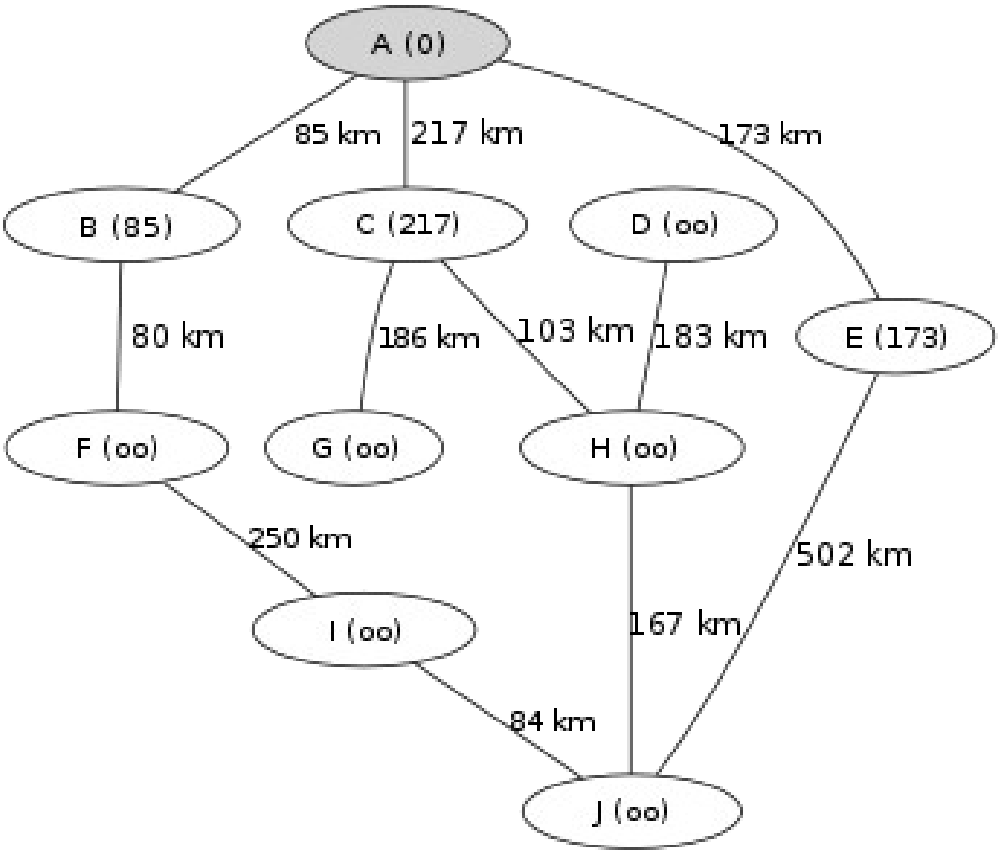
Année 2014-2015
Prépa CPP-2A
Matthieu ZIMMER
Jean-Philippe MANGEOT

TP N°1

Durée : *4h*
Objectifs :
Maitriser la représentation de graphes et implantation de l'algorithme de Dijkstra
Pré-requis : TD1, TD2.

Le plus court chemin dans un graphe



graphe **G1**

Année 2014-2015
Prépa CPP-2A
Matthieu ZIMMER
Jean-Philippe MANGEOT

Tuples et Dictionnaires

Un « tuple » permet de stocker des données de différents types dans une unique variable et de les récupérer facilement :

```
>>> t = ("une chaîne", 3)
>>> t
('une chaîne', 3)

>>> (a,b) = t
>>> a
'une chaîne'
>>> b
3
```

Un « dictionnaire » permet d'associer une clé à une valeur. Tandis qu'un tableau est représenté par une fonction :

$N \rightarrow$ Y (où on définit Y de la manière qu'on veut)

Un « dictionnaire » est représenté par

X → Y (où on définit à la fois X et Y)

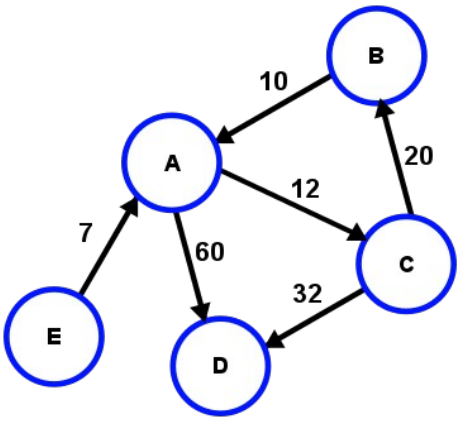
```
>>> dico = {'A' : 3}
>>> dico['A']
3

>>> dico['B'] = 2
>>> dico
{'A': 3, 'B': 2}

>>> dico[2]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 2
```

Représenter un graphe en python

Pour représenter un graphe orienté valué, nous allons utiliser 3 types de structures différentes : dictionnaire, tuple et liste.



graphe **G2**

Le tuple nous permet de stocker à la fois le nom du nœud et la valeur associée à l'arc.
Par exemple, le sommet **A** peut atteindre:

- le nœud **C** relié par un arc de valeur 12 : ('**C**',12)
- le nœud **D** relié par un arc de valeur 60 : ('**D**',60)

Pour représenter l'ensemble des voisins de **A**, on utilisera la liste suivante :
[('**C**',12), ('**D**',60)]

Pour stocker le graphe complet, on se servira du dictionnaire = { Clé : Valeur, ... }

```
graphe = { 'A' : [('C',12), ('D',60)],
           'E' : [('A',7)],
           'B' : … }
```

On pourra alors récupérer directement l'ensemble des nœuds (voisins) accessibles du nœud x avec : graphe['x'] :

```
>>> graphe['A']
>>> [('C',12), ('D',60)]

>>> graphe['B']
>>> [('A',10)]
```

Question 1

1) Retranscrire le graphe **G2** en python.

2) Récupérer l'ensemble des nœuds accessibles :
   - depuis C :
   ```
   >>>[('B', 20), ('D', 32)]
   ```
   - depuis D :
   ```
   >>>[]
   ```

3) Récupérer la valeur de l'arc C vers B
   ```
   >>>20
   ```

Question 2

4) Écrire une fonction pour afficher plus lisiblement un graphe :
   afficher(graphe) : void

   En effet, sans formater l'affichage un graphe devient vite illisible :

   ```
   >>>graphe
   >>>{'A':[ ('C',12), ('D', 60) ], 'E' : [('A', 7)], 'C' : [('A',2), ('B', 2)], 'T':[], 'R':[('A', 2),('R', 3),('Q', 5)]}
   ```

   Votre fonction doit pouvoir afficher plutôt (***l'ordre n'est important***) :

   ```
   >>>afficher(graphe)
   >>>A -> [('C', 12), ('D', 60)]
   >>>C -> [('A', 2), ('B', 2)]
   >>>R -> [('A', 2), ('R', 3), ('Q', 5)]
   >>>E -> [('A', 7)]
   >>>T -> []
   ```

5) Afficher le graphe **G2** avec votre fonction précédente.

Astuce :

- la fonction « print » permet d'afficher des variables et du texte
- la fonction « graphe.keys() » permet de retourner l'ensemble des clés d'un dictionnaire

Question 3

6) Transcrire le graphe **G1** en python en imaginant qu'il est orienté de haut en bas

7) Vérifier qu'il est valide à l'aide de la fonction « estValide »

Question 4

L'algorithme de *Dijsktra* s'effectue sur un graphe non orienté, il va donc falloir transformer notre graphe.

8) Écrire une fonction qui transforme un graphe orienté en graphe non orienté :
transformNonOriente(graphe) : graphe

Par exemple, dans le graphe **G1**, la liste des voisins de C était [G, H] en orienté, elle doit devenir [G, H, A] après la fonction.

9) Vérifier que votre fonction est correcte à l'aide de 2 tests :

- ```
  #on suppose que le graphe est stocké dans la variable g
  >>> transformNonOriente(g) != g
  True

  >>>transformNonOriente (transformNonOriente(g)) == transformNonOriente(g)
  True
  ```
  le premier test échoue, c'est que vous avez mal copié g.
- Si le second échoue, c'est que votre fonction fait grossir le graphe à l'infini.

Astuce :

- pour copier une variable en python :

```
import copy as cp
nouveauGraphe = cp.deepcopy(ancienGraphe)
```

- la fonction append() permet d'ajouter un élément dans une liste :

```
>>> l=[1,2,3];
>>> l
[1, 2, 3]
>>> l.append(4)
>>> l
```

Question 5

10) Écrire une fonction qui initialise le marquage de chaque nœud à False :
initialiserMarquage(graphe) : liste<booléen>
```

Cette liste sert à déterminer si on est déjà passé sur un sommet ou non. La taille de la liste retournée doit être égale au nombre de nœud.

11) Ecrire une fonction qui retourne une liste de nombre initialisé à l'infini :
initialiserDistance(graphe) : liste<nombre>

Cette liste représentera la distance à parcourir entre le nœud initial et le nœud de la liste. La taille de la liste doit être égale au nombre de nœud.

Astuces :

- l'infini peut être représenté par « float("inf") » en python
- la taille d'une liste peut être obtenu à l'aide de « len(liste) »

```
>>> l
[1, 2, 3, 4]
>>> len(l)
4
```

Question 6

A partir des 2 fonctions précédentes, on peut trouver le nœud qui n'a pas encore été marqué (False dans la liste des marquages) et dont la distance est la plus petite.

12) Écrire une fonction qui répond à ces critères :
noeudMin(marquages, distances) : entier

13) Écrire une fonction qui transforme un nœud en un entier :
noeudVersIndice(graphe, nœud) : entier

14) Écrire une fonction pour savoir si l'ensemble des nœuds a été marqué :
toutMarque(marquage) : booleen

Astuces :

- la fonction min retourne la valeur minimale d'une liste

```
>>> l
[3, 2, 5, 1]
>>> min(l)
1
```

- la fonction index retourne l'indice dans laquelle l'élément a été trouvé

```
>>> l
[3, 2, 5, 1]
>>> l.index(5)
2
```

- en combinant les 2, on peut trouver l'indice du plus petit élément dans une liste :

```
>>> l
[3, 2, 5, 1]
>>> l.index(min(l))
3
```

Question 7

15) Écrire une fonction qui met à jour la liste des distances à partir d'un nœud donné :
majDistance(graphe, noeud, distances) : void

On regarde si on peut aller plus rapidement à un des voisins du nœud en passant par celui ci pour tous les voisins. La nouvelle distance à un voisin devient : *min(distance[voisin], distance[noeud] + arc(noeud, voisin))*

16) Écrire la fonction **dijsktra(graphe, nœud)** à l'aide de toutes les autres fonctions.

Question 8 : Aller plus loin

17) Améliorer l'algorithme pour qu'il retienne également les chemins les plus courts en plus des distances.