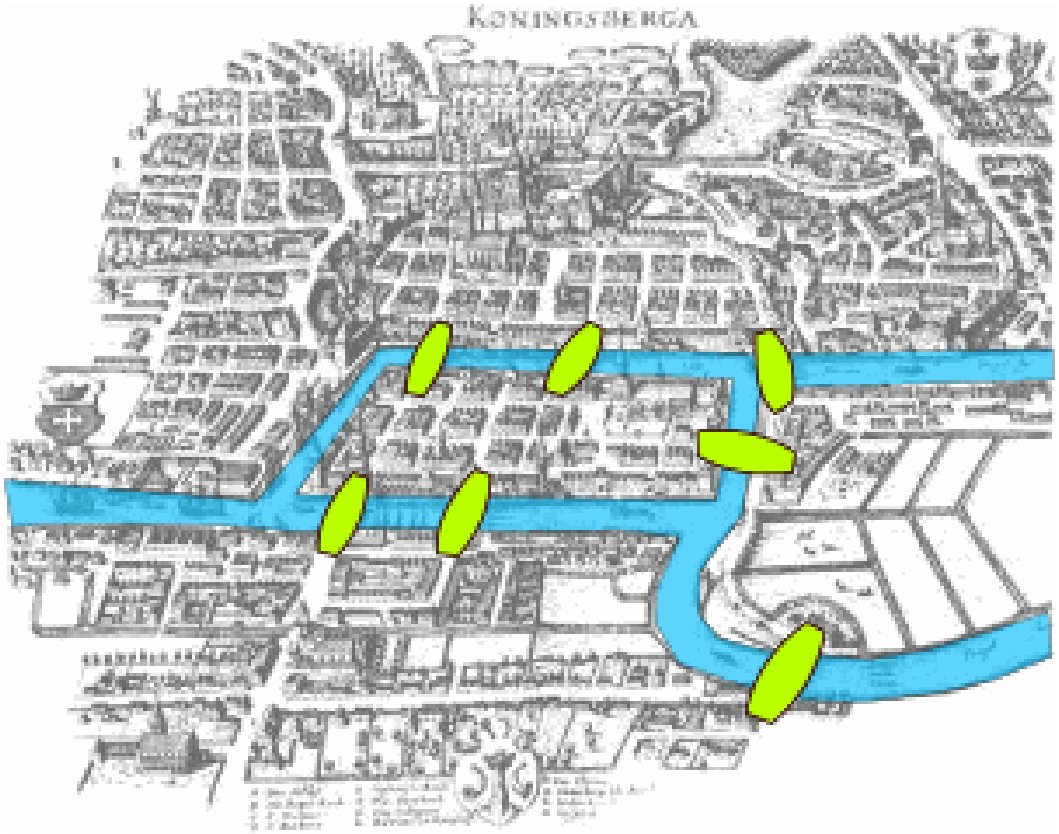
Les Graphes

Plan

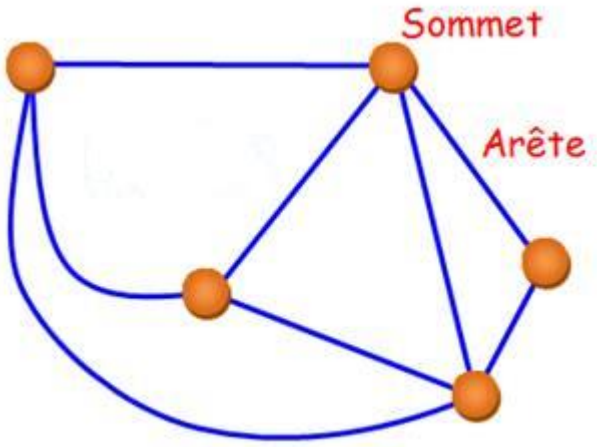
- Définitions et terminologies
- Implémentation des graphes
- Algorithmes de parcours d’un graphe
- Algorithme de Dijkstra

Ponts de Konigsberg



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.



Exemples des situations modélisées par un graphe

- Transport :
  - Carte géographique : Recherche du chemin le plus court entre deux villes.
  - Un réseau ferroviaire : chaque gare est un sommet, les voies entre deux gares sont les arêtes. Idem avec un réseau routier.
  - Les lignes aériennes
- Informatique :
  - 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.

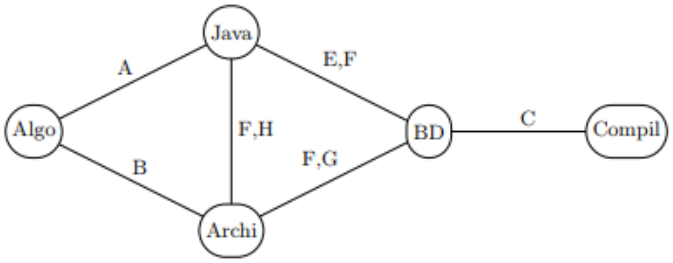
– Java : étudiants A, E, F et H.

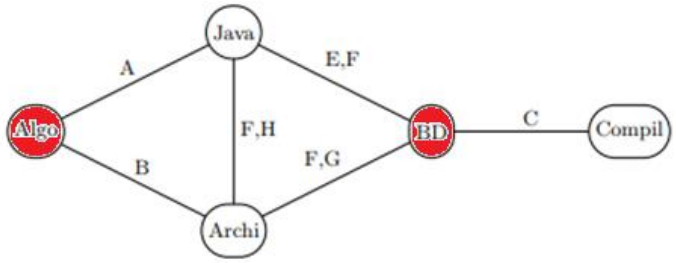
– Architecture : étudiants B, F, G et H.

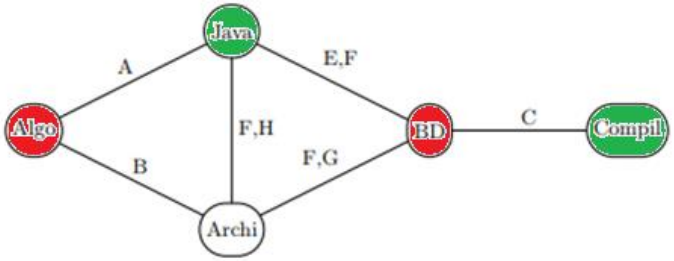
On cherche à organiser la session d’examen la plus courte possible.

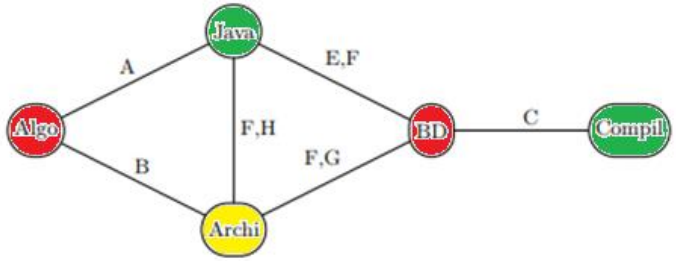
Exemple d'application 1

Solution :









Exemple d’application 1

Solution :

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

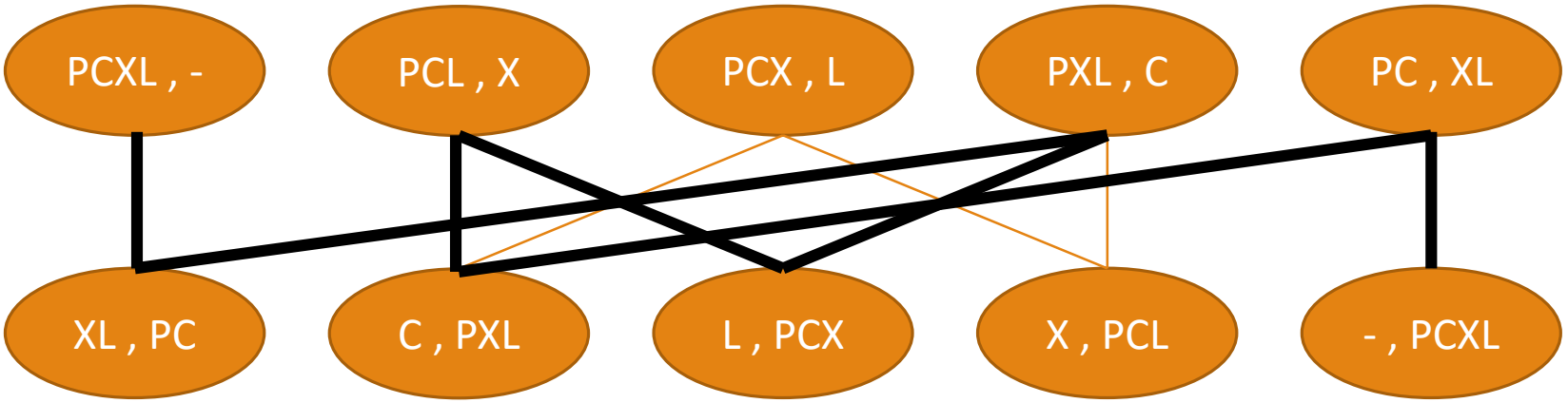
Exemple d’application 2

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

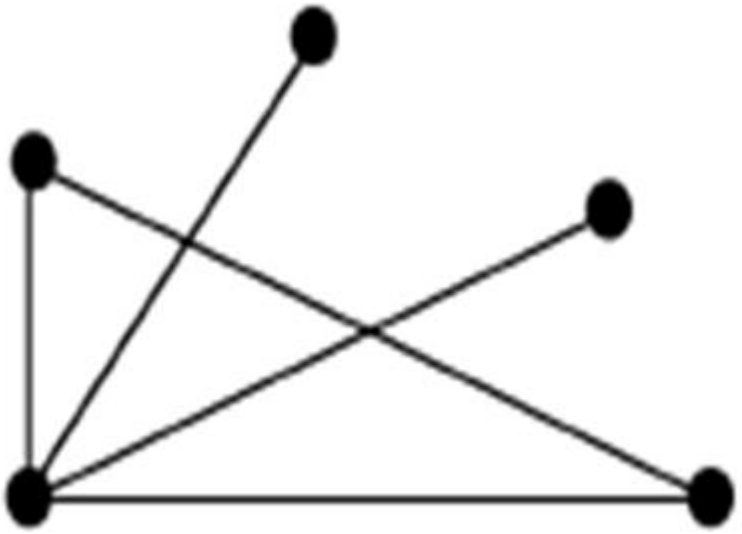
Solution :

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.

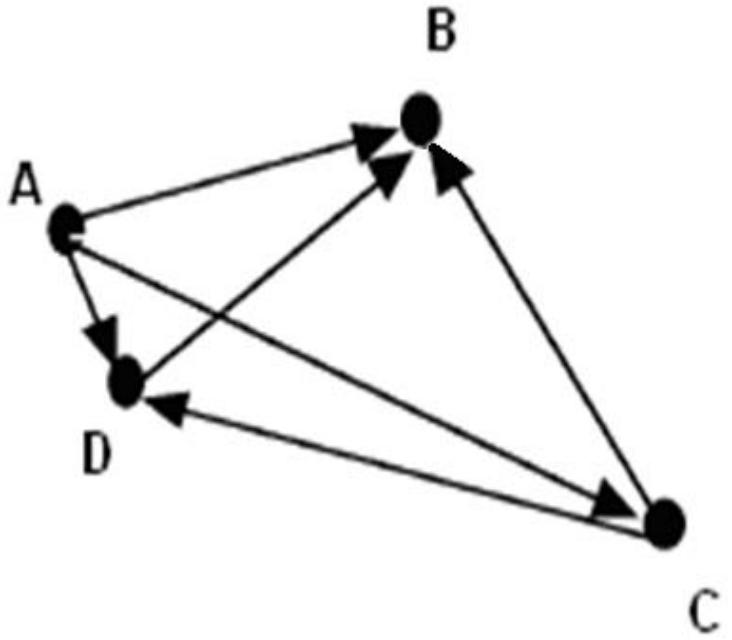


Définitions et terminologies

Graphe non orienté :

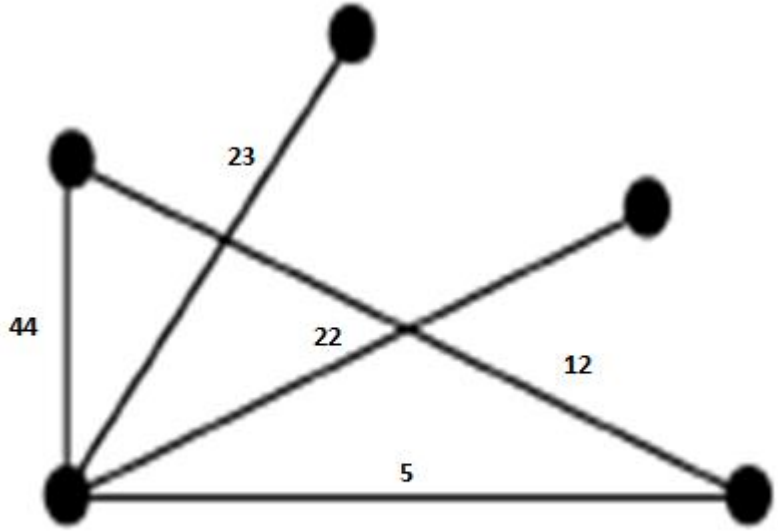


Graphe orienté :

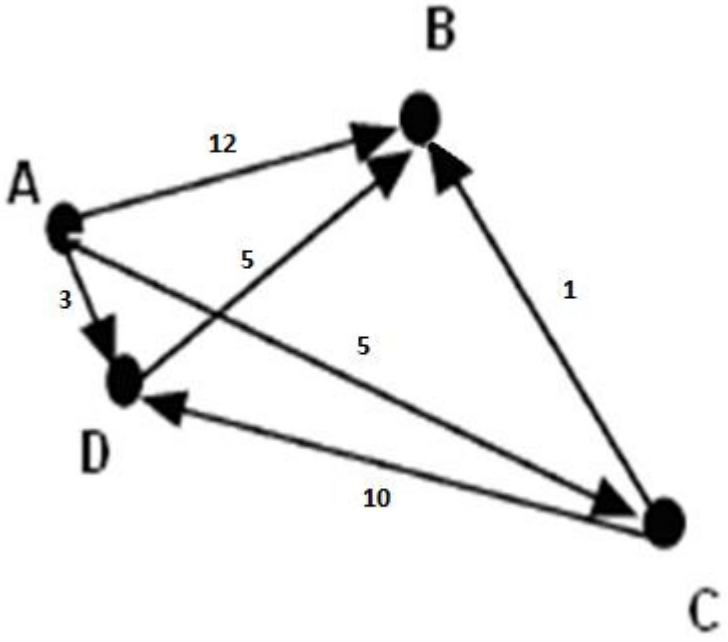


Définitions et terminologies

Graphe non orienté pondéré :

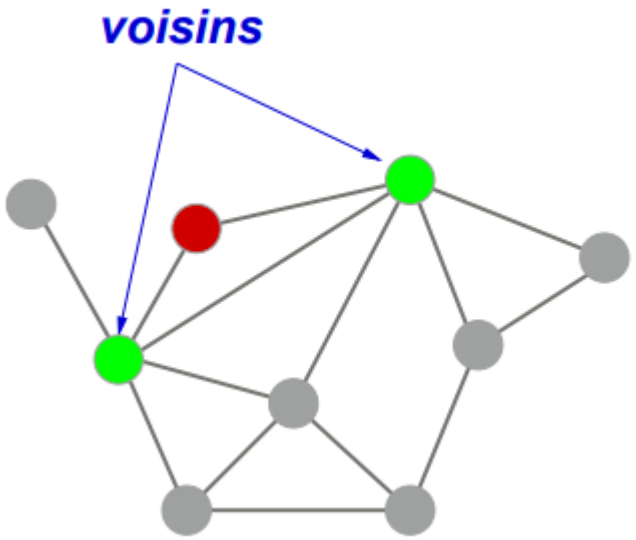


Graphe orienté pondéré:



Définitions et terminologies

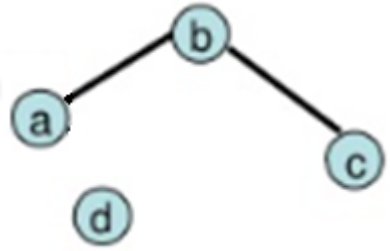
- Deux sommets d'un graphe sont dits *adjacents* s'il existe une arête (ou un arc) qui les relie.



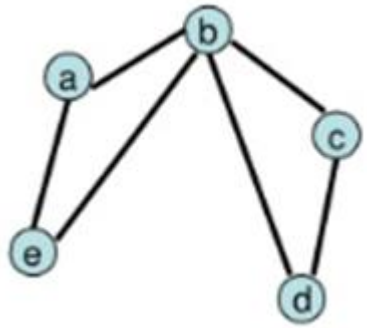
Définitions et terminologies

- L'ordre d'un graphe est le nombre de ses sommets.

Graphe d'ordre 4

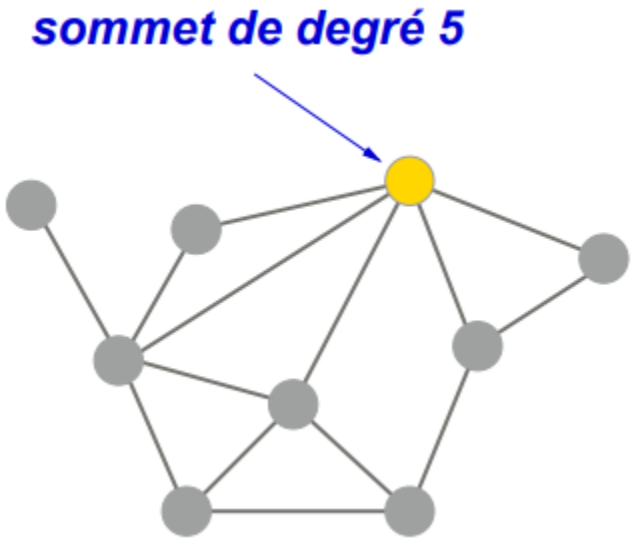


Graphe d'ordre 5



Définitions et terminologies

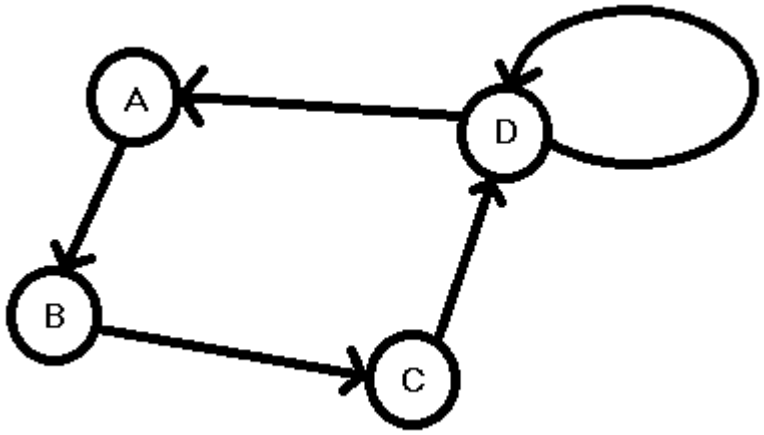
- Degré d'un sommet : nombre d'arêtes reliées à ce sommet.



Définitions et terminologies

- ***Un arc boucle :*** est un arc qui part d'un sommet vers le même sommet.

L'arc <D,D> est une boucle



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.

Des chaînes de longueur 2 :

A D C

A B C

Des chaînes de longueur 3 :

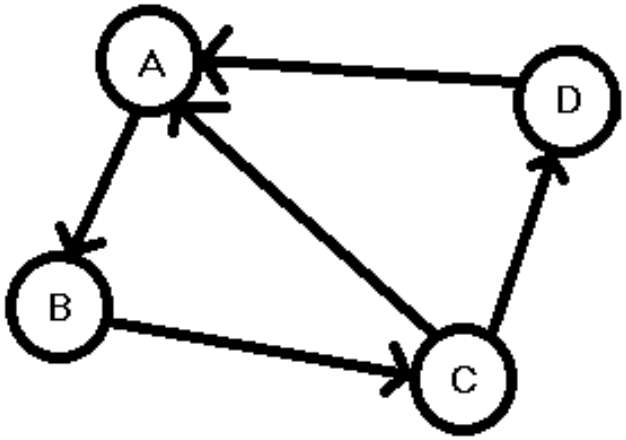
A B C D

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 une chaine qui permet de partir d'un sommet et revenir a ce sommet en parcourant une et une seule fois les autres sommets.

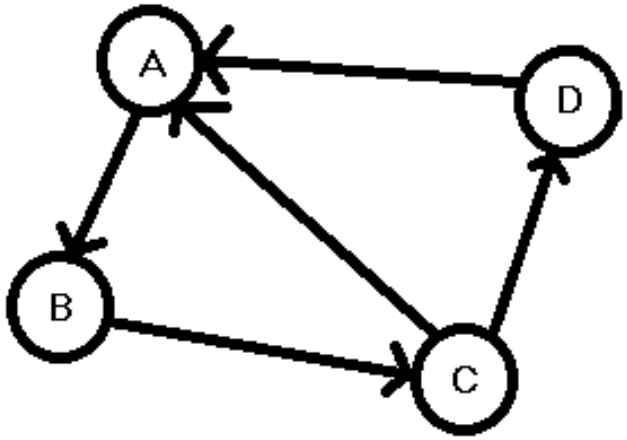
Exemples :

A B C D A est un cycle

A B C A est un cycle

D C A D est un cycle

A B C A D C A n'est un cycle



Définitions et terminologies

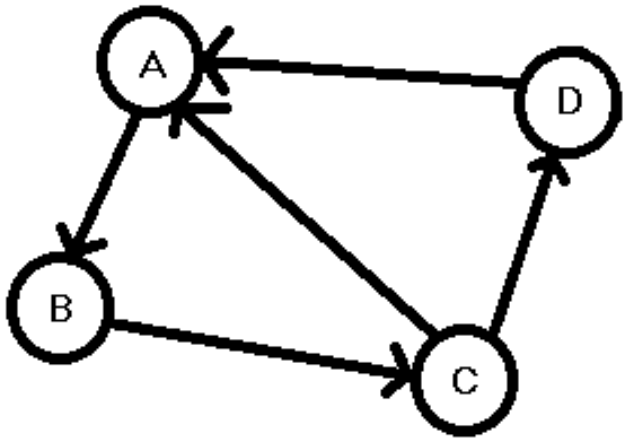
- ***Chemin :*** c'est une chaine bien orientée

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

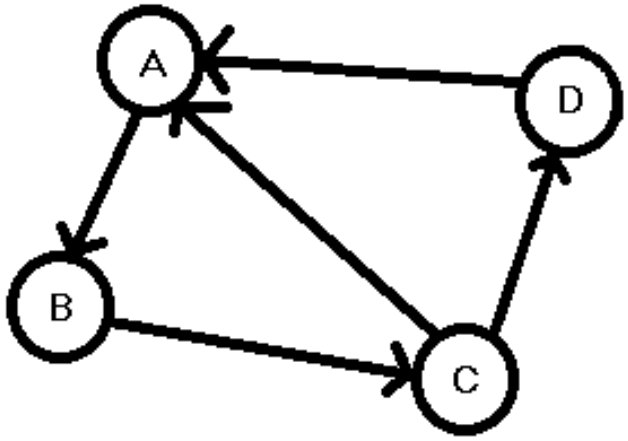
- ***Circuit :*** est un cycle "bien orienté", à la fois cycle et chemin.

Exemples :

A B C A est un circuit

A B C D A es un circuit

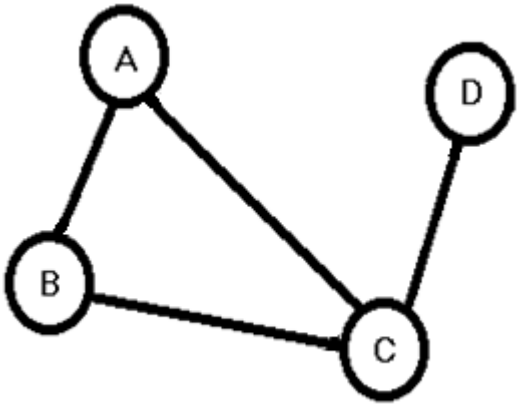
A D C A est un cycle mais pas un circuit



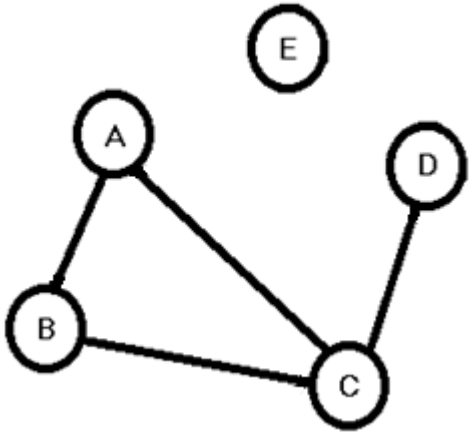
Définitions et terminologies

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 :

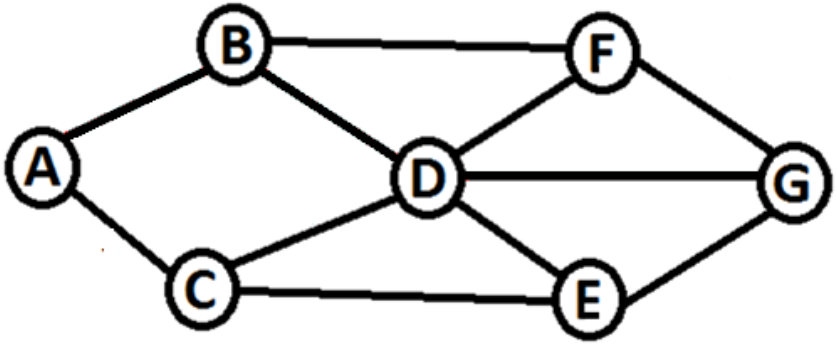


Graphe non connexe:

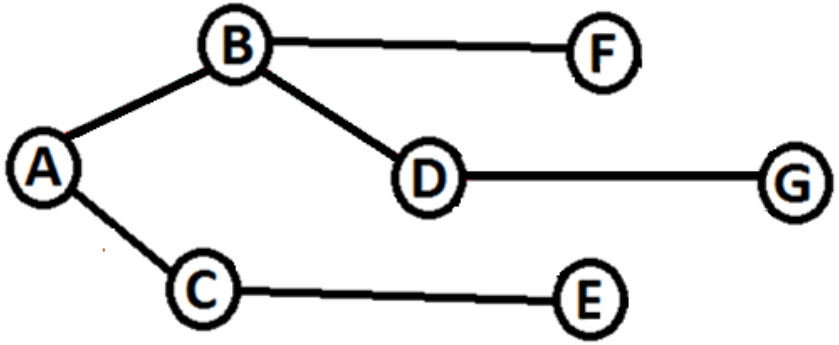


Définitions et terminologies

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

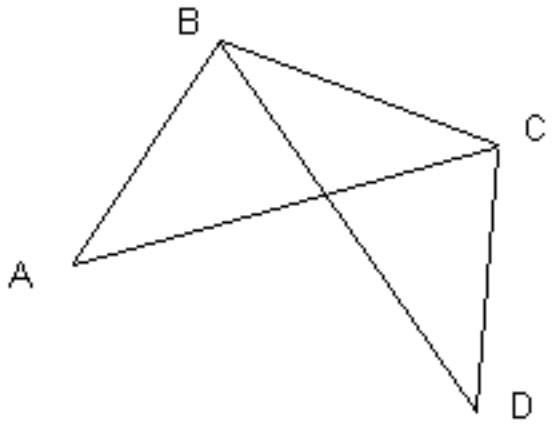
Définitions et terminologies

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

Exemples :

ABCD, ABDC, ACBD,ACBD

ACDB



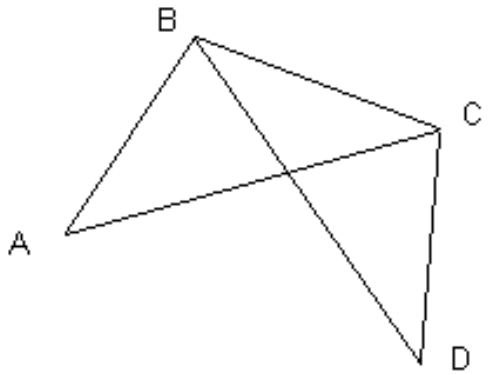
Définitions et terminologies

- **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



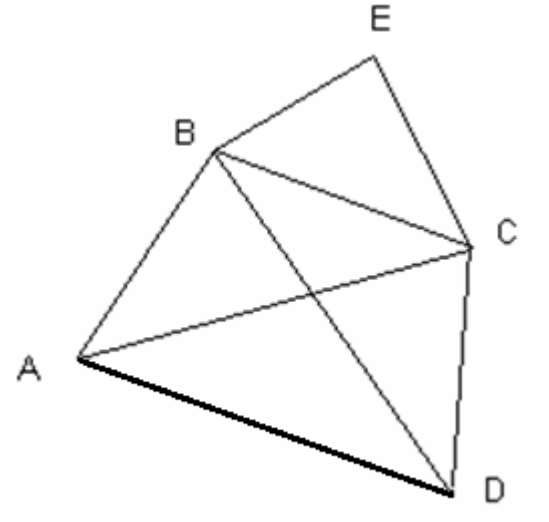
Définitions et terminologies

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

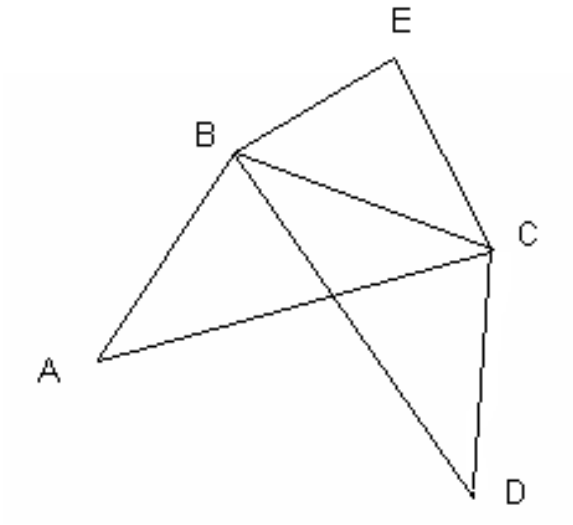
- **Cycle eulérien :** passant une seule fois par toutes les arêtes d'un graphe et revenant au sommet de départ.

Exemple :

Existe-t-il un cycle eulérien ?



Réponse : NON



Réponse : ABECDBCA

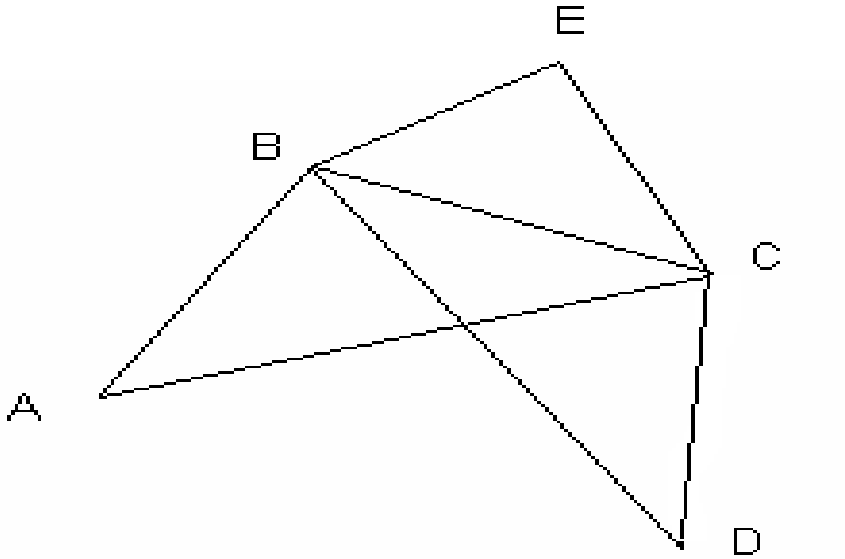
Définitions et terminologies

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

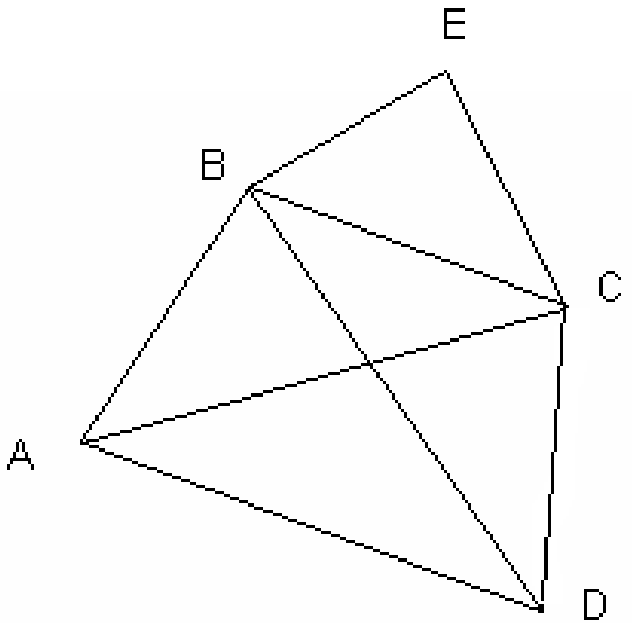
- **Graphe eulérien :** Graphe qui possède au moins un cycle Eulérien

Théorème d'Euler (1766)

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

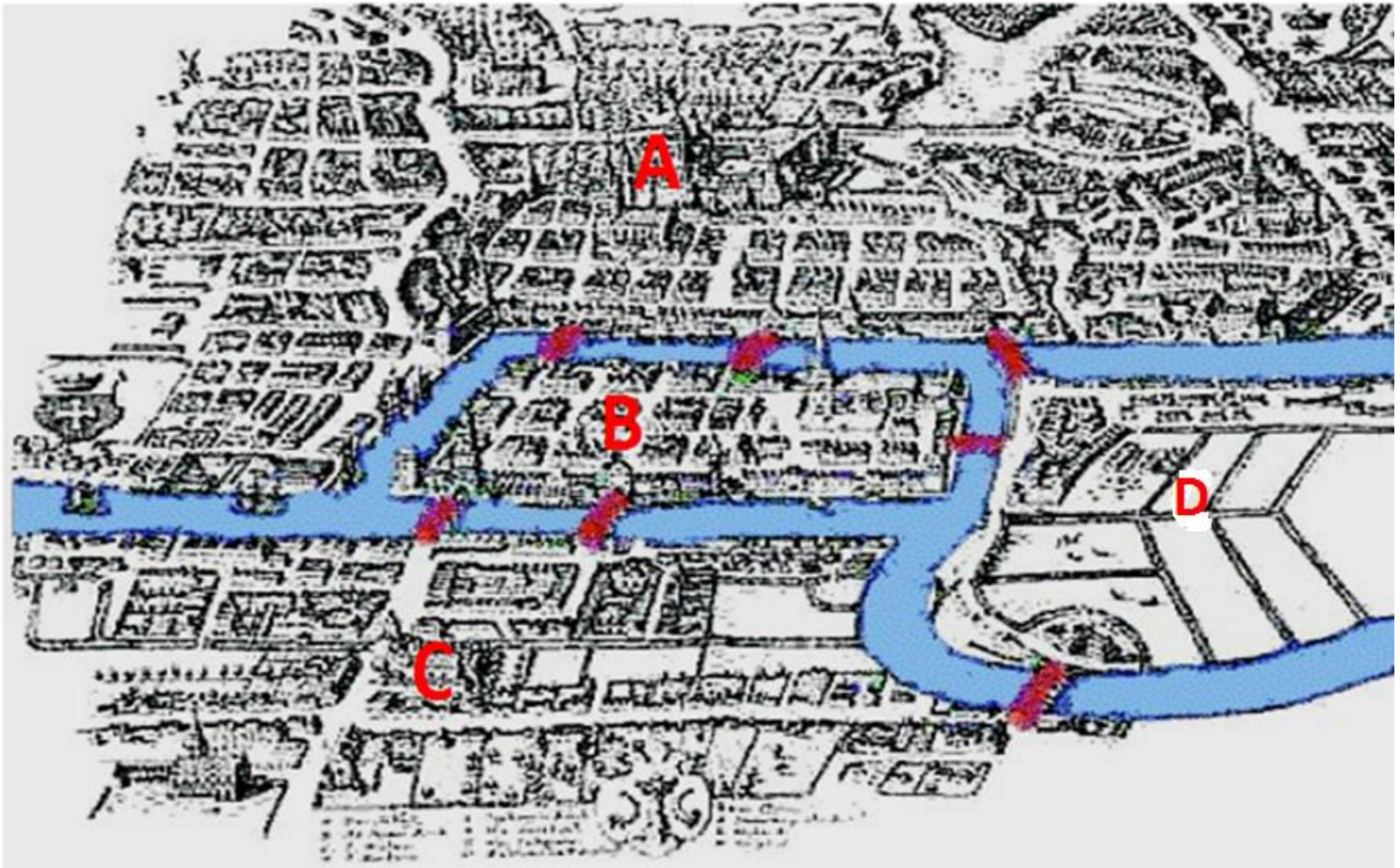


OUI



NON

Retour à Konigsberg



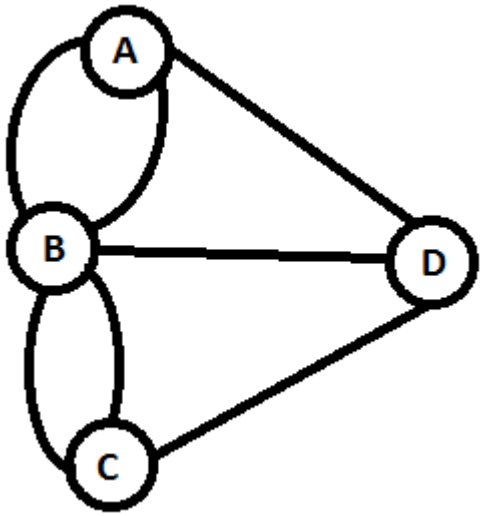
Sous forme de graphe

Les sommets = quartiers

Les arcs = Les ponts

Le problème ⇔ le graphe est il eulérien ?

Théorème ⇒ NON



Implémentation des graphes

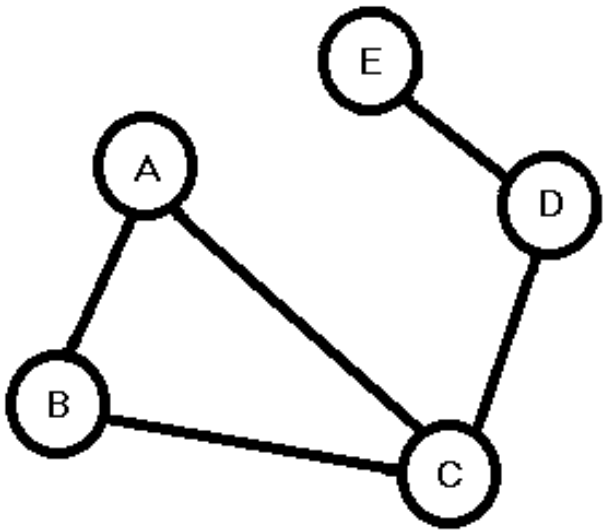
Deux implémentations possibles :

1. Par les dictionnaires de précédence

2. Par la matrice d'adjacences

Implémentation par : Dictionnaire de précédence

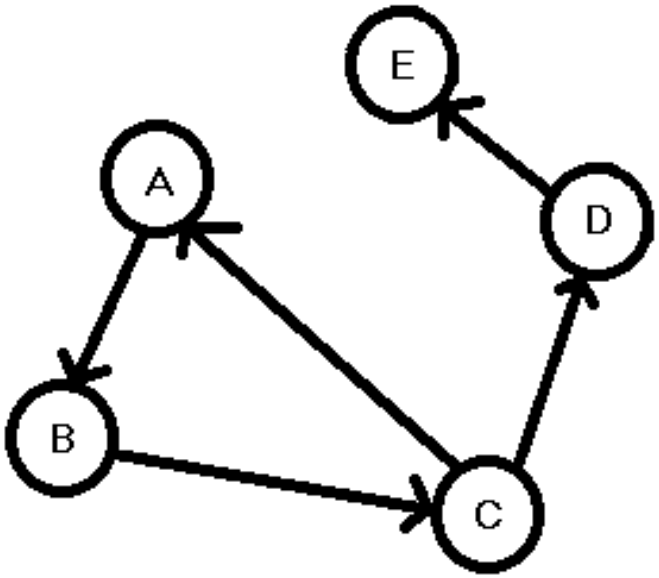
Graphe non orienté :




```
G = {
        'A' : ['B', 'C'],
        'B' : ['A', 'C'],
        'C' : ['A', 'B', 'D'],
        'D' : ['C', 'E'],
        'E' : ['D']
}
```

Implémentation par : Dictionnaire de précédence

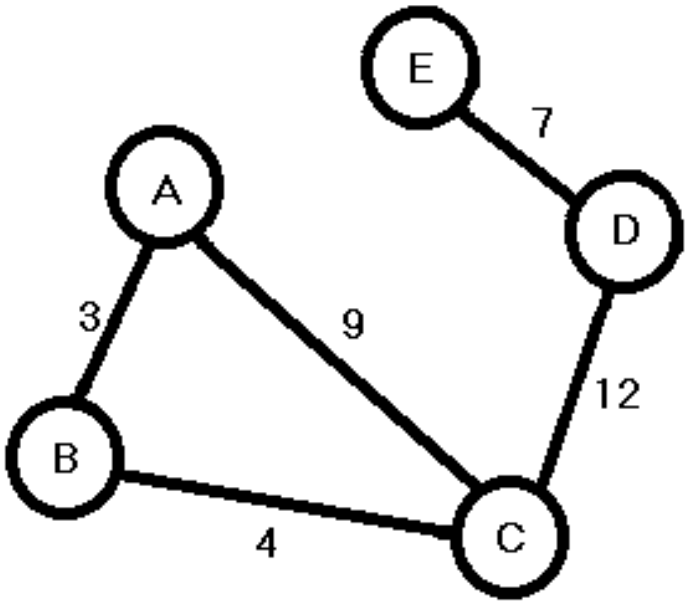
Graphe orienté :




```
G = {
        'A' : ['B'],
        'B' : ['C'],
        'C' : ['A', 'D'],
        'D' : ['E'],
        'E' : []
}
```

Implémentation par : Dictionnaire de précédence

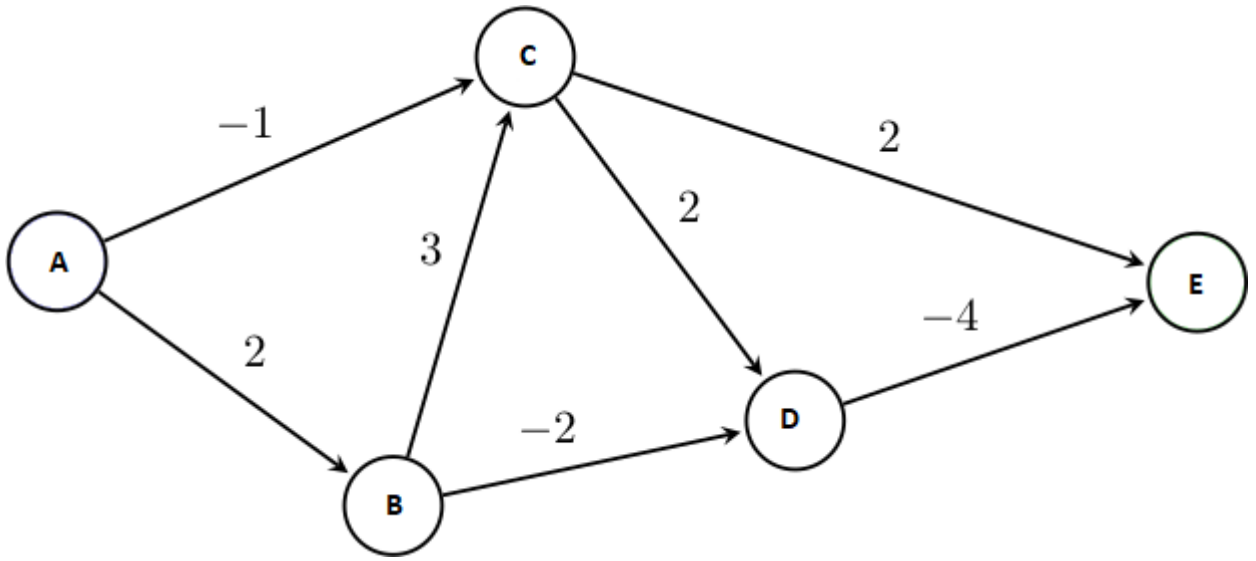
Graphe non orienté et valué :




```
G = {
    'A' : [(3,'B'), (9,'C')],
    'B' : [(3,'A'), (4,'C')],
    'C' : [(9,'A'), (4,'B')],
    'D' : [(7,’E'), (12,'C')],
    'E' : [(7,'D')]
}
```

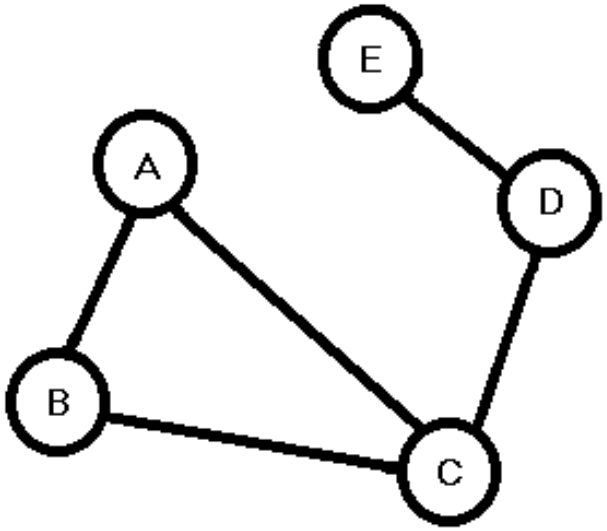
Implémentation par : Dictionnaire de précédence

Exercice : Donner l'implémentation avec le dictionnaire de précédence du graphe suivant.



Implémentation par : Matrice d'adjacence

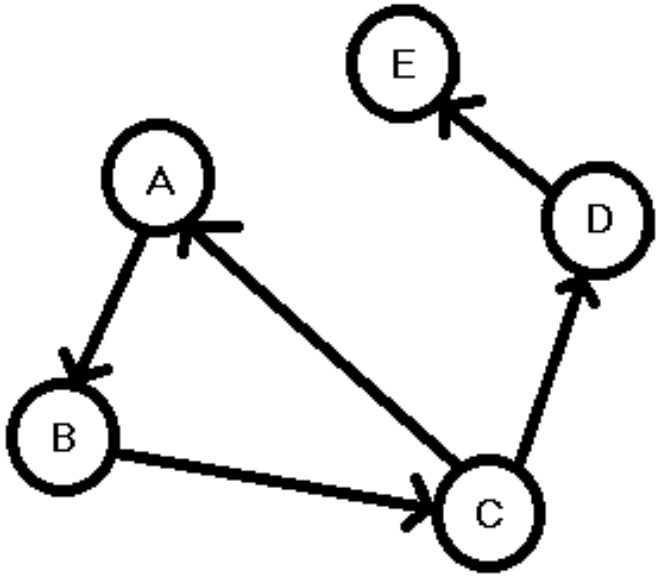
Graphe non orienté :



	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 par : Matrice d'adjacence

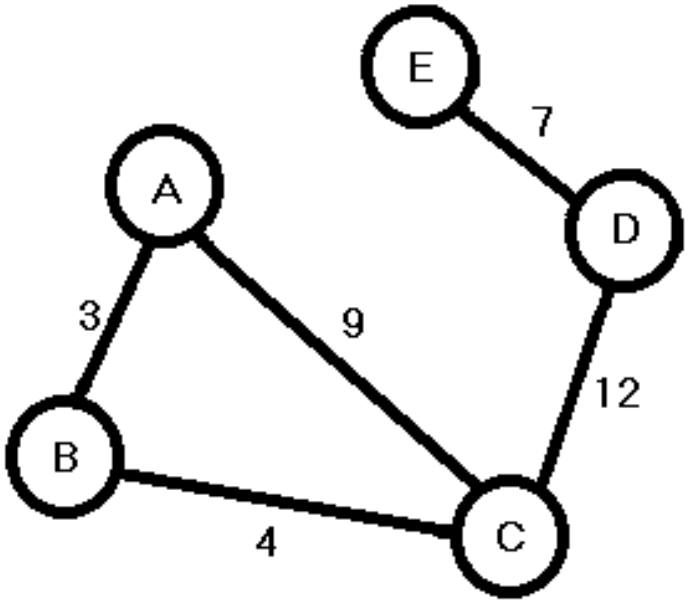
Graphe orienté :



	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 par : Matrice d'adjacence

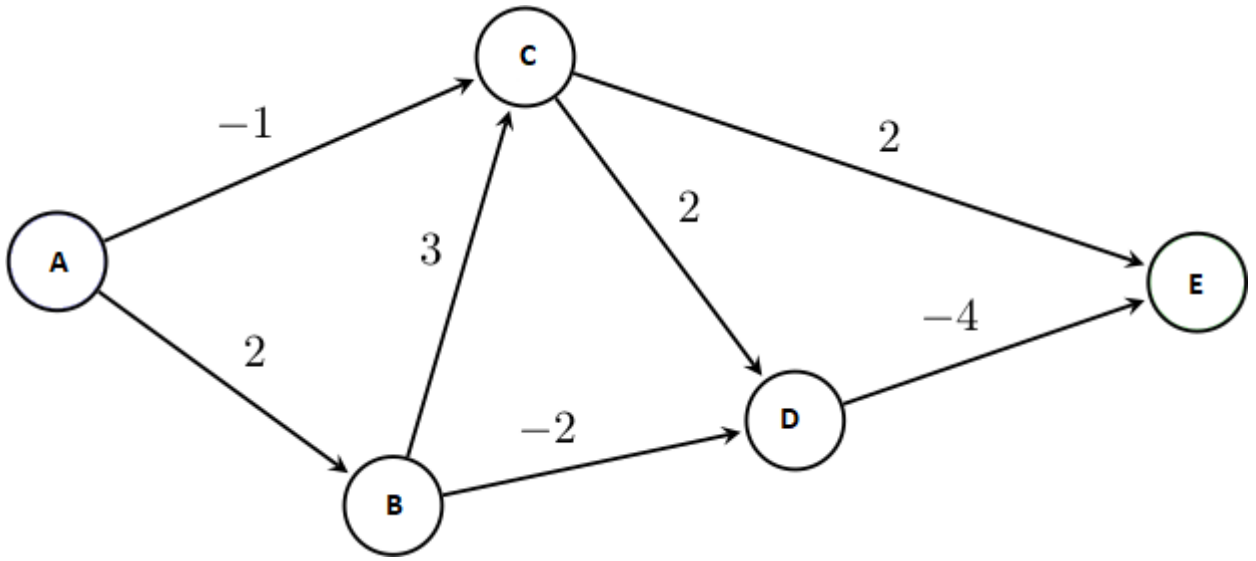
Graphe non orienté et valué :



	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 par : Matrice d'adjacence

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



Algorithmes sur les graphes

- Parcours en largeur
- Parcours en profondeur
- Algorihtme de Dijkstra

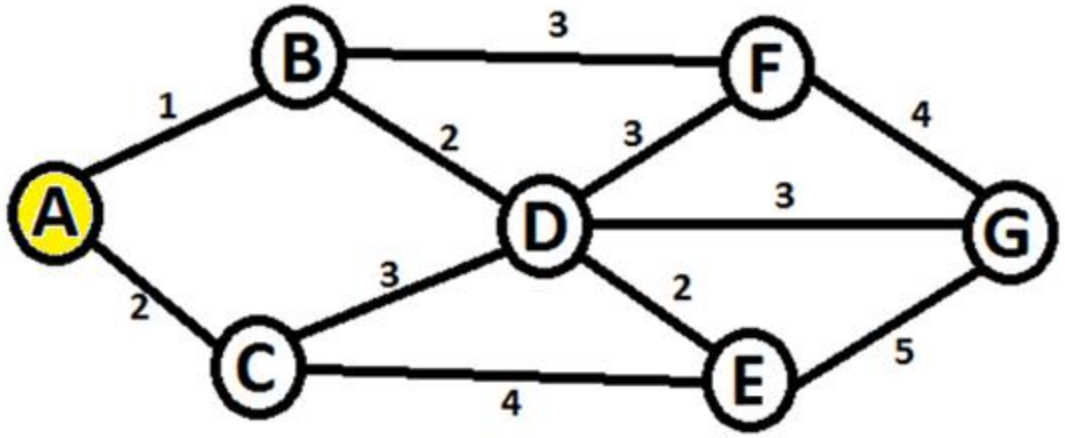
Parcours en largeur d’un graphe

Ce parcours peut être utilisé pour :

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

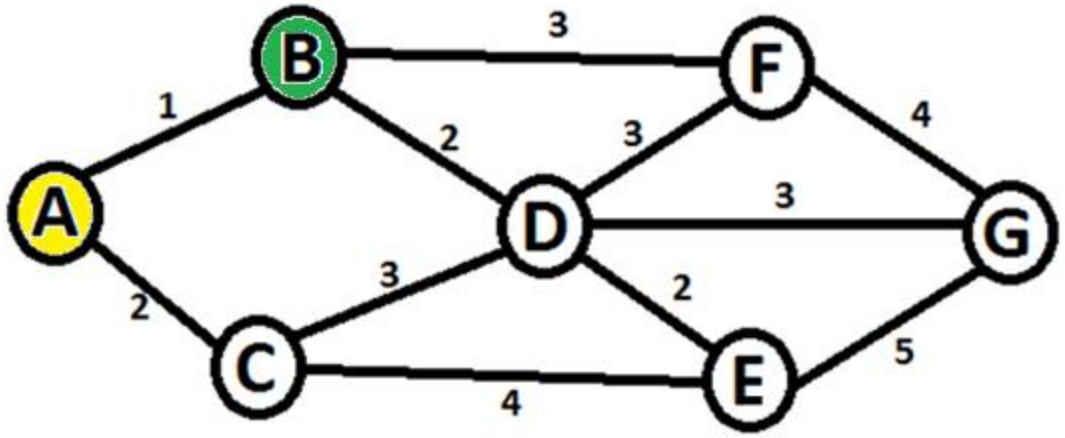
Algorithme :
1- S est le sommet de départ
2- Cet algorithme liste d'abord les voisins de S pour ensuite les explorer un par un.
2- Répétez (1) pour chaque voisin.

Parcours en largeur d'un graphe : Exemple



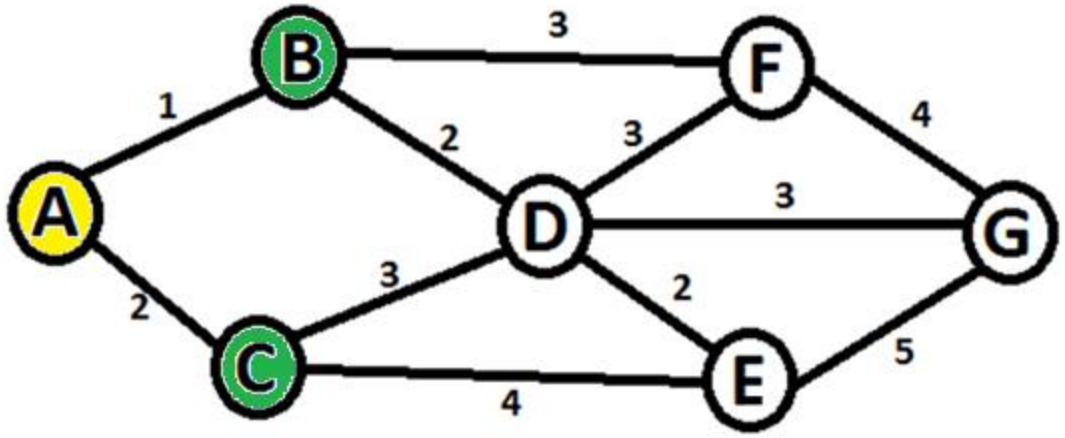
Liste des sommets visités : A,

Parcours en largeur d'un graphe : Exemple



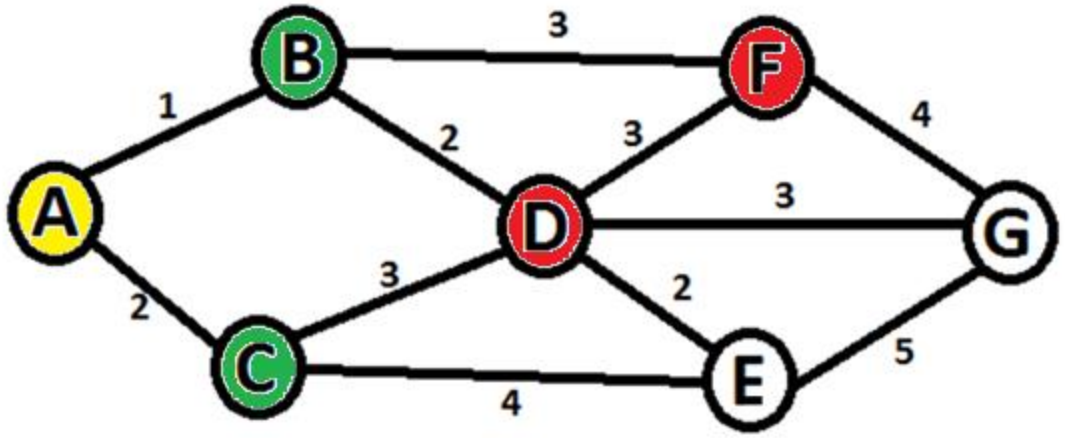
Liste des sommets visités : A, B

Parcours en largeur d'un graphe : Exemple



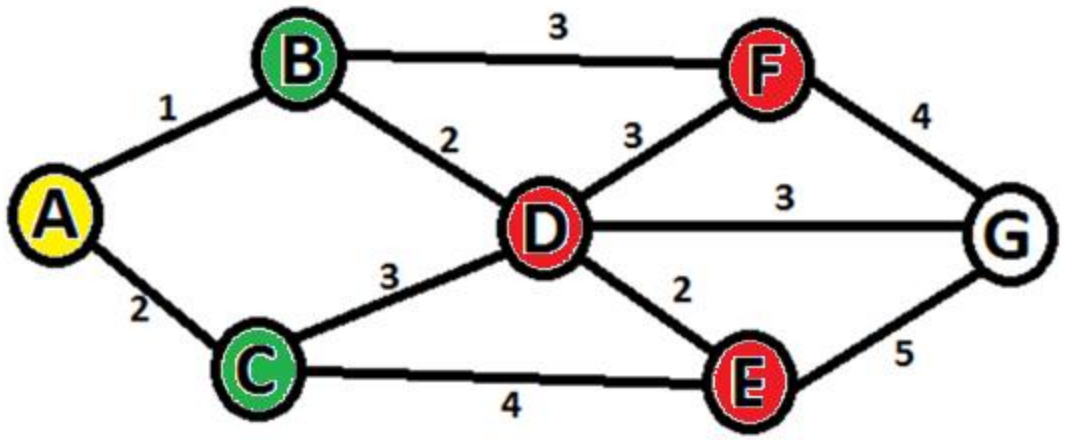
Liste des sommets visités : A, B, C

Parcours en largeur d'un graphe : Exemple



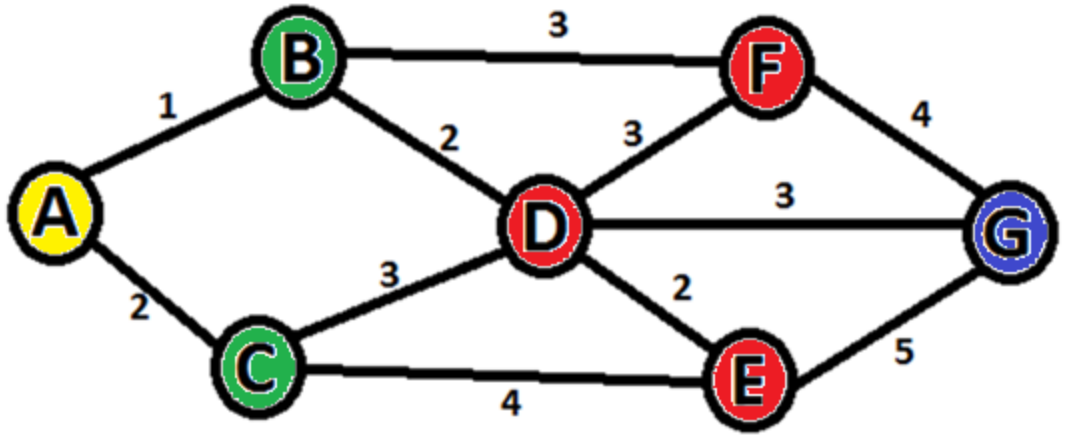
Liste des sommets visités : A, B, C, F, D

Parcours en largeur d'un graphe : Exemple



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

Parcours en largeur d'un graphe : Exemple

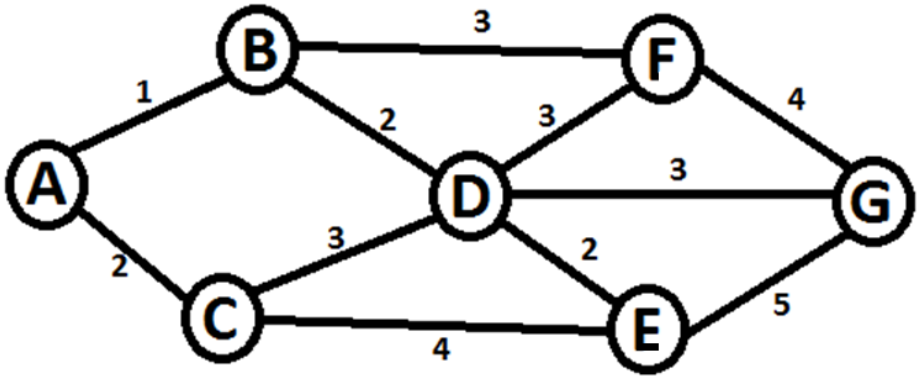


Liste des sommets visités : A, B, C, F, D, E, G

Parcours en largeur d'un graphe

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

On utilisera la représentation par dictionnaire de précédence d'un graphe.



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

Parcours en largeur d'un graphe : Solution

```
def parcours_largeur(G, d):
    visited = []
    file_to_visite = [d]
    while file_to_visite!=[]:
        s = file_to_visite.pop(0)
        if s in visited :
            continue
        visited.append(s)
        voisins = G[s]
        for dv, v in voisins:
            if v not in visited :
                file_to_visite.append(v)
    return visited
```

Parcours en profondeur d'un graphe

Ce parcours peut être utilisé pour :

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

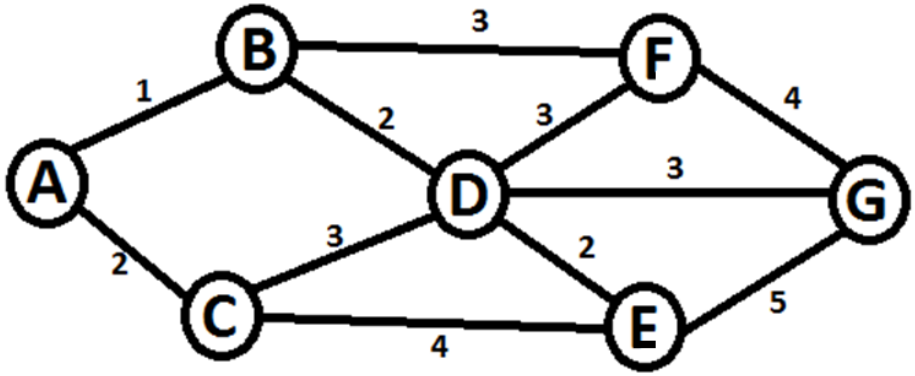
Algorithme :

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

Parcours en profondeur d'un graphe

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

On utilisera la représentation par dictionnaire de précédence d'un graphe.



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

Parcours en longueur d'un graphe : Solution

```
def parcours_profondeur(G, d, visited):
    visited.append(d)
    voisins = G[d]
    for dv, v in voisins :
        if v not in visited :
                parcours_profondeur(G, v, visited)
```

Pour tester :

```
L=[]
parcours_profondeur(GRAPHE, 'A', L)
print(L)
```

Algorithme de Dijkstra

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

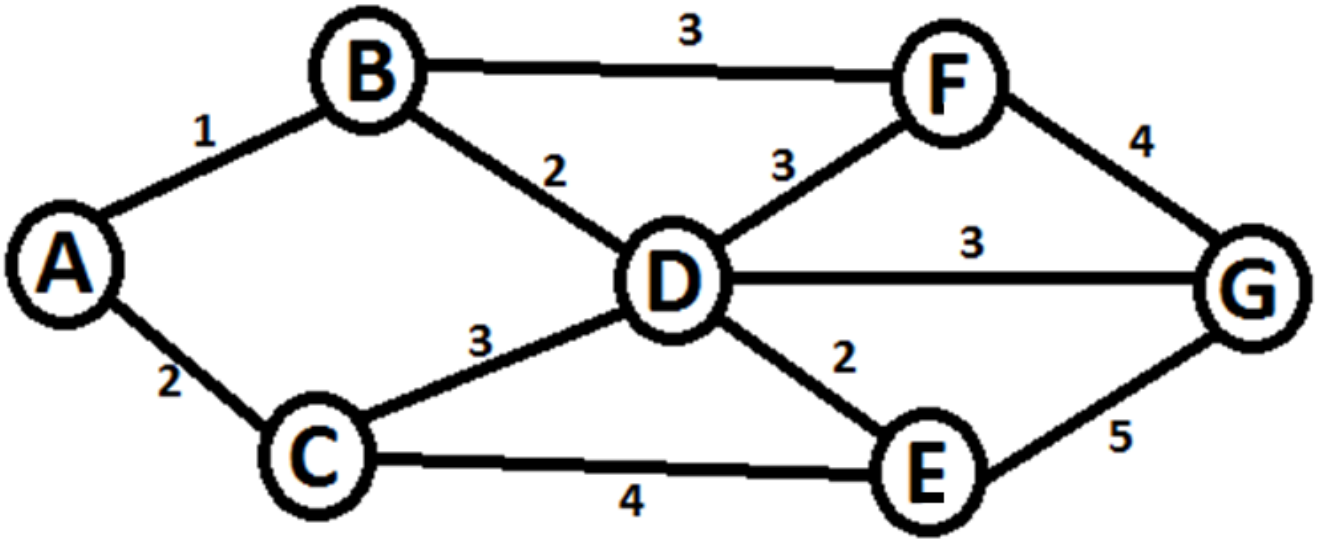
Il s'applique à un graphe connexe dont le poids lié aux arêtes est un réel positif.

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. ….

Algorithme de Dijkstra

Exemple : Trouver le chemin optimal entre A et G



Algorithme 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.

Algorithme de Dijkstra

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)**.

Algorithme de Dijkstra

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.

Algorithme de Dijkstra

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.

Algorithme 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							

Algorithme 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

Algorithme 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						

Algorithme 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						

Algorithme 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						

Algorithme de Dijkstra

Etapes de l'algorithme :

	A	**B**	**C**	**D**	**E**	**F**	**G**
Etape 1	0	∞	∞	∞	∞	∞	∞
Etape 2	X	<u>(1,A)</u>	(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					

Algorithme de Dijkstra

Etapes de l'algorithme :

	A	**B**	**C**	**D**	**E**	**F**	**G**
Etape 1	0	∞	∞	∞	∞	∞	∞
Etape 2	X	<u>(1,A)</u>	(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					

Algorithme 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)		
Etape 5	X	X	X				
Etape 6	X	X	X				
Etape 7	X	X	X				

Algorithme 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				

Algorithme 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			

Algorithme 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	

Algorithme 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)

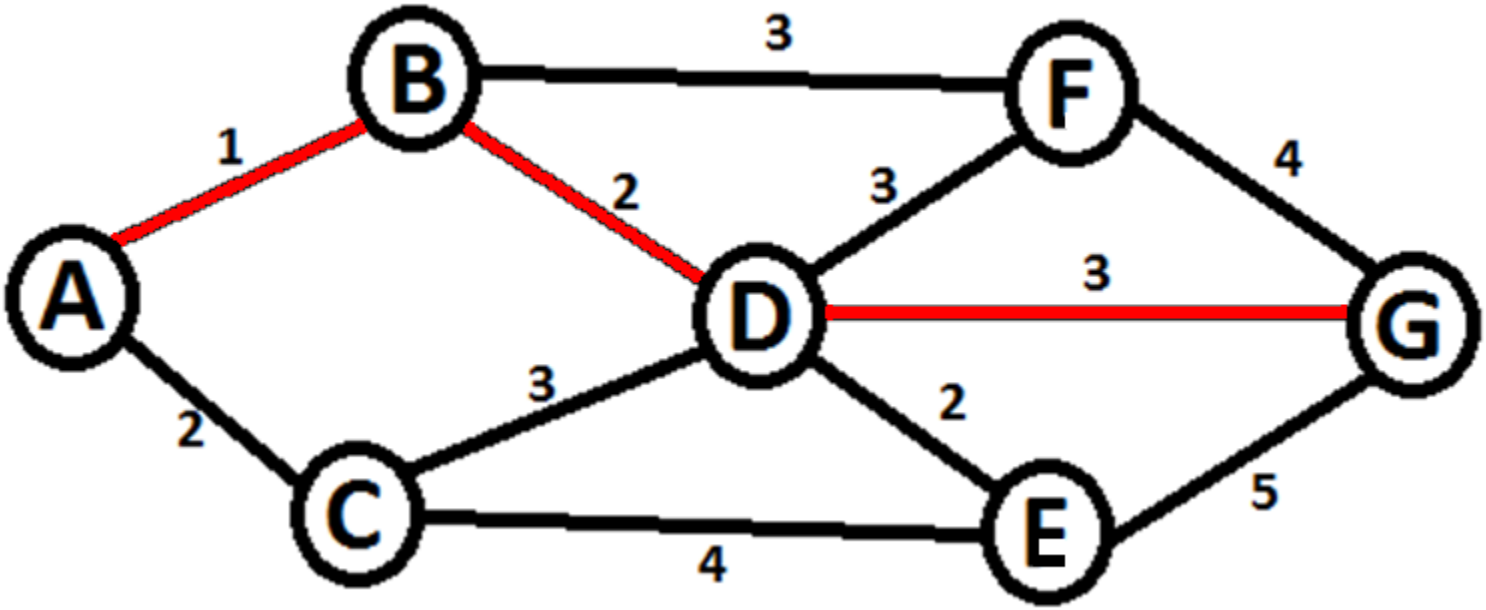
Algorithme de Dijkstra

Etapes de l'algorithme :

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

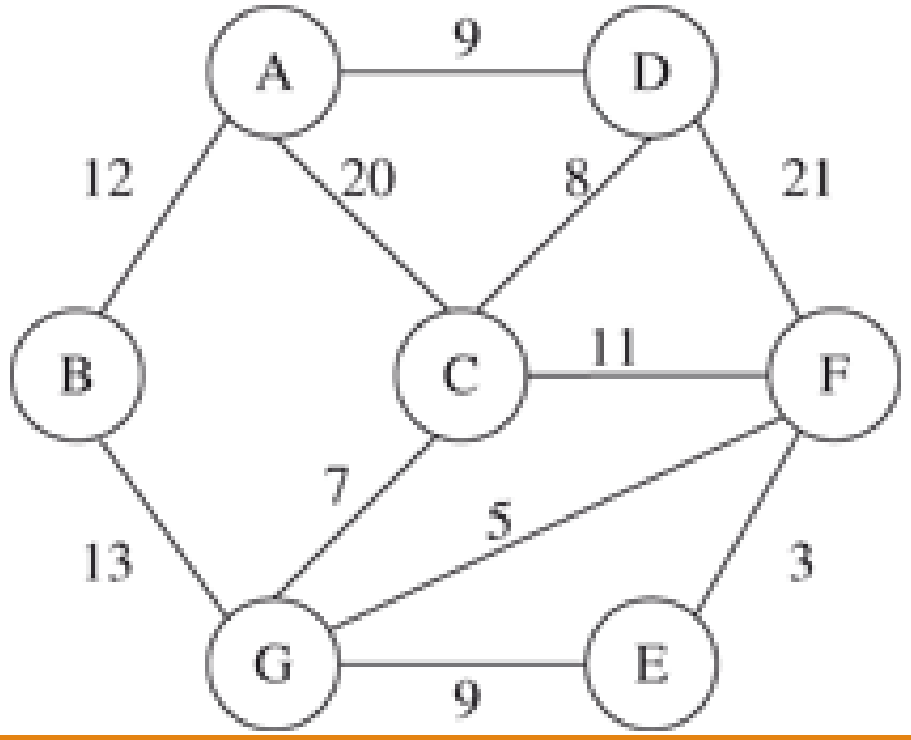
Algorithme de Dijkstra

Le chemin optimal est : ABDG de coût égal à 6



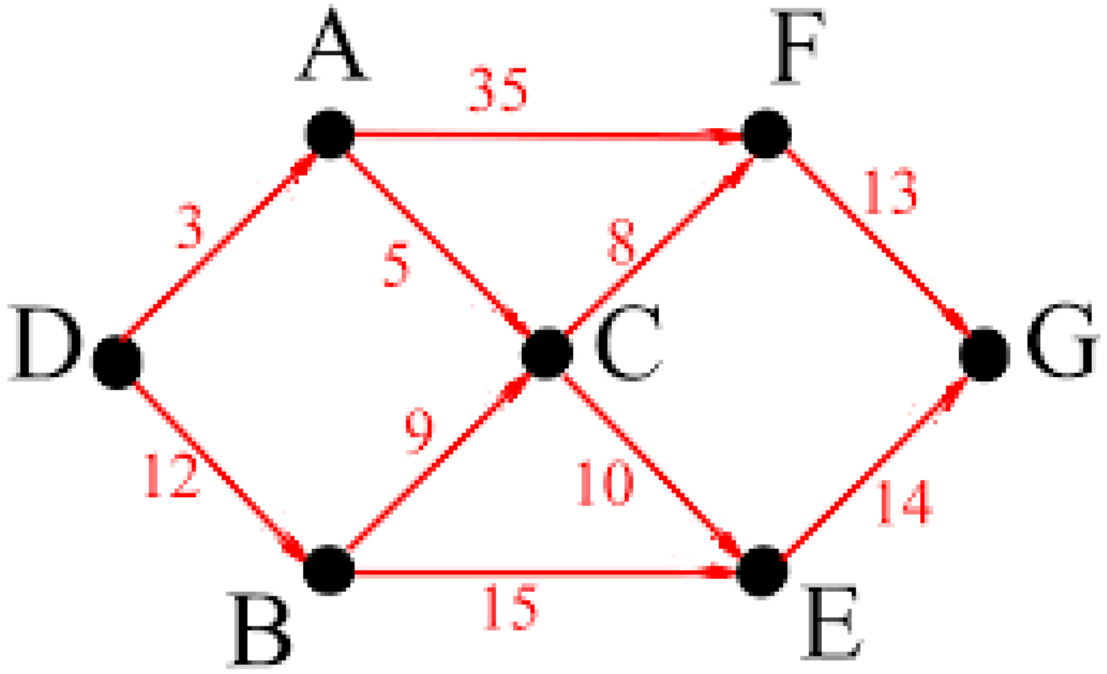
Algorithme de Dijkstra

Exercice : Trouver le chemin optimal entre A et E



Algorithme de Dijkstra

Exercice : Trouver le chemin optimal entre D et G



Algorithme de Dijkstra

Etapes de l'algorithme :

	A	**B**	**C**	**D**	**E**	**F**	**G**
Etape 1	0	∞	∞	∞	∞	∞	∞
...							
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

Programme de Dijkstra

```
def index_min_file(file):
    id = 0
    for i in range(1, len(file)):
        if file[i][0]< file[id][0] :
        id=i
    return id
```

```
def dijkstra(G, d, f):
    # Dictionnaires des distances minimales
    D={d:0}
    # Dictionnaire des précédents
    P={}
    # Liste des sommets visités
    visited = []
    # Liste des sommets en attente de
    traitement
    file = [(0,d)]
    while file!=[] :
        # On traite le sommet à distance
        minimale
        id = index_min_file(file)
        ds, s= file.pop(id)
        # Traiter le sommet s
        voisins =G[s]
        for dv, v in voisins:
                if v in visited:
                        continue
                dn = dv + ds
                if v not in D or dn < D[v] :
                        D[v]=dn
                        file.append((dn, v))
                        P[v]=s
        visited.append(s)
        if f in visited :
                break
# Calculer le chemin optimale
path = [f]
x=f
while x!=d:
    x=P[x]
    path.insert(0,x)
return D[f], path
```

Exercice :

Implémenter l’algorithme de Dijsktra en utilisant la représentation du graphe par une matrice d’adjacence.