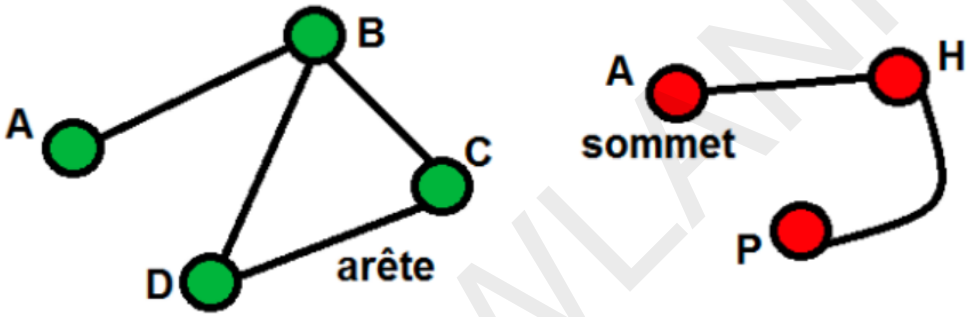
Les graphes

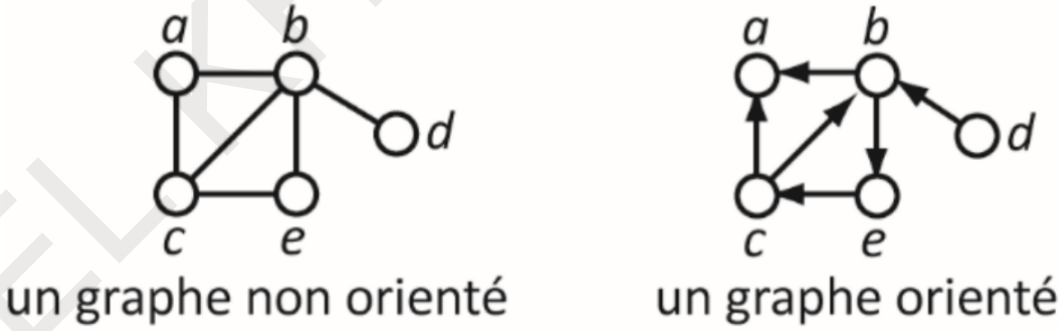
February 17, 2024

1 Les graphes - Généralités

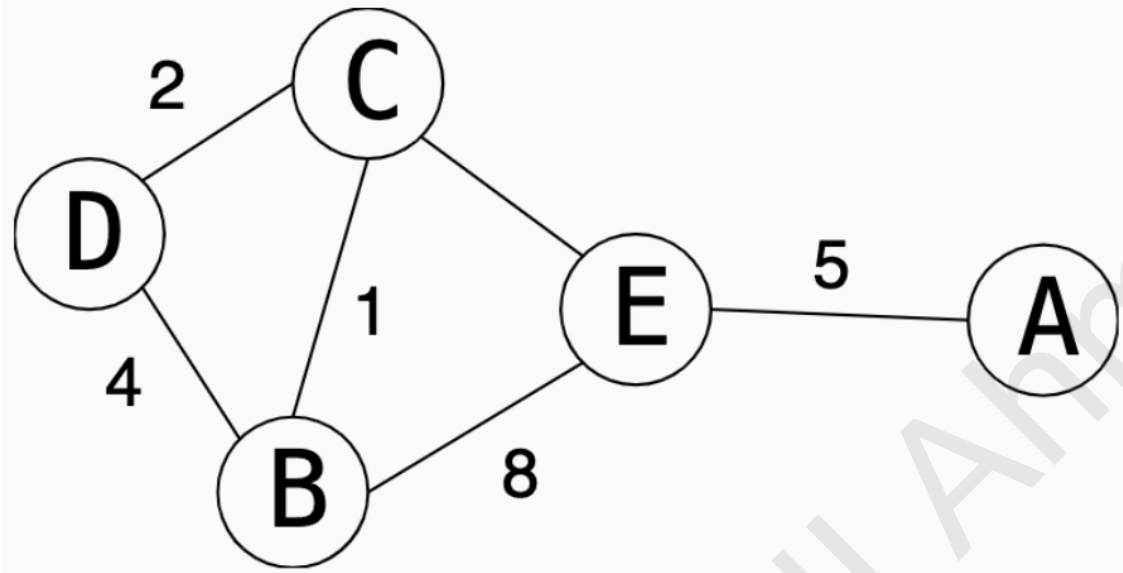
Un graphe est la donnée d'un certain nombre de points du plan, appelés sommets, certains étant reliés par des segments de droites ou de courbes appelés arêtes, la disposition des sommets et la forme choisie pour les arêtes n'intervenant pas.



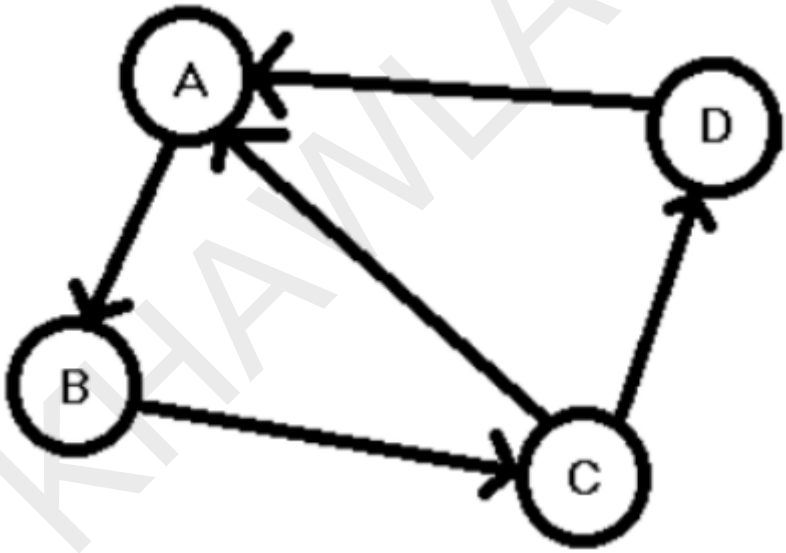
- Deux sommets d'un graphe sont dits adjacents ou voisins s'ils sont reliés par une arête.
- L'ordre d'un graphe représente le nombre de sommet d'un graphe.
- Le degré d'un sommet est le nombre d'arêtes reliés à ce sommet.



Un graphe pondéré : Un graphe dont les arêtes ont des valeurs.



1.1 Chaîne, cycle, chemin, circuit



- Une chaîne de longueur n est une suite de n arêtes qui relit un sommet i à un autre j ou à lui même;
- L'orientation (Au cas d'un graphe orienté) n'interagit pas dans les chaînes;
- Exemple d'une chaîne : ABC, ADC, ADCBA.
- Un cycle est une chaîne qui permet de partir d'un sommet et revenir au même sommet en parcourant une seule fois tous les autres sommets;
- Exemple d'un cycle : ABCDA, ABCA.
- Un chemin est une chaîne bien orientée;
- Exemple d'un chemin : ABC.
- Un circuit est un cycle bien orienté (chemin et cycle à la fois).

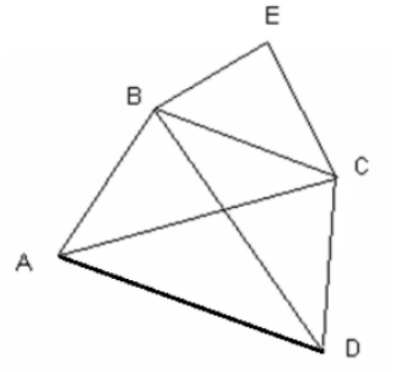
- Chaine hamiltonienne : Chaine passant une seule fois par tous les sommets d'un graphe;
- Exemple d'une chaine hamiltonienne : ABCD, ADCB.

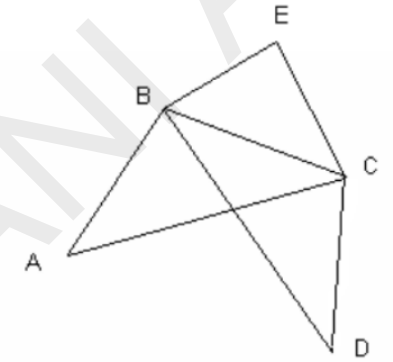
- Chaîne eulérienne : Chaîne passant une seule fois par toutes les arêtes d'un graphe;
- Exemple d'une chaîne eulérienne : ABCDAC.

- Cycle hamiltonien : Passant une seule fois par tous les sommets d'un graphe et revenenat au sommet de départ;
- Cycle Eulérien : Passant une seule fois par toutes les arêtes et revenant au sommet de départ.

- Graphe Hamiltonien : Graphe qui possède au moins un cycle hamiltonien;
- Graphe Eulérien : Graphe qui possède au moins un cycle eulérien.

Trouvez un cycle eulérien dans les deux graphes suivants :

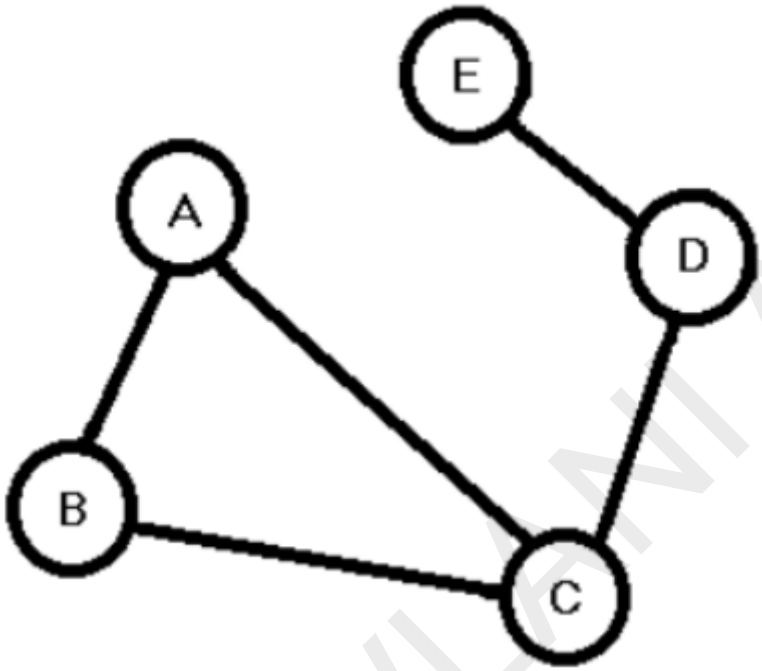




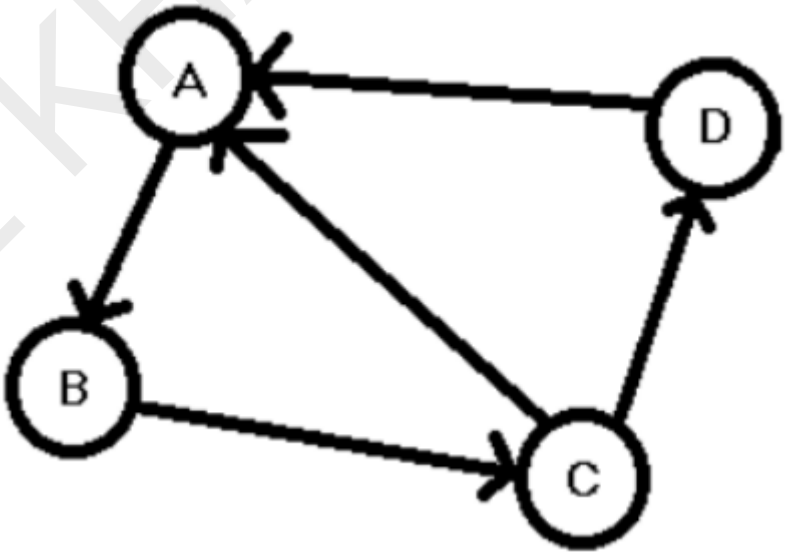
- Pour le graphe à gauche, il n'existe aucun cycle eulérien
- Pour le graphe à droite : ABECDBCA, ECDBACBE

2 Implémentation des graphes sur python

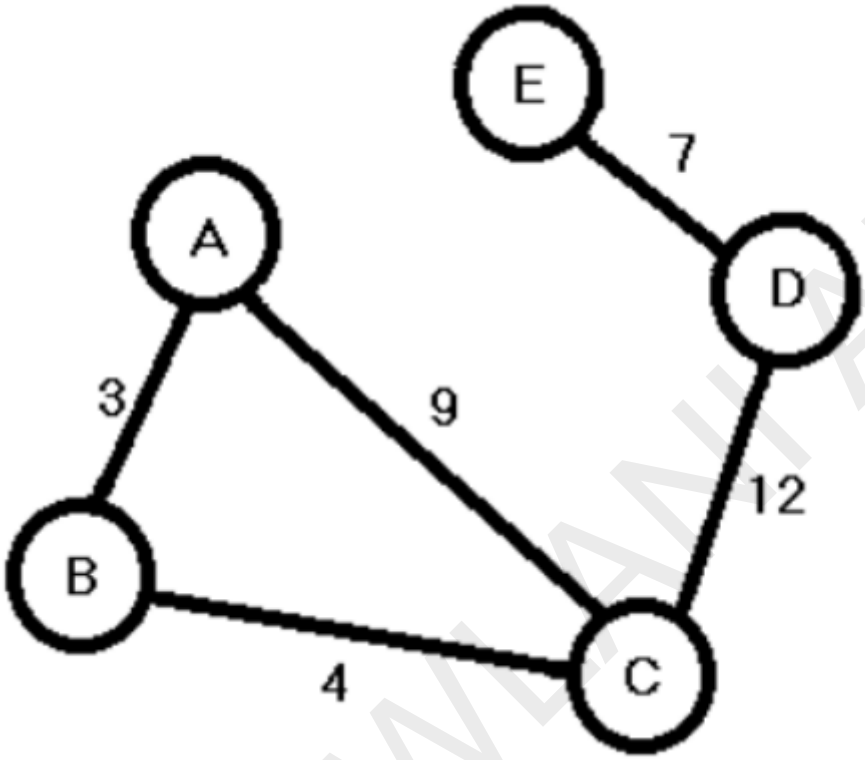
2.1 A l'aide des dictionnaires de précédences



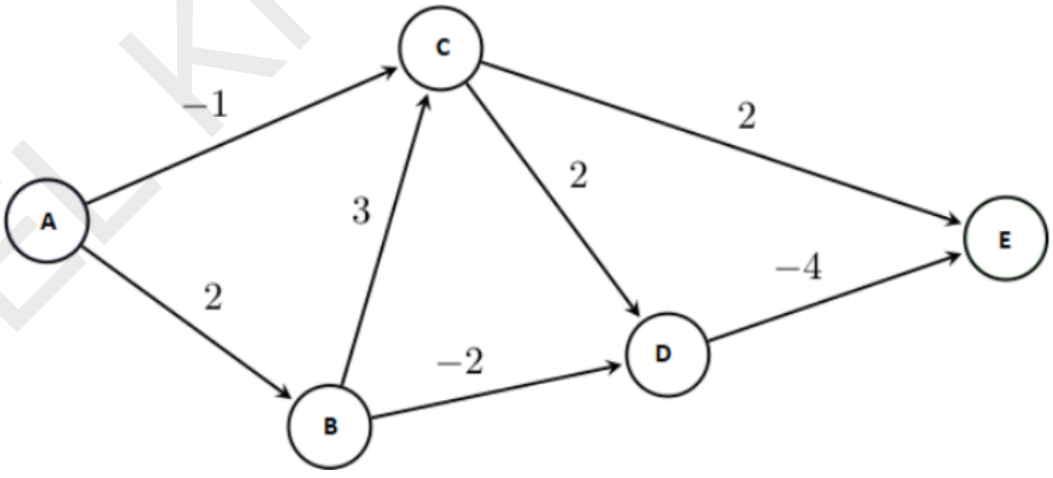
[2]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A': ['B', 'C'], 'B': ['A', 'C'], 'C': ['A', 'B', 'D'], 'D':['E', 'C'],␣
 ↪'E': ['D']}
```



[3]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A': ['B'], 'B': ['C'], 'C': ['A', 'D'], 'D': ['A']}
```



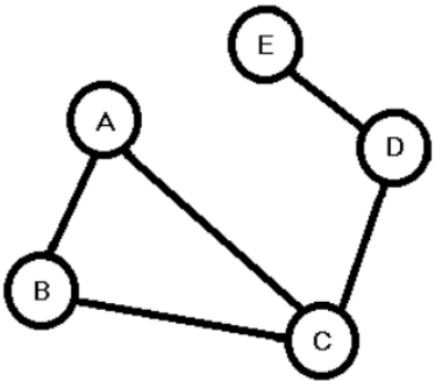
[4]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A': [(3, 'B'), (9, 'C')], 'B': [(3, 'A'), (4, 'C')], 'C': [(9, 'A'), (4,␣
↪'B'), (12, 'D')], 'D': [(12, 'C'), (7, 'E')], 'E': [(7, 'D')]}
```



[5]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
```

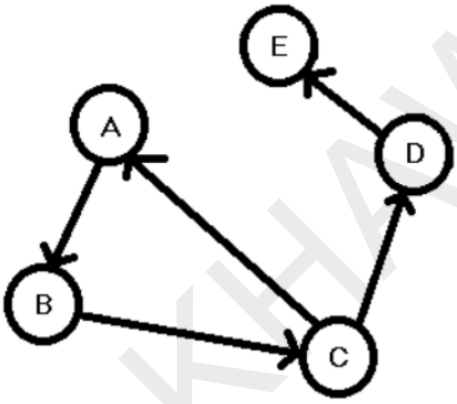
```
G = {'A' : [(-1, 'C'), (2, 'B')], 'B' : [(3, 'C'), (-2, 'D')], 'C':[(2, 'D'),
 ↪(2, 'E')], 'D':[(-4, 'E')], 'E' : []}
```

2.2 A l'aide des matrices d'adjacences



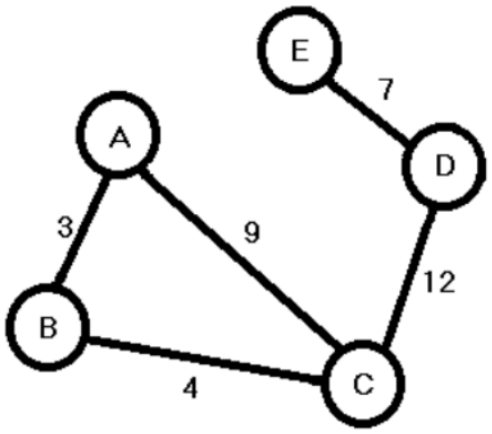
	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	0
C	1	1	0	1	0
D	0	0	1	0	1
E	0	0	0	1	0

[6]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 1, 1, 0, 0], [1, 0, 1, 0, 0], [1, 1, 0, 1, 0], [0, 0, 1, 0, 1], [0, 0,
 ↪0, 1, 0]]
```



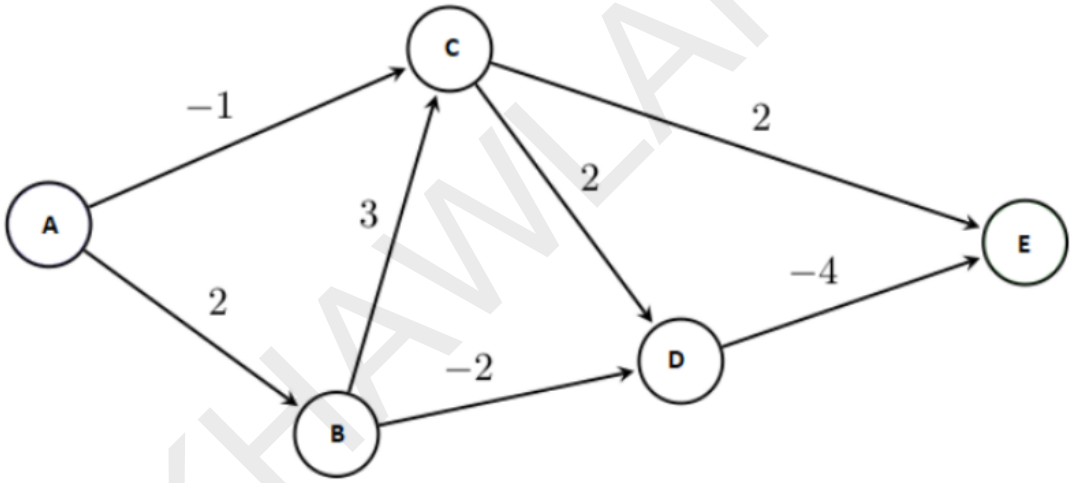
	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	1	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

[7]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 1, 0, 0, 0], [0, 0, 1, 0, 0], [1, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0,
 ↪0, 0, 0]]
```



	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

[8]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 3, 9, 0, 0], [3, 0, 4, 0, 0], [9, 4, 0, 12, 0], [0, 0, 12, 0, 7], [0,␣
 ↪0, 0, 7, 0]]
```



[9]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 2, 0, 2, 0], [0, 0, 0, 0, 0], [-1, 0, 0, 0, 2], [0, -4, 0, 0, 0], [3,␣
↪0, 0, -2, 0]]
```

3 Exercices

3.1 Exercice 1

Ecrire une fonction est_voisin(G, S1, S2) aui retourne True si S1 est un voisin de S2 dans le graphe représenté par le dictionnaire G et False Sinon. On doit ecrire la fonction pour les 2 cas de graphes suivants: - Graphe non pondere ; - Graphe pondere.

[10]:
```
# Cas d'un graphe non pondéré
def est_voisin(G, S1, S2):
  return (S2 in G[S1]) or (S1 in G[S2])
```

[11]:
```
# Cas d'un graphe pondéré
def est_voisin(G, S1, S2):
  for t in G[S1]:
    if S2 == t[1]:
      return True
  for t in G[S2]:
    if S1 == t[1]:
      return True
  return False
```

3.2 Exercice 2

Ecrire une fonction est_voisin(G, i, j) qui retourne True si le sommet d'indice i est un voisin du sommet d'indice j dans le graphe représenté par la matrice G (liste de listes) et False Sinon.

[12]:
```
def est_voisin(G, i, j):
  return (G[i][j] != 0) or (G[j][i] != 0)
```

3.3 Exercice 3

Ecrire une fonction ordre(G) qui prend en paramètre un graphe (dictionnaire ou matrice) et qui retourne l'ordre du graphe.

[13]:
```
def ordre(G):
  return len(G)
```

3.4 Exercice 4

Ecrire une fonction degre(G, s) qui retourne le degré du sommet s dans le graphe G non orienté représenté par un dictionnaire.

[14]:
```
def degre(G, s):
  return len(G[s])
```

3.5 Exercice 5

Ecrire une fonction degre(G, s) qui retourne le degre du sommet s dans le graphe G non orienté représenté par une matrice.

[15]:
```
def degre(G, s):
  L = G[s]
  d = 0
  for x in G[s]:
    if x != 0:
      d = d + 1
  return d
```

4 Parcours des graphes

4.1 Parcours en largeur

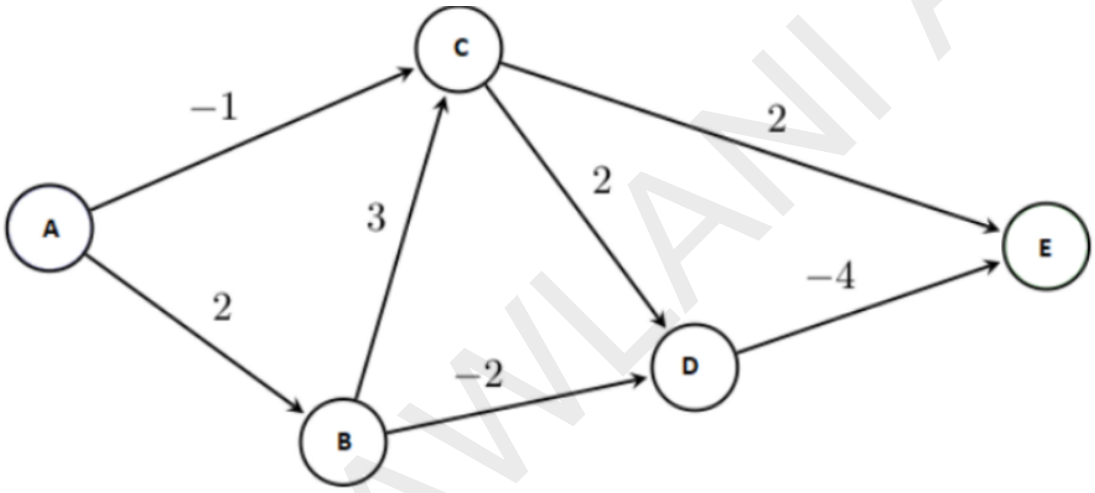
Cet algorithme permet de lister (Afficher ou stocker dans une liste) tous les sommets d'un graphe de la manière suivante :
- S est un sommet de départ - On commence par lister tous les voisins de S - On refait la meme chose pour chaque voisin de S

4.1.1 Fonction 1

Ecrire une fonction parcours_largeur1(G, d) qui prend en paramètres un graphe pondéré représenté par le dictionnaire G et un sommet de départ d. La fonction doit afficher tous les sommets du graphe

G dans l'ordre du parcours en largeur.

[16]:
```
def parcours_largeur1(G, d):
  sommets = []
  file = [d]
  while file != []:
    S = file.pop(0)
    if not S in sommets:
      print(S, end = "   ")
      sommets.append(S)
      for t in G[S]:
        if not t[1] in sommets:
          file.append(t[1])
```



[17]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A' : [(-1, 'C'), (2, 'B')], 'B' : [(3, 'C'), (-2, 'D')], 'C':[(2, 'D'),␣
 ↪(2, 'E')], 'D':[(-4, 'E')], 'E' : []}
parcours_largeur1(G, 'A')
```

```
A   C   B   D   E
```

4.1.2 Fonction 2

Ecrire une fonction parcours_largeur2(G, d) qui prend en paramètres un graphe pondéré représenté par le dictionnaire G et un sommet de depart d. La fonction doit retourner une liste qui contient tous les sommets du graphe G dans l'ordre du parcours en largeur.

[18]:
```
def parcours_largeur2(G, d):
  sommets = []
  file = [d]
  while file != []:
    S = file.pop(0)
    if not S in sommets:
```

```
      sommets.append(S)
      for t in G[S]:
        if not t[1] in sommets:
          file.append(t[1])
  return sommets
```

[19]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A' : [(-1, 'C'), (2, 'B')], 'B' : [(3, 'C'), (-2, 'D')], 'C':[(2, 'D'),␣
 ↪(2, 'E')], 'D':[(-4, 'E')], 'E' : []}
sommets = parcours_largeur2(G, 'A')
print(sommets)
```

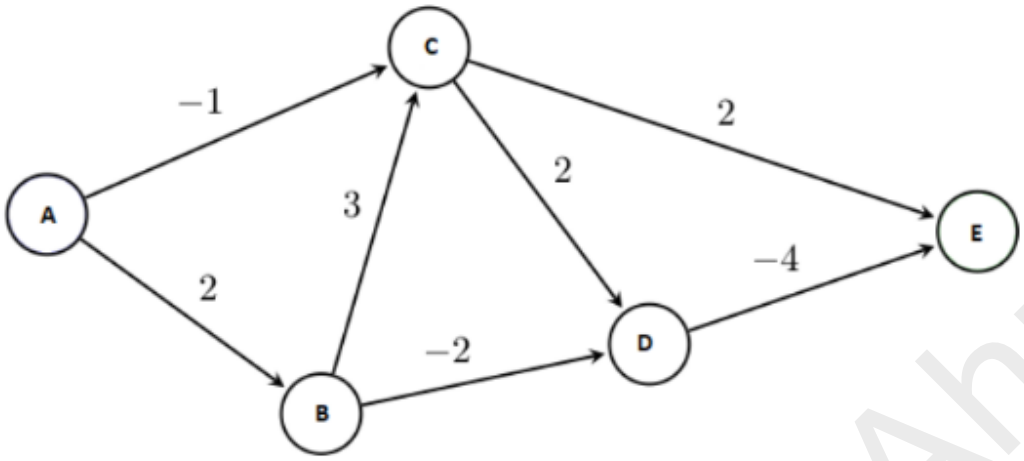
```
['A', 'C', 'B', 'D', 'E']
```

4.1.3 Fonction 3

Ecrire une fonction parcours_largeur3(G, d) qui prend en paramètres un graphe pondéré représenté par la matrice G et un sommet de depart d (d un indice). La fonction doit afficher tous les sommets (indices) du graphe G dans l'ordre du parcours en largeur. On peut definir d'abord une fonction liste_voisins(G, i) qui retourne la liste des voisins d'un sommets d'indice i.

[20]:
```
def liste_voisins(G, i):
  n = len(G[i])
  V = []
  for j in range(n):
    if G[i][j] != 0:
      V.append(j)
  return V

def parcours_largeur3(G, d):
  sommets = []
  file = [d]
  while file != []:
    S = file.pop(0)
    if not S in sommets:
      print(S, end = "   ")
      sommets.append(S)
      for v in liste_voisins(G, S):
        if not v in sommets:
          file.append(v)
```



	C	E	A	D	B
C	0	2	0	2	0
E	0	0	0	0	0
A	-1	0	0	0	2
D	0	-4	0	0	0
B	3	0	0	-2	0

[21]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 2, 0, 2, 0], [0, 0, 0, 0, 0], [-1, 0, 0, 0, 2], [0, -4, 0, 0, 0], [3,␣
 ↪0, 0, -2, 0]]
parcours_largeur3(G, 2) # 2 c'est l'indice de A
```

```
2   0   4   1   3
```

4.1.4 Fonction 4

Ecrire une fonction parcours_largeur4(G, d) qui prend en paramètres un graphe pondéré représenté par la matrice G et un sommet de depart d (d un indice). La fonction doit retourner une liste qui contient tous les sommets (indices) du graphe G dans l'ordre du parcours en largeur. On peut définir d'abord une fonction liste_voisins(G, i) qui retourne la liste des voisins d'un sommets d'indice i.

[22]:
```
def liste_voisins(G, i):
    n = len(G[i])
    V = []
    for j in range(n):
```

```
    if G[i][j] != 0:
      V.append(j)
  return V

def parcours_largeur4(G, d):
  sommets = []
  file = [d]
  while file != []:
    S = file.pop(0)
    if not S in sommets:
      sommets.append(S)
      for v in liste_voisins(G, S):
        if not v in sommets:
          file.append(v)
  return sommets
```

[23]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 2, 0, 2, 0], [0, 0, 0, 0, 0], [-1, 0, 0, 0, 2], [0, -4, 0, 0, 0], [3,␣
↪0, 0, -2, 0]]
sommets = parcours_largeur4(G, 2)
print(sommets)
```

```
[2, 0, 4, 1, 3]
```

4.2 Parcours en profondeur

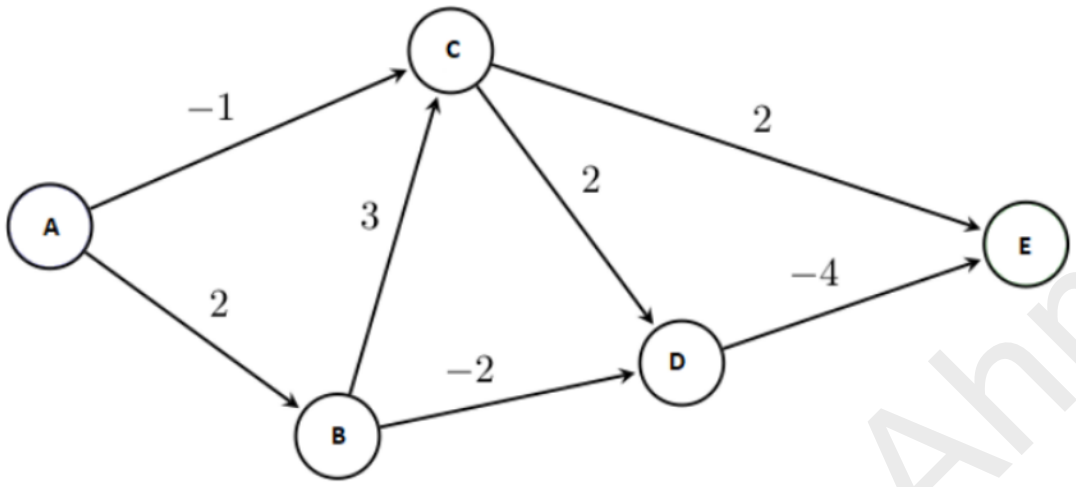
Parcours qui progresse à partir d'un sommet S en s'appelant récursivement pour chaque sommet voisin de S :

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.

4.2.1 Fonction 1

Ecrire une fonction récursive parcours_profondeur1(G, d) qui prend en paramètres un graphe pondéré représenté par le dictionnaire G et un sommet de départ d. La fonction doit stocker tous les sommets du graphe G dans une variable globale sommets initialise par une liste vide.

[24]:
```
sommets = []
def parcours_profondeur1(G, d):
  sommets.append(d)
  for t in G[d]:
    if not t[1] in sommets:
      parcours_profondeur1(G, t[1])
```



[25]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A' : [(-1, 'C'), (2, 'B')], 'B' : [(3, 'C'), (-2, 'D')], 'C':[(2, 'D'),␣
 ↪(2, 'E')], 'D':[(-4, 'E')], 'E' : []}
parcours_profondeur1(G, 'A')
print(sommets)
```

```
['A', 'C', 'D', 'E', 'B']
```

4.2.2 Fonction 2

Ecrire une fonction itérative parcours_profondeur2(G, d) qui prend en paramètres un graphe pondéré représenté par le dictionnaire G et un sommet de départ d. La fonction doit retourner la liste de tous les sommets du graphe G dans l'ordre du parcours en profondeur.

[26]:
```
def parcours_profondeur2(G, d):
  sommets = []
  pile = [d]
  while pile != []:
    S = pile.pop()
    if not S in sommets:
      sommets.append(S)
      for t in G[S]:
        if not t[1] in sommets:
          pile.append(t[1])
  return sommets
```

[27]:
```
# Le graphe ci dessus est représenté par le dictionnaire suivant
G = {'A' : [(-1, 'C'), (2, 'B')], 'B' : [(3, 'C'), (-2, 'D')], 'C':[(2, 'D'),␣
 ↪(2, 'E')], 'D':[(-4, 'E')], 'E' : []}
parcours_profondeur2(G, 'A')
print(sommets)
```

```
['A', 'C', 'D', 'E', 'B']
```

4.2.3 Fonction 3

Ecrire une fonction parcours_profondeur3(G, d) qui prend en paramètres un graphe pondéré représenté par la matrice G et un sommet de départ d (d un indice). La fonction doit retourner une liste qui contient tous les sommets (indices) du graphe G dans l'ordre du parcours en profondeur. On peut definir d'abord une fonction liste_voisins(G, i) qui retourne la liste des voisins d'un sommets d'indice i.

[28]:
```
def liste_voisins(G, i):
  n = len(G[i])
  V = []
  for j in range(n):
    if G[i][j] != 0:
      V.append(j)
  return V

def parcours_profondeur3(G, d):
  sommets = []
  pile = [d]
  while pile != []:
    S = pile.pop()
    if not S in sommets:
      sommets.append(S)
      for v in liste_voisins(G, S):
        if not v in sommets:
          pile.append(v)
  return sommets
```

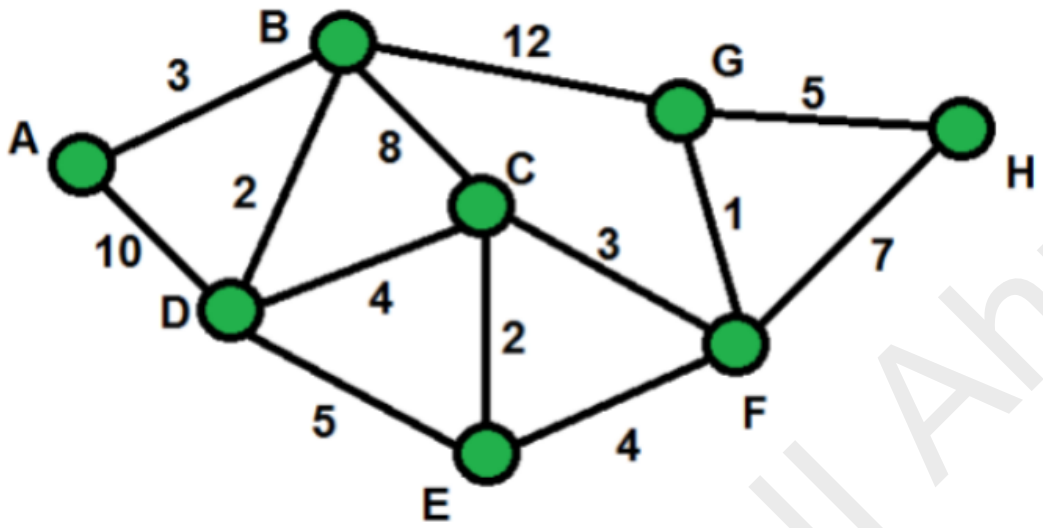
[29]:
```
# Le graphe ci dessus est représenté comme suit
G = [[0, 2, 0, 2, 0], [0, 0, 0, 0, 0], [-1, 0, 0, 0, 2], [0, -4, 0, 0, 0], [3,␣
↪0, 0, -2, 0]]
sommets = parcours_profondeur3(G, 2)
print(sommets)
```

```
[2, 4, 3, 1, 0]
```

5 Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme utilisé pour trouver le chemin le plus court entre un nœud source et tous les autres nœuds dans un graphe pondéré et dirigé.

Exemple : Trouver un chemin optimale entre A et H dans le graphe suivant:



L'algorithme de Dijkstra consiste au remplissage du tableau suivant :

	A	B	C	D	E	F	G	H
Etape 1								
Etape 2								
Etape 3								
Etape 4								
Etape 5								
Etape 6								
Etape 7								
Etape 8								
Etape 9								
Etape 10								

Règles pour remplir les cases de chaque cellule:

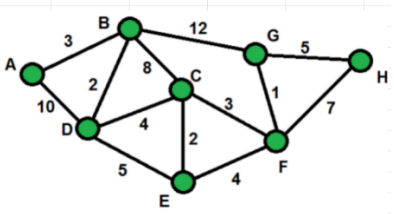
Soit dep le sommet de départ.

1. Chaque cellule contient un couple : (d(v), p(v)), où d(v) représente la distance minimale du sommet de départ dep au sommet v, et p(v) est le sommet précédent sur le chemin optimal reliant dep et v;
2. Initialisation : d(dep) est fixé à 0, car dep est le sommet de départ;
3. Initialisation : Les distances d(v) sont initialisées à l'infini (∞) tant qu'elles n'ont pas été calculées.

- Le sommet de départ, noté A, n'a pas de prédécesseur. Ainsi, le couple de la première cellule est (0,);
- Tant que les distances ne sont pas encore calculées, elles sont représentées par la valeur ∞;
- Puisque le sommet A est visité, des X sont insérés dans la colonne A pour indiquer qu'il ne sera plus revisité.

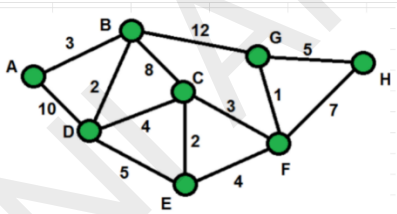
La premiere etape sera comme suit :

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X							
Etape 3	X							
Etape 4	X							
Etape 5	X							
Etape 6	X							
Etape 7	X							
Etape 8	X							
Etape 9	X							
Etape 10	X							



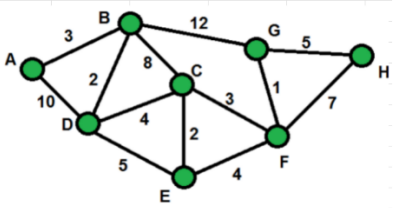
À partir du point A, deux options se présentent : visiter le sommet B avec une distance de 3 ou visiter le sommet D avec une distance de 10. Dans les deux cas, le sommet précédent est A. Les distances vers les autres sommets restent non calculées et sont symbolisées par ∞.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X							
Etape 4	X							
Etape 5	X							
Etape 6	X							
Etape 7	X							
Etape 8	X							
Etape 9	X							
Etape 10	X							



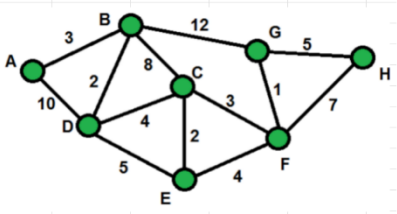
À l'étape 2, le couple contenant la distance minimale cumulée du sommet de départ est (3, A), ce qui indique que nous continuerons notre chemin à partir du sommet B. Maintenant que le sommet B est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X						
Etape 4	X	X						
Etape 5	X	X						
Etape 6	X	X						
Etape 7	X	X						
Etape 8	X	X						
Etape 9	X	X						
Etape 10	X	X						



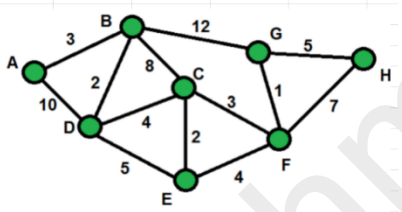
A partir du point B dont la valeur du couple est (3, A) nous pouvons parcourir : - Le sommet A a déjà été visité et contient des X; - En parcourant le sommet C avec une distance de 8, la distance cumulée sera 11. La valeur de la cellule correspondante sera (11, B) puisque B est désormais le prédécesseur; - En parcourant le sommet D avec une distance de 2, la distance cumulée sera 5, inférieure à la distance précédente pour D (10, A). Ainsi, sa valeur sera modifiée en (5, B). - En parcourant le sommet G avec une distance de 12, la distance cumulée sera 15. La valeur de la cellule correspondante sera (15, B). Les autres cellules gardent le symbole ∞.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X						
Etape 5	X	X						
Etape 6	X	X						
Etape 7	X	X						
Etape 8	X	X						
Etape 9	X	X						
Etape 10	X	X						



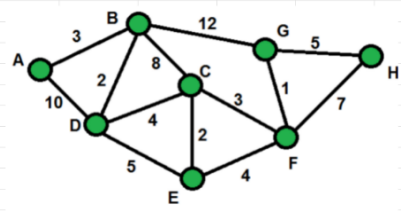
À l'étape 3, le couple contenant la distance minimale cumulée du sommet de départ est (5, B), ce qui indique que nous continuerons notre chemin à partir du sommet D. Maintenant que le sommet D est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X		X				
Etape 5	X	X		X				
Etape 6	X	X		X				
Etape 7	X	X		X				
Etape 8	X	X		X				
Etape 9	X	X		X				
Etape 10	X	X		X				



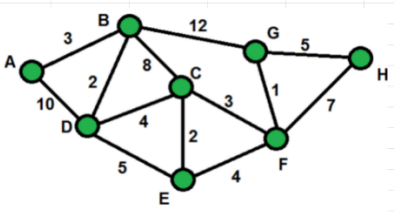
A partir du point D dont la valeur du couple est (5, B) nous pouvons parcourir : - Les sommets A et B ont déjà été visités et contiennent des X; - En parcourant le sommet C avec une distance de 4, la distance cumulée sera 9, inférieure à la distance précédente pour C qui est (11, B). Ainsi, sa valeur sera modifiée en (9, D); - En parcourant le sommet E avec une distance de 5, la distance cumulée sera 10. La valeur de la cellule correspondante sera (10, D); - Le sommet G ne peut pas être parcouru à partir de D, donc on garde l'ancienne valeur; - Les autres cellules gardent le symbole ∞.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X		X				
Etape 6	X	X		X				
Etape 7	X	X		X				
Etape 8	X	X		X				
Etape 9	X	X		X				
Etape 10	X	X		X				



À l'étape 4, le couple contenant la distance minimale cumulée du sommet de départ est (9, D), ce qui indique que nous continuerons notre chemin à partir du sommet C. Maintenant que le sommet C est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X				
Etape 6	X	X	X	X				
Etape 7	X	X	X	X				
Etape 8	X	X	X	X				
Etape 9	X	X	X	X				
Etape 10	X	X	X	X				

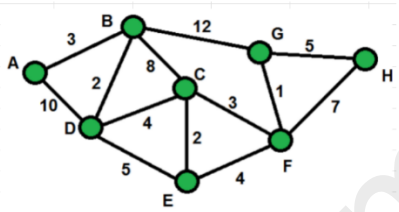


A partir du point C dont la valeur du couple est (9, D) nous pouvons parcourir :

- Les sommets B et D ont déjà été visités et contiennent des X;
- En parcourant le sommet F avec une distance de 3, la distance cumulée sera 12. Sa valeur sera modifiée par (12, C);
- En parcourant le sommet E avec une distance de 2, la distance cumulée sera 11, supérieure à la valeur (11, D) déjà existante dans E. On garde l'ancienne valeur (11, D);

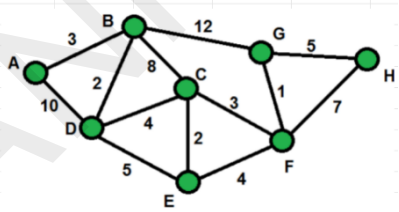
- Le sommet G ne peut pas être parcouru à partir de C, donc on garde l'ancienne valeur;
- Les autres cellules gardent le symbole ∞.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X				
Etape 7	X	X	X	X				
Etape 8	X	X	X	X				
Etape 9	X	X	X	X				
Etape 10	X	X	X	X				



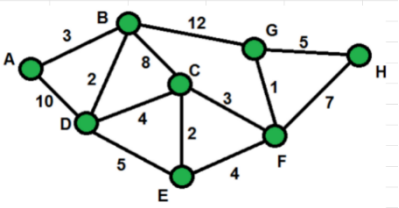
À l'étape 5, le couple contenant la distance minimale cumulée du sommet de départ est (10, D), ce qui indique que nous continuerons notre chemin à partir du sommet E. Maintenant que le sommet E est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X			
Etape 7	X	X	X	X	X			
Etape 8	X	X	X	X	X			
Etape 9	X	X	X	X	X			
Etape 10	X	X	X	X	X			



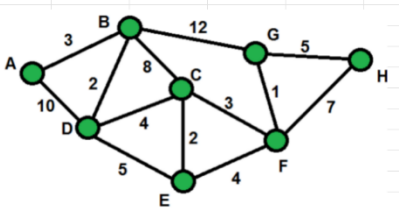
A partir du sommet E ou la valeur est (10, D) nous pouvons soit :
- Les sommets C et D ont déjà été visités et contiennent des X; - En parcourant le sommet F avec une distance de 4, la distance cumulée sera 14, supérieure à l'ancienne valeur (12, C) dans F. On garde l'ancienne valeur; - Le sommet G ne peut pas être parcouru à partir de E, donc on garde l'ancienne valeur; - Les autres cellules gardent le symbole ∞.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X			
Etape 8	X	X	X	X	X			
Etape 9	X	X	X	X	X			
Etape 10	X	X	X	X	X			



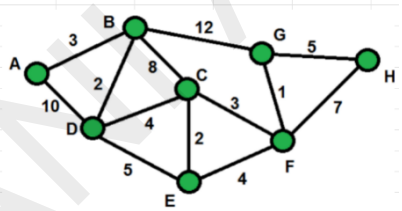
À l'étape 6, le couple contenant la distance minimale cumulée du sommet de départ est (12, C), ce qui indique que nous continuerons notre chemin à partir du sommet F. Maintenant que le sommet F est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X	X		
Etape 8	X	X	X	X	X	X		
Etape 9	X	X	X	X	X	X		
Etape 10	X	X	X	X	X	X		



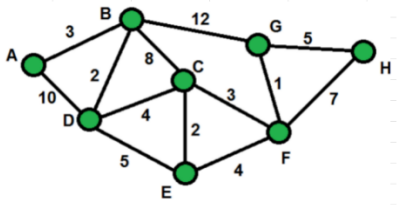
A partir du sommet F ou la valeur est (12, C) nous pouvons soit :
- Les sommets C et E ont déjà été visités et contiennent des X; - En parcourant le sommet G avec une distance de 1, la distance cumulée sera 13, inférieure à l'ancienne valeur (15, B) dans G. On change la valeur avec (13, F); - En visitant le sommet H avec une distance de 7, la distance cumulée sera 19. On met donc la valeur avec (19, F).

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X	X	(13, F)	(19, F)
Etape 8	X	X	X	X	X	X		
Etape 9	X	X	X	X	X	X		
Etape 10	X	X	X	X	X	X		



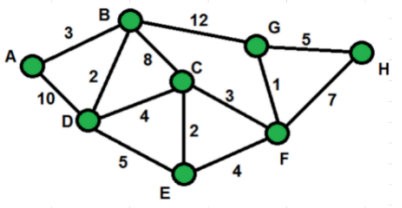
À l'étape 7, le couple contenant la distance minimale cumulée du sommet de départ est (13, F), ce qui indique que nous continuerons notre chemin à partir du sommet G. Maintenant que le sommet G est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X	X	(13, F)	(19, F)
Etape 8	X	X	X	X	X	X	X	
Etape 9	X	X	X	X	X	X	X	
Etape 10	X	X	X	X	X	X	X	



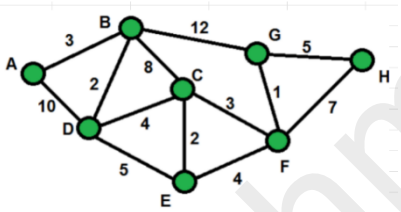
A partir du sommet G ou la valeur est (13, F) nous pouvons soit :
- Les sommets F et B ont déjà été visités et contiennent des X; - En parcourant le sommet H avec une distance de 5, la distance cumulée sera 18, inférieure à l'ancienne valeur (19, F) dans G. On change la valeur avec (18, G);

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X	X	(13, F)	(19, F)
Etape 8	X	X	X	X	X	X	X	(18, G)
Etape 9	X	X	X	X	X	X	X	
Etape 10	X	X	X	X	X	X	X	



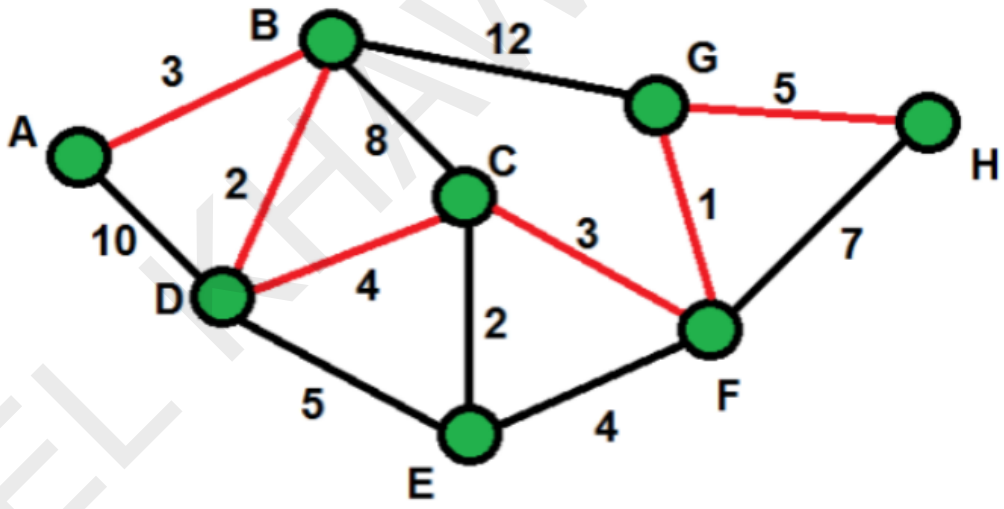
À l'étape 8, le couple contenant la distance minimale cumulée du sommet de départ est (18, G), ce qui indique que nous continuerons notre chemin à partir du sommet H. Maintenant que le sommet H est visité, nous devons mettre des X dans la colonne correspondante pour indiquer qu'il ne sera plus revisité.

	A	B	C	D	E	F	G	H
Etape 1	(0,)	∞	∞	∞	∞	∞	∞	∞
Etape 2	X	(3, A)	∞	(10, A)	∞	∞	∞	∞
Etape 3	X	X	(11, B)	(5, B)	∞	∞	(15, B)	∞
Etape 4	X	X	(9, D)	X	(10, D)	∞	(15, B)	∞
Etape 5	X	X	X	X	(10, D)	(12, C)	(15, B)	∞
Etape 6	X	X	X	X	X	(12, C)	(15, B)	∞
Etape 7	X	X	X	X	X	X	(13, F)	(19, F)
Etape 8	X	X	X	X	X	X	X	(18, G)
Etape 9	X	X	X	X	X	X	X	X
Etape 10	X	X	X	X	X	X	X	X



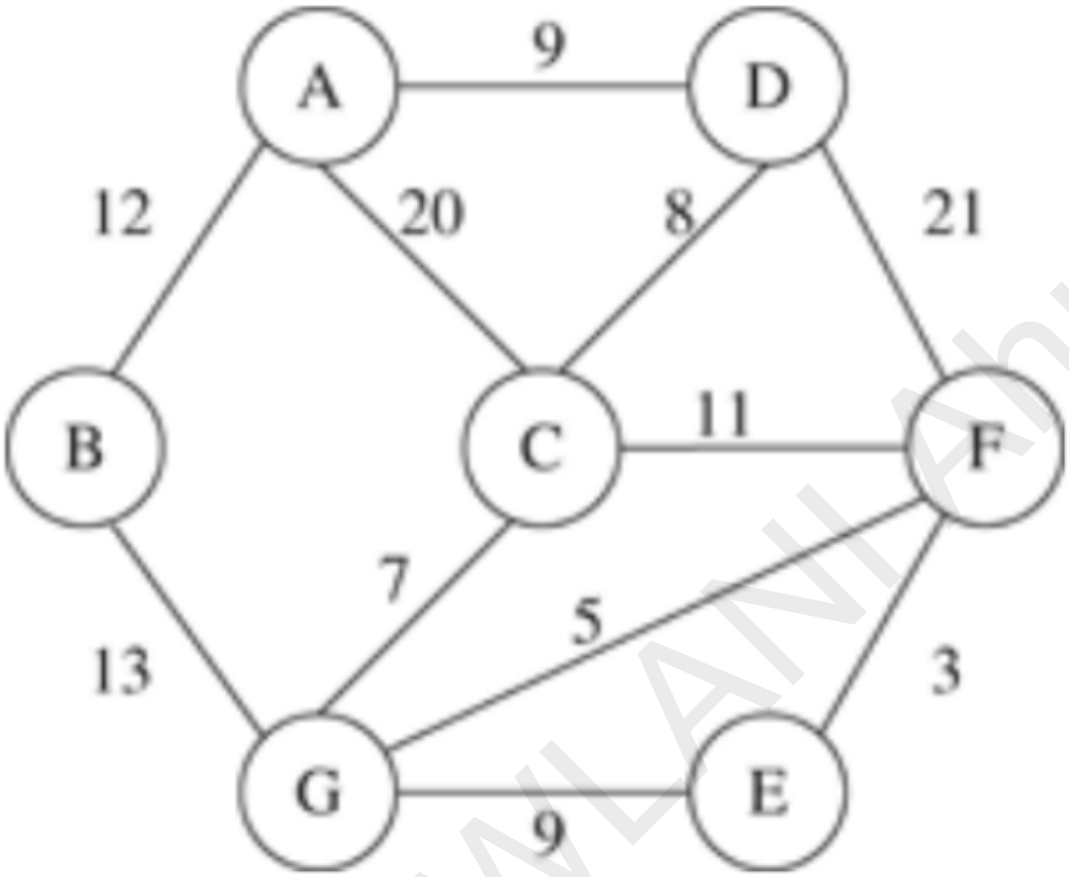
Le sommet H est visite l'algorithme s'arrete, la distance minimale entre A et G est 18. Pour constituer le chemin le plus cours :

- Nous commençons par le sommet d'arrivé H;
- Son prédécesseur est G, tel qu'indiqué dans le tableau;
- Le prédécesseur de G est F;
- Le prédécesseur de F est C;
- Le prédécesseur de C est D;
- Le prédécesseur de D est B;
- Enfin, le prédécesseur de B est A, qui est le sommet de départ. Le chemin le plus court est donc A - B - D - C - F - G - H.



5.1 Exercice 1

Trouver le chemin entre le sommet A et le sommet E dans le graphe suivant.



	A	B	C	D	E	F	G
Etape 1	(0,.)	∞	∞	∞	∞	∞	∞
Etape 2	X	(12, A)	(20, A)	(9, A)	∞	∞	∞
Etape 3	X	(12, A)	(17, D)	X	∞	(30, D)	∞
Etape 4	X	X	(17, D)	X	∞	(30, D)	(25, B)
Etape 5	X	X	X	X	∞	(28, C)	(24, C)
Etape 6	X	X	X	X	(33, G)	(28, C)	X
Etape 7	X	X	X	X	(31, F)	X	X

Le chemin optimal entre A et E	A	D	C	F	E

La distance optimal entre A et E est 31

5.2 Exercice 2

Ecrire sur python une fonction Dikjstra(G, d, f) qui prend en parametre un graphe G pondere represente par un dictionnaire de precedence, un sommet de depart d et un sommet d'arrive f. La fonction retourne le chemin le plus cours entre d et f sous forme de liste ainsi que la distance

minimale.

On commence par écrire une fonction qui prend en paramètre une liste de tuples et qui retourne l'indice du tuple qui contient la distance (le premier element du tuple) minimale.

Exemple de la liste [(6, 'A'), (5, 'B'), (17, 'C'), (4, 'D')]. Dans ce cas la fonction doit retourner l'indice du tuple (4, 'D')

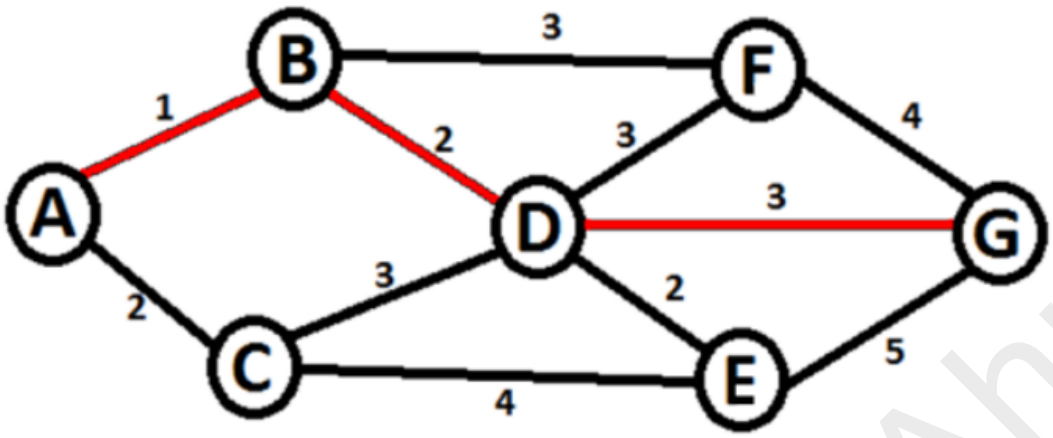
[30]:
```
def indice_tuple_min(L):
  k = 0
  for i in range(len(L)):
    if L[i][0] < L[k][0]:
      k = i
  return k

def Dijkstra(G, d, f):
  D = {d: 0}   # Dictionnaire des distances minimales cumulees
  P = {} # Dictionnaire des precedents
  visite = [] # Liste des sommets visites
  file = [(0, d)] # Une file qui contient les sommets a traiter sous forme␣
 ↪(distance cumulee, sommet actuel)

  while (file != []) and (f not in visite):
    i = indice_tuple_min(file)
    ds, s = file.pop(i)
    voisins = G[s]

    for v in voisins:
      dv, sv = v
      if not sv in visite:
        dn = ds + dv
        if (not sv in D) or (D[sv] > dn):
          D[sv] = dn
          P[sv] = s
          file.append((dn, sv))
    visite.append(s)

  chemin = [f]
  x = f
  while x != d:
    x = P[x]
    chemin.insert(0, x)
  return chemin, D[f]
```



[31]:
```
# Le graphe ci dessus est represente par le dictionnaire suivant
G = {'A': [(1, 'B'), (2, 'C')], 'B': [(1, 'A'), (2, 'D'), (3, 'F')], 'C': [(3,␣
 ↪'D'), (2, 'A'), (4, 'E')],
     'D': [(2, 'B'), (3, 'C'), (2, 'E'), (3, 'F'), (3, 'G')], 'E': [(4, 'C'),␣
 ↪(5, 'G'), (2, 'D')],
     'F': [(3, 'B'), (3, 'D'), (4, 'G')], 'G': [(4, 'F'), (5, 'E'), (3, 'D')]}

chemin_A_G = Dijkstra(G, 'A', 'G')
print(chemin_A_G)

chemin_A_F = Dijkstra(G, 'A', 'F')
print(chemin_A_F)

chemin_A_E = Dijkstra(G, 'A', 'E')
print(chemin_A_E)
```

```
(['A', 'B', 'D', 'G'], 6)
(['A', 'B', 'F'], 4)
(['A', 'B', 'D', 'E'], 5)
```

6 Algorithme de A*

6.1 Notion d'heuristique

En général, une heuristique est une méthode ou une technique qui fournit une solution approchée à un problème, souvent utilisée lorsque des méthodes exactes prennent trop de temps pour produire une solution. Le terme "heuristique" vient du grec "heuriskein", qui signifie "trouver" ou "découvrir".

Une heuristique est une approche pratique pour résoudre des problèmes en fournissant des solutions approximatives qui sont souvent satisfaisantes dans des conditions où une solution exacte n'est pas possible ou pratique à obtenir.

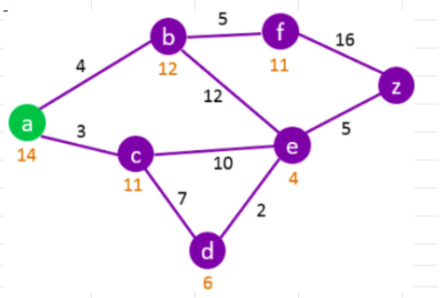
6.2 Algorithme de A*

L'algorithme A* est un algorithme de recherche de chemin qui trouve un chemin optimal entre un nœud de départ et un nœud d'arrivée dans un graphe pondéré, en utilisant une heuristique pour guider la recherche. Voici comment cela fonctionne :

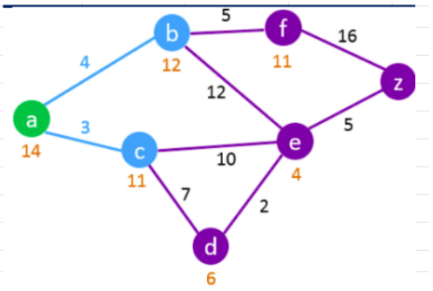
- Initialisation: L'algorithme commence par initialiser deux ensembles de nœuds : un ensemble ouvert et un ensemble fermé. Le nœud de départ est placé dans l'ensemble ouvert.
- Sélection du nœud: À chaque itération, l'algorithme sélectionne le nœud dans l'ensemble ouvert avec le coût total le plus bas, qui est la somme du coût du chemin depuis le nœud de départ et d'une estimation du coût restant (l'heuristique) jusqu'au nœud d'arrivée.
- Expansion des nœuds: Le nœud sélectionné est retiré de l'ensemble ouvert et ajouté à l'ensemble fermé. Ensuite, ses nœuds voisins sont examinés. Pour chaque voisin, l'algorithme calcule le coût total en tenant compte du coût actuel pour atteindre ce voisin depuis le nœud de départ et de l'estimation du coût restant (l'heuristique) jusqu'à l'arrivée. Si ce coût total est inférieur à tout coût précédemment calculé pour ce voisin, le chemin pour atteindre ce voisin est mis à jour et ce voisin est ajouté à l'ensemble ouvert.
- Arrêt: L'algorithme s'arrête lorsque le nœud d'arrivée est ajouté à l'ensemble fermé, ou lorsque l'ensemble ouvert est vide, ce qui signifie qu'il n'y a pas de chemin possible entre le nœud de départ et le nœud d'arrivée.
- 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 : - 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, cette valeur represente l'heuristique; - f(n) : est le coût total estimé pour aller d'un état initial vers un état final en passant par n : f(n) = g(n) + h(n).

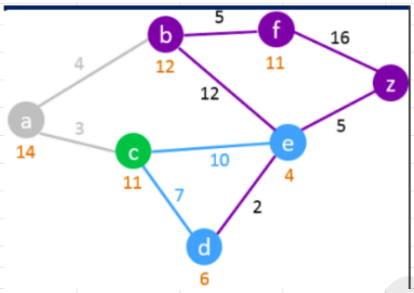
Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ouvert	0	14	14	-
b		∞	12		
c		∞	11		
d		∞	6		
e		∞	4		
f		∞	11		
z		∞	0		



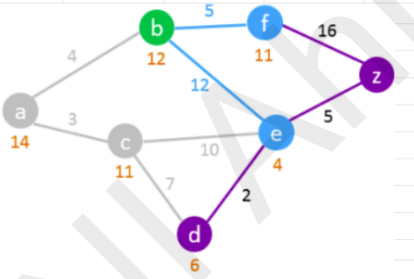
Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ouvert	4	12	16	A
c	Ouvert	3	11	14	A
d		∞	6		
e		∞	4		
f		∞	11		
z		∞	0		



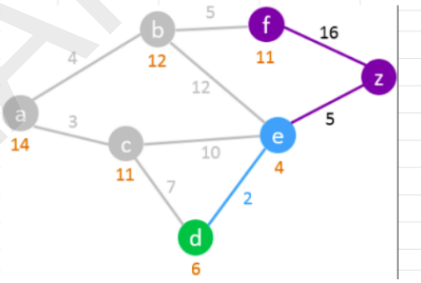
Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ouvert	4	12	16	A
c	Ferme	3	11	14	A
d	Ouvert	10	6	16	C
e	Ouvert	13	4	17	C
f		∞	11		
z		∞	0		



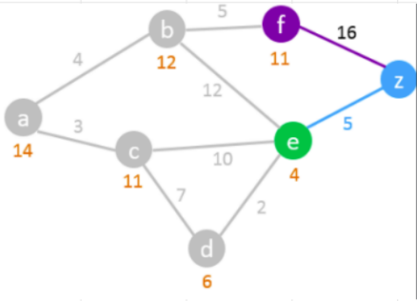
Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ferme	4	12	16	A
c	Ferme	3	11	14	A
d	Ouvert	10	6	16	C
e	Ouvert	13	4	17	C
f	Ouvert	9	11	20	B
z		∞	0		



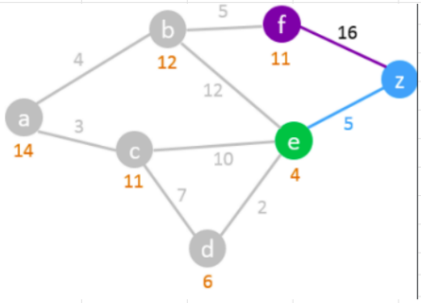
Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ferme	4	12	16	A
c	Ferme	3	11	14	A
d	Ferme	10	6	16	C
e	Ouvert	12	4	16	D
f	Ouvert	9	11	20	B
z		∞	0		



Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ferme	4	12	16	A
c	Ferme	3	11	14	A
d	Ferme	10	6	16	C
e	Ferme	12	4	16	D
f	Ouvert	9	11	20	B
z	Ouvert	17	0	17	E



Noeud	Ouvert / Ferme	g(n)	h(n)	f(n)	Precedent
a	Ferme	0	14	14	-
b	Ferme	4	12	16	A
c	Ferme	3	11	14	A
d	Ferme	10	6	16	C
e	Ferme	12	4	16	D
f	Ouvert	9	11	20	B
z	Ferme	17	0	17	E



- La distance optimale entre a et z : 17;

- Le chemin le plus court : a c d e z

6.2.1 Exercice

Ecrire une fonction astar(G, d, f, heuristique) qui prend en parametre :
- Un graphe pondere G represente par un dictionnaire; - Sommet de depart d; - Sommet d'arrivee f; - heuristique : un dictionnaire qui associe a chaque sommet du graphe G une distance estimee le reliant au sommet f.

La fonction retourne le chemin le plus court entre d et f.

[32]:
```
def minimum_f(ouvert, fh):
  m = ouvert[0]
  for x in ouvert:
    if fh[x] < fh[m]:
      m = x
  return m


def astar(G, d, f, heuristique):
    ouvert = [d]
    precedent = {}
    gh = {d: 0}
    fh = {d: heuristique[d]}
    actuel = d
    while ouvert != [] and actuel != f:
      ouvert.remove(actuel)
      for t in G[actuel]:
        tentative_g = gh[actuel] + t[0]
        if (t[1] not in gh) or tentative_g < gh[t[1]]:
          precedent[t[1]] = actuel
          gh[t[1]] = tentative_g
          fh[t[1]] = gh[t[1]] + heuristique[t[1]]
          if t[1] not in ouvert:
            ouvert.append(t[1])
      actuel = minimum_f(ouvert, fh)

    chemin = [f]
    x = f
    while actuel != d:
      x = precedent[x]
      chemin.insert(0, x)
    return chemin
```