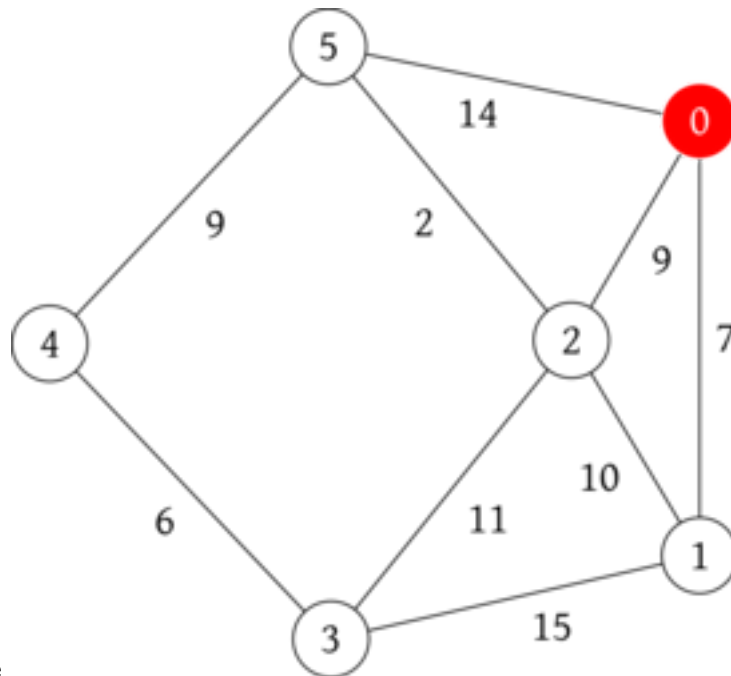


Dijkstra

September 6, 2022

1 Graphes pondérés

Pour représenter un graphe pondéré on va utiliser des listes d'adjacences où les éléments ne sont plus seulement les indices des voisins mais des couples (y, w) où y est le sommet et w est le poids de l'arête.



Ainsi le graphe

sera représenté par :

```
[3]: g = [  
    [(1,7), (2,9), (5,14) ],  
    [(0, 7), (2, 10), (3, 15) ],  
    [(0, 9), (1, 10), (3, 11), (5, 2) ],  
    [(1,15), (2,11), (4, 6)],  
    [(3,6), (5,9)],  
    [(0,14), (2,2), (4,9)]  
]
```

```
[1]: def poids(g,x,y):  
    # prend en entrée un graphe pondéré et renvoie soit le poids  
    # de l'arête x->y quand une telle arête existe ou la valeur spéciale   
    ↪ float('inf')
```

```

# qui, pour ce qu'on va faire, se comporte comme oo

# BEGIN SOLUTION
for z, w in g[x]:
    if z == y:
        return w
return float('inf')
# END SOLUTION

```

```

[6]: from civ import *
init()
test(poids, g, 3, 4, 6)
test(poids, g, 4, 2, float('inf'))
valide()

```

OK!: poids([[(1, 7), (2, 9), (5, 14)], [(0, 7), (2, 10), (3, 15)], [(0, 9), (1, 10), (3, 11), (5, 2)], [(1, 15), (2, 11), (4, 6)], [(3, 6), (5, 9)], [(0, 14), (2, 2), (4, 9)]], 3, 4) vaut 6

OK!: poids([[(1, 7), (2, 9), (5, 14)], [(0, 7), (2, 10), (3, 15)], [(0, 9), (1, 10), (3, 11), (5, 2)], [(1, 15), (2, 11), (4, 6)], [(3, 6), (5, 9)], [(0, 14), (2, 2), (4, 9)]], 4, 2) vaut inf

Tous les tests sont valides

```

[7]: def poids_chemin(g, ch):
    # prend en entr e un graphe pond r e g et une liste de sommet ch
    # repr esentant un chemin. Renvoie le poids du chemin, c'est- -dire la somme
    # des poids de ses ar tes

    # BEGIN SOLUTION
    p = 0
    for i in range(len(ch)-1):
        p += poids(g, ch[i], ch[i+1])
    return p
    # END SOLUTION

```

```

[10]: from civ import *
init()
test(poids_chemin, g, [0,2,1,3,4], 40)
test(poids_chemin, g, [0], 0)
valide()

```

OK!: poids_chemin([[(1, 7), (2, 9), (5, 14)], [(0, 7), (2, 10), (3, 15)], [(0, 9), (1, 10), (3, 11), (5, 2)], [(1, 15), (2, 11), (4, 6)], [(3, 6), (5, 9)], [(0, 14), (2, 2), (4, 9)]], [0, 2, 1, 3, 4]) vaut 40

OK!: poids_chemin([[(1, 7), (2, 9), (5, 14)], [(0, 7), (2, 10), (3, 15)], [(0, 9), (1, 10), (3, 11), (5, 2)], [(1, 15), (2, 11), (4, 6)], [(3, 6), (5, 9)], [(0, 14), (2, 2), (4, 9)]], [0]) vaut 0

Tous les tests sont valides

2 Algorithme de Dijkstra

On rappelle l'algorithme de Dijkstra vu en classe :

Algorithme de Dijkstra

Entrée

- Un graphe pondéré $G = (S, A, \pi)$ où $\pi : A \rightarrow \mathbb{R}_+$
- Un sommet $s \in S$

Algorithme

- On crée un tableau d avec $d(s) = 0$ et $d(x) = \infty$ pour $x \neq s$.
- On crée un tableau `a_traiter` et on ajoute `s`
- Les sommets sont initialement non marqués
- Tant que `a_traiter` est non vide
 - On retire le sommet `x` de `a_traiter` de valeur $d(x)$ **minimale**
 - Si `x` est non marqué
 - * On marque `x`
 - * Pour chaque arête $x \rightarrow y$ où $d(x) + \pi(x \rightarrow y) < d(y)$
 - On pose $d(y) = d(x) + \pi(x \rightarrow y)$
 - On ajoute `y` dans `a_traiter`

2.1 Stratégie pour retirer un sommet

Pour cela on commence par écrire des fonctions permettant de réaliser ce retrait dans une liste. On ne cherche pas ici à ce que les opérations soient efficaces.

```
[3]: def minimum(l, d):  
    # l est un tableau de p valeurs entre 0 et n-1 et d est un tableau de n  
    ↪valeurs  
    # renvoie i dans l tel que d[i] soit minimal  
    # BEGIN SOLUTION  
    m = l[0]  
    for x in l:  
        if d[m] > d[x]:  
            m = x  
    return m  
    # END SOLUTION
```

```
[4]: from civ import *  
init()  
test(minimum, [0,1,2,3], [0,1,2,3], 0)  
test(minimum, [0,1,2,3], [3,0,1,2], 1)  
test(minimum, [0], [12], 0)  
test(minimum, [5], [0,1,2,3,4,5], 5)  
valide()
```

OK!: minimum([0, 1, 2, 3], [0, 1, 2, 3]) vaut 0
OK!: minimum([0, 1, 2, 3], [3, 0, 1, 2]) vaut 1
OK!: minimum([0], [12]) vaut 0
OK!: minimum([5], [0, 1, 2, 3, 4, 5]) vaut 5
Tous les tests sont valides

Pour retirer un élément dans un tableau, on peut utiliser `.remove` comme on peut le voir dans cet exemple :

```
[9]: t = [1,2,3]
      t.remove(2)
      t
```

```
[9]: [1, 3]
```

```
[19]: def extrait_minimum(l, d):
      # renvoie i dans l tel que d[i] soit minimal
      # et l'enleve de l
      # BEGIN SOLUTION
      m = minimum(l, d)
      l.remove(m)
      return m
      # END SOLUTION
```

```
[20]: from civ import *
      init()
      t = [0,1,2,3]
      test(extrait_minimum, t, [3,0,1,2], 1)
      assert(t == [0,2,3])
      test(extrait_minimum, t, [3,0,1,2], 2)
      assert(t == [0,3])
      valide()
```

OK!: extrait_minimum([0, 1, 2, 3], [3, 0, 1, 2]) vaut 1
OK!: extrait_minimum([0, 2, 3], [3, 0, 1, 2]) vaut 2
Tous les tests sont valides

2.2 Implémentation de Dijkstra

```
[21]: def dijkstra(g, src):
      # Renvoie le tableau d des distances à src
      # où g est un graphe pondéré
      # BEGIN SOLUTION
      n = len(g)
      inf = float('inf')
      d = [ inf ] * n
      d[src] = 0
      a_traiter = [ src ]
```

```

marques = [ False ] * n
while a_traiter != []:
    x = extrait_minimum(a_traiter, d)
    if not marques[x]:
        marques[x] = True
        for y, poids in g[x]:
            n_d = d[x] + poids
            if n_d < d[y]:
                d[y] = n_d
                a_traiter.append(y)
return d # distandes finales
#END SOLUTION

```

```

[23]: from civ import *
init()
test(dijkstra, g, 0, [0, 7, 9, 20, 20, 11])
valide()

```

OK!: dijkstra([[(1, 7), (2, 9), (5, 14)], [(0, 7), (2, 10), (3, 15)], [(0, 9), (1, 10), (3, 11), (5, 2)], [(1, 15), (2, 11), (4, 6)], [(3, 6), (5, 9)], [(0, 14), (2, 2), (4, 9)]], 0) vaut [0, 7, 9, 20, 20, 11]

Tous les tests sont valides

3 Vers l'infini et au-delà

Maintenant, vous voilà pret à affronter des problèmes de graphes en ligne sur des sites de programmation. Voici une selection de problèmes de difficulté croissante :

- [HIGHWAYS](#)
- [MICEMAZE](#)
- [TRAFFICN](#)
- [SAMERO8A](#)

Si vous ne vous en sentez pas capables ce n'est pas grave et c'est sûrement une bonne idée de reprendre les TPs précédents pour consolider vos connaissances.

```

[ ]: n = int(input())
a, b = map(int, input().split())
n, a, b

```