8 Algorithmes gloutons

July 26, 2024

1 1. Rendu de monnaie

1.1 Version récursive

[1]:
```
def rendu(s:int, pieces=(200, 100, 50, 20, 10, 5, 2, 1))->[int]:
    """
    Retourne la liste des pièces qu'on doit rendre, dans l'ordre décroissant.
    """
    if s == 0 or pieces == []:
        return []
    else:
        if s - pieces[0] >= 0:
            return [pieces[0]] + rendu(s - pieces[0], pieces)
        else:
            return rendu(s, pieces[1:])
```

[2]:
```
rendu(479)
```

1.2 Avec une complexité optimale

Pire cas : on parcourt toute la liste de pièces et ensuite on rend systématiquement la pièce de valeur 1 (si elle existe) jusqu'à ce que la somme à rendre soit nulle. La complexité est alors maximale en $O(S + n)$ avec n la taille de la liste de pièces.

La complexité est donc bornée par une fonction linéaire des deux tailles du problème, mais on peut faire mieux.

[3]:
```
def rendu(s:int, pieces=(200, 100, 50, 20, 10, 5, 2, 1))->[int]:
    """
    Retourne la liste des pièces qu'on doit rendre, dans l'ordre décroissant.
    """
    if s == 0 or pieces == []:
        return []
    else:
        return [pieces[0]]*(s//pieces[0]) + rendu(s%pieces[0], pieces[1:])
```

[4]:
```
rendu(479)
```

1.3 Avec un dictionnaire

[5]:
```
def rendu(s:int, pieces=(200, 100, 50, 20, 10, 5, 2, 1), d=None)->dict:
    """
    Retourne un dictionnaire (pièce:occurrences) qui correspond au rendu de␣
 ↪monnnaie.
    """
    if d is None:
        d = {}

    if s == 0 or pieces == []:
        return d
    else:
        n = s//pieces[0]
        if n != 0:
            d[pieces[0]] = n
        return rendu(s%pieces[0], pieces[1:], d)
```

[6]:
```
rendu(479)
```

1.4 Version impérative

[7]:
```
def rendu(s:int, pieces=(200, 100, 50, 20, 10, 5, 2, 1))->[int]:
    """
    """
    r = []
    i = 0
    while s != 0 and i < len(pieces):
        if s - pieces[i] >= 0:
            r.append(pieces[i])
            s -= pieces[i]
        else:
            i += 1
    return r
```

[8]:
```
rendu(479)
```

2 2. Problème du voyageur de commerce

[18]:
```
from random import random
import matplotlib.pyplot as plt
from math import sqrt
from copy import deepcopy
```

[19]:
```
def gen_pts(n:int)->[(float, float)]:
    """
    Retourne une liste de n points aléatoires.
```

```
    """
    return [(random(), random()) for i in range(n)]
```

[20]:
```
def unzip(points:[(float, float)])->([float], [float]):
    """
    Retourne la liste des abscisses et la liste des ordonnées
    des points de la liste de points.
    """
    lix, liy = [], []
    for pt in points:
        x, y = pt
        lix.append(x)
        liy.append(y)
    return lix, liy
```

[21]:
```
def affiche(points:[(float, float)], chemin:[(float, float)])-> None:
    """
    Affiche le nuage de points et le chemin.
    """
    Xc, Yc = unzip(chemin)
    Xp, Yp = unzip(points)
    fig, ax = plt.subplots()
    ax.scatter(Xp, Yp)
    ax.plot(Xc, Yc, alpha=0.6, ls='--')
    for i in range(len(chemin)):
        x, y = chemin[i]
        ax.text(x, y, str(i), fontsize=20)
    ax.axis('equal')
```

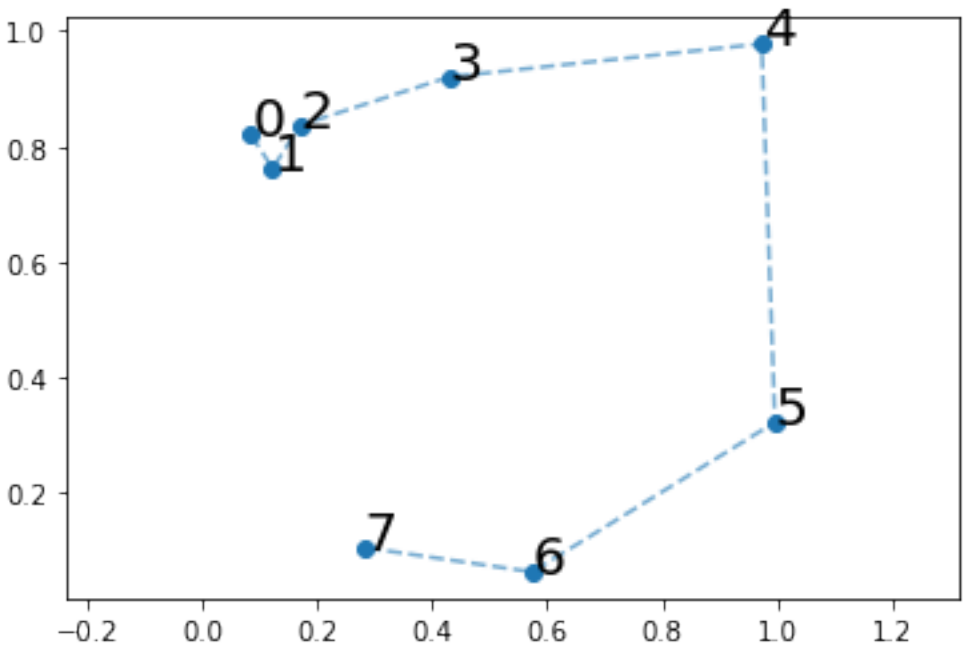
[22]:
```
def d(A:(float, float), B:(float, float))->float:
    """
    Retourne la distance euclidienne entre les points A et B
    """
    xA, yA = A
    xB, yB = B
    return sqrt((xB - xA)**2 + (yB - yA)**2)
```

[23]:
```
def nextPt(P:(float, float), points:[(float, float)])->int:
    """
    Retourne l'indice du point de la liste points qui correspond
    au point Q le plus proche du point P.
    """
    Q = points[0]
    imin = 0
    dmin = d(P, Q)
    for i in range(1, len(points)):
        if d(P, points[i]) < dmin:
```

```
            Q = points[i]
            imin = i
            dmin = d(P, points[i])
    return imin
```

[24]:
```
def chemin_glouton(points:[(float, float)], i0=0)->[(float, float)]:
    """
    Retourne le chemin qui correspond à l'algorithme glouton
    appliqué sur la liste de points.
    """
    pts = deepcopy(points)
    P = pts.pop(i0)
    ch = [P]
    while pts != []:
        iP = nextPt(P, pts)
        P = pts.pop(iP)
        ch.append(P)
    return ch
```

[25]:
```
points = gen_pts(8)
affiche(points, chemin_glouton(points))
```



[ ]: