Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Filière: Informatique

Spécialité:

Master 1 BIG DATA ANALYTIC

Thème

Problèmes de cheminement et applications

Présenté par :

CHAIBI racim

CHICHIOU manel

KHELILI chams el sabah

Table de matière

Dans ce rapport on va résoudre des Problèmes de cheminement et applications suivants avec une implémentation avec python:

1. Graphes values

2. Problème du plus court chemin

3. Algorithm de Prim

1. Graphes valués

Un graphe value est un graphe auquel on ajoute un poids sur ses arcs/arêtes, Il peut être représenté par sa matrice de poids.

1.1 construction d'un graphe valué

Construire la matrice d'adjacence M du graphe orienté G à l’aide d’algorithme suivants « implémenté avec Python » qui a comme des entrés les sommets et les poids des arcs qu’ils le construire.

```python
    return arbre
#__________________Graphe valués____________________________

def add_vertex(v):
  global graph
  global vertices_no
  global vertices
  if v in vertices:
    print("Vertex ", v, " already exists")
  else:
    vertices_no = vertices_no + 1
    vertices.append(v)
    if vertices_no > 1:
        for vertex in graph:
            vertex.append(0)
    temp = []
    for i in range(vertices_no):
        temp.append(0)
    graph.append(temp)
def add_edge(v1, v2, e):
    global graph
    global vertices_no
    global vertices

    if v1 not in vertices:
        print("Vertex ", v1, " does not exist.")

    elif v2 not in vertices:
        print("Vertex ", v2, " does not exist.")

    else:
        index1 = vertices.index(v1)
        index2 = vertices.index(v2)
        graph[index1][index2] = e

def print_graph():
  global graph
  global vertices_no
  for i in range(vertices_no):
    for j in range(vertices_no):
      if graph[i][j] != 0:
        print(vertices[i], " -> ", vertices[j], \
        " edge weight: ", graph[i][j])
```

Notre Application :

```
****************************************************************************************************
                  "Bienvenue """ Problèmes de cheminement et applications""""
****************************************************************************************************

 vous devez choisir un des choix suivants en entrant le numéro correspondant:

1 :Problème des Graphes valué

2: Problème du plus court chemin
```

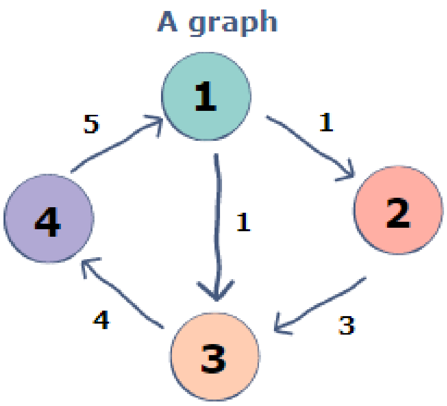
On choisit 1 :

```
*************************************************************************
                  "1-  Problème des Graphes valués"
*************************************************************************



**********************************Aide:**********************************
---------------------------------------------------------------------
| Maintenant nous voulons construire notre  matrice d'adjascence     |
| Vous devez entrer n le nombre des sommets.                         |
| Donner le cardinale d'ensemble des voisinage de chaque sommet      |
| Pour chaque sommet :                                               |
| Entrer le numero de sommet adjascent et le poids d'arc correspondant|
| Mi,j=0 si il n'y pas d'arcs/arêtes reliant i à j                   |
| Mi,j=valuation de l'arc/arête {i, j}                               |
----------------------------------------------------------------------
```

En commençant le déroulement par exemple on prend ce graphe et on doit afficher la matrice correspondante suivante :



Adjacency list	Adjacency matrix
{"1" : [[2, 1], [3,1]], "2" : [[3, 3]], "3" : [[4, 4]], "4" : [[1, 5]]}	[[0, 1, 1, 0], [0, 0, 3, 0], [0, 0 ,0 , 4], [5, 0, 0 ,0]]

On tape les entrés demandés :

```
----------------------------------------------------------
Entrer le nombre des sommets : 4
entre le nombre des voisins de ce sommet0 : 2
entre le nombre des voisins de ce sommet1 : 1
entre le nombre des voisins de ce sommet2 : 1
entre le nombre des voisins de ce sommet3 : 1
entrer le numero de sommet adjascente a ce sommet0 : :2
entrer le poid  d'arc correspondant : 1
```

On continue… et Voilà :

La résultat finale :

```
0  ->  1  edge weight:  1
0  ->  2  edge weight:  1
1  ->  2  edge weight:  3
2  ->  3  edge weight:  4
3  ->  0  edge weight:  5
Internal representation:  [[0, 1, 1, 0], [0, 0, 3, 0], [0, 0, 0, 4], [5, 0, 0, 0]]
```

L'algorithme d'affichage est le suivant :

```
n=int(input("Entrer le nombre des sommets : "))
for i in range (n):
    add_vertex(i)
    m=int(input(f"entre le nombre des voisins de ce sommet{i} : "))
    neighbour.append(m)

for i in range (n):
    for j in range (neighbour[i]):
        v=int(input(f"entrer le numero de sommet adjascente a ce sommet{i} :"))
        p=int(input("entrer le poid  d'arc correspondant : "))
        add_edge(i, v, p)


print_graph()
print("Internal representation: ", graph)

print("\n vouler vous sortir : tapez 0 ")
s=input()
if s=="0":
    break
```

Comme vous voyez à la fin il nous demande est ce que on veut quitter l'application 1, si oui on tape 0… Nous on ne veut pas et on tape 1 pour essayer l'application 2 des « problèmes du plus court chemin » :

```
******************************************************************************
                    "2-  Problème du plus court chemin"
******************************************************************************

 vous devez choisir un des choix suivants en entrant le numéro correspondant:

1 :algorithme de Bellman

2: algorithme de Dijkstra

3:Application d'ordonnancement

4: algorithme de Prim
```

2. Problème du plus court chemin

En théorie des graphes, le problème de plus court chemin est le problème algorithmique qui consiste à trouver un chemin d'un sommet à un autre de façon que la somme des poids des arcs de ce chemin soit minimale.

2.1 algorithme de Bellman

L'algorithme de Bellman-Ford résout le problème des plus courts chemins avec origine unique dans le cas le plus général où les poids des arcs peuvent avoir des valeurs négatives. Étant donné un graphe orienté pondéré G = (V, E), de fonction de poids w, et une origine s, l'algorithme retourne une valeur booléenne indiquant s'il existe un circuit de poids négatif accessible depuis. S'il n'en existe pas, l'algorithme donne les plus courts chemins ainsi que leurs poids.

L’algorithme :

```
définition de bellman(M)

Dist = liste des distances depuis le sommet 0, dont chaque élément est initialisé
à l'infini

Dist[0] = 0

tant que Dist change

        pour tout sommet i

                pour tout sommet j

                        si Dist[i] + M[i,j] < Dist[j]

                        alors Dist[j] = Dist[i] + M[i,j]

                        fin si

                fin pour

        fin pour

fin tant que

retourner Dist
```

On applique cette algorithme dans langage Python :

```python
#______ algorithme de bellman ___________
def bellman(MV):

    dict1 = {
      "valeur": [0]

    }
    dict2 = {
      "valeur": [inf]

    }
    for i in range(len(MV)-1):
        dict1["valeur"].append(inf)
        dict2["valeur"].append(inf)

    while(dict1["valeur"]!= dict2["valeur"]):
        dict2["valeur"]=copy(dict2,dict1)

        for i in range(len(MV)):
            for j in range(len(MV)):
                if (dict1["valeur"][i]+MV[i][j ]< dict1["valeur"][j]):
                    dict1["valeur"][j]=dict1["valeur"][i]+MV[i][j]

    return dict1["valeur"]
```

Donc on a créé une fonction appelée Bellman qui a comme entrée la matrice de poids

On a créé 2 dictionnaires un contient la solution du plus court chemin et l’autre et utilisant pour comparer dict1 change

Et en à utiliser fonction copy pour enregistrer les valeurs de dict1 a dict2 sans que dict2 soit un pointeur sur les même valeurs donc si dict1 change dict2 change aussi

On initialise dict1 avec 0 et on ajoute des valeurs (inf) a la taille du nombre de sommets

On met une boucle while qui vérifier si dict1 change ou pas ou il sort de la boucle si dict1 n’a pas changer veut dire que l’algorithme et terminer de marque et à trouver la solution.

En plus il y a doublé boucle a l’intérieur qui affect une valeur a un sommet Y Si la valeur de sommet adjacent X + poids de l’arcs(X,Y) et inferieur a Y.

Apres que l’algorithme termine il retourne une liste des distances depuis le sommet 0.

2.2 algorithme de Dijkstra

L'algorithme de Dijkstra résout le problème des plus courts chemins avec origine unique dans le cas le plus général où les poids des arcs peuvent avoir seulement des valeurs positives pour éviter les cycles absorbants.

Algorithme :

définition de dijkstra(M)

Dist = liste des distances depuis le sommet 0, dont chaque élément est initialisée à l'infini

Dist[0] = 0

    tant que il y a des sommets non marqués

    choisir un sommet i non marqué, tel que Dist[i] soit minimale

    marquer i

        pour tout sommet j non marqué

            si Dist[i] + M[i,j] < Dist[j]

alors Dist[j] = Dist[i] + M[i,j]

fin si

fin pour

tant que

retourner Dist

On applique cette algorithme dans langage Python :

```
#_______ algorithme de djikstra ___________
def djikstra(MV):

    dict1 = {
      "valeur": [0],
      "marquer": [1]
    }
    a=0
    for i in range(len(MV)-1):
        dict1["valeur"].append(MV[0][i+1])
        dict1["marquer"].append(0)
    while test(dict1):
        val= inf
        for i in range(len(MV)):
            if (dict1["marquer"][i]==0 and dict1["valeur"][i]<=val):
                val=dict1["valeur"][i]
                a=i


        dict1["marquer"][a]=1
        for j in range(len(MV)):
            if (dict1["marquer"][j]==0):
                if(dict1["valeur"][a]+MV[a][j]<dict1["valeur"][j]):
                    dict1["valeur"][j]=dict1["valeur"][a]+MV[a][j]

    return dict1["valeur"]
```

Donc on a créé une fonction appelée Djikstra qui a comme entrée la matrice de poids

On a créé un dictionnaire qui contient la solution du plus court chemin « valeur » et une liste de marquage « marquer »

Et en a utiliser fonction « test » pour vérifier si tous les sommets sont marquer ou non

On initialise dict1[valeur] avec 0 et dict1[marquer] par 1 pour dire que le premier sommet est marqué et initialise par la valeur 0 on ajoute des valeurs (inf) a la taille du nombre des sommets.

On met une boucle while qui vérifier si les sommets sont tous marquer sinon il continuer le marquage.

On met une boucle pour choisir un sommet i non marqué, tel que Dist[i] soit minimale et on le marque.

Apres on voit tous les sommets Y non marque et voisin a le sommet marquer X on affect une valeur a le sommet Y Si la valeur de sommet adjacent X + poids de l'arcs(X,Y) et inferieur a Y.

L'algorithme boucle jusqu'à tous les sommets sont marquer, l'algorithme termine il retourne une liste des distances minimum depuis le sommet 0.

2.3 Application : ordonnancement

Pour l'application d'ordonnancement on chercher de trouver le plus long chemin donc on a utilisé le principe de l'algorithme Bellman avec quelques modifications pour avoir un algorithme adapter au plus long chemin

Algorithme

définition de bellman_long(M)

Dist = liste des distances depuis le sommet 0, dont chaque élément est initialisé à – (l'infini)

Dist[0] = 0

tant que Dist change

pour tout sommet i

pour tout sommet j

si Dist[i] + M[i,j] > Dist[j]

alors Dist[j] = Dist[i] + M[i,j]

fin si

fin pour

fin pour

fin tant que

retourner Dist

On applique cette algorithme dans langage Python :

```
#_______ algorithme de bellman_long  ___________
def bellman_long(MV):

    dict1 = {
      "valeur": [0.0]

    }
    dict2 = {
      "valeur": [-inf]

    }
    for i in range(len(MV)-1):
        dict1["valeur"].append(-inf)
        dict2["valeur"].append(-inf)

    while(dict1["valeur"]!= dict2["valeur"]):
        dict2["valeur"]=copy(dict2,dict1)

        for i in range(len(MV)):
            for j in range(len(MV)):
                if (dict1["valeur"][i]+MV[i][j ]> dict1["valeur"][j]):
                    dict1["valeur"][j]=dict1["valeur"][i]+MV[i][j]

    return dict1["valeur"]
```

Donc on a créé une fonction appelée Bellman_long qui a comme entrée la matrice de poids mais les valeurs inf seront négatives donc (-inf)

On a créé 2 dictionnaires un contient la solution du plus long chemin et l'autre et utilisant pour comparer dict1 change

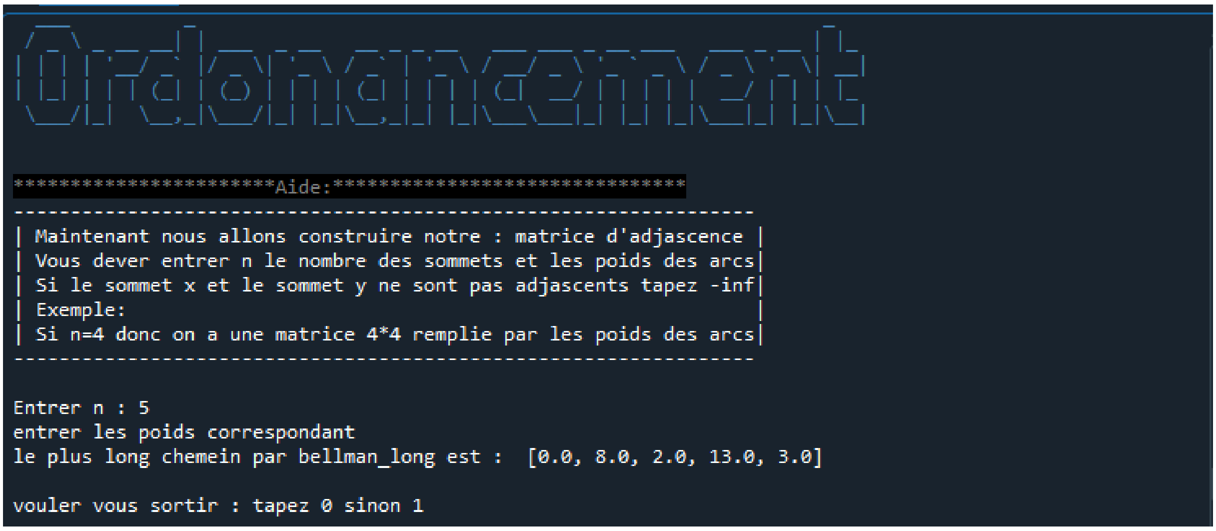
Et en à utiliser fonction copy pour enregistrer les valeurs de dict1 a dict2 sans que dict2 soit un pointeur sur les mêmes valeurs donc si dict1 change dict2 change aussi

On initialise dict1 avec 0 et on ajoute des valeurs (-inf) a la taille du nombre de sommets

On met une boucle while qui vérifier si dict1 change ou pas ou il sort de la boucle si dict1 n'a pas changer veut dire que l'algorithme et terminer de marque et à trouver la solution.

En plus il y a doublé boucle a l'intérieur qui affect une valeur a un sommet Y Si la valeur de sommet adjacent X + poids de l'arcs(X,Y) et supérieur a Y.

Après que l'algorithme termine il retourne une liste des distances depuis le sommet 0.



3. algorithme de Prim

L'algorithme de Prim est un algorithme glouton qui calcule un arbre couvrant minimal dans un graphe connexe valué et non orienté.il trouve un sous-ensemble d'arêtes formant un arbre sur l'ensemble des sommets du graphe initial et tel que la somme des poids de ces arêtes soit minimale.

Algorithme

définition de prim(M)

marquer le sommet 0

arbre = arbre résultat, ne contenant initialement pas d'arêtes.

tant qu' il reste des sommets non marqués

{x, y} = arête de poids minimal, joignant un sommet marqué x à un sommet non Marqué y

marquer y

ajouter l'arête {x,y} à l'arbre

fin tant que

retourner arbre

On applique cette algorithme dans langage Python :

```python
#________ Algorithme De Prim ______________
def Prim(MV):
    dict1 = {
      "marquer": [1]
    }
    arbre ={
        "arborecence":[],
        "valeur":[]
            }
    a=0
    val= inf
    for i in range(len(MV)-1):
        dict1["marquer"].append(0)
    while test(dict1):
        for i in range(len(MV)):
            val= inf
            a=inf
            b=inf
            for j in range(len(MV)):

                if (dict1["marquer"][i] == 0  and dict1["marquer"][j]== 1):
                    if(MV[i][j]<val):
                        val = MV[i][j]
                        a=j
                        b=i
            if(val != inf):
                arbre["arborecence"].append([a,b])
                arbre["valeur"].append(val)
                dict1["marquer"][b]=1

    return arbre
```

Donc on a créé une fonction appelée Prim qui a comme entrée la matrice de poids.

On a créé 2 dictionnaires un contient la solution de l'arbre-couvrant minimal en plus les valeurs des arcs et l'autre contient la liste des sommets marquer

Et on a utiliser fonction « test » pour vérifier si tous les sommets sont marqués ou non

On met une boucle while qui vérifier si dict1 change ou pas ou il sort de la boucle si dict1 n'a pas changer veut dire que l'algorithme et terminer de marque et à trouver la solution.

Avec les double boucles on cherche de trouver un sommet non marquer Y adjacent a sommet marquer X et que le poids de l'arc entre eux est minimal, si oui en affect arc (X,Y) a le dictionnaire arbre[« arboressence »] et ça valeur a arbre[« valeur »] et on marque le sommet Y.

Après que l'algorithme termine il retourne un dictionnaire qui contient l'arbre-couvrant minimal et les valeurs de ces arcs.

```
 -------------------------------------------------------------
| Maintenant nous allons construire notre : matrice d'adjascence |
| Vous dever entrer n le nombre des sommets et les poids des arcs|
| Si le sommet x et le sommet y ne sont pas adjascents tapez  inf|
| Exemple:                                                       |
| Si n=4 donc on a une matrice 4*4 remplie par les poids des arcs|
 -------------------------------------------------------------

Entrer n : 5
entrer les poids correspondant
L'arbre couvrant de poids minimal est :  {'arborecence': [[0, 1], [0, 3], [3, 4], [3, 2]], 'valeur':
[2.0, 3.0, 1.0, 2.0]}

vouler vous sortir : tapez 0
```

Guide pour utiliser notre application :

Pour utiliser la deuxième partie des algorithme de cheminement vous taper « 2 » sur la console

```
*************************************************************************************
                 "Bienvenue """ Problèmes de cheminement et applications""""
*************************************************************************************

 vous devez choisir un des choix suivants en entrant le numéro correspondant:

1 :Problème des Graphes valué

2: Problème du plus court chemin
```

Après il vous donne le choix de l'algorithme que vous voulez utiliser.

```
*****************************************************************
                 "2-  Problème du plus court chemin"
*****************************************************************

 vous devez choisir un des choix suivants en entrant le numéro correspondant:

1 :algorithme de Bellman

2: algorithme de Dijkstra

3:Application d'ordonnancement

4: algorithme de Prim
```

Après choisir l'algorithme il vous demande les dimension de la matrice après il commence l'insertion des valeurs en commençant par la première ligne et ajoute des colonne.