Les graphes

HESTIM ENGINEERING & BUSINESS SCHOOL

Plan

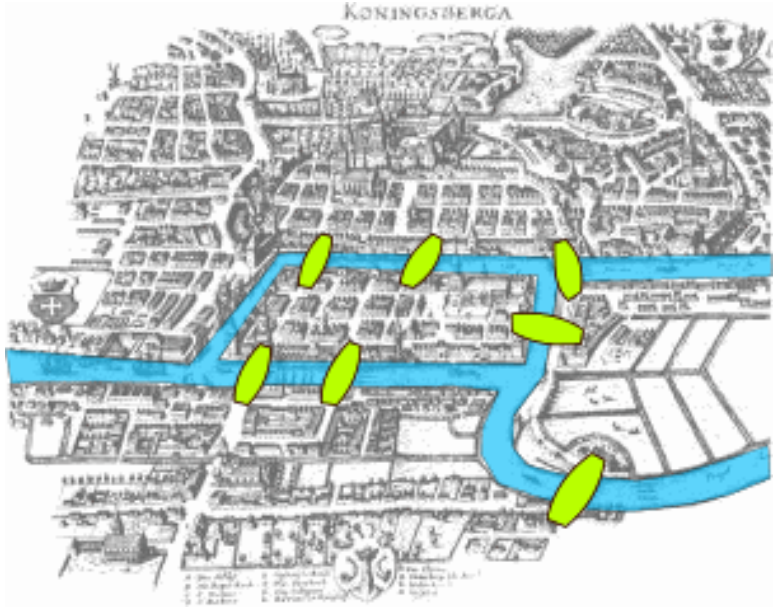
- Définitions et terminologies
- Implémentation des graphes
- Algorithmes de parcours d'un graphe
- Applications
  - Plus courts chemin(Algorithme Dijkstra, Bellman-Ford, Floyd-Warsha
  - Arbre couvrant (Kruskal, Prim)
  - Détection de cycles et connexité
  - Coloration de graphes
  - Algorithme A*

Objectifs

- Comprendre les notions de base des graphes
- Savoir modéliser certains problèmes afin de les résoudre à l'aide des techniques et des algorithmes simples.
- Maîtriser les algorithmes de recherche des plus courts chemins (Dijkstra et A*), des composantes connexes (Tarjan), des arbres couvrants minimaux (Kruskal et Prim), de coloration des sommets (Welch-Powell et Dsatur), etc.

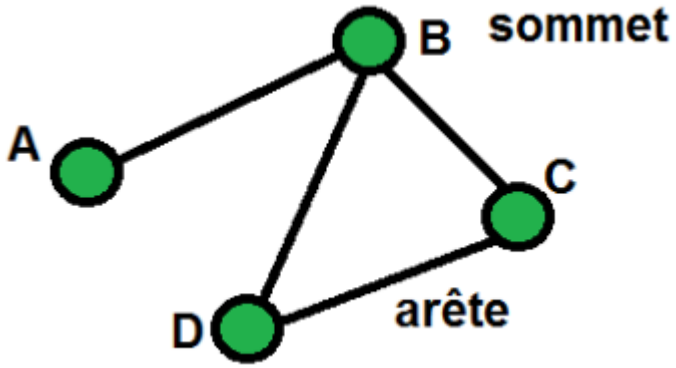
Origine

La théorie des graphes est née en 1736 quand Euler a posé et résolu le problème des sept ponts de la ville russe de Konigsberg (aujourd'hui Kaliningrad) :
Peut-on, partant d'un point quelconque (rive ou ile) parcourir les 7 ponts une fois et une seule et revenir à son point de départ ?



Définitions et terminologies

Un graphe est la donnée d'un certain nombre de points du plan, appelés ***sommets***, certains étant reliés par des segments de droites ou de courbes appelés ***arêtes***, la disposition des sommets et la forme choisie pour les arêtes n'intervenant pas.



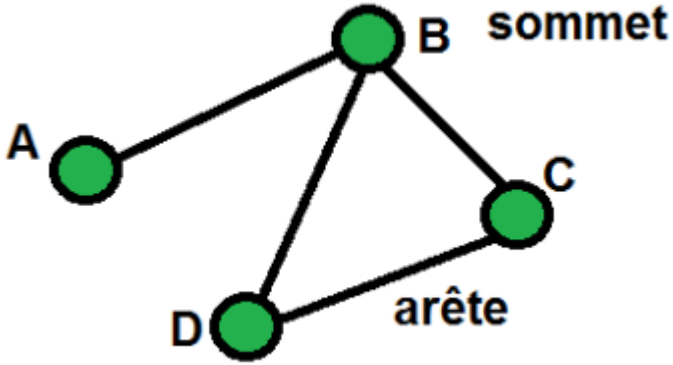
Définitions et terminologies

Un graphe non orienté G = (S, A) est déterminé par deux ensembles :

- S un ensemble fini des sommets,
- A un ensemble de couples non ordonnés de sommets $(a_i, a_j) \in S^2$.

Exemple :

G = ({A,B,C,D} , { (A,B),(B,C),(C,D),(B,D) })



Définitions et terminologies

Exemples des situations modélisées par un graphe :

- Carte géographique : Recherche du chemin le plus court entre deux villes.
- Les lignes aériennes
- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.
- Le routage des réseaux informatiques
- Un réseau social : les sommets sont les personnes, deux personnes sont adjacentes dans ce graphe lorsqu'elles sont "amies". Si la notion d'amitié n'est pas réciproque, le graphe est orienté.

Exemple d'application 1

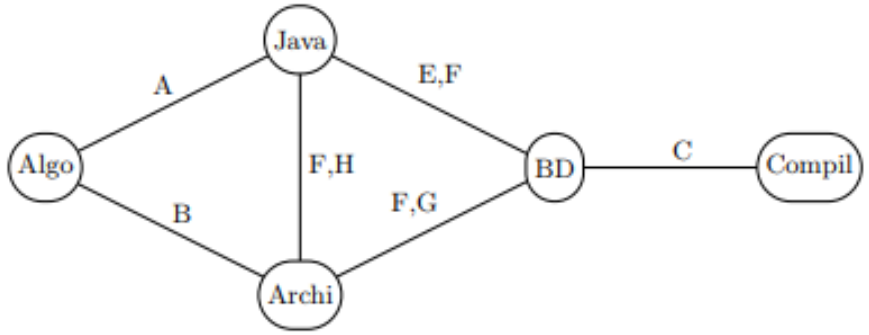
Des étudiants A, B, C, D, E et F doivent passer des examens dans différentes disciplines, chaque examen occupant une demi-journée :

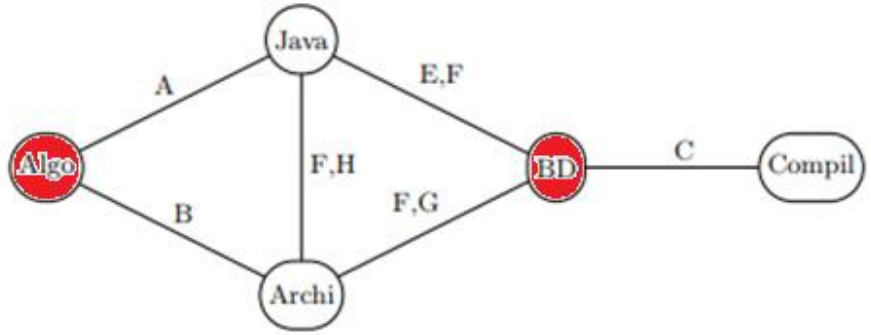
– Algorithmique : étudiants A et B.
– Compilation : étudiants C et D.
– Bases de données : étudiants C, E, F et G.
– Java : étudiants A, E, F et H.
– Architecture : étudiants B, F, G et H.

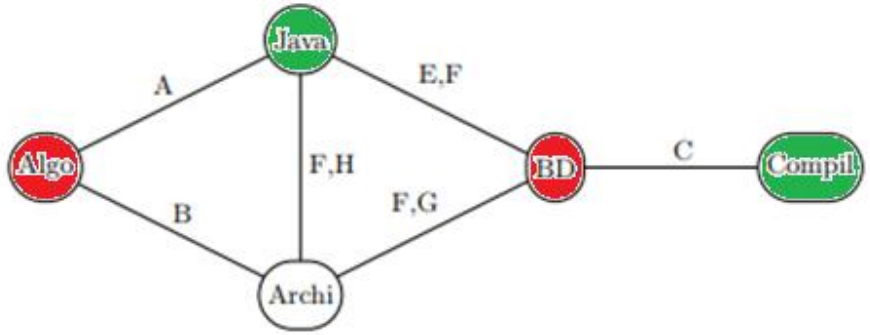
On cherche à organiser la session d'examen la plus courte possible.

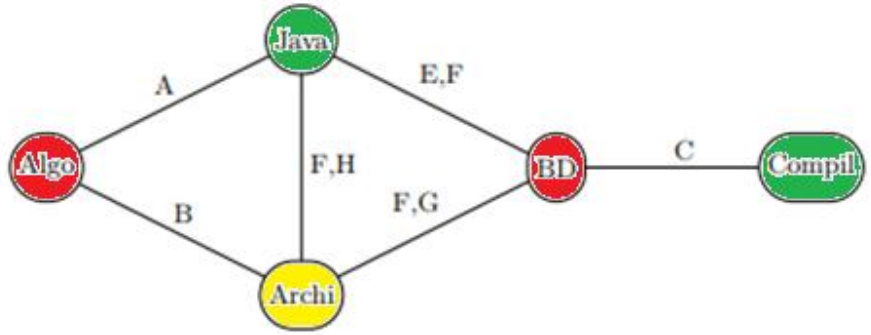
Définitions et terminologies

Exemple d'application 1









Exemple d'application 1

Session 1:
Algo : A,B
BD : C, E, F, G

Session 2:
Java : A, E, F, H
Comp : C

Session 3:
Arch : B, F, G, H

Exemple d'application 2

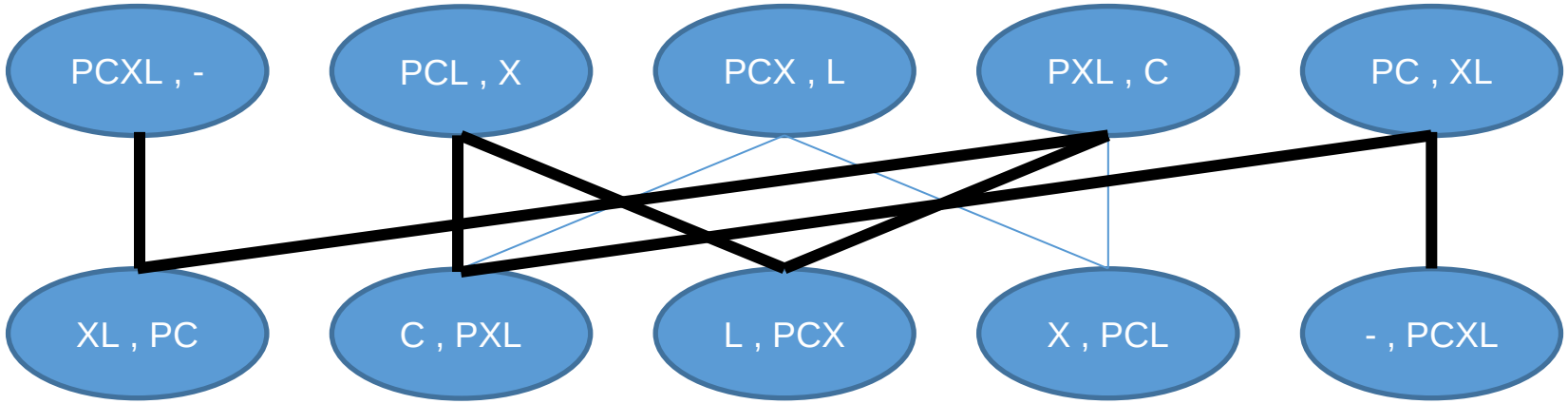
Une chèvre, un chou et un loup se trouvent sur la rive d'un fleuve ; un passeur souhaite les transporter sur l'autre rive mais, sa barque étant trop petite, il ne peut transporter qu'un seul d'entre eux à la fois.

Comment doit-il procéder afin de ne jamais laisser ensemble et sans surveillance le loup et la chèvre, ainsi que la chèvre et le chou ?

Exemple d'application 2

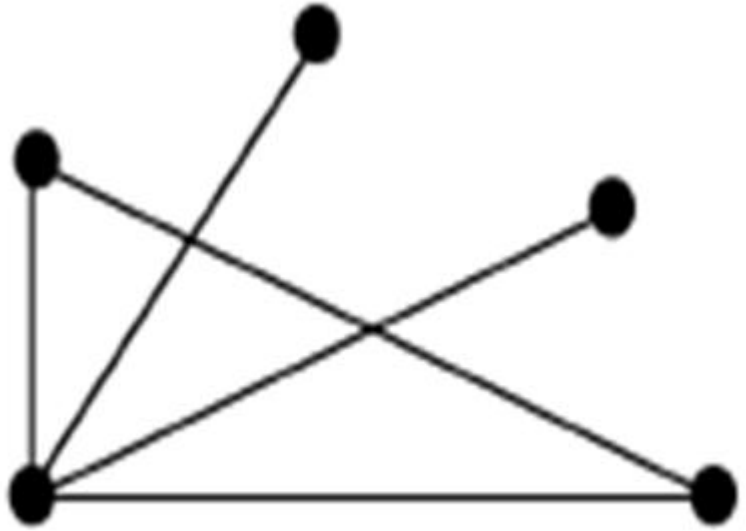
Cette situation peut être modélisée à l'aide d'un graphe.

Désignons par **P** le passeur, par **C** la chèvre, par **X** le chou et par **L** le loup. Les sommets du graphe sont des couples précisant qui est sur la rive initiale, qui est sur l'autre rive. Ainsi, le couple **(PCX,L)** signifie que le passeur est sur la rive initiale avec la chèvre et le chou (qui sont donc sous surveillance), alors que le loup est sur l'autre rive.

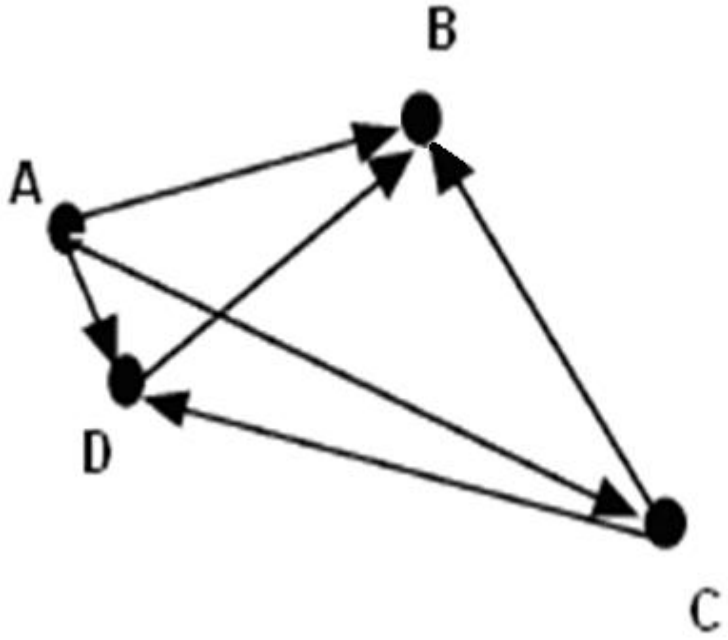


Définitions et terminologies

- **Graphe non orienté :**

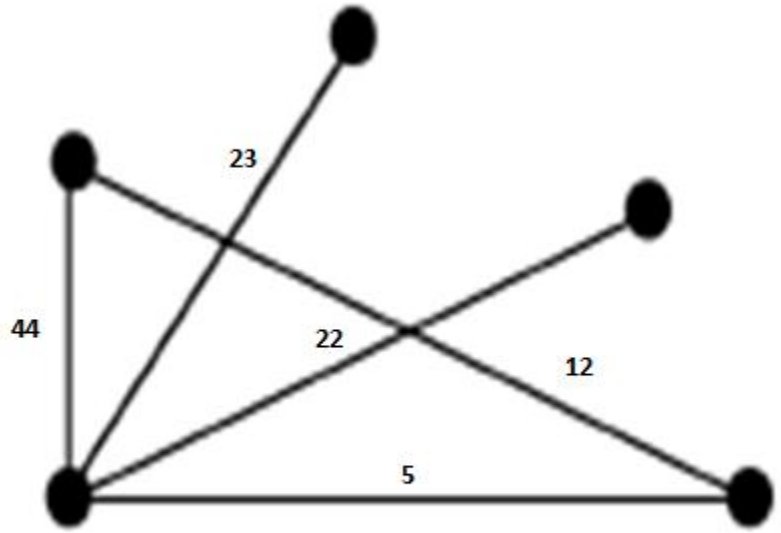


- **Graphe orienté :**

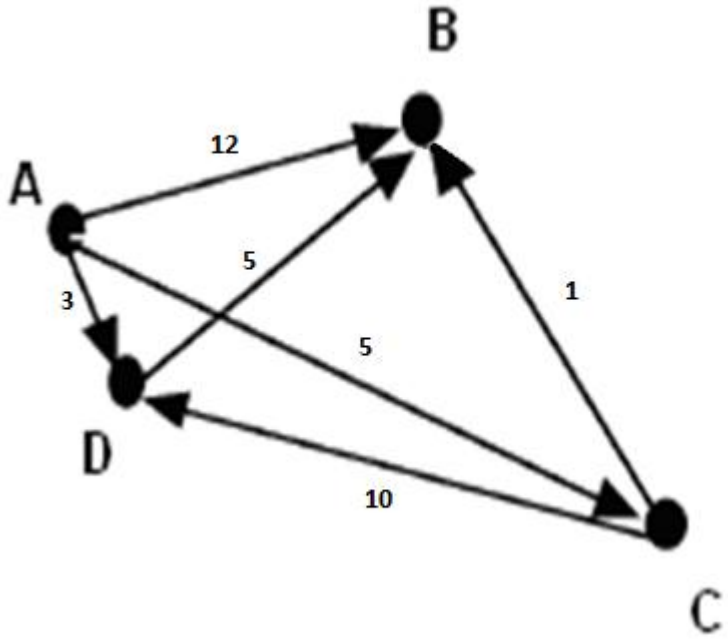


Définitions et terminologies

- **Graphe non orienté pondéré :**

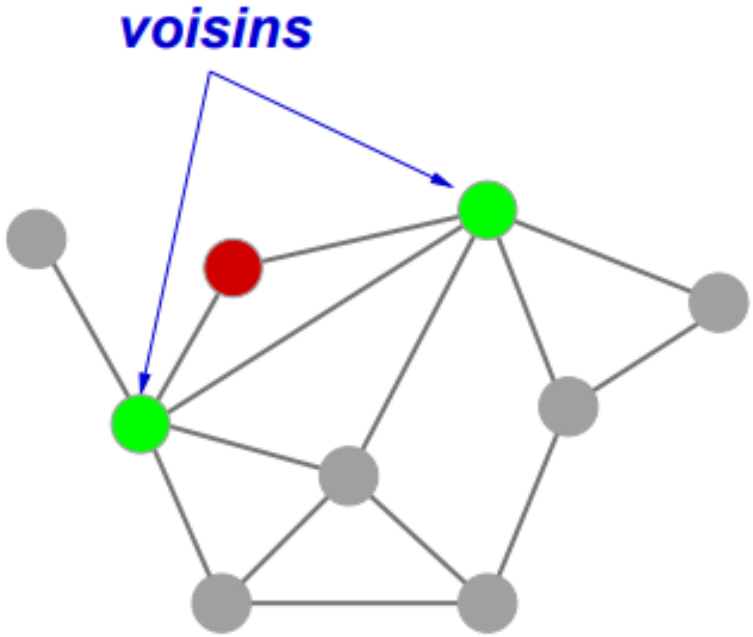


- **Graphe orienté pondéré:**



Définitions et terminologies

Deux sommets d'un graphe sont dits *adjacents* s'il existe une arête (ou un arc) qui les relie.



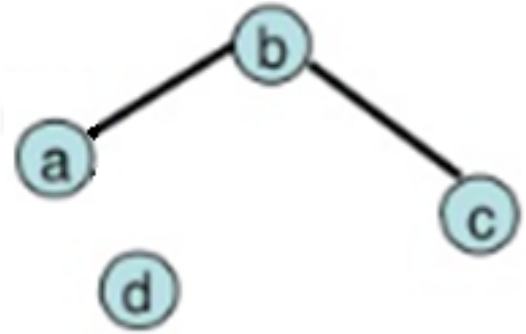
Définitions et terminologies

L'ordre d'un graphe G(S, E) est le nombre de ses sommets. n =|S|

La **taille** d'un graphe G(S, E) est le nombre de ses arêtes. m =|E|

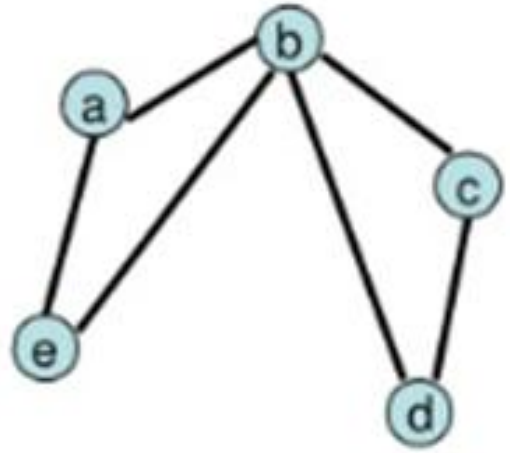
Graphe d'ordre 4

Graphe de taille 2



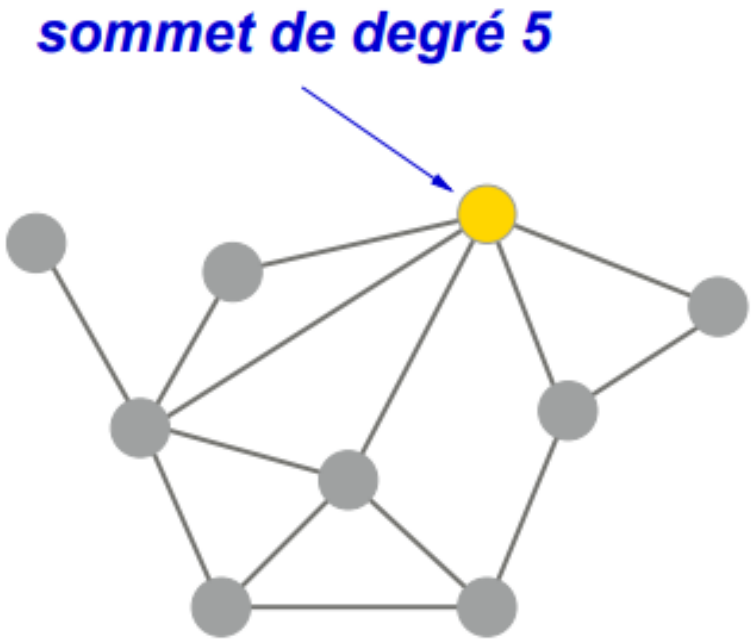
Graphe d'ordre 5

Graphe de taille 6



Définitions et terminologies

Degré d'un sommet : nombre d'arêtes reliées à ce sommet.



Définitions et terminologies

Propriété 1.

Dans un graphe non orienté G = (S, E), la somme des degrés des sommets d'un graphe est égale à deux fois le nombre d'arêtes :

$$\sum_{x \in S} d(x) = 2 \mid E \mid$$

Définitions et terminologies

Exercice 1:

Dans une réunion de 5 personnes, combien de poignées de main sont échangées ? On suppose que tout le monde salue tout le monde

Solution :

On peut modéliser la situation par un graphe non orienté G = (S, E) où les sommets représentent les personnes et les arêtes représentent les poignées de mains échangées.

Le problème revient à trouver le nombre des arêtes possibles (la taille du graphe|E|) sachant que :

- l'ordre du graphe égale à 5 (n = |S| = 5) et
- le degré de chaque sommet égale à 4

$$\sum_{x \in S} d(x) = 2 \mid E \mid \Leftrightarrow 5 * 4 = 2|E \mid \Leftrightarrow \quad E = 10$$

Définitions et terminologies

Exercice 2:

Est-il possible de relier 15 ordinateurs de sorte que chaque ordinateur soit relié exactement avec 3 autres?

Solution :

On peut modéliser la situation par un graphe non orienté G = (S, E) où les sommets

représentent les ordinateurs et les arêtes représentent les liens entre eux.

Le problème revient à trouver le nombre des arêtes possibles (la taille du graphe|E|) sachant que :

- l'ordre du graphe égale à 15 (n = |X| = 15) et
- le degré de chaque sommet égale à 3

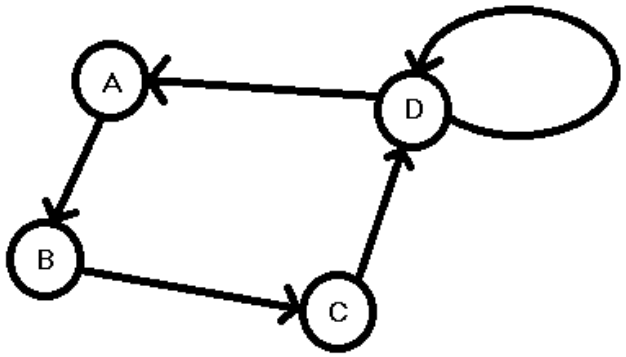
$$\sum_{x \in S} d(x) = 2 \mid E \mid \Leftrightarrow 15 * 3 = 2|E \mid \Leftrightarrow |E| = 22.5$$

Impossible

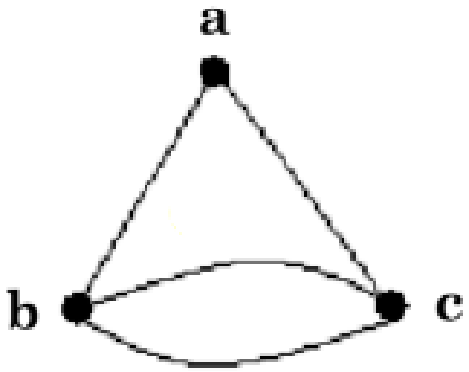
Définitions et terminologies

- *Une boucle $\{x_i, x_i\}$ est une arête reliant un sommet x_i à lui-même.*
- *Un graphe non-orienté est dit simple s'il ne comporte pas de boucles, et s'il ne comporte jamais plus d'une arête entre deux sommets distincts.*
- *Un graphe non orienté qui n'est pas simple est un multi-graphe.*

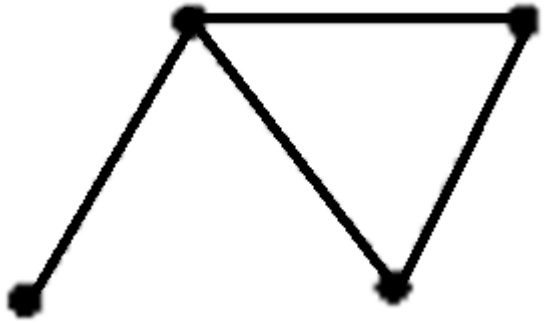
Exemples:



Multi-graphe



Multi-graphe



Graphe simple

Définitions et terminologies

Chaîne : Une chaine de longueur n est une suite de n arêtes qui relient un sommet i à un autre j ou à lui-même.

Exemples :

Des chaînes de longueur 2 :

A D C

A B C

Des chaînes de longueur 3 :

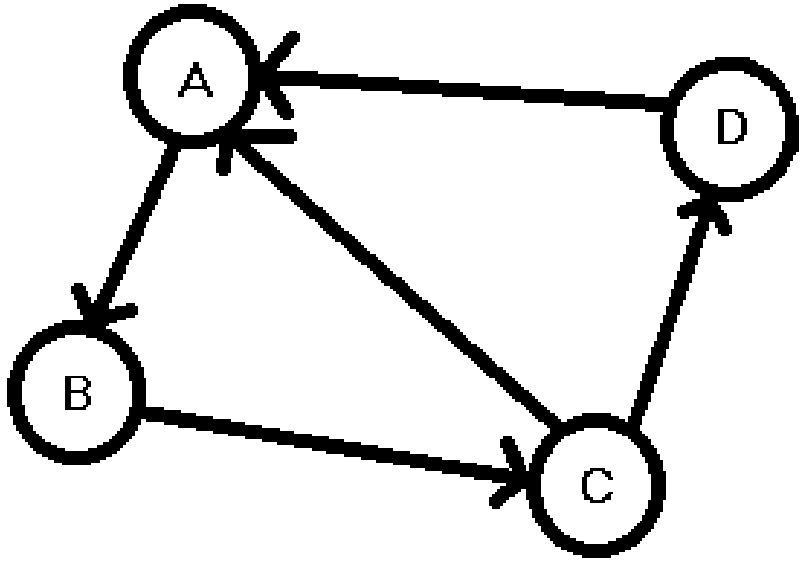
A B C D

B A D C

Des chaînes de longueur 4 :

A B C D A

A B C A D



Définitions et terminologies

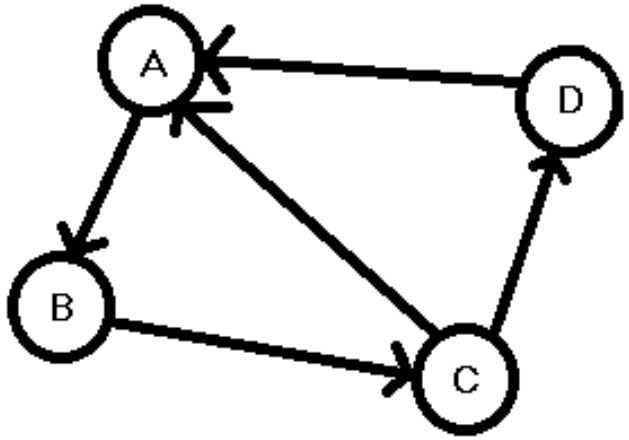
Cycle : Un cycle est défini comme une chaine fermée, tel que l'extrémité initiale est à la fois extrémité finale.

Exemples :

A B C D A est un cycle

A B C A est un cycle

D C A D est un cycle



Définitions et terminologies

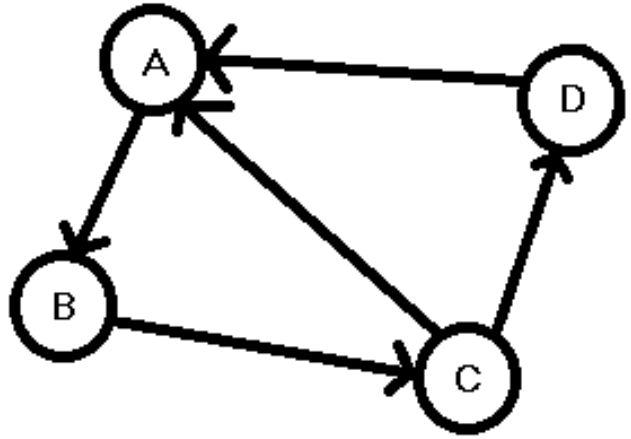
Chemin : c’est une chaine bien orientée

Exemples :

A B C D est un chemin

A D C A B n'est pas un chemin

A B C A D n'est pas un chemin



Définitions et terminologies

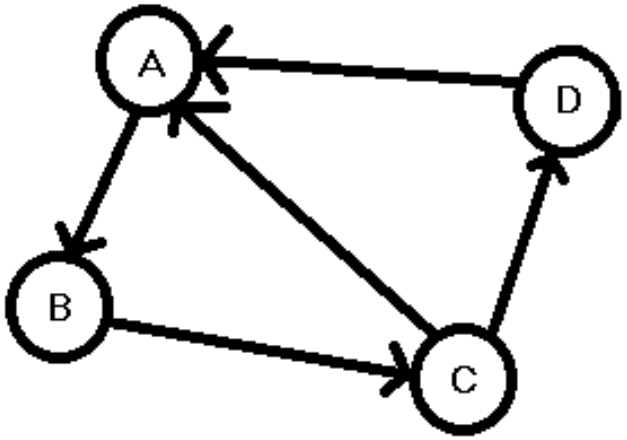
Circuit : est un cycle "bien orienté", à la fois cycle et chemin.

Exemples :

A B C A est un circuit

A B C D A es un circuit

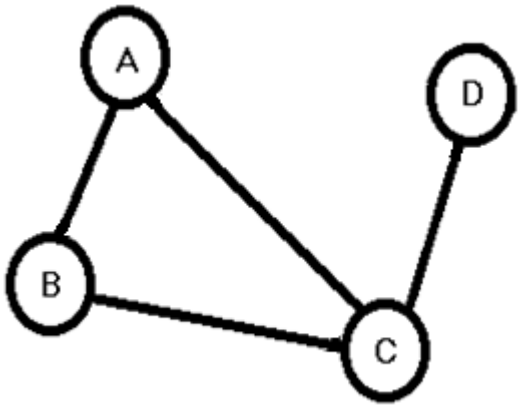
A D C A est un cycle mais pas un circuit



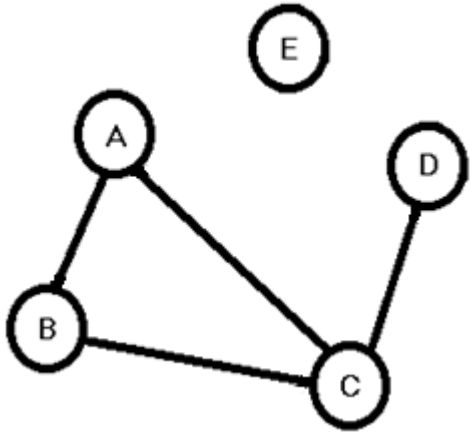
Définitions et terminologies

Graphe Connexe : Un graphe connexe est un graphe dont tout couple de sommets peut être relie par une chaine de longueur n>=1

- **Graphe connexe :**

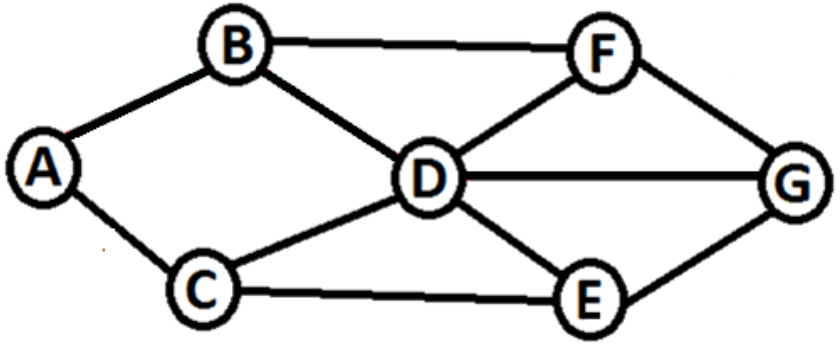


- **Graphe non connexe:**

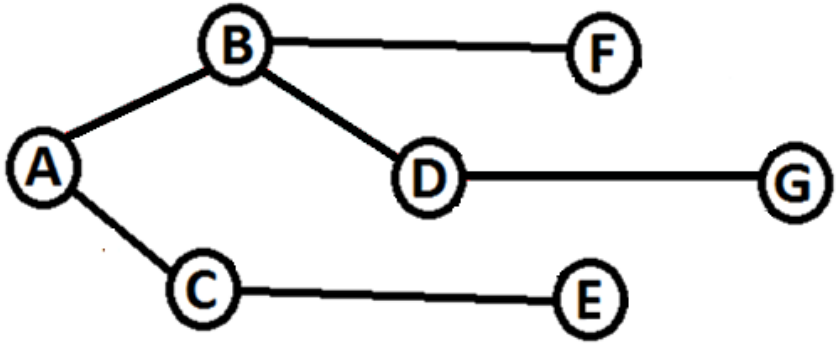


Définitions et terminologies

Propriété 2 : Un graphe est connexe s'il possède un arbre couvrant.



Graphe G



Un Arbre couvrant de G

Algorithme de recherche d'un arbre minimal d'un graphe :

Algorithme de Kuskal

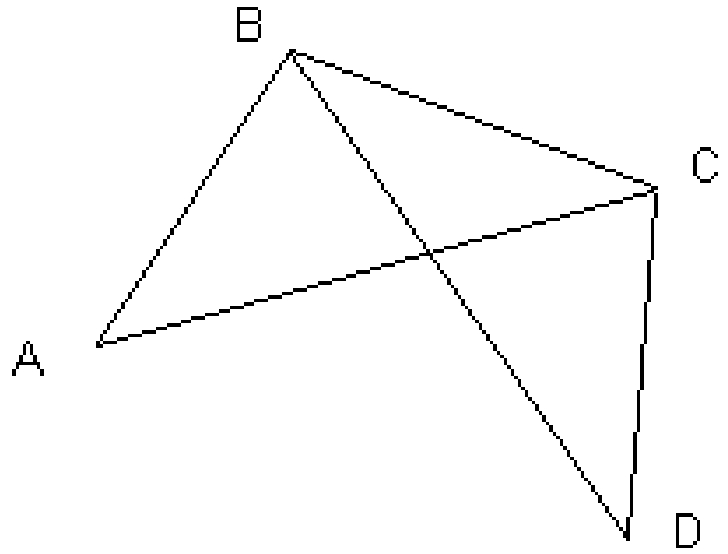
Algorithme de Prime

Définitions et terminologies

Chaîne hamiltonienne : Chaîne passant une seule fois par tous les sommets d'un graphe.

Exemples :

ABCD, ABDC, ACBD,ACBD, ACDB



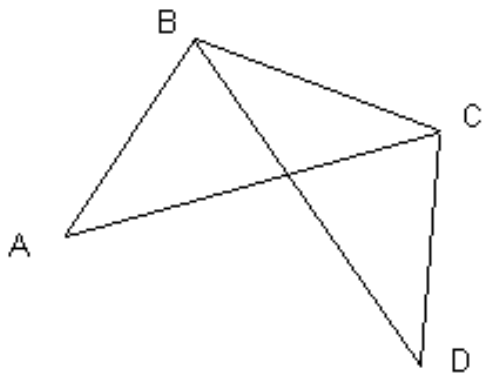
Définitions et terminologies

Chaîne eulérienne : Chaîne passant une seule fois par toutes les arêtes d'un graphe.

Exemples :

BACBDC est une chaîne eulérienne

ACDBACB n'est pas une chaîne eulérienne



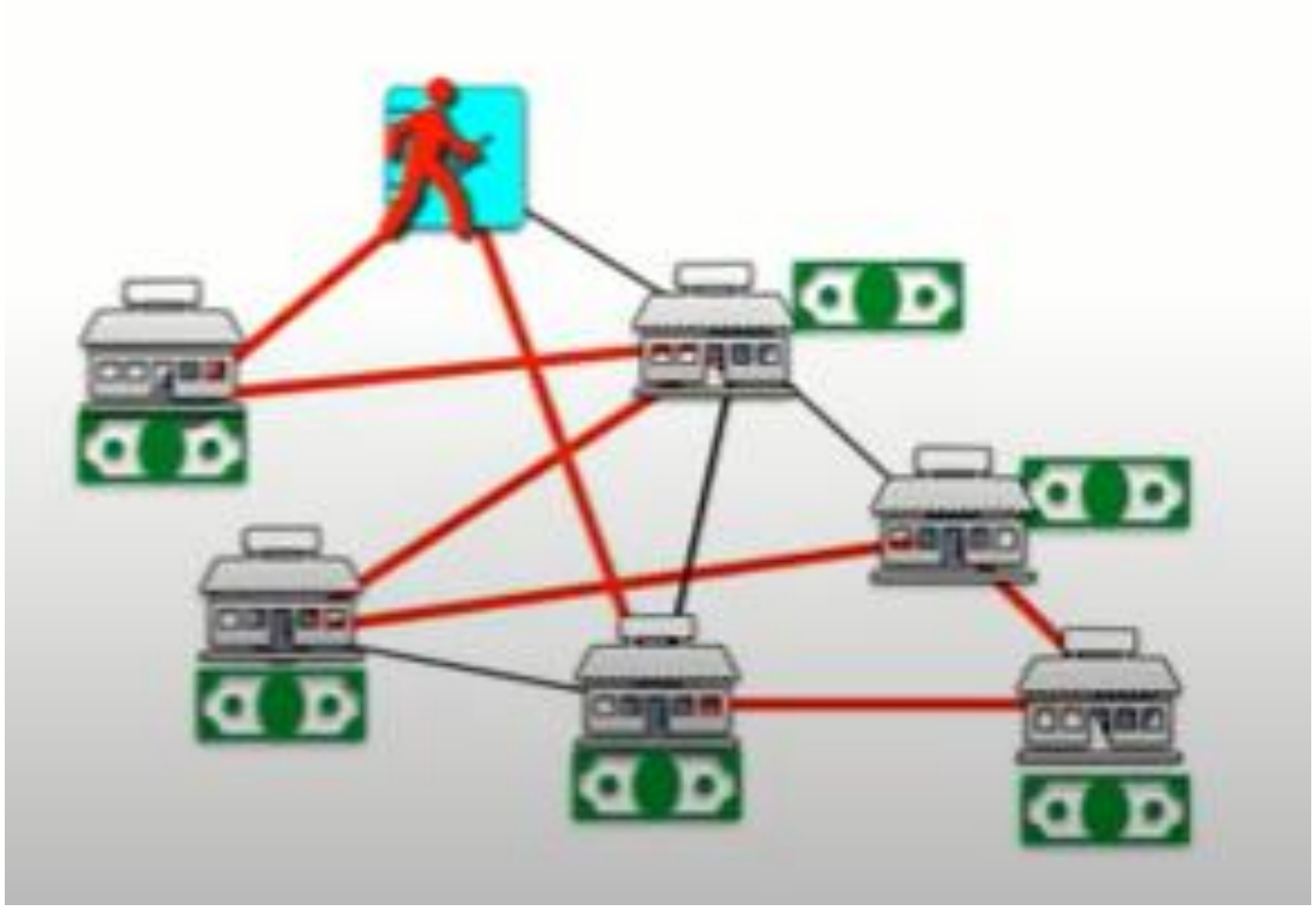
Définitions et terminologies

Cycle hamiltonien : passant une seule fois par tous les sommets d'un graphe et revenant au sommet de départ

Cycle eulérien : passant une seule fois par toutes les arêtes d'un graphe et revenant au sommet de départ.

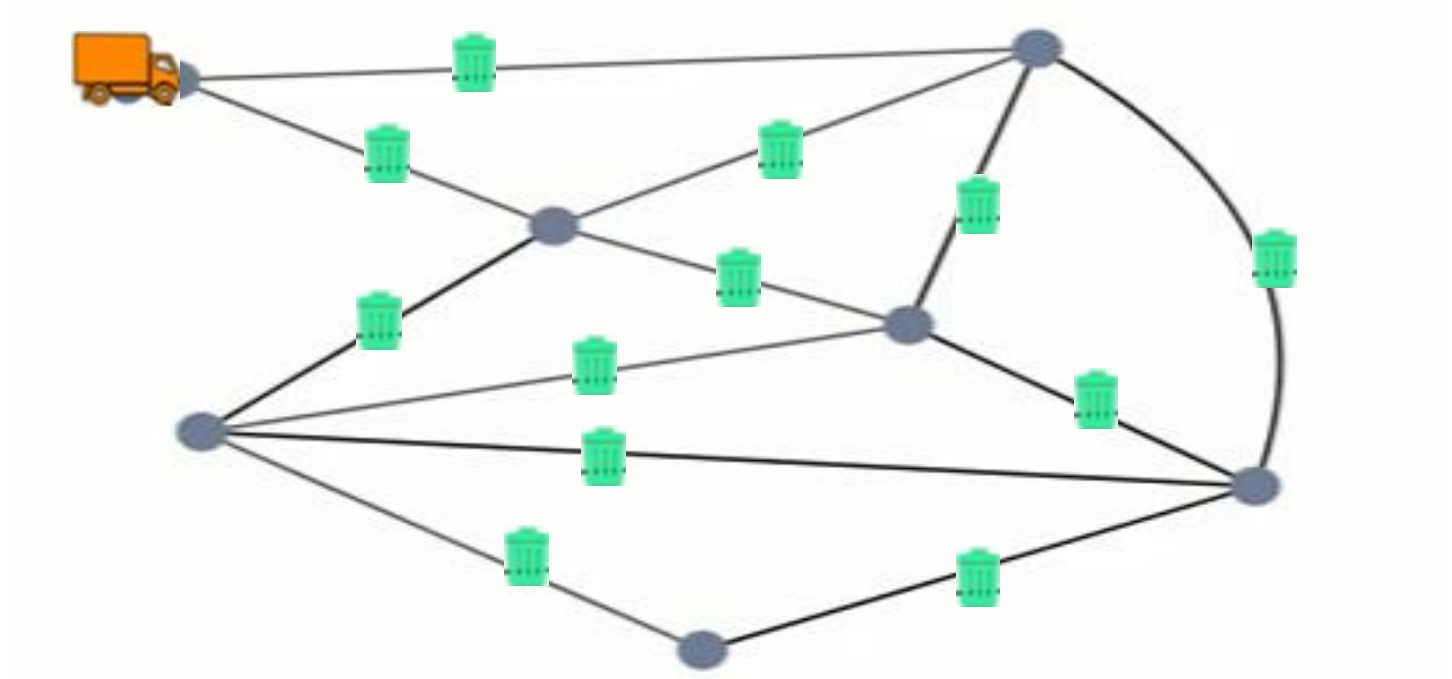
Définitions et terminologies

Exemple cycle Hamiltonien :



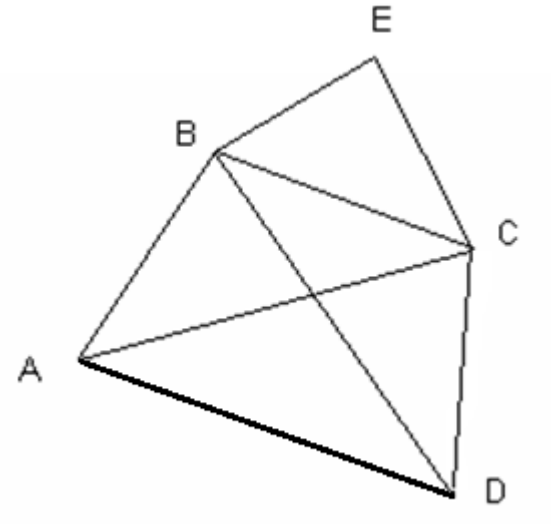
Définitions et terminologies

Exemple cycle Eulérien :

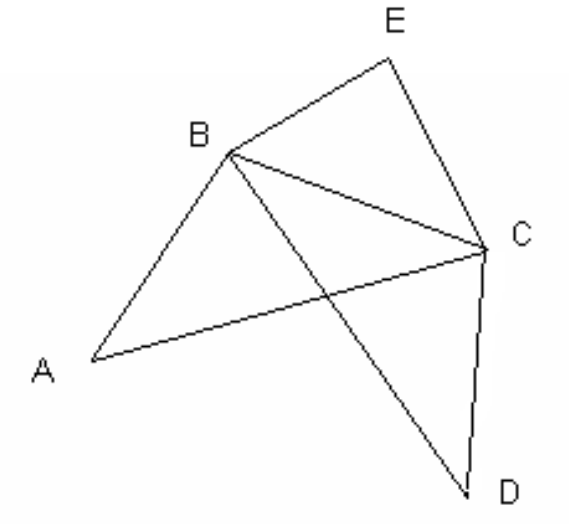


Définitions et terminologies

Existe-t-il un cycle eulérien ?



Réponse :
NON



Réponse : ABECDBCA

Définitions et terminologies

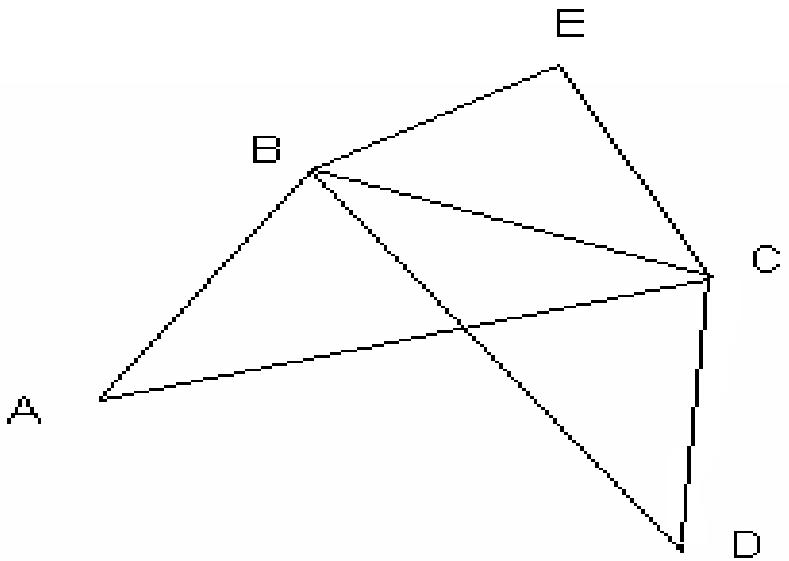
- **Graphe hamiltonien :** Graphe qui possède au moins un cycle Hamiltonien

- **Graphe eulérien :** Graphe qui possède au moins un cycle Eulérien

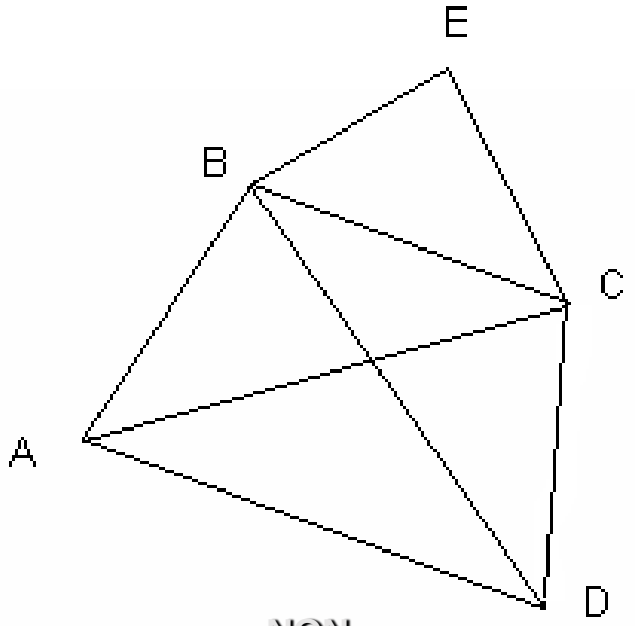
Définitions et terminologies

Théorème d'Euler (1766)

Un graphe est eulérien si tous les sommets du graphe ont un degré pair



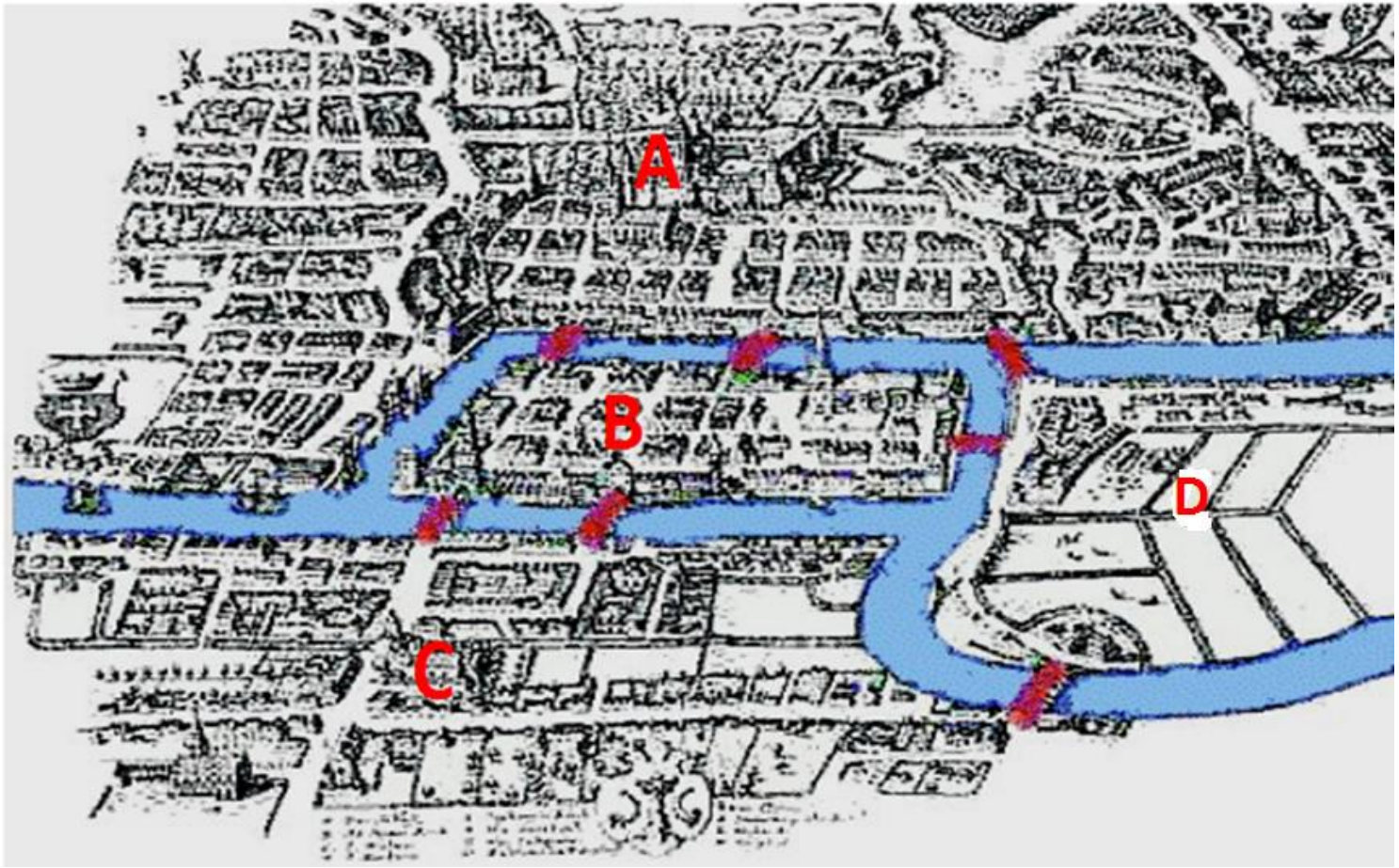
OUI



NON

Définitions et terminologies

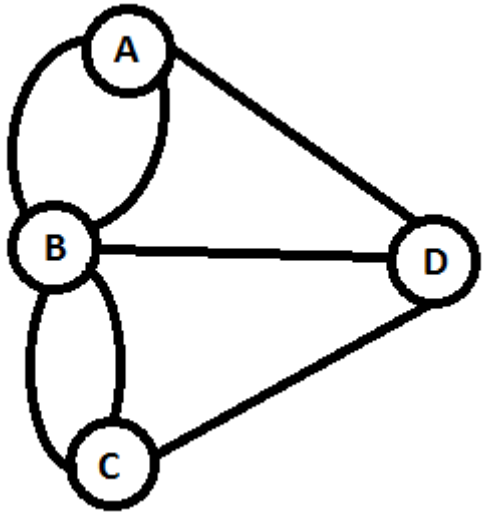
Retour à Konigsberg



Définitions et terminologies

Sous forme de graphe

- **Les sommets = quartiers**
- **Les arcs = Les ponts**
- **Le problème ⇔ le graphe est il eulérien ?**
- **Théorème ⇒ NON**



Implémentation des graphes

Deux implémentations possibles :

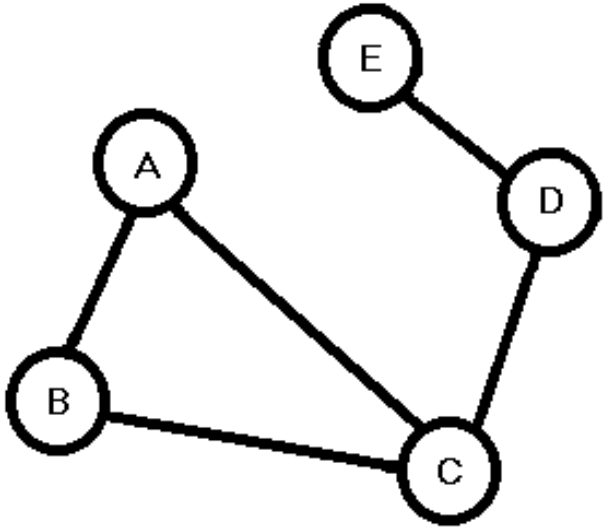
1. Par les listes d’adjacences

2. Par la matrice d’adjacences

Implémentation des graphes

Implémentation par : Matrice d'adjacence

Graphe non orienté :

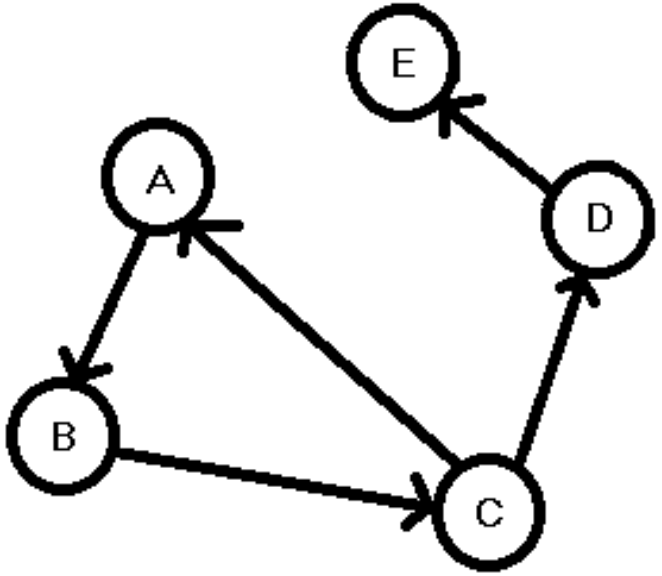


	A	B	C	D	E
A	0	**1**	**1**	0	0
B	**1**	0	**1**	0	0
C	**1**	**1**	0	**1**	0
D	0	0	**1**	0	**1**
E	0	0	0	**1**	0

Implémentation des graphes

Implémentation par : Matrice d'adjacence

Graphe orienté :

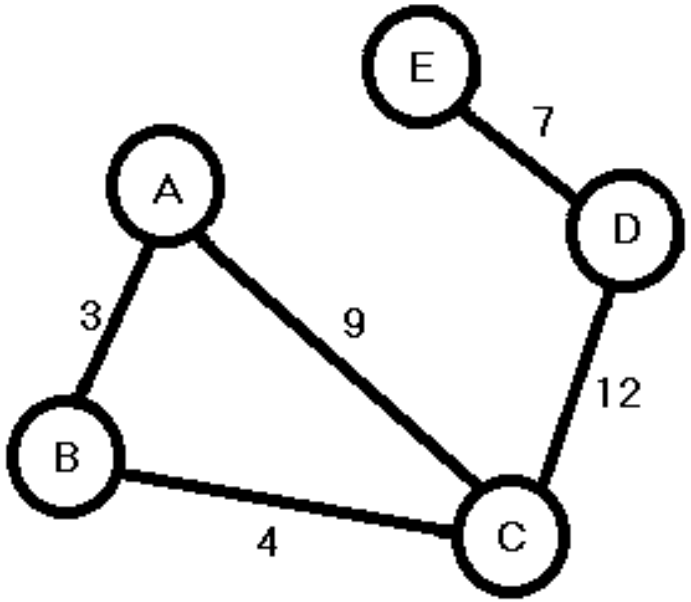


	A	B	C	D	E
A	0	**1**	0	0	0
B	0	0	**1**	0	0
C	**1**	0	0	**1**	0
D	0	0	0	0	**1**
E	0	0	0	0	0

Implémentation des graphes

Implémentation par : Matrice d'adjacence

Graphe non orienté et valué :

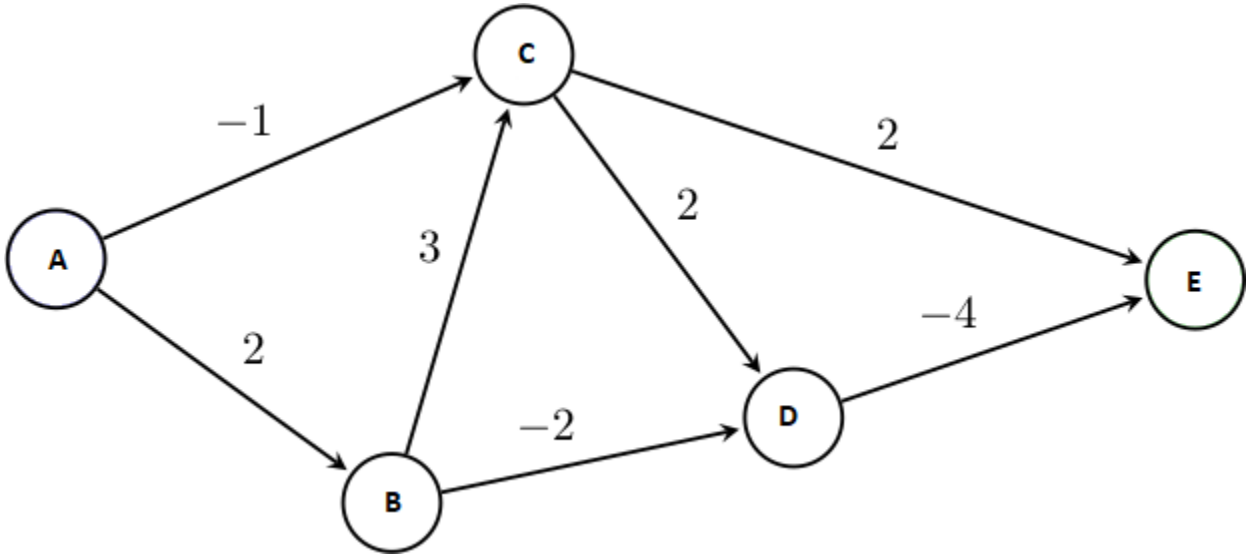


	A	B	C	D	E
A	0	**3**	**9**	0	0
B	**3**	0	**4**	0	0
C	**9**	**4**	0	**12**	0
D	0	0	**12**	0	**7**
E	0	0	0	**7**	0

Implémentation des graphes

Implémentation par : Matrice d'adjacence

Exercice : Donner l'implémentation avec la matrice d'adjacence du graphe suivant.



0	2	-1	0	0
0	0	3	-2	0
0	0	0	2	2
0	0	0	0	-4
0	0	0	0	0

Implémentation des graphes

Implémentation par : Matrice d'adjacence

```
#define MAX 100
typedef struct {
    int matrice[MAX][MAX];
    int ordre;
} Graphe;
```

Algorithmes

- **Parcours d'un graphe**
- **Recherche chemin plus court : Dijkstra, L'algorithme A*,…**
- Coloration de Graphe
- Arbre couvrant minimal
- Détection de la présence de cycles
- Détection de la connexité d'un graphe non orienté
- Distances dans un graphe (Floyd-Warshall).
- ….

Algorithmes de parcours

Le parcours d'un graphe sert à :

- explorer tous les sommets et arêtes,
- vérifier la connectivité,
- trouver un chemin,
- détecter des cycles,
- générer des arbres couvrants,
- résoudre des problèmes (labyrinthes, réseaux, web crawling…).

Le parcours d'un graphe sert à :

- Parcours en largeur
- Parcours en profondeur

Algorithmes de parcours

Parcours en largeur :

Ce parcours peut être utilisé pour :

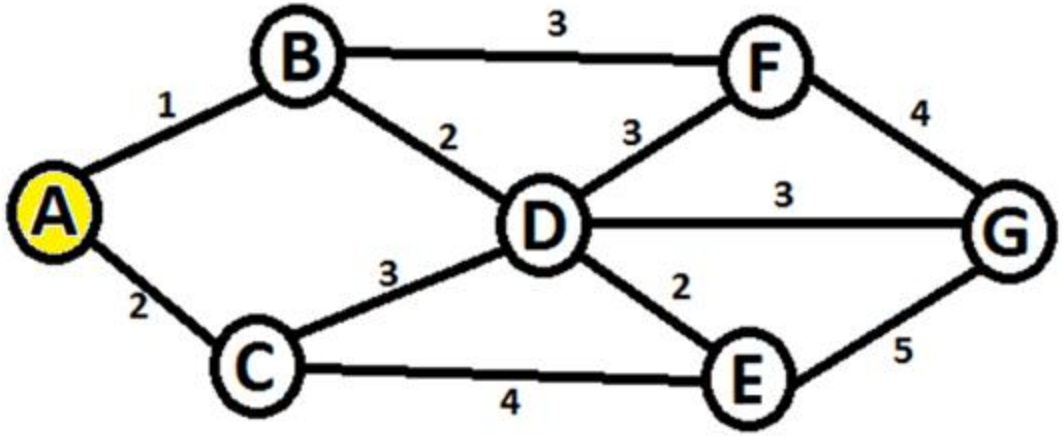
- Chemin plus court entre deux sommets
- Vérifier si un graphe est connexe

Algorithme :

1- **d** est le sommet de départ
2- Cet algorithme liste d'abord les voisins de d pour ensuite les explorer un par un.
3- Répétez (1) pour chaque voisin.

Algorithmes de parcours

Parcours en largeur :

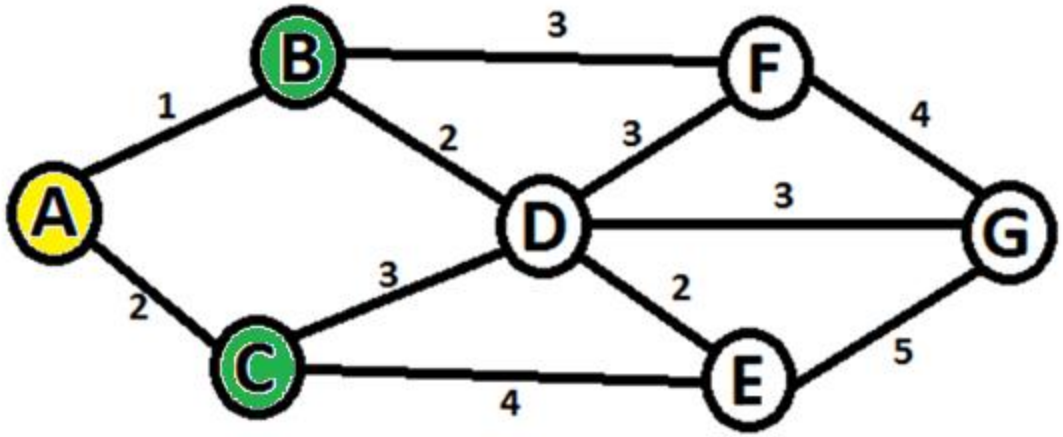


Liste des sommets visités : [ ]

Liste des sommets à visités : [A]

Algorithmes de parcours

Parcours en largeur :

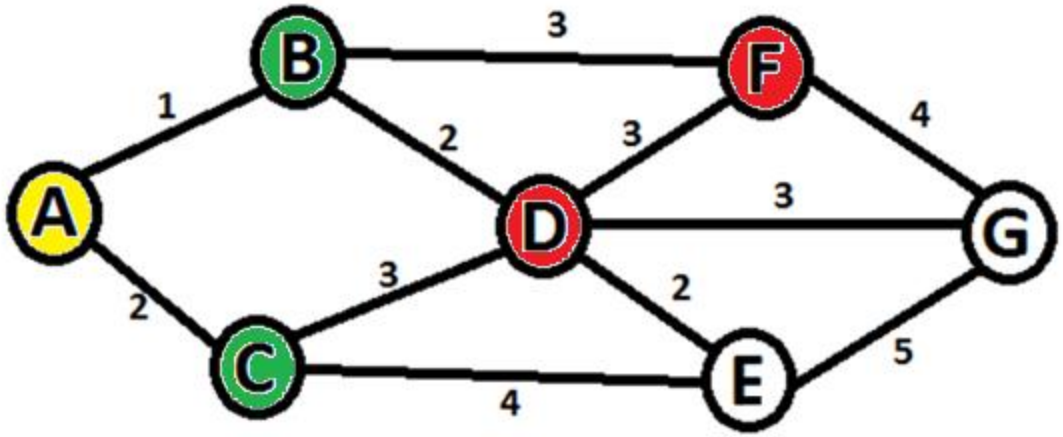


Liste des sommets visités : [A]

Liste des sommets à visités : [B, C]

Algorithmes de parcours

Parcours en largeur :

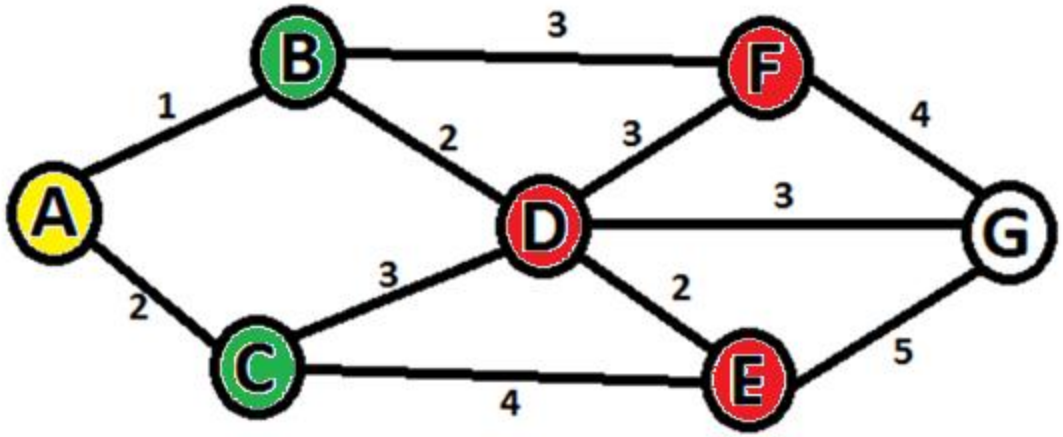


Liste des sommets visités : [A, B]

Liste des sommets à visités : [C, F, D]

Algorithmes de parcours

Parcours en largeur :

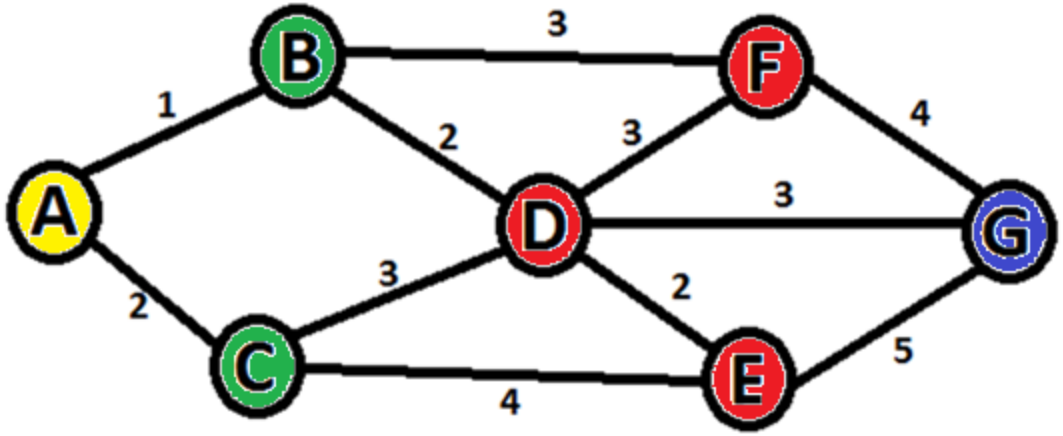


Liste des sommets visités : [A, B, C]

Liste des sommets à visités : [F, D, E]

Algorithmes de parcours

Parcours en largeur :

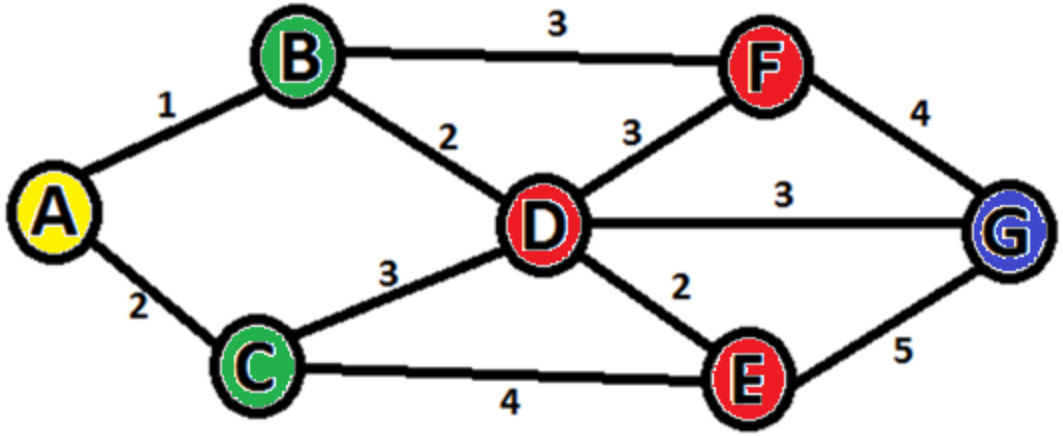


Liste des sommets visités : [A, B, C, F]

Liste des sommets à visités : [D, E, G]

Algorithmes de parcours

Parcours en largeur :

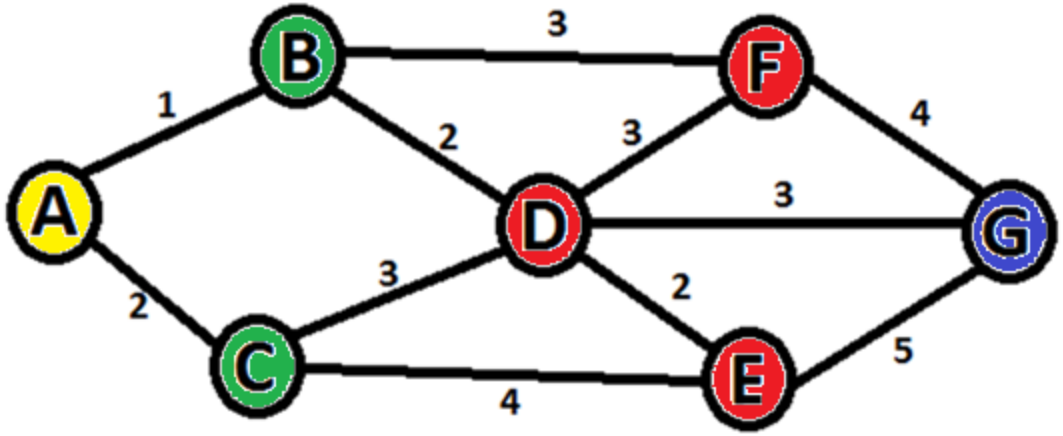


Liste des sommets visités : [A, B, C, F, D]

Liste des sommets à visités : [E, G]

Algorithmes de parcours

Parcours en largeur :

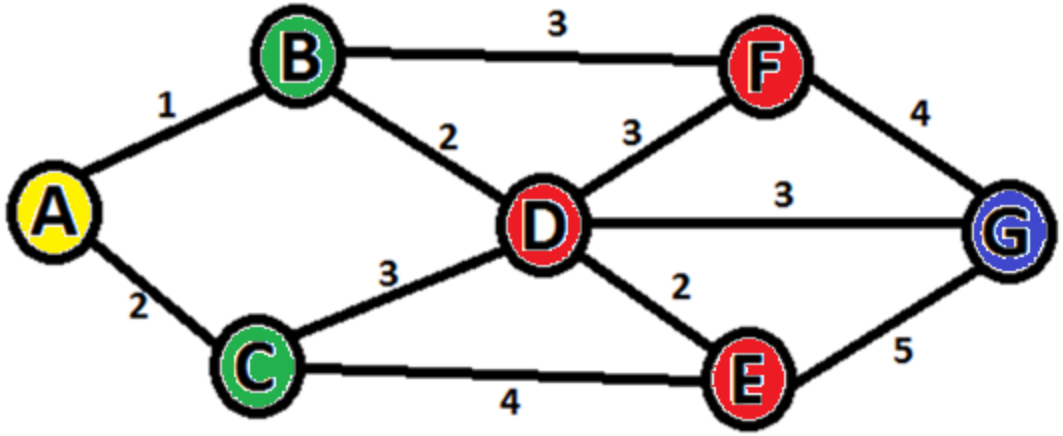


Liste des sommets visités : [A, B, C, F, D, E]

Liste des sommets à visités : [G]

Algorithmes de parcours

Parcours en largeur :



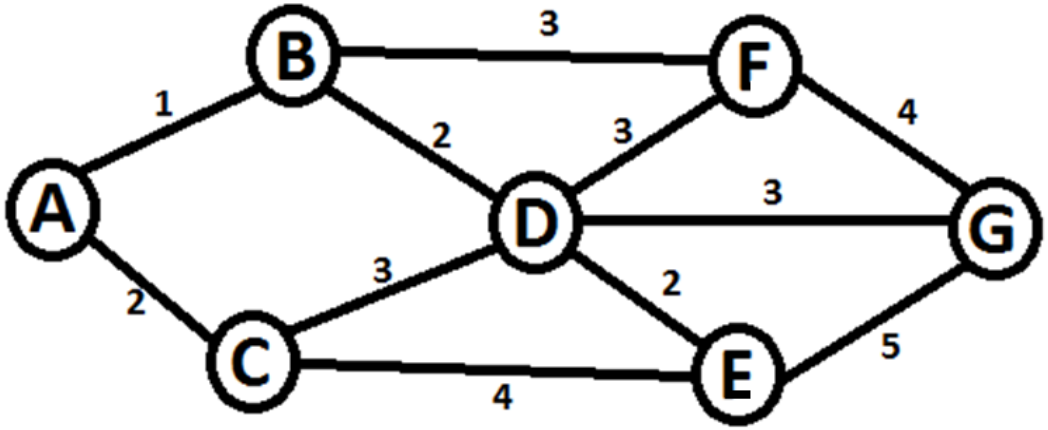
Liste des sommets visités : [A, B, C, F, D, E, G]

Liste des sommets à visités : [ ]

Algorithmes de parcours

Parcours en largeur :

Ecrire la fonction « parcours_largeur(Graphe* g, int debut) » qui fait le parcours en largeur du graphe **G** passé en paramètre à partir du sommet de départ **d**.



Algorithmes de parcours

Parcours en largeur :

```
void parcoursLargeur(Graphe* g, int debut) {
    int visite[MAX] = {0};
    int file[MAX];
    int debutFile = 0, finFile = 0;

    visite[debut] = 1;
    file[finFile++] = debut;

    while (debutFile < finFile) {
        int sommet = file[debutFile++];
        // Traiter le sommet (par exemple, l'afficher)
        for (int i = 0; i < g->ordre; i++) {
            if (g->matrice[sommet][i] == 1 && !visite[i]) {
                visite[i] = 1;
                file[finFile++] = i;
            }
        }
    }
}
```

Algorithmes de parcours

Parcours en profondeur :

Ce parcours peut être utilisé pour :

- Chemin plus court entre deux sommets
- Vérifier si un graphe est connexe

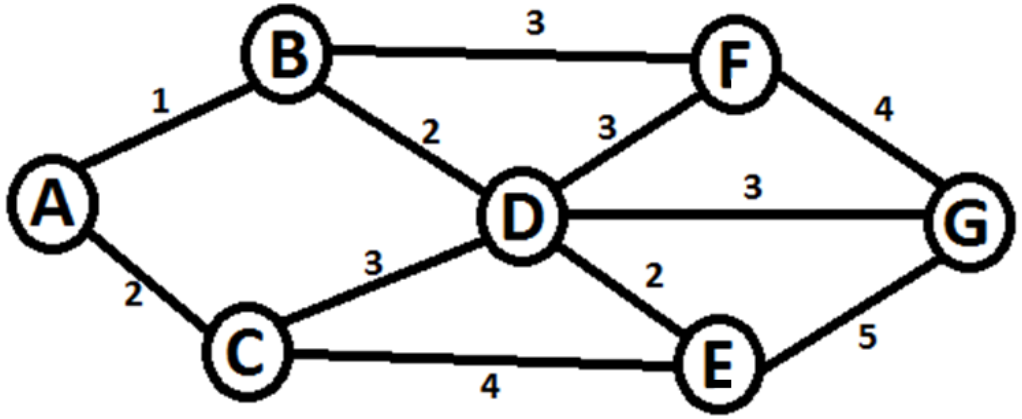
Algorithme :

Recherche qui progresse à partir d'un sommet d en s'appelant récursivement pour chaque sommet voisin de d :
Pour chaque sommet, il prend le premier sommet voisin jusqu'à ce qu'un sommet n'ait plus de voisins (ou que tous ses voisins soient marqués), et revient alors au sommet père.

Algorithmes de parcours

Parcours en profondeur :

Ecrire la fonction « parcours_profondeur» qui fait le parcours en profondeur du graphe **G** passé en paramètre à partir du sommet de départ **d**.



Le résultat du parcours en profondeur du graphe en dessus à partir de A sera : A, B, F ,G ,D, E, C

Algorithmes de parcours

Parcours en profondeur :

```
void parcoursProfondeur(Graphe* g, int sommet, int *visite) {
    visite[sommet] = 1;
    // Traiter le sommet (par exemple, l'afficher)
    for (int i = 0; i < g->ordre; i++) {
        if (g->matrice[sommet][i] == 1 && !visite[i]) {
            parcoursProfondeur(g, i, visite);
        }
    }
}
```

Algorithmes de Dijkstra

En théorie des graphes, l'**algorithme de Dijkstra** sert à résoudre le problème du plus court chemin.

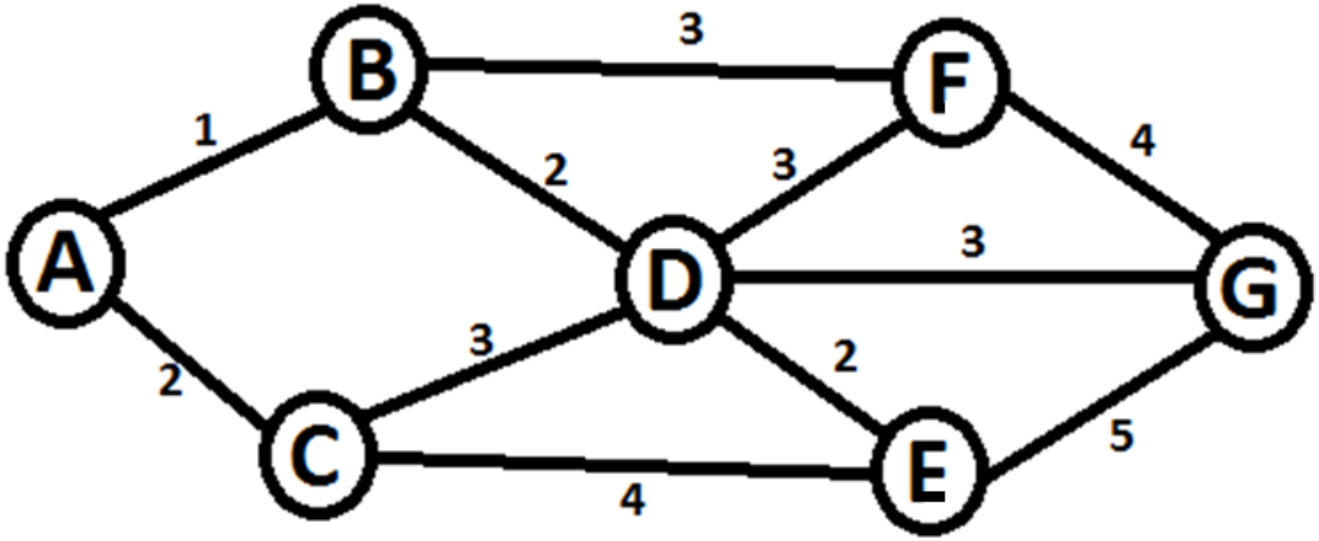
Il s'applique à un graphe connexe dont le poids lié aux arêtes est un réel positif.

Applications :

1. Chemin le plus court entre deux villes
2. Chemin le plus rapide entre deux poins en ville.
3. Routage d’information optimal
4. Intéligence artificielle (A*)
5. ….

Algorithmes de Dijkstra

Exemple : Trouver le chemin optimal entre A et G



Algorithmes de Dijkstra

Initialisation de l'algorithme :

Étape 1 : On affecte le poids 0 au sommet origine (E) et on attribue provisoirement un poids ∞ aux autres sommets.

Répéter les opérations suivantes de l'étape 2 et 3 tant que le sommet de sortie (s) n'est pas affecté d'un poids définitif :

Étape 2 : Parmi les sommets dont le poids n'est pas définitivement fixé choisir le sommet X de poids *p* minimal. Marquer définitivement ce sommet **X** affecté du poids **p(X)**.

Étape 3 : Pour tous les sommets Y qui ne sont pas définitivement marqués, adjacents au dernier sommet fixé **X** :

- Calculer la somme *s* du poids de X et du poids de l'arête reliant X à Y.
- Si la somme *s* est inférieure au poids provisoirement affecté au sommet Y, affecter provisoirement à Y le nouveau poids *s* et indiquer entre parenthèses le sommet X pour se souvenir de sa provenance.

Quand le sommet s est définitivement marqué

Le plus court chemin de E à S s'obtient en écrivant de gauche à droite le parcours en partant de la fin S.

Algorithmes de Dijkstra

Etapes de l’algorithme :

	A	B	C	D	E	F	G
Etape 1							
Etape 2							
Etape 3							
Etape 4							
Etape 5							
Etape 6							
Etape 7							

Algorithmes de Dijkstra

Règles pour remplir les cases de chaque cellule:

Soit **e** le sommet de départ.

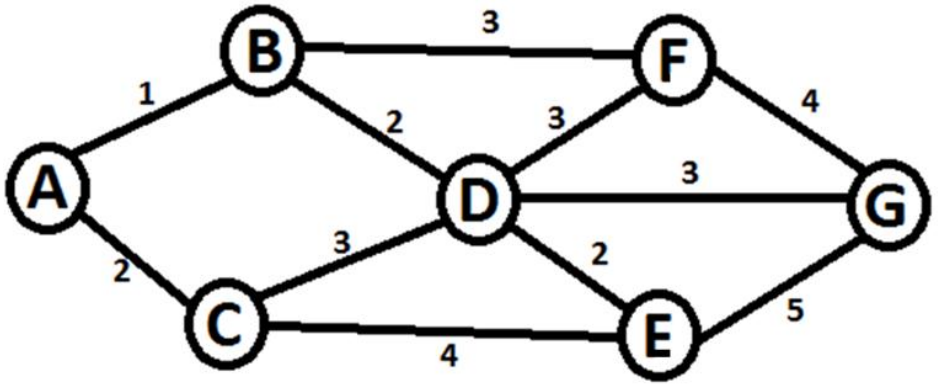
1. La valeur de chaque cellule est un couple : (d(v), p(v))

Où : d(v) est la distance minimale du sommet de départ **e** au somment **v** et p(v) représente le sommet précédent du chemin optimal reliant e et v.

2. d(e)=0 (e est le sommet de départ)
3. d(v)=∞ si la distance est non encore calculé

Algorithmes de Dijkstra

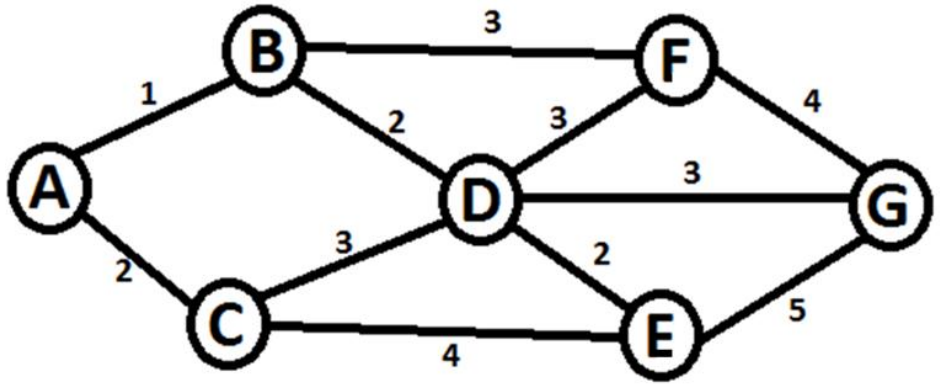
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X						
Etape 3	X						
Etape 4	X						
Etape 5	X						
Etape 6	X						
Etape 7	X						

Algorithmes de Dijkstra

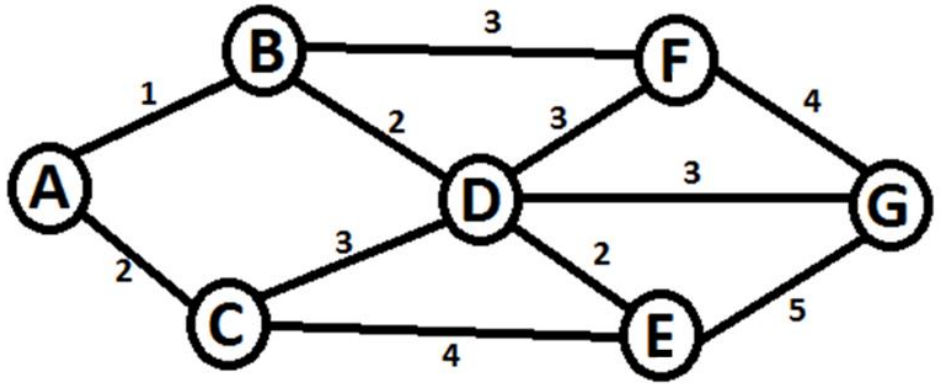
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)				
Etape 3	X						
Etape 4	X						
Etape 5	X						
Etape 6	X						
Etape 7	X						

Algorithmes de Dijkstra

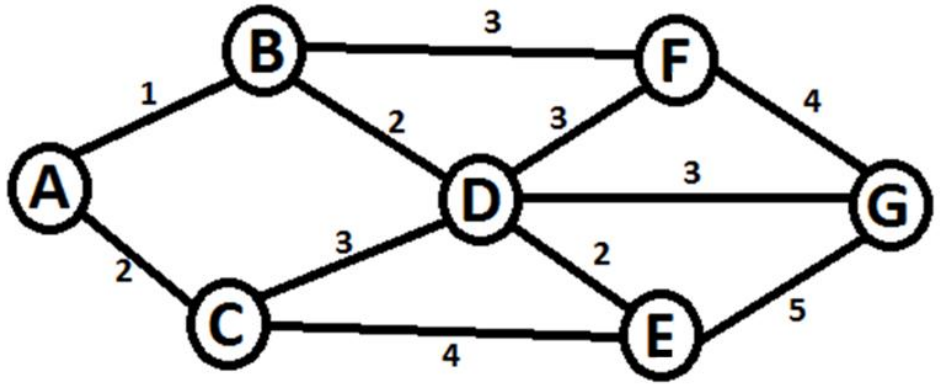
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X						
Etape 4	X						
Etape 5	X						
Etape 6	X						
Etape 7	X						

Algorithmes de Dijkstra

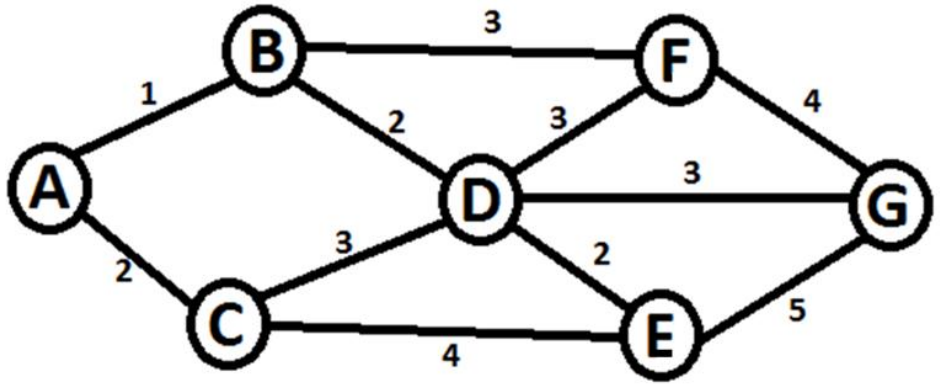
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X					
Etape 4	X	X					
Etape 5	X	X					
Etape 6	X	X					
Etape 7	X	X					

Algorithmes de Dijkstra

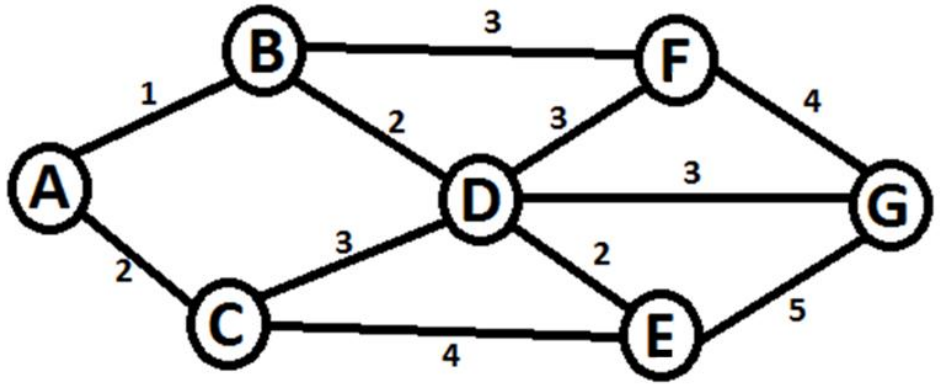
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X		(3,B)		(4,B)	
Etape 4	X	X					
Etape 5	X	X					
Etape 6	X	X					
Etape 7	X	X					

Algorithmes de Dijkstra

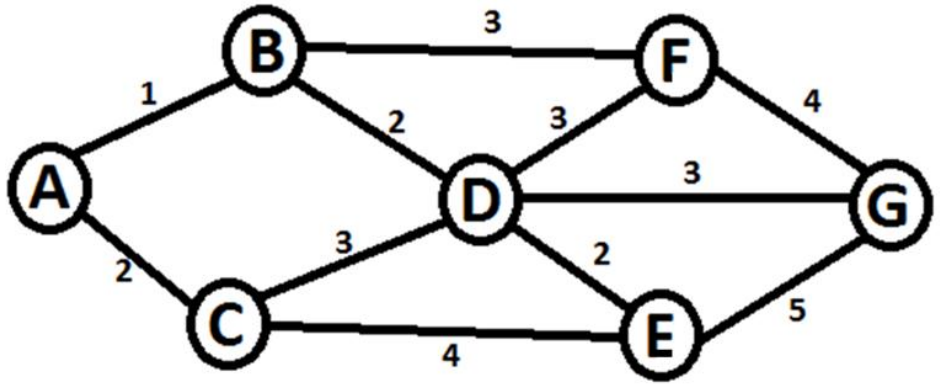
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X					
Etape 5	X	X					
Etape 6	X	X					
Etape 7	X	X					

Algorithmes de Dijkstra

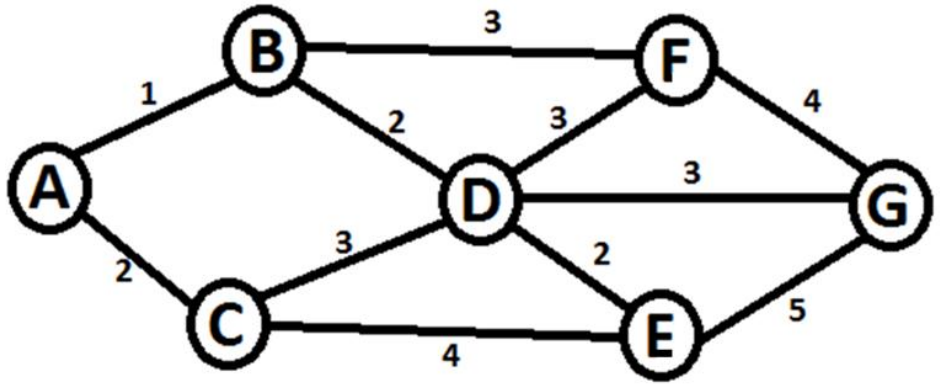
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X		(6, C)		
Etape 5	X	X	X				
Etape 6	X	X	X				
Etape 7	X	X	X				

Algorithmes de Dijkstra

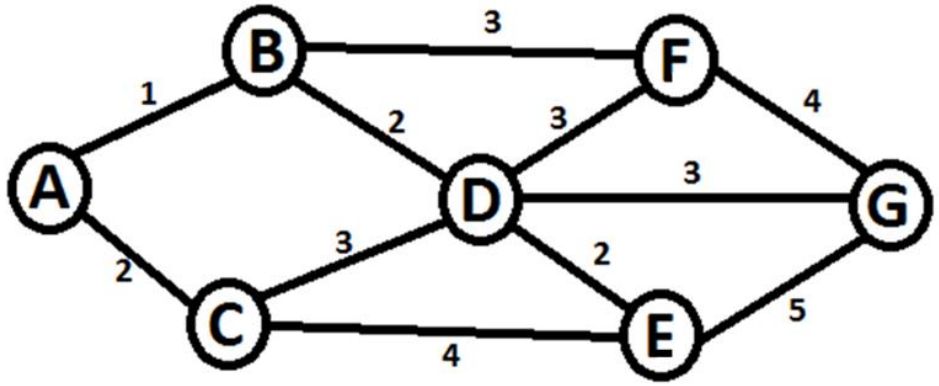
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X	(3,B)	(6, C)	(4,B)	∞
Etape 5	X	X	X				
Etape 6	X	X	X				
Etape 7	X	X	X				

Algorithmes de Dijkstra

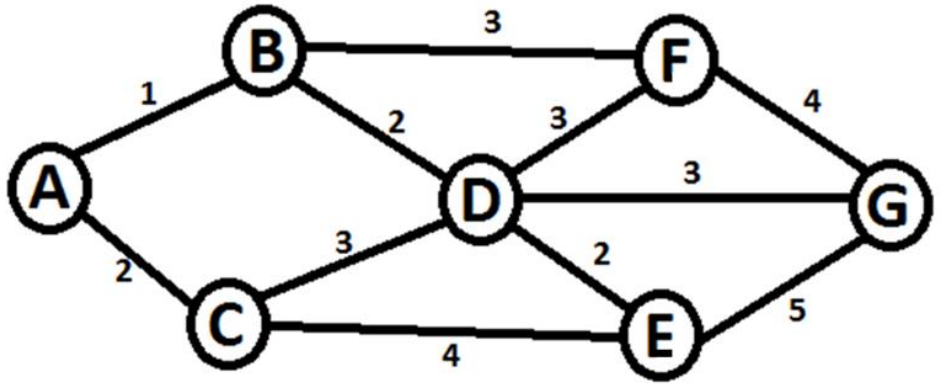
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X	(3,B)	(6,C)	(4,B)	∞
Etape 5	X	X	X	X	(5,D)	(4,B)	(6,D)
Etape 6	X	X	X	X			
Etape 7	X	X	X	X			

Algorithmes de Dijkstra

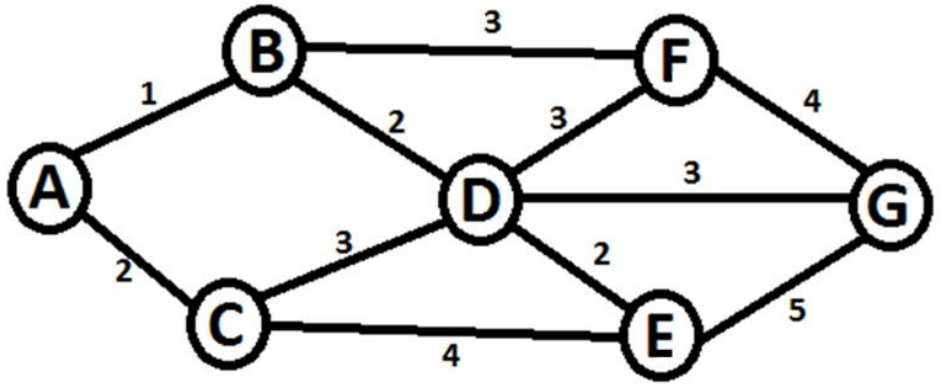
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X	(3,B)	(6,C)	(4,B)	∞
Etape 5	X	X	X	X	(5,D)	(4,B)	(6,D)
Etape 6	X	X	X	X		X	
Etape 7	X	X	X	X		X	

Algorithmes de Dijkstra

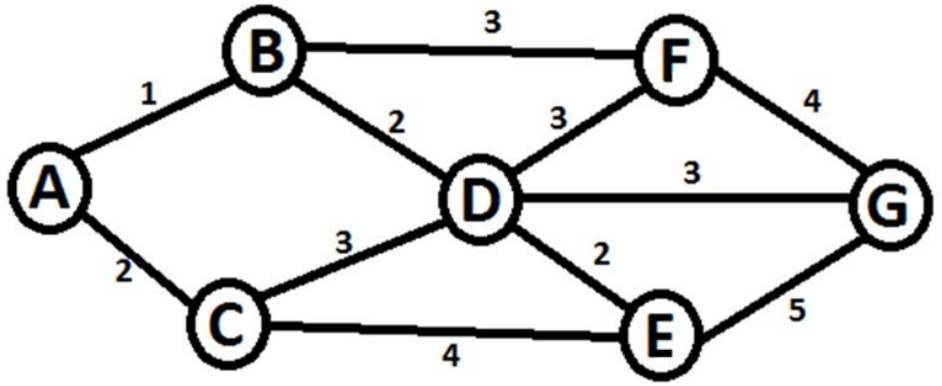
Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X	(3,B)	(6,C)	(4,B)	∞
Etape 5	X	X	X	X	(5,D)	(4,B)	(6,D)
Etape 6	X	X	X	X	(5,D)	X	(6,D)
Etape 7	X	X	X	X		X	

Algorithmes de Dijkstra

Etapes de l'algorithme :



	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
Etape 2	X	(1,A)	(2,A)	∞	∞	∞	∞
Etape 3	X	X	(2,A)	(3,B)	∞	(4,B)	∞
Etape 4	X	X	X	(3,B)	(6,C)	(4,B)	∞
Etape 5	X	X	X	X	(5,D)	(4,B)	(6,D)
Etape 6	X	X	X	X	(5,D)	X	(6,D)
Etape 7	X	X	X	X	X	X	**(6,D)**

Algorithmes de Dijkstra

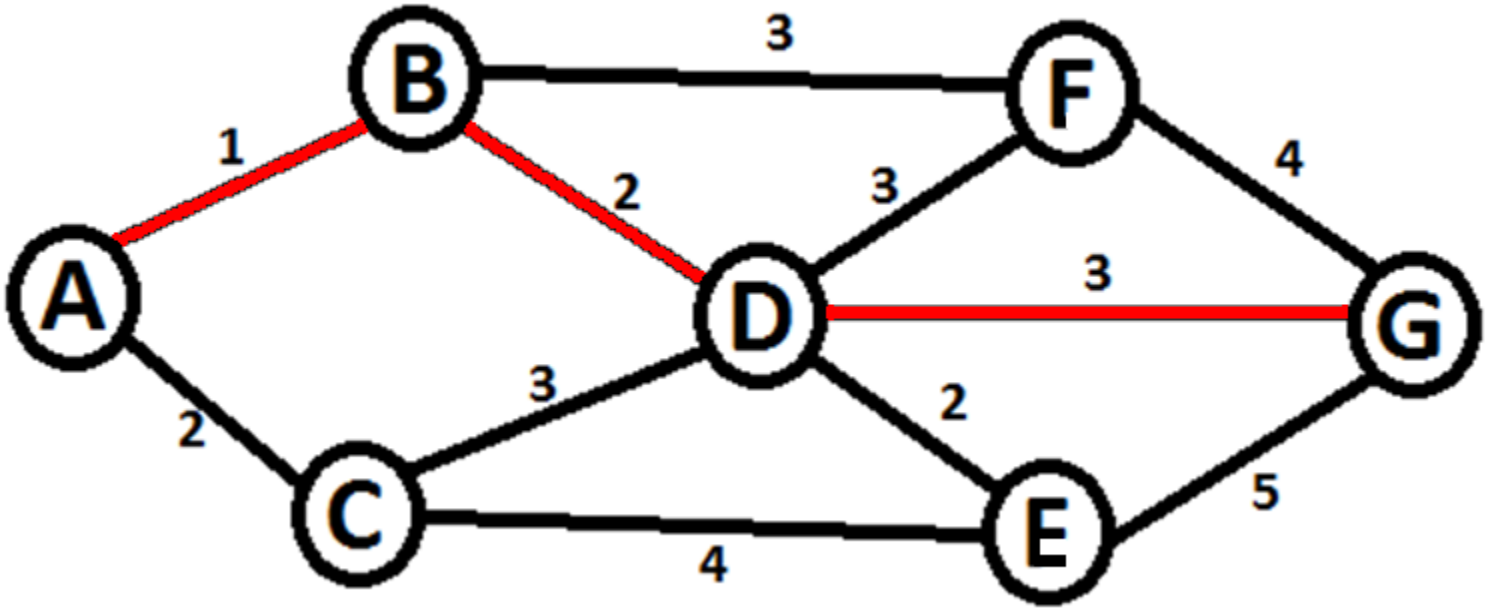
Etapes de l'algorithme :

Etapes de l'algorithme :

	A	B	C	D	E	F	G
Etape 1	(0,)	∞	∞	∞	∞	∞	∞
...							
Etape 7	(0,)	(1,A)	(2,A)	(3,B)	(5,D)	(4,B)	(6,D)

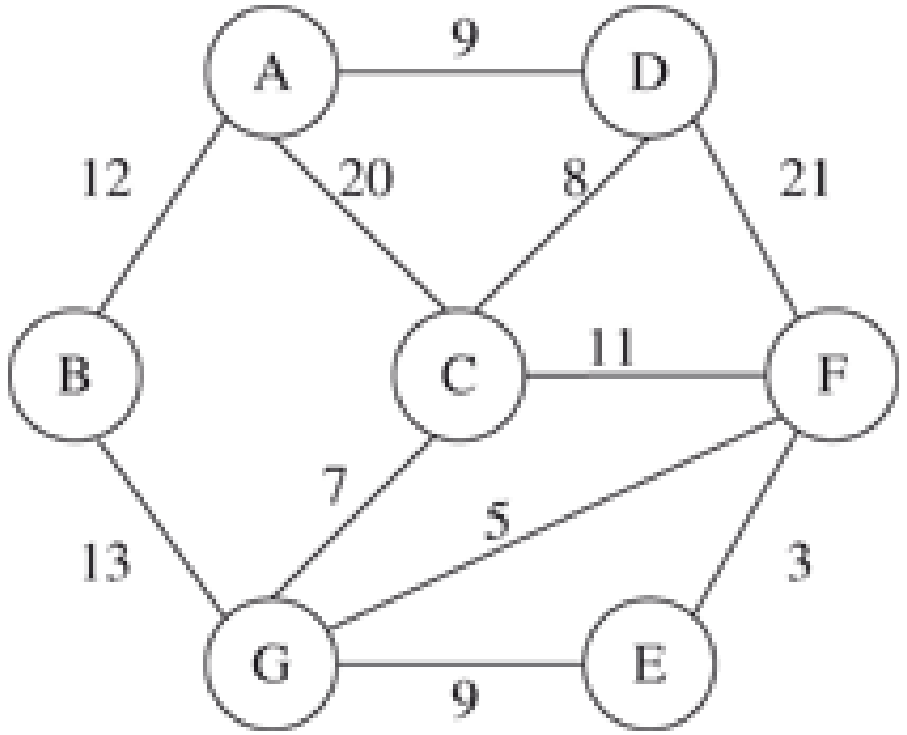
Algorithmes de Dijkstra

Le chemin optimal est : ABDG de coût égal à 6



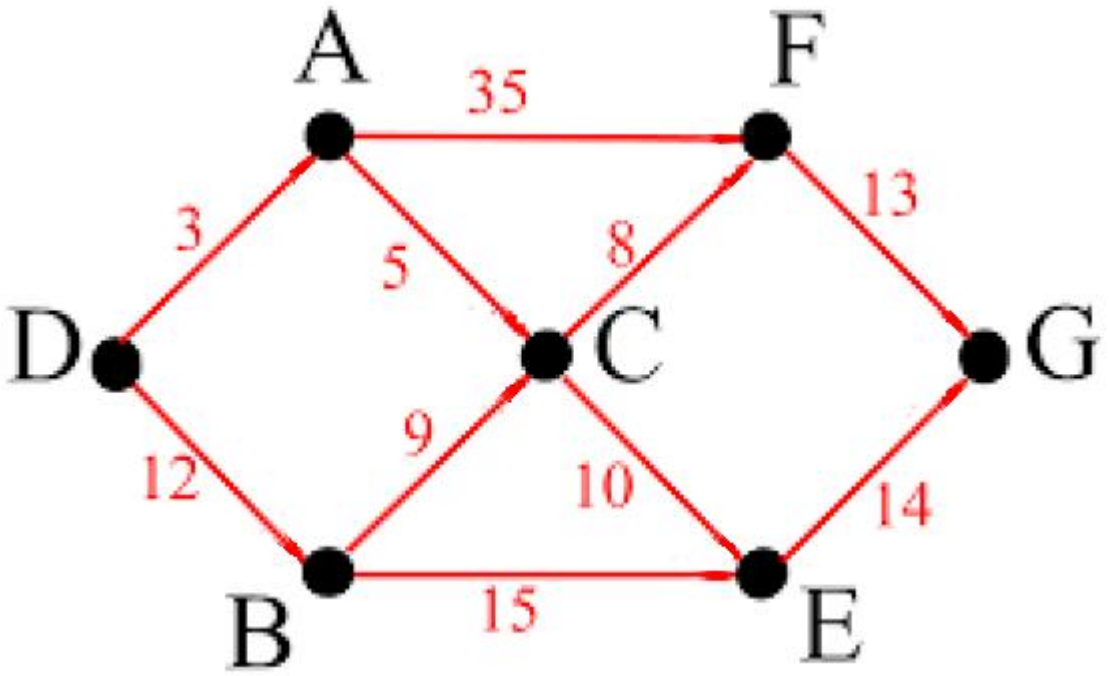
Algorithmes de Dijkstra

Exercice : Trouver le chemin optimal entre A et E



Algorithmes de Dijkstra

Exercice : Trouver le chemin optimal entre D et G



Algorithmes de Dijkstra

Etapes de l'algorithme :

	A	B	C	D	E	F	G
Etape 1	(0,A)	∞	∞	∞	∞	∞	∞
...							
Etape 7	(0,A)	(1,A)	(2,A)	(3,B)	(5,D)	(4,B)	(6,D)

En python on doit calculer deux dictionnaires :

D ={ 'A':0, 'B':1, 'C':2, 'D':3, 'E':5, 'F':4, 'G':6 } les distances

P ={ 'B':'A', 'C':'A', 'D':'B', 'E':'D', 'F':'B', 'G':'D' } dictionnaire des précédents

Algorithmes de Dijkstra

Programme de Dijkstra

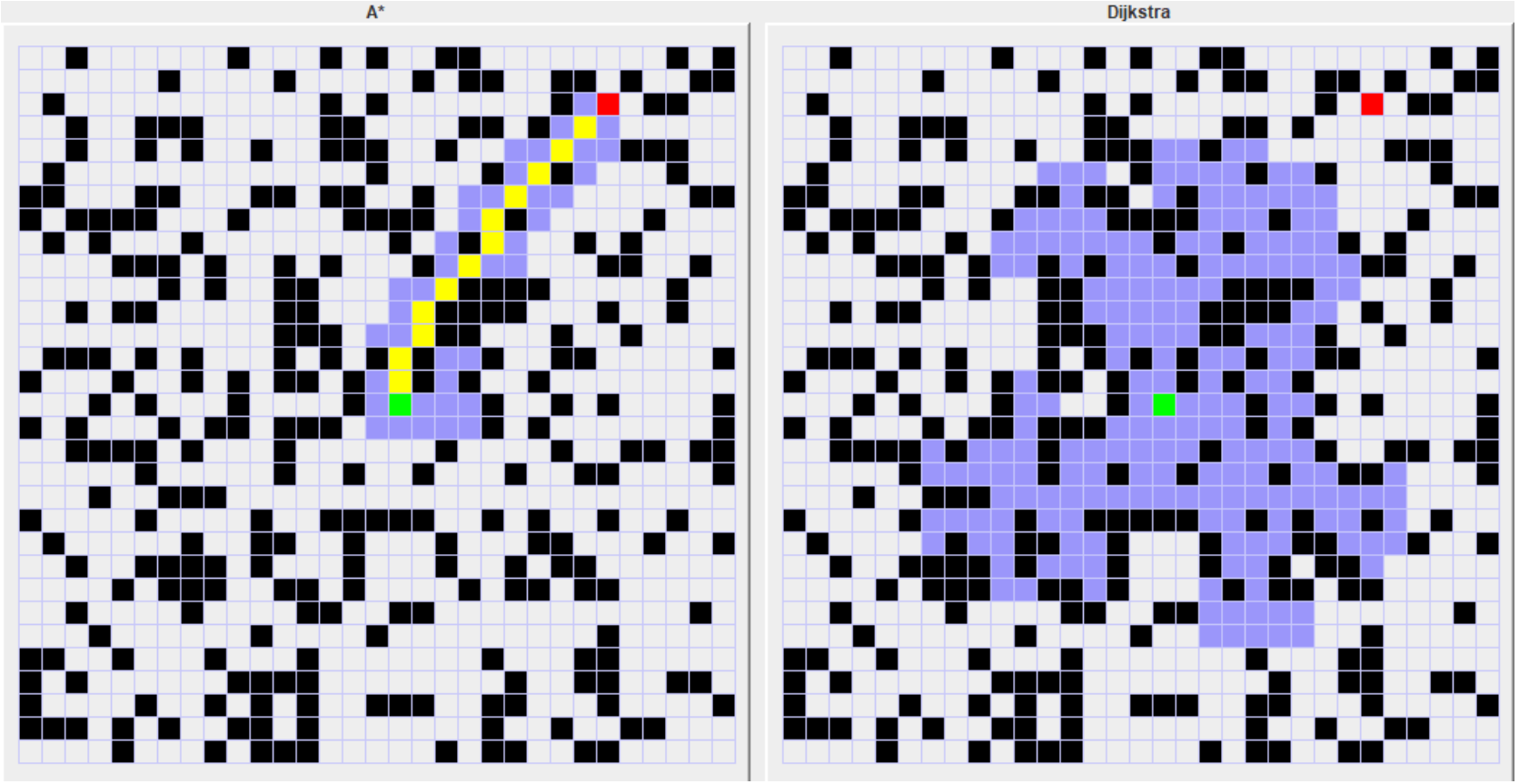
```c
int minDistance(int dist[], int visited[], int n) {
    int min = INF, index = -1;
    for (int i = 0; i < n; i++) {
        if (!visited[i] && dist[i] < min) {
            min = dist[i];
            index = i;
        }
    }
    return index;
}
```

```c
int* dijkstra_path(Graphe* g, int source, int destination, int* path_length) {

    int n = g->ordre;
    int dist[MAX], visited[MAX], parent[MAX];

    // Initialisation
    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        visited[i] = 0;
        parent[i] = -1;
    }

    dist[source] = 0;

    // Dijkstra
    for (int count = 0; count < n - 1; count++) {

        int u = minDistance(dist, visited, n);
        if (u == -1) break;

        visited[u] = 1;

        for (int v = 0; v < n; v++) {
            int poids = g->matrice[u][v];

            if (!visited[v] &&
                poids > 0 &&
                dist[u] != INF &&
                dist[u] + poids < dist[v]) {

                dist[v] = dist[u] + poids;
                parent[v] = u;   // mémoriser le prédécesseur
            }
        }
    }

    // Vérifier si destination est atteignable
    if (dist[destination] == INF) {
        *path_length = 0;
        return NULL;
    }

    // Reconstruction du chemin (à l'envers)
    int temp_path[MAX];
    int idx = 0;
    int current = destination;

    while (current != -1) {
        temp_path[idx++] = current;
        current = parent[current];
    }

    // Le chemin est inversé → on le remet dans l'ordre
    *path_length = idx;
    int* path = malloc(idx * sizeof(int));

    for (int i = 0; i < idx; i++) {
        path[i] = temp_path[idx - 1 - i];
    }

    return path;
}
```

Algorithme A*

Rechercher le chemin le court :

Dijkstra : Donne toujours une solution optimale mais trop lent

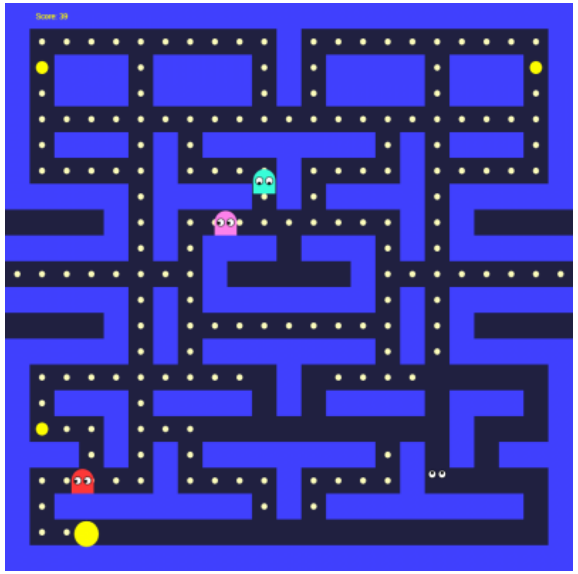


Algorithme A*

L’algorithme A*

- L’algorithme A* est un algorithme heuristique
- Objectif : Trouver très rapidement un court chemin entre deux points avec d’éventuels obstacles.

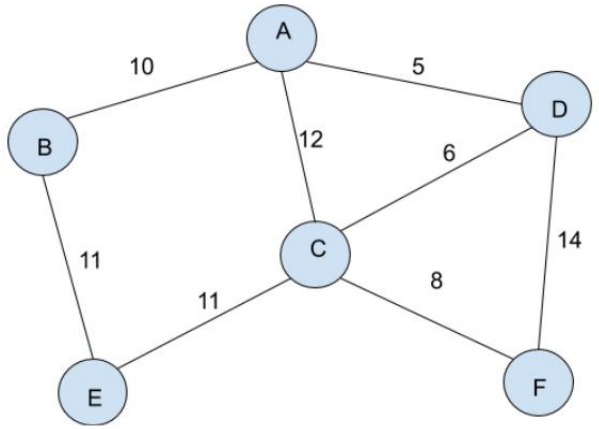
Outre sa vitesse, cet algorithme est réputé pour garantir une solution en sortie.



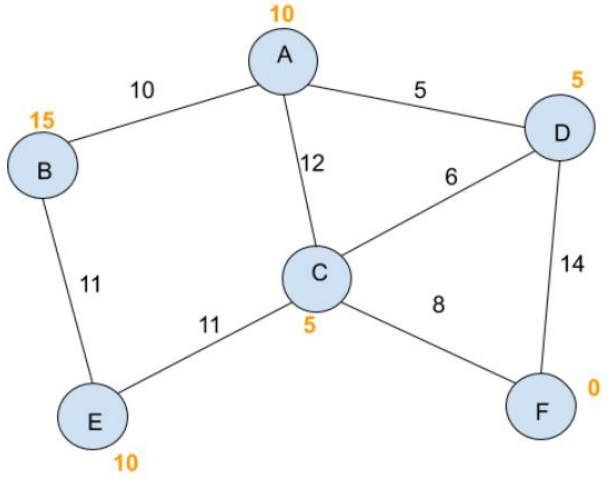
Algorithme A*

A* est une extension de l'algorithme de Dijkstra.

→ Ajout d'une heuristique



Dijkstra se base sur les distances réelles

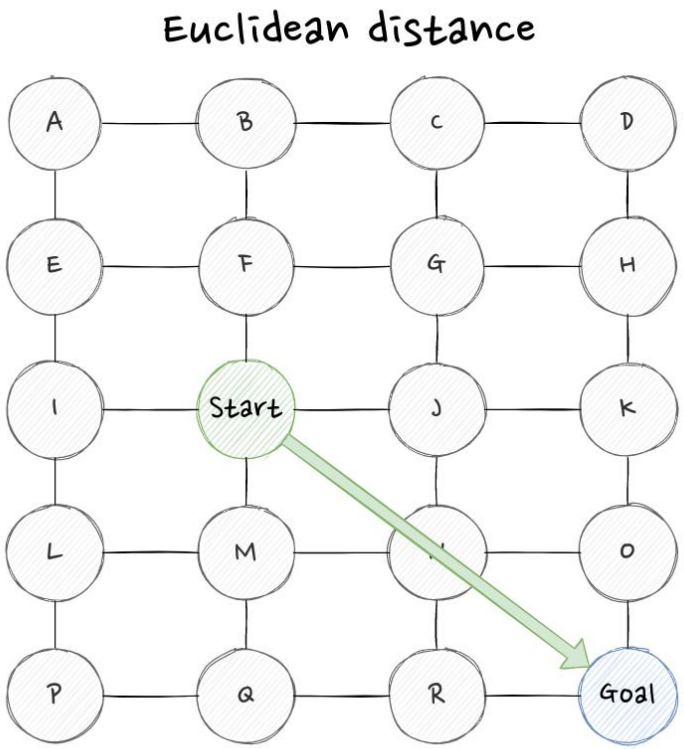


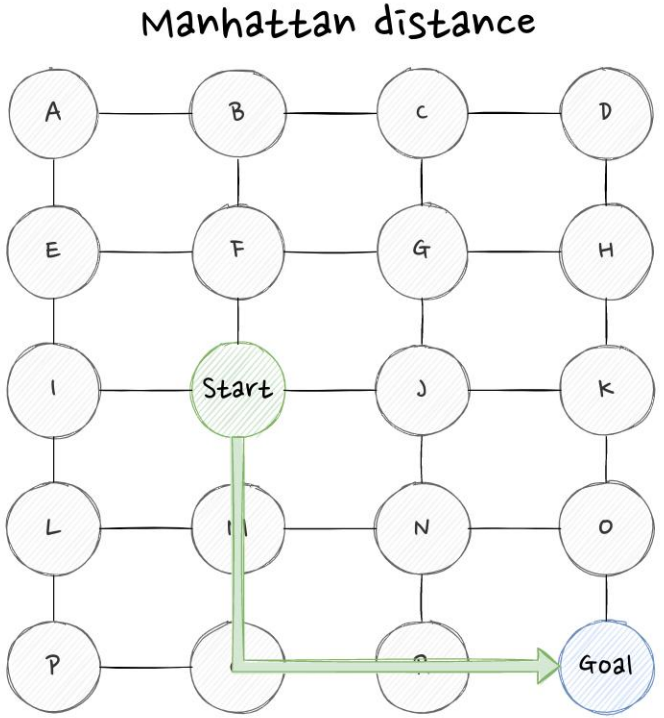
A* utilise distances réelles + une heuristique

Algorithme A*

Exemple heuristique :

Par exemple, L'heuristique peut être obtenu par un calcul de la distance euclidienne (à vol d'oiseau) entre les différents sommets et le sommet but.





Algorithme A*

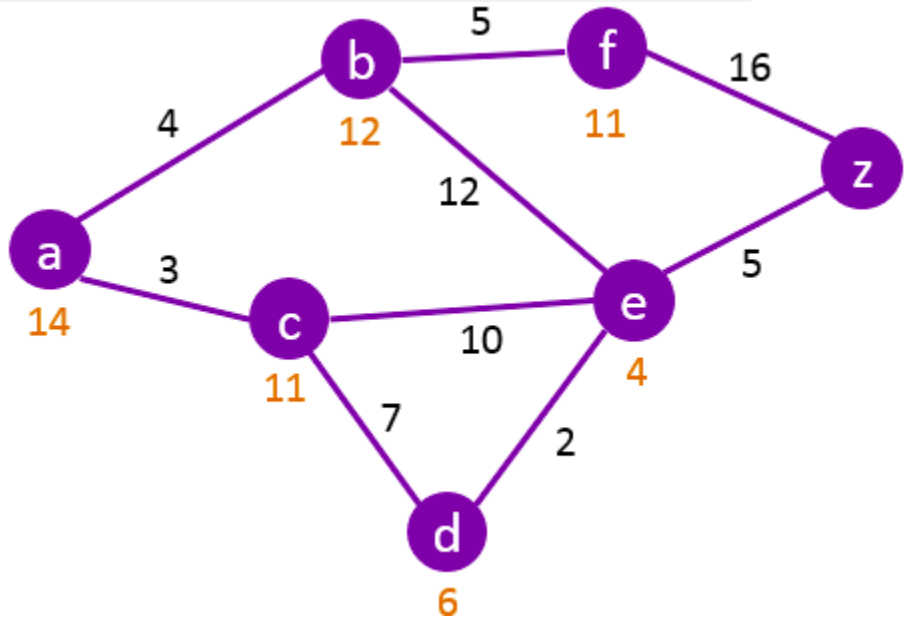
- On évite d'explorer les chemins qui sont déjà chers.
- La mesure d'utilité est donnée par une fonction d'évaluation f.

Pour chaque sommet n, $f(n) = g(n) + h(n)$

- g(n) : est le coût jusqu'à présent pour atteindre n
- h(n) : est le coût estimé pour aller de n vers un état final.
- f(n) : est le coût total estimé pour aller d'un état initial vers un état final en passant par n.

Algorithme A*

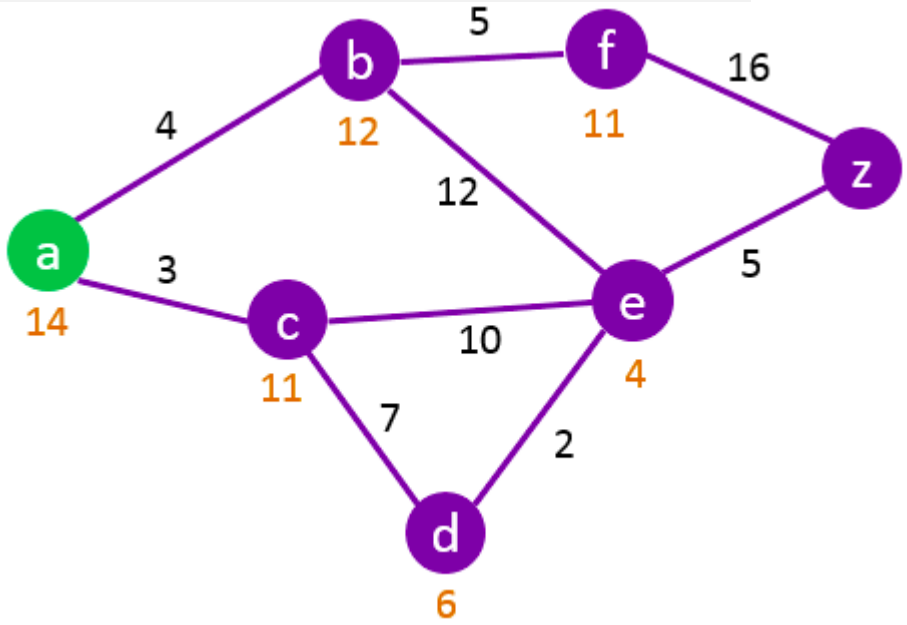
Exemple : Chercher le minimal chemin de **a** à **z**



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A					
B					
C					
D					
E					
F					
Z					

Algorithme A*

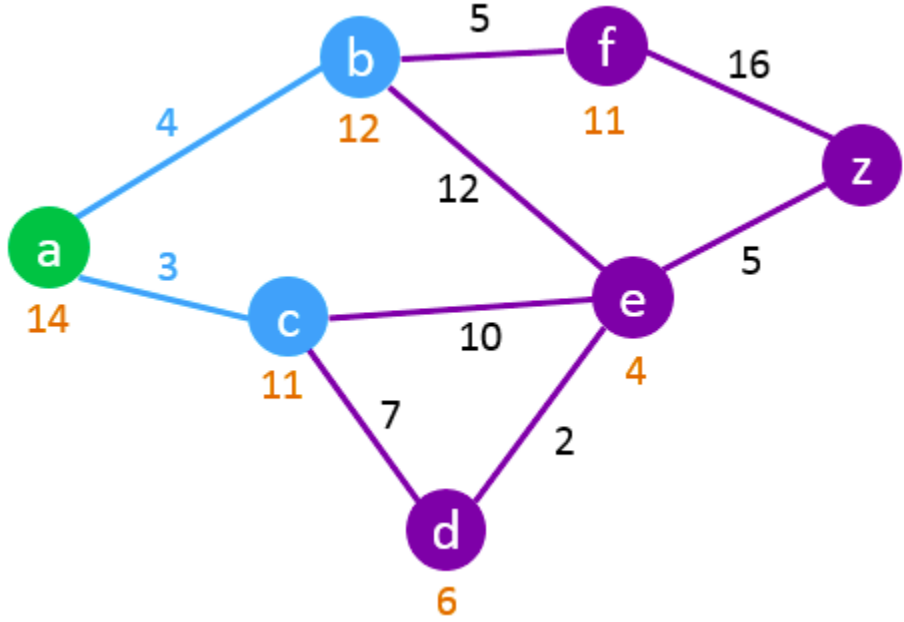
Etape 1



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Open	0	14	14	
B		∞	12		
C		∞	11		
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

Algorithme A*

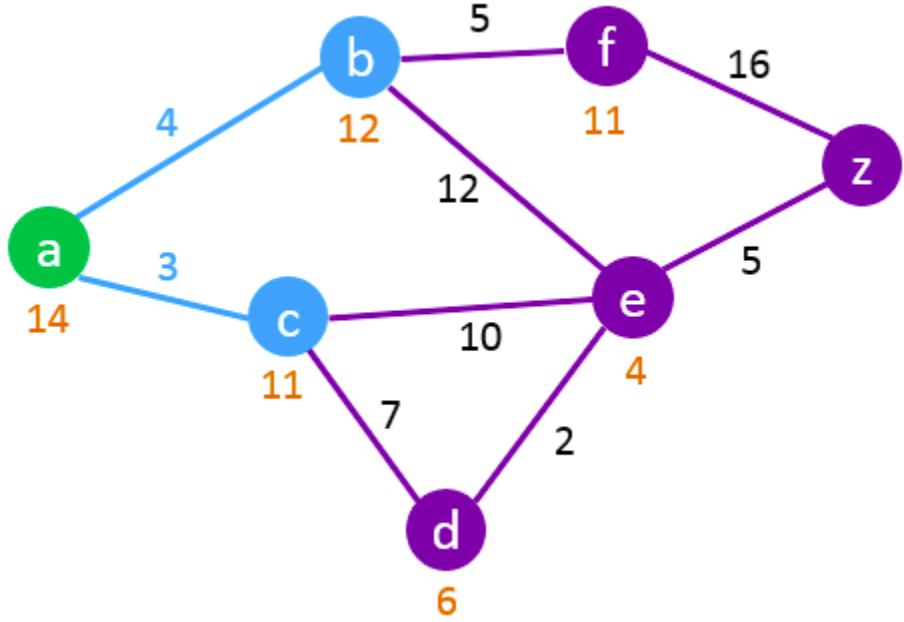
Etape 3



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Open	0	14	14	
B		∞ → 4	12		
C		∞ → 3	11		
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

Algorithme A*

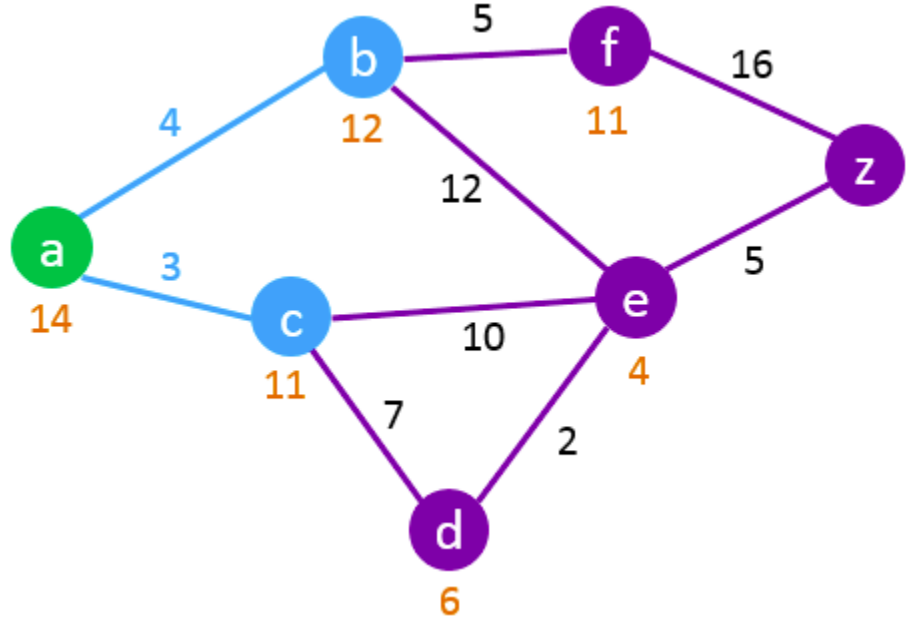
Etape 3



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Open	0	14	14	
B		∞ → 4	12	16	A
C		∞ → 3	11	14	A
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

Algorithme A*

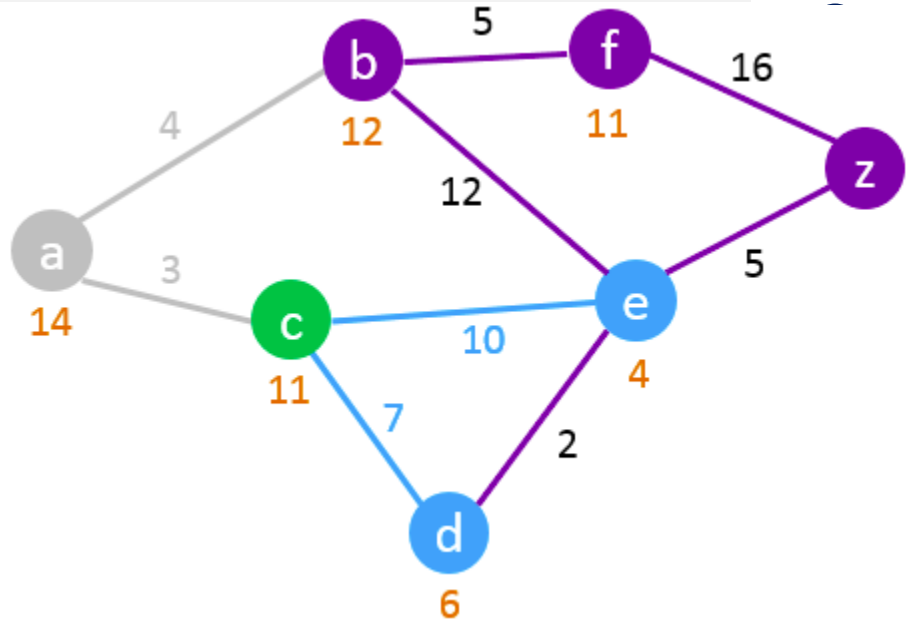
Etape 4



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Closed	0	14	14	
B	Open	∞ → 4	12	16	A
C	Open	∞ → 3	11	14	A
D		∞	6		
E		∞	4		
F		∞	11		
Z		∞	0		

Algorithme A*

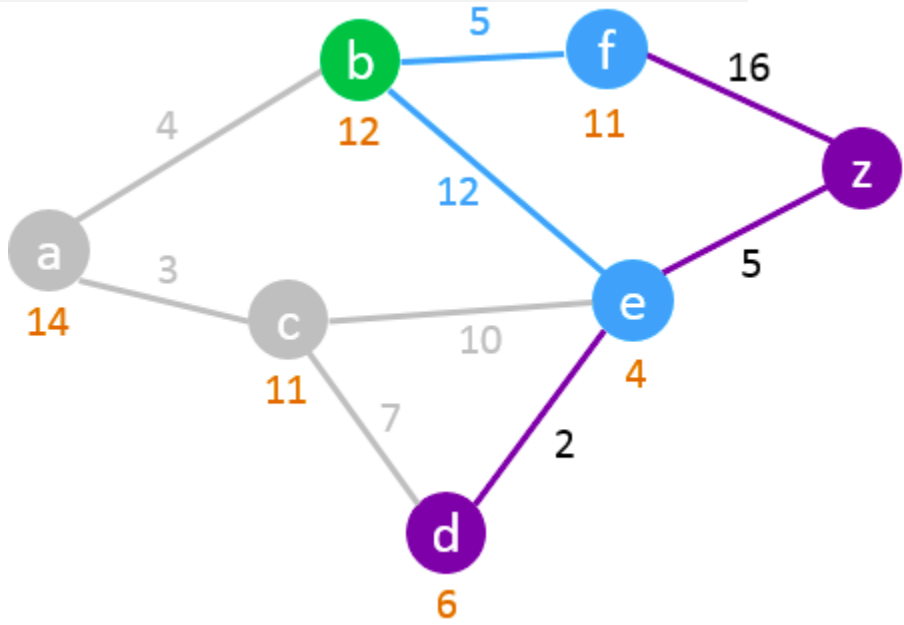
Etape 5



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	**Closed**	0	14	14	
B	Open	4	12	16	A
C	**Closed**	3	11	14	A
D	Open	∞ → 10	6	16	C
E	Open	∞ → 13	4	17	C
F		∞	11		
Z		∞	0		

Algorithme A*

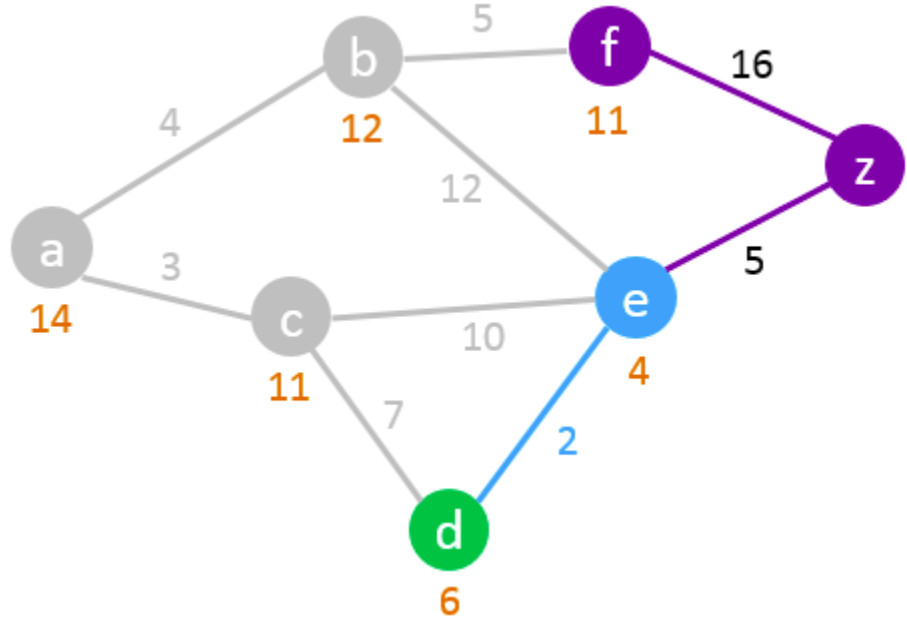
Etape 6



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Closed	0	14	14	
B	Closed	4	12	16	A
C	Closed	3	11	14	A
D	Open	10	6	16	C
E	Open	13	4	17	C
F	Open	9	11	20	B
Z		∞	0		

Algorithme A*

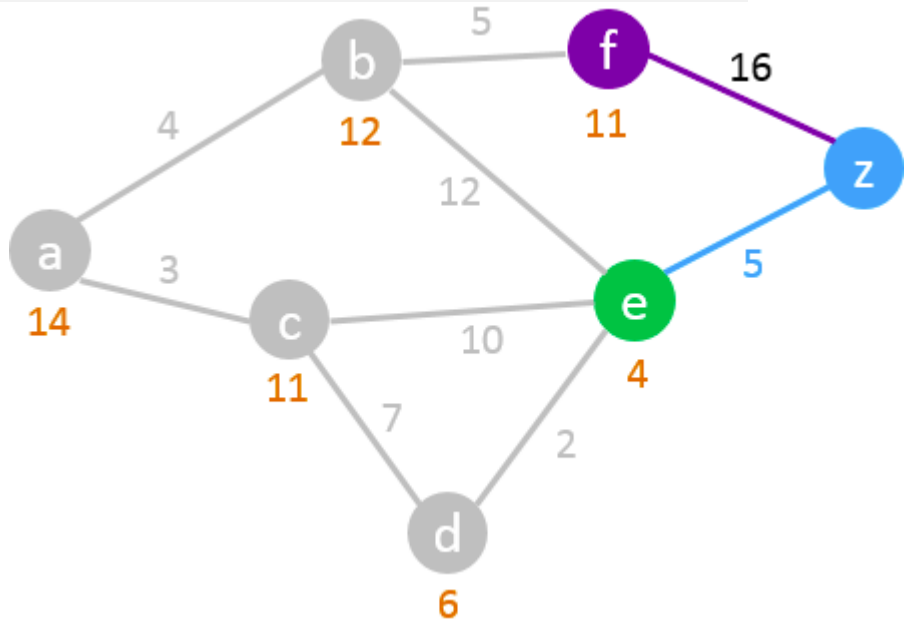
Etape 7



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	**Closed**	0	14	14	
B	**Closed**	4	12	16	A
C	**Closed**	3	11	14	A
D	**Closed**	10	6	16	C
E	Open	13 → 12	4	17	D
F	Open	9	11	20	B
Z		∞	0		

Algorithme A*

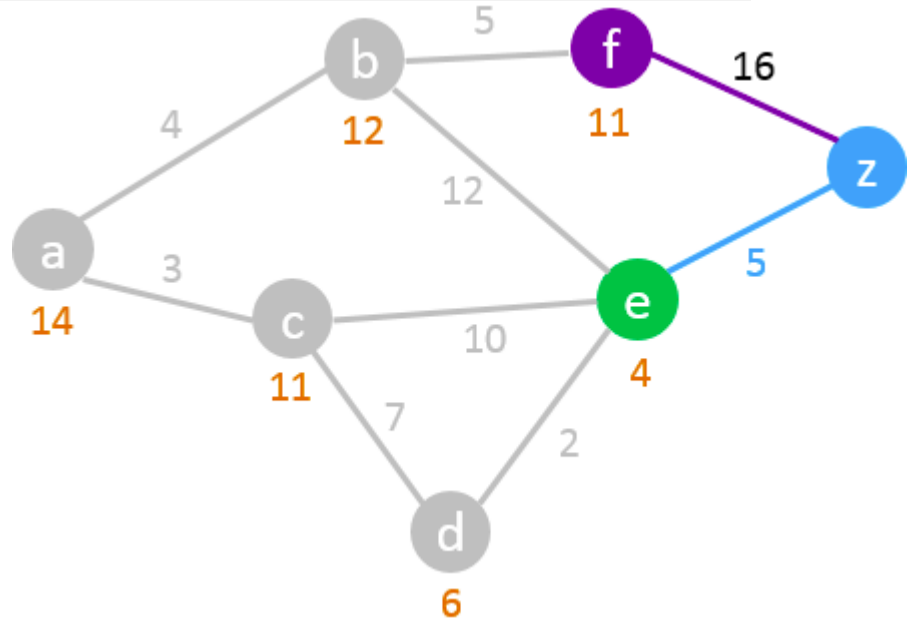
Etape 8



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Closed	0	14	14	
B	Closed	4	12	16	A
C	Closed	3	11	14	A
D	Closed	10	6	16	C
E	Closed	12	4	17	D
F	Open	9	11	20	B
Z	Open	∞ → 17	0	17	E

Algorithme A*

Etape 9



Nœud n	Statut	g(n)	h(n)	f(n)	Précédent
A	Closed	0	14	14	
B	Closed	4	12	16	A
C	Closed	3	11	14	A
D	Closed	10	6	16	C
E	Closed	12	4	17	D
F	Open	9	11	20	B
Z	Open	17	0	17	E

Coût : 17
Chemin : A-C-D-E-Z

Algorithme A*

Pseudo code:

Entrées:

Sommet source (S)

Sommet destination (D)

Initialisation

Liste des sommets à explorer (**open**) : sommet source S

Liste des sommets visités (**closed**) : vide

Tant que (la liste open est non vide) et (D n'est pas dans open) Faire

+ Récupérer le sommet X de coût total F minimum.

+ Ajouter X à la liste **closed**

+ Ajouter les successeurs de X (non déjà visités) à la liste **open** en évaluant leur coût total f et en identifiant leur prédécesseur.

+ Si (un successeur est déjà présent dans **open**) et (nouveau coût est inférieur à l'ancien) Alors

Changer son coût total

Changer son prédécesseur

FinSi

FinFaire

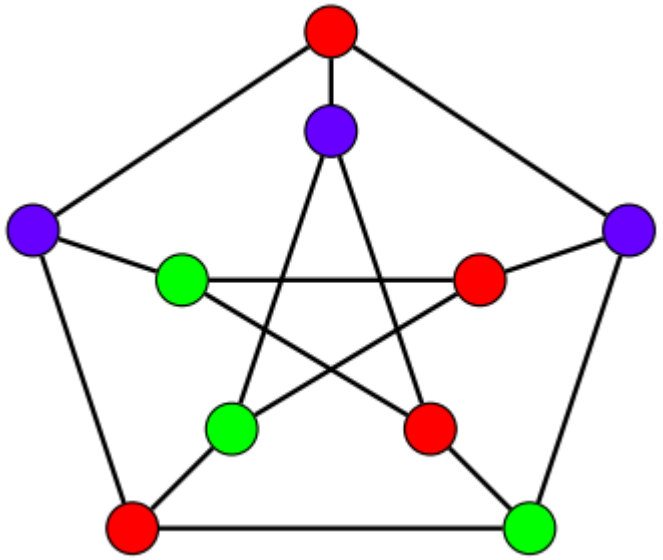
Algorithme A* : Propriétés

Le choix de l'heuristique d'estimation est crucial :

- Sur-estimation → Risque de ne pas envoyer le chemin optimal
- Sous-estimation → la complexité de l'algorithme augmente
- Heuristique nulle → algorithme de Dijkstra

Algorithme Coloration de graphe

Colorer un réseau signifie attribuer une couleur à chacun de ses nœuds de manière que deux nœuds voisins soient de couleur différente.



Une coloration du graphe de Petersen avec 3 couleurs

Algorithme Coloration de graphe

Applications :

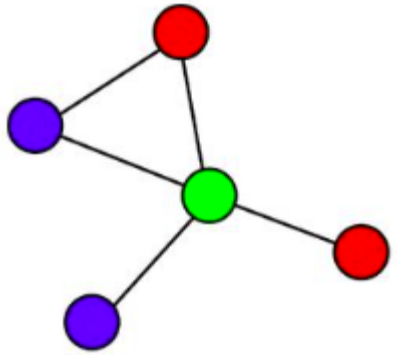
La coloration de sommet a de nombreuse applications pratiques. Par exemple :

- Résoudre des problèmes d'organisation (Session minimale à programmer).
- Allocation de fréquences dans les réseaux GSM
- Coloration de carte géographique
- ….

Algorithme Coloration de graphe

Algorithmes :

Quel algorithme permet de colorier un graphe ?



Deux algorithmes principaux :

1. Algorithme naïf (glouton), non optimal mais rapide
2. Welsh et Powell

Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

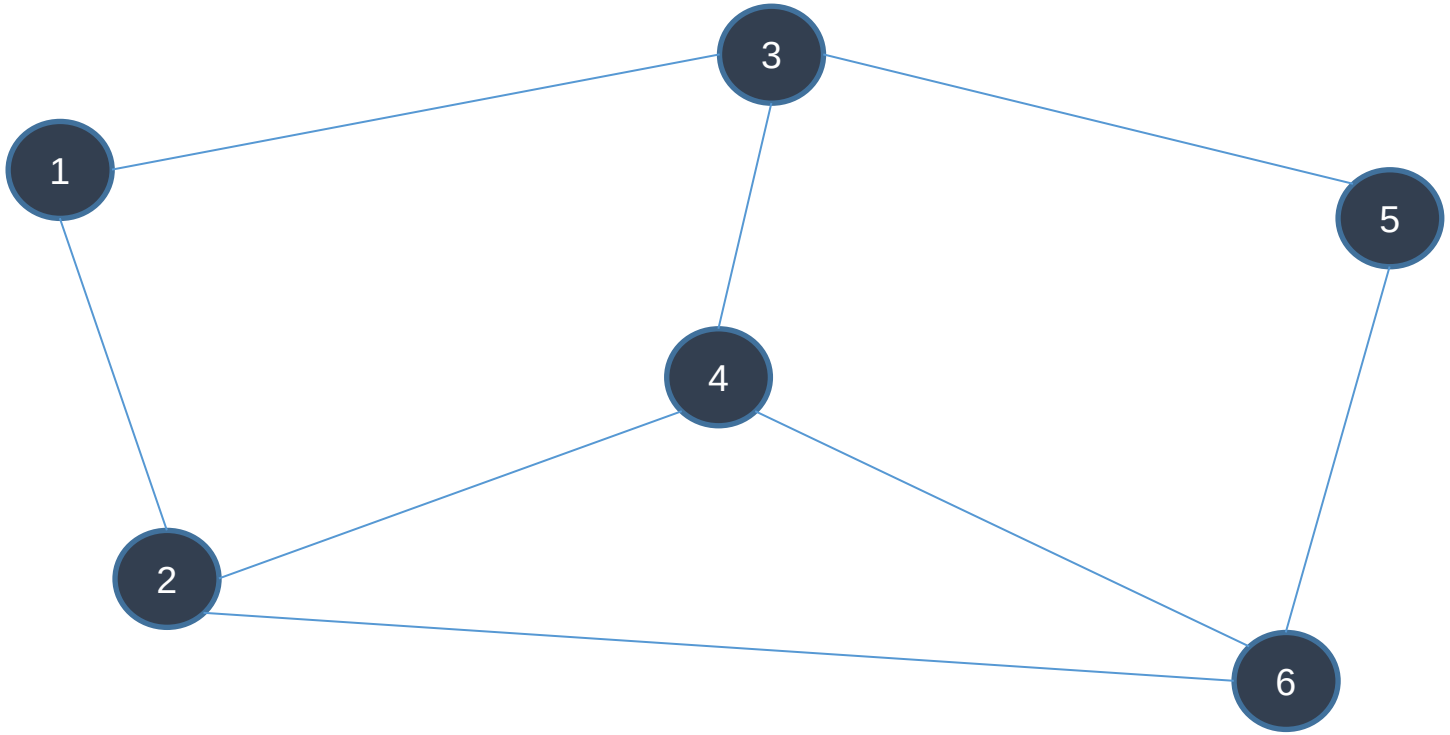
Algorithm 1: Algorithme glouton

tant que *il existe un noeud non-coloré v* **faire**

    Colorié *v* avec la couleur minimale qui n'entre pas en conflit avec ces voisins

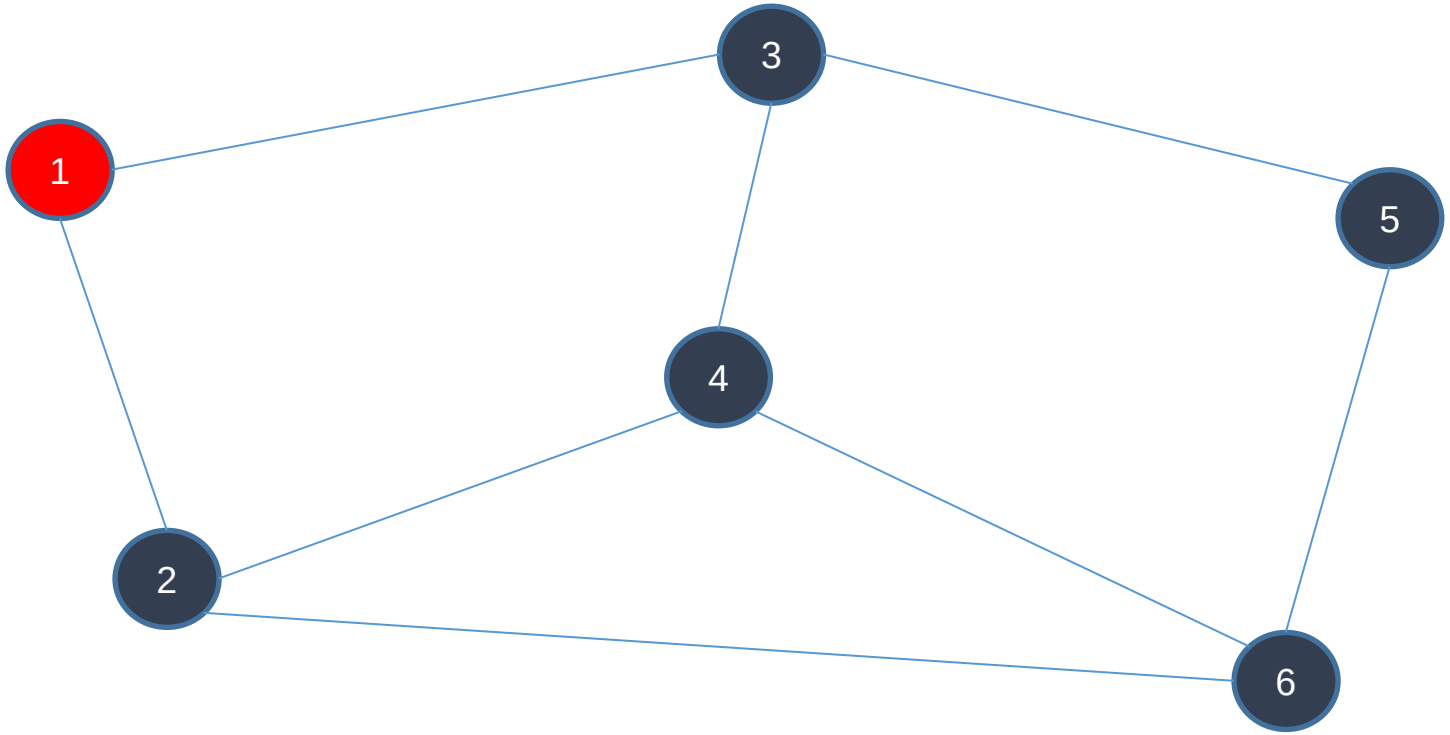
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



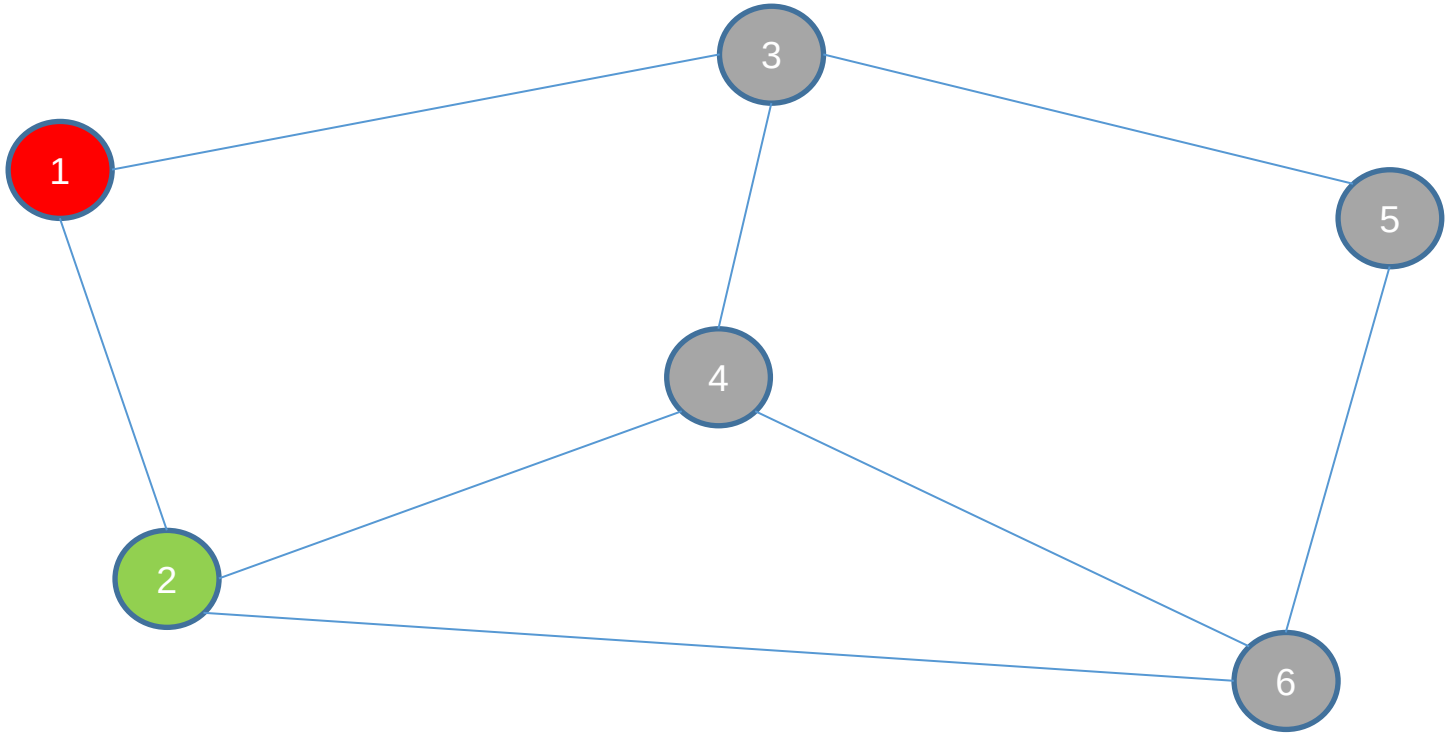
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



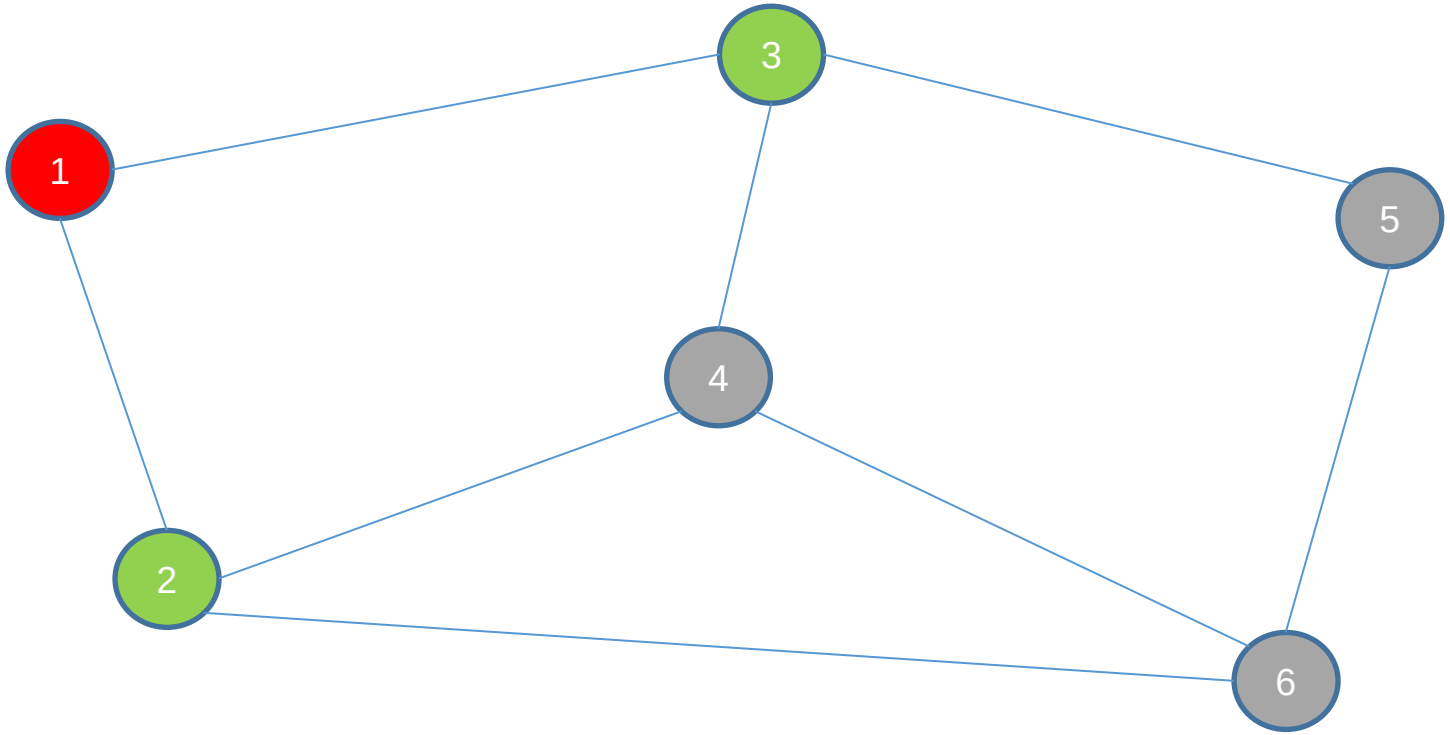
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



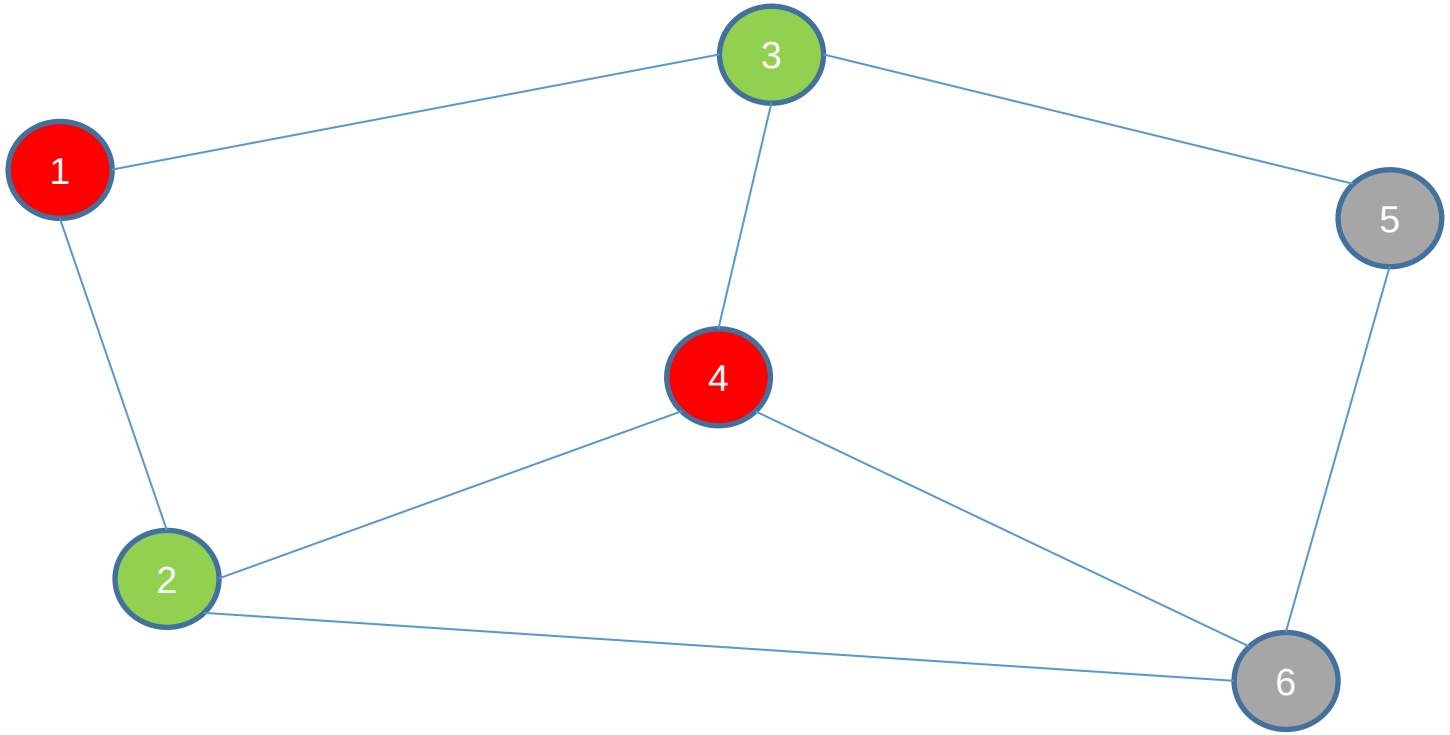
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



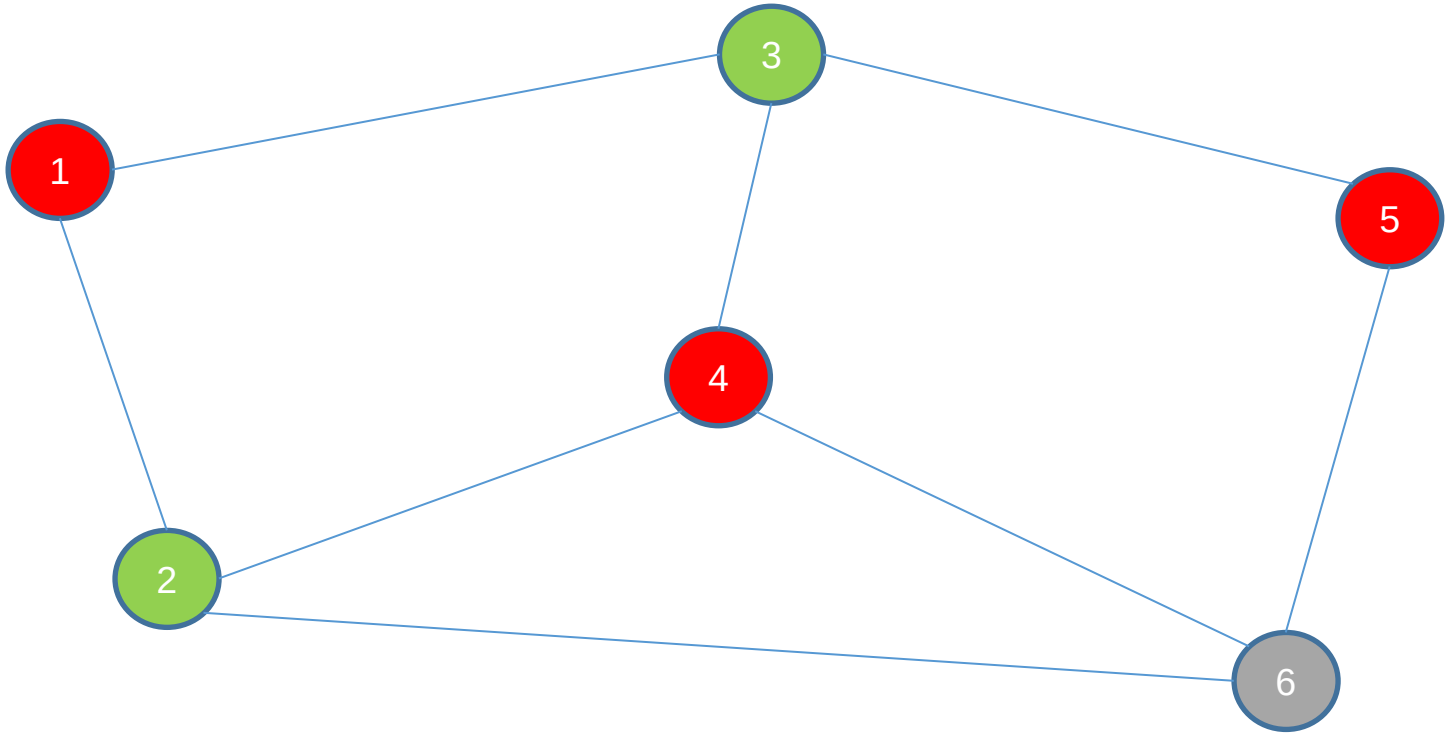
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



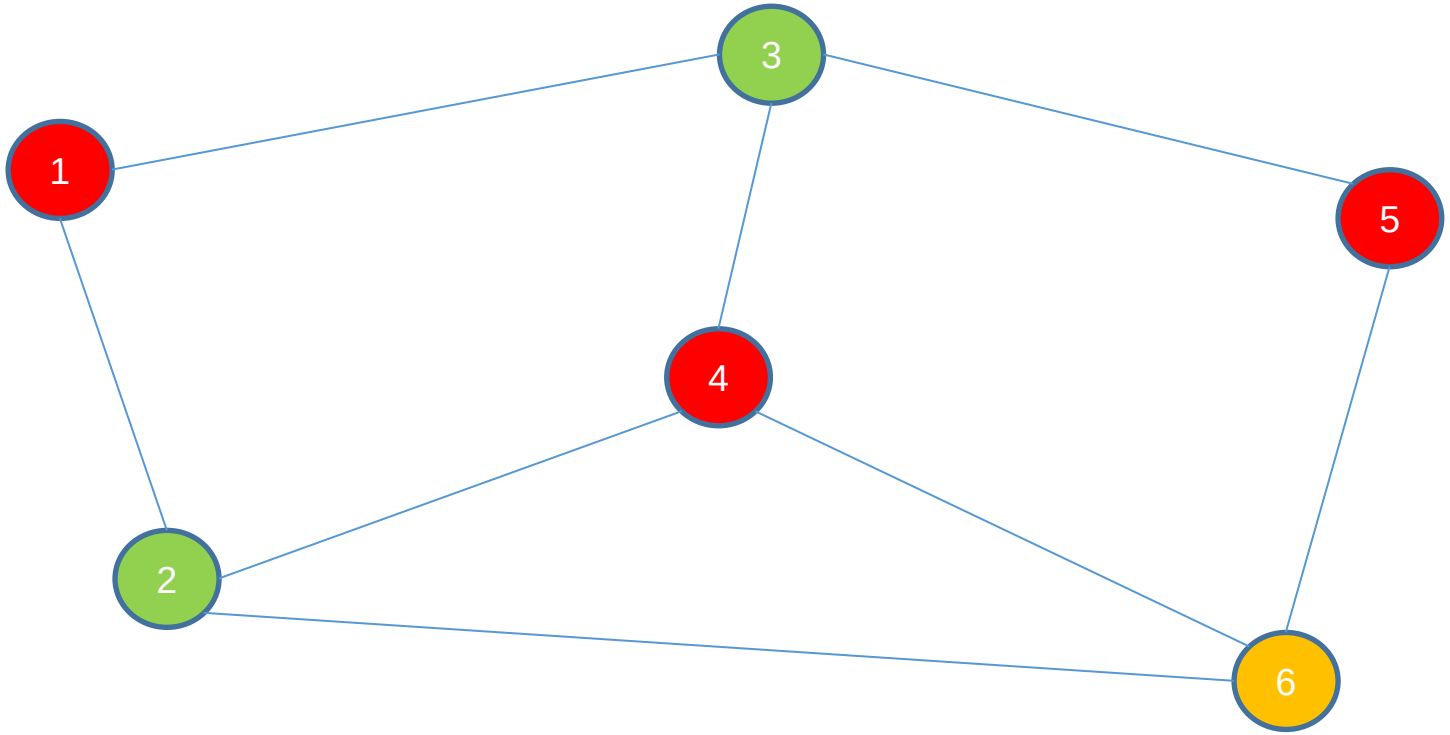
Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle



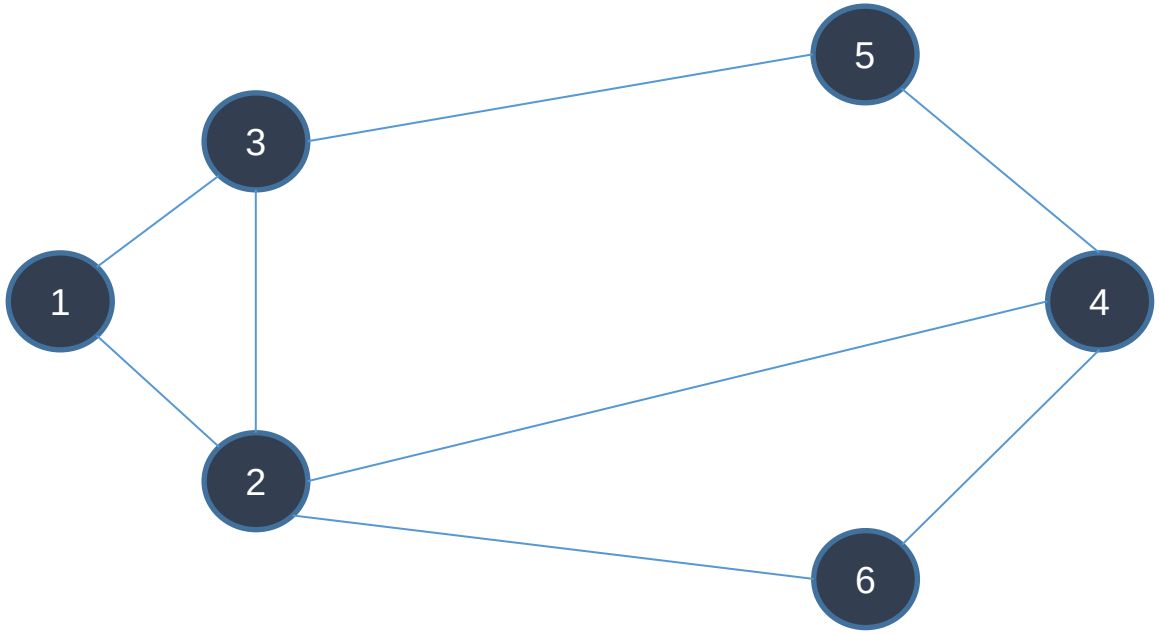
Nombre chromatique : 3

Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

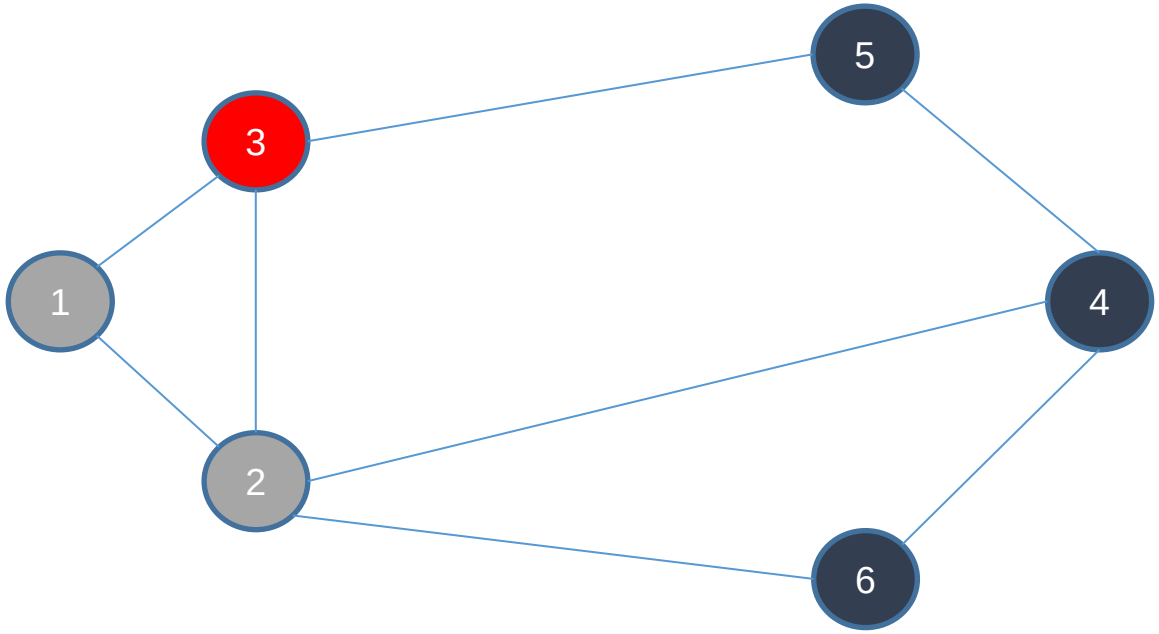


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

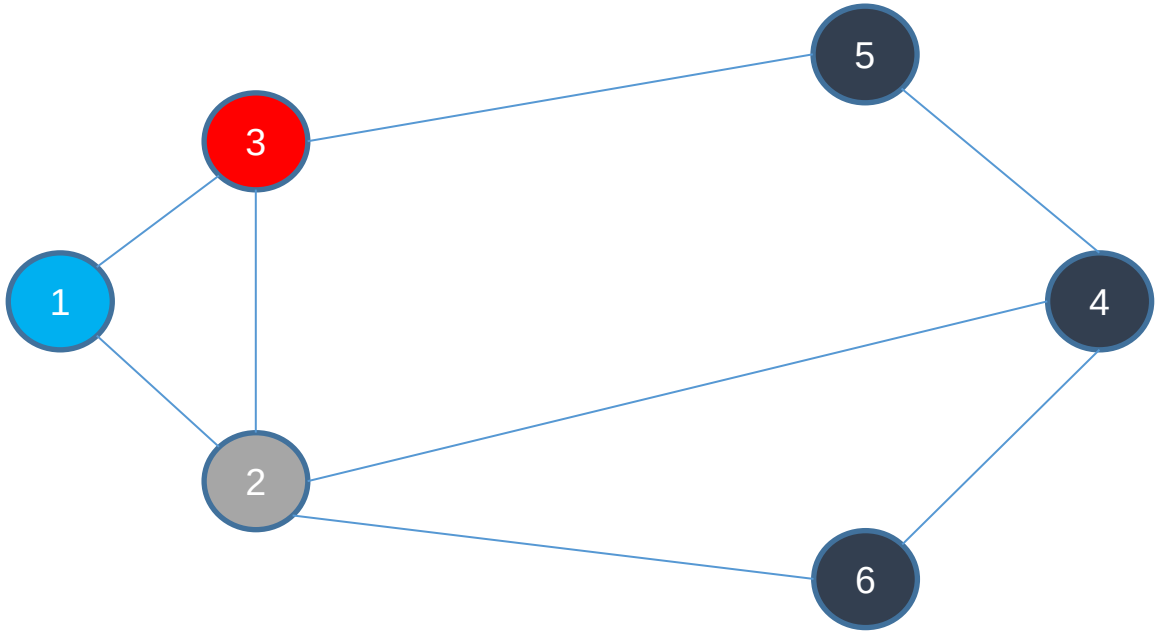


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

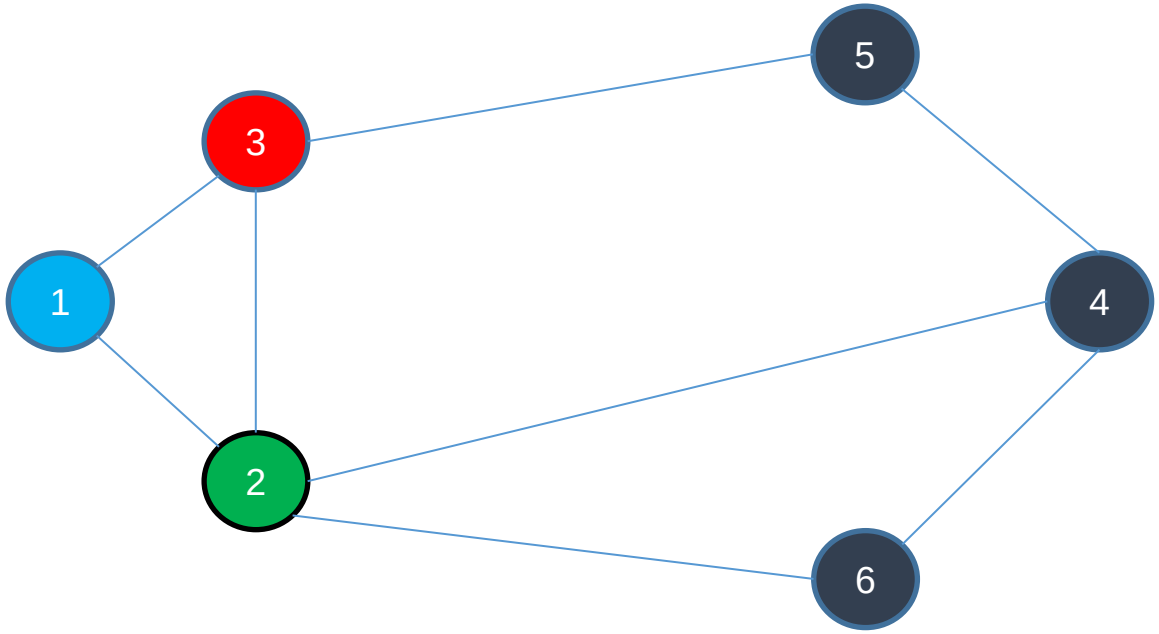


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

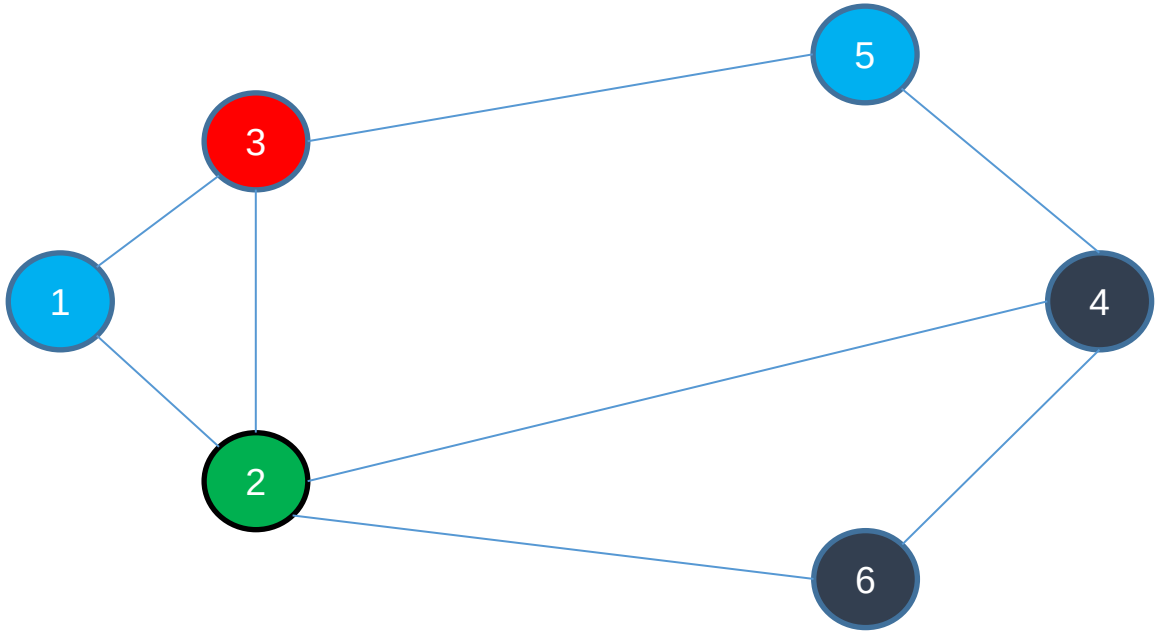


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

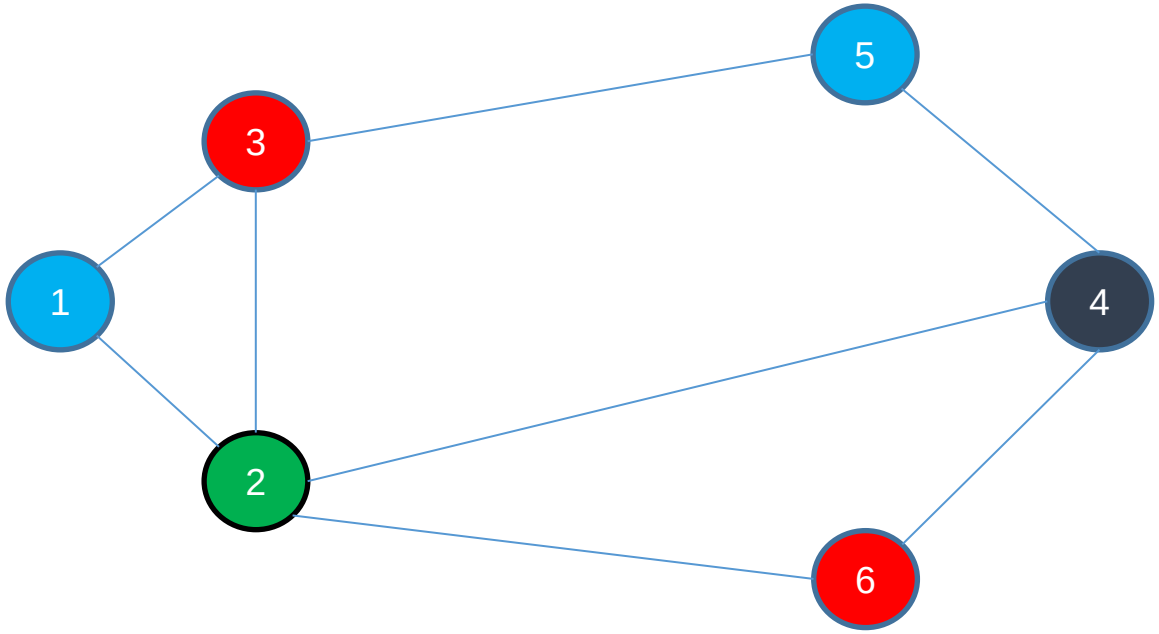


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:

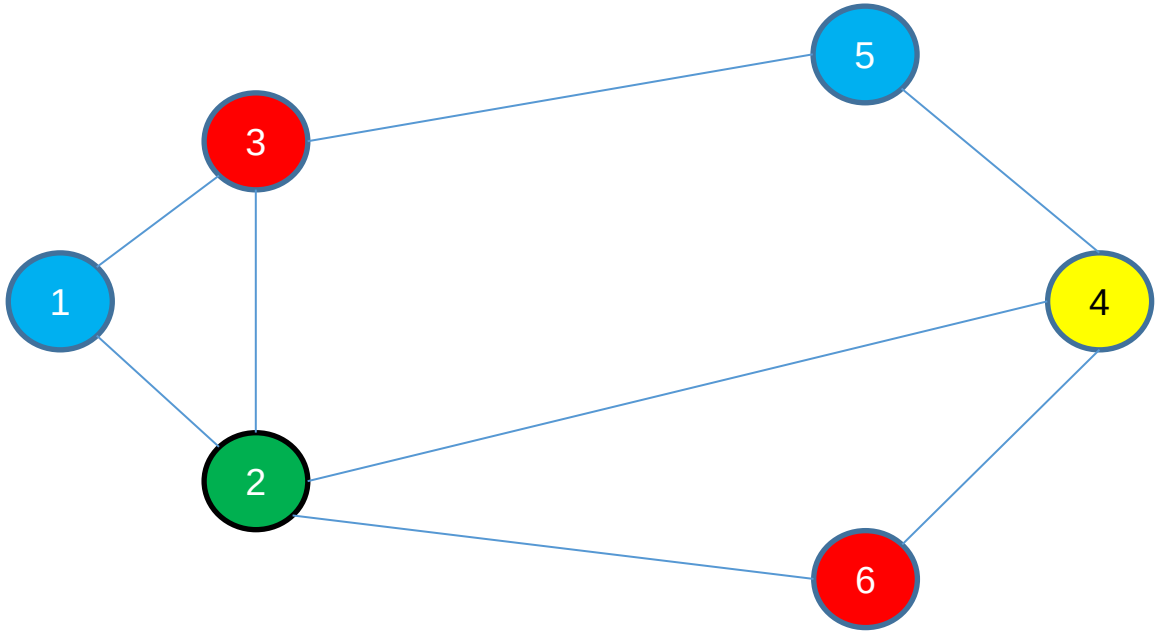


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 1:



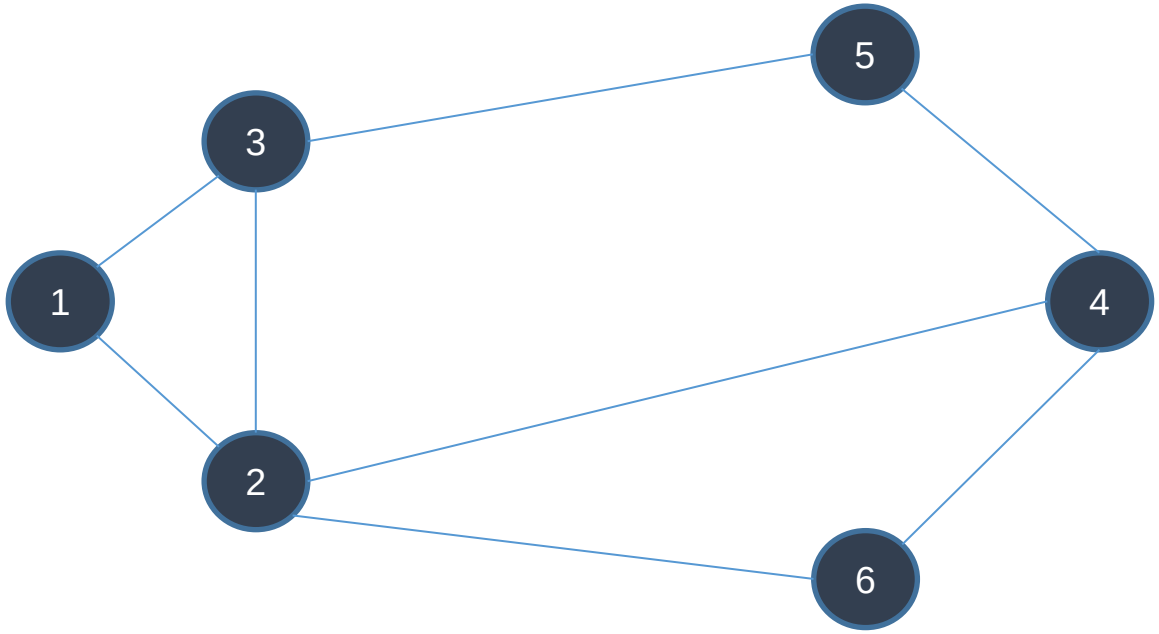
4 couleurs

Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

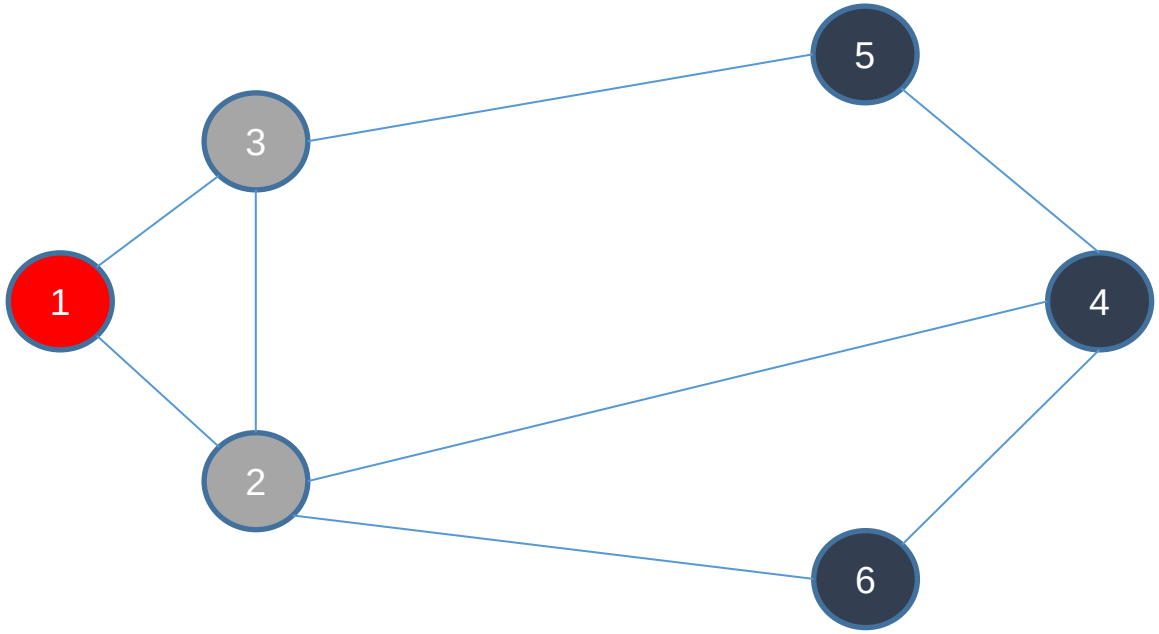


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

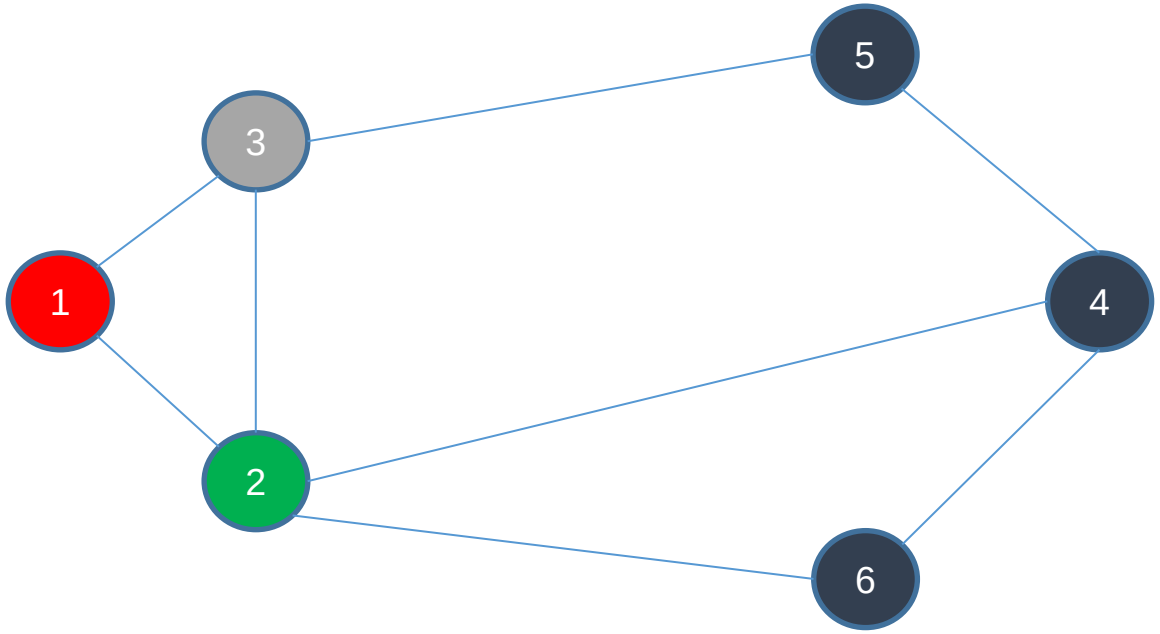


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

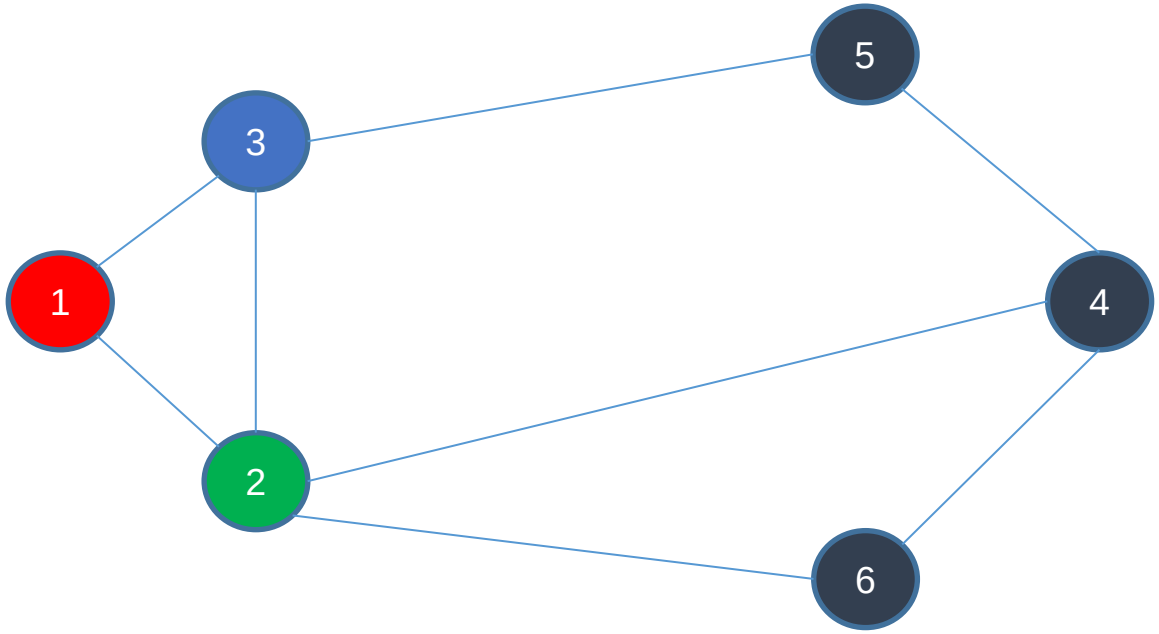


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

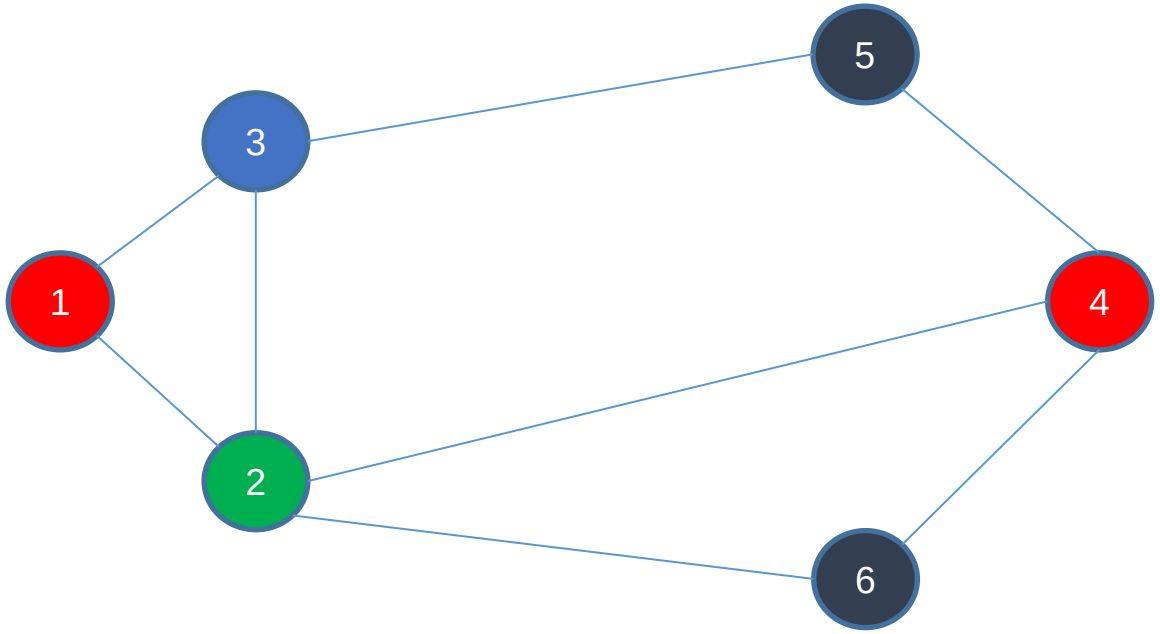


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

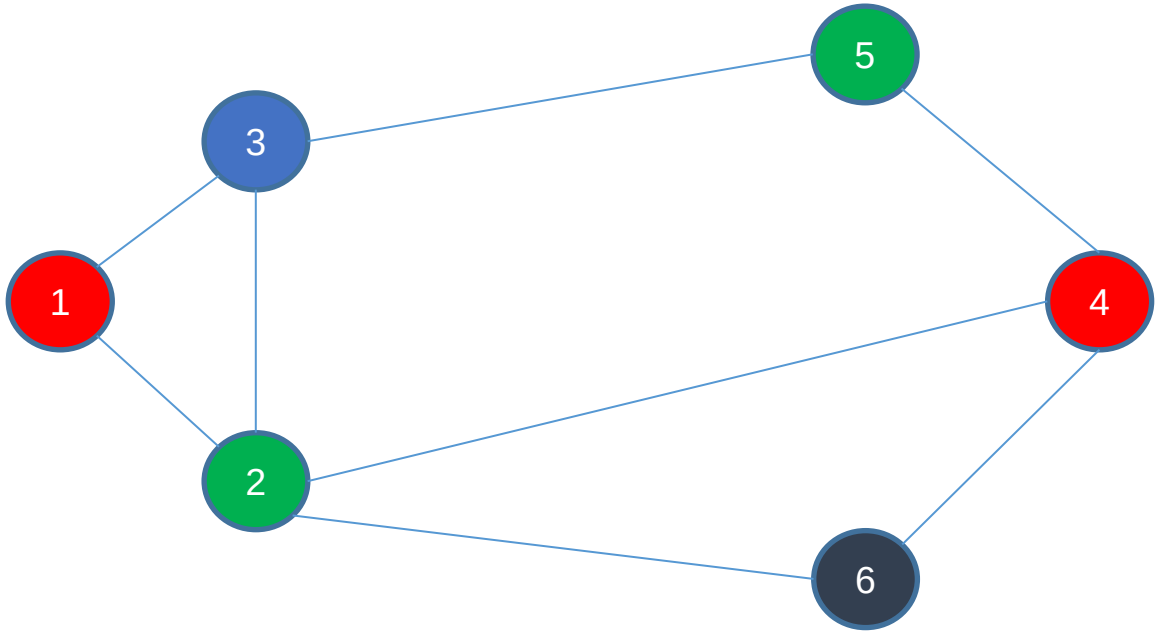


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:

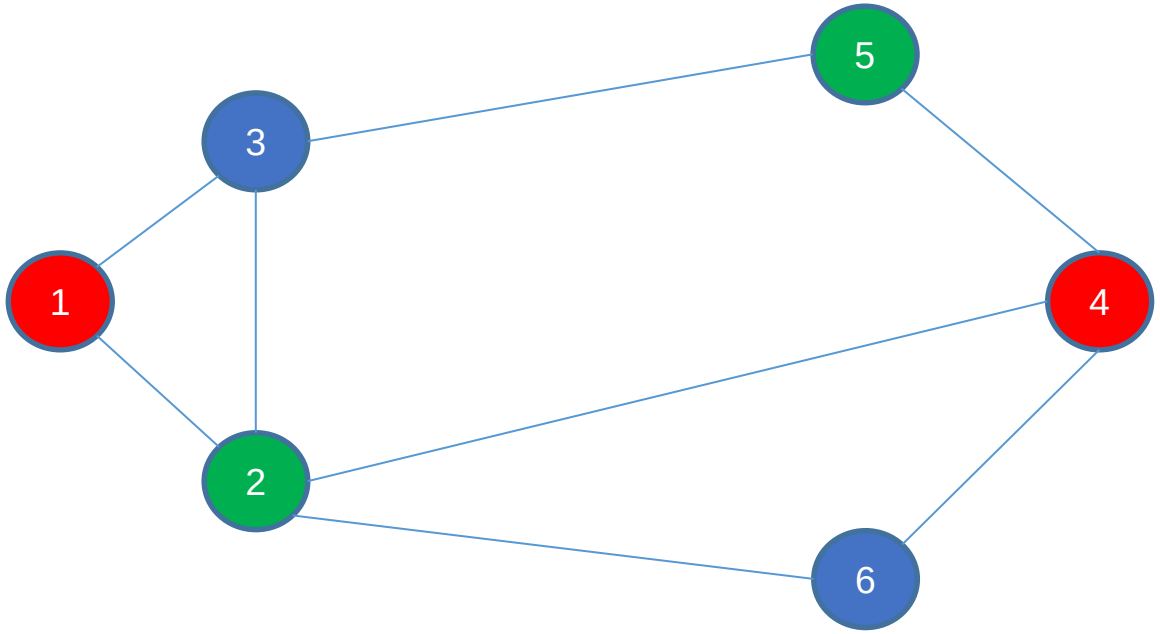


Algorithme Coloration de graphe

Algorithme 1 : Coloration séquentielle

Problème : Le nombre de couleur dépend du sommet de départ, de la manière de parcourt, …

Exemple 2:



3 couleurs

Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell :

L’algorithme de Welsh & Powell consiste à colorer séquentiellement le graphe en visitant les sommets par ordre de degré décroissant. L’idée est que les sommets ayant beaucoup de voisins seront plus difficiles à colorer, et donc il faut les colorer en premier.

Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell :

1.Repérer le degré de chaque sommet.

2.Ranger les sommets par ordre de degrés décroissants. (dans certains cas plusieurs possibilités)

3.Attribuer au premier sommet (A) de la liste une couleur.

4.Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).

5.Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.

6.Continuer jusqu'à ce que la liste soit finie.

7.Prendre une deuxième couleur pour le premier sommet (D) non encore colorié de la liste.

8.Répéter les opérations 4 à 6.

9.Continuer jusqu'à avoir coloré tous les sommets.

Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell :

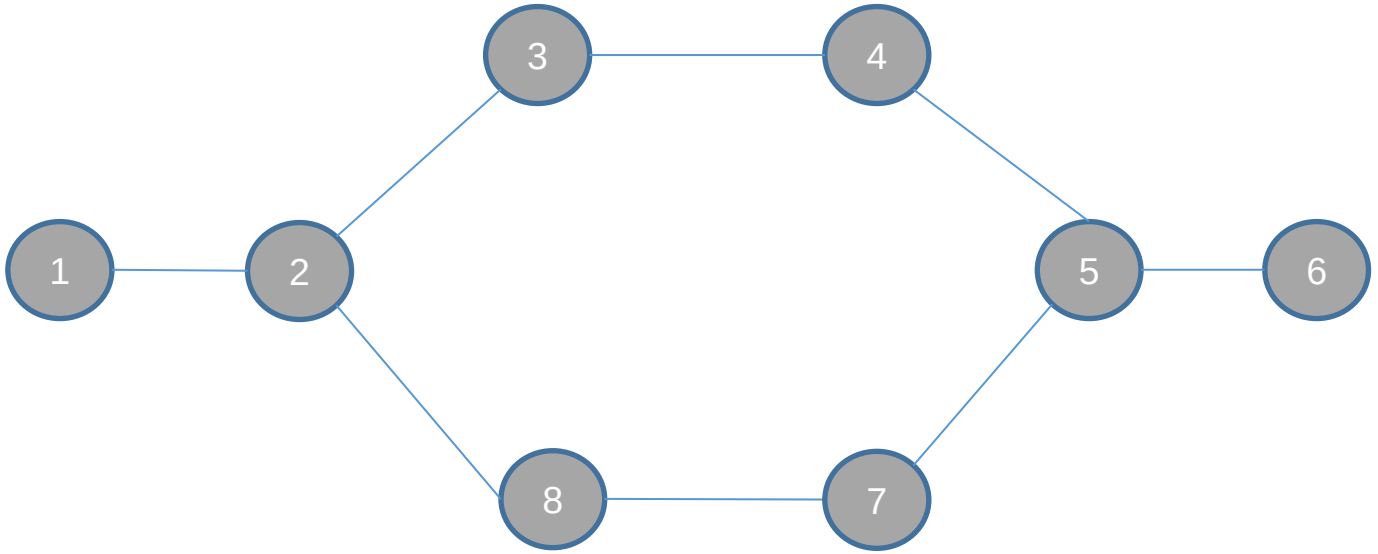
On note L la liste des sommets si classes suivant l'ordre décroissant de leur degré :
(s1) d(s2) d(s3) ::: d(sn).
Initialisation :
L : liste des sommets dans l'ordre décroissant du degré
couleur = 0
Tant que L ≠ ∅ ; faire
    couleur=couleur+1
    couleur(s) = couleur
    V = voisins(s)
    Pour tout t dans L faire
        Si t ∉ V
            couleur(t)=couleur
            V = V ∪ t
        Fin si
    Fin faire
    Retirer les sommets colores de L
Fin faire

Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell : Contre exemple

Liste sommets triés par degrés décroissants :

2
5
4
3
7
8
6
1

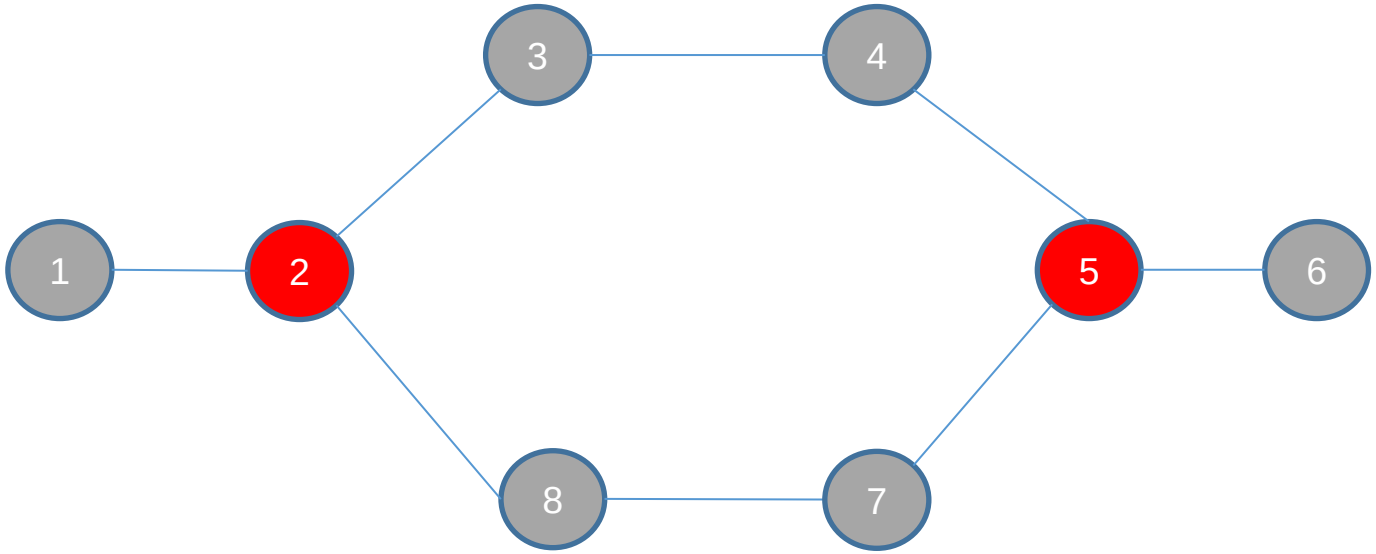


Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell : Contre exemple

Liste sommets triés par degrés décroissants :

4
3
7
8
6
1



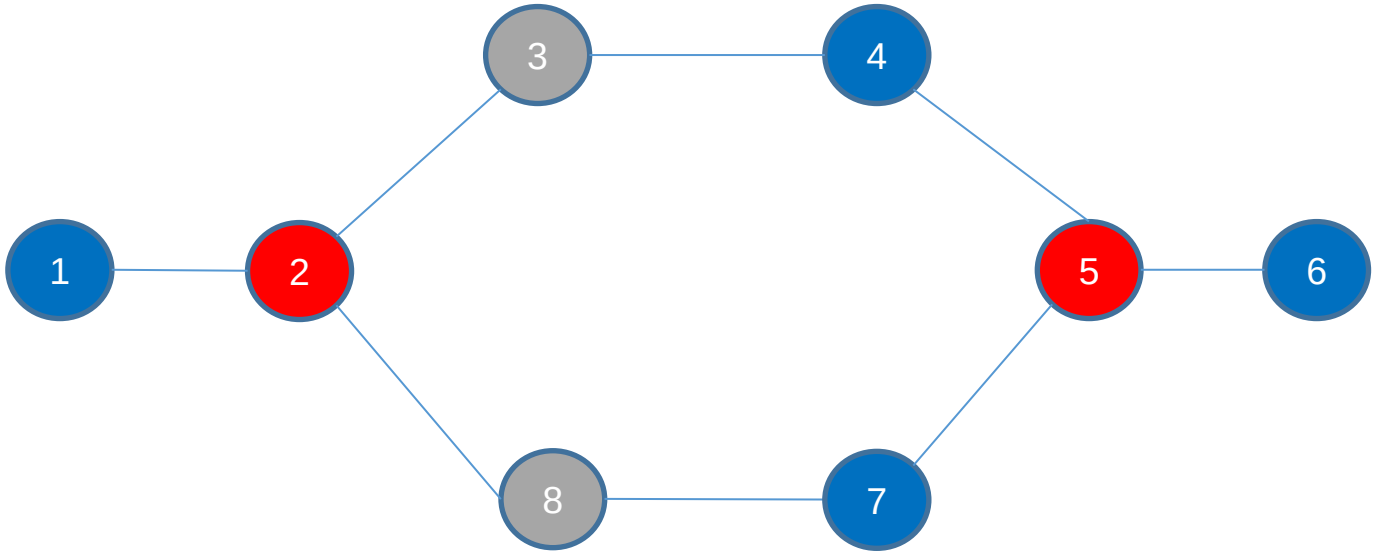
Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell : Contre exemple

Liste sommets triés par degrés décroissants :

3

8



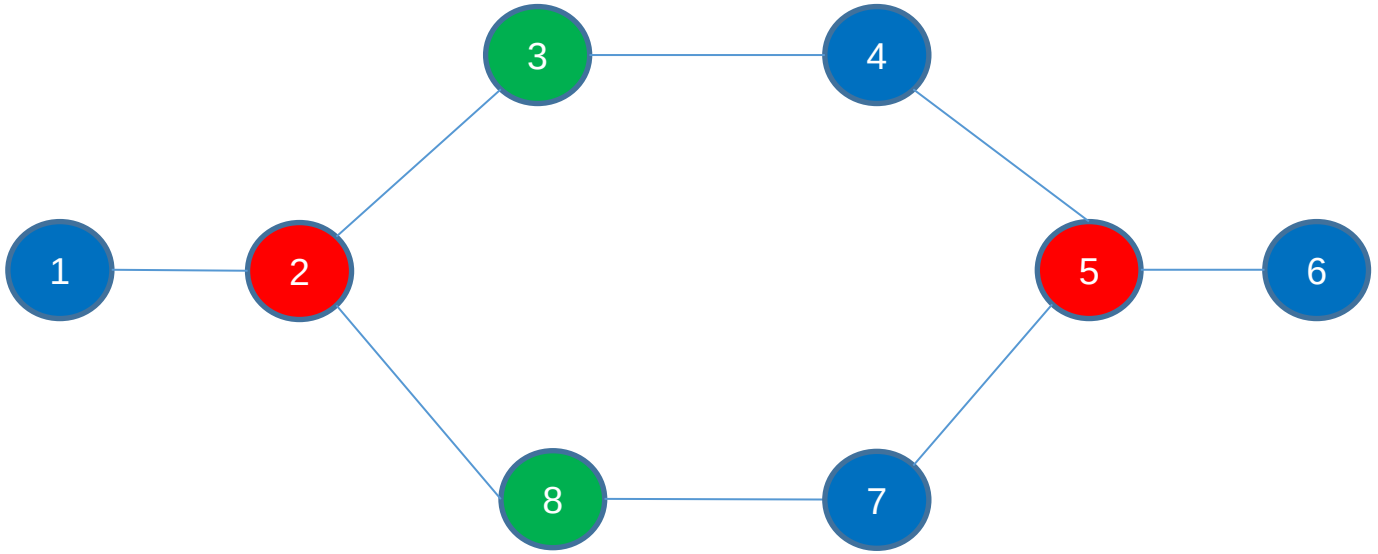
Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell : Contre exemple

Liste sommets triés par degrés décroissants :

3

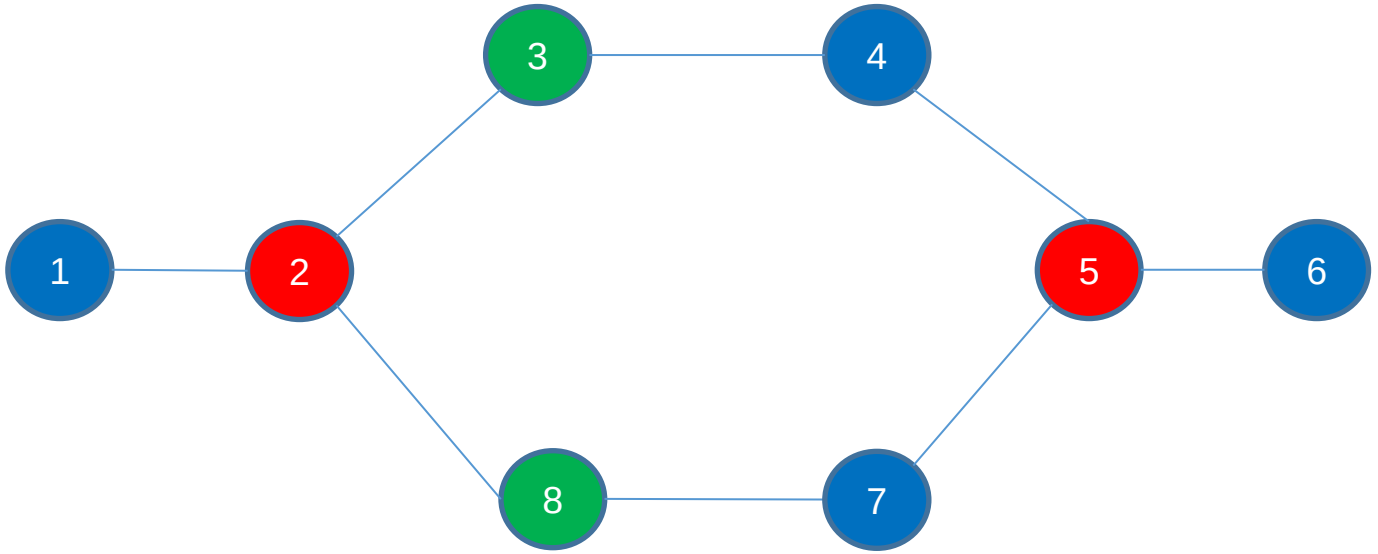
8



Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell: Contre exemple

Nombre chromatique : 3



Algorithme Coloration de graphe

Algorithme 2: de Welsh et Powell: Contre exemple

Alors que il existe une autre solution :
Nombre chromatique : 2