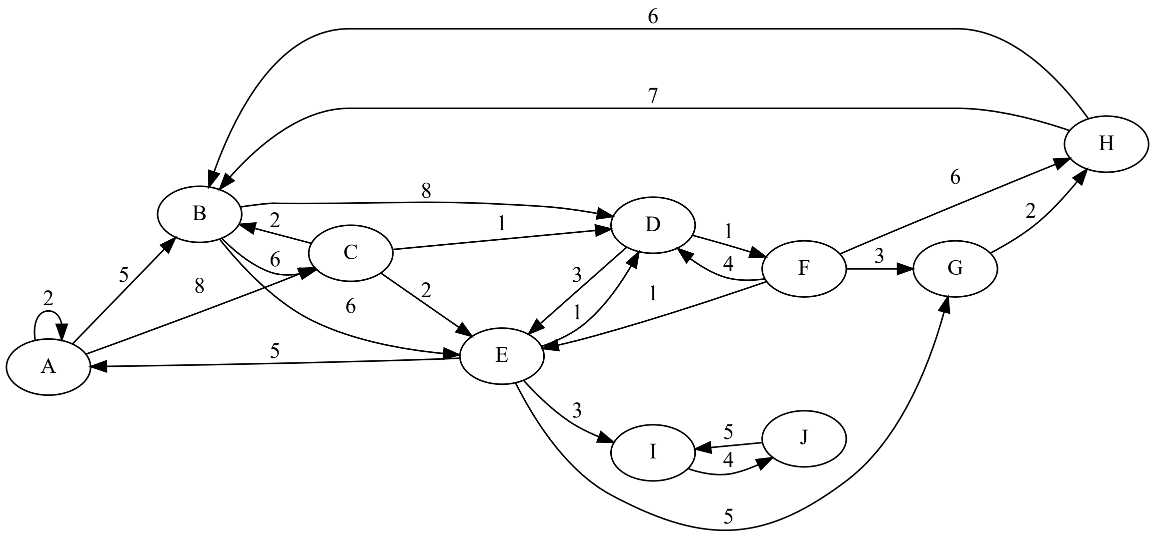
Projet du Module 104 - Informatique (LDD3 SPI)

Notes explicatives par Bruno DENIS, ENS Paris-Saclay

Sommaire

Objectifs

- **Modéliser** et **traduire en POO** (Programmation Orientée Objets) l'architecture d'un module logiciel à l'aide d'un modèle de classes UML
- **Coder** un algorithme existant de la théorie des graphes
- Interagir avec des données structurées représentées dans un fichier au format texte
    - en écriture : **générer un fichier** au format DOT pour Graphviz et au format HTML pour afficher les résultats
    - en lecture : **parser un fichier** au format ad hoc décrivant un graphe



Définition du projet

Etapes proposées

- Étape 1 - Établir le diagramme des classes pour un graphe orienté pondéré
- Étape 2
  - Coder en Python votre diagramme de classes
  - Prévoir une fonction d'affichage (format texte) pour contrôler la bonne gestion des données
- Étape 3 - Étudier l’algorithme de Dijkstra pour identifier :
  - les éléments manipulés
  - les données à stocker
- Étape 4 - Implanter l’algorithme de Dijkstra en Python
- Étape 5
  - Afficher le tableau des distances entre 2 sommets (dans la console)
  - Afficher le plus long des plus courts chemins

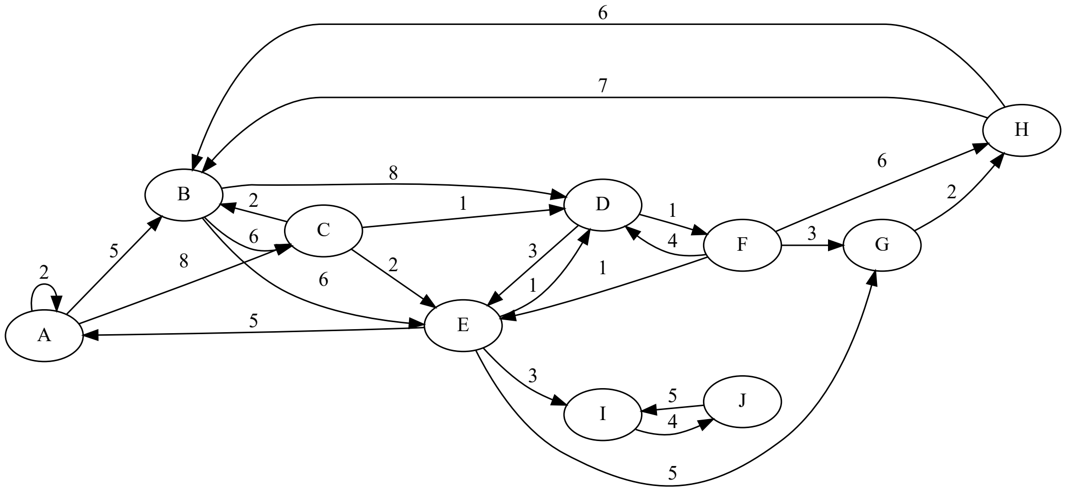
- Étape 6 - Enrichir votre programme pour des résultats au format HTML
- Étape 7 - Enrichir votre programme pour intégrer une représentation graphique du graphe
- Étape 8 - Enrichir votre programme pour que la définition du graphe se fasse depuis un fichier texte

Données de départ

Les données de départ sont disponibles suivant 2 formats :

- Un tuple Python à copier dans le fichier contenant votre programme

In [ ]:
```python
donnees = ('GrapheDeTest', ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
           [('A', 'A', 2), ('A', 'B', 5), ('A', 'C', 8), ('B', 'C', 6),
            ('B', 'D', 8), ('B', 'E', 6), ('C', 'B', 2), ('C', 'D', 1),
            ('C', 'E', 2), ('D', 'E', 3), ('D', 'F', 1), ('E', 'A', 5),
            ('E', 'D', 1), ('E', 'G', 5), ('F', 'D', 4), ('F', 'E', 1),
            ('F', 'G', 3), ('F', 'H', 6), ('E', 'I', 3), ('G', 'H', 2),
            ('H', 'B', 6), ('H', 'B', 7), ('I', 'J', 4), ('J', 'I', 5)])
```



- Un texte devant faire l'objet d'une analyse syntaxique (application du Cours 3 : Analyse syntaxique d'un texte)

```
<GRAPHE Name="GrapheDeTest" >

# Exemple de structure

<SOMMETS>
# Commentaire qui
  A ;  B ; C ; D ; E ; F ;
  G ; H ; I ; J ;
</SOMMETS>

<ARCS>
  A : A : 2 ;  A : B : 5 ;  A : C : 8 ;  B : C : 6 ;  B : D : 8 ;
  B : E : 6 ;  C : B : 2 ;  C : D : 1 ;  C : E : 2 ;  D : E : 3 ;  D : F : 1 ;
  E : A : 5 ;  E : D : 1 ;  E : G : 5 ;  F : D : 4 ;  F : E : 1 ;  F : G : 3 ;
  H : B : 6 ;  H : B : 7 ;  I : J : 4 ;  J : I : 5 ;
</ARCS>

</GRAPHE>
```

Présentation des résultats

La présentation des résultats obtenus fait partie des attentes de votre programme. Deux modes d'affichage sont possibles, un en version textuelle dans la console (partie obligatoire), l'autre sous la forme d'une page HTML (partie optionnelle),

- Affichage d'un texte dans la console (partie obligatoire),

```
Contenu du graphe 'GrapheDeTest' :
- 10 sommets : A, B, C, D, E, F, G, H, I, J
- 24 arcs    : (A,A,2), (A,B,5), (A,C,8), (B,C,6), (B,D,8), (B,E,6), (C,B,2), (C,D,1),
(C,E,2), (D,E,3), (D,F,1), (E,A,5), (E,D,1), (E,G,5), (F,D,4), (F,E,1), (F,G,3),
(F,H,6), (E,I,3), (G,H,2), (H,B,6), (H,B,7), (I,J,4), (J,I,5)

Calcul des plus courts chemins entre tous les sommets :
```

Tableau des distances :

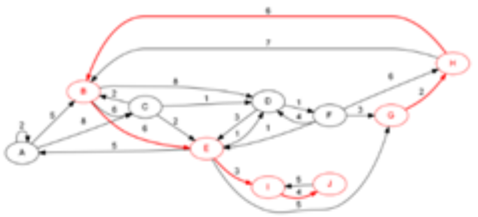
	A	B	C	D	E	F	G	H	I	J
A	-	5	8	9	10	10	13	15	13	17
B	11	-	6	7	6	8	11	13	9	13
C	7	2	-	1	2	2	5	7	5	9
D	7	12	15	-	2	1	4	6	5	9
E	5	10	13	1	-	2	5	7	3	7
F	6	11	14	2	1	-	3	5	4	8
G	19	8	14	15	14	16	-	2	17	21
H	17	6	12	13	12	14	17	-	15	19
I	-	-	-	-	-	-	-	-	-	4
J	-	-	-	-	-	-	-	-	5	-

Plus long des plus courts chemins :

- Entre G et J : distance de 21
  - G -> H -> B -> E -> I -> J

- Génération d'une page HTML (partie optionnelle)

Graphe étudié : Graphe1



Le plus long des plus courts chemins entre deux sommets est représenté en rouge.

Plus long des plus courts chemins : entre G et J de distance 21

Arcs contenus dans le chemin :

- Origine : **G**, Extrémité : **H**, Poids : **2**
- Origine : **H**, Extrémité : **B**, Poids : **6**
- Origine : **B**, Extrémité : **E**, Poids : **6**
- Origine : **E**, Extrémité : **I**, Poids : **3**
- Origine : **I**, Extrémité : **J**, Poids : **4**

Distance minimale entre les sommets :

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A	-	5	8	9	10	10	13	15	13	17
B	11	-	6	7	6	8	11	13	9	13
C	7	2	-	1	2	2	5	7	5	9
D	7	12	15	-	2	1	4	6	5	9
E	5	10	13	1	-	2	5	7	3	7
F	6	11	14	2	1	-	3	5	4	8
G	19	8	14	15	14	16	-	2	17	**21**
H	17	6	12	13	12	14	17	-	15	19
I	-	-	-	-	-	-	-	-	-	4
J	-	-	-	-	-	-	-	-	5	-

Distance minimale entre les sommets

Évaluation

L'évaluation portera essentiellement sur le programme obtenu à l'issue des 3 séances de 4 heures planifiées. Il sera également évalué le diagramme de classes établi lors de la première séance.

Cette évaluation tiendra également compte de votre investissement durant ces séances afin de tenir compte de vos progrès.

Notation de la partie obligatoire (12 pts)

La notation de la partie obligatoire tient compte de :

- du diagramme de classe présenté,
- des commentaires présents dans le code,
- du nommage des variables,
- du respect des concepts python,
- de votre implémentation de l'algorithme de Dijkstra,
- de votre capacité à identifier le plus long des plus courts chemins,
- de votre capacité à représenter sous forme tabulaire les différentes distances entre les sommets.

Notation des parties optionnelles

La notation de la **partie optionnelle** tient compte de votre capacité à :

- représenter sous forme d'une page HTML les différentes distances entre les sommets,
- générer la représentation graphique du graphe (en utilisant le module graphviz pour Python),
- utiliser l'outil pyparsing pour collecter les données depuis la description textuelle proposée.

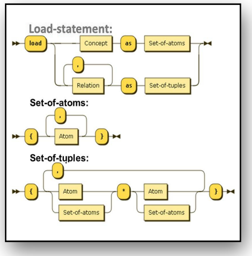
Éclairages sur certains aspects du projet

PYTHON

UNIFIED MODELING LANGUAGE™

GrapViz

HTML

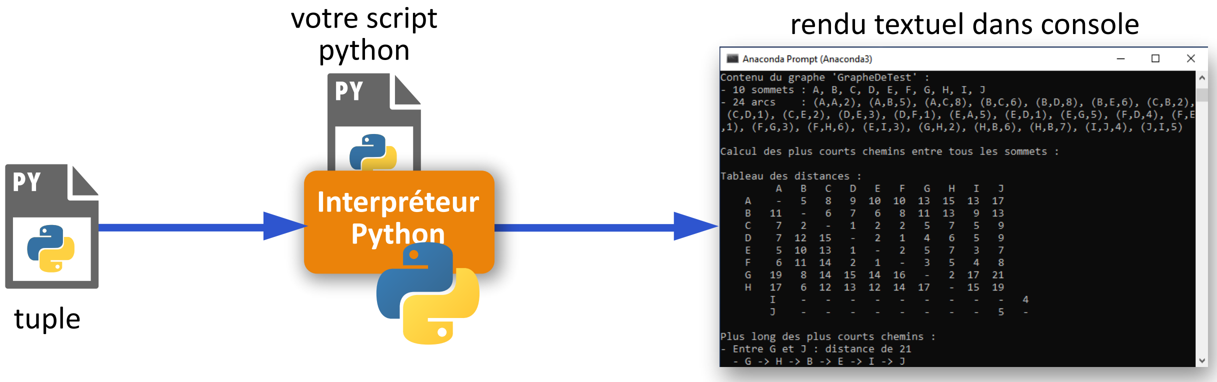


Grammaire

Flux de travail (workflow) attendu du script Python

Flux de travail à sortie textuelle

C'est le résultat minimum attendu dans la partie dite "obligatoire" où *votre script Python* traite en entrée le tuple définissant le graphe pour afficher ses résultats d'analyse dans la *console* sous la forme de texte tabulé.



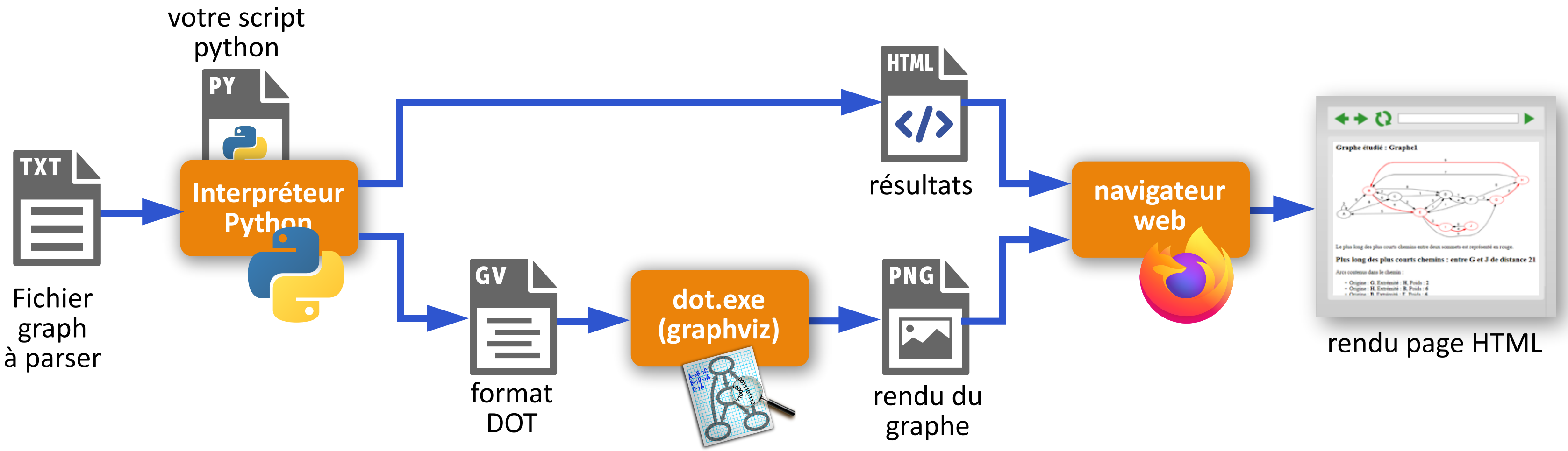
Structure attendue du script Python pour un flux de travail à sortie textuelle

In [ ]:

```
# définition des classes et de leur méthodes
class Sommet(objet):
    ...
class Arc(objet):
    ...
class Graphe(objet):
    ...

if __name__ == "__main__":
    # Si ce script est appellé directement, alors la suite est exécutée ; si il est
    # appellé comme un module depuis un autre script la suite n'est pas exécutée

    donnees = ('Graphe1', ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
               [('A', 'A', 2), ('A', 'B', 5), ('A', 'C', 8), ('B', 'C', 6),
                ('B', 'D', 8), ('B', 'E', 6), ('C', 'B', 2), ('C', 'D', 1),
                ('C', 'E', 2), ('D', 'E', 3), ('D', 'F', 1), ('E', 'A', 5),
                ('E', 'D', 1), ('E', 'G', 5), ('F', 'D', 4), ('F', 'E', 1),
                ('F', 'G', 3), ('F', 'H', 6), ('E', 'I', 3), ('G', 'H', 2),
                ('H', 'B', 6), ('H', 'B', 7), ('I', 'J', 4), ('J', 'I', 5)])

    # instantiation de l'objet 'g' à partir de la classe 'Graphe'
    g = Graphe()
    # Initiasation du graphe 'g' à partir du tuple 'donnees'
    g.initialiser_depuis_un_tuple_python(donnees)
    # Affichage d'un rapport textuel sur les chemins les plus courts
    g.generer_rapport_texte()
```

Flux de travail à sortie enrichie

C'est le résultat attendu dans la partie dite "optionnelle" où votre script Python analyse un fichier texte définissant le graphe pour construire une page HTML contenant ses résultats d'analyse afin qu'un navigateur WEB puisse le visualiser sous la forme d'un texte enrichi de graphiques et de tableaux.



La génération du fichier au format DOT et l'exécution de `dot.exe` pour obtenir le rendu graphique du graphe peuvent être directement gérées par le module python `graphviz`

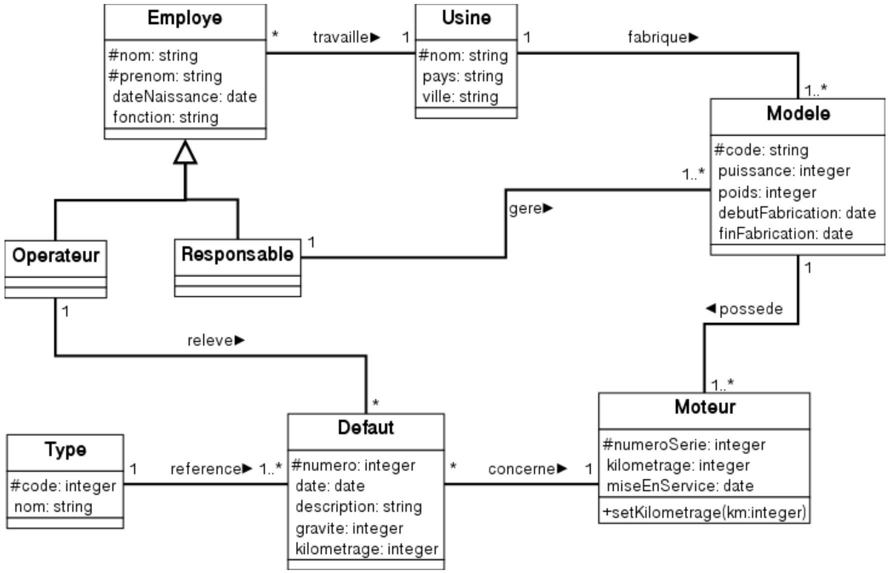
Structure attendue du script Python pour un flux de travail à sorties graphiques

In [ ]:

```python
# importation des modules nécessaires à votre code
import graphviz

# définition des classes et de leur méthodes
class Sommet():
    pass
class Arc():
    pass
class Graphe():
    pass

if __name__ == "__main__":
    # Si ce script est appellé directement, alors la suite est exécutée ; si il est
    # appellé comme un module depuis un autre script la suite n'est pas exécutée

    # Instantiation de l'objet 'g' à partir de la classe 'Graphe'
    g = Graphe()
    # Initiasation du graphe 'g' à partir de l'analyse lexicale du fichier 'nomfichier'
    g.initialiser_depuis_un_fichier('nomfichier')
    # Génération d'un rapport HTML sur les chemins les plus courts avec visu du graphe
    g.generer_rapport_html()
```

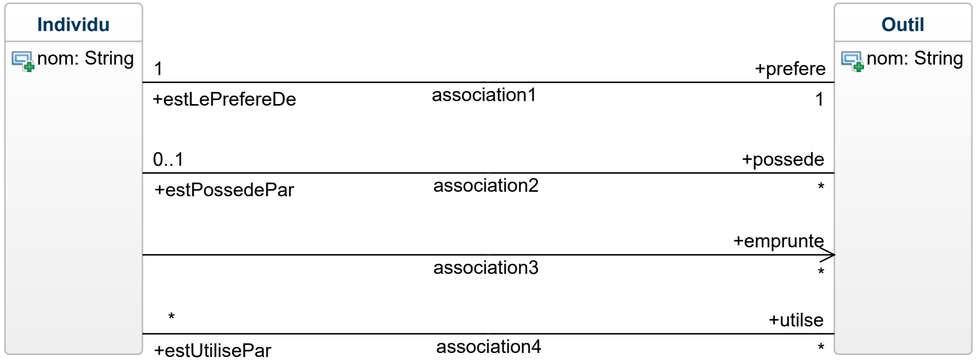
Du diagramme de classes UML au code Python

UML (Unified Modeling Language) est un **langage de modélisation** qui permet de spécifier et documenter les développements de logiciels orientés objet. Le **diagramme de classe**, souvent considéré comme l'élément central d'UML, permet dans le cadre de ce projet de modéliser la struture de notre programme codé avec le **langage de programmation** Python.



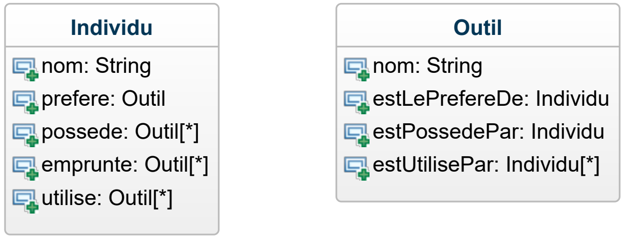
Modèle conceptuel avec un diagramme de classes

Un modèle conceptuel est élaboré très tôt dans la chronologie d'un projet, il favorise la documentation et les échanges entre les intervenants. Il met particulièrement l'accent sur les associations qui mettent en relation les classes d'objets.



Modèle d'implantation (avec un diagramme de classes pour les besoins de l'exemple)

Plus proche du codage en POO dont les langages n'intègrent pas explicitement les associations, il est construit à partir du modèle conceptuel en transformant les associations en attributs. Exemple :



Interprétations des associations

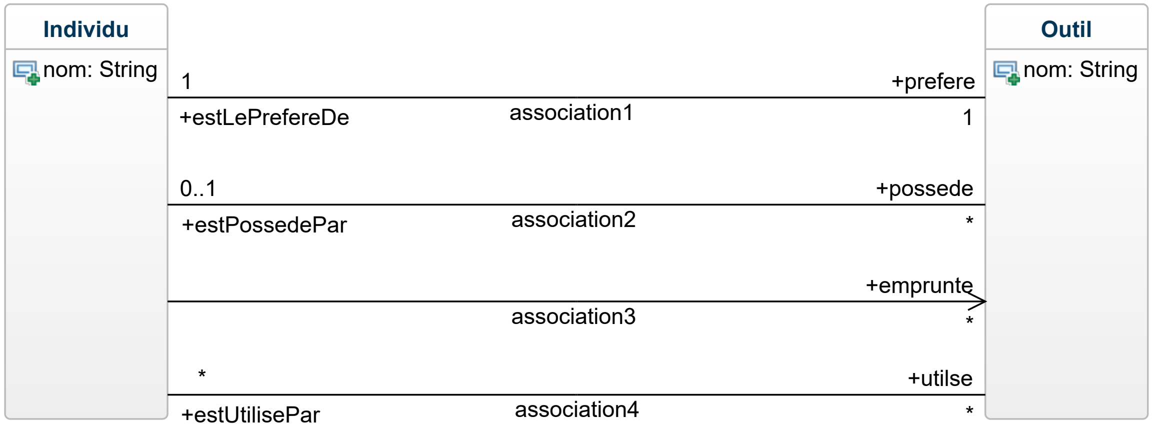
Le **modèle conceptuel** avec un diagramme de classes ci-dessous propose 4 associations différentes (modèle réalisé avec l'éditeur en ligne gratuit GenMyModel pouvant également générer la trame d'un code Python) :

Les **terminaisons d'association** sont porteuses d'un *nom*, d'une *visibilité* (`+` public, `#` protégé, `-` privé), d'une multiplicité (`1` exactement un, `*` plusieurs, `1..*` au moins 1) et d'une navigabilité (absence ou présence de flèche). Une association ne possède pas forcément toutes ses terminaisons d'association.

Les associations binaires de l'exemple peuvent se formuler en langage naturel en utilisant le nom de la classe de départ comme sujet, le nom de la terminaison distante pour former un groupe verbal (à défaut "est associé.e à"), le nom de la classe distante comme complément d'objet et la multiplicité de la terminaison distante pour dénombrer le complément d'objet :

Association	Individu vers Outil	Outil vers Individu
association1	Un `Individu` prefère **un** `Outil`	un `Outil` est le préféré d'**un** `Individu`
association2	Un `Individu` possède **plusieurs** `Outil` s	un `Outil` est possédé par **au plus un** `Individu`
association3	Un `Individu` emprunte **plusieurs** `Outil` s	non navigable
association4	Un `Individu` utilise **plusieurs** `Outil` s	un `Outil` est utilisé par **plusieurs** `Individu` s

où `Individu` désigne une instance de la classe `Individu` et `Outil` désigne une instance de la classe `Outil`



Implémentation des associations binaires

Pour chaque association binaire issue d'une classe "C", si l'association est navigable en direction de la classe distante, on la remplace par un nouvel attribut dans la classe "C" donc le type est soit celui d'un objet de la classe distante, soit une liste d'objets de la classe distance selon la multiplicité distante. Le nom de cet attribut est généralement le nom de la terminaison distante ou à défaut, le nom de la classe distante. Cela permet aux objets de connaitre leurs associés distants au travers d'un attribut dédié. Exemple

Association	**Attribut ajouté à `Individu`**	**Attribut ajouté à `Outil`**
association1	prefere : Outil	estLePreferede : Individu
association2	possede : Outil[*]	estPossedePar : Individu
association3	emprunte : Outil[*]	
association4	utilise : Outil[*]	estUtilisePar : Individu

Codage des associations dans Python

Les nouveaux attributs mis en évidence dans le modèle d'implantation sont initialisés dans les constructeurs respectifs de chaque classe (méthode `self.__init__`) et seront mis à jour par les différentes méthodes selon les besoins.

In [ ]:
```
class Individu(object):
    def __init__(self):
        self.nom = ""        # Attribut issu du modèle conceptuel
        self.prefere = None # Attribut issu du modèle d'implantation
        self.possede = []   # Attribut issu du modèle d'implantation
        self.emprunte = []  # Attribut issu du modèle d'implantation
        self.utilse = []    # Attribut issu du modèle d'implantation
```

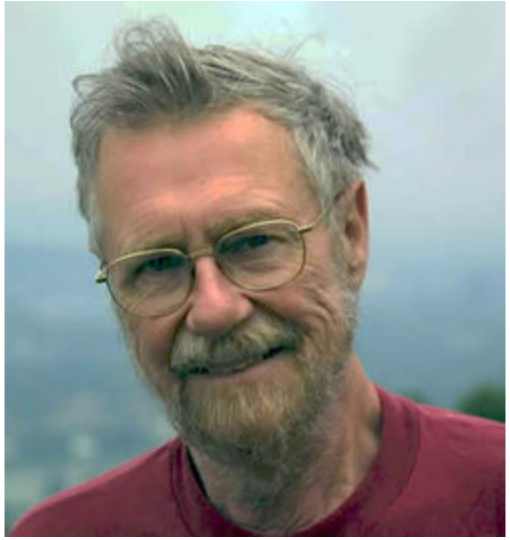
In [ ]:
```
class Outil(object):
    def __init__(self):
        self.nom = ""               # Attribut issu du modèle conceptuel
        self.estLePrefereDe = None # Attribut issu du modèle d'implantation
        self.estPossedePar = None  # Attribut issu du modèle d'implantation
        self.estUtilisePar = []    # Attribut issu du modèle d'implantation
```

Algorithme d'Edsger Dijkstra

L'algorithme de Dijkstra (prononcé [dɛɪkstra]) résoudre le problème du plus court chemin dans un graphe. Plus précisément, il calcule les plus courts chemins à partir d'une source vers tous les autres sommets dans un graphe orienté pondéré par des réels positifs.

Les détails de l'algorithme sont disponibles sur de nombreux sites WEB. Par exemple :

- Algorithme de Dijkstra (Wikipédia)



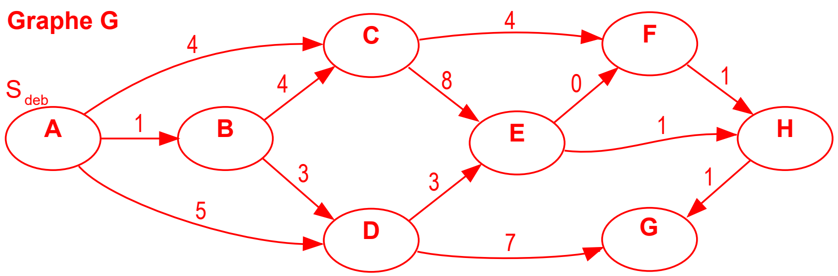
Les pages suivantes proposent une animation de l'algorithme

Algorithme de Dijkstra [adapté de Wikipédia]

```
Entrées :
  - G(S, A) un graphe avec une pondération positive poids des arcs
  - Sdeb un sommet de S
Sorties :
  - P(S, Af) un graphe (arbre de sommet Sdeb)
    avec une pondération positive d (distance) des sommets

P := ∅
d[a] := +∞ pour chaque sommet a
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
    Choisir un sommet a hors de P de plus petite distance d[a]
    Mettre a dans P
    Pour chaque sommet b hors de P voisin de a
        Si d[b] > d[a] + poids(a, b)
            d[b] := d[a] + poids(a, b)
            prédécesseur[b] := a
        Fin Si
    Fin Pour
Fin Tant que
```

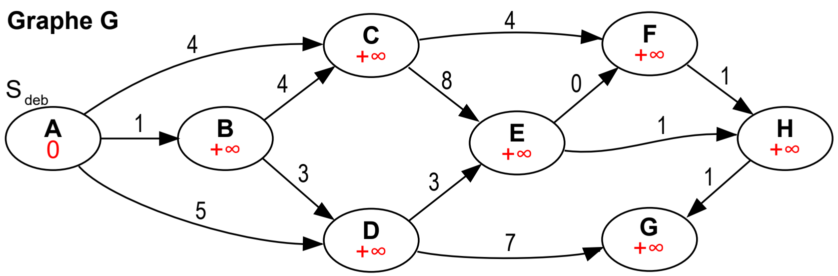
Les entrées

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs  <┐
  Sdeb un sommet de S                                              <┴──

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```



Initialisation

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```



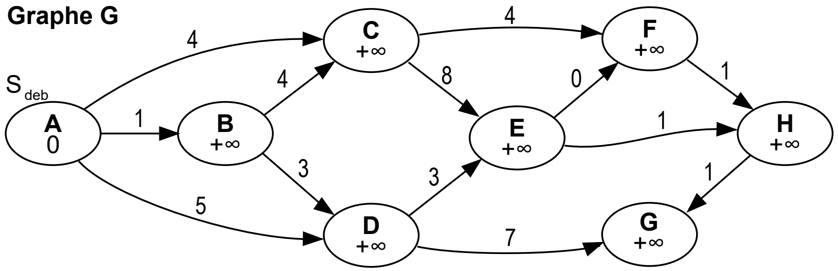
Graphe P

$\varnothing$

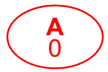
Etape 1 — Sommet courant **A**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]      a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G

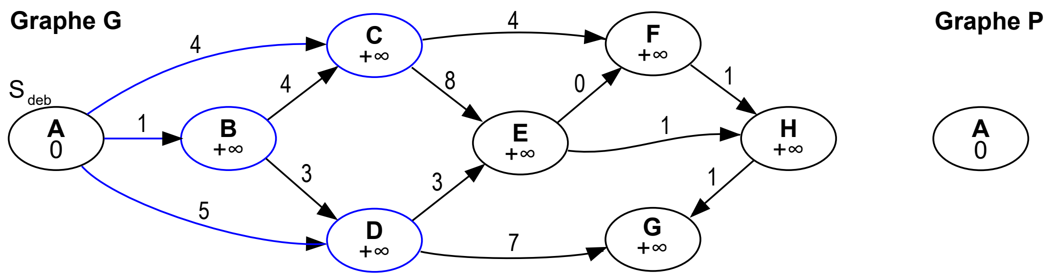


Graphe P



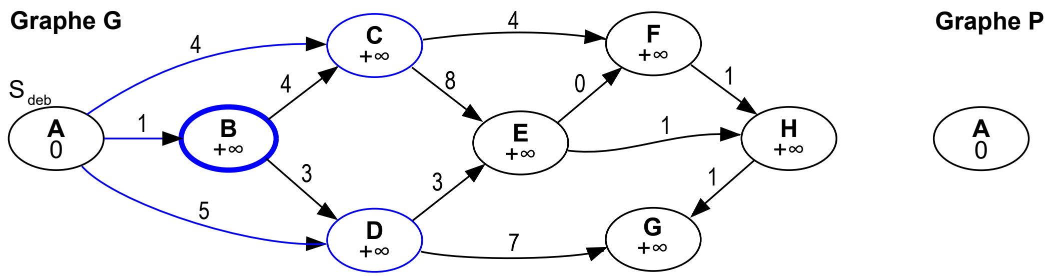
Etape 1 — Sommet courant **A** — recherche des successeurs

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                   <—— b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```



Etape 1 — Sommet courant **A** — Sommet voisin **B**

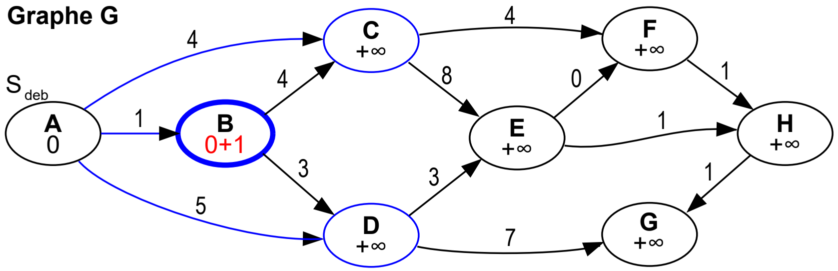
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)                          <———    b = B  et  +∞ > 0 + 1
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```



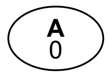
Etape 1 — Sommet courant **A** — Sommet voisin **B**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)                                  b = B  et  +∞ > 0 + 1
               d[b] := d[a] + poids(a, b)                       <———— d[B] := d[A] + poids(A, B) = 0 + 1
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G

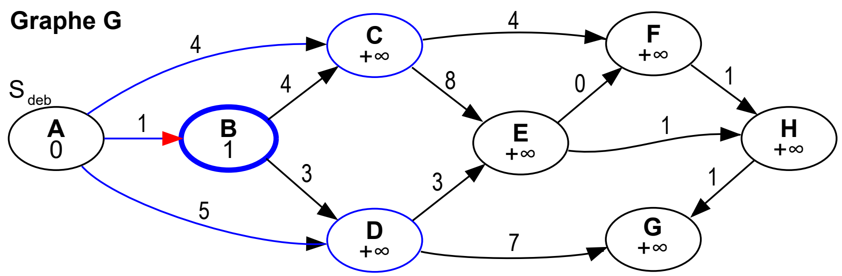


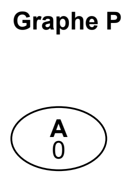
Graphe P



Etape 1 — Sommet courant **A** — Sommet voisin **B**

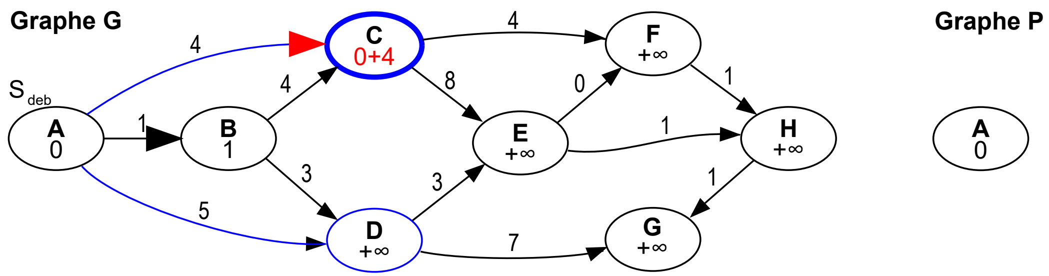
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)                                  b = B  et  +∞ > 0 + 1
               d[b] := d[a] + poids(a, b)                               d[B] := d[A] + poids(A, B) = 0 + 1
               prédécesseur[b] := a                                <——— prédécesseur[B] := A
          Fin Si
     Fin Pour
Fin Tant que
```





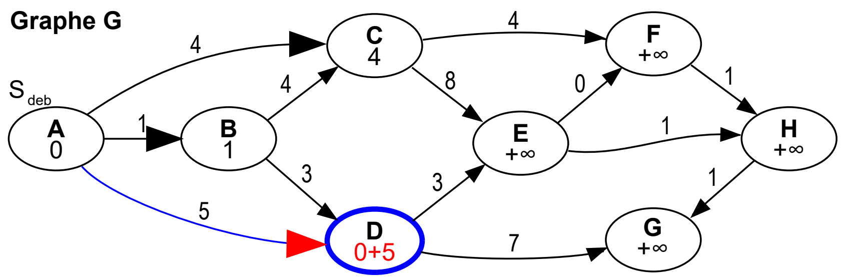
Etape 1 — Sommet courant **A** — Sommet voisin **C**

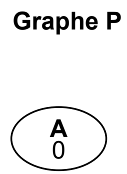
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]          a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                          b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)                              <——— b = C  et  +∞ > 0 + 1
               d[b] := d[a] + poids(a, b)                           <——— d[C] := d[A] + poids(A, C) = 0 + 1
               prédécesseur[b] := a                                 <——— prédécesseur[C] := A
          Fin Si
     Fin Pour
Fin Tant que
```



Etape 1 — Sommet courant **A** — Sommet voisin **D**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {B, C, D}
          Si d[b] > d[a] + poids(a, b)                             <——— b = D  et  +∞ > 0 + 1
               d[b] := d[a] + poids(a, b)                          <——— d[D] := d[A] + poids(A, D) = 0 + 1
               prédécesseur[b] := a                                <——— prédécesseur[D] := A
          Fin Si
     Fin Pour
Fin Tant que
```

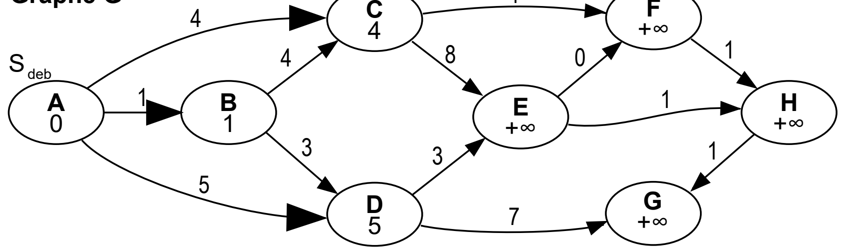




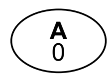
Etape 1 — Sommet courant **A** — fin de l'exploration des voisins

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = A et d(A) = 0
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G



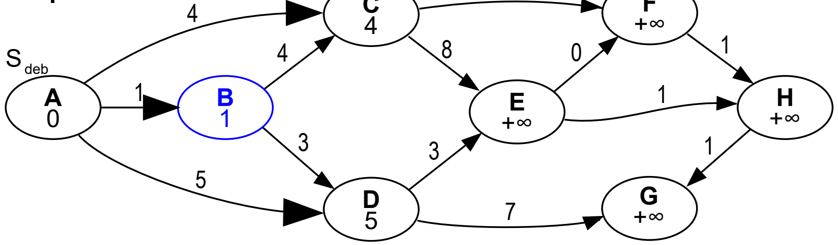
Graphe P



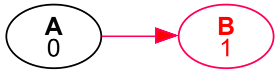
Etape 2 — Sommet courant **B**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <─┬── a = B et d(B) = 1
     Mettre a dans P                                              <─┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G

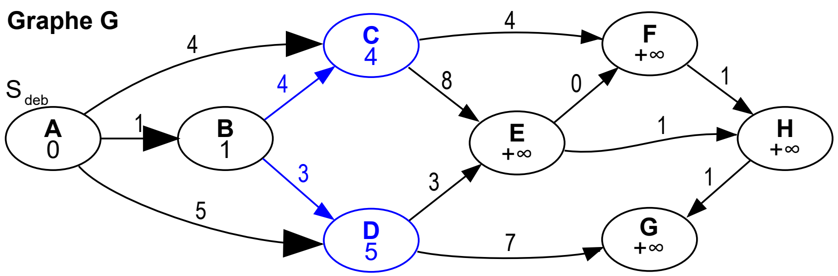


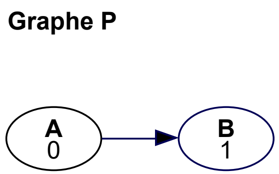
Graphe P



Etape 2 — Sommet courant **B** — recherche des successeurs

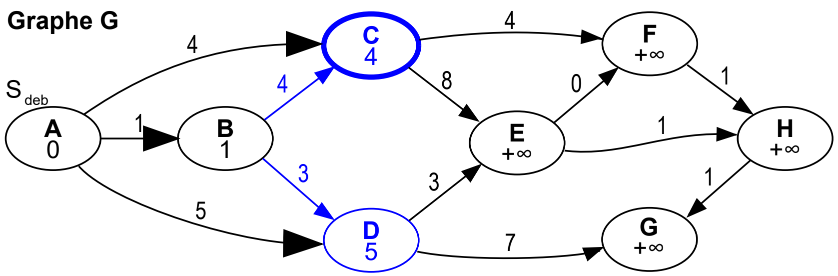
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = B et d(B) = 1
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                   <—— b ∈ {C, D}
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

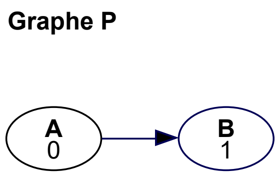




Etape 2 — Sommet courant **B** — Sommet voisin **C**

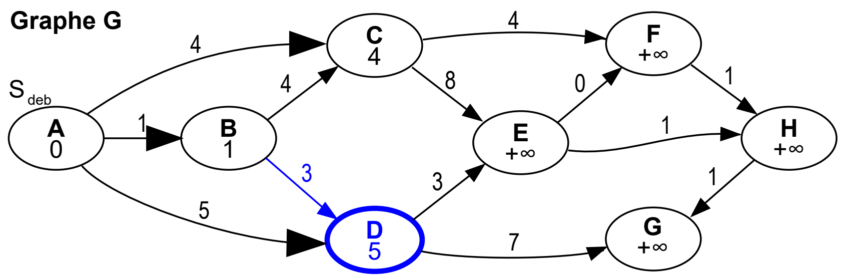
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = B et d(B) = 1
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {C, D}
          Si d[b] > d[a] + poids(a, b)                         <——— b = C   et   4 ≯ 1 + 4
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

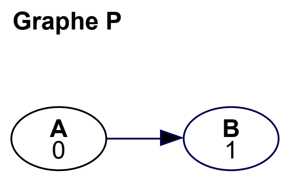




Etape 2 — Sommet courant **B** — Sommet voisin **D**

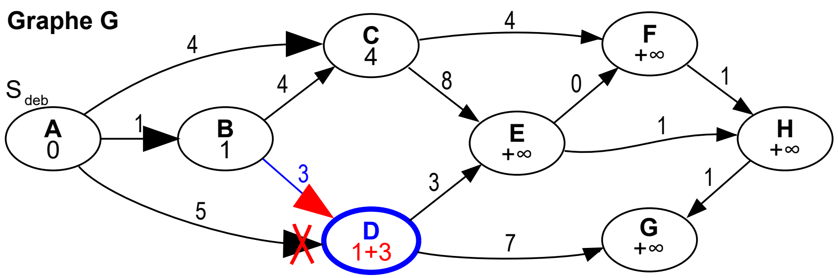
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = B et d(B) = 1
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                         b ∈ {C, D}
          Si d[b] > d[a] + poids(a, b)                              <—— b = D  et  5 > 1 + 3
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

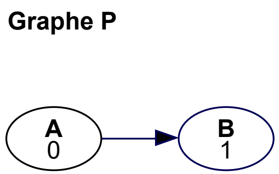




Etape 2 — Sommet courant **B** — Sommet voisin **D**

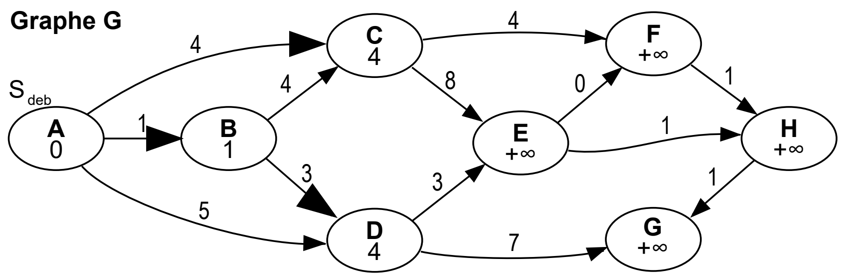
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]          a = B et d(B) = 1
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                          b ∈ {C, D}
          Si d[b] > d[a] + poids(a, b)                                   b = D  et  5 > 1 + 3
               d[b] := d[a] + poids(a, b)                          <———— d[D] := d[B] + poids(B, D) = 1 + 3
               prédécesseur[b] := a                                <———— prédécesseur[D] := B
          Fin Si
     Fin Pour
Fin Tant que
```

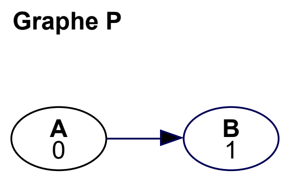




Etape 2 — Sommet courant **B** — fin de l'exploration des voisins

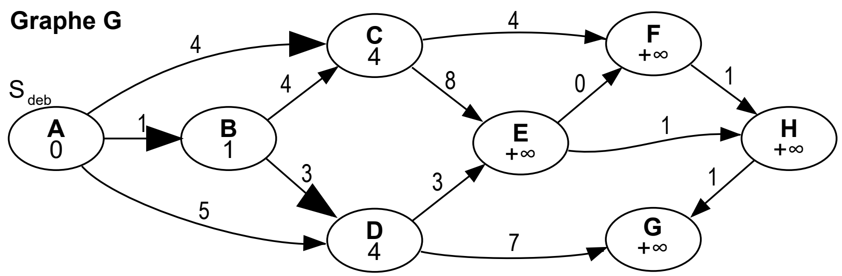
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = B et d(B) = 1
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour                                                     <——
Fin Tant que
```

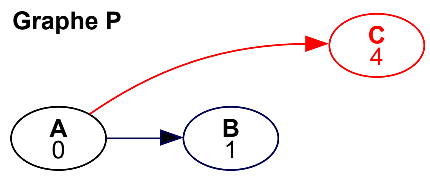




Etape 3 — Sommet courant **C**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <┬── a = C et d(C) = 4 (ou a = D)
     Mettre a dans P                                              <┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

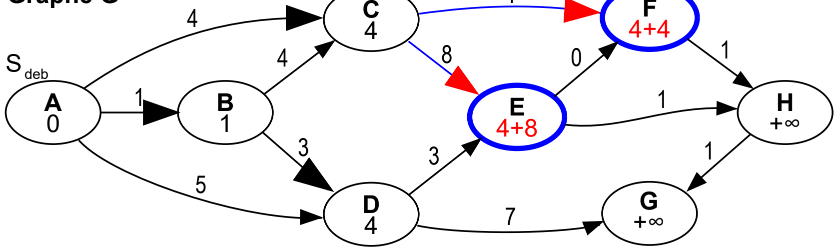




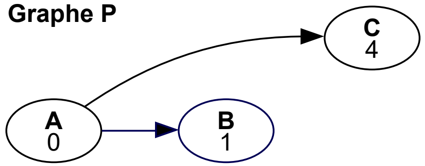
Etape 3 — Sommet courant **C** — étude des successeurs

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = C et d(C) = 4
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G



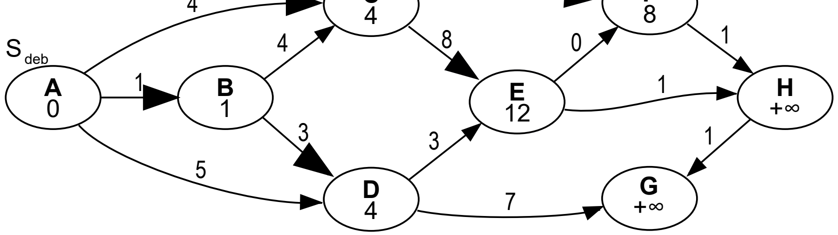
Graphe P



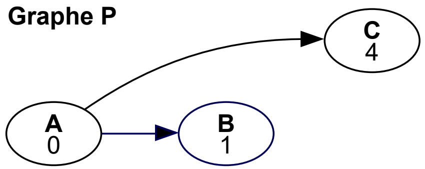
Etape 3 — Sommet courant **C** — fin de l'exploration des voisins

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = C et d(C) = 4
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G

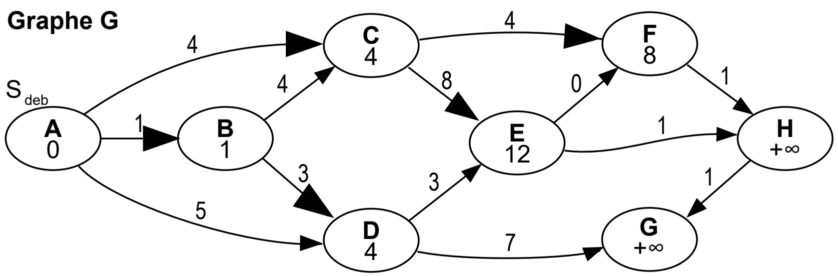


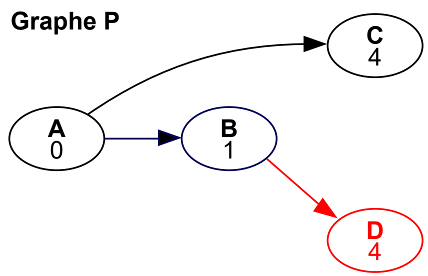
Graphe P



Etape 4 — Sommet courant **D**

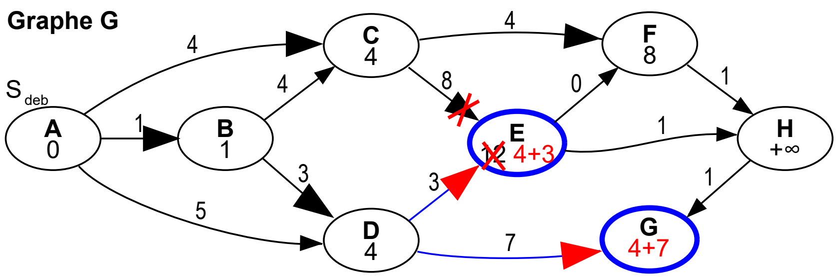
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <─┬── a = D et d(D) = 4
     Mettre a dans P                                              <─┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

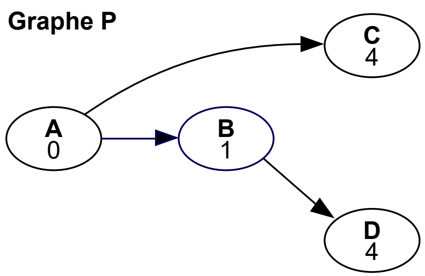




Etape 4 — Sommet courant **D** — étude des successeurs

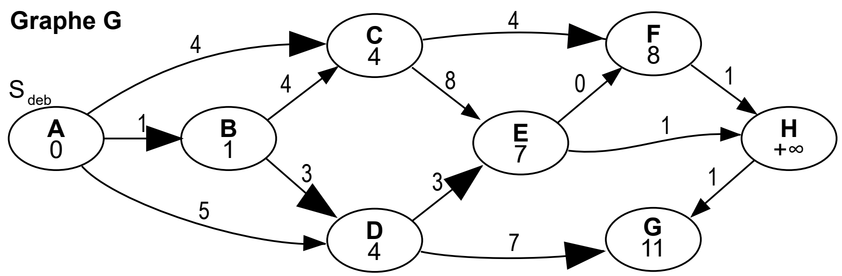
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = D et d(D) = 4
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

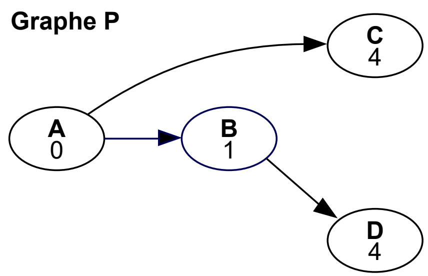




Etape 4 — Sommet courant **D** — fin de l'exploration des voisins

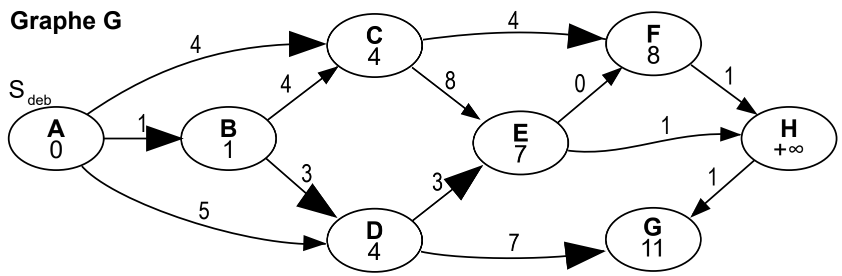
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = D et d(D) = 4
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour                                                      <———
Fin Tant que
```

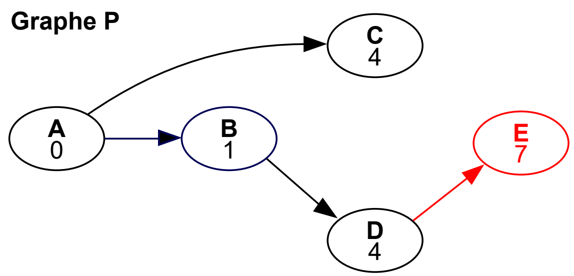




Etape 5 — Sommet courant **E**

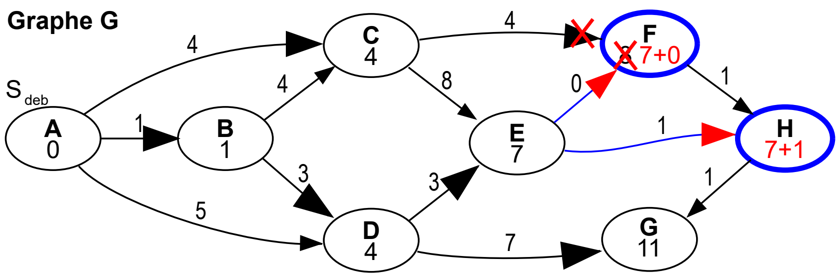
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <─┬── a = E et d(E) = 7
     Mettre a dans P                                              <─┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

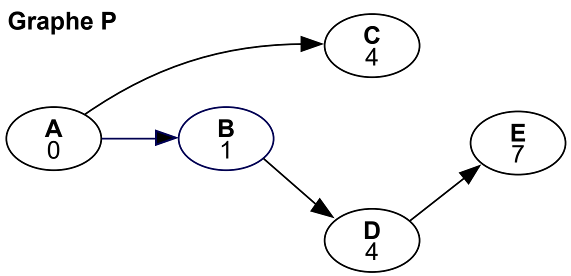




Etape 5 — Sommet courant **E** — étude des successeurs

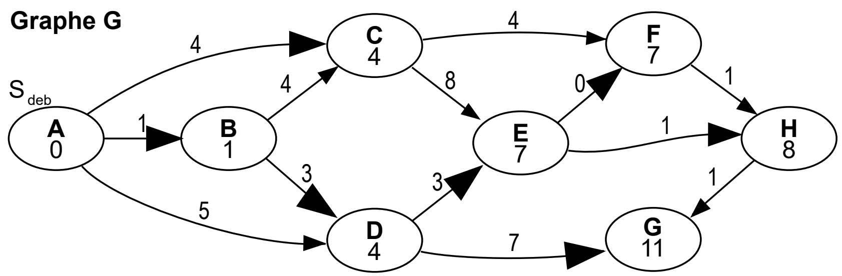
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = E et d(E) = 7
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

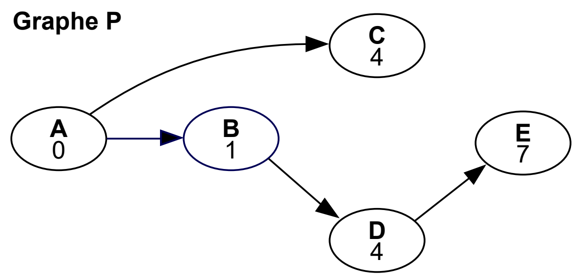




Etape 5 — Sommet courant **E** — fin de l'exploration des voisins

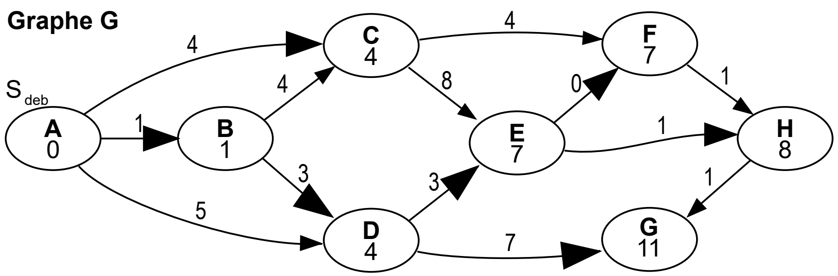
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = E et d(E) = 7
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour                                                         <———
Fin Tant que
```

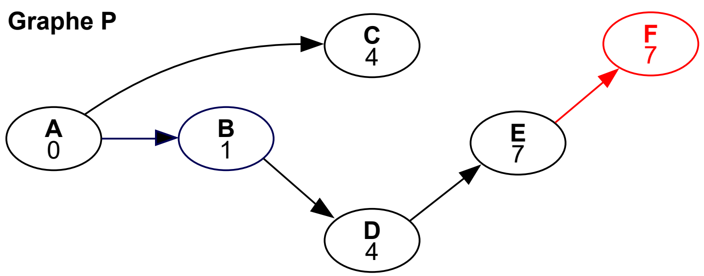




Etape 6 — Sommet courant **F**

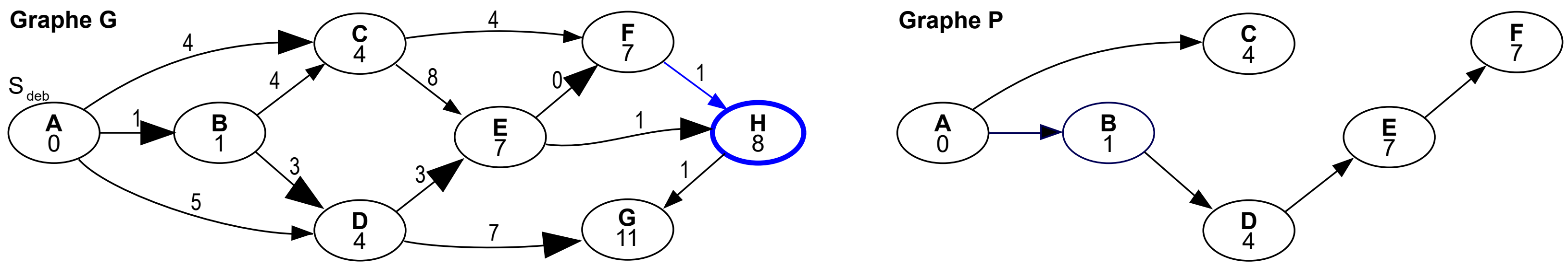
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <┬── a = F et d(F) = 7
     Mettre a dans P                                              <┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```





Etape 6 — Sommet courant **F** — étude des successeurs

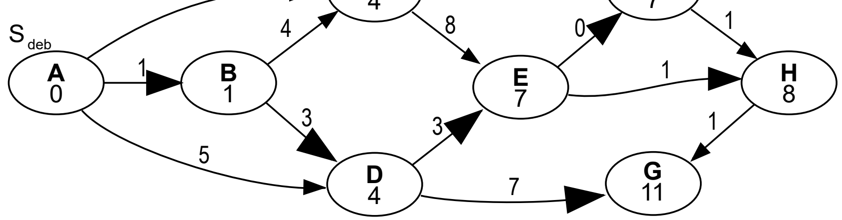
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = F et d(F) = 7
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```



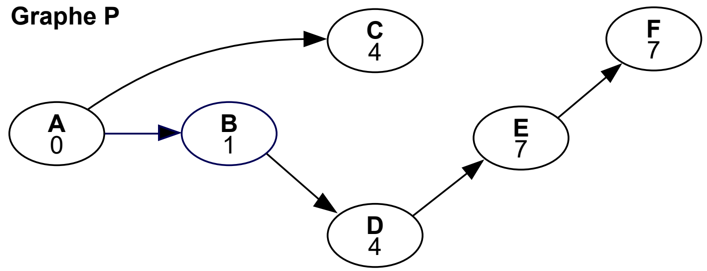
Etape 6 — Sommet courant **F** — fin de l'exploration des voisins

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = F et d(F) = 7
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour                                                        <——
Fin Tant que
```

Graphe G

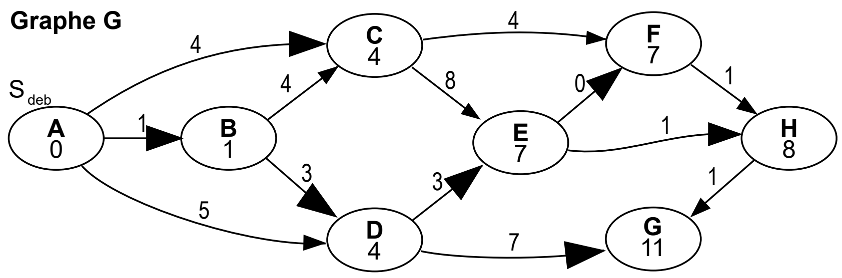


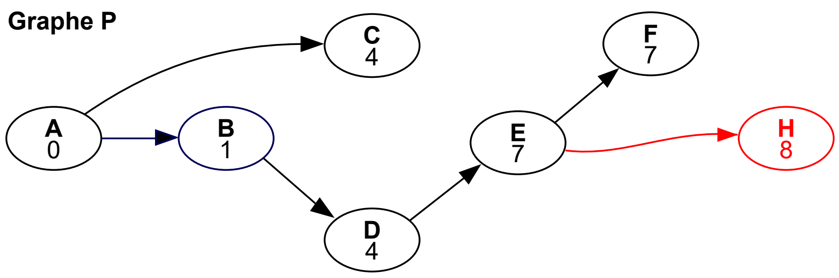
Graphe P



Etape 7 — Sommet courant **H**

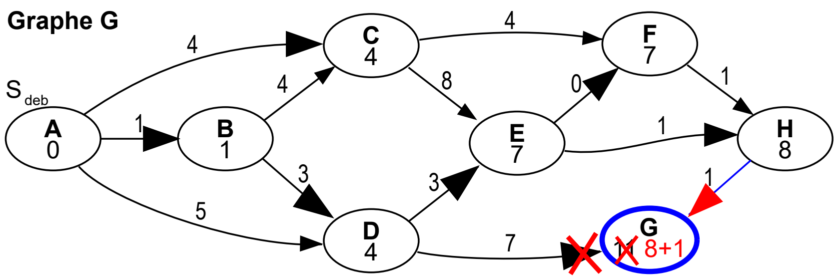
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]    <─┬── a = H et d(H) = 8
     Mettre a dans P                                               <─┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

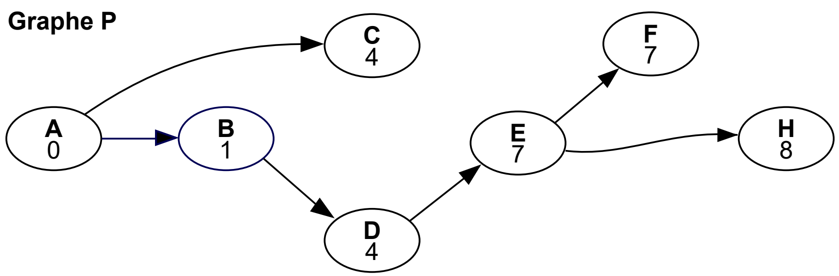




Etape 7 — Sommet courant **H** — étude des successeurs

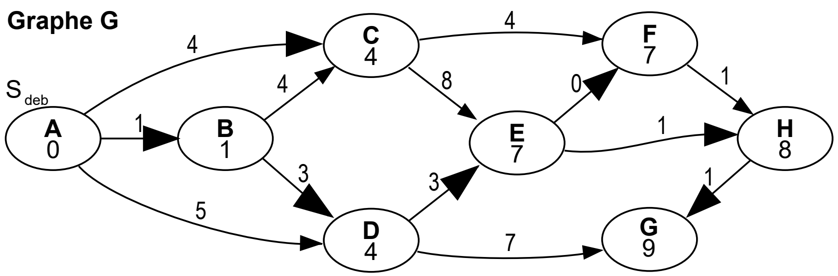
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = H et d(H) = 8
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

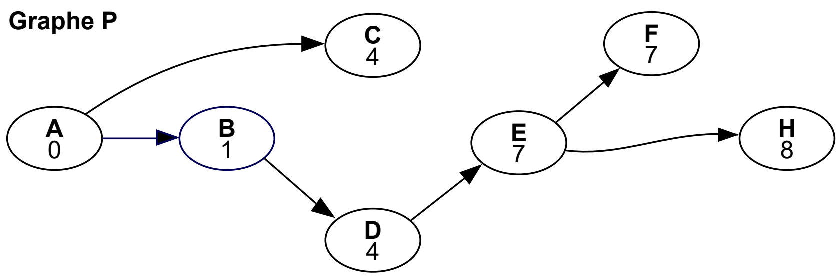




Etape 7 — Sommet courant **H** — fin de l'exploration des voisins

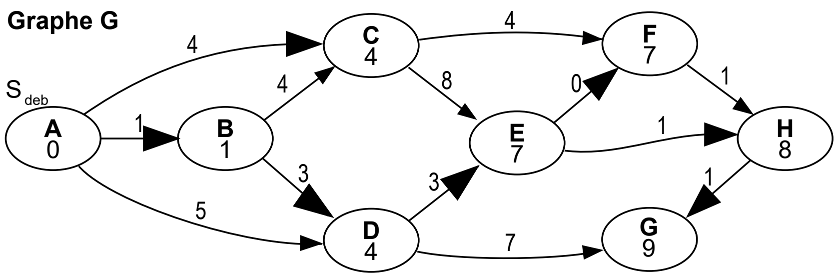
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = H et d(H) = 8
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour                                                       <———
Fin Tant que
```

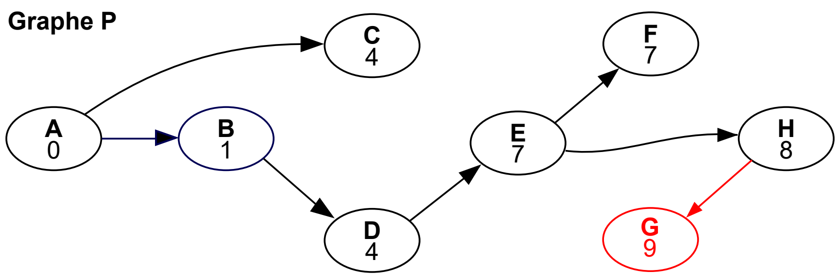




Etape 8 — Sommet courant **G**

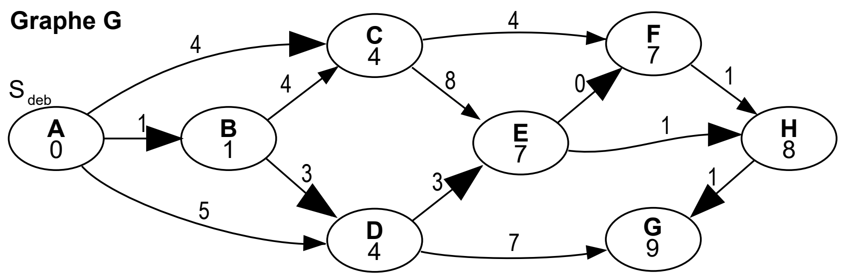
```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]   <─┬── a = G et d(G) = 9
     Mettre a dans P                                              <─┘
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

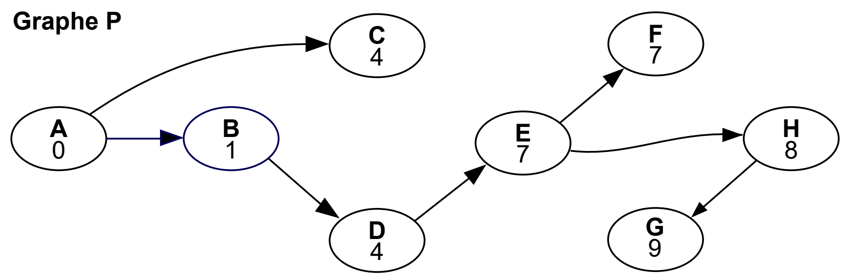




Etape 8 — Sommet courant **G** — pas de successeur à étudier

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P
     Choisir un sommet a hors de P de plus petite distance d[a]         a = G et d(G) = 9
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a                       ┐
          Si d[b] > d[a] + poids(a, b)                                 |
               d[b] := d[a] + poids(a, b)                              | Pas de voisin
               prédécesseur[b] := a                                    |
          Fin Si                                                       ┘
     Fin Pour
Fin Tant que
```

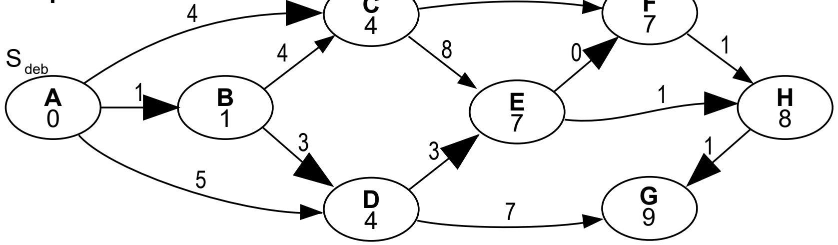




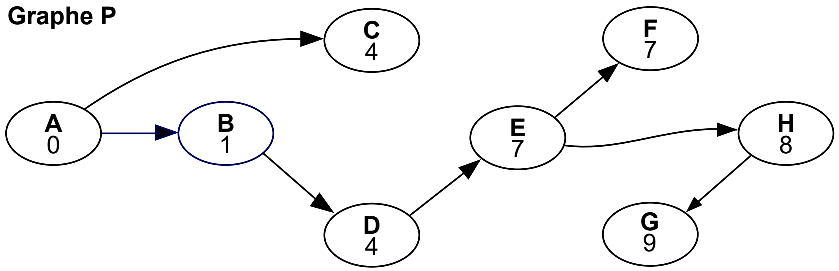
Plus aucun nouveau sommet à étudier — **Fin de l'algorithme**

```
Entrées :
  G(S, A) un graphe avec une pondération positive poids des arcs
  Sdeb un sommet de S

P := {}
d[a] := +∞ pour chaque sommet a de S
d[sdeb] := 0
Tant qu'il existe un sommet hors de P                           <——— plus aucun sommet
     Choisir un sommet a hors de P de plus petite distance d[a]
     Mettre a dans P
     Pour chaque sommet b hors de P voisin de a
          Si d[b] > d[a] + poids(a, b)
               d[b] := d[a] + poids(a, b)
               prédécesseur[b] := a
          Fin Si
     Fin Pour
Fin Tant que
```

Graphe G



Graphe P

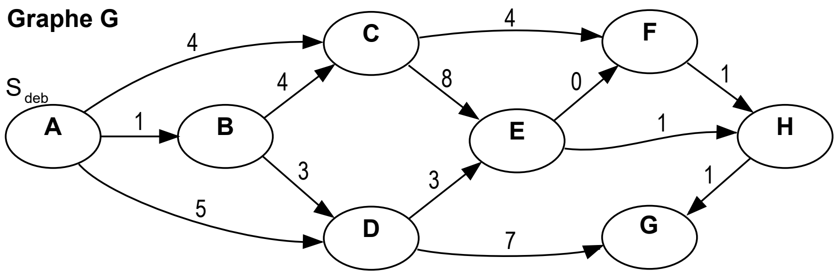


Bilan de l'exécution de l'algorithme

Entrées :

- G(S, A) un graphe avec une pondération positive poids des arcs
- S_{deb} un sommet de S

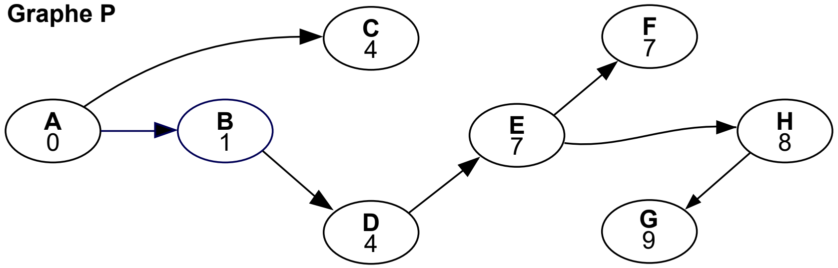
Graphe G

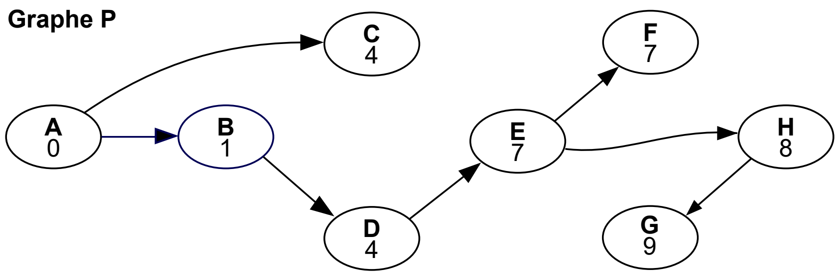


Sorties :

- P(S, Af) un graphe (arbre de sommet Sdeb)
  avec une pondération positive d (distance) des sommets

Graphe P



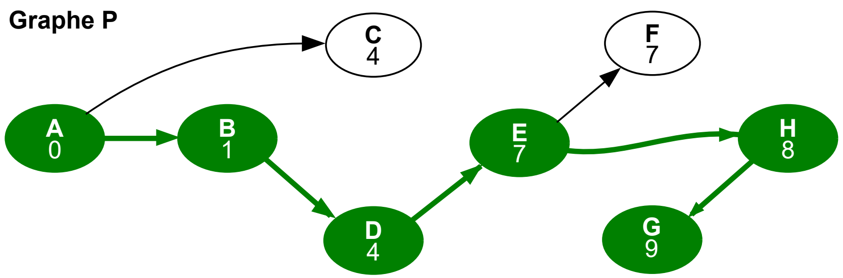


Le tableau récapiculatif des plus courtes distances depuis **A** se lit directement sur l'arbre P

de \ vers	A	B	C	D	E	F	G	H
A	0	1	4	4	7	7	**9**	8

Le somment le plus éloigné de **A** est **G** avec une distance de **9**

Le chemin se construit en remontant du sommet feuille **G** vers le sommet racine **A** dans l'arbre P



Utilisation de Graphviz

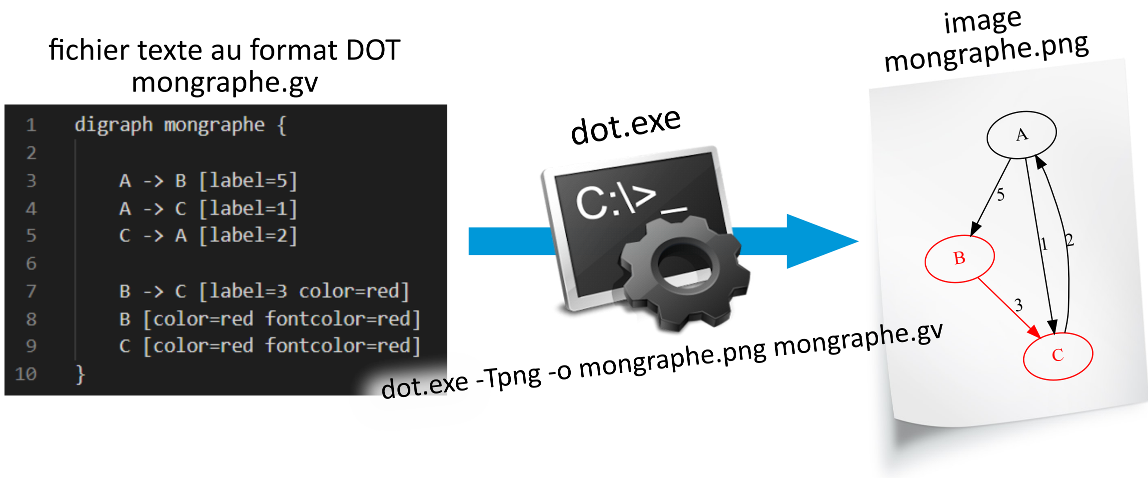
Graphviz (Graph Visualization Software) est un ensemble d'outils open source qui dessinent des graphes définis à l'aide de scripts suivant le langage DOT

- Site web
- Grammaire formelle du langage DOT
- Guide d'utilisation l'exécutable dot.exe
- Documantation du module graphviz de Python

Utilisation directe de l'exécutable dot.exe

À partir d'un fichier texte `mongraphe.gv` qui décrit le graphe selon le format DOT, l'exécution `dot.exe` crée un rendu graphique. Pour un rendu au format PNG (option `-Tpng`) dans le fichier de sortie `mongraphe.png` (option `-o mongraphe.png`), la commande suivante est exécutée dans interface en ligne de commande du système d'exploitation

```
> dot.exe -Tpng -o mongraphe.png mongraphe.gv
```

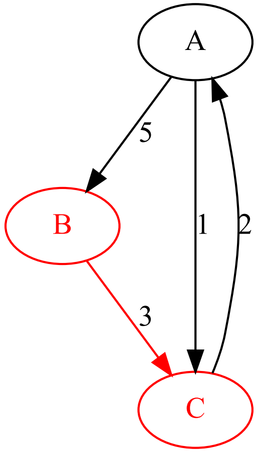


Pour exécuter cette commande directement depuis un script en langage Python, il est possible d'utiliser la fonction `os.system()` du module `os` de Python. Au préalable, il faudra avoir créé le fichier `mongraphe.gv` avec Python.

Exemple de fichier `mongraphe.gv`

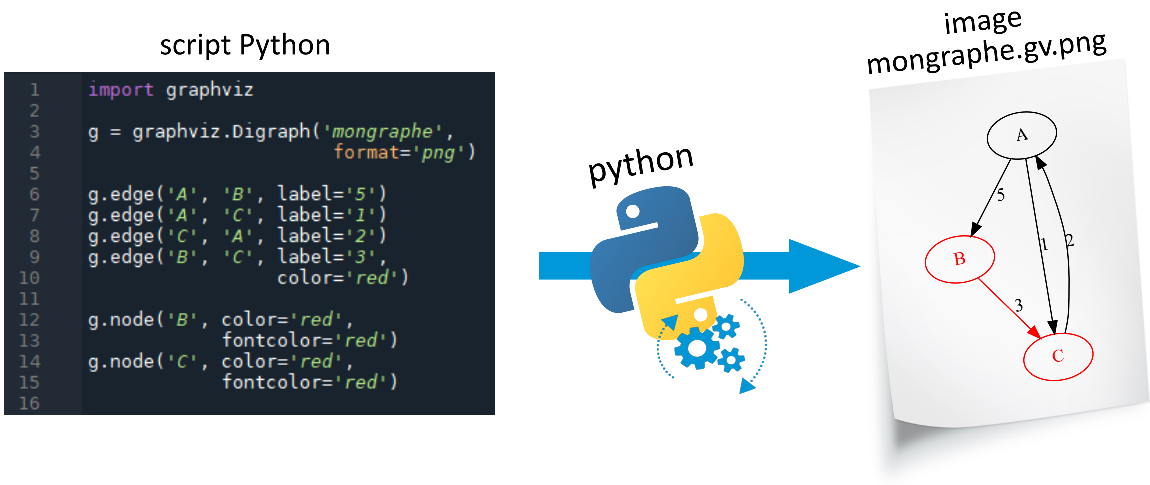
```
digraph mongraphe {

    A -> B [label="5"];
    A -> C [label="1"];
    C -> A [label="2"];
    B -> C [label="3", color=red];

    B [color=red, fontcolor=red];
    C [color=red , fontcolor=red];
}
```

Rendu obtenu par l'exécution de `dot.exe -Tpng -o mongraphe.png mongraphe.gv`



Utilisation du module python `graphviz`

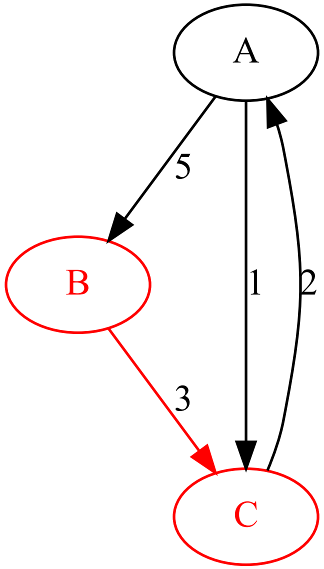
Le module `graphviz` de Python propose une classe `graphviz.Digraph` pour générer un rendu graphique des graphes orientés. Ses méthodes `edge()` et `node()` permettent d'ajouter des arcs et des sommets, tandis que sa méthode `render()` génère le fichier au format DOT et exécute dot.exe qui génère à son tour un fichier contenant l'image de rendu du graphe.



Exemple de script Python avec le module Graphviz

In [2]:
```python
import graphviz

g = graphviz.Digraph('mongraphe', filename='mongraphe.gv', format='svg')

g.edge('A', 'B', label='5')
g.edge('A', 'C', label='1')
g.edge('C', 'A', label='2')
g.edge('B', 'C', label='3', color='red')

g.node('B', color='red', fontcolor='red')
g.node('C', color='red', fontcolor='red')

g # affiche le rendu
```

Out[2]:



Langage HTML et les tableaux

HTML (HyperText Markup Language) est un langage de balisage de texte, c'est-à-dire un langage qui enrichit du texte pur avec des balises délimitant des séquences de caractères ou marquant une position à l'intérieur du texte afin de le structurer en paragraphes, en titres, en citations, en partie à mettre en exergue... Parmi les autres langages de balisage on retrouve par exemple $\LaTeX$ et SVG (Scalable Vector Graphics) pour les images.

HTML décrit la structure d'une page Web, sous la forme d'une série d'**éléments** balisés. Chaque élément indique au navigateur comment afficher son **contenu** en l'informant de sa nature (entête, paragraphe, lien, liste, tableau...) et éventuellement de ses **attributs**.

Les structures types d'un élément HTML sont

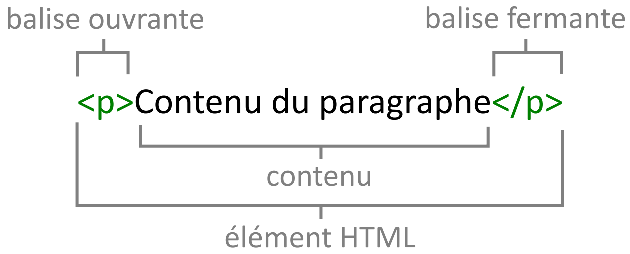
- `<nomBalise>Le contenu de l'élément</nomBalise>` pour les éléments avec contenu
- `<nomBalise>` pour les éléments sans contenu qui ne possède donc pas de balise fermante

Exemples de balises HTML :

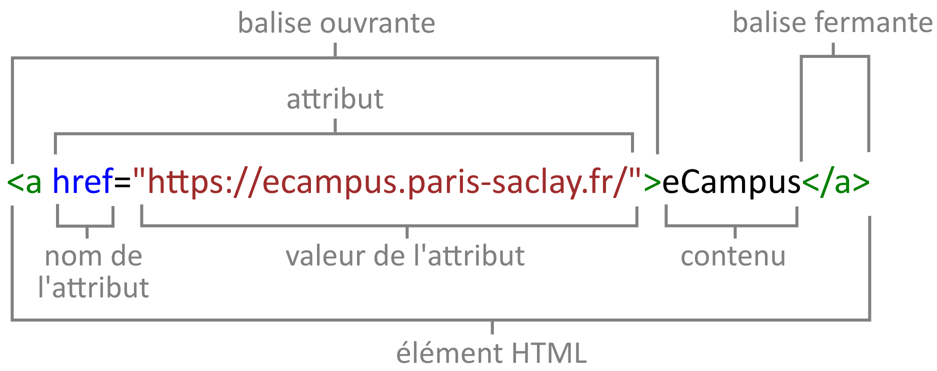
- `<html>` élément racine de la page
- `<head>` élément contenant les métadonnées de la page
- `<title>` élément contenant la métadonnée "titre de la page"
- `<body>` élément contenant le corps de la page qui contient tout ce qui sera visible
- `<h1>` élément contenant un titre de niveau 1
- `<h2>` élément contenant un titre de niveau 2
- `<p>` élément contenant un paragraphe de texte
- `<a>` élément qui définit un hyperlien sur son contenu
- `<table>` élément contenant un tableau
- `<img>` élément pour insérer une image dans une page (pas de contenu ni de balise de fin)

La syntaxe complète du langage de balisage `HTML` peut être obtenue sur le site w3schools.com.

La figure suivante montre la constitution d'un élément *paragraphe*



Les attributs d'un élément vont se placer au sein de la balise ouvrante. Ils précisent ou apportent des informations complémentaires sur l'élément. La figure suivante montre un élément *hyperlien* dont le contenu est le texte "eCampus" et dont l'attribut *href* précise la destination de l'hyperlien vers https://ecampus.paris-saclay.fr/



La structure type d'une page est :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Titre de la page</title>
    </head>
    <body>
        Le contenu est ici...
    </body>
</html>
```

In [41]:
```python
# Affichage du rendu de la page html
import IPython.display
IPython.display.IFrame(src="page.html", width=250, height=40)
```

Out[41]: Le contenu est ici...

Les balises d'éléments HTML dédiées aux tableaux sont :

- `<table>` élément contenant un tableau
- `<tr>` élément contenant une ligne de tableau (tr pour "table row")
- `<td>` élément contenant une cellule de tableau (td pour "table data")
- `<th>` élément contenant une cellule d'entête de tableau (th pour "table heading")

Et la structure type d'un tableau dans le contenu d'une page html :

```
<table>
  <tr>
    <th>header col 1</th>
    <th>header col 2</th>
  </tr>
  <tr>
    <td>raw col 1</td>
    <td>raw col 1</td>
  </tr>
</table>
```

In [9]:
```python
# Affichage du rendu du tableau html
import IPython.display
IPython.display.IFrame(src="table.html", width=250, height=75)
```

Out[9]:

header col 1	header col 2
raw col 1	raw col 1

Des exemples de mises en forme de tableaux sont disponibles sur le site GeeksforGeeks

Analyse syntaxique

Une grammaire est un quadruplet $G = (T, NT, P, S)$ où

- T est l'ensemble des symboles **terminaux** (caractère de l'alphabet)
- NT est l'ensemble des symboles **non terminaux**
- P est un ensemble de **règles de production** de la forme $\alpha \to \beta$, avec $\alpha \in (NT \cup N)^+$ et $\beta \in (NT \cup N)^*$ pour un langage algébrique aussi appelé langage non contextuel
- S est le symbole de départ, un élément de NT appelé l'**axiome**

Dans la suite, un symbole terminal sera noté entre simples guillemets `'symbole_terminal'` tandis qu'un symbole non terminal sera noté entre les signes inférieur et supérieur `<symbole_non_terminal>`

Exemple type de texte à analyser

```
<SOMMETS> A_1 ;  B_2 ; </SOMMETS>
```

Formalisation de la grammaire : règles de production

```
(R01) <element_sommets> → <balise_os> <liste_sommets> <balise_fs>
(R02)       <balise_os> → '<' 'SOMMETS' '>'
(R03)       <balise_fs> → '<' '/' 'SOMMETS' '>'
(R04)   <liste_sommets> → <nom> ';' <liste_sommets>
(R05)   <liste_sommets> → <nom> ';'

(R06)             <nom> → <lettre>
(R07)             <nom> → <nom> <lettre>
(R08)             <nom> → <nom> <chiffre>
(R09)             <nom> → <nom> '_'
```

où `<balise_os>` signifie balise en ouverture des sommets, `<balise_fs>` signifie balise en fermeture des sommets, `<balise_oa>` signifie balise en ouverture des arcs, `<nom>` signifie nom d'un sommet avec

```
(R10) à (R19)  <chiffre> ∈ Digit = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}
(R20) à (R71)  <lettre> ∈ Letter = {'A', ..., 'Z', 'a', ..., 'z'}
```

Ainsi, la grammaire est définie par le quadruplet $G = (T, NT, P, S)$ où

- l'ensemble des symboles terminaux $T =$ { 'SOMMETS' , '<' , '/' , '>' , ';' , '_' } ∪ { '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' } ∪ { 'A' , ..., 'Z' , 'a' , ..., 'z' }
- l'ensemble des symboles non terminaux $NT =$ { <element_sommets> , <balise_os> , <balise_fs> , <liste_sommets> , <nom> , <lettre> , <chiffre> }
- l'ensemble des règles de production $P = \{Ri\}_{1 \leq i \leq 71}$
- l'axiome $S =$ <element_sommets>

Les règles sous forme factorisée

```
(R01)    <element_sommets> → <balise_os> <liste_sommets> <balise_fs>
(R02)          <balise_os> → '<' 'SOMMETS' '>'
(R03)          <balise_fs> → '<' '/' 'SOMMETS' '>'
(R04)(R05) <liste_sommets> → <nom> ';' <liste_sommets> | <nom> ';'
(R06)(R07)(R08)(R09) <nom> → <lettre> | <nom> <lettre> | <nom> <chiffre> | <nom> '_'
(R10) à (R19)    <chiffre> → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
(R20) à (R71)    <lettres> → 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
```

où <balise_os> signifie balise en ouverture des sommets, <balise_fs> signifie balise en fermeture des sommets, <balise_oa> signifie balise en ouverture des arcs et <nom> signifie nom d'un sommet

Pour découvrir le module `pyparsing` de Paul McGuire

- Introduction to PyParsing
- PyParsing's documentation

Chaque symbole de la grammaire, terminal ou non, est codé par un objet de la classe `ParserElement` ou de l'une de ses sous-classes.

In [21]:
```
import pyparsing as pp

# ------------------
# Symboles terminaux

LEFT_CHEVRON = pp.Literal('<')
RIGHT_CHEVRON = pp.Literal('>')
SLASH = pp.Literal('/')
SEMICOLON = pp.Literal(';')
UNDERSCORE = pp.Literal('_')
SOMMETS = pp.CaselessKeyword('SOMMETS')
# ...
```

In [24]:
```
# ----------------------
# Symboles non terminaux

# règles de R20 à R71
lettre = pp.alphas
# règles de R10 à R19
chiffre = pp.nums
# règles R06 à R06
nom = pp.Word(lettre, lettre + chiffre + str(UNDERSCORE))
# règles R04 et R05 <liste_sommets> → <nom> ';' <liste_sommets> | <nom> ';'
liste_sommets = pp.OneOrMore(nom + pp.Suppress(SEMICOLON))
# règles R03 <balise_fs> → '<' '/' 'SOMMETS' '>'
balise_fs = pp.Group(LEFT_CHEVRON + SLASH + SOMMETS + RIGHT_CHEVRON)
# règles R02 <balise_os> → '<' 'SOMMETS' '>'
balise_os = pp.Group(LEFT_CHEVRON + SOMMETS + RIGHT_CHEVRON)
# règles R01
# <element_sommets> → <balise_os> <liste_sommets> <balise_fs>
element_sommets = pp.Group(pp.Suppress(balise_os)
                           + liste_sommets
                           + pp.Suppress(balise_fs))('sommets')
```

L'élément de parser `element_sommets` est nommé par la chaine de caractères 'sommets' pour servir de clés à un dictionnaire qui facilitera leur accès. L'ajout de se nom se faire par appel implicite à la méthode `__call__()`

L'objet `element_sommets` est l'axiome S de notre grammaire (symbole de départ pour l'analyseur). C'est à partir de cet objet que l'on analyse une chaine de caractères (`parse_string(chaine_de_caracteres)`) ou le contenu d'un fichier texte (`parse_file(nom_de_fichier)`)

In [45]:
```
chaine = """
<SOMMETS>
   A_1 ;
   B_2 ;
</SOMMETS>"""

resultatDuParser = element_sommets.parse_string(chaine)
```

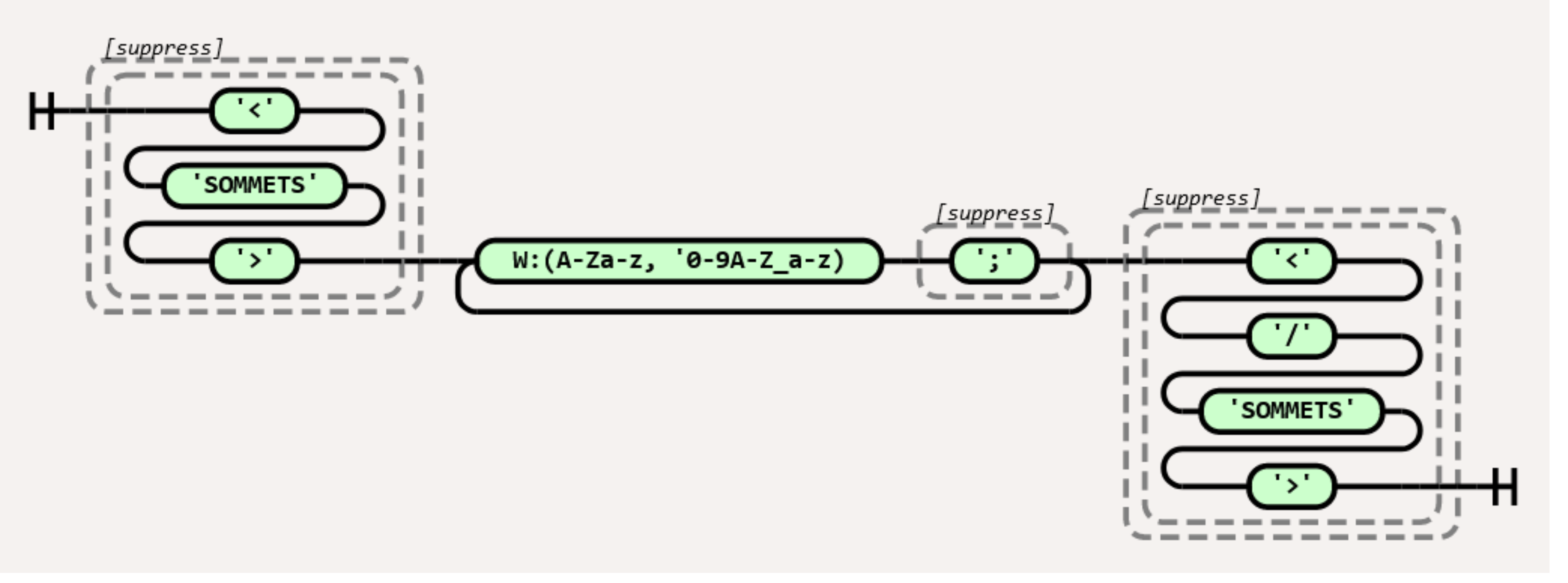
Le résultat de l'analyse `resultatDuParser` peut être mis en forme dans un dictionnaire Python à l'aide de la méthode `asDict()` .

In [46]:
```python
# Affiche le dictionnaire résultat de l'analyse grammaticale (parsing)
# de la chaine 'file_content' par l'analyseur (parser) 'element_sommets'
resultatDuParser.asDict()
```

Out[46]:
```
{'sommets': ['A_1', 'B_2']}
```

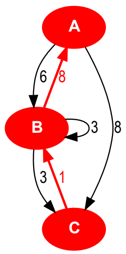
Les versions ressentes de `pyparsing` permettent la génération d'une représentation graphique de la grammaire (le module `railroad-diagrams` est également nécessaire).

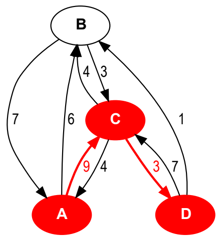
In [42]:
```python
# génération du diagramme de syntaxe si version >= 3
if int(pp.__version__.split('.')[0]) >= 3:
    element_sommets.create_diagram('parser_element_sommets_diag.html')
```

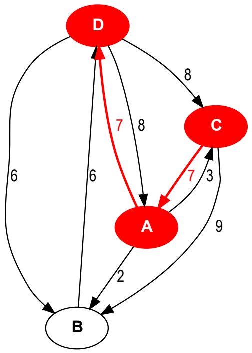


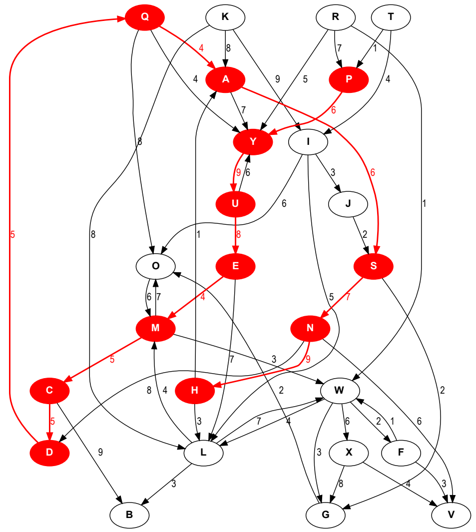
Graphes de test

- Example031 : 3 noeuds, 1 seule solution
- Example041 : 4 noeuds, 1 seule solution
- Example043 : 4 noeuds, 3 solutions
- Example251 : 25 noeuds, 1 solution







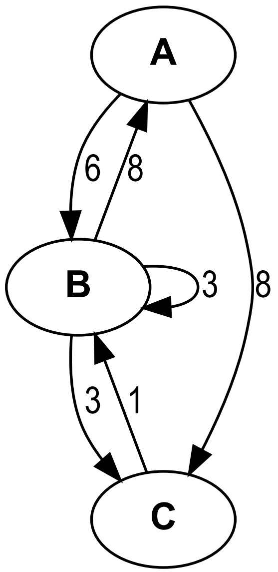


Example031 : 3 noeuds, 1 seule solution

Représentation sous forme de tuple python du graphe

```
(   'example031', ['A', 'B', 'C'],
    [   ('A', 'B', 6), ('C', 'B', 1), ('A', 'C', 8), ('B', 'C', 3),
        ('B', 'B', 3), ('B', 'A', 8)])
```

Représentation graphique du graphe



Représentation xml-like du graphe

```
<GRAPHE Name="example031" >
<SOMMETS>
    A;
    B;
    C;
</SOMMETS>
<ARCS>
    A:B:6;
    A:C:8;
    B:A:8;
    B:B:3;
    B:C:3;
    C:B:1;
</ARCS>
</GRAPHE>
```

Il n'y a qu'un plus court chemin qui atteint la longueur maximale de **9**.

Le chemin allant de **C** à **A**

- C — 1 → B
- B — 8 → A

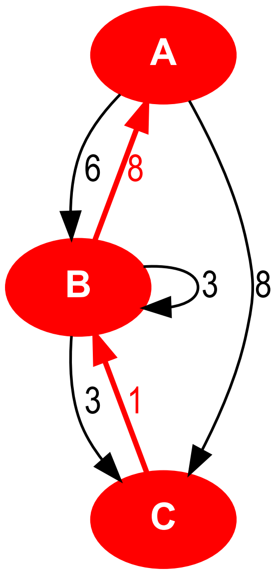


Table des distances minimales de sommet à sommet

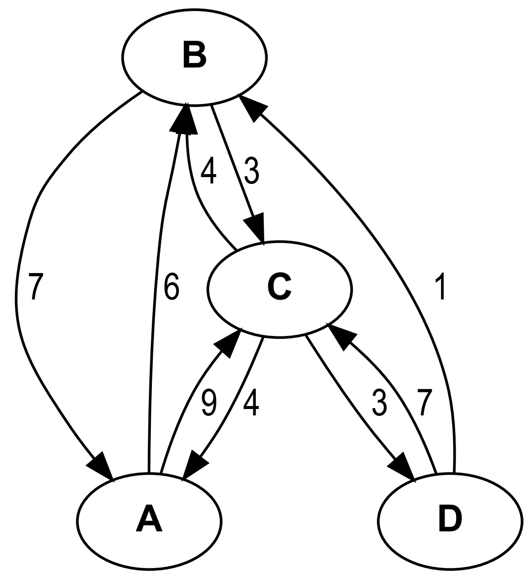
de \ vers	A	B	C
A	-	6	8
B	8	-	3
C	**9**	1	-

Example041 : 4 noeuds, 1 seule solution

Représentation sous forme de tuple python du graphe

```
(   'example041', ['A', 'B', 'C', 'D'],
    [   ('B', 'C', 3), ('B', 'A', 7), ('C', 'A', 4), ('A', 'C', 9),
        ('D', 'C', 7), ('A', 'B', 6), ('C', 'B', 4), ('D', 'B', 1),
        ('C', 'D', 3)])
```

Représentation graphique du graphe



Représentation xml-like du graphe

```
<GRAPHE Name="example041" >
<SOMMETS>
    A;
    B;
    C;
    D;
</SOMMETS>
<ARCS>
    A:B:6;
    A:C:9;
    B:A:7;
    B:C:3;
    C:A:4;
    C:B:4;
    C:D:3;
    D:B:1;
    D:C:7;
</ARCS>
</GRAPHE>
```

Il n'y a qu'un plus court chemin qui atteint la longueur maximale de **12**

Le chemin allant de **A** à **D**

- A — 9 → C
- C — 3 → D

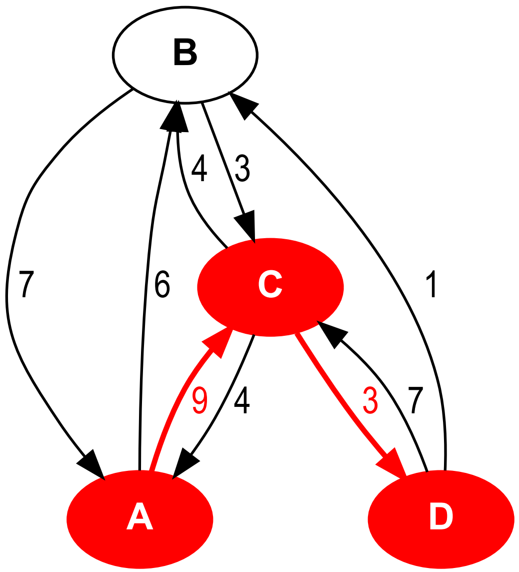


Table des distances minimales de sommet à sommet

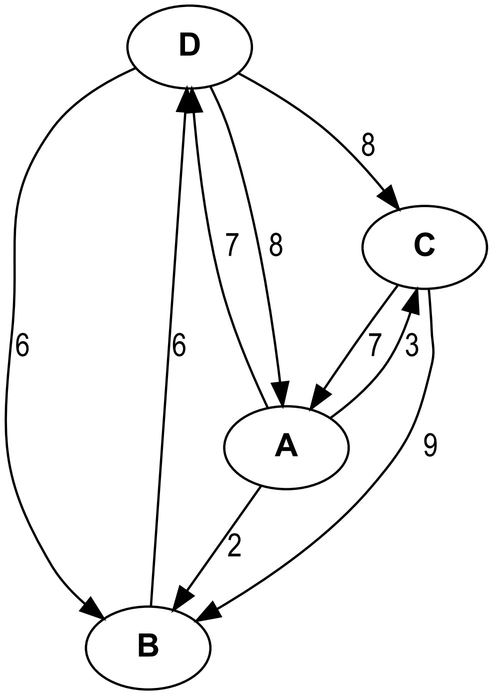
de \ vers	A	B	C	D
A	-	6	9	**12**
B	7	-	3	6
C	4	4	-	3
D	8	1	4	-

Example043 : 4 noeuds, 3 solutions

Représentation sous forme de tuple python du graphe

```
(   'example043', ['A', 'B', 'C', 'D'],
    [   ('D', 'C', 8), ('D', 'B', 6), ('B', 'D', 6), ('A', 'C', 3),
        ('A', 'B', 2), ('A', 'D', 7), ('C', 'A', 7), ('C', 'B', 9),
        ('D', 'A', 8)])
```

Représentation graphique du graphe



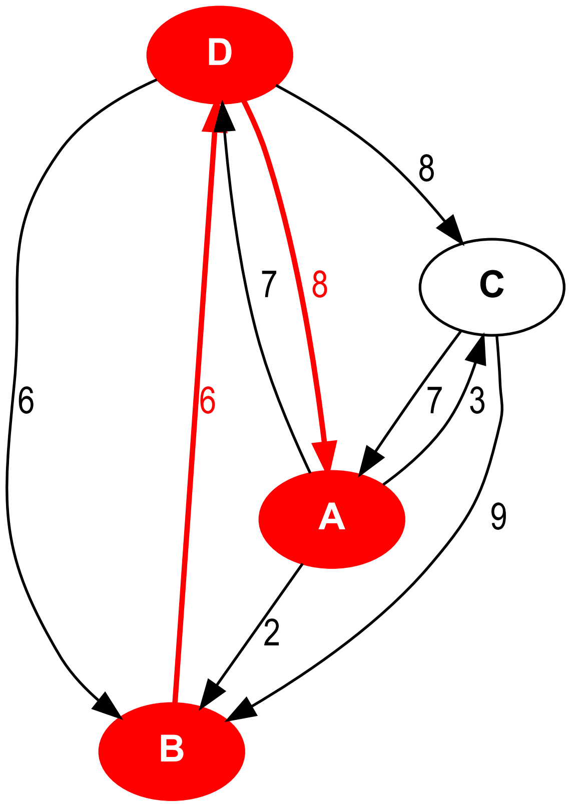
Représentation xml-like du graphe

```
<GRAPHE Name="example043" >
<SOMMETS>
    A;
    B;
    C;
    D;
</SOMMETS>
<ARCS>
    A:B:2;
    A:C:3;
    A:D:7;
    B:D:6;
    C:A:7;
    C:B:9;
    D:A:8;
    D:B:6;
    D:C:8;
</ARCS>
</GRAPHE>
```

Il y a **3** plus courts chemins qui atteignent de longueur maximale de **14**

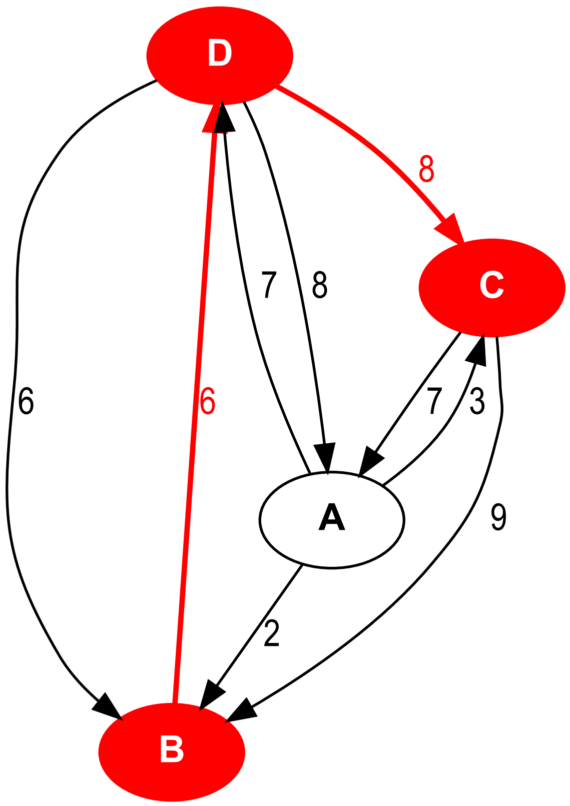
Le chemin allant de **B** à **A**

- B — 6 → D
- D — 8 → A



Le chemin allant de **B** à **C**

- B — 6 → D
- D — 8 → C



Le chemin allant de **C** à **D**

- C — 7 → A
- A — 7 → D

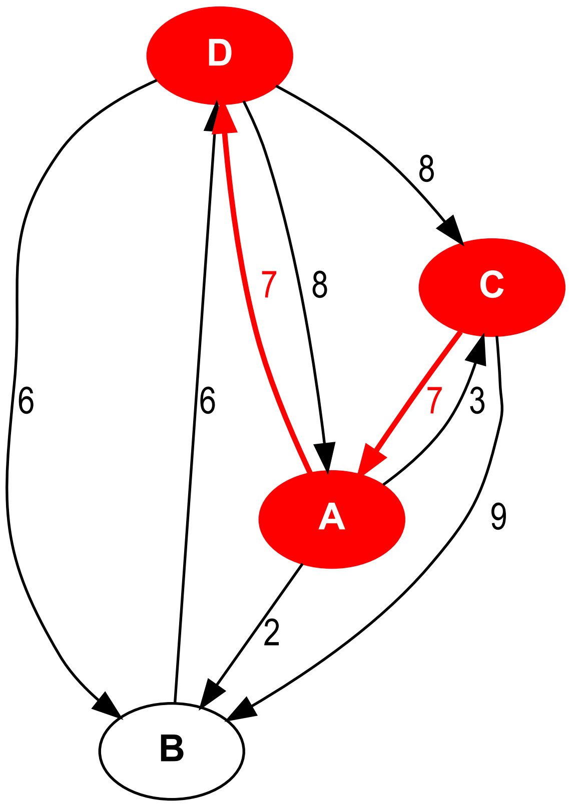


Table des distances minimales de sommet à sommet

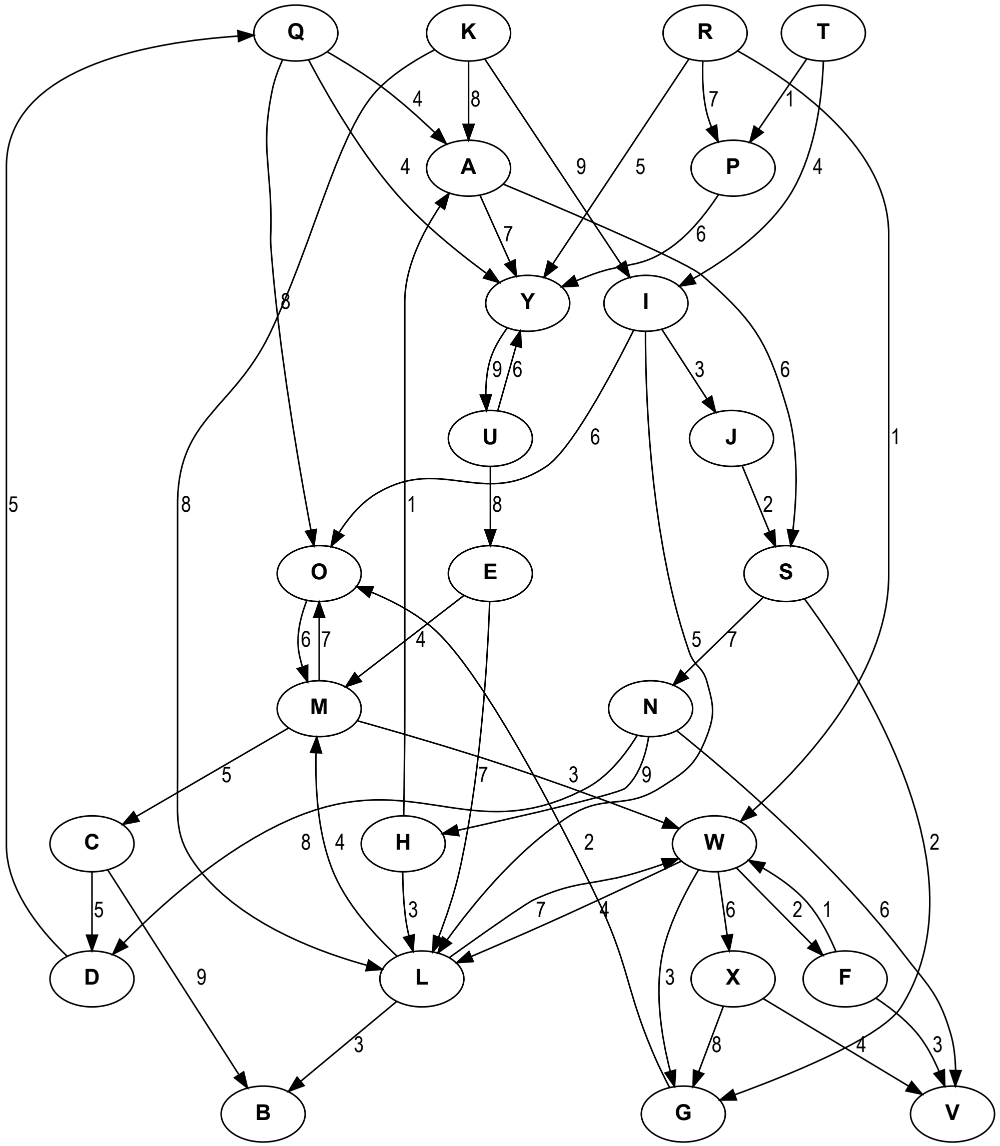
de \ vers	A	B	C	D
A	-	2	3	7
B	**14**	-	**14**	6
C	7	9	-	**14**
D	8	6	8	-

Example251 : 25 noeuds, 1 solution

Représentation sous forme de tuple python du graphe

```
(   'example251',
    [   'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
        'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y'],
    [   ('Q', 'O', 8), ('I', 'L', 5), ('X', 'G', 8), ('E', 'M', 4),
        ('K', 'A', 8), ('Q', 'Y', 4), ('N', 'H', 9), ('A', 'Y', 7),
        ('M', 'W', 3), ('A', 'S', 6), ('C', 'D', 5), ('D', 'Q', 5),
        ('J', 'S', 2), ('N', 'D', 8), ('X', 'V', 4), ('N', 'V', 6),
        ('R', 'P', 7), ('L', 'M', 4), ('W', 'X', 6), ('W', 'L', 4),
        ('Q', 'A', 4), ('L', 'B', 3), ('U', 'Y', 6), ('H', 'A', 1),
        ('L', 'W', 7), ('F', 'V', 3), ('E', 'L', 7), ('W', 'F', 2),
        ('M', 'C', 5), ('T', 'I', 4), ('R', 'Y', 5), ('I', 'O', 6),
        ('W', 'G', 3), ('M', 'O', 7), ('P', 'Y', 6), ('K', 'L', 8),
        ('F', 'W', 1), ('S', 'N', 7), ('I', 'J', 3), ('H', 'L', 3),
        ('U', 'E', 8), ('K', 'I', 9), ('O', 'M', 6), ('S', 'G', 2),
        ('R', 'W', 1), ('G', 'O', 2), ('C', 'B', 9), ('Y', 'U', 9),
        ('T', 'P', 1)])
```

Représentation graphique du graphe



Représentation xml-like du graphe

```
<GRAPHE Name="example251" >
<SOMMETS>
    A; B; C; D; E; F; G; H; I; J;
    K; L; M; N; O; P; Q; R; S; T;
    U; V; W; X; Y;
</SOMMETS>
<ARCS>
    A:S:6; A:Y:7; C:B:9; C:D:5; D:Q:5; E:L:7; E:M:4; F:V:3; F:W:1; G:O:2;
    H:A:1; H:L:3; I:J:3; I:L:5; I:O:6; J:S:2; K:A:8; K:I:9; K:L:8; L:B:3;
    L:M:4; L:W:7; M:C:5; M:O:7; M:W:3; N:D:8; N:H:9; N:V:6; O:M:6; P:Y:6;
    Q:A:4; Q:O:8; Q:Y:4; R:P:7; R:W:1; R:Y:5; S:G:2; S:N:7; T:I:4; T:P:1;
    U:E:8; U:Y:6; W:F:2; W:G:3; W:L:4; W:X:6; X:G:8; X:V:4; Y:U:9;
</ARCS>
</GRAPHE>
```

Il n'y a qu'un plus court chemin qui atteint la longueur maximale de **68**

Le chemin allant de **P** à **H**

- P — 6 → Y
- Y — 9 → U
- U — 8 → E
- E — 4 → M
- M — 5 → C
- C — 5 → D
- D — 5 → Q
- Q — 4 → A
- A — 6 → S
- S — 7 → N
- N — 9 → H

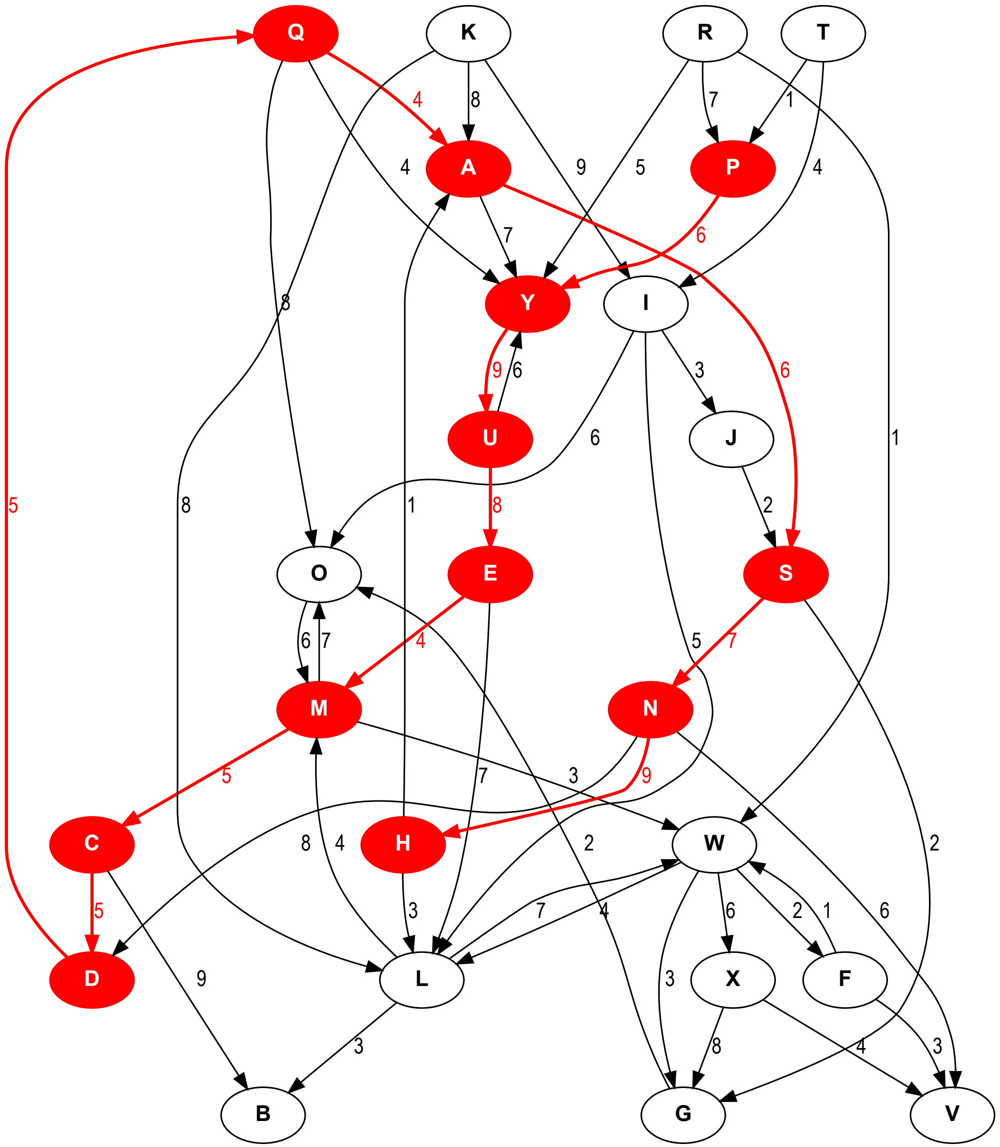

Q
K
R
T
A
P
Y
I
U
J
O
E
S
M
N
C
H
W
D
L
X
F
B
G
V

Table des distances minimales de sommet à sommet

de \ vers	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A	-	26	21	21	24	21	8	22	-	-	-	23	16	13	10	-	26	-	6	-	16	19	19	25	7
B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C	14	9	-	5	31	29	22	36	-	-	-	31	24	27	18	-	10	-	20	-	23	32	27	33	14
D	9	29	24	-	26	24	17	31	-	-	-	26	19	22	13	-	5	-	15	-	18	27	22	28	9
E	23	10	9	14	-	9	10	45	-	-	-	7	4	36	11	-	19	-	29	-	32	12	7	13	23
F	28	8	14	19	45	-	4	50	-	-	-	5	9	41	6	-	24	-	34	-	37	3	1	7	28
G	27	18	13	18	44	13	-	49	-	-	-	15	8	40	2	-	23	-	33	-	36	16	11	17	27
H	1	6	12	17	25	12	9	-	-	-	-	3	7	14	11	-	22	-	7	-	17	15	10	16	8
I	22	8	14	19	45	14	7	21	-	3	-	5	9	12	6	-	24	-	5	-	37	17	12	18	28
J	19	22	17	17	43	17	4	18	-	-	-	19	12	9	6	-	22	-	2	-	35	15	15	21	26
K	8	11	17	22	32	17	16	30	9	12	-	8	12	21	15	-	27	-	14	-	24	20	15	21	15
L	23	3	9	14	40	9	10	45	-	-	-	-	4	36	11	-	19	-	29	-	32	12	7	13	23
M	19	10	5	10	36	5	6	41	-	-	-	7	-	32	7	-	15	-	25	-	28	8	3	9	19
N	10	15	21	8	34	21	18	9	-	-	-	12	16	-	20	-	13	-	16	-	26	6	19	25	17
O	25	16	11	16	42	11	12	47	-	-	-	13	6	38	-	-	21	-	31	-	34	14	9	15	25
P	46	33	32	37	23	32	33	**68**	-	-	-	30	27	59	34	-	42	-	52	-	15	35	30	36	6
Q	4	24	19	24	21	19	12	26	-	-	-	21	14	17	8	-	-	-	10	-	13	22	17	23	4
R	28	8	14	19	22	3	4	50	-	-	-	5	9	41	6	7	24	-	34	-	14	6	1	7	5
S	17	20	15	15	41	15	2	16	-	-	-	17	10	7	4	-	20	-	-	-	33	13	13	19	24
T	26	12	18	23	24	18	11	25	4	7	-	9	13	16	10	1	28	-	9	-	16	21	16	22	7
U	31	18	17	22	8	17	18	53	-	-	-	15	12	44	19	-	27	-	37	-	-	20	15	21	6
V	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W	27	7	13	18	44	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	8	[illegible]	5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5	[illegible]	6	27

W	27	7	13	18	44	2	3	49	-	-	-	4	8	40	5	-	23	-	33	-	36	5	-	6	27
X	35	26	21	26	52	21	8	57	-	-	-	23	16	48	10	-	31	-	41	-	44	4	19	-	35
Y	40	27	26	31	17	26	27	62	-	-	-	24	21	53	28	-	36	-	46	-	9	29	24	30	-