Théorie des graphes

Représentation des graphes

François Delbot

27 août 2019

Introduction

Il existe de nombreuses manières de représenter les graphes en mémoire. Parmi les plus connues, on peut citer les représentations par :

- Matrice.
- Liste d'adjacence.
- Liste d'arêtes.
- Champs de bits.

Mais cela ne sont que des exemples, adaptables aux besoins réels de votre application.

Introduction

Attention

Il n'existe pas de meilleure représentation mémoire d'un graphe. Par contre, étant donnée une situation précise, il existe des représentations qui ne sont pas du tout adaptées.

C'est à vous qu'il revient de choisir la bonne représentation, voire d'en concevoir une nouvelle. Il s'agit généralement d'un compromis entre quantité de mémoire utilisée, temps d'exécution et ... temps de développement.

Les questions à se poser 1/2

Étant donné l'application que vous souhaitez réaliser, il convient de se poser un certain nombre de questions sur le graphe :

1. Est-ce que le graphe est orienté ?
2. Est-ce que les sommets ont des attributs ?
3. Est-ce que les arêtes ont des attributs ?
4. Quel sera le nombre (en ordre de grandeur au moins) maximum de sommets à manipuler ?
5. Quel sera le nombre (en ordre de grandeur au moins) maximum d'arêtes à manipuler ?

Les questions à se poser 2/2

Étant donné l'application que vous souhaitez réaliser, il convient de se poser un certain nombre de questions sur votre environnement et le contexte d'utilisation :

1. Quelles sont les contraintes de mon application (mémoire, vitesse, accès aux données) ?
2. Quelles seront les opérations de manipulation de graphe le plus souvent utilisées ?
3. Est-ce qu'il y a des contraintes matérielles (processeur, mémoire) ?
4. ...

Étude préalable

En fonction des réponses apportées à ces différentes questions, vous pourrez faire un choix de conception. Si vous souhaitez dépasser le stade du prototype, vous devez absolument détailler l'ensemble de vos décisions et être capable de les défendre.

Fil conducteur

Deux graphes

Voici deux graphes, l'un non-orienté et l'autre orienté. Nous allons les utiliser pour comparer les différentes représentations.

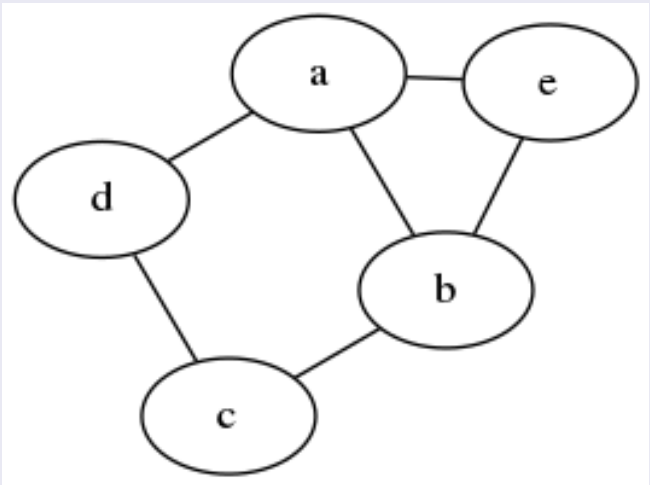


FIGURE – Graphe 1

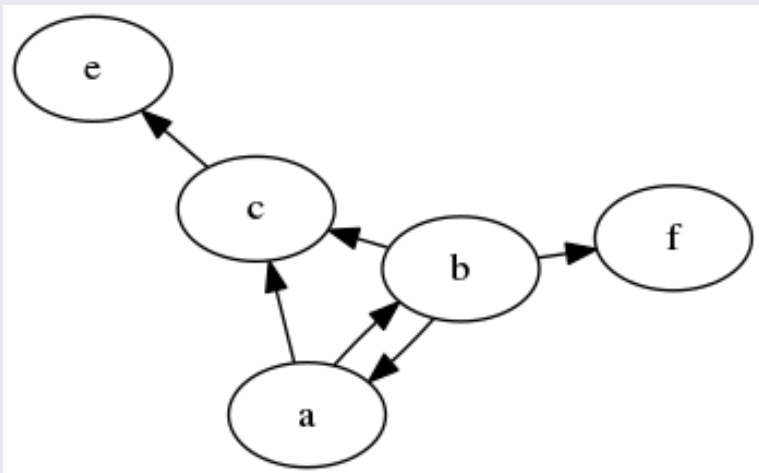


FIGURE – Graphe 2

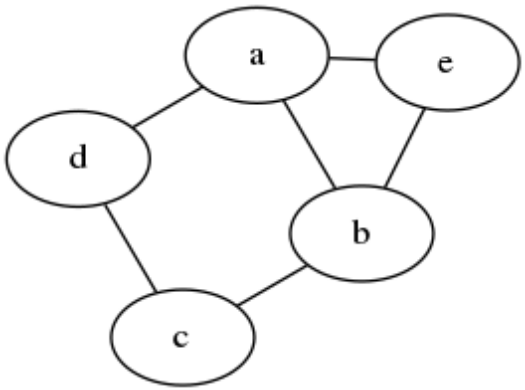
Représentation par matrice

Représentation

Soit $G = (V, E)$ un graphe de taille n. On peut associer à un graphe une matrice carrée M de taille $n \times n$.

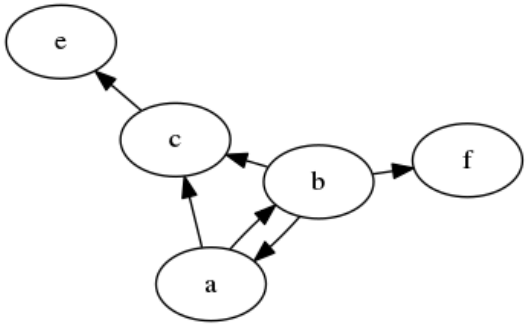
1. La colonne i de la matrice est associée au $i$ème sommet du graphe.
2. La ligne i de la matrice est associée au $i$ème sommet du graphe.
3. La valeur présente dans la case d'indice (x, y) représente l'arête (arc) allant du sommet x vers le sommet y. Un 0 signifie que l'arête (arc) (x, y) n'existe pas tandis qu'un 1 signifie que cette arête (arc) existe.

Exemple de matrice associée à un graphe non-orienté



	a	b	c	d	e
a	0	1	0	1	1
b	1	0	1	0	1
c	0	1	0	1	0
d	1	0	1	0	0
e	1	1	0	0	0

Exemple de matrice associée à un graphe orienté



	a	b	c	e	f
a	0	1	1	0	0
b	1	0	1	0	1
c	0	0	0	1	0
e	0	0	0	0	0
f	0	0	0	0	0

Quantité de mémoire utilisée

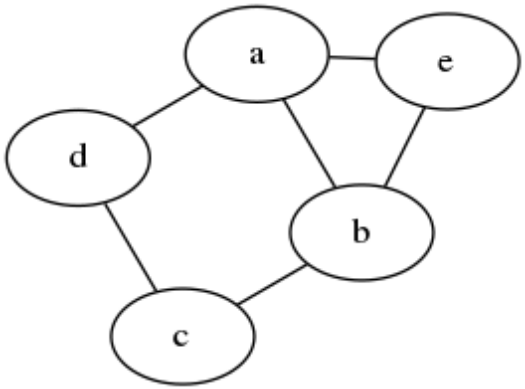
Soit n la taille du graphe. La quantité de mémoire utilisée par cette représentation est de n^2 emplacements mémoire.

On peut toutefois remarquer que la matrice associée à un graphe non-orienté est symétrique par la diagonale puisque l'existence de l'arête (x, y) implique l'existence de l'arête (y, x). Ainsi, il est possible de n'utiliser que $\frac{n^2}{2}$ emplacements mémoire dans ce cas.

De plus, si le graphe non-orienté est sans boucle, on peut se passer des valeurs présentes dans la diagonale. On arrive à une utilisation de $\frac{n^2}{2} - n$ emplacements mémoire dans ce cas.

Exemple de matrice associée à un graphe non-orienté

En reprenant les remarques précédentes, il n'est pas nécessaire d'enregistrer les valeurs présentes en rouge.



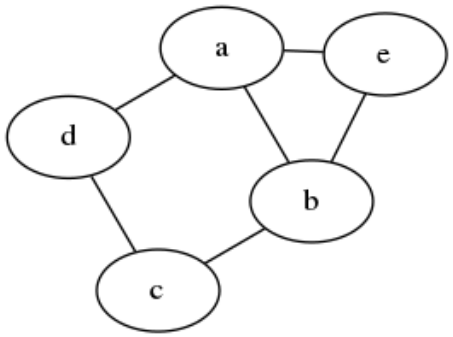
	a	b	c	d	e
a	0	1	0	1	1
b	1	0	1	0	1
c	0	1	0	1	0
d	1	0	1	0	0
e	1	1	0	0	0

Avantages et inconvénients de cette représentation

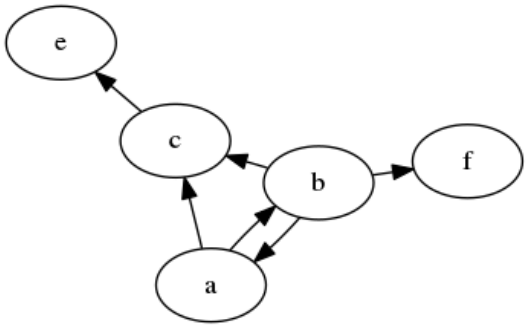
1. Supporte facilement l'ajout d'attributs sur les arêtes (arcs).
2. L'accès à une arête, pour lecture, ajout, suppression ou modification, se fait directement, soit en $O(1)$.
3. La quantité de mémoire utilisée est en $O(n^2)$.
4. Déterminer le degré, ou le voisinage d'un sommet se fait en parcourant tous les voisins d'un sommet, soit en $O(n)$.
5. L'ajout ou la suppression d'un sommet est une opération délicate.
    1. L'ajout d'un sommet nécessite l'ajout d'une colonne et d'une ligne.
    2. La suppression d'un sommet implique de supprimer la ligne et la colonne qui correspondent au sommet.

Représentation par liste d'arêtes

Ce format impose de donner la liste des arêtes du graphe. Cette représentation possède l'avantage d'être extrêmement simple, et ne nécessite par de structure de donnée particulière.



(a,b), (a,d), (a,e), (b,c), (b,e), (c,d)



(a,c), (a,b), (b,a), (b,c), (b,f), (c,e)

Représentation par liste d'arêtes

Cependant cette représentation possède un certain nombre d'inconvénients :

1. Savoir si une arête est présente nécessite un temps en $O(m)$.
2. ajouter et/ou supprimer une arête sans vérification peut se faire en $O(1)$. Si on souhaite vérifier que l'arête n'est pas déjà présente, cela nécessite donc un temps en $O(m)$.
3. déterminer la liste des voisins d'un sommet, ou calculer son degré nécessite un temps en $O(m)$.
4. l'ajout d'un sommet se fait de manière transparente.

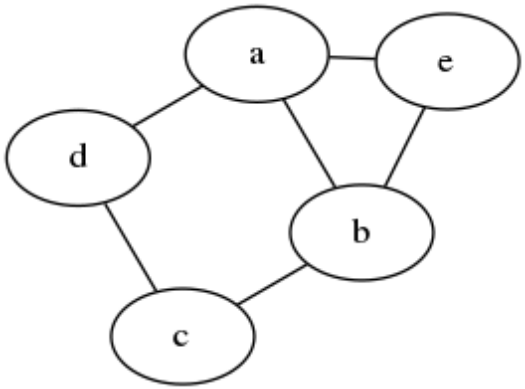
Si le nombre d'arêtes est faible, alors cette représentation est pratique. Malheureusement le nombre d'arêtes peut être au maximum de $n \times (n-1)$ soit en $O(n^2)$, rendant couteuses les différentes opérations.

Représentation par liste d'adjacence

Comme nous venons de le voir, la représentation par liste d'arêtes pose de nombreux problèmes. Occupation mémoire importante lorsque le nombre d'arêtes est important, temps d'exécution important pour certaines fonctions ...

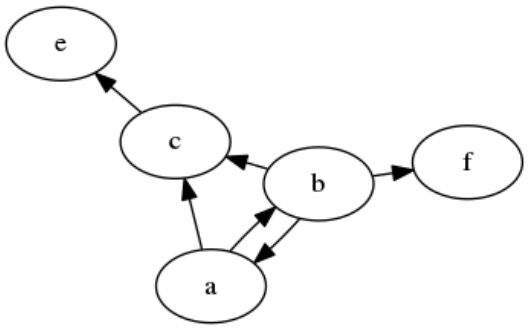
Cependant, l'idée d'une structure adaptée aux graphes possédant peu d'arêtes, mais néanmoins efficace en temps d'exécution reste intéressante. La représentation par liste d'adjacence tente d'améliorer la représentation par liste d'arêtes en permettant d'accéder directement aux arcs (et arêtes) ayant pour origine un sommet donné.

Exemple de liste d'adjacence associée à un graphe non-orienté



a	b	d	e
b	a	c	e
c	b	d	
d	a	c	
e	a	b	

Exemple de matrice associée à un graphe orienté



a	b	c	
b	a	c	f
c	e		
e			
f			

Quantité de mémoire utilisée

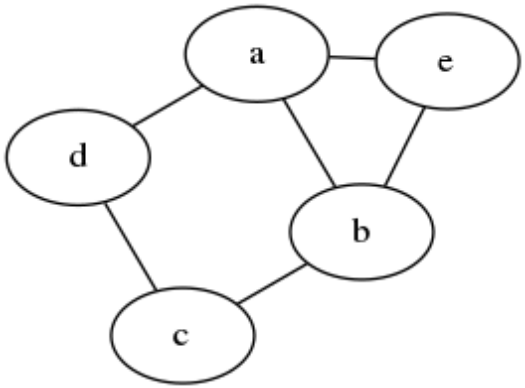
Soit $G = (V, E)$ un graphe. La quantité de mémoire utilisée par cette représentation est de $|V|$ emplacements mémoire pour la liste des sommets. Chaque sommet est associé à une liste d'arêtes (arcs).

1. Si G est non-orienté, chaque arête va apparaitre deux fois. La quantité de mémoire utilisée, au total, sera de $|V| + 2 \times |E|$.
2. Si G est orienté, chaque arc apparait une fois exactement. La quantité de mémoire utilisée, au total, sera de $|V| + |E|$.

Bien entendu, il est possible de ne pas dupliquer les arêtes si le graphe n'est pas orienté. Par exemple en plaçant les arêtes uniquement dans l'ordre lexicographique. Cependant, ce choix aura un impact sur la performance de certaines fonctions (calcul du voisinage par exemple).

Exemple de matrice associée à un graphe non-orienté

En reprenant la remarque précédente, il n'est pas nécessaire d'enregistrer les valeurs présentes en rouge.



a	b	d	e
b	a	c	e
c	b	d	
d	a	c	
e	a	b	

Avantages et inconvénients de cette représentation

1. L'accès à une arête, pour lecture, suppression ou modification, se fait en parcourant la liste des voisins des sommets concernés $O(N(u) + N(v))$.
2. L'ajout d'une arête peut se faire en temps constant si on l'ajoute au début de la liste. Mais l'insertion triée par ordre lexicographique peut aussi avoir un intérêt et nécessitera un temps en $O(N(u) + N(v))$, avec u et v les sommets concernés.
3. La quantité de mémoire utilisée est de $|V| + 2 \times |E|$.
4. Déterminer le degré, ou le voisinage d'un sommet v se fait en parcourant uniquement les voisins d'un sommet, soit en $O(N(v))$.

Avantages et inconvénients de cette représentation

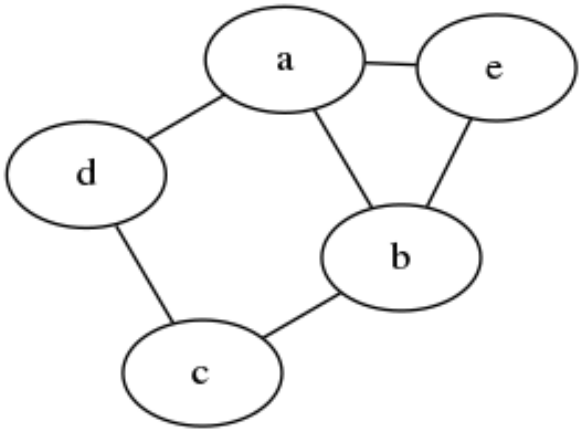
L'ajout ou la suppression d'un sommet est une opération plus simple qu'avec la représentation par matrice.

1. L'ajout d'un sommet se fait très facilement : il suffit d'augmenter la taille de la liste des sommets et associer au nouveau sommet une liste vide.
2. La suppression d'un sommet implique de :
   1. Supprimer l'ensemble des arêtes (arcs) ayant ce sommet pour extrémité.
   2. Supprimer la liste des voisins associée à ce sommet.
   3. Diminuer la liste des sommets.

Représentation par champs de bits

Une représentation par champs de bits est une version spécialisée de la représentation par matrice. Dans cette représentation, un seul bit sera utilisé pour déterminer la présence ou non d'une arête (d'un arc).

Exemple de champs de bits associé à un graphe



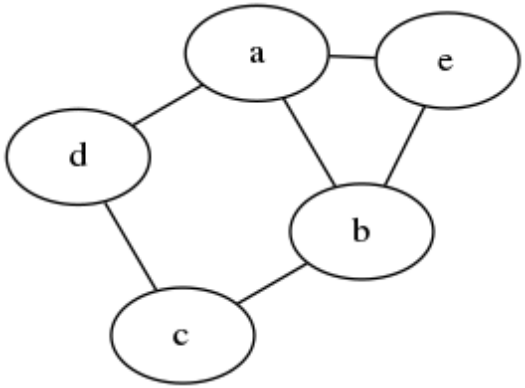
abcde
a : 01011
b : 10101
c : 01010
d : 10100
e : 11000

Or, chaque ligne correspond à la représentation binaire d'un entier. Il suffit donc de n'enregistrer que cet entier :

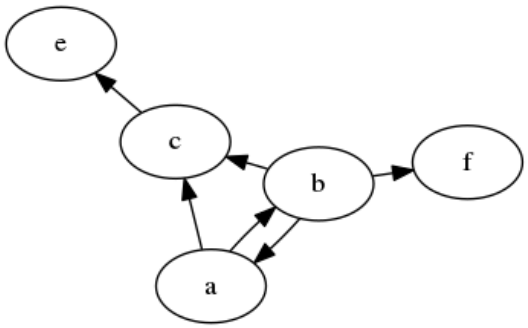
a : 11 $(0^4 + 1^3 + 0^2 + 1^1 + 1^0)$
b : 21 $(1^4 + 0^3 + 1^2 + 0^1 + 1^0)$
c : 10 $(0^4 + 1^3 + 0^2 + 1^1 + 0^0)$
d : 20 $(1^4 + 0^3 + 1^2 + 0^1 + 0^0)$
e : 24 $(1^4 + 1^3 + 0^2 + 0^1 + 0^0)$

Exemple de champs de bits associé à un graphe orienté

On peut donc représenter un graphe par un tableau d'entiers :



a	b	c	d	e
11	21	10	20	e : 24



a	b	c	e	f
12	21	2	0	0

Génération de graphes particuliers

Dans cette partie, nous allons considérer différentes familles de graphes et voir comment les générer algorithmiquement. Certains sont triviaux, d'autres nécessitent des méthodes algorithmiques. :

1. Graphe complet.
2. Graphe aléatoire de Erdös-Renyi.
3. Grille $l \times h$.
4. Arbre aléatoire.
5. Hypercube de dimension d.

Génération de graphes particuliers

Graphe complet

Un graphe complet est un graphe pour lequel chaque sommet est voisin de tous les autres sommets.

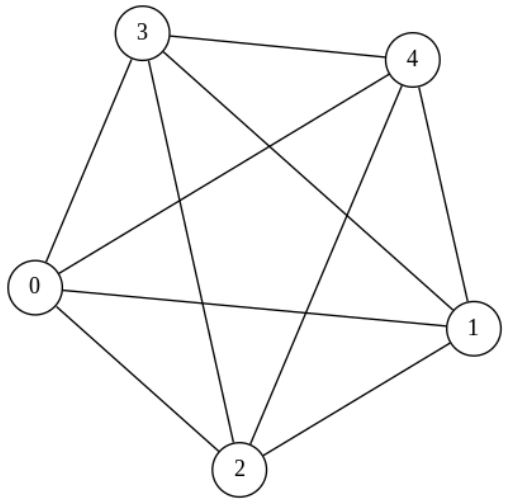


FIGURE – Exemple de graphe complet à 5 sommets

Générer un tel graphe est trivial, puisqu'il suffit d'insérer toutes les arêtes possibles à partir d'un graphe vide.

Génération de graphes particuliers

Graphe aléatoire de Erdös-Renyi

Un graphe aléatoire de Erdös-Renyi est un graphe pour lequel chaque arête existe avec une certaine probabilité constante (éventuellement liée à la taille du graphe).

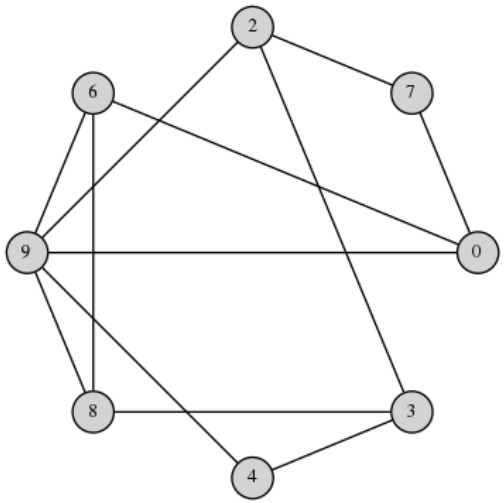


FIGURE – Exemple de graphe aléatoire de Erdös-Reyni avec une probabilité p=0.2

Génération de graphes particuliers

Graphe aléatoire de Erdös-Renyi

Pour chaque couple de sommets (i, j), ajouter l'arête (i, j) avec probabilité p. Attention, si le graphe n'est pas orienté, il ne faut considérer l'arête qu'une seule fois.

```
1       for(i=0; i<n; i++)
2       {
3           for(j=i; j<n; j++)
4           {
5               val = tirage aleatoire
6               si val <= p
7                   ajouter l arete (i,j)
8           }
9       }
```

Génération de graphes particuliers

Grille de taille $h \times l$

Une grille est un graphe à deux dimensions h et l dont les sommets correspondent aux points (coordonnées entières) du plan, les abscisses comprises dans la plage $1, \ldots, l$, les ordonnées étant comprises dans la plage $1, \ldots, h$. Deux sommets étant reliés par une arête lorsque les points correspondants sont à une distance d'exactement 1.

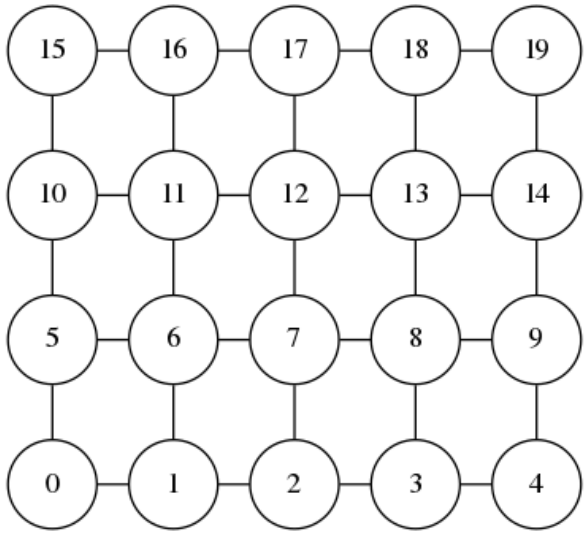


FIGURE – Exemple de grille de taille 5×4

Génération de graphes particuliers

Grille de taille $h \times l$

Pour générer un tel graphe, une méthode très simple consiste à :

1. Générer un graphe vide de taille $h \times l$.
2. Ajouter toutes les arêtes verticales.
3. Ajouter toutes les arêtes horizontales.

Génération de graphes particuliers

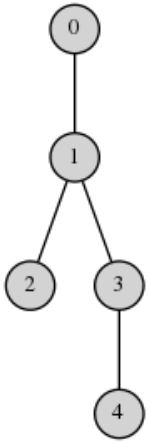
Grille

```
 1        for(i=0; i<h; i++)
 2        {
 3            for(j=0; j<l-1; j++)
 4                ajouter arete(j+l*i,(j+l*i)+1)
 5        }
 6        for(i=0; i<l; i++)
 7        {
 8            for(j=0; j<h-1; j++)
 9                ajouter arete(i+l*j,i+l*(j+1))
10        }
```

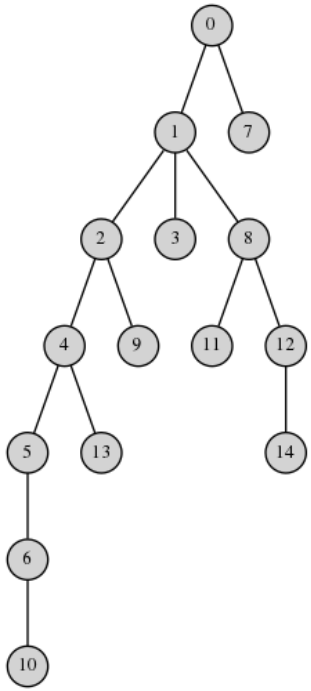
Génération de graphes particuliers

Arbre

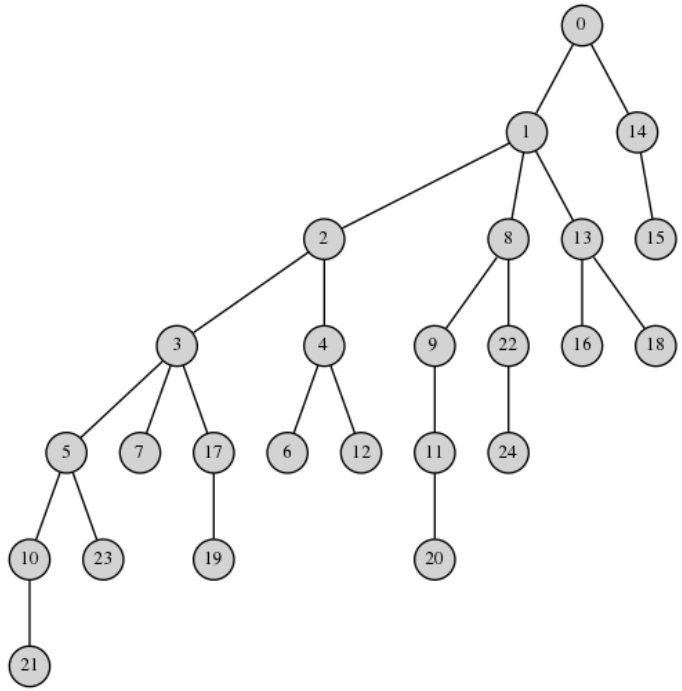
Un arbre est un graphe connexe sans cycle. Il existe de nombreuses sous-catégories d'arbres (réguliers de degré x, aléatoires, ...).



Arbre de taille 5



Arbre de taille 15



Arbre de taille 25

Génération de graphes particuliers

Arbre

Pour générer un arbre aléatoire il suffit de remarquer qu'un arbre contient exactement $n-1$ arêtes. Il est possible d'ajouter ces arêtes une à une en faisant attention à ne pas créer de cycle. Une méthode consiste à considérer chaque sommet, dans l'ordre, en créant une arête ayant comme extrémité de ce sommet et pour seconde extrémité un sommet préalablement considéré (éventuellement choisi au hasard).

Génération de graphes particuliers

Hypercube

Un hypercube peut être vu comme la généralisation à n dimensions d'un cube. Une manière astucieuse de construire un hypercube de dimension d consiste à relier chaque sommet i parmi les 2^d sommets à tous les sommets dont la décomposition en binaire est à une distance de Hamming de 1 de i.

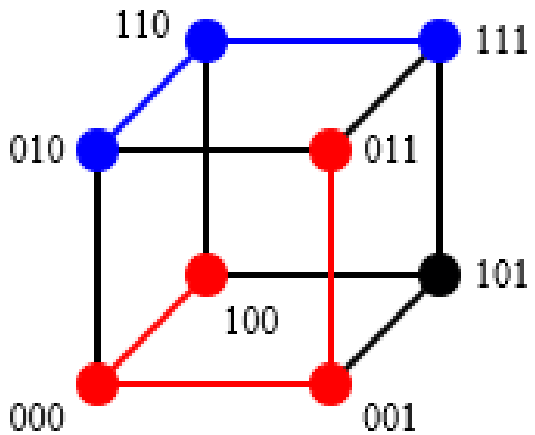
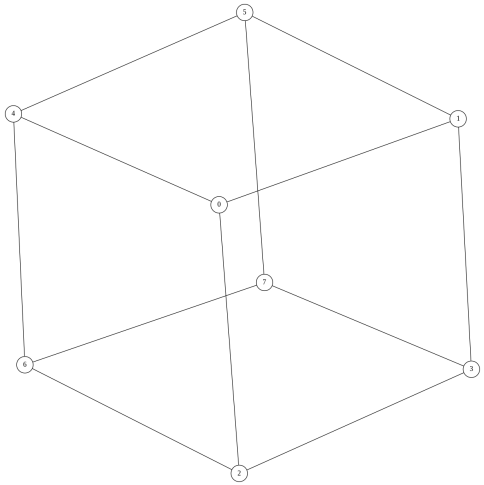


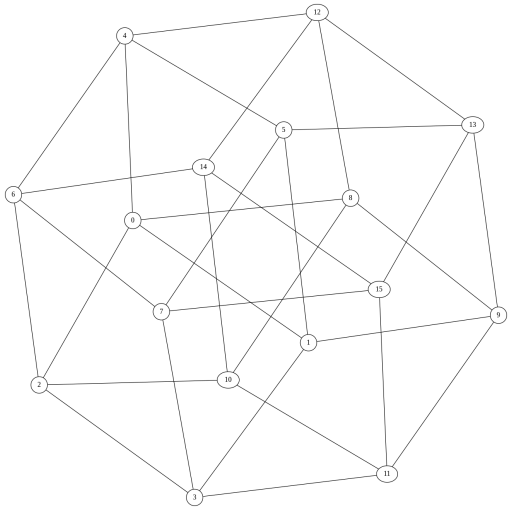
FIGURE – Exemple d'hypercube de dimension 3

Génération de graphes particuliers

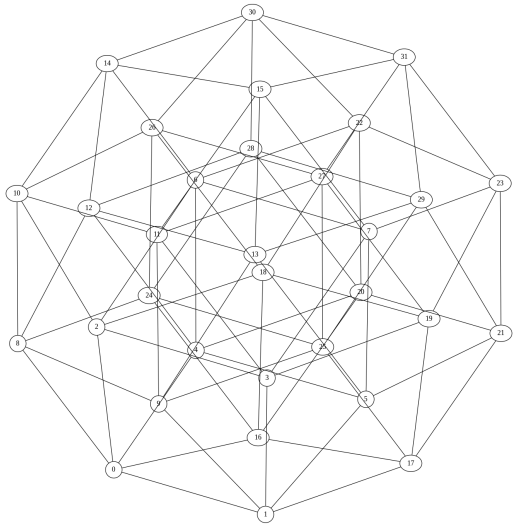
Hypercube



Hypercube de dimension 3



Hypercube de dimension 4



Hypercube de dimension 5

Génération de graphes particuliers

Hypercube

On utilise l'opérateur de décalage de bit $<<$ qui décale les bits d'une variable d'un bit vers la gauche. Par exemple $100101 << 1$ donne 001010. L'opérateur & permet d'obtenir une conjonction logique (un ET logique). Soit d la dimension de l'hypercube.

```
 1        int taille=1<<d;
 2        for(i=0; i<taille; i++)
 3        {
 4            for(j=0; j<d; j++)
 5            {
 6                if((1<<j)&i)
 7                    ajouter arete (i,i-(1<<j))
 8                else
 9                    ajouter arete (i,i+(1<<j))
10            }
11        }
12        return g;
13    }
```

La bibliothèque GraphViz

Graphviz (`https://www.graphviz.org/`) est un ensemble de logiciels (et bibliothèques) open source de visualisation de graphes.

A partir de la description d'un graphe dans un langage texte simple (langage dot), ils permettent de générer des images de ce graphe dans différents formats utiles (jpg, png, svg, pdf, ps ...).

Graphviz offre de nombreuses options forme, couleurs, police ...), mais aussi différents algorithmes de positionnement.

La bibliothèque GraphViz

GraphViz contient plusieurs programmes utilisant différents algorithmes de placement. A vous de les essayer pour déterminer celui qui vous conviendra le mieux. Les résultats peuvent différer de manière très importante en fonction de votre graphe :

1. dot : outils par défaut.
2. neato
3. fdp
4. sfdp
5. twopi
6. circo

La bibliothèque GraphViz

Exemple de graphe décrit dans le langage dot :

```
graph mon_graphe {
    size="5.0,5.0";
    ratio="fill";
    node [height=0.6, width=0.6];
    0 -- 1 -- 2 -- 3 -- 4;
    5 -- 6 -- 7 -- 8 -- 9;
    10 -- 11 -- 12 -- 13 -- 14;
    15 -- 16 -- 17 -- 18 -- 19;
    0 -- 5 -- 10 -- 15 ;
    1 -- 6 -- 11 -- 16 ;
    2 -- 7 -- 12 -- 17 ;
    3 -- 8 -- 13 -- 18 ;
    4 -- 9 -- 14 -- 19 ;
}
```

La bibliothèque GraphViz

- **Ligne 1 :** graph indique que l'on manipule un graphe non orienté.
  mon_graph est le nom (facultatif) donné au graphe.
- **Ligne 2 :** permet de fixer la taille de l'image produite. (facultatif)
- **Ligne 3 :** indique au programme d'utiliser tout l'espace à sa disposition.
- **Ligne 4 :** impose les dimensions souhaitée pour la hauteur et la largeur des noeuds du graphe.
- **Lignes** 5 **à** 13 **:** liste des arêtes du graphe. Par exemple $0 - -1$ indique qu'il existe une arête entre les sommets 0 et 1. Pour économiser de la place, il est possible de donner des suites d'arêtes : $0 - -1 - -2$ indique qu'il existe une arête entre les sommets 0 et 1 et une arête entre les sommets 1 et 2.

La bibliothèque GraphViz

Pour compiler un fichier .dot avec chaque programme et produire une image :

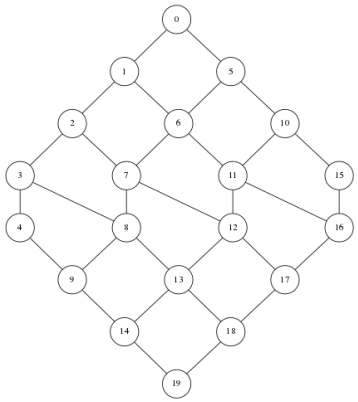
```
1          dot -Tpng grille.dot -o grille1.png
2          neato -Tpng grille.dot -o grille2.png
3          fdp -Tpng grille.dot -o grille3.png
4          sfdp -Tpng grille.dot -o grille4.png
5          twopi -Tpng grille.dot -o grille5.png
6          circo -Tpng grille.dot -o grille6.png
```

N'oubliez jamais :

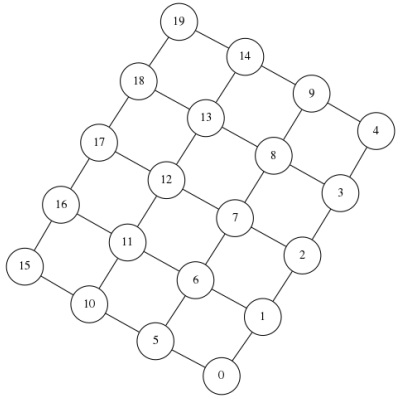
1. Lisez la documentation en ligne.
2. Lisez la page man (exemple : man dot).

La bibliothèque GraphViz

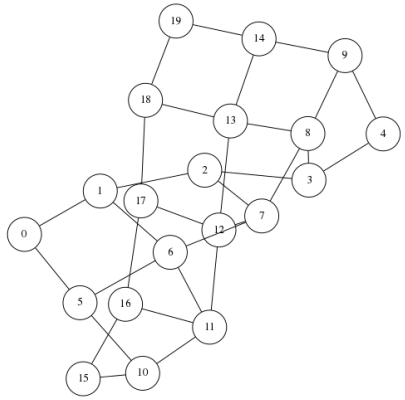
Voici le résultat de ces différents programmes sur notre exemple :



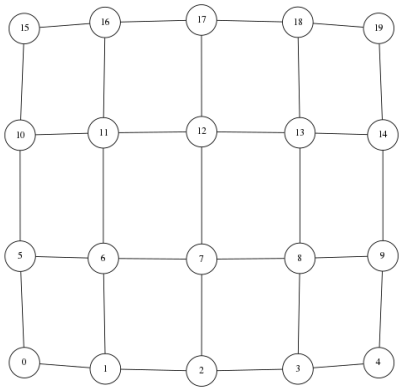
dot



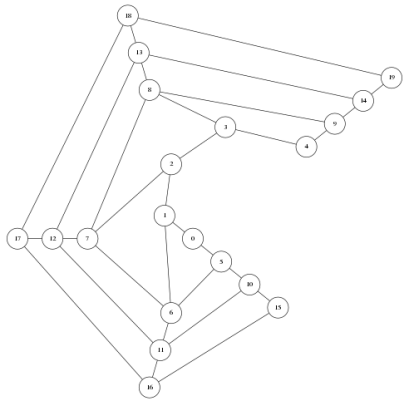
neato



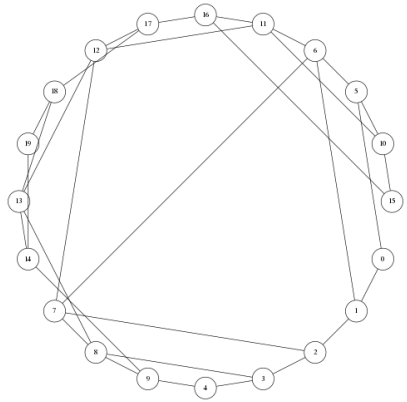
fdp



sfdp



twopi



circo

Idées de projets en théorie des graphes

1. Tweetomatic
2. Graphe de réseau social (si légal...)
3. Coloration de graphes
4. Couplage dans le cadre d'un site de rencontre
5. Génération du plan du RER, du métro ...
6. promenade passant par tous les sommets d'un graphe, application à des données en open data.
7. Trouver une solution au jeu de l'âne rouge.
8. Génération automatique de déroulement d'algorithme.