T NSI - TP routage

Algorithme de Dijkstra

1. Introduction

Cet algorithme, publié en 1959, est dû à Edsger Wybe Dijkstra, mathématicien et informaticien néerlandais (1930 – 2002).
Dans un graphe orienté ou non, pondéré avec des poids positifs, l'algorithme de Dijkstra permet de trouver, à partir d'un sommet s de départ fixé à l'avance, les plus courts chemins menant à tout sommet accessible depuis s. C'est l'algorithme qui est utilisé dans les GPS pour déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région.
Dans le cadre d'un grand réseau informatique comprenant de nombreux routeurs, le protocole de routage OSPF utilise l'algorithme de Dijkstra pour déterminer la route la "moins côuteuse" (celle qui assurera les meilleurs débits en bits/s) pour faire passer de l'information d'une machine source à une machine destination.

2. Description de l'algorithme de Dijkstra en langage naturel

L'objectif est ici d'appliquer l'algorithme "à la main" sur un graphe de petites dimensions.
On se place ici dans le cadre d'un graphe non orienté. Les sommets directement reliés à un sommet donné par une arête sont dits "adjacents" à ce sommet. Pour appliquer l'algorithme à un graphe orienté, il suffit de remplacer le mot "adjacent" par le mot "suivant".

1) Initialisations :
-Marquer le sommet de départ D par le poids 0 et **fixer** ce sommet.
-Marquer chacun des sommets adjacents au sommet de départ D par le poids de l'arête joignant ce sommet à D et indiquer entre parenthèses le sommet de provenance (ici, D).
-Marquer les autres sommets par $+\infty$.

2) Regarder tous les sommets **non fixés**, repérer celui qui a **la plus petite marque**, et **fixer** ce sommet. Soit X ce sommet.
Pour chaque sommet Y **non fixé** adjacent à X, calculer la somme "marque de X + poids de l'arête reliant X et Y".
Si cette somme est inférieure à la marque de Y, barrer la marque de Y et la remplacer par cette somme, en indiquant entre parenthèses le sommet de provenance de cette nouvelle marque dite améliorée.

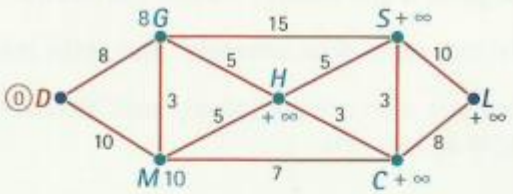
3) Itération : on recommence à partir du 2) jusqu'à ce que tous les sommets soient fixés.

A la fin de l'algorithme : toutes les marques ayant été améliorées, la marque du sommet Y fixé est le poids d'une plus courte chaîne entre les sommets D et Y.
Ainsi, le poids d'une plus courte chaîne entre D et le sommet d'arrivée L sera donné par la marque du sommet fixé L. Pour obtenir un plus court chemin entre D et L, on remonte à l'envers en regardant les sommets de provenance successifs (ceux entre parenthèses).

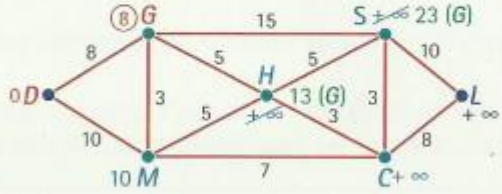
Exemple :

1) Initialisations :

• On fixe D à 0



• On fixe G à 8

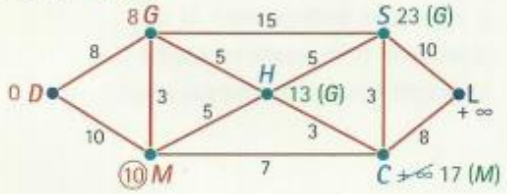


• Sommets de marque non fixée, adjacents à G

M	H	S
8 + 3 = 11	8 + 5 = 13	8 + 15 = 23
moins bon	on barre + ∞	on barre + ∞
on garde 10	on marque 13(G)	on marque 23(G)

2) 3) itérations successives :

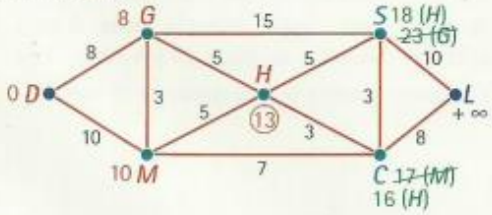
• On fixe M à 10



• Sommets de marque non fixée, adjacents à M

H	C
10 + 5 = 15	10 + 7 = 17
moins bon que 13	on barre + ∞
on garde 13	on écrit 17(M)

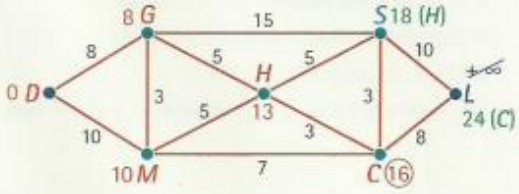
• On fixe H à 13



• Sommets de marque non fixée, adjacents à H

S	C
13 + 5 = 18	13 + 3 = 16
on barre 23	on barre 17
on écrit 18 (H)	on écrit 16 (H)

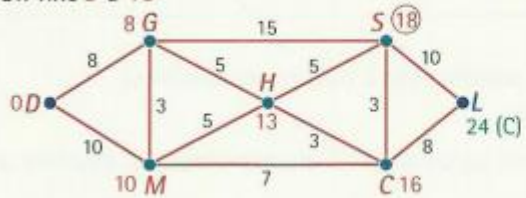
• On fixe C à 16



• Sommets de marque non fixée, adjacents à C

S	L
16 + 3 = 19	16 + 8 = 24
moins bon	on barre + ∞
on garde 18	on écrit 24 (C)

• On fixe S à 18



Il reste L comme sommet

18 + 10 = 28, moins bon, on garde 24

Fin de l'algorithme : une plus courte chaîne entre D et L a un poids de 24, obtenue par l'itinéraire D-G-H-C-L, obtenue en remontant dans les schémas et les sommets indiqués entre parenthèses.

Présentation à l'aide d'un tableau :

On peut placer les opérations successives de cet algorithme au fur et à mesure dans un tableau, en suivant sur le graphe pour ajouter à chaque marque le poids des arêtes.

sommet de marque fixée à chaque étape	sommets et marques améliorées						
	D	*G*	*M*	*H*	*C*	*S*	*L*
D : 0	0	8 (*D*)	10 (D)	+ ∞	+ ∞	+ ∞	+ ∞
G : 8				13 (*G*)		23 (*G*)	
M : 10					17 (*M*)		
H : 13					16 (*H*)	18 (*H*)	
C : 16							24 (*C*)
S : 18	0	8 (*D*)	10 (*D*)	13 (*G*)	16 (*H*)	18 (*H*)	24 (*C*)

3. Implémentation en Python

-On pourra représenter le graphe par un dictionnaire, dont chaque clé est un sommet du graphe et la valeur associée est la liste des sommets adjacents (c'est-à-dire voisins) avec le poids de chaque arête.

```
graphe={"D":[(8,"G"),(10,"M")],
        "G":[(8,"D"),(3,"M"),(15,"S"),(5,"H")],
        "M":[(10,"D"),(3,"G"),(5,"H"),(7,"C")],
        "H":[(5,"G"),(5,"S"),(3,"C"),(5,"M")],
        "S":[(15,"G"),(5,"H"),(3,"C"),(10,"L")],
        "C":[(7,"M"),(3,"H"),(3,"S"),(8,"L")],
        "L":[(10,"S"),(8,"C")]}
```

-Une valeur "infinie" sera représentée par la variable **`inf`** en tapant :

```
inf=float('inf')
```

-On initialisera **trois autres dictionnaires** :
`SNF = {}` (contiendra chaque sommet non fixé (clé) et la marque (poids) associée (valeur))
`SF = {}` (contiendra chaque sommet fixé (clé) et la marque (poids) associée (valeur))
`precedent = {}` (contiendra chaque sommet (clé) et le sommet de provenance (valeur))

a. Créer une fonction **`voisins(s)`** qui prend en paramètre un sommet (par exemple "D") et retourne la liste des sommets voisins avec le poids de l'arête.
Par exemple, **`voisins("D")`** doit retourner **[(8, "G"), (10, "M")]**.

b. Créer une fonction **`init(s)`** qui prend en paramètre le sommet de départ et :
met le sommet s dans le dictionnaire `SF` avec la valeur 0.
met dans le dictionnaire SNF les sommets voisins de s avec leurs poids associés, et tous les autres sommets (sauf s) avec le poids inf.
met dans le dictionnaire `precedent` : les sommets voisins de s avec le sommet de provenance (ici s), et tous les autres sommets avec pour valeur associée `None` (car ils n'ont aucun sommet de provenance pour le moment).

Tester cette fonction :

```
>>> init("D")
>>> SNF
{'G': 8, 'M': 10, 'H': inf, 'S': inf, 'C': inf, 'L': inf}
>>> SF
{'D': 0}
>>> precedent
{'D': None, 'G': 'D', 'M': 'D', 'H': None, 'S': None, 'C': None, 'L': None}
```

c. Créer une fonction **`minimum(dico)`** qui prend en paramètre un dictionnaire dont les clés sont des sommets et les valeurs sont la marque de chaque sommet, et qui retourne le sommet qui a la plus petite marque (le premier sommet rencontré en cas d'égalité).
Après avoir lancé init("D"), on doit avoir :

```
>>> minimum(SNF)
'G'
```

Il est permis d'utiliser la fonction Python **`min`**, à condition d'affecter au paramètre `key` une fonction qui retourne la valeur associée à une clé, car on veut classer les sommets par rapport à leur valeur.

d. Créer la fonction principale **`Dijkstra( s , t )`** qui prend en paramètre un sommet s de départ et un sommet t d'arrivée et retourne la distance minimale entre s et t, ainsi que le chemin minimal sous forme d'une liste de sommets de la forme [s, …., t].
Dans l'exemple précédent :

```
>>> Dijkstra("D","L")
(24, ['D', 'G', 'H', 'C', 'L'])
```

La fonction Dijkstra commence par :

```
def Dijkstra( s , t) :
    init(s)
    while SNF != {}:
        X = minimum(SNF)
        #Ensuite, il faut fixer X, donc l'enlever de SNF et
          #le mettre dans SF
        #Pour chaque voisin Y de X non fixé,
        #si marque(X) + poids(X,Y) < marque(Y)
        #alors remplacer marque(Y) par cette somme
        #et faire precedent[Y]=X
    #à la sortie du while, récupérer la distance mini, puis
    #reconstituer le chemin avec le dictionnaire precedent.
```