ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

Tính toán song song

Bài tập lớn - Giai đoạn 2

Viết chương trình và đánh giá thuật toán PageRank

GVHD: Thoại Nam

SV: Phạm Quốc Trung - 1814522
Vũ Duy Bình - 1927006
Đặng Ngọc Tâm - 1813910

TP. HỒ CHÍ MINH, THÁNG 11/2020

Mục lục

1 Thuật toán PageRank

1.1 Giới thiệu

PageRank là một thuật toán đánh giá và xếp hạng các trang web được phát triển bởi Larry Page (do đó thuật toán được đặt tên là PageRank) và sau đó cùng với Sergey Brin tiếp tục nghiên cứu và phát triển, bắt đầu từ năm 1995 và là tiền đề cho công cụ tìm kiếm Google sau này.

PageRank phân tích các siêu liên kết (hyper link) để đánh giá tầm quan trọng, độ nổi tiếng và độ uy tín của một trang web, thông qua việc tính toán số lượng và chất lượng các liên kết trỏ đến trang đó nhằm cung cấp cho người dùng các kết quả phù hợp nhất với mục đích tìm kiếm của họ.

Thuật toán PageRank không chỉ sử dụng cho các công cụ tìm kiếm mà còn được dùng rộng rãi trong các bài toán Graph Mining như sinh học, hóa học, các bài toán khoa học xã hội, vật lý, ...

1.2 Mô tả thuật toán

Giả sử chúng ta có tập n trang web được đánh số từ $1...n$, PageRank của trang web i được tính dựa trên các liên kết trang web khác đến nó (trang web j trỏ liên kết đến trang web i). Thuật toán PageRank được xây dựng dựa trên hai ý tưởng cơ bản sau:

- Trang web A trỏ liên kết đến trang web B, nếu A là một trang web có xếp hạng cao trên bảng thì phải giúp B có xếp hạng cao hơn.
- Trang web A trỏ liên kết đến trang web B, lượng trang web mà A trỏ đến nghịch biến với xếp hạng của B, hay nói cách khác A trỏ đến càng nhiều trang thì thứ hạng của B tăng càng ít.

Bản chất của Pagerank là phân bố xác suất, được sử dụng để thể hiện khả năng khi người dùng click chuột ngẫu nhiên vào đường link và sẽ tới được trang web cụ thể. Khi bắt đầu tính toán thì sự phân bổ đó được chia đều cho tất cả những trang web có trong tập dữ liệu. Các tính toán Pagerank cần một số lần "lặp đi lặp lại" qua các liên kết trong tập để có thể đạt đến độ chính xác mong muốn.

1.2.1 Mô hình ban đầu

Đặt $L_{i,j} = 1$ nếu như trang web j có trỏ liên kết đến trang web i (ký hiệu $j \to i$), ngược lại $L_{i,j} = 0$. Lúc này ta có $m_i = \sum_{k=1}^{n} L_{k,j}$ là tổng số liên kết mà web j trỏ đến trang web khác. Với hai ý tưởng xây dựng thuật toán được nêu trên, ta xây dựng công thức PageRank ban đầu:

$$p_i = \sum_{j \to i} \frac{p_j}{m_j} = \sum_{j=1}^{n} \frac{L_{i,j}}{m_j} p_j \tag{1}$$

PageRank p_i của trang web i sẽ bằng tổng của tất cả các điểm PageRank thành phần của các trang web j có link trỏ đến nó chia cho số linkout (link trỏ đến trang khác) của chính trang web j đó.

Gọi $\mathbf{p}, \mathbf{L}, \mathbf{M}$ lần lượt là các ma trận $n \times 1, n \times n, n \times n$ lưu giữ thông tin của $p_i, L_{i,j}, m_i$ như sau:

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_3 \end{bmatrix}; \mathbf{L} = \begin{bmatrix} L_{1,1} & L_{1,2} & \cdots & L_{1,n} \\ L_{2,1} & L_{2,2} & \cdots & L_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} \end{bmatrix}; \mathbf{M} = \begin{bmatrix} m_1 & 0 & \cdots & 0 \\ 0 & m_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & m_n \end{bmatrix}$$

Viết lại dưới dạng ma trận: $\mathbf{p} = \mathbf{L}\mathbf{M}^{-1}\mathbf{p}$, nếu như đặt $\mathbf{A} = \mathbf{L}\mathbf{M}^{-1}$ ta có $\mathbf{A} = \mathbf{L}\mathbf{M}^{-1}$. Dễ nhận thấy $\mathbf{p}$ chính là vector riêng của ma trận $\mathbf{A}$ với trị riêng là 1. Tuy nhiên lúc này lại nảy sinh 2 vấn đề:

- Đồ thị có thể không liên thông: khi đó ma trận $\mathbf{A}$ có nhiều hơn một vector riêng;
- Có thể tồn tại các điểm chết (dead ends): là những những đỉnh không trỏ liên kết về bất kì trang web nào dù có liên kết trang web khác trỏ tới.

Do đó ta cần phải thêm yếu tố Damping vào mô hình.

1.2.2 Yếu tố Damping

Lý thuyết PageRank cho rằng, ngay cả một người dùng giả thiết click ngẫu nhiên vào các trang web cuối cùng cũng sẽ dừng lại. Xác suất người dùng tiếp tục click bất kỳ link nào có trong một trang web gọi là yếu tố Damping. Công thức tính Pagerank có tính đến yếu tố Damping sử dụng mô hình khi người dùng ngẫu nhiên cảm thấy chán sau khi click và được chuyển đến một số trang ngẫu nhiên.

Tại mỗi trang web i có m_i liên kết khác nhau, ta xem xác suất mà người dùng chuyển sang trang web khác là đồng nhất hay nói cách khác xác suất chuyển sang trang web khác là $1/m_i$. Yếu tố Damping (d) thêm vào lúc này:

- Nếu có liên kết từ i đến j: $p_i = \dfrac{1-d}{n} + \dfrac{d}{m_j}$;
- Ngược lại: $p_i = \dfrac{1-d}{n}$;

với $0 < d < 1$. Có nhiều nghiên cứu về việc lựa chọn giá trị d để tính toán, Google sử dụng $d = 0.85$. Nếu viết lại dưới dạng ma trận chúng ta có thể phát biểu thuật toán PageRank như sau:

Tìm vector $\mathbf{p}$ thỏa mãn:

$$\mathbf{p} = \left(\frac{1-d}{n}E + d\mathbf{L}\mathbf{M}^{-1}\right)\mathbf{p} \tag{2}$$

với E là ma trận $n \times n$ trang web có tất cả phần tử bằng 1.

1.3 Phương pháp tính toán

Việc cài đặt thuật toán PageRank để tìm vector $\mathbf{p}$ dựa trên các phương pháp truyền thống, dùng các thư viện đại số tuyến tính, thông thường sẽ mất khoảng N^3 phép toán. Hơn nữa số lượng trang web N rất lớn, quá trình tính toán mất nhiều thời gian cũng như chi phí. Do đó ta cần một phương pháp tính toán hiệu quả hơn. Như đã trình bày ở trên, nhóm quyết định sử dụng phương pháp lặp (Power iteration). Đây là phương pháp thường được áp dụng cho các ma trận lớn khi mà ta chỉ đi tìm duy nhất một vector riêng. Các bước thực hiện như sau:

- Khởi tạo một vector $\mathbf{p}^{(0)}$ bất kỳ;

- Tính ma trận ngẫu nhiên Google: $\mathbf{G} = \frac{1-d}{n}E + d\mathbf{L}\mathbf{M}^{-1}$;
- Thực hiện lặp đến khi hội tụ: $\mathbf{p}^{(t+1)} = \mathbf{G}\mathbf{p}^{(t)}$.

1.4 Thuật toán PageRank cải tiến: Personalize PageRank

Trong thực tế, việc người dùng click ngẫu nhiên vào 1 đường link không mang tính ngẫu nhiên hoàn toàn: họ có xu hướng click vào 1 số đường link theo các chủ đề quen thuộc (dựa trên thói quen và mục đích truy cập lên các trang). Thuật toán Personalize PageRank dựa trên đặc tính này để xây dựng 1 tập các đỉnh **S** (là tập chứa các trang web khởi đầu) cho trước: bất cứ khi nào người dùng mở 1 trang ngẫu nhiên, thì trang này luôn thuộc về tập **S**. Nói cách khác, việc mở 1 trang web mới ở đây "đã có định hướng", việc định hướng này tùy thuộc vào mỗi người dùng khác nhau (cá nhân hóa - personaliztion). Qua đó ta rút ra nhận xét sau:

- Việc truy cập các trang web sẽ luôn bắt đầu từ tập **S** các đỉnh cho trước.
- Người dùng mở 1 trang ngẫu nhiên, trang này chắc chắn xuất phát từ tập **S**.

Dựa vào Personalized PageRank, nhiều ứng dụng thực tế đã được phát triển như: gợi ý kết bạn trên mạng xã hội, hệ thống đề xuất sản phẩm cho khách hàng trên cách trang thương mại điện tử, phát hiện các giao dịch đáng ngờ trong lĩnh vực tài chính, v.v.

2 Nguồn dữ liệu

Ở gian đoạn 2, nhóm xây dựng hàm sinh ma trận ngẫu nhiên để phục vụ việc tính toán:

```
import numpy as np

x = np.random.randint(2, size=(100, 100))

np.savetxt('values.csv', x, fmt="%d", delimiter=",")
```

Ma trận sinh là ma trận vuông A kích thước $n = 100$, với mỗi phần tử $A_{i,j}$ nhận 1 trong 2 giá trị: 0 (không có liên kết) hoặc 1 (có liên kết) được lưu vào file `values.csv`.

3 Dự kiến kết quả

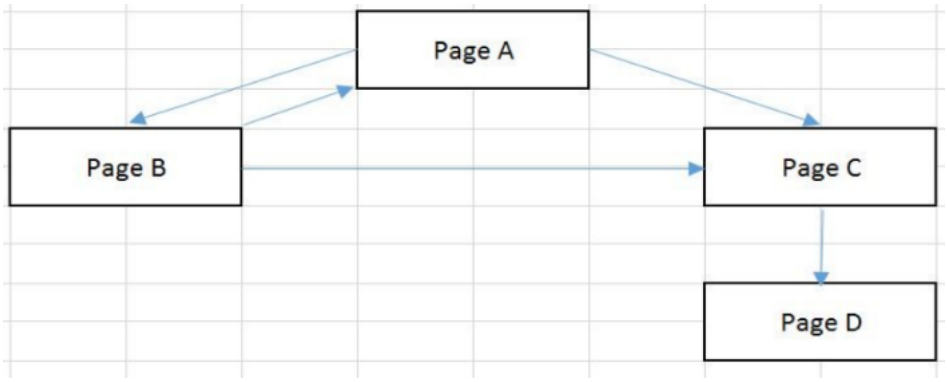
3.1 Xây dựng mô hình

3.1.1 Mô hình ban đầu

Mũi tên thể hiện trang nguồn có liên kết trỏ đến trang đích.

3.1.2 Lập ma trận tính toán

- Ma trận cạnh: thể hiện liên kết nội bộ giữa các page. Nếu page A có liên kết trỏ đến page B thì giá trị tại ô đó bằng 1, nếu không có thì giá trị đó bằng 0. Cột $L(p_i)$ dùng để chỉ tổng số liên kết ra ngoài của mỗi trang.



Hình 1: Mô hình liên kết giữa các trang web

Edge matrix		Page A	Page B	Page C	Page D	L(p_i)
(represent a graph by	Page A	0	1	1	0	2
a matrix)	Page B	1	0	1	0	2
	Page C	0	0	0	1	1
	Page D	0	0	0	0	0
Stochastic matrix		**Page A**	**Page B**	**Page C**	**Page D**	**Testing**
(ma trận chuẩn hóa)	Page A	0	0.5	0.5	0	1
	Page B	0.5	0	0.5	0	1
	Page C	0	0	0	1	1
	Page D	0.25	0.25	0.25	0.25	1
Damping factor (d)	0.85		**N**	4		
Google matrix		**Page A**	**Page B**	**Page C**	**Page D**	**Testing**
(ma trận Google)	Page A	0.0375	0.4625	0.4625	0.0375	1
	Page B	0.4625	0.0375	0.4625	0.0375	1
	Page C	0.0375	0.0375	0.0375	0.8875	1
	Page D	0.25	0.25	0.25	0.25	1

Hình 2: Các ma trận phục vụ mục đích tính toán

- Ma trận Stochastic: Cũng giống với ma trận cạnh nhưng giá trị ở mỗi ô sẽ được tính bằng cách lấy giá trị ô đó ở ma trận cạnh chia cho giá trị ở ô $L(p_i)$ cùng hàng. Riêng đối với trường hợp giá trị ở ô $L(p_i)$ bằng 0 thì giá trị ở mỗi ô trong hàng bằng 1 chia cho số cạnh của ma trận.

- Ma trận Google: Trên thực tế, mỗi trang web đều có các liên kết dẫn đến trang khác. Tuy nhiên xác suất để người dùng chọn liên kết đến khi dừng là không giống nhau. Google quyết định hệ số d (Damping) $= 0.85$ là xác suất mà một người dùng quyết định đi tiếp ở thời điểm bất kỳ.
  Xây dựng Google matrix dựa vào 2 ma trận đã có: ma trận liền kề và ma trận Markov.
  Chúng ta có thể biểu diễn Google matrix dưới dang đơn giản hơn qua công thức:
  $G_{ij} = (1 - damping)S_{ij} + damping\frac{1}{N}$
  S: ma trận Markov
  Kết quả của Google matrix phụ thuộc vào Damping (là một hệ số tự chọn), nên có thể nói, Google matrix là một ma trận ngẫu nhiên.

Nhóm quyết định sử dụng Damping = 0.85 (theo Page và Brin công bố và được Google sử dụng).

3.1.3 Tính giá trị PageRank

Giá trị PageRank khởi tạo cho mỗi page ban đầu là 1. Thực hiện qua 10 bước lặp tính toán thì giá trị PageRank dần dần hội tụ đến một giá trị giới hạn.

Iterator	Page A	Page B	Page C	Page D	Testing
0	1	1	1	1	4
1	0.7875	0.7875	1.2125	1.2125	4
2	0.742344	0.742344	1.077031	1.4382813	4
3	0.771131	0.771131	1.086627	1.3711113	4
4	0.769092	0.769092	1.096822	1.3649941	4
5	0.766925	0.766925	1.093789	1.3723603	4
6	0.76757	0.76757	1.093513	1.3713474	4
7	0.767628	0.767628	1.093846	1.3708974	4
8	0.767558	0.767558	1.0938	1.3710845	4
9	0.767568	0.767568	1.09378	1.3710854	4
10	0.767572	0.767572	1.093788	1.3710683	4
11	0.76757	0.76757	1.093788	1.3710718	4
12	0.76757	0.76757	1.093787	1.3710726	4
13	0.76757	0.76757	1.093787	1.3710721	4
14	0.76757	0.76757	1.093787	1.3710722	4
15	0.76757	0.76757	1.093787	1.3710722	4
16	0.76757	0.76757	1.093787	1.3710722	4
17	0.76757	0.76757	1.093787	1.3710722	4
18	0.76757	0.76757	1.093787	1.3710722	4
19	0.76757	0.76757	1.093787	1.3710722	4
20	0.76757	0.76757	1.093787	1.3710722	4
Iterator	Page A	Page B	Page C	Page D	Testing

Hình 3: Minh họa các bước lặp tính PageRank

Và kết quả cuối cùng thu được như Hình 4: page D có PageRank cao nhất.

3.2 Hiện thực

Nhóm sử dụng giao thức MPI để hiện thực việc tính toán song song (**Parallel**), đồng thời so sánh với phương pháp tính toán tuần tự truyền thống (**Sequence**) bằng ngôn ngữ Python.

3.2.1 Phương pháp tuần tự

Mã nguồn chi tiết:

```
1 import timeit
2 import numpy as np
3
4 DAMPING_FACTOR = 0
```

Hình 4: Kết quả thu được cuối cùng

```
5
6   print("Enter size: ")
7   SIZE = int(input())
8   EPSILON = 1e-10
9
10  start = timeit.default_timer()
11  filename = "values.csv"
12  data = np.loadtxt(filename, delimiter=",")
13  matrix_nor = np.array([sum(elem) for elem in data])
14
15  distribution = np.full(SIZE, 1/SIZE)
16  stepCounter = 0
17
18  while True:
19      stepCounter = stepCounter + 1
20      temp_distribution = np.array([0.0 for i in range(0, SIZE)])
21      for j in range (0, SIZE):
22          temp_distribution[j] = DAMPING_FACTOR / SIZE + (1 -
            ↪  DAMPING_FACTOR)*sum(distribution*(data[j] / matrix_nor))
23      isClose = list(np.isclose(distribution, temp_distribution, atol = EPSILON))
24      if False not in isClose:
25          break
26      distribution = temp_distribution
27
28  stop = timeit.default_timer()
29
30  print("Time: ", stop - start)
31  print("Nums of step: ", stepCounter)
32  print(distribution)
```

Với kích thước ma trận $n = 100$ và sai số $\epsilon = 10^{-6}$. Kết quả tính toán thu được:

```
[0.01118001 0.01180113 0.01055889 0.00869557 0.00848855 0.01076595
 0.01118003 0.0099378  0.00869561 0.00828148 0.01035185 0.01076595
 0.00745335 0.01118002 0.01242222 0.00973076 0.01076595 0.01055893
 0.01014485 0.01055892 0.00910967 0.0082815  0.0093167  0.00910965
 0.01035189 0.00890261 0.00910967 0.01138707 0.01035187 0.01076595
 0.01035188 0.01014483 0.00910965 0.01180116 0.00910966 0.01035187
 0.01014486 0.01076593 0.01014483 0.01014486 0.01014483 0.01138706
 0.01138705 0.00973077 0.01055892 0.01014485 0.00973077 0.01055892
 0.01035188 0.01159409 0.00724631 0.01118001 0.00910966 0.0089026
 0.00952372 0.00890261 0.01055891 0.01055893 0.01035188 0.01159409
 0.01014484 0.00807442 0.00869555 0.00993778 0.00931669 0.01118004
 0.01035189 0.00931668 0.00993782 0.00952373 0.01014484 0.00973076
 0.0099378  0.00910965 0.01014481 0.01035187 0.01200818 0.01097299
 0.01076594 0.00931669 0.01200815 0.00973076 0.00869558 0.00973075
 0.00890261 0.01159409 0.01076596 0.00931669 0.01055892 0.01138706
 0.00931668 0.00952372 0.00931669 0.01014485 0.01138705 0.00931668
 0.01014485 0.00952371 0.0122152  0.0122152 ]
```

Hình 5: Kết quả tính PageRank theo phương pháp tuần tự

3.2.2 Phương pháp song song sử dụng MPI

Mã nguồn chi tiết:

```python
import os.path
from mpi4py import MPI
import numpy as np
import timeit
from functools import reduce

# start time of program :
start = timeit.default_timer()

#define const
DAMPING_FACTOR = 0.85 # d= 0.85
LENGTH = None

# MPI.Init()
Comm = MPI.COMM_WORLD
size = Comm.Get_size()
rank = Comm.Get_rank()

# processor 0 : is the main processor
if rank == 0:

    # readfile
    filename = input("File : ")
    data = np.loadtxt(filename, delimiter=",")

    LENGTH = len(data)
    # create buffer is Distribution (row) of any node
```

```
    # example : [[0 1], [1,1]] has [1,2]
    sendDistribution = np.array([sum(ele) for ele in data])

    #create a buffer has (size) element from processor 0 to send all processor
    #element of buffer is lines of file (row of Matrix)
    chunks = [[] for _ in range(size)]
    counts = 0
    for index in range(size - 1):
        for count in range(int(LENGTH/size)):
            chunks[index].append(data[counts])
            counts = counts + 1

    while counts < len(data):
        chunks[size - 1].append(data[counts])
        counts = counts + 1
    #end of create

else:
    data = None
    sendDistribution = None
    sendRowBuffer = None
    chunks = None

Comm.Barrier()

# all process has received the data (the rows of Matrix ) from process 0
recvRowBuffer = Comm.scatter(chunks, root = 0)
# distribution : all process has
recv_Distribution = Comm.bcast(sendDistribution, root = 0)
LENGTH = Comm.bcast(LENGTH, root = 0)
# create a start resOld is all [1/length] or one of node is 1 and any is 0
resOld = np.full(len(recvRowBuffer),1/LENGTH)

# use the loop :
# define number of loop
for index in range(0,19):

    # update data in processor 0 to use calculator of result
    updateData = Comm.gather(resOld,root = 0)

    if rank == 0:
        # updateData in line 65 has : list[numpy.ndarray+] => numpu.ndarray[element+]
        if type(updateData[0]) is np.ndarray:
            updateData = np.array(reduce(lambda a,b: a + b.tolist(), updateData, []))
    if rank != 0:
        updateData = None
    # send all data from process 0 to all process to calculator result
    resOld = Comm.bcast(updateData, root = 0)
    resNew = []
    # PAGE_RANK :
    for ele in recvRowBuffer:
```

```
79          resx = DAMPING_FACTOR / LENGTH + (1- DAMPING_FACTOR) *sum( ele/recv_Distribution *
            ↪  resOld)
80          resNew.append(resx)
81      # UPDATE THE resOld
82      resOld = np.array([ele for ele in resNew])
83      Comm.Barrier()
84
85  #end time
86  stop = timeit.default_timer()
87
88  result = Comm.gather(resOld, root = 0)
89  if rank == 0:
90      result = np.array(reduce(lambda a,b: a + b.tolist(), result, []))
91      np.savetxt('results.csv', result, fmt="%f", delimiter=",")
92      print("Time running is : ",(stop - start))
93
```

Kết quả tính toán PageRank được xuất ra file `results.csv`:

0.010181	0.01025	0.010087	0.009835	0.009781	0.010123	0.010173	0.009999	0.009821	0.009736
0.010059	0.010109	0.009594	0.010196	0.010383	0.009944	0.010124	0.010125	0.010023	0.010102
0.009881	0.009744	0.00991	0.009826	0.01003	0.009836	0.009875	0.010177	0.010037	0.010113
0.010044	0.010016	0.009858	0.010273	0.009865	0.010066	0.010005	0.010123	0.01002	0.010035
0.010051	0.010185	0.010227	0.009956	0.010092	0.010023	0.009937	0.010067	0.010042	0.010254
0.009595	0.010171	0.009873	0.009859	0.009906	0.009838	0.010074	0.010119	0.010069	0.010212
0.010072	0.009713	0.009817	0.010003	0.009874	0.010176	0.010021	0.009898	0.009973	0.009917
0.010048	0.009971	0.009975	0.009851	0.01003	0.010074	0.01029	0.01013	0.010114	0.009899
0.010297	0.009973	0.009781	0.009961	0.009847	0.010266	0.010092	0.009904	0.010069	0.010183
0.009891	0.009922	0.009878	0.010021	0.010241	0.009892	0.010019	0.009918	0.010344	0.010327

Hình 6: Kết quả tính PageRank theo phương pháp song song

Tài liệu

[1] Truy cập ngày 14/10/2020. *Mining of Massive Datasets - Chapter 5: Link Analysis.* URL: http://infolab.stanford.edu/~ullman/mmds/ch5.pdf

[2] Peter Lofgren. *Efficient Algorithms for Personalized PageRank. PhD Thesis.* URL: https://cs.stanford.edu/people/plofgren/bidirectional_ppr_thesis.pdf

[3] Wenqing Lin. *Distributed Algorithms for Fully Personalized PageRank on Large Graphs.* URL: https://arxiv.org/pdf/1903.11749.pdf

[4] Truy cập ngày 15/10/2020. *PageRank của Google.* URL: https://viblo.asia/p/pagerank-cua-google-RQqKLvVOl7z

[5] Truy cập ngày 15/10/2020. *PageRank và mối liên hệ xích Markov.* URL: https://thetalog.com/machine-learning/page-rank/

[6] Truy cập ngày 16/10/2020. *PageRank.* URL: https://vi.wikipedia.org/wiki/PageRank

[7] Truy cập ngày 19/10/2020. *Dataset information of Google web graph.* URL: https://snap.stanford.edu/data/web-Google.html