Compte rendu d'Algo

chrifmhm_mohameml

2023-05-04

I.Algo de notre Programme :

Dans notre programme connectes.py on a implementer deux fonctions :

1.la fonction comm_connexe :

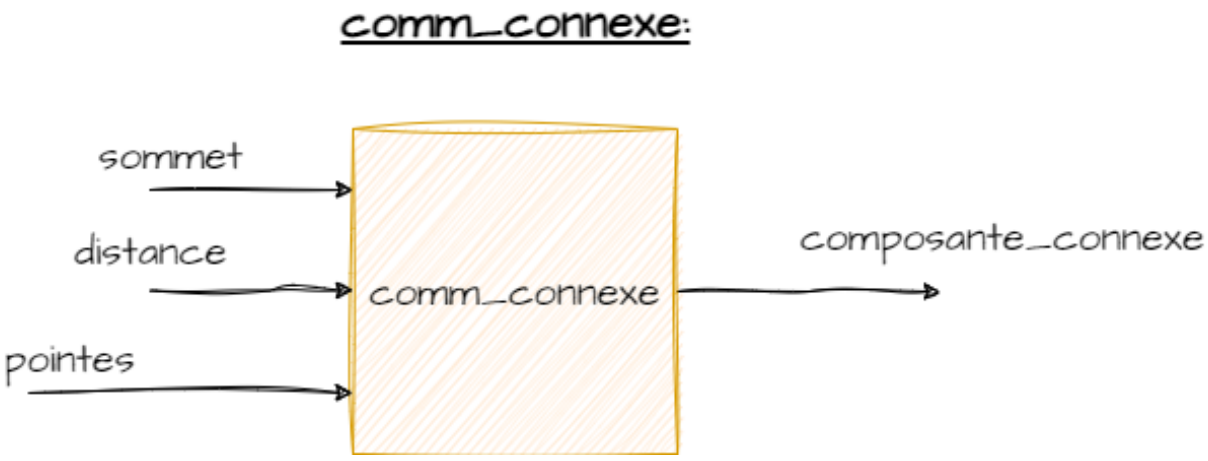

Figure 1: la fonction comm_connexe

- la fonction comm_connexe :
  - prend a l' entrée : une sommet et l'ensembles des points et la distance suiel.
  - et donne à la sortie : la composante connexe à laquelle apartient cette sommet .

```
def comm_connexe(sommet,points,distance):

    col={s:"blanc" for s in points}
    col[sommet]="gris"
    pile=[sommet]


    while pile :

        # on prend la  tete de la pile
        u=pile[-1]
```

```
        # on cherche les enfants non parcouris
        R=[y for y in points if col[y]=="blanc"  and  u.distance_to(y) < distance ]

        if R :
             v=R[0] # on prend le premiere sommet de R
             col[v]="gris" # on le marque comme visite
             pile.append(v) # on l'empile v dans la pile

        else :
            pile.pop()


    return  [s for s in points if col[s]=="gris"]
```

voici le code de la fonction comm_connexe :

Explication de code de la fonction comm_connexe :

Dans la fonction comm_connexe on a utiliser l'idéé de parcour en profondeur d'un graphe (DFS).

On commance avec le sommet fournis en paramétre et on explore chaque branche complétement avant d'explorer la suivante : On utilise une pile *LIFO(last In/First Out)*

Pour la mise en place de l'algo , on va proceder en mettant les sommets successifs dans une pile :

i. on commance par marquer que toutes les sommets sont blancs (n'est pas encore visites) et on empile le sommete de départ .

ii.si le sommet de la pile posséde des voisins *ie:distance(sommet,autre)< seuil* qui ne sont pas deja visites *(dans le code :col[y]==“blanc”)* , on sélectionne l'un de ces voisins et on l'empile et on le marque comme visite *dans le code col[v]=“gris”*

iii. si non, on le dépile

iv. Tant que la pile n'est pas vide , On réittire sur ii et iii .

v. on return a la fin l'ensembles des points marquer comme “gris” qui correspendetn bien a la composantes connexe à laquelle appartient le sommet fournis en paramétre .

2. la fonction print_components_sizes :

- la fonction print_components_sizes :
  - prend a l' entrée : l'ensembles des points et la distance suiel.
  - et donne à la sortie : les tailles des composantes conexes triéés par ordre décroissante .

Explication de code de la fonction comm_connexe :

i.L'ideé ici consiste a éviter la redondance donc pour cela on utiliser un dictionnaire *col* et on a marquer toutes les sommets *blanc* (ne sont pas encore traites)

ii. On boucle sur toutes les pointes : si le point n'est pas encore traite ie *col[point]!=“noir”* on recuper son composante connexes et on ajout dans la liste tailles la taille de cette composante connexes et On marque toutes les points de cette composantes connexes comme des pointes traites (pour eviter la redondance) ie *col[s]=“noir”* .

iii. On affiche la listes des tailles des composantes conexes triéés par ordre décroissante

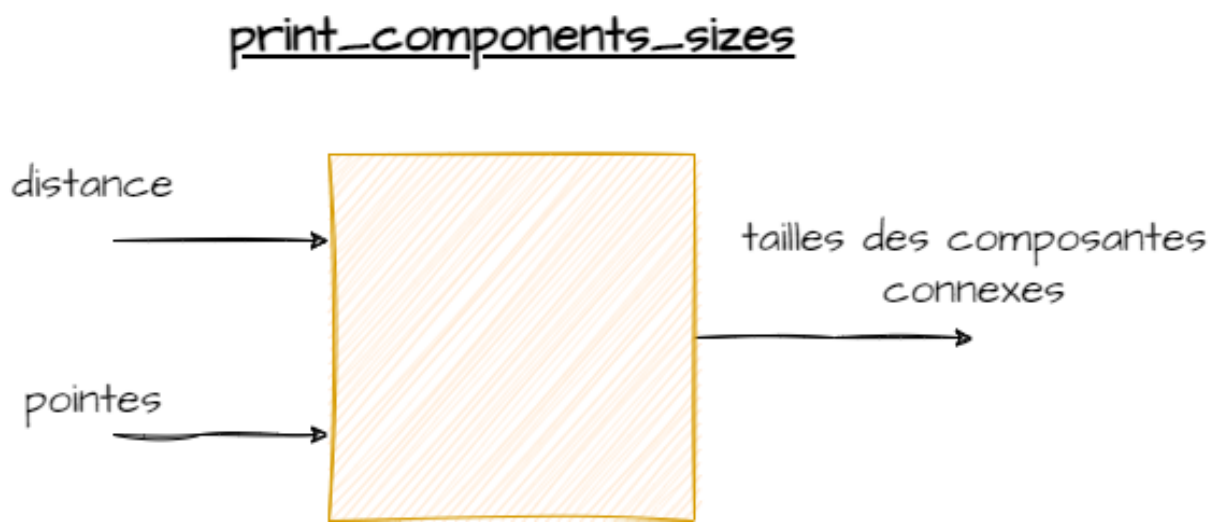

Figure 2: la fonction print_components_sizes

```
def print_components_sizes( distance, vocabulaire):

    " affichage des tailles triees de chaque composante "


    # Les tailles des  composantes connexes :
    tailles=[]


    col={v:"blanc" for v in vocabulaire}
    for point in vocabulaire :

        if col[point]!="noir" :
            com=comm_connexe(point,vocabulaire,distance)
            tailles.append(len(com))

        for s in com :
            col[s]="noir"

    # affichage final
    tailles.sort(reverse=True)
    print(tailles)
```

voici le code de la fonction comm_connexe :

II . La complexite de notre programme :

Pour calcluer la complexite:

1.On a develloper tout d'aborde une fonction qui génre dans un fichier des nombres aleatoires entre [0,1] et prend un entree un entier k

Donc voici le code de cette fonction

```
def fichier_aléatoire(k):
    fich=open(f"exemples_{k}.pts","w")
    print(f"{k/50000}",file=fich)
    for _ in range(k) :
        n1=random.random()/10
        n2=random.random()/10
        print(f'{n1},',n2,file=fich)

    fich.close()
```

2. Apres la fonction *fichier_aléatoire(k)* on a utiliser la biblo time pour calculer le temps d'execution de notre programme

et la biblo matplotlib pour dessiner la courbe de complexite.

```
import time
import matplotlib.pyplot as plt
from connectes import main

nb_points=[k for k in range(1000,4000,100)]
temps=[]

for i in nb_points :

    fichier_aléatoire(i)
    fich=f"exemples_{i}.pts"
    t1=time.time()
    main(fich)
    t2=time.time()
    temps.append(t2-t1)


y_nlogn=[n*np.log(n) for n in nb_points]
y_n_2=[n**2 for n in nb_points]
y_n_3=[n**3 for n in nb_points]

plt.plot(nb_points,temps,label="temps d'exc")
plt.plot(nb_points,y_nlogn,label="nlog(n)")
plt.plot(nb_points,y_n_2,label="n au carre")
plt.plot(nb_points,y_n_3,label="n au cube")
plt.legend()
plt.show()
```

Courbe de complexite :

III. Des Autres Outiles :

Pour la verificattion de la sortie de notre programme on developpe en progaarmme a l'aide de la module tycat Qui affiche le graphe de en argument.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

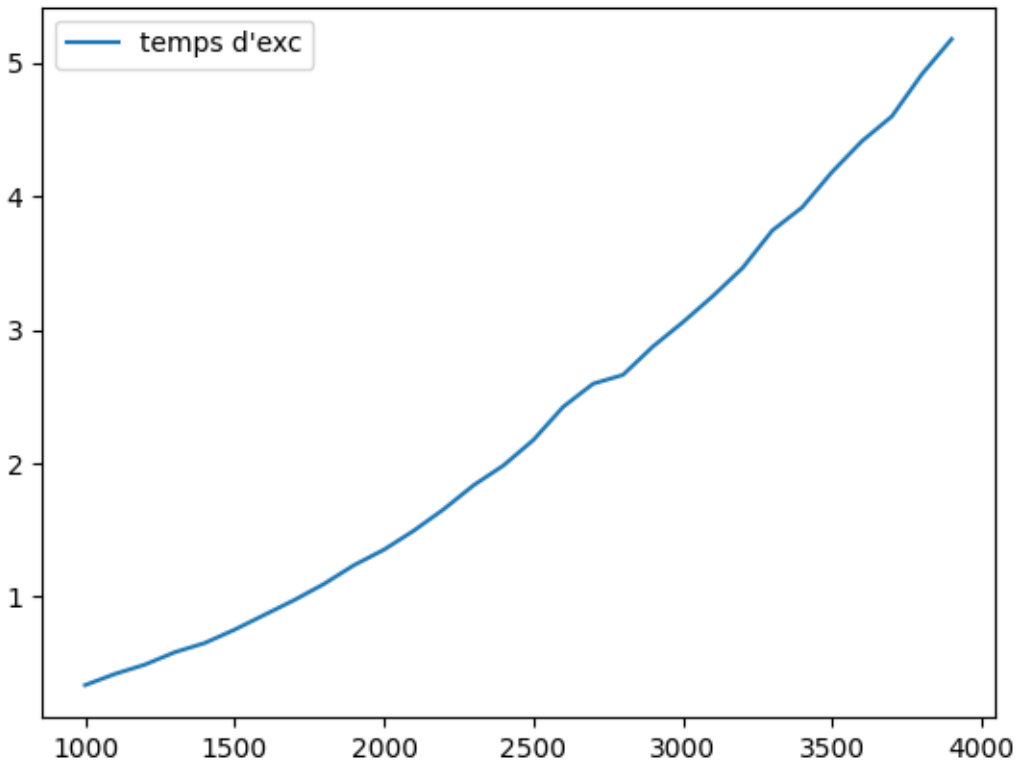

Figure 3: Courbe de complexite

```
import sys
from geo.point import Point
from geo.tycat import tycat
from geo.segment import Segment


def load_instance(filename):
    """
    charge le fichier .pts.
     renvoie la limite de distance et les points.
    """
    with open(filename, "r") as instance_file:
        lines = iter(instance_file)
        distance = float(next(lines))
        points = [Point([float(f) for f in l.split(",")]) for l in lines]

    return distance, points

def print_components_sizes( distance, pts):

    # Remplissage de notre graphe G

    v=pts
    G={s:[] for s in  pts}
    for  pi   in v :
         for pj  in v :
             if pj.distance_to(pi) < distance and pj!=pi:
                        G[pi].append(pj)

    # representation de notre graphe avec tycat
    rep=[v]
    for cle , valeur in G.items():
        for point in valeur :
            rep.append(Segment([cle,point]))

    tycat(rep)







def main():
    """
    ne pas modifier: on charge une instance et on affiche les tailles
    """
    for instance in sys.argv[1:]:
        distance, points = load_instance(instance)
        print_components_sizes(distance, points)

main()
```

voici le code :

Demo :

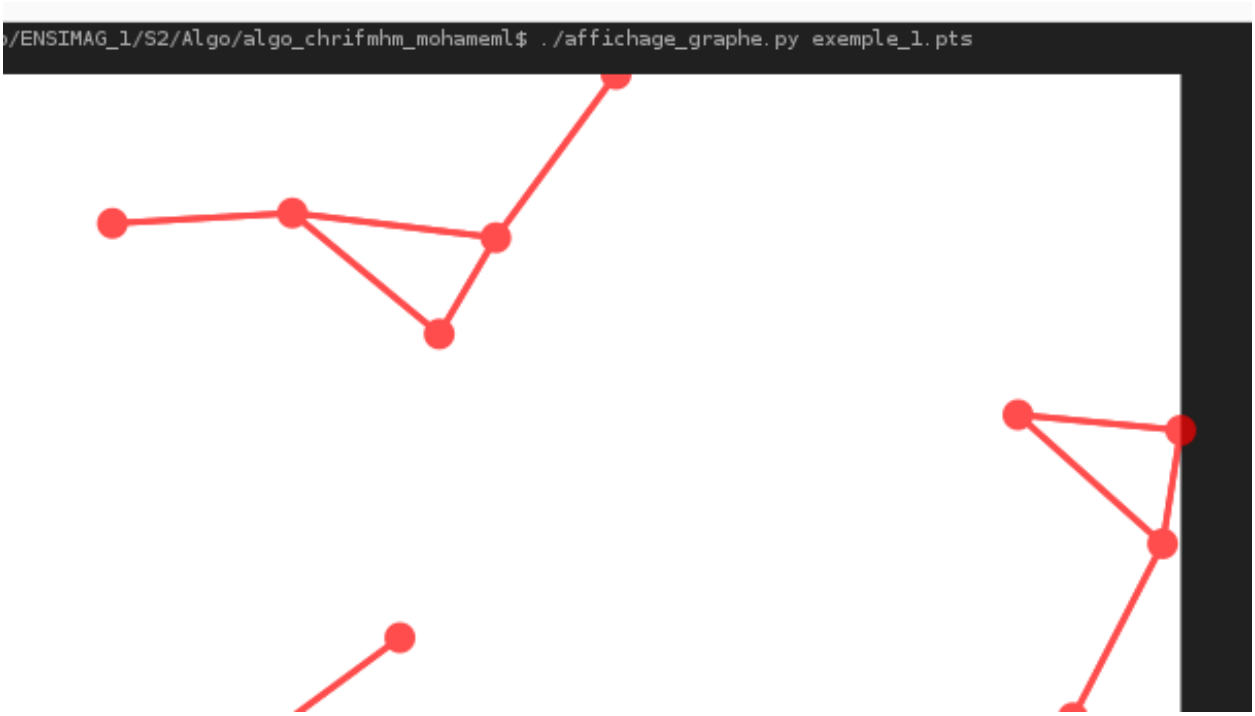

Figure 4: Demo pour l'affichage