Tme6_Densest_Subgraph

April 8, 2019

1 TME 6 - Densest Subgraph

1.0.1 Fonctions pour la construction de la liste d'adjacence

In [ ]:
```
import copy

        # On trie les edges dans un nouveau fichier sorted_data
        # Pour chaque edge, on met le point d'indice plus petit en debut de ligne
        def clear_dataset(file,source,separator):
            graph = open(file+source, "r")
            sorted_graph = open(file+"sorted_data", "w+")
            for e in graph:
                if e.startswith("#"):
                    continue
                e = e.rstrip("\n")
                array_e = e.split(separator)
                if(int(array_e[0]) > int(array_e[1])):
                    array_e[0],array_e[1] = array_e[1],array_e[0]
                    str_e = ' '.join(array_e)
                    sorted_graph.write(str_e + "\n")
                    continue
                str_e = ' '.join(array_e)
                sorted_graph.write(str_e + "\n")

            sorted_graph.close()
            graph.close()

            # A partir de sorted_data
            # on cree un fichier cleaned_data contenant les edges sans doublon
            sorted_graph = open(file+"sorted_data", "r")
            cleaned_graph = open(file+"cleaned_data", "w+")
            for e in sorted_graph:
                if e.startswith("#"):
                    continue
                e = e.rstrip("\n")
                indice_point = e.split(" ")
                if int(indice_point[0]) == int(indice_point[1]):
```

```
            continue
        if not(e in cleaned_graph):
            cleaned_graph.write(str(e) + "\n")
    cleaned_graph.close()
    sorted_graph.close()


# Dans le graph, trouver le sommet d'indice maximum
def indice_max(file):
    cleaned_graph = open(file+"cleaned_data", "r")
    max_int = -1
    for e in cleaned_graph:
        if e.startswith("#"):
            continue
        e = e.rstrip("\n")
        indice_point = e.split(" ")
        if int(indice_point[0])>max_int:
            max_int = int(indice_point[0])
        if int(indice_point[1])>max_int:
            max_int = int(indice_point[1])
    cleaned_graph.close()
    return max_int

# Renvoi une structure de données : liste d'adjacence
def liste_adjacence(file):
    cleaned_graph = open(file+"cleaned_data", "r")
    map_sommet_voisin_no_filtred = dict()
    for i in range(indice_max(file)+1):
        map_sommet_voisin_no_filtred[i] = list()
    for e in cleaned_graph:
        if e.startswith("#"):
            continue
        edge_str = e.rstrip("\n")
        edge = edge_str.split(" ")
        if(not(int(edge[1]) in map_sommet_voisin_no_filtred[int(edge[0])])):
            map_sommet_voisin_no_filtred[int(edge[0])].append(int(edge[1]))
        if(not(int(edge[0]) in map_sommet_voisin_no_filtred[int(edge[1])])):
            map_sommet_voisin_no_filtred[int(edge[1])].append(int(edge[0]))
    cleaned_graph.close()

    # On filtre notre map en enlevant les sommets sans voisins
    map_sommet_voisin = dict()
    for k in map_sommet_voisin_no_filtred.keys():
        if len(map_sommet_voisin_no_filtred[k]) == 0:
            continue
        map_sommet_voisin[k] = map_sommet_voisin_no_filtred[k]
    return map_sommet_voisin
```

```python
# Même fonction que indice_max mais prend directement un fichier clean
def indice_max_from_clean_file(file):
    cleaned_graph = open(file, "r")
    max_int = -1
    for e in cleaned_graph:
        if e.startswith("#"):
            continue
        e = e.rstrip("\n")
        indice_point = e.split("\t")
        if int(indice_point[0])>max_int:
            max_int = int(indice_point[0])
        if int(indice_point[1])>max_int:
            max_int = int(indice_point[1])
    cleaned_graph.close()
    return max_int

# Même fonction que liste_adjacence mais prend directement un fichier clean
def liste_adjacence_from_clean_file(file):
    cleaned_graph = open(file, "r")
    map_sommet_voisin_no_filtred = dict()
    for i in range(indice_max_from_clean_file(file)+1):
        map_sommet_voisin_no_filtred[i] = list()
    for e in cleaned_graph:
        if e.startswith("#"):
            continue
        edge_str = e.rstrip("\n")
        edge = edge_str.split("\t")
        if(not(int(edge[1]) in map_sommet_voisin_no_filtred[int(edge[0])])):
            map_sommet_voisin_no_filtred[int(edge[0])].append(int(edge[1]))
        if(not(int(edge[0]) in map_sommet_voisin_no_filtred[int(edge[1])])):
            map_sommet_voisin_no_filtred[int(edge[1])].append(int(edge[0]))
    cleaned_graph.close()
    map_sommet_voisin = dict()
    for k in map_sommet_voisin_no_filtred.keys():
        if len(map_sommet_voisin_no_filtred[k]) == 0:
            continue
        map_sommet_voisin[k] = map_sommet_voisin_no_filtred[k]
    return map_sommet_voisin

def nb_edge(file):
    nb_edge = 0
    graph = open(file, "r")
    for e in graph:
        if e.startswith("#"):
            continue
        nb_edge = nb_edge + 1
    return nb_edge
```

```
In [ ]: clear_dataset("graph/email/","email-Eu-core.txt"," ")
        email = liste_adjacence("graph/email/")

        #clear_dataset("graph/amazon/","com-amazon.ungraph.txt","\t")
        #amazon = liste_adjacence("graph/amazon/")

        #livreJournal = liste_adjacence_from_clean_file("graph/lj/com-lj.ungraph-clean.txt")

        #orkut = liste_adjacence_from_clean_file("graph/orkut/com-orkut.ungraph-clean.txt")
```

1.0.2 MinHeap

```
In [ ]: class BinHeap:
            def __init__(self):
                self.heapList = []
                self.currentSize = 0

            def percUp(self,i,map_index_tas):
                while ((i+1 // 2)  > 0):
                    if self.heapList[i][1] < self.heapList[(i - 1)// 2][1]:
                        tmp = self.heapList[(i-1) // 2]
                        self.heapList[(i-1) // 2] = self.heapList[i]
                        self.heapList[i] = tmp
                        if(map_index_tas):
                            tmp_ind = map_index_tas[self.heapList[(i-1) // 2][0]]
                            map_index_tas[self.heapList[(i - 1)// 2][0]] = (i - 1)// 2
                            map_index_tas[self.heapList[i][0]] = tmp_ind
                    i = (i-1)//2

            def insert(self,k,map_index_tas):
                self.heapList.append(k)
                self.currentSize = self.currentSize + 1
                self.percUp(self.currentSize-1,map_index_tas)

            def percDown(self,i,map_index_tas):
                while (i * 2) <= self.currentSize-1 :
                    mc = self.minChild(i)
                    if self.heapList[i][1] > self.heapList[mc][1]:
                        tmp = self.heapList[i]
                        self.heapList[i] = self.heapList[mc]
                        self.heapList[mc] = tmp
                        if(map_index_tas):
                            tmp_ind = map_index_tas[self.heapList[i][0]]
                            map_index_tas[self.heapList[i][0]] = i
                            map_index_tas[self.heapList[mc][0]] = tmp_ind
                    i = mc
                    if i == 0:
                        break
```

```
    def minChild(self,i):
        if i * 2 + 1 > self.currentSize-1:
            return i * 2
        else:
            if self.heapList[i*2][1] < self.heapList[i*2+1][1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self,map_index_tas):
        retval = self.heapList[0]
        self.heapList[0] = self.heapList[self.currentSize-1]
        self.currentSize = self.currentSize - 1
        self.heapList.pop()
        if self.currentSize > 0:
            del map_index_tas[self.heapList[0][0]]
        self.percDown(0,map_index_tas)
        return retval

    def buildHeap(self,alist,map_index_tas):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [] + alist[:]
        while (i > 0):
            self.percDown(i,map_index_tas)
            i = i - 1
```

1.1 Exercice 1 : k-core Algorithm

```
In [ ]: import heapq
        import copy

        def core_decomposition(graph):
            c = 0
            n = len(list(graph.keys())) + 1
            subgraph_order = dict()
            tas = BinHeap()
            map_index_tas = dict()
            visited = dict()

            for s in list(graph.keys()):
                tas.insert((s,len(graph[s])),map_index_tas)
                visited[s] = False

            for i, t in enumerate(tas.heapList):
                map_index_tas[t[0]] = i
```

```
    while(len(tas.heapList) > 0):
        v = tas.heapList[0]
        c = max(c,v[1])

        for v_voisin in graph[v[0]]:
            if(visited[v_voisin] == True):
                continue
            v_voisin = int(v_voisin)
            lst = list(tas.heapList[map_index_tas[v_voisin]])
            lst[1] = tas.heapList[map_index_tas[v_voisin]][1] - 1
            t = tuple(lst)
            tas.heapList[map_index_tas[v_voisin]] = t
            tas.percUp(map_index_tas[v_voisin],map_index_tas)
        tas.delMin(map_index_tas)
        visited[v[0]] = True
        subgraph_order[v[0]] = n
        n = n - 1
    print(c)
    return subgraph_order

def get_subgraph_ssj(subgraph_order):
    subgraph = sorted(subgraph_order.items(), key=lambda kv: kv[1])
    return subgraph

def get_density(graph, subgraph):
    nb_edge = 0
    # délation logique
    sommet_exist = [0] * (max(list(graph.keys())) + 1)
    density = list()
    for s in subgraph:
        sommet_exist[s[0]] = 1
        for v in graph[s[0]]:
            if sommet_exist[v] == 1:
                nb_edge = nb_edge + 1
        if(len(density) == 0):
            density.append(nb_edge)
        else:
            d = len(density)
            density.append(nb_edge/d)
    return density

def get_edge_density(graph,file):
    nbedge = nb_edge(file)
    return nbedge / (max(graph.keys()) * (max(graph.keys()) - 1))

def get_size_densest(density):
    return density.index(max(density))
```

1.1.1 1. Test

```
In [ ]: file = "graph/email/email-Eu-core.txt"
        graph = email

        subgraph_order = core_decomposition(graph)
        subgraph = get_subgraph_ssj(graph)
        density = get_density(graph,subgraph)
        edge_density = get_edge_density(graph,file)
```

1.2 Exercice 2

```
In [ ]: import matplotlib.pyplot as plt
        import copy
        import heapq

        def core_decomposition_lst(graph):
            c = 0
            n = len(list(graph.keys())) + 1
            coresDegrees = dict()
            tas = BinHeap()
            map_index_tas = dict()
            visited = dict()

            for s in list(graph.keys()):
                tas.insert((s,len(graph[s])),map_index_tas)
                visited[s] = False

            for i, t in enumerate(tas.heapList):
                map_index_tas[t[0]] = i

            while(len(tas.heapList) > 0):
                v = tas.heapList[0]
                c = max(c,v[1])

                for v_voisin in graph[v[0]]:
                    if(visited[v_voisin] == True):
                        continue
                    v_voisin = int(v_voisin)
                    lst = list(tas.heapList[map_index_tas[v_voisin]])
                    lst[1] = tas.heapList[map_index_tas[v_voisin]][1] - 1
                    t = tuple(lst)
                    tas.heapList[map_index_tas[v_voisin]] = t
                    tas.percUp(map_index_tas[v_voisin],map_index_tas)
                tas.delMin(map_index_tas)
                visited[v[0]] = True
                coresDegrees[v[0]] = (len(graph[v[0]]),c)
            return coresDegrees
```

```
clear_dataset("scholar/","net.txt"," ")
scholar = liste_adjacence("scholar/")

core_degree = dict()
core_degree = core_decomposition_lst(scholar)

degrees = list()
kcores = list()

for k,v in core_degree.items():
    degrees.insert(k, v[0])
    kcores.insert(k, v[1])

plt.scatter(degrees,kcores)
plt.title('net.txt')
plt.xlabel('Degrees')
plt.ylabel('Coreness')

plt.show()
```

In [ ]:
```
anomalies = list()
for d in range(len(degrees)):
    if(kcores[d] == 14):
        print("anomaly 1 : " + str(d))
        anomalies.append(d)
```

1.3 Exercice 3

In [ ]:
```
# Construction de la map r qui associe un noeud à son score
def init_r(file, separator):
    cleaned_graph = open(file, "r")
    max_int = -1
    for e in cleaned_graph:
        if e.startswith("#"):
            continue
        e = e.rstrip("\n")
        indice_point = e.split(separator)
        if int(indice_point[0])>max_int:
            max_int = int(indice_point[0])
        if int(indice_point[1])>max_int:
            max_int = int(indice_point[1])
    cleaned_graph.close()

    r = dict()
    for i in range(max_int+1):
        r[i] = 0
```

```
        return r

    # Compute density score
    def MKscore(file,t, separator):
        r = init_r(file, separator)
        cleaned_graph = open(file, "r")
        for time in range(t):
            for e in cleaned_graph:
                if e.startswith("#"):
                    continue
                edge_str = e.rstrip("\n")
                edge = edge_str.split(separator)
                i = int(edge[0])
                j = int(edge[1])
                if r[i] <= r[j]:
                    r[i] = r[i] + 1
                else:
                    r[j] = r[j] + 1

        for k in r:
            r[k] = r[k]/t

        cleaned_graph.close()
        return r
```

In [ ]:
```
iteration=10
separator = " "
result = MKscore("graph/email/email-Eu-core.txt",iteration,separator)
maximum = max(result, key=result.get)
```