Université de Paris

Algorithmie Avancée

Mise en Contexte / Mise en Oeuvre

Année 2024-2025 par Prof. Nicolas Loménie
Sur la base du cours de Prof. Etienne Birmelé (2016-2020)

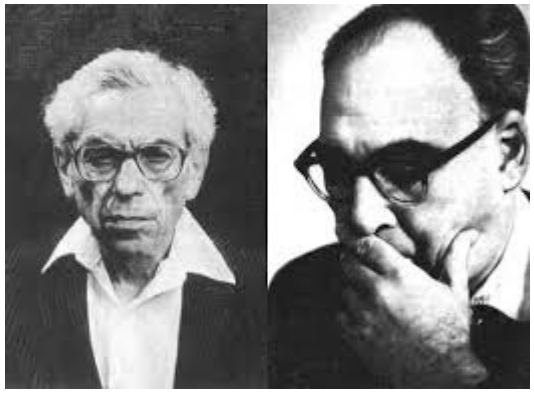
De Königsberg à Barasàbi

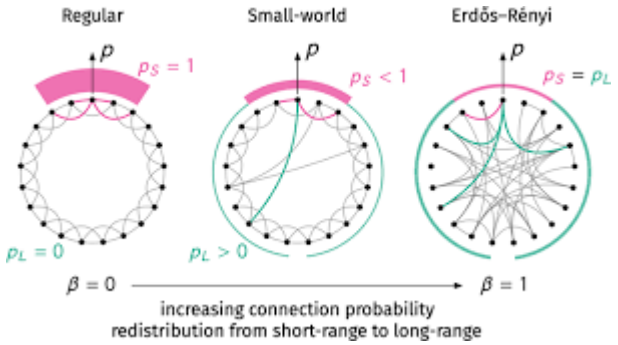
1735 : en Prusse Orientale, Léonard Euler

1959 – 1968 : Erdös et Rényi, Graphes aléatoires (Random trees/graphs)

« Un mathématicien est une machine à transformer le café en théorèmes ».

1999 : Barabàsi-Albert, Science des Réseaux (Scale-Free Networks)





Generalization of the small-world effect on a model approaching the Erdős–Rényi random graph by Benjamin F. Maier in https://www.nature.com/articles/s41598-019-45576-3 (Feb 2020)

De Königsberg à Barasàbi

Science

May 5, 2008

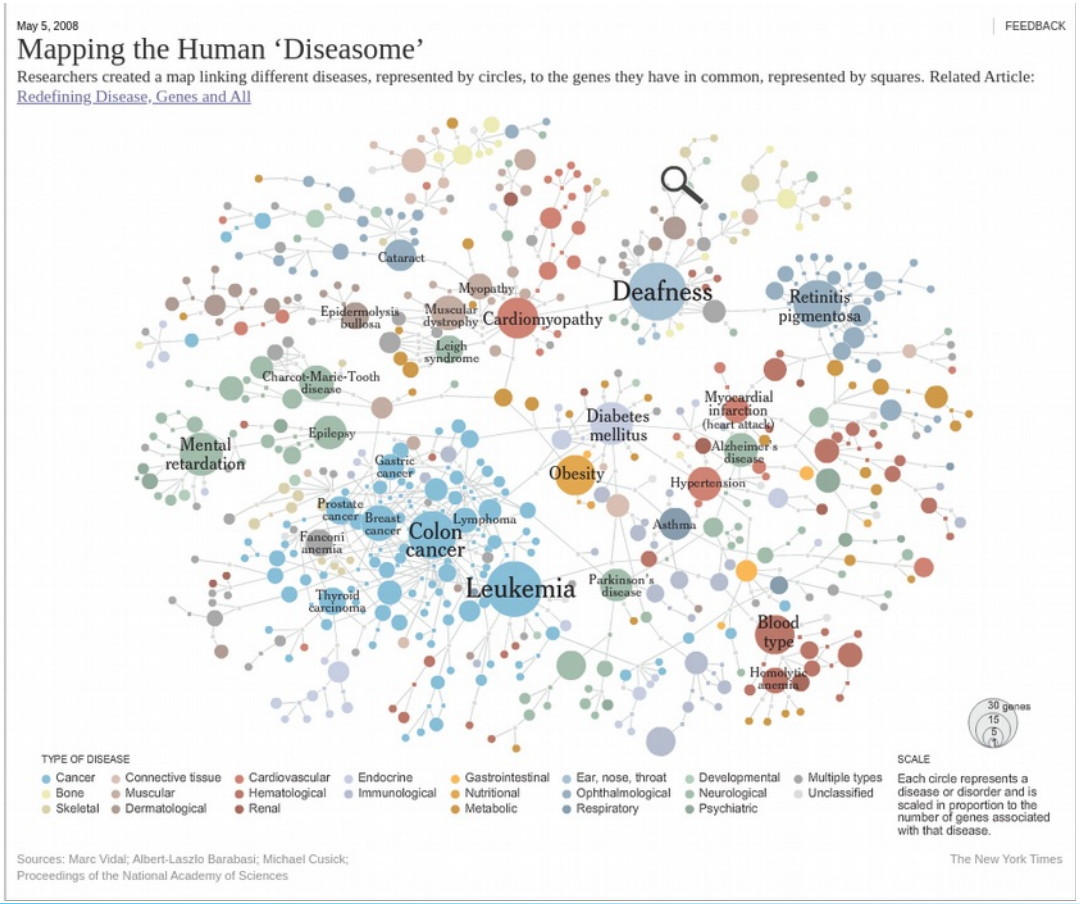
Mapping the Human ‘Diseasome’

Researchers created a map linking different diseases,

represented by circles, to the genes they have in common,

represented by squares. Related Article: Redefining Disease, Genes and All

Sources: Marc Vidal; Albert-Laszlo Barabasi; Michael Cusick



https://archive.nytimes.com/www.nytimes.com/interactive/2008/05/05/science/20080506_DISEASE.html?ref=health

Théorie des Graphes 3

AlgoAvanceePart1.pdf

Planche 33 à 63 (BFS DFS)

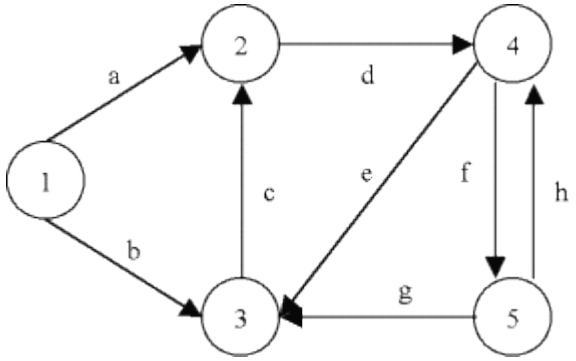
Structures de représentation

Exemple

Tableau dynamique d'arêtes

Ci-dessous, un extrait du tableau des arêtes :

	x	y	poids
1	1	2	5
2	1	3	1
3	2	4	2
4	3	2	4
5	4	3	1
6	4	5	7
7	5	3	9
8	5	4	1



Avantages :

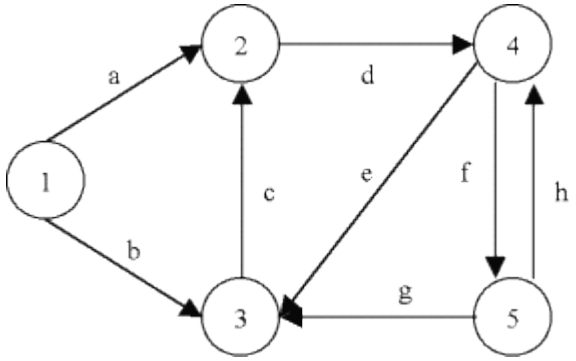
Inconvénients :

Structures de représentation

Exemple

La matrice d'adjacences

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	1	1	0

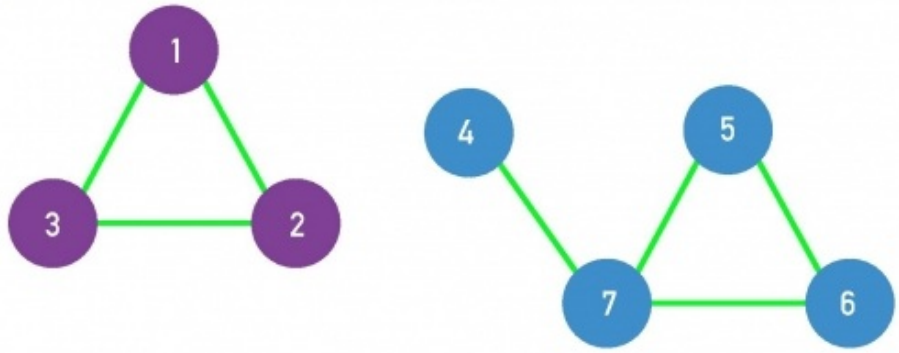


Avantages :

Inconvénients :

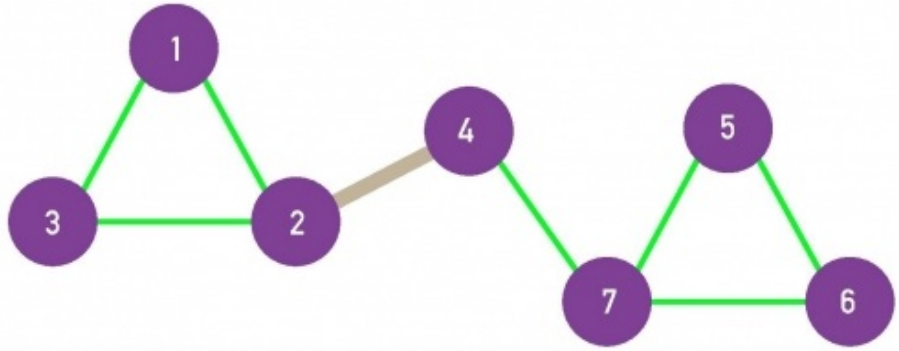
Matrice d’adjacence

a.



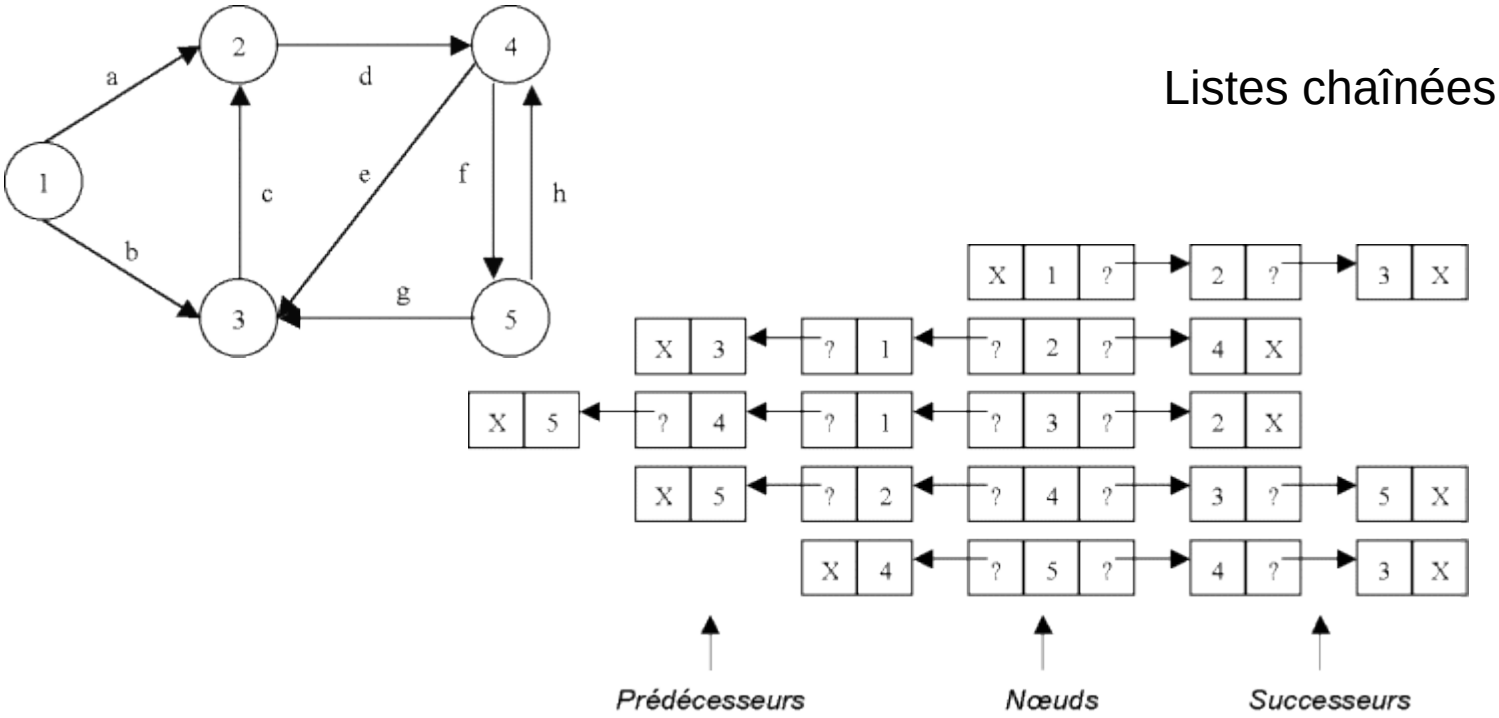
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

b.



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Structures de représentation



Listes chaînées et files d'attente

Avantages :

Inconvénients :

FIFO/LIF0

Exercice :

Nous voulons travailler sur une forme polygonale qui est composée d'une suite de points, repérés dans l'espace grâce à leurs coordonnées x et y (de type entier). Cette forme polygonale sera représentée par une liste doublement chaînée.

1) Ecrire les structures de données adaptées au problème
2) Ecrire les fonctions suivantes

cellule *NouvCel (point p) : qui alloue l'espace mémoire pour une cellule, remplit les champs de la structure cellule et retourne un pointeur sur cette cellule

void InsererCellule(int pl, cellule *cel, cellule *liste) : qui permet d'insérer la cellule *cel* après la place *pl* dans la liste *liste*

void SupprimeCellule(int pl , cellule *liste) : qui permet de supprimer la cellule à la position pl dans la liste

void Afficher(cellule *liste) : qui affiche la liste *liste*

3) Ecrire la fonction main qui permettra de créer une liste d'insérer, de supprimer des cellules, et enfin d'afficher la liste de points représentant la forme polygonale.

Ecrire la fonction main qui permettra de créer une liste d'insérer, de supprimer des cellules, et enfin d'afficher la liste de points représentant la forme polygonale.

FIFO/LIFO

```c
/*****************************************************************/
/*  Forme polygonale représentée par une liste doublement chaînée      */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>

typedef  struct point{
    int    x,y;
}point;
typedef struct cellule{
  point    pt;
  struct cellule  *suiv, *pred;
}cellule;

/*---------------------------------
    création d'un nouvelle cellule
---------------------------------*/
cellule *nouvCel(point  p){
    cellule  *cel;

    cel = (cellule *) malloc(sizeof(cellule)); /* alloc mémoire pour
cellule créée */

    if(cel == NULL) { /* si pas assez de memoire on sort de la fonction */
        printf("pas assez de memoire\n");
        return(NULL);
    }

     /* initialisation des champs de la cellule créée */
     cel->suiv = NULL;
     cel->pred = NULL;
     cel->pt = p;
     return(cel);
  }
/*-------------------------------------------------------------------
```

FIFO/LIFO

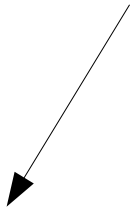
pop()

```
/*-------------------------------------------------------------------
  Enlève la cellule à la position pl dans la liste liste
-------------------------------------------------------------------*/
void supprimeCellule(int  pl, cellule *liste){
   cellule *celcour;
   int        q=0;
   celcour = liste;
   while(pl!= q && celcour!=NULL){
       celcour = celcour->suiv;
       q++;
   }
   if(celcour == NULL)
      printf("la position specifiee n'existe pas\n");
   else{ /*on est sur la bonne position, on supprime la cellule à cette place*/
       celcour->pred->suiv = celcour->suiv;
       if (celcour->suiv != NULL)
          celcour->suiv->pred = celcour->pred;
       free(celcour);
    }
}
/*-----------------------------------
   Affiche une liste
-----------------------------------*/
void afficherListe(cellule *liste){
  cellule  *celcour;
  int         i = 1;

  for(celcour = liste->suiv; celcour != NULL; celcour= celcour->suiv){
    printf("contenu de la cellule %d: \t x: %d\t  y:%d\n", i, celcour->pt.x, celcour->pt.y);
      i++;
  }
}
```

FIFO/LIFO

```
/*-------------------------------------------------------------------------
Ajoute la  cellule cel  dans la  liste chainee liste apres la place pl
-------------------------------------------------------------------------*/
 void insererCellule(int pl,  cellule *cel,cellule *liste){
```



Push()

Structures de représentation

Modélisation
Objet : POO

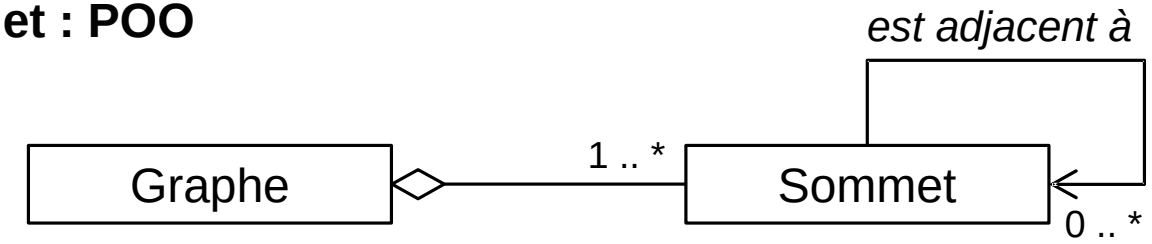


Diagramme de classes 1 **(⇨ modélisation par liste de successeurs)**

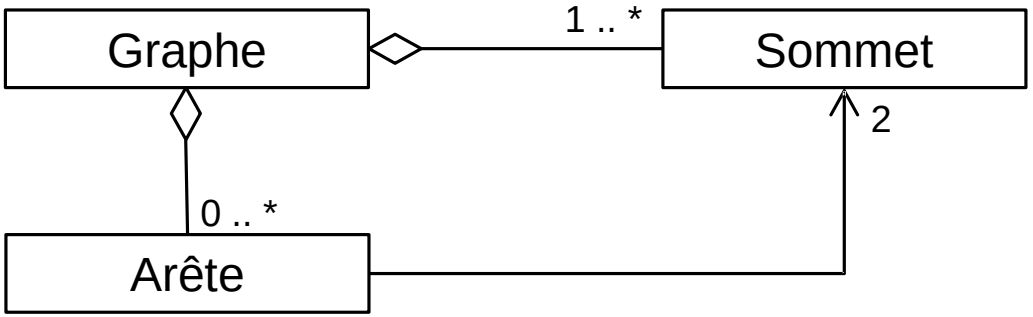
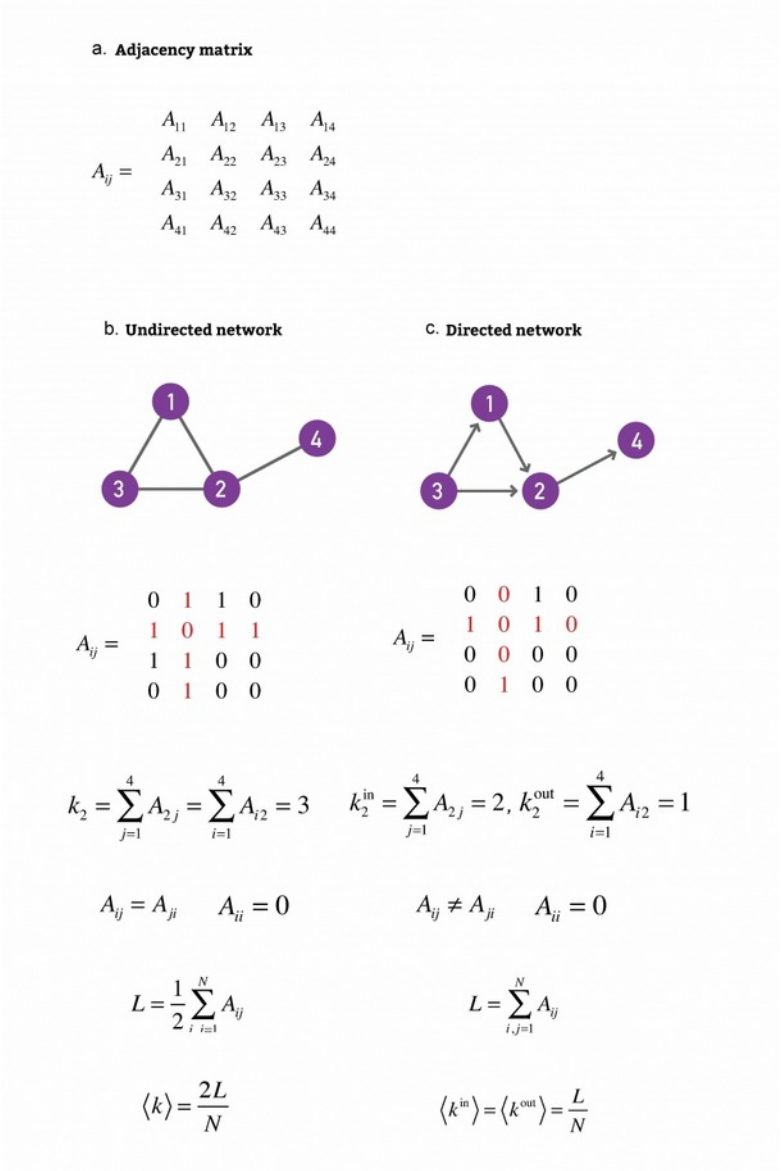


Diagramme de classes 2 **(⇨ modélisation par liste d'arêtes)**

Matrice d’adjacence



The Adjacency Matrix
(a) The labeling of the elements of the adjacency matrix.
(b) The adjacency matrix of an undirected network. The figure shows that the degree of a node (in this case node 2) can be expressed as the sum over the appropriate column or the row of the adjacency matrix. It also shows a few basic network characteristics, like the total number of links, L, and average degree, ‹k›, expressed in terms of the elements of the adjacency matrix.
(c)The same as in (b) but for a directed network.

The real networks are sparse :-)

Compléxité et Structure de données

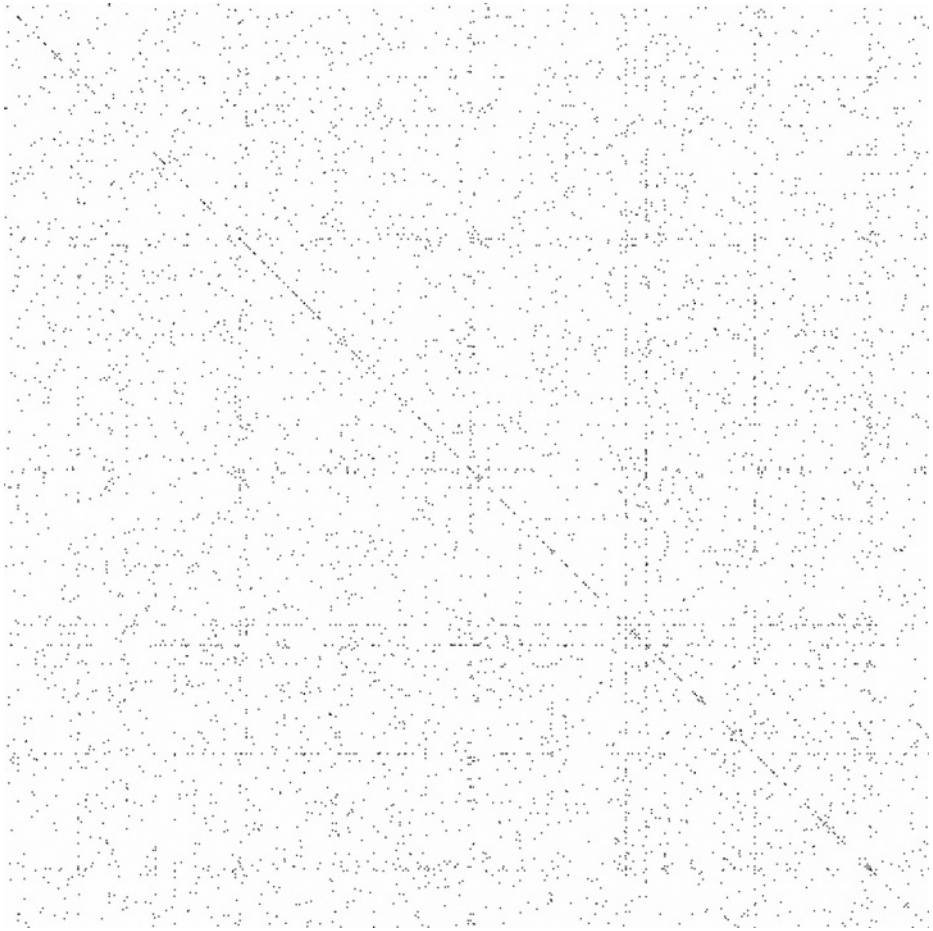
Opérations	Liste d'adjacence	Matrice d'adjacence
Retirer une arête	$O(d)$ avec d le degré du nœud	$O(1)$
Ajouter une arête	$O(1)$	$O(1)$
Itérer sur les voisins d'un nœud	$O(d)$ avec d le degré du nœud	$O(N)$
Tester si deux nœuds sont voisins	$O(d)$ avec d le degré du nœud	$O(1)$
Complexité mémoire	$O(N + A)$	$O(N^2)$

	Liste d'ajacence	Matrice d'adjacence	Matrice d'incidence
Créer le graphe	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Ajouter un sommet	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Ajouter une arête	$O(1)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Supprimer un sommet	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Supprimer une arête	$O(\|V\|)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Test d'adjacence entre deux sommets	$O(\|V\|)$	$O(1)$	$O(\|E\|)$
Remarques	Lent dans la suppression parce qu'il faut trouver les sommets ou arêtes	Lent dans l'adjonction ou suppression de sommets parce que la matrice doit être reformatée	Lent dans l'adjonction ou suppression de sommets ou d'arêtes parce que la matrice doit être reformatée

https://fr.wikipedia.org/wiki/Graphe_(type_abstrait)

https://zestedesavoir.com/tutoriels/681/a-la-decouverte-des-algorithmes-de-graphe/727_bases-de-la-theorie-des-graphes/3352_graphes-et-representation-de-graphe/

Compléxité et Structure de données



The Adjacency Matrix is Sparse
The adjacency matrix of the yeast protein-protein interaction network, consisting of 2,018 nodes, each representing a yeast protein. A dot is placed on each position of the adjacent matrix for which $A_{ij} = 1$, indicating the presence of an interaction. There are no dots for $A_{ij} = 0$. The small fraction of dots illustrates the sparse nature of the protein-protein interaction network.

http://networksciencebook.com/chapter/2#real-networks