le circuit (avec répétitions d'arêtes possibles) le plus court parcourant au moins une fois chaque arête.

Exemple 2.III.2.a



trouver un circuit sur le graphe précédent qui utilise une fois chaque arête sauf CD *(*C *et* D *étant les seuls sommets de degrés impairs) qu'il utilise deux fois est équivalent à trouver un circuit eulérien dans le graphe suivant :*



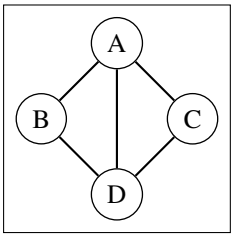
Méthode générale :
//TODO : fin manquante

2.III.3 Graphes hamiltoniens

2.III.3.a. **Cycle hamiltonien** *(Définition) :*
Un cycle est hamiltonien dans un grapheG s'il passe par tous les sommets de G.

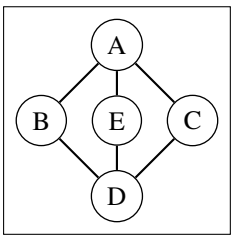
2.III.3.b. **Graphe hamiltonien** *(Définition) :*
Un graphe G est hamiltonien s'il admet un cycle hamiltonien.

Exemple 2.III.3.c (*graphe hamiltonien*)



est un graphe hamiltonien

Exemple 2.III.3.d (*graphe non-hamiltonien*)



n'est pas un graphe hamiltonien

2.III.3.d.1 Remarque : Dans le cas où toutes les arêtes sont valuées par 1, un cycle hamiltonien est une solution optimale pour le Traveling Salesman Problem.

2.III.3.e Complexité

2.III.3.f. **Ensemble des problèmes** P *(Définition) :*
Un problème est dans P s'il peut être résolu en un temps polynômial en la taille de l'instance.

Exemple 2.III.3.g

L'existence d'un cycle eulérien.

2.III.3.h. **Ensemble des problèmes** NP *(Définition) :*
Un problème est dans NP si l'on sait vérifier dans un temps polynômial si une solution est valide.

Exemple 2.III.3.i

L'existence d'un cycle hamiltonien.

2.III.3.j. **Problème** NP**-complet** *(Définition) :*
Un problème est NP-complet si
– il est dans NP
– prouver qu'il est aussi dans P équivaudrait à montrer que P = NP
On part du principe que P ≠ NP, c'est-à-dire que si un problème est NP-complet, il n'existe pas d'algorithme polynômial pour le résoudre.
Pour montrer qu'il problème est NP-complet, il suffit de montrer la chose suivante : savoir le résoudre en temps polynômial permet de savoir résoudre un autre problème connu pour être NP-complet.

2.III.3.k. **Réduction de problèmes** *(Définition) :*
On parle de réduction d'un problème P_2 à un problème P_1 si savoir résoudre P_1 permet de savoir résoudre P_2.

2.III.3.l. 3 – SAT *(Définition) :*
On considère n variables booléennes $x_1, \ldots, x_n$ et m clauses $c_1, \ldots, c_m$ de 3 variables ou de leurs négations réunies par des "OU".
On pose $C = \bigwedge_{i=1}^{m} c_i$.
Le problème 3 – SAT revient à répondre à "existe-t-il une assignation telles que C = 1 ?".

Karp a démontré en 1972 3 – SAT est un problème NP-complet.

Exemple 2.III.3.m

$c_1 = \overline{x_1} \vee \overline{x_2} \vee x_4$
$c_2 = x_1 \vee x_2 \vee x_3$
$c_3 = x_2 \vee x_3 \vee \overline{x_4}$
$C = c_1 \wedge c_2 \wedge c_3 = (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \overline{x_4})$.
La réponse est "oui en prenant $x_1 = 1, x_2 = 0, x_3 = 1$ *et* $x_4 = 0$*".*

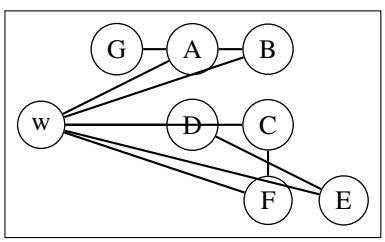
Théorème #2.III.3.m.1 :
Le problème "Déterminer l'existence d'un cycle hamiltonien" est NP-complet.

Démonstration du théorème

1. Le problème de l'existence du cycle hamiltonien est une réduction de la recherche de l'existence d'un chemin hamiltonien.
   On suppose qu'on a un algorithme CH qui résout le problème du cycle. Soient G un graphe et G' un graphe obtenu en ajoutant à G un sommet w relié à tous les autres sommets de G. Alors G admet un chemin hamiltonien si et seulement si G' admet un cycle hamiltonien. En effet, soit C' un cycle hamiltonien dans G' ; on peut obtenir un chemin hamiltonien dans G en regardant ce que devient C' si G' est privé de w et des arêtes incidentes à w.
2. Le problème de l'existence d'un chemin hamiltonien est une réduction de la recherche de l'existence d'un chemin hamiltonien orienté.
   Soient G un graphe orienté et G' un graphe non-orienté construit à partir de G en remplaçant tout sommet $v \in V(G)$ par une suite de trois sommets $\{v_{in}, v, v_{out}\} \in V(G')$ de telle façon que toute arête orientée $(U; v)$ devient une arête non orientée $(u_{out}; v_{in})$.
3. //TODO manque le début de la partie avec 3 – SAT et le schéma
   - On ajoute un sommet start qui voit $x_{1,1}$ et $x_{1,2m+2}$
   - On ajoute un sommet end qui est vu par $x_{n,1}$ et $x_{n,2m+2}$
   - Pour tout i, on ajoute les arêtes de $x_{i,1}$ et $x_{i,2m+2}$ à $x_{i+1,1}$ et $x_{i+1,2m+2}$
   - On a 2^n chemins hamiltoniens, tous de start vers end. Chacun d'eux traverse tous les niveaux. La bijection est une bijection entre les chemins hamiltoniens et les assignations $x_1, \ldots, x_n$.
   - Soit c_j une clause.
4.

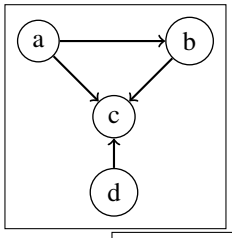
Comme 3 – SAT est NP-complet, il n'est pas possible d'en trouver une solution en temps polynômial, ce qui implique la même chose pour le problème de l'existence d'un chemin hamiltonien orienté, donc pour le problème de l'existence d'un chemin hamiltonien, donc pour le problème de l'existence d'un cycle hamiltonien. □

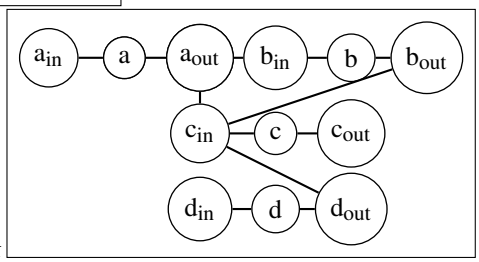
Exemple 2.III.3.n



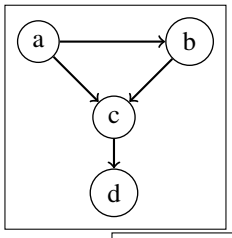
//TODO à finir

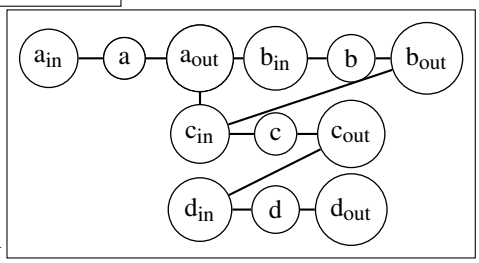
Exemple 2.III.3.o





donnerait





donnerait

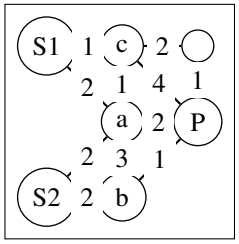
2.III.4 Problème du voyageur de commerce

Chapitre 3 : Flots

3.I Flots et coupes

On s'intéresse à la propagation de flux dans un réseau. Pour cela, on considère un graphe dirigé G tel que
- il existe deux ensembles de sommets S et P dont les éléments sont appelés respectivement sources et puits
- il existe un poids c(e) sur chaque arête $e \in E$, appelé capacité de l'arête, et qui correspond au flux maximal pouvant passer par cette arête

Exemple 3.I.0.a

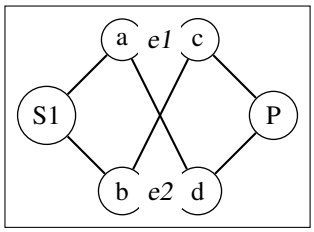


La question centrale va être de déterminer un flot maximal

3.I.1 Flots

Notation : Pour tout ensemble de sommets A, on note $\overline{A}$ sont complémentaire et $\left(A, \overline{A}\right)$ est l'ensemble des arêtes de A vers $\overline{A}$. Ainsi, $\left(\overline{\overline{A}}, \overline{A}\right) \neq \left(A, \overline{A}\right)$ car cette notation n'est pas symétrique du fait de la direction des arêtes.

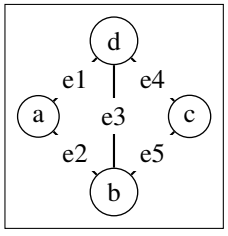
Exemple 3.I.1.a



$A = \{S1, a, b\}$ *et* $\left(A\overline{A}\right) = \{e1, e2\}$

Pour toute fonction $f : E(G) \to \mathbb{R}^+$, on note $f^+(A) = \sum_{e \in \left(A, \overline{A}\right)} f(e)$, $f^-(A) = \sum_{e \in \left(\overline{A}, A\right)} f(e)$ et $\Delta f(A) = f^+(A) - f^-A$

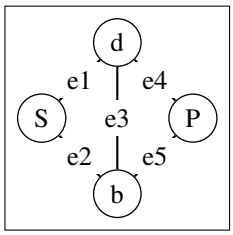
Exemple 3.I.1.b



f(e2) = 3
f(e1) = 2
f(e4) = 1
f(e3) = 1
f(e5) = 4
A = {a, b}
$f^+(A) = f(e1) + f(e5) = 6$
$f^-(A) = f(e3) = 1$
$\Delta f(A) = f^+(A) - f^-A = 5$

3.I.1.c. **Flot** *(Définition) :*
Un flot est une fonction $f : E(G) \to \mathbb{R}^+$ telle que
– $\forall e \in E(G) 0 \leqslant f(e) \leqslant c(e)$
– $\forall s \in V(G) \setminus \{S; P\}, f^+(\{v\}) = f^-(\{v\})$

Exemple 3.I.1.d



f(e2) = 3 *et* c(e2) = 3
f(e1) = 2 *et* c(e1) = 3
f(e4) = 1 *et* c(e4) = 3
f(e3) = 1 *et* c(e3) = 1
f(e5) = 4 *et* c(e5) = 5

Propriété #3.I.1.d.1 :
Pour tout flot f, $\Delta f(S) = -\Delta f(P)$. Cette valeur est notée val(f) est est appelé intensité du flot.

Exemple 3.I.1.e

Dans l'exercice précédent, val(f) = 5

Démonstration de la propriété
voir l'exercice 3.2 □

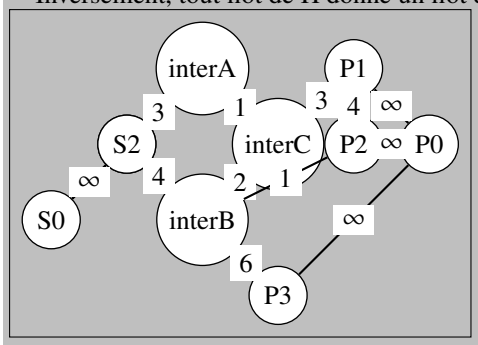
3.I.1.f. **Flot maximal** *(Définition) :*
Un flot f est maximal s'il n'y a pas de flot f′ tel que val(f′) > val(f)

Proposition #3.I.1.f.1 :
Les problèmes suivants sont équivalents :

1. Trouver un flot maximal dans un graphe dirigé avec S et P quelconques
2. Trouver un flot maximal dans un graphe dirigé avec |(|S) = 1 et |(|P) = 1 (une seule source et un seul puit)

Démonstration de la proposition – de 1 vers 2, c'est trivial car 2 est un cas particulier de 1

– de 2 vers 1 : soit G un graphe dirigé ayant un ensemble de sources S, un ensemble de puits P et des capacités. On ajoute au graphe une source S0 qui voit tous les sommets de S avec des arêtes de capacité infinie. De même on ajoute au graphe un puit P0 qui est vu par tous les sommets de P avec des arêtes de capacité infinie. On obtient ainsi un graphe H ayant une seule source S0 et un seul puit P0.
  Tout flot de G peut alors être étendu en un flot dans H de même intensité : il suffit de poser $\forall s \in S, f_H(S_0, s) = \Delta f_G(s)$ et $\forall p \in P, f_H(p, P_0) = \Delta f_G(p)$
  Inversement, tout flot de H donne un flot dans G.



//FIXME : arêtes manquantes ? □

À partir de là, on considèrera toujours que |(|S) = |(|P) = 1

3.I.1.g. **Coupe** *(Définition) :*
Une coupe est un ensemble d'arêtes $K = \left(A, \overline{A}\right)$ telle que $s \in A$ et $p \in \overline{A}$

3.I.1.h. **Capacité d'une coupe** *(Définition) :*
La capacité d'une coupe K est $cap(K) = \sum_{e \in K} c(e)$

Propriété #3.I.1.h.1 :
Pour toute coupe $K = \left(A, \overline{A}\right)$ et tout flot f, $val(f) = \Delta f(A)$

Propriété #3.I.1.h.2 :
Pour toute coupe K et tout flot f, $val(f) \leqslant cap(K)$.
En particulier, s'il existe un flot f^* et une coupe K^* tels que $val(f^*) = cap(K^*)$, alors f^* est un flot maximal et K^* est une coupe minimale.

Démonstration de la propriété
Dans un graphe G, soit f un flot quelconque et K une coupe quelconque.
D'après la propriété , $val(f) = f^+(A) - f^-(A)$.
Or, $f^+(A) = \sum_{e \in \left(A, \overline{A}\right)} f(e) \leqslant \sum_{e \in \left(A, \overline{A}\right)} c(e) = cap(K)$.
De plus, $f^-(A) = \sum_{e \in \left(A, \overline{A}\right)} f(e) \geqslant 0$
Finalement, $val(f) \leqslant cap(K) - 0 = cap(K)$.
Par ailleurs, soient f^* et K^* tels que $val(f^*) = cap(K^*)$.
Alors $val(f) \leqslant cap(K^*) \leqslant val(f^*)$ donc f^* est bien d'intensité maximale.
Similairement, $cap(K) \geqslant val(f^*) = cap(K^*)$ dont K^* est bien de capacité minimale. □

3.II Théorème et algorithme du Max-Flow-Min-Cut

3.II.1 Le théorème

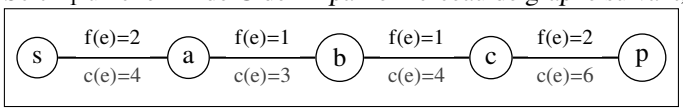
Théorème #3.II.1..3 :
Dans tout réseau (G, s, c, p),le flot maximal a pour intensité la capacité de la coupe minimale. En d'autres termes, il existe toujours f^* et K^* tels que $val(f^*) = cap(K^*)$.
La preuve est constructive, c'est-à-dire qu'on peut précisément construire f^*.

3.II.1.a. **Saturation d'une chaîne** *(Définition) :*
Soit f un flot et P un chemin entre s et p,on note $e^+(P)$ l'ensemble des arêtes parcourues dans le bon sens par P et $e^-(P)$ l'ensemble des arêtes qui sont parcourues à contre-sens par P.
La saturation de P est $sat(P) = \min_{e \in e^+(P)} \min c(e) - f(e), \min_{e \in e^-(P)} f(e)$. Si la $sat(P) = 0$, on dit que P est saturé.

Exemple 3.II.1.b

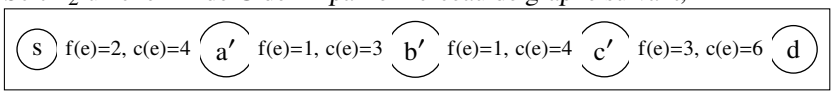
Soit un graphe G *parcouru par un flot* f.

1. *Soit* P_1 *un chemin de* G *défini par le morceau de graphe suivant,*



$sat(P_1) = 2$

2. *Soit* P_2 *un chemin de* G *défini par le morceau de graphe suivant,*



$sat(P_2) = \min\{(\}\min\{(\}4-2;4-1), \min\{(\}3;1)) = 1$

Propriété #3.II.1.b.1 :
Un flot f est maximal si et seulement s'il n'existe pas de chemin non-saturé entre s et p

Démonstration de la propriété

Soit un réseau (G, s, c, p)

$\Rightarrow$: Supposons qu'il existe un chemin P non saturé entre s et p

On définit une fonction f′ par $f'(e) = \begin{cases} f(e) + sat(P) & \text{si } e \in e^+(P) \\ f(e) - sat(P) & \text{si } e \in e^-(P) \\ f(e) & \text{si } e \notin P \end{cases}$

Soit $v \in V(G) \setminus \{s; p\}$

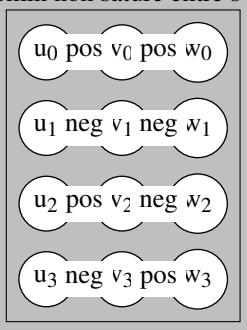
- si $v \notin P$, $f'^+(v) = \sum_{e \to v} f'(v) = \sum_{e \to v} f(v) = \sum_{v \to e} f(v) = \sum_{v \to e} f'(v) = f'^-(v)$
- si u, v et w sont des sommets de P tels que $e_{uv}(u; v) \in e^+(P)$ et $e_{vw}(v; w) \in e^+(P)$, $f'^+(v) = f^+(v) + sat(P) = f^-(v) + sat(P) = f'^-(v)$
- si u, v et w sont des sommets de P tels que $e_{uv}(u; v) \in e^-(P)$ et $e_{vw}(v; w) \in e^-(P)$, $f'^+(v) = f^+(v) - sat(P) = f^-(v) - sat(P) = f'^-(v)$
- si u, v et w sont des sommets de P tels que $e_{uv}(u; v) \in e^+(P)$ et $e_{vw}(v; w) \in e^-(P)$, $f'^+(v) = f^+(v) + sat(P) = f^-(v) - sat(P) = f'^-(v)$
- si u, v et w sont des sommets de P tels que $e_{uv}(u; v) \in e^-(P)$ et $e_{vw}(v; w) \in e^+(P)$, $f'^+(v) = f^+(v) - sat(P) = f^-(v) + sat(P) = f'^-(v)$

Donc f est un flot et s'il existe un chemin non-saturé, f n'est pas maximal.

Par contraposée, si f est maximal,alors il n'existe pas de chemin non-saturé entre s et p

$\Leftarrow$: Supposons qu'il n'existe pas de chemin non saturé entre s et p

Soit A l'ensemble des sommets v tels qu'il existe un chemin non saturé entre s et v



Alors, la coupe $K = \left(A, \overline{A}\right)$ est telle que $val(f) = cap(K)$

□

Démonstration de la propriété

//TODO □

3.II.2 Construction algorithmique d'un flot maximal

La construction d'un flot maximal se fait de la façon suivante :

1. On initialise en prenant un flot connu, par défaut le flot nul
2. Tant qu'on trouve un chemin non saturé en appliquant la procédure suivante, on remet le flot à jour :
   (a) on fait un parcours en profondeur ou en largeur en partant de s et en autorisant les arêtes "dans le bon sens" si leur capacité n'est pas atteinte par leur flux et "à contre-sens" si leur flux est strictement positif (non nul) : les arêtes non autorisées sont donc celles dont le flux est soit égal à la capacité, soit nul
   (b) si l'arbre atteint p, il existe un chemin non-saturé entre s e p ; sinon, un tel chemin n'existe pas

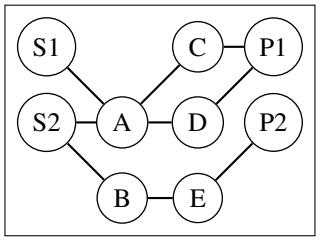
3.III Conséquence : le théorème de Menger

Théorème #3.III.0..2 : Théorème de Menger - version arêtes
Soient S et P deux ensembles de sommets dans un graphe G et k un entier.
Alors, soit il existe k chemins arêtes-disjoints entre S et P, soit il existe un ensemble de k−1 arêtes dont la suppression déconnecte S et P.

Démonstration du théorème
On donne une capacité 1 à tout arête et on ne considère que des flots binaires. Alors, la valeur maximale d'un flot est égale au nombre de chemins arêtes-disjoints et la valeur minimale d'une coupe est égale au nombre d'arêtes minimal à enlever pour couper toute relation entre S et P.
Or val(f) = cap(K). Ainsi, deux cas se présentent :
– soit val(f) $\geqslant$ k : il y a k chemins arêtes-disjoints
– soit val(f) $\leqslant$ k – 1 : il faut k – 1 suppressions pour déconnecter S et P
□

3.III.0..1 Remarque : cela reste valable dans le cas non-orienté (on remplace chaque arête non orientée par un couple d'arêtes orientées, un dans chaque sens)

Exemple 3.III.0.a



Théorème #3.III.0.a.1 : Théorème de Menger - version sommets
Soient S et P deux ensembles de sommets dans un graphe G et k un entier.
Alors, soit il existe k chemins sommets-disjoints entre S et P, soit il existe un ensemble de k – 1 sommets dont la suppression déconnecte S et P.

Chapitre 4 : Chaînes de Markov

4.I Définition

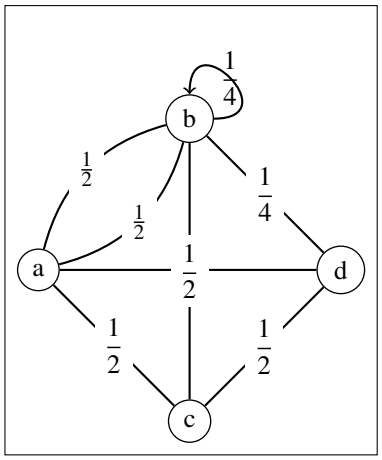
4.I.1 Marche aléatoire sans mémoire sur un graphe

4.I.1.a. **Marche aléatoire homogène** *(Définition) :*
Soit G un graphe orienté à n sommets $v_1, \cdots, v_n$ et dont les arêtes sont valuées par des poids p_{uv} tels que $\forall u \in V$, $\sum_{v \in N^+(u)} p_{uv} = 1$.
Une marche aléatoire homogène sur G est une suite $(x_n)_{n>0}$ où :
– x_0 est la position initiale
– si à l'instant i, $x_i = v$, $\mathbb{P}(x_{i+1} = w | x_1 = v) = p_{vw}$

4.I.1.a.1 Remarque : Il existe également des marches aléatoires non homogènes, mais qui ne seront pas étudiées dans ce cours. Elles sont semblables aux marches aléatoires homogènes à ceci près que le les valeurs des poids peuvent évoluer au cours du temps.

4.I.1.a.2 Sommets visités L'ensemble des sommets susceptibles d'être visités est l'ensemble des sommets v tels qu'il existe un chemin orienté de x_0 à v dont toutes les arêtes ont des probabilités non nulles.
Un sommet va être visité régulièrement si ses voisins le sont également, s'il a de nombreux voisins et si les probabilités de transition sont importantes.
C'est ce que font de base les moteurs de recherche qui consistent à classer les pages suivant le nombre de visites par des marches aléatoires.
Les questions qui se posent sont :
– Quelle est lq loi de la position de la marche après un grand nombre d'itérations ?
– Combien de pas faudra-t-il faire avant de visiter un sommet donné ?
– Quelle est l'influence du choix de la position de départ ?

Exemple 4.I.1.b



4.I.2 Chaînes de Markov

4.I.2.a. **Chaîne de Markov** *(Définition) :*
De nombreux problèmes a priori éloignés des marches aléatoires peuvent s'écrire avec le même formalisme : il suffit que l'objet d'intérêt soit une suite $(S_n)_{n>0}$ telle que :
– le nombre d'arêtes possibles est fini
– on considère des pas de temps discrets
– les probabilités de transition $p_{ij} = \mathbb{P}(s_{n+1} = j | s_n = 1)$ sont invariables dans le temps
– la chaîne est sans mémoire (le pas n + 1 ne dépend que du pas n)
Dans ce cas, on parle de chaînes de Markov finies homogènes.

4.I.2.a.1 Remarque il existe des généralisations :
– à un nombre d'états infini
– à temps continu
– à probabilité de transitions dépendant du temps
– à la prise en compte des k dernières positions, pour k un entier fixé

4.II Classification des chaînes de Markov

4.II.1 Classification des états dune chaîne de Markov

Il y a essentiellement deux types d'états dans une chaîne de Markov :
– ceux qu'on quittera définitivement au bout d'un certain temps pour ne jamais y revenir
– ceux qu'on visitera une infinité de fois si on les visite

4.II.1.a. **État récurrent et état transient/transitoire** *(Définition) :*
Soit v un état d'une chaîne de Markov. $\rho_v = \mathbb{P}$(la marche revient en v | $s_0 = v$).

– si $\rho_v = 1$, le sommet v est dit récurrent
– si $\rho_v < 1$, le sommet v est dit transitoire (ou transient)

Propriété #4.II.1.a.1 :
Si v est récurrent, toute machine qui passe en v passera une infinité de fois en v.

Propriété #4.II.1.a.2 :
Si v est transitoire, toute machine passe une nombre fini de fois en v.

Propriété #4.II.1.a.3 :
On considère la décomposition en composantes fortement connexes de la représentation graphique.
Soit H le graphe ayant un sommet u pour chaque composante connexe C_u de G et tel que $(u; w) \in E(G) \iff$ il existe une arête entre C_u et C_w dans G.
Alors, $\begin{cases} d_H^+(u) \geqslant 1 \implies C_u \text{ est composé d'états transitoires} \\ d_H^+(u) = 0 \implies C_u \text{ est composé d'états récurrents} \end{cases}$

Pour étudier le comportement de la chaîne à long terme, il faut :

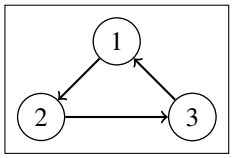
1. Décomposer en composantes fortement connexes et repérer celles correspondant à des états récurrents
2. Déterminer la probabilité d'aboutir dans chacun de ces ensembles étant donné la distribution de départ
3. Déterminer la distribution S_n à l'intérieur des composantes correspondant à des états récurrents

4.II.1.b. **Chaîne de Markov irréductible** *(Définition) :*
Une chaîne de Markov est irréductible si son graphe associé est fortement connexe

4.II.2 Chaînes de Markov périodiques

4.II.2.a. **Chaîne de Markov périodique** *(Définition) :*
Soit P la matrice de transition d'une chaîne de Markov. La chaîne est périodique s'il existe un entier $k > 1$ tel que $P^k = P$

Exemple 4.II.2.b



$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = P^3$$

D'un point de vue du graphe, cela revient à un graphe où tous les cycles sont de longueur multiple de k. Ainsi, pour que ce ne soit pas le cas, il suffit de 1 cycle de longueur 1 (c'est-à-dire une probabilité non nulle de rester sur un état) ou de 2 cycles de longueurs premières entre elles (c'est-à-dire dont le pgcd soit de 1).
Dans la suite, on considère les chaînes apériodiques.

4.III Distribution asymptotique pour une chaîne irréductible apériodique

4.III.1 Rappels d'algèbre linéaire

4.III.1.a Valeurs et vecteurs propres

Soit A une matrice réelle carrée d'ordre n.
$\lambda \in \mathbb{R}$ est une valeur propre et $X \in \mathbb{R}^n$ est un vecteur propre à droite de A associé à la valeur propre λ si $AX = \lambda X$
$\lambda \in \mathbb{R}$ est une valeur propre et $X \in \mathbb{R}^n$ est un vecteur propre à gauche de A associé à la valeur propre λ si ${}^tXA = \lambda {}^tX$, c'est-à-dire si ${}^t({}^tXA) = \lambda X$, donc si X est vecteur propre à droite de ${}^tA$ associé à la valeur propre λ

4.III.1.a.1 Remarques

- A et ${}^tA$ ont les mêmes valeurs propres, mais les vecteurs propres à droite de A sont vecteurs propres à gauche de ${}^tA$ et vice-versa
- L'ensemble des vecteurs propres associés à λ est un sous-espace vectoriel de $\mathbb{R}^n$
- Tout vecteur propre dont la somme des coefficients est non nulle peut être transformé en un vecteur propre de somme 1
- Un vecteur propre dont toues les coordonnées sont positives peut-être assimilé à une distribution de probabilité

Exemple 4.III.1.b

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -2 & 3 & 1\,0 \\ & -1 & -1 \end{pmatrix} \textit{ et } X = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

X *est un vecteur propre à droite de* A *associé à la valeur propre* $\lambda = 2$ *et* $X' = \frac{1}{3}X = \begin{pmatrix} \frac{1}{3} \\ 0 \\ \frac{2}{3} \end{pmatrix}$ *est un vecteur propre à droite de* A *de somme* 1 *associé à la valeur propre* $\lambda = 2$.

Propriété #4.III.1.b.1 :
Si A est une matrice carrée réelle diagonalisable (c'est-à-dire qu'il existe Q inversible et D diagonale telles que $A = QDQ^{-1}$), il existe ...//TODO

4.III.1.c Matrices stochastiques

4.III.1.d. **Matrice stochastique** *(Définition) :*
Une matrice P est stochastique si
– $\forall i \neq j, 0 \leqslant p_{ij} \leqslant 1$
– $\sum_j p_{ij} = 1$

Propriété #4.III.1.d.1 :
Le vecteur $\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ est un vecteur propre associé à la valeur propre 1 pour toute matrice stochastique.
De plus, toute autre valeur propre λ vérifie $|\lambda| \leq 1$

Démonstration de la propriété – $P\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} \sum_j p_{1j} \\ \vdots \\ \sum_j p_{ij} \\ \vdots \\ \sum_j p_{nj} \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ car P est stochastique

– Soit λ une autre valeur propre et $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ un vecteur associé. Soit x_k le coefficient non nul de plus grande valeur absolue.

Alors, $P\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ donne pour la ligne k,

$$\sum_j p_{kj} x_j = \lambda x_k$$
$$\implies \sum_j \frac{p_{kj} x_j}{x_k} = \lambda x_k$$
$$\implies \sum_j p_{kj} \left|\frac{x_j}{x_k}\right| \geqslant |\lambda x_k|$$
$$\implies \sum_j p_{kj} \geqslant \sum_j p_{kj} \left|\frac{x_j}{x_k}\right| \geqslant |\lambda x_k|$$
$$\implies 1 = \sum_j p_{kj} \geqslant |\lambda x_k|$$

□

4.III.2 Retour aux chaînes de Markov

Propriété #4.III.2..2 :
On considère une chaîne de Markov à N états et on note X_n le vecteur de distribution au bout de n pas.
on a $X_n \in \mathbb{R}^N$.
Alors, ${}^tX_n = {}^tX_0 P^n$

Démonstration de la propriété

Il suffit de démontrer que ${}^tX_n = {}^tX_0P^n$ en procédant par récurrence.

Initialisation : Pour n = 0, c'est évident

Itération pour n ***fixé*** : Supposons que la propriété est vraie pour un rang n fixé : $\mathbb{P}(S_{n+1} = i) = \sum_{j \text{ état}} \mathbb{P}(S_{n+1} = i, S_n = j)$

$\implies \mathbb{P}(S_{n+1} = i) = \sum_{\text{état } j} \mathbb{P}(S_{n+1} = i \mid S_n = j)\,\mathbb{P}(S_n = j)$

$\implies \mathbb{P}(S_{n+1} = i) = \sum_{\text{état } j} p_{ji}\,\mathbb{P}(S_n = j)$ car la chaîne de Markov passe en i avec la probabilité p_{ji} si elle est en j

$\implies X_{n+1}(i) = \mathbb{P}(S_{n+1} = i) = \sum_{\text{état } j} p_{ji} X_n(i)$

Or la i-ème cordonnée de ${}^tX_nP = \begin{pmatrix} X_n(1) \\ \vdots \\ X_n(N) \end{pmatrix} \begin{pmatrix} p_{11} \cdots p_{1N} \\ \vdots \\ p_{N1} \cdots p_{NN} \end{pmatrix}$ vaut $\sum_j X_n(j)p_{ji}$.

On a par conséquent ${}^tX_{n+1} = {}^tX_nP$

Donc ${}^tX_n = {}^tX_0P^n \implies {}^tX_{n+1} = {}^tX_0P^nP = {}^tX_0P^{n+1}$

Conclusion : Par conséquent, l'hypothèse est vraie pour tout entier n positif ou nul

Résultat : D'où $\forall n \in \mathbb{N}, \forall n \in \mathbb{N}, {}^tX_n = {}^tX_0P^n$

□

Pour connaître la répartition à long terme, il suffit de connaître la répartition à l'origine et la matrice de transition.
Il est en fait possible de montrer que l'influence de la répartition à l'origine X_0 devient nulle quand n devient grand.

4.III.2.a. **Mesure invariante** *(Définition) :*

On appelle mesure invariante liée à P tout vecteur μ tel que :
- $\forall i, 0 \leqslant \mu_i \leqslant 1$
- $\sum_i \mu_i = 1$
- ${}^t\mu P = {}^tmu$

En d'autres termes, une mesure invariante est un vecteur propre à gauche de P lié à la valeur propre 1.
Le terme mesure invariante vient du fait que ${}^tX_{n+1} = {}^tX_nP$: si $X_n = \mu$, alors $X_m = \mu\ \forall m \geqslant n$

Théorème #4.III.2.a.1 : Théorème de Perron-Frobenius

Soit P la matrice d'une chaîne de Markov irréductible et apériodique. Alors :

a) Toute valeur propre $\lambda \neq 1$ de P vérifie $|\lambda| \leqslant 1$

b) La valeur propre 1 est simple et ses vecteurs propres associés ont tous leurs coefficients de même signe.

4.III.2.a.1 Remarque Dire qu'une valeur propre Λ est simple signifie que si X est un vecteur propre associé à Λ, tout autre vecteur propre associé à Λ est un multiple de X.
En particulier, dire que 1 est valeur propre simple signifie qu'il n'y a qu'un seul vecteur propre à gauche dont la somme des coefficients vaut 1.
En effet, soient deux vecteurs propres à gauche X et Y de cordonnées respectivement x_i et y_i dont la somme des coefficients vaille 1. Alors, $(\forall\ i,\ x_i = \alpha y_i) \implies \left(\sum_i x_i = \alpha \sum_i y_i\right)$ et $\sum_i x_i = \sum_i y_i = 1 \implies \alpha = 1 \implies x = y$.

Démonstration du théorème

Démonstration admise. □

Soit μ le vecteur propre à gauche de P associé à la valeur propre 1 dont la somme des coefficients vaut 1.
Comme le théorème dit que pour toutes les coordonnées de μ sont de même signe, μ est une mesure invariante.
Par conséquent, il existe une mesure invariante unique pour toute chaîne de Markov irréductible apériodique.

Théorème #4.III.2.a.2 :
Soient P la matrice d'une chaîne de Markov apériodique irréductible de mesure invariante μ et λ_2 la valeur propre à gauche de P différente de 1 de plus grande valeur absolue.
Alors la distribution X_n au bout de n pas tend vers μ quand n tend vers l'infini. De plus, la vitesse de convergence est en $|\lambda_2|^n$

Démonstration du théorème
Dans le cas où P est diagonalisable.
Soient $\lambda_1, \cdots, \lambda_n$ les valeurs propres de P comptées avec leur multiplicité et rangées par ordre de valeur absolue décroissante (ainsi, $\lambda_1 = 1$).
P étant diagonalisable, il existe une base $(e_1, \cdots, e_N)$ de $\mathbb{R}^N$ telle que ${}^te_iP = \lambda_i {}^te_i$

e_1 et μ étant multiples l'un de l'autre, $(\mu, \cdots, e_N)$ est bien une base de $\mathbb{R}^n$ et on peut écrire ${}^tX_0 = \alpha_1 {}^t\mu + \sum_{i=2}^{N} \alpha_i {}^te_i$.

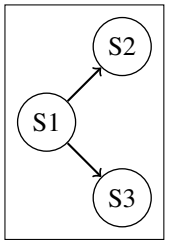
Or, ${}^tX_n = {}^tX_0P^n$, donc ${}^tX_n = \alpha_1 {}^t\mu + \sum_{i=2}^{N} {\lambda_i}^n \alpha_i {}^te_i$
Comme $\forall i > 1, \lambda_i < 1 \implies \lim_{n\to\infty} {\lambda_i}^n \to 0$, on a que $\lim_{n\to\infty} {}^tX_n = \alpha_1 {}^t\mu$.
De plus, comme tous les vecteurs sont de somme 1, on a forcément $\alpha_1 = 1$.
Donc, $\left({}^tX_n\right)_{n\in\mathbb{N}}$ tend vers μ et ${}^tX_n - \mu$ est dominé par $|\lambda_2|^n$. □

Par conséquent, pour connaître le comportement à longue échéance d'une chaîne de Markov irréductible apériodique, il suffit de chercher l'unique vecteur propre à gauche de somme 1 associé à la valeur propre 1.

4.IV Distribution asymptotique dans d'autre cas

4.IV.1 Dans le cas non-irréductible et apériodique

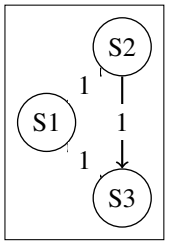
Exemple 4.IV.1.a



Soit μ_2 la mesure invariante si on se restreint à S2 *et μ_3 la mesure invariante si on se restreint à* S3.
Dans ce cas, toute mesure de la forme $\mu = \begin{pmatrix} 0 \\ \alpha\mu_2 \\ (1-\alpha)\mu - 3 \end{pmatrix}$ *est une mesure invariante, mais la mesure vers laquelle on tend dépend de* X_0 *et n'est donc pas unique.*

4.IV.2 Dans le cas irréductible et périodique

Exemple 4.IV.2.a



Si $X_0 = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $\forall n \in \mathbb{N}$, $X_{3n} = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $X_{3n+1} = \begin{pmatrix}0\\1\\0\end{pmatrix}$ *et* $X_{3n+2} = \begin{pmatrix}0\\0\\1\end{pmatrix}$
On perd donc la convergence.

4.V Méthode de Monte-Carlo

Revenons au problème du voyageur de commerce.

4.V.1 Problème et approche aléatoire

On dispose m villes et numérotées de 1 à m qu'un voyageur doit visiter en revenant à la fin à son point de départ. On note $\delta(ij)$ la distance entre la ville i et la ville j.

4.V.1..1 Problème : trouver le parcours qui minimise la distance parcourue.
Cela revient à trouver une permutation σ des villes (c'est-à-dire l'ordre dans lequel elles sont parcourues) telle que $H(\sigma) = \sum_{i=1}^{m-1} \delta(\sigma_i, \sigma_{i+1}) + \delta(\sigma_m, \sigma_1)$ soit minimal.
Ce problème est NP-complet (c.f. Problème du voyageur de commerce).
Une manière d'obtenir une solution approchée est d'utiliser une approche aléatoire.
Pour cela, on voudrait pouvoir tirer des permutations au hasard suivant une loi μ qui favorise les σ vérifiant $H(\sigma)$ petit.
Par exemple, $\mu(\sigma) = C\frac{1}{H(\sigma)}$ où $C = \frac{1}{\sum_\sigma \frac{1}{H(\sigma)}}$, ou même $\mu(\sigma) = Ce^{-H(\sigma)}$.
Le problème pour tirer selon μ est que la constante C ne peut pas être calculée car elle est composée d'une somme de n! termes.
La solution à ce problème va être de construire une chaîne de Markov dont μ est la mesure invariante. Une fois qu'on a une telle chaîne, on considèrera les valeurs successives des trajectoires pour des tirages.

4.V.2 Approche de Métropolis

On veut tirer selon une loi proportionnelle à une fonction f.
La méthode de Métropolis est en deux étapes :

1. on construit une chaîne de Markov irréductible et symétrique déterminée par une matrice de transition entre les états P
   Pour le voyageur de commerce, on autorise la transition de σ à σ' si σ' est obtenue à partir de σ en échangeant deux villes, et toutes les transitions sont équiprobables $\frac{1}{\binom{m}{2}}$
2. on considère la chaîne de Markov de transition Q avec $Q(x, y) = P(x, y)\min\left(1, \frac{H(\sigma)}{H(\sigma')}\right)$
   Pour le voyageur de commerce : $Q(\sigma, \sigma') = P(\sigma, \sigma')\min\left(1, \frac{H(\sigma)}{H(\sigma')}\right)$.
   $H(\sigma') \leqslant H(\sigma) \implies \min\left(1, \frac{H(\sigma)}{H(\sigma')}\right) = 1 \implies Q(\sigma, \sigma') = P(\sigma, \sigma')$: si P me propose σ' et que $H(\sigma') \leqslant H(\sigma)$, je le prends avec sa probabilité dans P (qui est équiprobable)
   $H(\sigma') > H(\sigma) \implies \min\left(1, \frac{H(\sigma)}{H(\sigma')}\right) = \frac{H(\sigma)}{H(\sigma')} \implies Q(\sigma, \sigma') = \frac{H(\sigma)}{H(\sigma')}$: si P me propose σ' et que $H(\sigma') \leqslant H(\sigma)$, je le prends mais avec la probabilité $\frac{H(\sigma)}{H(\sigma')}$ plus petite que l'originale. En effet, l'intérêt de garder avec une probabilité non nulle un "saut" (changement) désavantageux est de sortir des minima locaux.

Théorème #4.V.2..1 :
Soit μ la mesure définie par $\mu(x) = Cf(x)$ où $C = \frac{1}{\sum_x f(x)}$. μ est la mesure invariante associée à la chaîne irréductible Q.

Démonstration du théorème

Soient x et y deux états avec $P(x, y) \neq 0$ et $f(x) \geqslant f(y)$.

Alors, $\mu(x)q(x, y) = Cf(x)q(x, y) = Cf(x)p(x, y)\dfrac{f(y)}{f(x)}$

$\implies \mu(x)q(x, y) = Cf(y)p(y, x) = \mu(y)q(y, x)$

Pour $f(x) < f(y)$, on pet faire le même calcul et on obtient que :

$$\forall\ x, y,\ \mu(x)q(x, y) = \mu(y)q(y, x)$$

On somme cette égalité sur tous les y :

$$\mu(x)\underbrace{\sum_y q(x, y)}_{1} = \sum_y \mu(y)q(y, x) = (\mu Q)(x)$$

Vu que ceci est vrai pour tout x, on s'aperçoit que $\mu = \mu Q$, donc que μ est une mesure invariante (donc l'unique mesure invariante). □

On ne sait pas calculer μ directement, mais on sait tirer suivant μ en lançant une chaîne de Markov de transition Q.

4.V.3 Recuit simulé

L'idée est d'améliorer la vitesse avec laquelle on s'approche du minimum en jouant sur l'évolution des transitions (en pénalisant de plus en plus les grands sauts).

On rend la chaîne non homogène en diminuant avec le temps la probabilité de faire des sauts non avantageux.

Définitions