Contenu du fichier community\_detection.py :

def create\_network(list\_of\_friends):

"""Retourne le dictionnaire correspondant au réseau en parcourant une seule fois le tableau."""

network={}

i=0

while i<len(list\_of\_friends):

if list\_of\_friends[i] not in network :

tab=[]

if i%2==0: #si l'indice est pair

tab.append(list\_of\_friends[i+1])

else:

tab.append(list\_of\_friends[i-1])

network[list\_of\_friends[i]]=tab

else:

tab2=network[list\_of\_friends[i]]

if i%2==0:

tab2.append(list\_of\_friends[i+1])

else:

tab2.append(list\_of\_friends[i-1])

network[list\_of\_friends[i]]=tab2

i+=1

return network

def get\_people(network):

"Retourner la liste des personnes de ce réseau dans un tableau"

return list(network)

def are\_friends(network, person1, person2):

"""Retourner True si les deux personnes sont amis selon le réseau sinon retourner False"""

return person1 in network[person2]

def all\_his\_friends(network, person, group):

"""Retourner True si la personne est ami avec tout les membres du groupe et False sinon"""

#Retourner True si le tableau des amis est vide

if len(group)== 0 :

return True

i=0

var=True

while i<len(group):

if not are\_friends(network,person,group[i]):

return False

i+=1

return var

def is\_a\_community(network, group):

"""Retourner True si le groupe est une communauté et False sinon"""

j=0

var=True

while j<len(group):

person=group[j]

group2=group.copy()

group2.remove(person)

if not all\_his\_friends(network,person,group2):

return False

j+=1

return var

def find\_community(network, group):

"""Retourner une communauté en fonction de l'heuristique décrite"""

communaute=[]

i=0

while i<len(group):

if all\_his\_friends(network,group[i],communaute):

communaute.append(group[i])

i+=1

return communaute

def order\_by\_decreasing\_popularity(network, group):

"""Retourner la communauté trouvée en appliquant l'heuristique de construction de communauté maximale"""

#tri\_a\_bulle

i=0

statut=False

while not statut :

i=0

while i<len(group)-1:

if len(network[group[i]]) < len(network[group[i+1]]):

statut=False

group[i+1],group[i]=group[i],group[i+1]#échanger les valeurs

i=-1 # reprendre le tableau depuis le début

i+=1

statut=True

return group

def find\_community\_by\_decreasing\_popularity(network):

"""Retourner la communauté en triant l'ensemble des personnes du réseau selon l'ordre décroissant de popularité et en appliquant l'heuristique de construction maximale"""

people=get\_people(network)

people\_trie=order\_by\_decreasing\_popularity(network,people)

return find\_community(network,people\_trie)

def find\_community\_from\_person(network, person):

"""Retourner une communauté maximale contenant un sous-ensemble des amis de la personne plus elle-même selon l'heuristique décrite"""

communaute=[]

communaute.append(person)

amis=order\_by\_decreasing\_popularity(network,network[person])

i=0

while i<len(amis):

if all\_his\_friends(network,amis[i],communaute):

communaute.append(amis[i])

i+=1

return communaute

def find\_max\_community(network):

"""Retourner la plus grande communauté trouvée"""

people=get\_people(network)

max=0

i=0

while i<len(people):

community=find\_community\_from\_person(network, people[i])

if len(community)> max :

max\_community=community

max=len(max\_community)

i+=1

return max\_community

Contenu du fichier test\_community\_detection.py :

reseau={

"Alice" : ["Bob", "Dominique"],

"Bob" : ["Alice", "Charlie", "Dominique"],

"Charlie" : ["Bob"],

"Dominique" : ["Alice", "Bob"]

}

reseau2={'Yasmine': ['Muriel', 'Joël', 'Thomas'],

'Muriel': ['Yasmine', 'Joël'],

'Joël': ['Yasmine', 'Muriel', 'Nassim', 'Ali', 'Andrea'],

'Thomas': ['Yasmine', 'Daria', 'Carole'],

'Nassim': ['Joël', 'Ali', 'Andrea'],

'Andrea': ['Ali', 'Joël', 'Nassim', 'Valentin'],

'Ali': ['Andrea', 'Joël', 'Nassim'],

'Daria': ['Thomas'],

'Carole': ['Thomas'],

'Thierry': ['Axel', 'Léo'],

'Axel': ['Thierry', 'Léo'],

'Léo': ['Thierry', 'Axel', 'Valentin'],

'Valentin': ['Léo', 'Andrea']}

amis = ["Alice", "Bob","Charlie","Bob", "Alice", "Dominique", "Bob", "Dominique"]

def test\_create\_network():

create\_network(amis)=={'Alice': ['Bob', 'Dominique'],

'Bob': ['Alice', 'Charlie', 'Dominique'],

'Charlie': ['Bob'],

'Dominique': ['Alice', 'Bob']}

print("Le test de la fonction create\_network est : OK")

test\_create\_network()

def test\_get\_people():

assert get\_people(reseau)==['Alice', 'Bob', 'Charlie', 'Dominique']

assert not get\_people(reseau2)==['Yasmine','Muriel','Joël','Thomas','Nassim','Andrea']

print("Le test de la fonction get\_people ets : OK")

test\_get\_people()

def test\_are\_friends():

assert are\_friends(reseau,"Bob","Alice")==True

assert are\_friends(reseau, "Alice","Bob")==True

assert are\_friends(reseau,"Alice","Charlie")==False

assert are\_friends(reseau2,"Yasmine","Muriel")==True

print("Le test de la fonction are\_friends est : OK")

test\_are\_friends()

def test\_all\_his\_friends():

assert all\_his\_friends(reseau,"Alice",["Bob","Dominique"])==True

assert all\_his\_friends(reseau,"Alice",["Bob","Charlie"])==False

assert all\_his\_friends(reseau2, "Yasmine",["Muriel", "Joël"])==True

print("Le test de la fonction all\_his\_friends est : OK")

test\_all\_his\_friends()

def test\_is\_a\_community():

assert is\_a\_community(reseau,["Alice", "Bob", "Dominique"])==True

assert is\_a\_community(reseau,["Alice", "Bob", "Charlie"])==False

assert is\_a\_community(reseau2,["Yasmine","Muriel","Joël"])==True

print("Le test de la fonction is\_a\_community est : OK")

test\_is\_a\_community()

def test\_find\_community():

assert find\_community(reseau,["Alice", "Bob", "Charlie", "Dominique"])==["Alice", "Bob", "Dominique"]

assert find\_community(reseau,["Charlie", "Alice", "Bob", "Dominique"])==["Charlie", "Bob"]

assert find\_community(reseau,["Charlie", "Alice", "Dominique"])==["Charlie"]

assert find\_community(reseau2,["Yasmine","Muriel","Joël"])==['Yasmine', 'Muriel', 'Joël']

reseau3=create\_network(lecture\_reseau("files/Communaute1.csv"))

assert find\_community(reseau3,['Barbra','Cloe','Louis'])

print("Le test de la fonction find\_community est : OK")

test\_find\_community()

def test\_order\_by\_decreasing\_popularity():

assert order\_by\_decreasing\_popularity(reseau,["Alice","Charlie","Bob"])==['Bob', 'Alice', 'Charlie']

assert order\_by\_decreasing\_popularity(reseau,['Dominique', 'Alice', 'Charlie'])==['Dominique', 'Alice', 'Charlie']

assert order\_by\_decreasing\_popularity(reseau,['Alice','Bob'])==['Bob','Alice']

assert order\_by\_decreasing\_popularity(reseau, ["Alice", "Charlie", "Dominique"])==["Alice","Dominique","Charlie"]

assert order\_by\_decreasing\_popularity(reseau2,["Yasmine","Muriel","Joël"])==['Joël', 'Yasmine', 'Muriel']

reseau3=create\_network(lecture\_reseau("files/Communaute1.csv"))

assert order\_by\_decreasing\_popularity(reseau3,['Barbra','Cloe','Louis'])==['Barbra', 'Cloe', 'Louis']

print("Le test de la fonction order\_by\_decreasing\_popularity est : OK")

test\_order\_by\_decreasing\_popularity()

def test\_community\_by\_decreasing\_popularity():

assert find\_community\_by\_decreasing\_popularity(reseau)==['Bob', 'Alice', 'Dominique']

assert find\_community\_by\_decreasing\_popularity(reseau2)==['Joël', 'Andrea', 'Nassim', 'Ali']

print("Le test de la fonction community\_by\_deccreasing\_popularity est : OK")

test\_community\_by\_decreasing\_popularity()

def test\_community\_from\_person():

assert find\_community\_from\_person(reseau,"Alice")==['Alice', 'Bob', 'Dominique']

assert find\_community\_from\_person(reseau, "Charlie")==['Charlie', 'Bob']

assert find\_community\_from\_person(reseau, "Dominique")==["Dominique","Bob","Alice"]

assert find\_community\_from\_person(reseau,"Bob")==["Bob","Alice","Dominique"]

assert find\_community\_from\_person(reseau2,"Yasmine")==['Yasmine', 'Joël', 'Muriel']

assert not find\_community\_from\_person(reseau2,"Joël")==['Joël', 'Andrea', 'Nassim']

print("Le test de la fonction find\_community\_from\_person est : OK")

test\_community\_from\_person()

def test\_find\_max\_community():

assert find\_max\_community(reseau)==['Alice', 'Bob', 'Dominique']

assert find\_max\_community(reseau2)==['Joël', 'Andrea', 'Nassim', 'Ali']

print("Le test de la fonction find\_max\_community est : OK")

test\_find\_max\_community()

Contenu du fichier using\_community\_detection.py :

## Question 2 : Comparaison théorique :

Pour comparer les deux fonctions `dico\_reseau` et `create\_network` il faut analyser la complexité asymptotique des deux fonctions:

La compléxité asymptotique de la fonction dico\_reseau est quadratique

=> la fonction contient une boucle while qui sera éxecuté n fois , et chaque itération appelle la fonction list\_amis qui

parcours le tableau des amis et qui est de compléxité asymptotique linéaire , du coup O(n)n => O(n^2)

la compléxité asymptotique de la fonction create\_network est linéaire O(n)

=> la fonction parcours une seule fois le tableau (contient une seule boucle

while qui sera exécuté n fois et un nombre d opération élémentaires qui est négligeable)

## Comparaison expérimentale :

from cummunity\_detection import \*

from time import time

from random import \*

def mesure\_temps(fonction,n):

i=0

tic1=time()

while i<n:

fonction(amis)

i+=1

tac1=time()

return round(((tac1-tic1)\*1000),6)

print("Le temps moyen nécessaire pour la création un dictionnaire", 100000, "fois :")

print("\t", mesure\_temps(dico\_reseau, 100000), "ms pour la fonction dico\_reseau")

print("\t", mesure\_temps(create\_network, 100000), "ms pour la fonction create\_network")

## Question 11 : Comparaison théorique :

la fonction `find\_community\_by\_decreasing\_popularity` retourne la liste des personnes de ce réseau O(n) et les trient par décroissance de popularité, de compléxité de O(n log n). Ensuite trier la communauté selon l heuristique de compléxité O(n^2). La compléxité totale de la fonction est donc de compléxité O(n log n) + O(n^2) + O(n) .

La fonction `find\_community\_from\_person` trie en premier les amis de la personne donnée de compléxité O(n log n), ensuite une boucle while qui à chaque itération appelle la fonction all\_his\_friend , ce qui a une compléxité de O(n^2). La compléxité totale de la fonction ets donc de O(n^2) +O(n log n).

## Comparaison expérimentale :

departure1=time()

i=0

while i<10000 :

find\_community\_by\_decreasing\_popularity(reseau)

i+=1

end1=time()

print("le temps nécessaire pour l'éxecusion de la fonction find\_community\_by\_decreasing\_popularity est",round((end1-departure1)\*1000/10000,6), "ms")

departure2=time()

i=0

tab=les\_plus\_pop(reseau)

while i<10000 :

find\_community\_from\_person(reseau, tab[0])

i+=1

end2=time()

print("le temps nécessaire pour l'éxecusion de la fonction find\_community\_from\_person est",round((end2-departure2)\*1000/10000,6), "ms")