INTERNATIONALE DATASCIENCE INSTITUE

MASTER DATASCIENCE-BIG DATA

PLUS COURT CHEMIN D'UN NŒUD A TOUS LES AUTRES NŒUDS AVEC ALGORITHME DE DIJKSTRA

Etudiants :

ZEGBOLOU GBIZIE MARC-DEXTER

BOUA MARIE-ESTER EMMANUELLA

GAKPA KOMLAN CLAUDY

Professeur :

PROF DARIO COLLAZO

SOMMAIRE

1) Objectif du projet

Le projet a pour objectif de trouver le plus court chemin d'un nœud A à tous les autres d'un Graphe. Pour y arriver le projet nous recommande l'utilisation de l'algorithme de Dijkstra.
Pour vérifier la performance de nos algorithmes nous les testerons sur un graphe de nœud plus grand.

2) Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme qui permet de résoudre les problèmes de plus court chemin, cependant il d'autre type d'algorithme qui permettent de résoudre les problèmes de plus courts chemin telles que Bellman, Ford ,Kruskal pour les arbres couvrant de poids minimal.
Tous au long de notre travaille nous allons nous focaliser sur le graphe suivant :

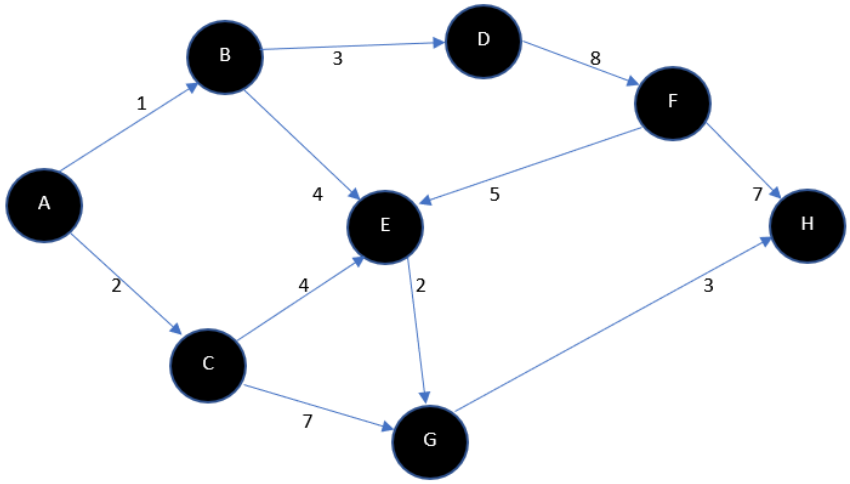

Figure 1 : Graphe exemple

Un graphe est représenté par l'ensemble de ses nœuds S et l'ensemble de ses arêtes.
Le graphe est entièrement déterminé par la connaissance de ces nœuds, des successeurs de ces nœuds et des distances qui séparent un nœud et son successeur.
Ainsi nous avons :

- A B 1
- A C 2
- C G 7
- C E 4
- B D 3
- D F 8
- F E 5

- E G 2
- G H 3
- F H 7
- B E 4
- H H 0

Application de l'algorithme de Dijkstra au graphe précédent :

A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
.	1A	2A	∞	∞	∞	∞	∞
.	.	2A	4B	5B	∞	∞	∞
.	.	.	4B	6C/5B	∞	9C	∞
.	.	.	.	5B	12D	9C/7E	∞
.	.	.	.	.	12D	7E	∞
.	.	.	.	.	12D	.	10G
.	.	.	.	.	12D	.	19F/10G.
.	.	.	.	.	.	.	10G .

Tableau : déroulement de l'algorithme de Dijkstra

Le plus court chemin de A le nœud source à tous les autres :

- A B 1
- A C 2
- A B D 4
- A B E 5
- A B D F 12
- A B E G 7
- A B E G H 10

Ainsi le plus court chemin de A à D est ''A B D'' avec une distance de 4

3) Implémentation

(a) Description des données

Les données que nous utiliserons seront de la forme
Nœud ->Successeur ->Distance

Exemple :

- B D 3
- D F 8
- F E 5

(b) Mapper

Notre mapper consistera à ordonner les données que nous allons remplir dans un dictionnaire. Cette solution n'est cependant pas optimale nous nous en sommes rendu compte très tard mais elle est facile à implémenter. L'accès se faisant par clé elle est aussi très rapide.
Les données émises par le mapper sont du type (K, V) au K est un nœud et V un couple (B, d) où B est un successeur de K et d la distance KV.

Pseudo algorithme :

ProcedureMapper(nœud ,successeur ,distance(nœud ,successeur))

emit(nœud ,successeur ,distance(nœud ,successeur))

(c) Reducer

Dans le Reducer nous allons remplir notre graph, calculer les plus courtes distances et les afficher.

Pseudo algorithme :

G=(S' ,U ,l) où G est un graphe d'ensemble de sommet S ,d'ensemble d'arête U, l les distances associées à chaque aux arêtes.

ProcedureReducer(G)

Début

on pose $S' = \{x_2, \ldots, x_n\}$,

$\prod(x_1) = 0$

$\mathbf{\prod(x_i) = l_{1i}}$ si $x_i \in S(x_1)$ et $+\infty$ sinon, $l_{ij} = l(x_i, x_j)$

Tanque S'≠ Ensemble Vide Faire

$x_i \in \mathbf{S'}$ tel que $\mathbf{\prod(x_i) = Min \{ \prod(x_k) \}}$ avec $x_k \in S'$,

Faire $\mathbf{S' = S' - \{x_i\}}$, si $\mathbf{|\ S'| = 0}$ aller à **5**.

Faire pour tout $\mathbf{x_j \in\ S(x_i) \cap S'}$,

$\prod(x_j) := Min \{ \prod(x_j), \prod(x_i) + l_{ij} \}$.

4) Description du code

(a) Hadoop

Nous avons en premier temps écrit un mapper qui nous permettait d'obtenir les données triées et dans un second temps nous avons utilisé le Reducer pour remplir un dictionnaire sur lequel nous avons appliqué l'algorithme.

(b) Spark

Le programme est dans le même ordre d'idée que le tableau décrit au début il d'abord au départ le chargement des données. Ensuite une boucle « while » est utilisé pour trouver tous les chemins et les distances minimales.

5) Résultat

(a) Hadoop

Les résultats obtenus avec MapReduce sont de la forme :

- A B 1
- A C 2
- A B D 4
- A B E 5
- A B D F 12
- A B E G 7
- A B E G H 10

Le résultat s'interprète comme suite, le nœud source A, la séquence du plus court chemin au nœud but et la distance minimale entre les deux nœuds.

(b) Spark

```
>>> dijkstra
[(1.0, (u'A', u'B')), (12.0, (u'D', u'F')), (2.0, (u'A', u'C')), (4.0, (u'B', u'
D')), (10.0, (u'G', u'H')), (7.0, (u'E', u'G')), (5.0, (u'B', u'E'))]
>>>
```

Le résultat obtenu se lit comme suit :
Le plus court chemin de A à H est A B E G H avec une distance de 10 .

- A B 1
- A C 2
- A B D 4
- A B E 5
- A B D F 12
- A B E G 7
- A B E G H 10

6) Scalabilité

(a) Données

Nous avons téléchargé des données sur le open data de l'Université de Stanford https://snap.stanford.edu/data/egonets-Facebook.html .
Ces données de 4039 nœuds et de 88234 arêtes qui étaient sous cette forme :

- A 0 B,1 :C,2 :
- D 999 F,5 : G,2 :

Elles ont été traitées pour être fourni sous la forme :

- A B 1
- A C 2
- D F 5
- D G 2

Le code chargé de transformer le fichier est le suivant :

```
doc=open('C:\\Users\\pc-asus\\Desktop\\bigdata-master\\data\\fb.dat')
doc1=open('C:\\Users\\pc-asus\\Desktop\\bigdata-master\\data\\output.txt','w')
for line in doc:
    line=line.strip('\n').split(':')
    if len(line)>1:
        ligne=line[0].split(' ')
        sommet=ligne[0]
        #print(ligne)
        deb=ligne[2].split(',')
         #deb=line[0].split(' ')[0].split(',')
        doc1.write(str(sommet)+' '+str(deb[0])+' '+str(deb[1])+'\n')
        for item in line[2:]:
            if len(item.split(','))>1:
                doc1.write(str(sommet)+' '+str(item.split(',')[0])+' '+str(item.split(',')[1])+'\n')
                 #print(str(item.split(",")[0])+' '+str(item.split(",")[1])+'\n')
    else:
        ligne=line[0].split(' ')
        doc1.write(str(ligne[0])+' '+str(ligne[0])+' '+str('0')+'\n')
doc1.close()
```

(b) Hadoop

Après avoir tester le de Facebook sur notre machine de de 7 cœurs nous avons obtenues un temps d'exécutions de 203 s. Ce qui est acceptable en locale.

(c) Spark

Le code n'étant pas optimale nous nous sommes réservés de tester sa scalabilité.

7) Références

[1] Cours de Théorie des Graphes du Professeur ADAMA COULIBALY UNIVERSITE FELIX HOUPHOUET BOIGNY d'Abidjan Cocody
[2] https://spark.apache.org/docs/latest/api/python/pyspark.html
[3] https://github.com/bilal-elchami/hadoop_streaming_job_chaining
[4] Cours du Professeur DARIO COLAZZO

8) Annexes

(a) Mapper

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sys
#doc=open('/home/dexter/text.txt')
for line in sys.stdin:
    ligne = line.strip()
    ligne=ligne.split(' ')
    print(ligne[0]+' '+ligne[1]+' '+str(ligne[2]))
```

(b) Reducer

Première partie remplissage du graphe :

```
import time
import math
import sys
start_time = time.time()
noeudencour = None
#noeud_source=sys.argv[1]
graphvois = {}
graph={}
L=[]
noeudelu=None
itineraire=""
way=' '
#doc=open('/home/dexter/text.txt')
for line in sys.stdin:
        line = line.strip('\n').split(' ')
        noeud = line[0]
        successeur = line[1]
        distance = int(line[2])
        if noeudencour != None:
           if noeudencour != noeud:
                 if len(graphvois.keys())!= 0:
                   #print(graphvois)
                   graph[noeudencour]=graphvois
                 graphvois={}
                 graphvois[successeur]=distance
                 noeudencour = noeud
                 distance=0
           else:
                graphvois[successeur]=distance

        else:
              noeudencour = noeud
              graphvois[successeur]=distance
graph[noeudencour]=graphvois
#print(graph)
```

Deuxième partie calcule des distances minimales et choix des nœuds

```
#print(graph)
sommet=list(graph.keys())
noeud_source=sommet.pop(0)
def chemin(noeud1):
    global way
    for item in L:
        if item[0]==noeud1:
            way=item[1]+' '+way
            if item[1]!=noeud_source:
                chemin(item[1])
            else :
                way=noeud_source+way
    return way


for noeud in sommet:
    if noeud in graph[noeud_source].keys():
        L.append([noeud,noeud_source,graph[noeud_source][noeud]])
    else:
        L.append([noeud,None,math.inf])

while len(sommet)!=0:
    distmin=math.inf
    for i in range(len(L)):
        if L[i][2]<=distmin and (L[i][0] in sommet):
            distmin=L[i][2]
            noeudelu=L[i][0]
            indice=i
    itineraire=str(chemin(noeudelu))
    print(itineraire[1:]+noeudelu+' '+str(L[indice][2]))
    way=' '
    #print(L)
    del(sommet[sommet.index(noeudelu)])
    if len(graph[noeudelu].keys())!=0 :
       for noeud in graph[noeudelu].keys():
           for i in range(len(L)):
               if L[i][0]==noeud and noeud in sommet:
                  if L[i][2]>=graph[noeudelu][noeud]+distmin:
                     L[i][2]=graph[noeudelu][noeud]+distmin
                     L[i][1]=noeudelu
```

(c) Spark

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import time
start_time = time.time()
from pyspark import SparkContext
sc=SparkContext()

rdd1=sc.textFile('text.txt')
graph=rdd1.map(lambda x:x.split()).map(lambda x:(x[0],[float(x[2]),x[1]]))
sommet=graph.keys().distinct().collect()
noeudsource=sommet.pop(0)

def f(x):
     if x[0]==noeudsource:
       return (float(x[1][0]),(noeudsource,x[1][1]))
     else:
       return (float("inf"),(None,x[1][1]))


dijkstra=graph.map(f).distinct().collect()
graph=graph.groupByKey().map(lambda x:(x[0],list(x[1])))

while len(sommet)!=0:
      L1=[]
      L2=[]
      noeudist=sc.parallelize(dijkstra).filter(lambda x:x[1][1] in sommet).min()
      distmin=noeudist[0]
      noeudelu=noeudist[1][1]
      del(sommet[sommet.index(noeudelu)])
      success=graph.filter(lambda x : x[0]==noeudelu).flatMap(lambda x:x[1]).flatMap(lambda x: x).collect()
      L1=[x for x in success if (success.index(x)%2==0)]
      L2=[x for x in success if (success.index(x)%2==1)]
      if len(L2)!=0 :
         for noeud in L2:
             for i in range(len(dijkstra)):
                 if dijkstra[i][1][1]==noeud and noeud in sommet:
                    if dijkstra[i][0]>L1[L2.index(noeud)]+distmin:
                       dijkstra[i]=(L1[L2.index(noeud)]+distmin,(noeudelu,dijkstra[i][1][1]))
```