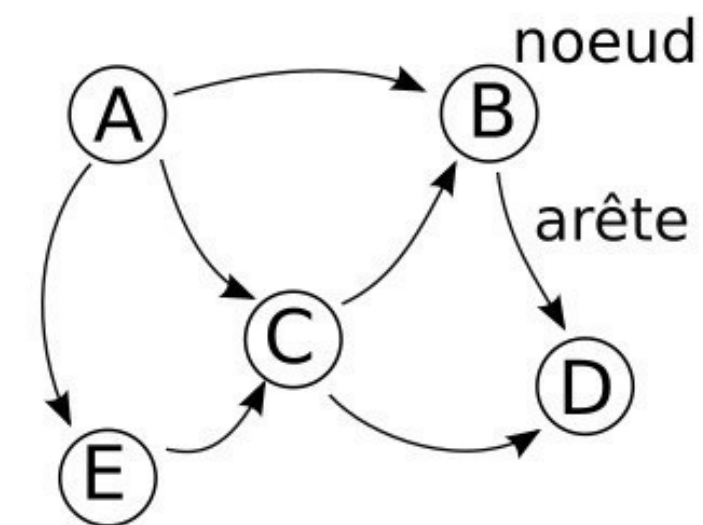


# Graphes

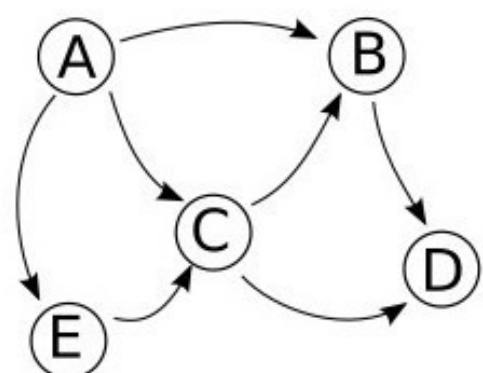
000

# Graphes



001

# Graphes



Modélisation par set avantageuse

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

print("Valence de C")
print(len(G['C']))

print("C est-il relie a B ?")
print('B' in G['C'])

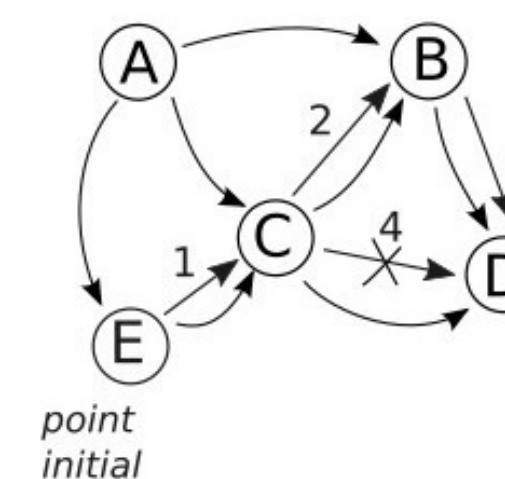
print("D est-il relie a A ?")
print('D' in G['A'])

print("Ensemble des voisins de A")
for v in G["A"]:
    print(v)
```

002

# Graphes

Parcours complet d'un graphe à partir d'un point initial



noeuds déjà visités

```
0: E
1: E, C
2: E, C, B
3: E, C, B, D
```

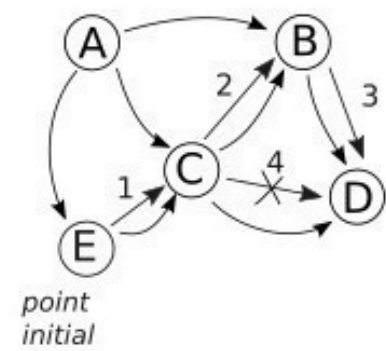
Algorithme: parcours en profondeur

```
visiter(noeud):
    Pour tous mes voisins v
        Si v n'a pas été visité
            visiter v
```

003

# Graphes

Parcours complet d'un graphe à partir d'un point initial



Algorithme: parcours en profondeur

```

visiter(noeud):
    Pour tous mes voisins v
        Si v n'a pas été visité
            visiter v
    
```

```

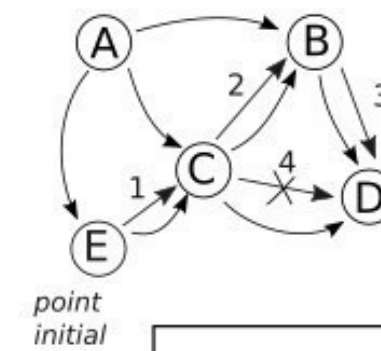
def parcours(graphe, noeud, visite):
    visite.add(noeud)
    if noeud in graphe:
        for voisin in graphe[noeud]:
            if voisin not in visite:
                print(voisin)
                parcours(graphe, voisin, visite)

visite=set()
parcours(G, "E", visite)
    
```

004

# Graphes

Parcours complet d'un graphe à partir d'un point initial



Algorithme: parcours en profondeur

```

visiter(noeud):
    Pour tous mes voisins v
        Si v n'a pas été visité
            visiter v
    
```

```

je suis sur A
un voisin de A est: C
je suis sur A et je vais visiter C
je suis sur C
un voisin de C est: D
je suis sur C et je vais visiter D
je suis sur D
D n'a pas de fils
un voisin de C est: B
je suis sur C et je vais visiter B
je suis sur B
un voisin de B est: D
je suis sur B et j'ai déjà visité D
un voisin de A est: B
je suis sur A et j'ai déjà visité B
un voisin de A est: E
je suis sur A et je vais visiter E
je suis sur E
un voisin de E est: C
je suis sur E et j'ai déjà visité C
    
```

```

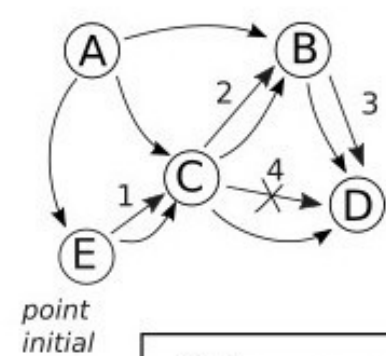
def parcours_explication(graphe, noeud, visite):
    visite.add(noeud)
    print("je suis sur", noeud)
    if noeud in graphe:
        for voisin in graphe[noeud]:
            print("un voisin de", noeud, " est:", voisin)
            if voisin not in visite:
                print("je suis sur", noeud, "et je vais visiter", voisin)
                parcours_explication(graphe, voisin, visite)
            else:
                print("je suis sur", noeud, "et j'ai déjà visité", voisin)
        else:
            print(noeud, "n'a pas de fils")

visite=set()
parcours_explication(G, "A", visite)
    
```

005

# Graphes

Parcours itératif : stocker les sommets à visiter



Algorithme: parcours en profondeur

```

visiter(sommets_a_visiter<-noeud initial):
    Tant qu'il y a des sommets a visiter
        s <- dépiler ler sommet de la liste
        Pour tous les voisins v de s
            Si v n'a pas déjà été visité
                Ajouter v aux sommets à visiter
    
```

```

def parcours_iteratif(graphe, a_visiter, deja_visite):
    while len(a_visiter) != 0:
        courant=a_visiter.pop()
        for voisin in graphe[courant]:
            if voisin not in deja_visite:
                a_visiter.add(voisin)
                deja_visite.add(courant)

deja_visite=set()
a_visiter={"A"}
parcours_iteratif(G, a_visiter, deja_visite)
print(deja_visite)
    
```

006

# Graphes

Construire le graphe modélisant le comportement suivant

Franck, Benjamin, Matilde, Francois, Maeva, Robert, Jean et Béatrice sont étudiants

Franck est ami avec Benjamin  
 Francois est ami avec Matilde  
 Robert est ami avec Matilde  
 Jean est ami avec Robert  
 Beatrice est ami avec Robert  
 Maeva n'a pas d'ami

Toutes les relations d'amitiés sont réciproques  
 c-a-d si A est ami avec B, alors B est ami avec A.

007

# Graphes

Construire le graphe modélisant le comportement suivant

Franck, Benjamin, Matilde, Francois, Maeva, Robert, Jean et Béatrice sont étudiants

Franck est ami avec Benjamin  
Francois est ami avec Matilde  
Robert est ami avec Matilde  
Jean est ami avec Robert  
Beatrice est ami avec Robert  
Maeva n'a pas d'ami

Toutes les relations d'amitiés sont réciproques  
c-a-d si A est ami avec B, alors B est ami avec A.

```
G={}
noms=["Franck","Benjamin","Matilde","Francois","Maeva",
      "Robert","Jean","Beatrice"]
for n in noms:
    G[n]=set()

def ami(G,nom_a,nom_b):
    G[nom_a].add(nom_b)
    G[nom_b].add(nom_a)

ami(G,"Franck","Benjamin")
ami(G,"Francois","Matilde")
ami(G,"Matilde","Robert")
ami(G,"Jean","Robert")
ami(G,"Beatrice","Robert")

print(G)
```

008

# Graphes

Afficher les amis de Roberts

Jean n'est plus ami avec Robert  
Maeva devient amie avec Benjamin  
Mettre à jour ce nouveau graphe

009

# Graphes

Afficher les amis de Roberts

Jean n'est plus ami avec Robert  
Maeva devient amie avec Benjamin  
Mettre à jour ce nouveau graphe

```
G={}
noms=["Franck","Benjamin","Matilde","Francois","Maeva","Robert",
      "Jean","Beatrice"]
for n in noms:
    G[n]=set()

def ami(G,nom_a,nom_b):
    G[nom_a].add(nom_b)
    G[nom_b].add(nom_a)

def non_ami(G,nom_a,nom_b):
    G[nom_a].remove(nom_b)
    G[nom_b].remove(nom_a)

ami(G,"Franck","Benjamin")
ami(G,"Francois","Matilde")
ami(G,"Matilde","Robert")
ami(G,"Jean","Robert")
ami(G,"Beatrice","Robert")

non_ami(G,"Jean","Robert")
ami(G,"Maeva","Benjamin")

print(G)
```

010

# Graphes

réseau = ami d'amis, d'amis, ...

Construire l'ensemble du réseau de "Robert"  
Déduire si Francois appartient au "réseau" de Robert  
Déduire si Franck appartient au "réseau" de Robert

011

## Graphes

réseau = ami d'amis, d'amis, ...  
 Construire l'ensemble du réseau de "Robert"  
 Déduire si Francois appartient au "réseau" de Robert  
 Déduire si Franck appartient au "réseau" de Robert

```
def parcours(graphe,nom,visite):
    visite.add(nom)
    if nom in graphe:
        for voisin in graphe[nom]:
            if voisin not in visite:
                parcours(graphe,voisin,visite)

def est_dans_reseau(nom_a,nom_b,reseau):
    if nom_a in reseau:
        print(nom_a,"est dans le reseau de",nom_b)
    else:
        print(nom_a,"n'est pas dans le reseau de",nom_b)

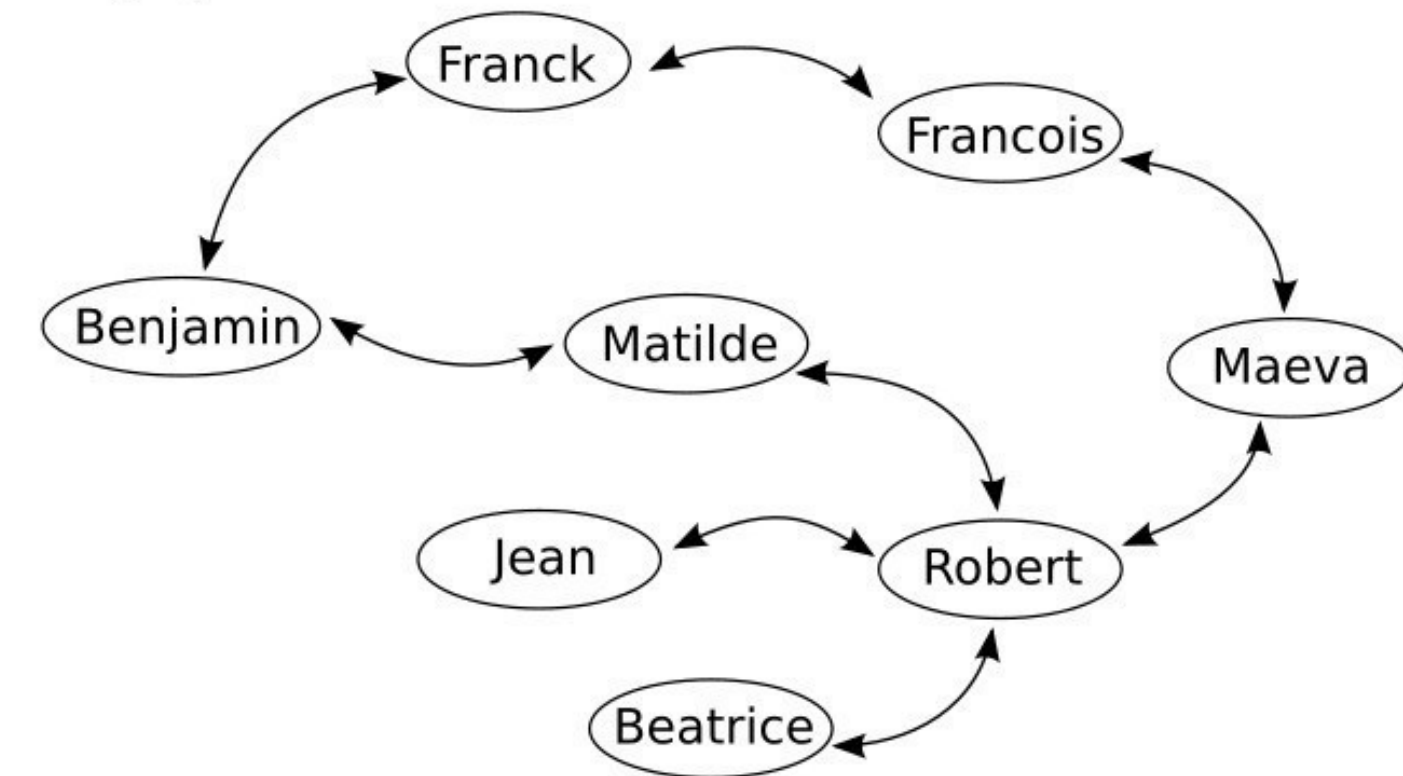
reseau=set()
parcours(G,"Robert",reseau)

print(reseau)
est_dans_reseau("Francois","Robert",reseau)
est_dans_reseau("Franck","Robert",reseau)
```

012

## Calcul de plus courts chemin: Dijkstra

Soit le graphe

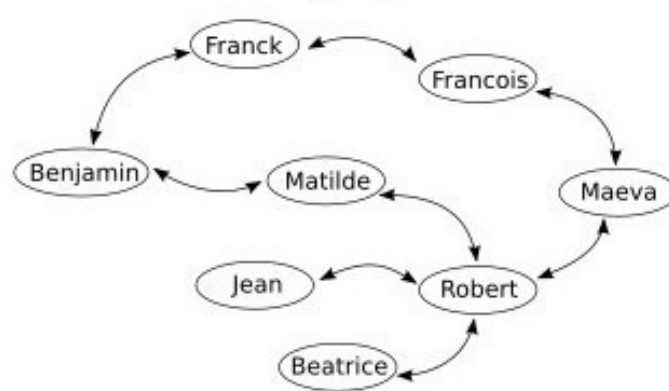


Calculer les "distances" les plus courtes en nombres d'amis séparant un élément d'un autre

013

## Calcul de plus courts chemin: Dijkstra

Soit le graphe



On stocke une carte des distances les plus courtes

On met à jour une distance si la précédente était plus longue

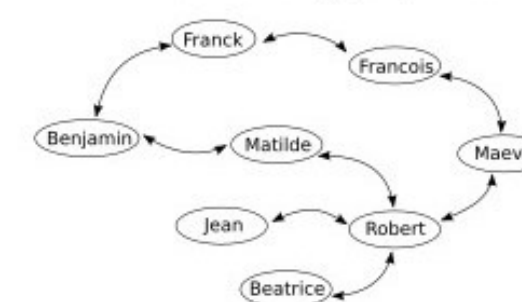
Algorithme:

```
Propagation_distance(noeud,distance,carte_distance)
Pour tous les voisins v du noeud
    Si carte_distance[v] < distance+1
        Mettre a jour carte_distance[v]
        Propagation_distance(v,distance+1,carte_distance)
```

014

## Calcul de plus courts chemin: Dijkstra

Soit le graphe



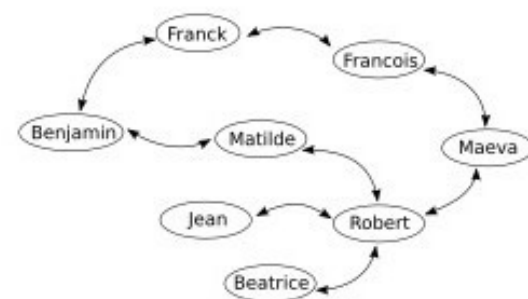
```
Propagation_distance(noeud,distance,carte_distance)
Pour tous les voisins v du noeud
    Si carte_distance[v] < distance+1
        Mettre a jour carte_distance[v]
        Propagation_distance(v,distance+1,carte_distance)
```

```
def parcours_dijkstra(graphe,nom,carte_distance,distance_courante):
    if nom in graphe:
        for voisin in graphe[nom]:
            if (voisin not in carte_distance) or
                (carte_distance[voisin] > distance_courante+1):
                carte_distance[voisin] = distance_courante+1
                parcours_dijkstra(graphe,voisin,carte_distance,distance_courante+1)
```

015

# Calcul de plus courts chemin: Dijkstra

Soit le graphe



Version itérative

```
def parcours_dijkstra_iteratif(graphe,a_visiter,carte_distance):  
    while len(a_visiter)>0:  
        nom=a_visiter.pop()  
        distance_courante=carte_distance[nom]  
        for voisin in graphe[nom]:  
            if (voisin not in carte_distance) or  
                (carte_distance[voisin]>distance_courante+1):  
                carte_distance[voisin]=distance_courante+1  
                a_visiter.add(voisin)  
  
a_visiter={"Jean"}  
carte_distance={"Jean":0}  
parcours_dijkstra_iteratif(G,a_visiter,carte_distance)  
print(carte_distance)
```