

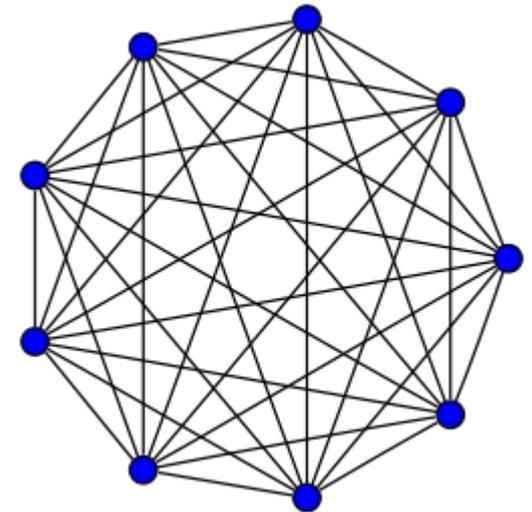
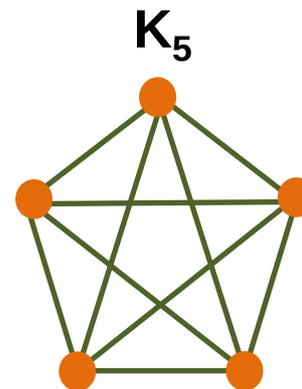
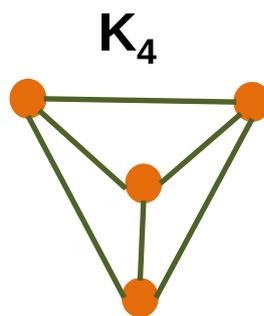
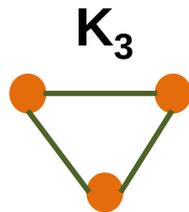
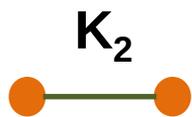
Algorithmie Avancée

Mise en Contexte / Mise en Oeuvre

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

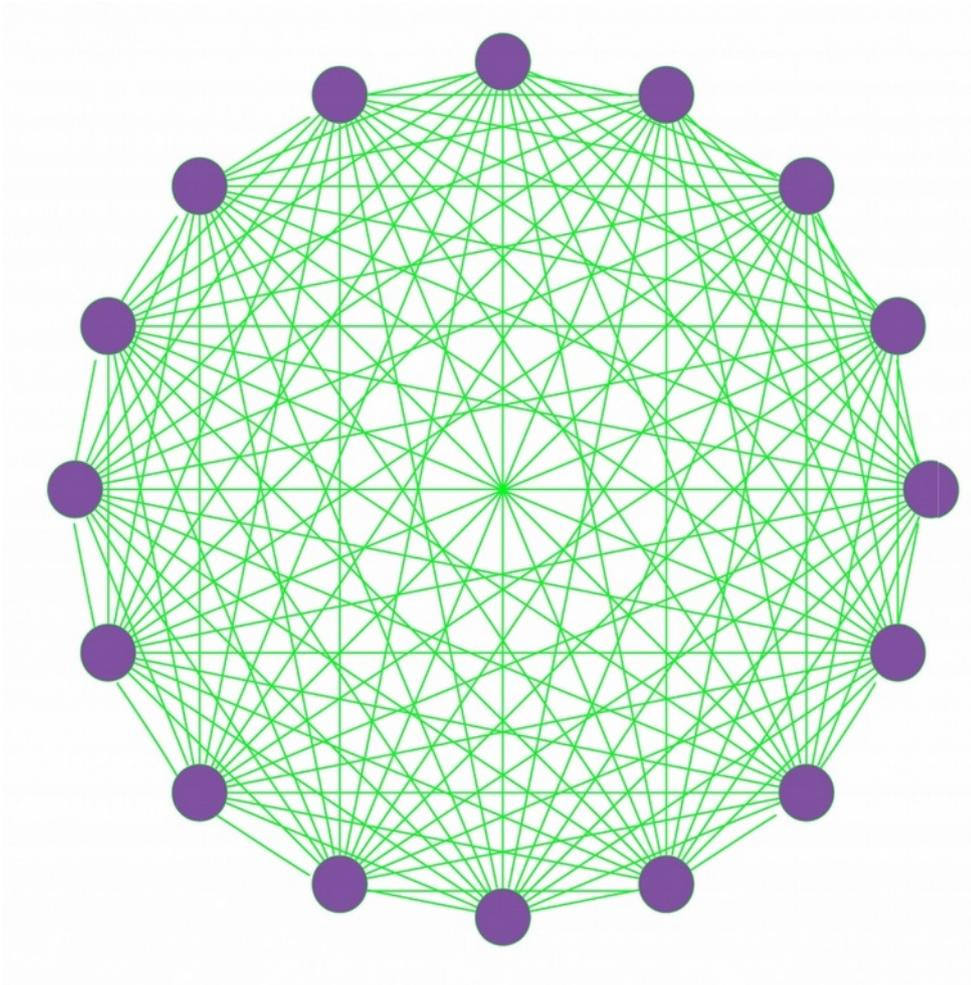
Un bestiaire de graphes

On note K_n le graphe complet d'ordre n .



K_n possède $\frac{n(n-1)}{2}$ arêtes

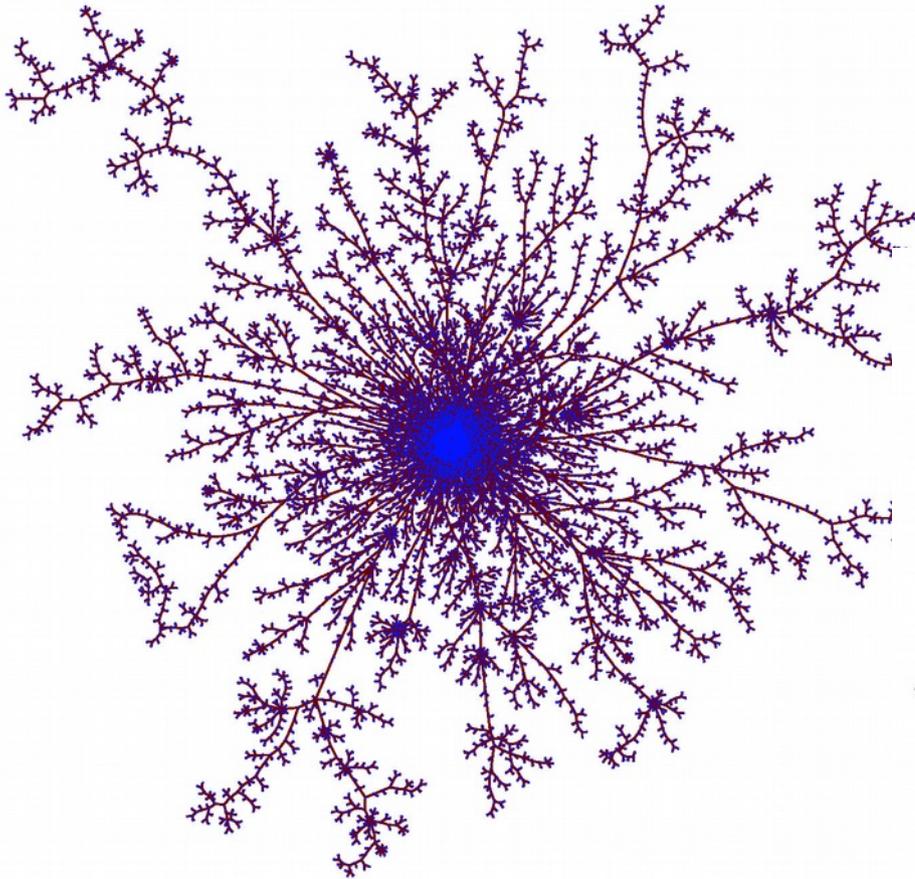
Un bestiaire de graphes



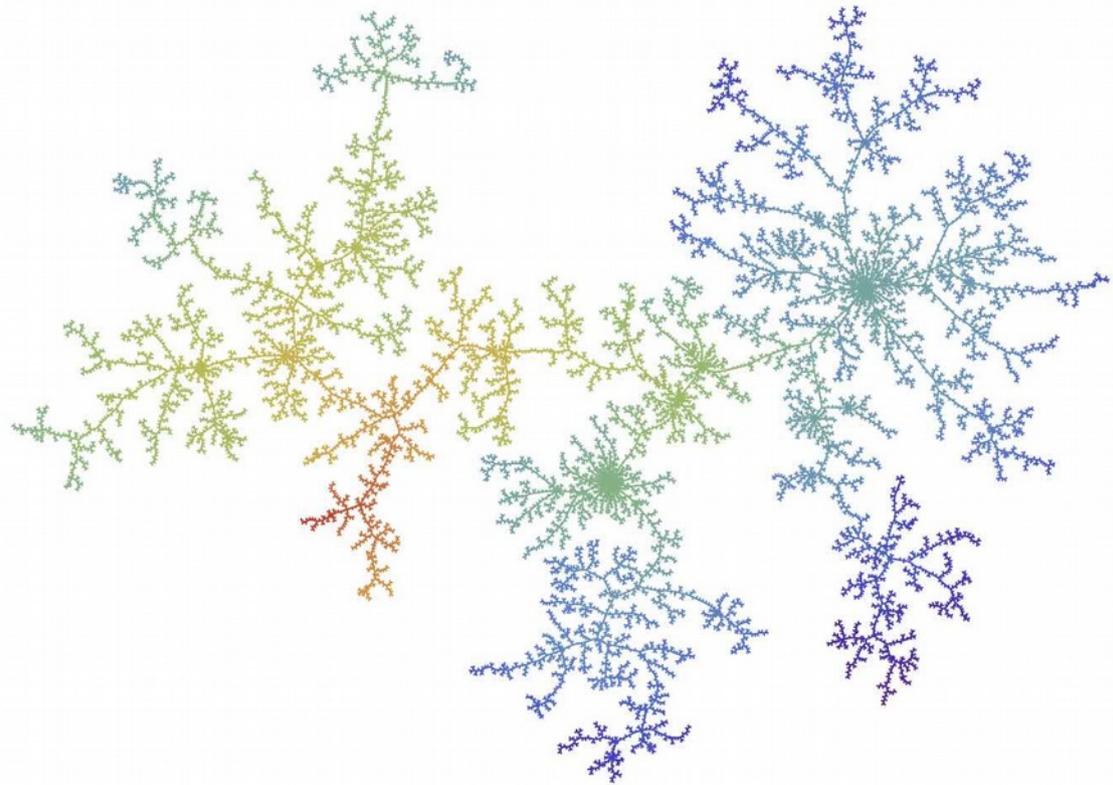
Complete Graph

A complete graph with $N = 16$ nodes and $L_{\max} = 120$ links. The adjacency matrix of a complete graph is $A_{ij} = 1$ for all $i, j = 1, \dots, N$ and $A_{ii} = 0$. The average degree of a complete graph is $\langle k \rangle = N - 1$. A complete graph is often called a clique, a term frequently used in community identification

Un bestiaire de graphes



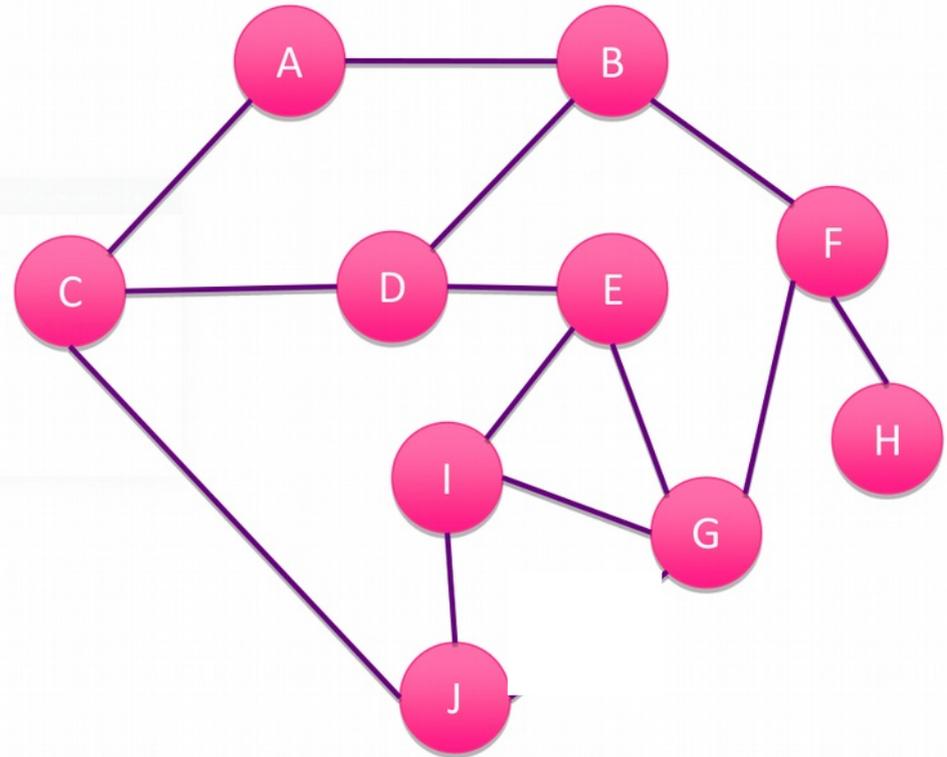
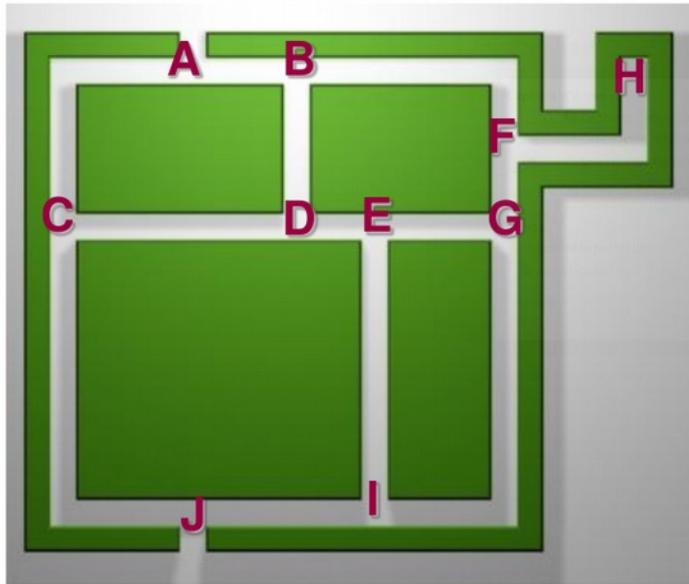
Arbres aléatoires de
Bienaymé--Galton-Watson



<https://mathinfo.alwaysdata.net/2016/12/7-arbres-fractals/>

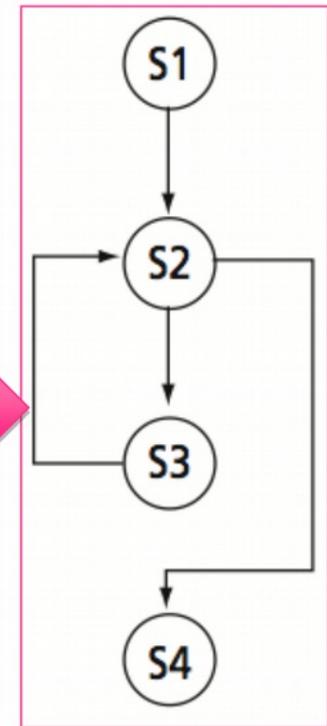
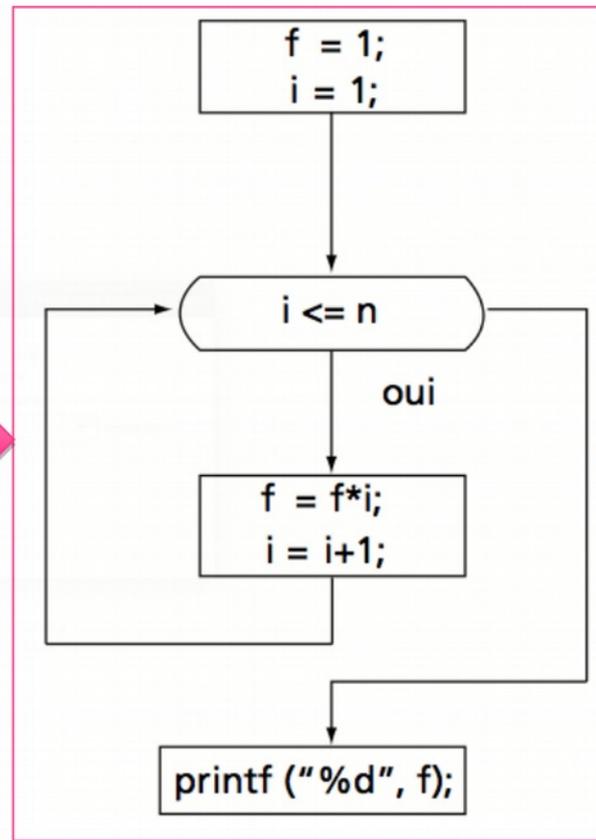
<https://igor-kortchemski.perso.math.cnrs.fr/>

Un bestiaire de graphes



Un bestiaire de graphes

```
int f, i;  
f = 1;  
for (i=1; i<=n; i++) {  
    f = f * i;  
}  
printf ("%d", f);
```



Un bestiaire de graphes → réseaux



Graphes et chemins

Les premières questions qui se posent naturellement

Comment parcourir un graphe ?

- parcourir ses sommets, parcourir ses arêtes ?
- dans quel ordre ? → BFS, DFS
- peut-on passer une et une seule fois par chaque sommet ? par chaque arête ? → graphe eulérien, hamiltonien
- comment éviter de « tourner en rond » ? → cycles

Comment aller d'un sommet à un autre ?

- est-ce toujours possible ? → connexité
- comment trouver le chemin le plus court ? → recherche PCC

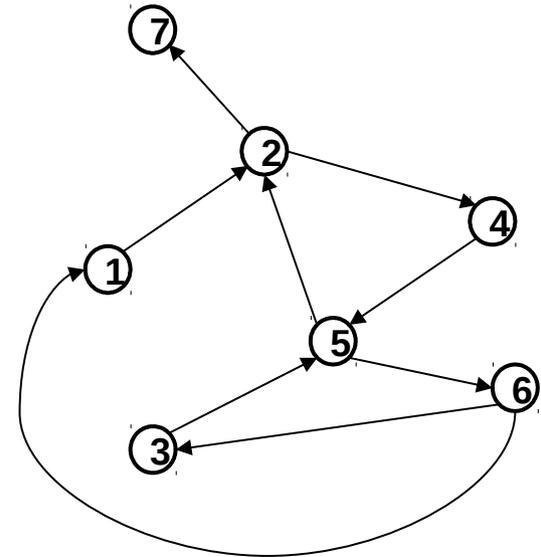
Graphes et chemins

Chemin simple

- x Un chemin est dit **simple** s'il ne comporte pas plusieurs fois le même arc.
- x $((1,2),(2,4),(4,5),(5,2),(2,7))$ est simple
- x $((1,2),(2,4),(4,5),(5,6),(6,1),(1,2),(2,7))$ n'est pas simple

Chemin élémentaire

- x Un chemin est dit **élémentaire** s'il ne passe pas plusieurs fois par le même sommet.
- x $((1,2),(2,4),(4,5),(5,6))$ est élémentaire
- x $((1,2),(2,4),(4,5),(5,2),(2,7))$ n'est pas élémentaire



Théorie des Graphes 2

- [AlgoAvanceeCoursPart1.pdf](#)

Support de cours

Planche 15 à 32 (chemin, cycle, connexité, arbre, arbre couvrant)

Une structure de données simple

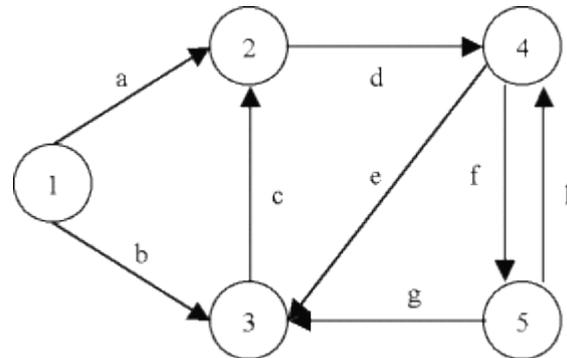
Dans un TP courant on vous propose d'implémenter cette fonction prototypée :

```
void marquerVoisins (int** adjacence, int ordre, int s) ;
```

Dans laquelle un graphe est représenté par sa matrice d'adjacence.

Exemple

	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



Le **malloc** associé :
L'adressage linéarisé correspondant :

```
adjacence = malloc(sizeof(int *) * ordre);  
for(int i=0; i<ordre; ++i) {  
    adjacence[i]=malloc(sizeof(int)*ordre);  
}
```

```
int *adj =(int*)malloc(sizeof(int) * ( ordre * ordre));  
...  
adj[ u*ordre + v ] = value;
```

Eventuellement POOisée

```
#include<stdio.h>
#include<stdlib.h>
struct Graph{
    int V;
    int E;
    int **adj;
};
struct Graph* adjmatrix(){
    int u,v,i;
    struct Graph* G=(struct Graph*)malloc(sizeof(struct
Graph));
    if(!G)
        printf("Memory Null");
    printf("enter the number of vertex and edges");
    scanf("%d %d",&G->V,&G->E);
    //Allocation de la mémoire : voir slide précédent
    ...
    ///
    for(u=0;u<G->V;u++){
        for(v=0;v<G->V;v++){
            G->adj[u][v]=0;
        }
    }
}
```

```
for(i=0;i<G->V;i++){
    printf("reading edge");
    scanf("%d %d",&u,&v);
    G->adj[u][v]=1;
    G->adj[v][u]=1;
}
return G;
}
```