

# Internet Measurement and Data Analysis (12)

Kenjiro Cho

2013-12-25

# review of previous class

## Class 11 Data Mining (12/18)

- ▶ Pattern extraction
- ▶ Classification
- ▶ Clustering
- ▶ exercise: clustering

# today's topics

## Class 12 Search and Ranking

- ▶ Search systems
- ▶ PageRank
- ▶ exercise: PageRank algorithm

# history of search engines

most Internet users use search engines everyday

- ▶ 1994 Yahoo! portal started
  - ▶ a pioneer of portal sites (directory-based)
  - ▶ initially, they published their favorite sites for others
- ▶ 1995 Altavista
  - ▶ a pioneering search engine with crawling robot, and multi-language support
  - ▶ issues with quality degradation by SPAM
- ▶ 1998 Google was established
  - ▶ automated search engine by the PageRank algorithm
  - ▶ web pages are scored based on the popularity of the pages

# search engine mechanisms

- ▶ directory based
  - ▶ manual registration and classification
  - ▶ high quality, but it does not scale
- ▶ robot based
  - ▶ automatically crawl web sites and create database
  - ▶ becomes the mainstream as the number of web pages increases

# robot-based search engine

- ▶ collect web pages
  - ▶ crawling
- ▶ manage database of collected information
  - ▶ index generation
- ▶ match web pages with a search query
  - ▶ search ranking

## index generation

- ▶ extract keywords from web pages
- ▶ create inverted index from keywords to web pages

# search ranking

when a search server receives a search query, it

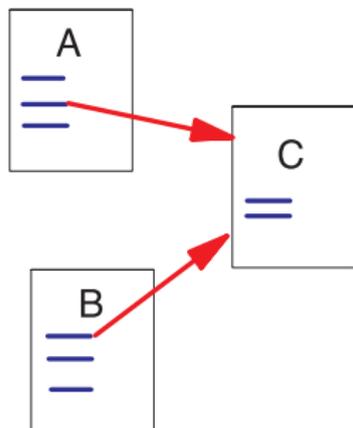
- ▶ obtains a list of related web pages by looking up the inverted index with the keywords
- ▶ orders the list by ranking, and send it back to the user

web page ranking

- ▶ requires a metric to show the importance of a web page
- ▶ PageRank: the ranking method proposed by Google

# PageRank: basic idea

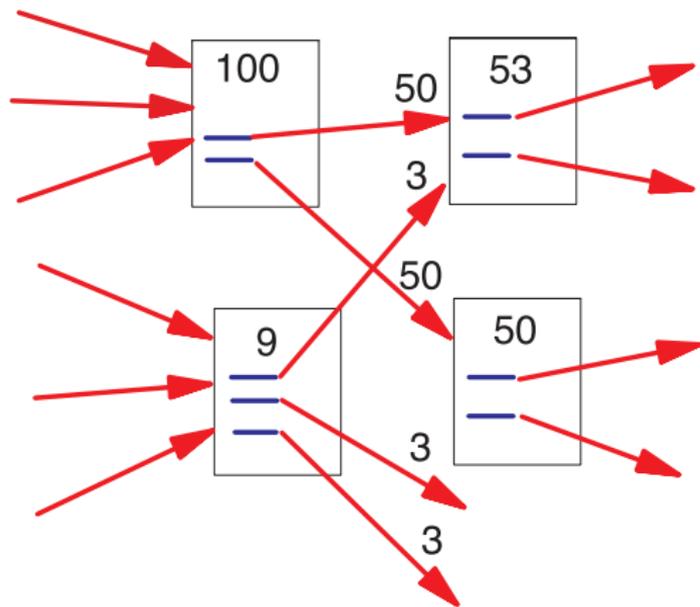
- ▶ score web pages only from the link relationship of web pages
  - ▶ it does not look at content at all



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

## PageRank: insights

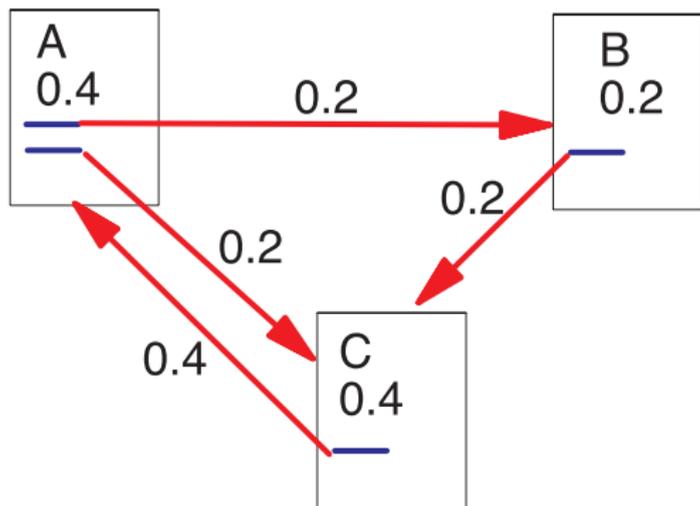
- ▶ high quality web pages are linked from many web pages
- ▶ a link from higher quality web page is more valuable
- ▶ as the number of links within a web page increases, the value of each link decreases



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

## PageRank: model

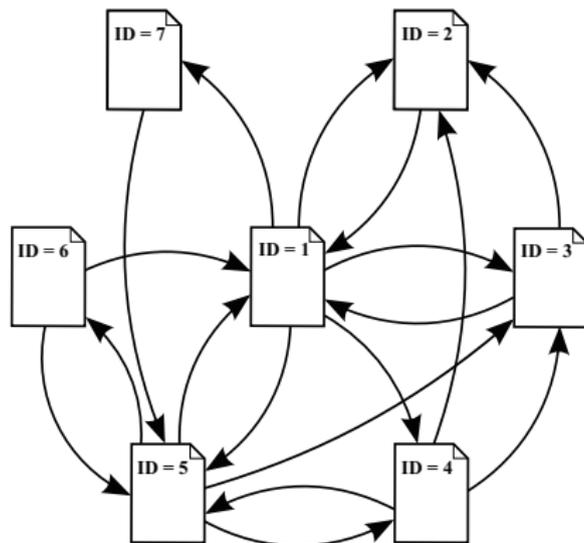
- ▶ web pages linked from high quality web pages are high quality
- ▶ random surfer model
  - ▶ a user clicks links within the same web page with the same probability



source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

# PageRank example

Page ID	OutLinks
1	2, 3, 4, 5, 7
2	1
3	1, 2
4	2, 3, 5
5	1, 3, 4, 6
6	1, 5
7	5



# matrix model

Matrix Notation (src  $\rightarrow$  dst)

$$A^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Transition Matrix (dst  $\leftarrow$  src): the sum of column is 1

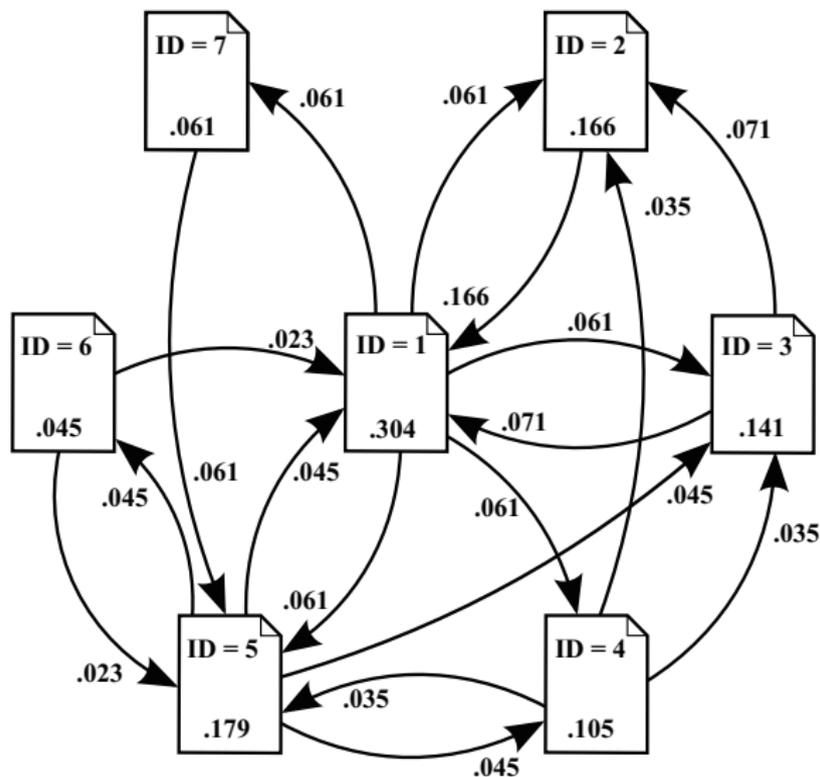
$$A = \begin{bmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R = cAR$$

pagerank vector  $R$  is an eigen vector of Transition Matrix  $A$ ,  $c$  is a reciprocal of the eigen value

## PageRank example: result

can be obtained by eigen value computation



## issues with simple PageRank model

- ▶ in reality
  - ▶ there exist nodes without outgoing links (dangling node)
  - ▶ there exist nodes without incoming links
  - ▶ there exist loops
- ▶ transition probability model is Markov chain's transition matrix
  - ▶ eventually converges to the equilibrium state
- ▶ convergence condition: the matrix is recurrent and irreducible
  - ▶ directed graph is strongly connected (there is a directed path from each node to every other nodes)
  - ▶ there exists one principal eigen vector

solution: add behavior to jump to random pages with a certain probability

# PageRank algorithm

start from an arbitrary initial state, and repeat transitions until the ranks of all pages converge

- ▶ case: node with outlinks ( $> 0$ )
  - ▶ randomly select a link within the page with probability  $d$
  - ▶ jump to a random page with probability  $(1 - d)$
- ▶ case: dangling node (no outlink)
  - ▶ jump to a random page

$$A' = dA + (1 - d)[1/N]$$

$d$ : damping factor (= 0.85)

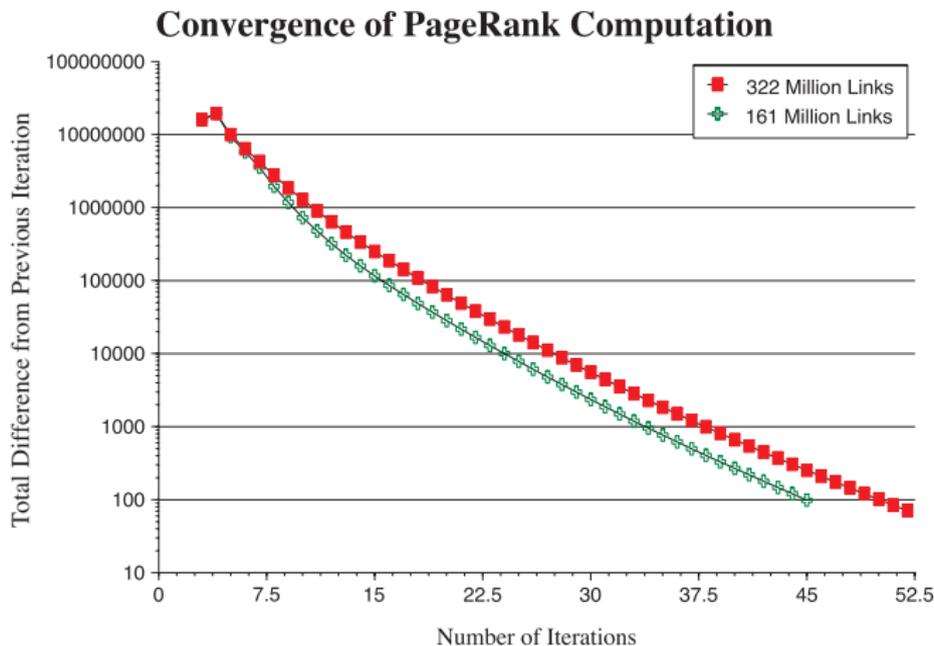
# computation by power iteration method

- ▶ eigenvalue computation is not practical for a large matrix
- ▶ but can be approximated by power iteration method

```
parameters:
    d: dampig_factor = 0.85
    thresh: convergence_threshold = 0.000001
initialize:
    for i
        r[i] = 1/N
loop:
    e = 0
    for i
        new_r[i] = d * (sum_inlink(r[j]/degree[j]) + sum_dangling(r[j])/N)
                + (1 - d)/N
        e += |new_r[i] - r[i]|
    r = new_r
while e > thresh
```

# PageRank convergence

- ▶ evaluation results show logarithmic convergence even for a large number of web pages



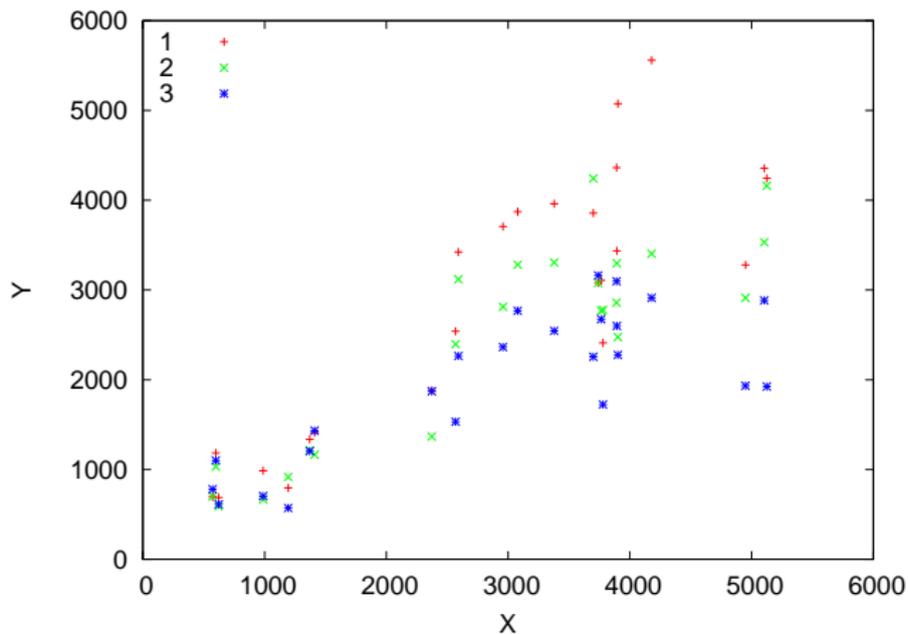
source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

# PageRank summary

- ▶ simple idea
  - ▶ web pages linked from high quality web pages are high quality
- ▶ formalize the idea by the transition matrix of Markov chain, and make it converge
- ▶ build a scalable implementation, and prove the effectiveness by real data
- ▶ start business, and become a top company
  
- ▶ note: this algorithm was introduced in 1998. the current algorithm used by Google must have evolved significantly since then.

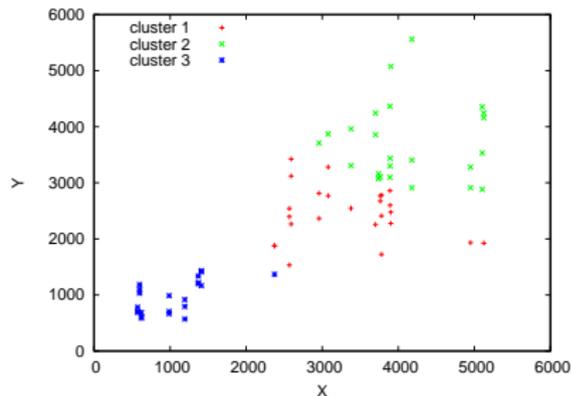
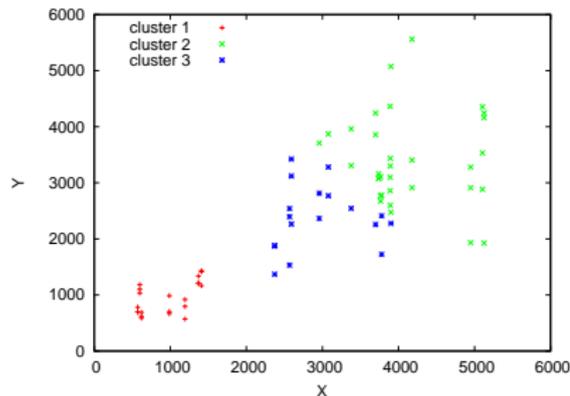
## previous exercise: k-means clustering

```
% ruby k-means.rb km-data.txt > km-results.txt
```



# k-means clustering results

- ▶ different results by different initial values



## k-means code (1/2)

```
k = 3 # k clusters
re = /^(\d+)\s+(\d+)/
INFINITY = 0x7fffffff

# read data
nodes = Array.new # array of array for data points: [x, y, cluster_index]
centroids = Array.new # array of array for centroids: [x, y]
ARGF.each_line do |line|
  if re.match(line)
    c = rand(k) # randomly assign initial cluster
    nodes.push [$1.to_i, $2.to_i, c]
  end
end

round = 0
begin
  updated = false

  # assignment step: assign each node to the closest centroid
  if round != 0 # skip assignment for the 1st round
    nodes.each do |node|
      dist2 = INFINITY # square of distance to the closest centroid
      cluster = 0 # closest cluster index
      for i in (0 .. k - 1)
        d2 = (node[0] - centroids[i][0])**2 + (node[1] - centroids[i][1])**2
        if d2 < dist2
          dist2 = d2
          cluster = i
        end
      end
      node[2] = cluster
    end
  end
end
```

## k-means code (2/2)

```
# update step: compute new centroids
sums = Array.new(k)
clsize = Array.new(k)
for i in (0 .. k - 1)
  sums[i] = [0, 0]
  clsize[i] = 0
end
nodes.each do |node|
  i = node[2]
  sums[i][0] += node[0]
  sums[i][1] += node[1]
  clsize[i] += 1
end

for i in (0 .. k - 1)
  newcenter = [Float(sums[i][0]) / clsize[i], Float(sums[i][1]) / clsize[i]]
  if round == 0 || newcenter[0] != centroids[i][0] || newcenter[1] != centroids[i][1]
    centroids[i] = newcenter
    updated = true
  end
end

round += 1

end while updated == true

# print the results
nodes.each do |node|
  puts "#{node[0]}\t#{node[1]}\t#{node[2]}"
end
```

# gnuplot script

```
set key left
set xrange [0:6000]
set yrange [0:6000]
set xlabel "X"
set ylabel "Y"

plot "km-results.txt" using 1:($3==0?$2:1/0) title "cluster 1" with points, \
"km-results.txt" using 1:($3==1?$2:1/0) title "cluster 2" with points, \
"km-results.txt" using 1:($3==2?$2:1/0) title "cluster 3" with points
```

## today's exercise: PageRank

```
% cat sample-links.txt
# PageID: OutLinks
1:      2      3      4      5      7
2:      1
3:      1      2
4:      2      3      5
5:      1      3      4      6
6:      1      5
7:      5

% ruby pagerank.rb -f 1.0 sample-links.txt
reading input...
initializing... 7 pages dampingfactor:1.00 thresh:0.000001
iteration:1 diff_sum:0.661905 rank_sum: 1.000000
iteration:2 diff_sum:0.383333 rank_sum: 1.000000
...
iteration:20 diff_sum:0.000002 rank_sum: 1.000000
iteration:21 diff_sum:0.000001 rank_sum: 1.000000
[1] 1 0.303514
[2] 5 0.178914
[3] 2 0.166134
[4] 3 0.140575
[5] 4 0.105431
[6] 7 0.060703
[7] 6 0.044728
```

# PageRank code (1/4)

```
require 'optparse'

d = 0.85 # damping factor (recommended value: 0.85)
thresh = 0.000001 # convergence threshold

OptionParser.new {|opt|
  opt.on('-f VAL', Float) {|v| d = v}
  opt.on('-t VAL', Float) {|v| thresh = v}
  opt.parse!(ARGV)
}

outdegree = Hash.new # outdegree[id]: outdegree of each page
inlinks = Hash.new # inlinks[id][src0, src1, ...]: inlinks of each page
rank = Hash.new # rank[id]: pagerank of each page
last_rank = Hash.new # last_rank[id]: pagerank at the last stage
dangling_nodes = Array.new # dangling pages: pages without outgoing link

# read a page-link file: each line is "src_id dst_id_1 dst_id_2 ..."
ARGF.each_line do |line|
  pages = line.split(/\D+/) # extract list of numbers
  next if line[0] == ?# || pages.empty?

  src = pages.shift.to_i # the first column is the src
  outdegree[src] = pages.length
  if outdegree[src] == 0
    dangling_nodes.push src
  end
  pages.each do |pg|
    dst = pg.to_i
    inlinks[dst] ||= []
    inlinks[dst].push src
  end
end
end
```

## PageRank code (2/4)

```
# initialize
# sanity check: if dst node isn't defined as src, create one as a dangling node
inlinks.each_key do |j|
  if !outdegree.has_key?(j)
    # create the corresponding src as a dangling node
    outdegree[j] = 0
    dangling_nodes.push j
  end
end

n = outdegree.length # total number of nodes
# initialize the pagerank of each page with 1/n
outdegree.each_key do |i| # loop through all pages
  rank[i] = 1.0 / n
end
$stderr.printf " %d pages dampingfactor:%.2f thresh:%f\n", n, d, thresh
```

## PageRank code (3/4)

```
# compute pagerank by power method
k = 0 # iteration number
begin
  rank_sum = 0.0 # sum of pagerank of all pages: should be 1.0
  diff_sum = 0.0 # sum of differences from the last round
  last_rank = rank.clone # copy the entire hash of pagerank

  # compute dangling ranks
  danglingranks = 0.0
  dangling_nodes.each do |i| # loop through dangling pages
    danglingranks += last_rank[i]
  end

  # compute page rank
  outdegree.each_key do |i| # loop through all pages
    inranks = 0.0
    # for all incoming links for i, compute
    # inranks = sum (rank[j]/outdegree[j])
    if inlinks[i] != nil
      inlinks[i].each do |j|
        inranks += last_rank[j] / outdegree[j]
      end
    end
  end

  rank[i] = d * (inranks + danglingranks / n) + (1.0 - d) / n
  rank_sum += rank[i]

  diff = last_rank[i] - rank[i]
  diff_sum += diff.abs
end

k += 1
$stderr.printf "iteration:%d diff_sum:%f rank_sum: %f\n", k, diff_sum, rank_sum
end while diff_sum > thresh
```

## PageRank code (4/4)

```
# print pagerank in the decreasing order of the rank
# format: [position] id pagerank
i = 0
rank.sort_by{|k, v| -v}.each do |k, v|
  i += 1
  printf "[%d] %d %f\n", i, k, v
end
```

## on the final report

- ▶ select A or B
  - ▶ A. PageRank computation of Wikipedia
  - ▶ B. free topic
- ▶ up to 8 pages in the PDF format
- ▶ submission via SFC-SFS by 2014-01-28 (Tue) 23:59

# final report topics

## A. PageRank computation of Wikipedia

- ▶ data: link data within Wikipedia English version (5.7M pages)
- ▶ A-1 investigate the distribution of pages
  - ▶ A-1-1 plot CDF and CCDF of the outdegree of pages
  - ▶ A-1-2 discussion on the outdegree distribution of Wikipedia pages
- ▶ A-2 PageRank computation
  - ▶ A-2-1 compute PageRank, and show the top 30 of the results
  - ▶ A-2-2 other analysis (optional)
  - ▶ A-2-3 discussion on the results

## B. free topic

- ▶ select a topic by yourself
- ▶ the topic is not necessarily on networking
- ▶ but the report should include some form of data analysis and discussion about data and results

## A. PageRank computation of Wikipedia

data: link data of Wikipedia English version (5.7M pages)

- ▶ created by Henry Haselgrove in 2009 (<http://haselgrove.id.au/wikipedia.htm>)
  - ▶ a local copy is available from the class web page
  - ▶ a test data set (a subset of 100K pages)
- ▶ links-simple-sorted.zip: link data (324MB compressed, 1GB uncompressed)
  - ▶ each page has a unique integer ID
  - ▶ format:  $from : to_1, to_2, \dots, to_n$
- ▶ titles-sorted.zip: title data (29MB compressed, 106MB uncompressed)
  - ▶  $n$ -th line: the title of page ID  $n$  (1 origin)

```
% head -3 links-simple-sorted.txt
1: 1664968
2: 3 747213 1664968 1691047 4095634 5535664
3: 9 77935 79583 84707 564578 594898 681805 681886 835470 ...
%
% sed -n '2713439p' titles-sorted.txt
Keio-Gijuku_University
```

## A-1 investigate the distribution of pages

A-1 investigate the distribution of pages

- ▶ A-1-1 plot CDF and CCDF of the outdegree of pages
  - ▶ include pages with outdegree 0
- ▶ A-1-2 discussion on the outdegree distribution of Wikipedia pages
  - ▶ optional other analysis
  - ▶ hint: you may compare low-degree pages and high-degree pages

## A-2 PageRank computation

### A-2 PageRank computation

- ▶ A-2-1 compute PageRank, and show top 30 of the results
  - ▶ format: rank PageRank\_value page\_ID page\_title
  - ▶ you may use the script for the exercise
    - ▶ use damping factor:0.85 thresh:0.000001
  - ▶ takes 5 hours with iMac with 8GB memory (requiring at least 4GB memory)
- ▶ A-2-2 other analysis (optional)
  - ▶ examples:
  - ▶ how to reduce the processing time
  - ▶ implement an improved version of the PageRank algorithm
- ▶ A-2-3 discussion on the results

# summary

## Class 12 Search and Ranking

- ▶ Search systems
- ▶ PageRank
- ▶ exercise: PageRank algorithm

## next class

### Class 13 Scalable measurement and analysis (1/8)

- ▶ Distributed parallel processing
- ▶ Cloud computing technology
- ▶ MapReduce
- ▶ exercise: MapReduce algorithm