td1a_correction_session7

January 21, 2022

1 1A.algo - Programmation dynamique et plus court chemin (correction)

Correction.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

On récupère le fichier `matrix_distance_7398.txt` depuis matrix_distance_7398.zip qui contient des distances entre différentes villes (pas toutes).

[2]:
```
import pyensae.datasource
pyensae.datasource.download_data("matrix_distance_7398.zip", website  = "xd")
```

[2]:
```
['matrix_distance_7398.txt']
```

[3]:
```
import pandas
df = pandas.read_csv("matrix_distance_7398.txt", sep="\t", header=None,␣
  ↪names=["v1","v2","distance"])
matrice = df.values
matrice[:5]
```

[3]:
```
array([['Boulogne-Billancourt', 'Beauvais', 85597],
       ['Courbevoie', 'Sevran', 26564],
       ['Colombes', 'Alfortville', 36843],
       ['Bagneux', 'Marcq-En-Baroeul', 233455],
       ['Suresnes', 'Gennevilliers', 10443]], dtype=object)
```

1.1 Exercice 1

[4]:
```
vil = { }
for row in matrice :
    vil [row[0]] = 0
    vil [row[1]] = 1
vil = list(vil.keys())
print (len(vil))
```

```
196
```

1.2 Exercice 2

La distance n'existe pas encore. L'exception du court programme suivant le montre. Rejoindre Bordeaux depuis Charleville nécessite plusieurs étapes.

[5]:
```
dist = { }
for row in matrice :
    a = row[0]
    b = row[1]
    dist[a,b] = dist[b,a] = row[2]
print (len(dist))
```

```
7888
```

[6]:
```
print ( dist["Charleville-Mezieres","Bordeaux"] )  # elle n'existe pas encore
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-6-0227958f5453> in <module>()
----> 1 print ( dist["Charleville-Mezieres","Bordeaux"] )  # elle n'existe pas encore

KeyError: ('Charleville-Mezieres', 'Bordeaux')
```

1.3 Exercice 3

On peut remplir facilement toutes les cases correspondant aux villes reliées à Charleville-Mézières, c'est-à-dire toutes les villes accessibles en une étape.

[7]:
```
d = { }
d['Charleville-Mezieres'] = 0
for v in vil : d[v] = 1e10
for v,w in dist :
    if v == 'Charleville-Mezieres':
        d[w] = dist[v,w]
print(len(d))
```

```
196
```

1.4 Exercice 4

Si on découvre que $d[w] > d[v] + dist[w, v]$, cela veut dire qu'il faut mettre à jour le tableau d car il ne contient pas la distance optimale. On répète cela pour toutes les paires (v, w).

[8]:
```
for v,w in dist :
    d2 = d[v] + dist[v,w]
    if d2 < d[w] :
        d[w] = d2
print ( d["Bordeaux"] )
```

```
798824
```

On trouve 813197 mètres pour la distance (Charleville-Mezieres, Bordeaux). Ce n'est pas forcément la meilleure. Pour être sûr, il faut répéter la même itération autant de fois qu'il y a de villes (car le plus long chemin contient autant d'étapes qu'il y a de villes).

[9]:
```
for i in range(0,len(d)) :
    for v,w in dist :
        d2 = d[v] + dist[v,w]
        if d2 < d[w] :
            d[w] = d2
print ( d["Bordeaux"] )
```

```
795670
```

1.5 Exercice facultatif

Pour montrer que l'algorithme suggéré permettra d'obtenir la solution optimale, il faut montrer qu'il n'est pas nécessaire d'envisager aucun autre ordre que celui des skieurs et des paires triés par taille croissante. Cela ne veut pas dire qu'un autre ordre ne sera pas optimal, cela veut dire que pour obtenir l'appariement de coût optimal, il existe une solution pour laquelle skieurs et skis sont rangés dans l'ordre.
On considère donc un appariement σ qui associé le skieur t_i à la paire $s_{\sigma(i)}$. Il suffit que montrer que :
$\forall i,j,\ t_i \leqslant t_j \iff s_{\sigma(i)} \leqslant s_{\sigma(j)}$
Pour montrer cela, on fait un raisonnement par l'absurde : pour i et j quelconques, on suppose qu'il existe un appariement optimal tel que $t_i \geqslant t_j$ et $s_{\sigma(i)} < s_{\sigma(j)}$. Le coût $C(\sigma)$ de cet appariement est :
$C(\sigma) = \sum_{k=1}^{N} \left|t_k - s_{\sigma(k)}\right| = \alpha + \left|t_i - s_{\sigma(i)}\right| + \left|t_j - s_{\sigma(j)}\right|$
Le coût de l'appariement en permutant les skieurs i et j (donc en les rangeant dans l'ordre croissant) est :
$C(\sigma') = \sum_{k=1}^{N} \left|t_k - s_{\sigma(k)}\right| = \alpha + \left|t_j - s_{\sigma(i)}\right| + \left|t_i - s_{\sigma(j)}\right|$
On calcule :
$C(\sigma) - C(\sigma') = \left|t_i - s_{\sigma(i)}\right| + \left|t_j - s_{\sigma(j)}\right| - \left|t_j - s_{\sigma(i)}\right| - \left|t_i - s_{\sigma(j)}\right|$
Premier cas $t_j \geqslant s_{\sigma(i)}$ et $t_i > t_j \geqslant s_{\sigma(i)}$ et :

$$\begin{aligned} C(\sigma) - C(\sigma') &= \left|t_i - s_{\sigma(i)}\right| + \left|t_j - s_{\sigma(j)}\right| - \left(t_j - s_{\sigma(j)} + s_{\sigma(j)} - s_{\sigma(i)}\right) - \left|t_i - s_{\sigma(j)}\right| \\ &= t_i - s_{\sigma(i)} + \left|t_j - s_{\sigma(j)}\right| - \left(t_j - s_{\sigma(j)}\right) - \left(s_{\sigma(j)} - s_{\sigma(i)}\right) - \left|t_i - s_{\sigma(j)}\right| \\ &= t_i - s_{\sigma(i)} - \left|t_i - s_{\sigma(i)}\right| + \left|t_j - s_{\sigma(j)}\right| - \left(t_j - s_{\sigma(j)}\right) \\ &= \left|t_j - s_{\sigma(j)}\right| - \left(t_j - s_{\sigma(j)}\right|) \\ &\geqslant 0 \end{aligned}$$

Second cas $t_j \leqslant s_{\sigma(i)}$ et $t_j \leqslant s_{\sigma(i)} \leqslant s_{\sigma(j)}$ et :

$$\begin{aligned} C(\sigma) - C(\sigma') &= \left|t_i - s_{\sigma(i)}\right| + s_{\sigma(j)} - t_j - \left(s_{\sigma(i)} - t_j\right) - \left|t_i - s_{\sigma(j)}\right| \\ &= \left|t_i - s_{\sigma(i)}\right| + s_{\sigma(j)} - s_{\sigma(i)} - \left|t_i - s_{\sigma(j)}\right| \\ &\geqslant \left|t_i - s_{\sigma(j)}\right| - \left|s_{\sigma(j)} - s_{\sigma(i)}\right| + s_{\sigma(j)} - s_{\sigma(i)} - \left|t_i - s_{\sigma(j)}\right| \\ &\geqslant 0 \end{aligned}$$

Dans les deux cas, on montre donc qu'il existe un appariement meilleur ou équivalent en permutant les deux skieurs i et j, c'est-à-dire en les triant par ordre croissant de taille. Nous avons donc montré que, si les paires de ski sont triées par ordre croissant de taille, il existe necéssairement un appariement optimal pour lequel les skieurs sont aussi triés par ordre croissant. Lors de la recherche de cet appariement optimal, on peut se restreindre à ces cas de figure.

1.6 Exercice 5

$p(n,m) = \min\left\{p(n-1,m-1) + |t_n - s_m|, p(n,m-1)\right\}$
Lorsqu'on considère le meilleur appariement des paires $1..m$ et des skieurs $1..n$, il n'y a que deux choix possibles pour la paire m :

- soit elle n'est associée à aucun skieur et dans ce cas : $p(n,m) = p(n,m-1)$,
- soit elle est associée au skieur n (et à aucun autre) : $p(n,m) = p(n-1,m-1) + |t_n - s_m|$.

1.7 Exercice 6

[10]:
```
import random
skieurs = [ random.gauss(1.75, 0.1) for i in range(0,10) ]
paires  = [ random.gauss(1.75, 0.1) for i in range(0,15) ]
skieurs.sort()
paires.sort()

p = { }
p [-1,-1] = 0
for n,taille in enumerate(skieurs) : p[n,-1] = p[n-1,-1] + taille
for m,paire  in enumerate(paires ) : p[-1,m] = 0
for n,taille in enumerate(skieurs) :
    for m,paire in enumerate(paires) :
        p1 = p.get ( (n  ,m-1), 1e10 )
        p2 = p.get ( (n-1,m-1), 1e10 ) + abs(taille - paire)
        p[n,m] = min(p1,p2)

print (p[len(skieurs)-1,len(paires)-1])
```

```
0.40180280093469833
```

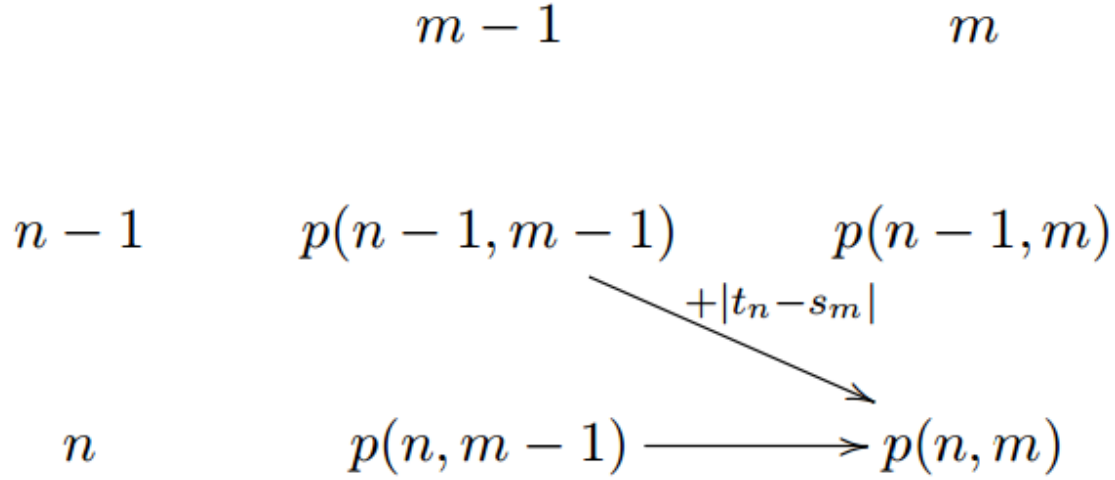
1.8 Exercice 7

Il faut imaginer que p peut être représenté sous forme de matrice et qu'à chaque fois, on prend le meilleur chemin parmi 2 :

- Chemin horizontal : on ne choisit pas la paire m.
- Chemin diagonal : on choisit la paire m pour le skieur n

[11]:
```
from pyquickhelper.helpgen import NbImage
NbImage('graph_notebook_ski.png')
```

[11]:




```
\xymatrix{  & m-1  & m \\      n-1   &  p(n-1,m-1) \ar[dr]^{ + \abs{t_n - s_m}} &
p(n-1,m) \\      n     &  p(n,m-1)   \ar[r] & p(n,m) \\ }
```

[12]:
```
p = { }
p [-1,-1] = 0
best = { }
for n,taille in enumerate(skieurs) : p[n,-1] = p[n-1,-1] + taille
for m,paire  in enumerate(paires ) : p[-1,m] = 0
for n,taille in enumerate(skieurs) :
    for m,paire in enumerate(paires) :
        p1 = p.get ( (n  ,m-1), 1e10 )
        p2 = p.get ( (n-1,m-1), 1e10 ) + abs(taille - paire)
        p[n,m] = min(p1,p2)

        if p[n,m] == p1 : best [n,m] = n,m-1
        else : best [n,m] = n-1,m-1

print (p[len(skieurs)-1,len(paires)-1])

chemin = [ ]
pos = len(skieurs)-1,len(paires)-1
while pos in best :
    print (pos)
    chemin.append(pos)
    pos = best[pos]
chemin.reverse()
print (chemin)
```

```
0.40180280093469833
(9, 14)
(8, 13)
(7, 12)
(6, 11)
(5, 10)
(4, 9)
(3, 8)
(2, 7)
(1, 6)
(0, 5)
(0, 4)
[(0, 4), (0, 5), (1, 6), (2, 7), (3, 8), (4, 9), (5, 10), (6, 11), (7, 12), (8,
13), (9, 14)]
```

1.9 Exercice 8

Les deux algorithmes ont un coût quadratique.

1.10 Prolongements : degré de séparation sur Facebook

[13]:
```
import pyensae.datasource # utiliser pyensae >= 0.8
files = pyensae.datasource.download_data("facebook.tar.gz",website="http://snap.
  ↪stanford.edu/data/")
import pandas
df = pandas.read_csv("facebook/1912.edges", sep=" ", names=["v1","v2"])
print(df.shape)
df.head()
```

```
(60050, 2)
```

[13]:

	v1	v2
0	2290	2363
1	2346	2025
2	2140	2428
3	2201	2506
4	2425	2557

[14]:

[15]: