Internet Measurement and Data Analysis (12)

Kenjiro Cho

2015-01-19

review of previous class

Class 11 Data Mining (1/14)

- Pattern extraction
- Classification
- Clustering
- exercise: clustering

today's topics

Class 12 Search and Ranking

- Search systems
- PageRank
- exercise: PageRank algorithm

computational complexity

metrics for the efficiency of an algorithm

- time complexity
- space complexity

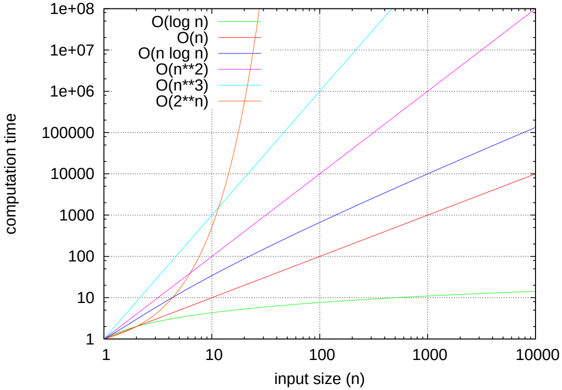
- average-case complexity
- worst-case complexity

big O notation

- describe algorithms simply by the growth order of execution time as input size n increases
  - example: $O(n), O(n^2), O(n \log n)$
- more precisely, "$f(n)$ is order $g(n)$" means:
  for function $f(n)$ and function $g(n)$, $f(n) = O(g(n)) \Leftrightarrow$ there exist constants C and n_0 such that
  $|f(n)| \leq C|g(n)| \; (\forall n \geq n_0)$

computational complexity

- logarithmic time
- polynomial time
- exponential time



example of computational complexity

search algorithms

- linear search: $O(n)$
- binary search: $O(\log_2 n)$

sort algorithms

- selection sort: $O(n^2)$
- quick sort: $O(n \log_2 n)$ on average, $O(n^2)$ for worst case

in general,

- linear algorithms (e.g., loop): $O(n)$
- binary trees: $O(\log n)$
- double loops for a variable: $O(n^2)$
- triple loops for a variable: $O(n^3)$
- combination of variables (e.g., shortest path): $O(c^n)$

distributed algorithms

parallel or concurrent algorithms

- split a job and process them by multiple computers
- issues of communication cost and synchronization

distributed algorithms

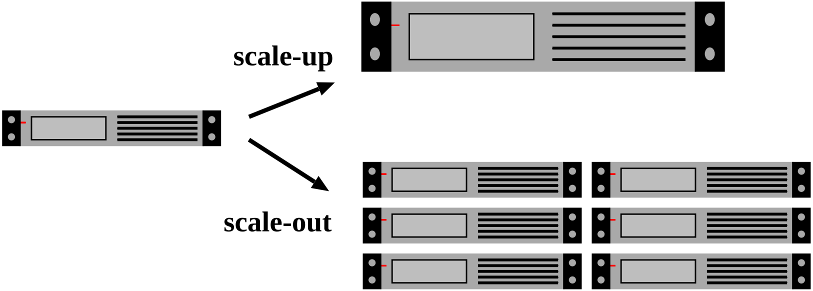
- assume that communications are message passing among independent computers
- failures of computers and message losses

merits

- scalability
  - improvement is only linear at best
- fault tolerance

scale-up and scale-out

- scale-up
  - strengthen or extend a single node
  - without issues of parallel processing
- scale-out
  - extend a system by increasing the number of nodes
  - cost performance, fault-tolerance (use of cheap off-the-shelf computers)



cloud computing

cloud computing: various definitions

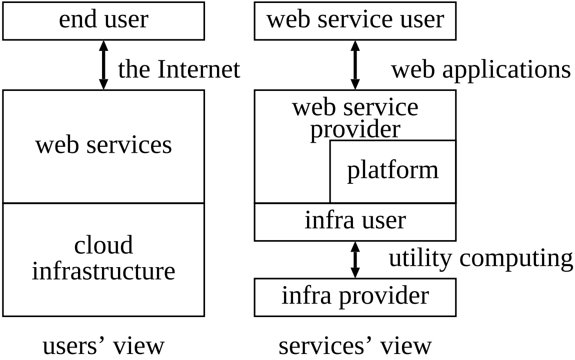
- broadly, computer resources behind a wide-area network

background

- market needs:
  - outsourcing IT resources, management and services
    - no initial investment, no need to predict future demands
  - cost reduction as a result
- as well as risk management and energy saving, especially after the Japan Earthquake
- providers: economy of scale, walled garden
  - efficient use of resource pool

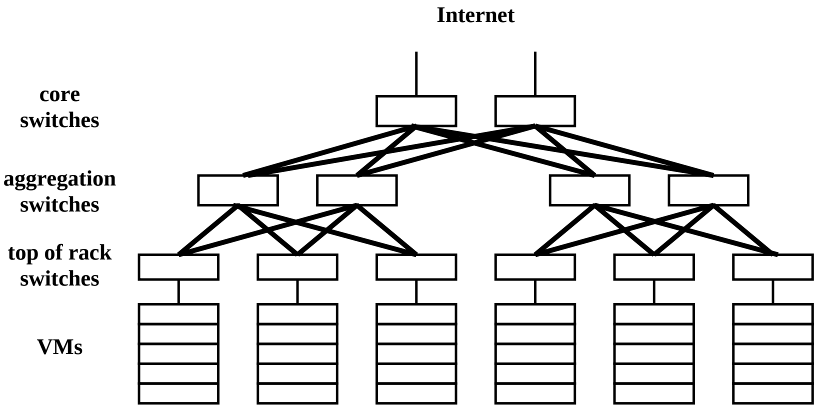
various clouds

- public/private/hybrid
- service classification: SaaS/PaaS/IaaS



physical clouds

typical cloud network topology



key technologies

- virtualization: OS level, I/O level, network level
- utility computing
- energy saving
- data center networking
- management and monitoring technologies
- automatic scaling and load balancing
- large-scale distributed data processing

- related research fields: networking, OS, distributed systems, database, grid computing
  - led by commercial services

economics of cloud

- economies of scale (purchase cost, operation cost, statistical multiplexing)
- commodity hardware
- economical locations (including airconditioning, electricity, networking)

Will Japanese clouds be competitive in the global market?
(The bigger, the better?)

history of search engines

most Internet users use search engines everyday

- 1994 Yahoo! portal started
  - a pioneer of potal 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 serach 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

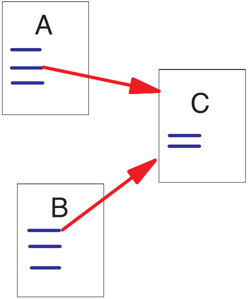
- obtains a list of related web pages by looking up the inverted index with the keywords
- orders the list by ranking, and send it back to the user

web page ranking

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

PageRank: basic idea

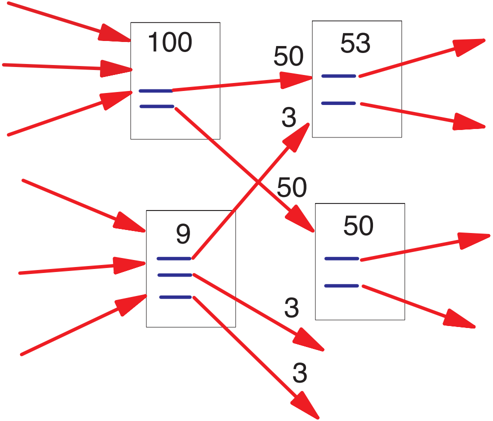
- score web pages only from the link relationship of web pages
  - it does not look at content at all



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

PageRank: insights

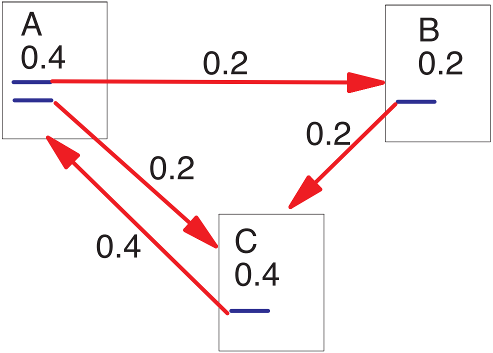
- high quality web pages are linked from many web pages
- a link from higher quality web page is more valuable
- as the number of links within a web page increases, the value of each link decreases



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

PageRank: model

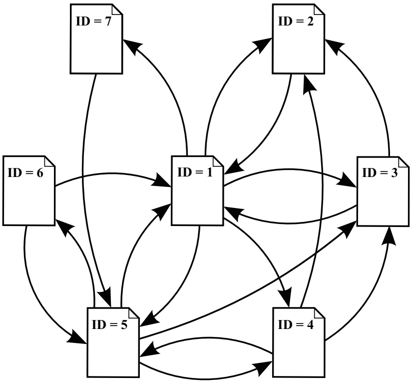
- web pages linked from high quality web pages are high quality
- random surfer model
  - a user clicks links within the same web page with the same probability



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)

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

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

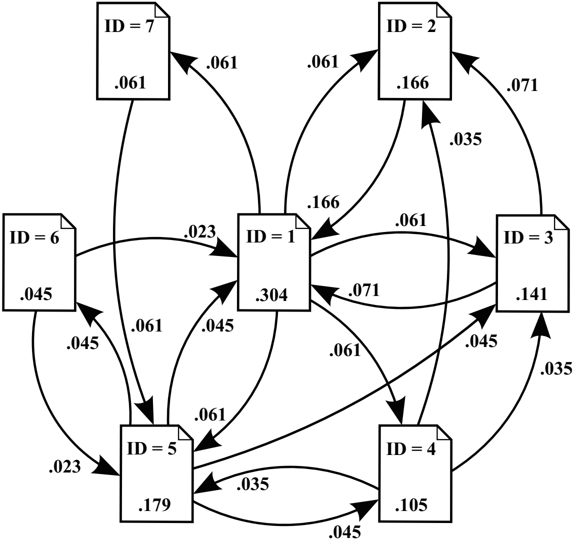
$$A = \begin{bmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathsf{R} = c\mathsf{AR}$$

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

$$\mathsf{A'} = d\mathsf{A} + (1 - d)[1/N]$$

d: damping factor ($=$ 0.85)

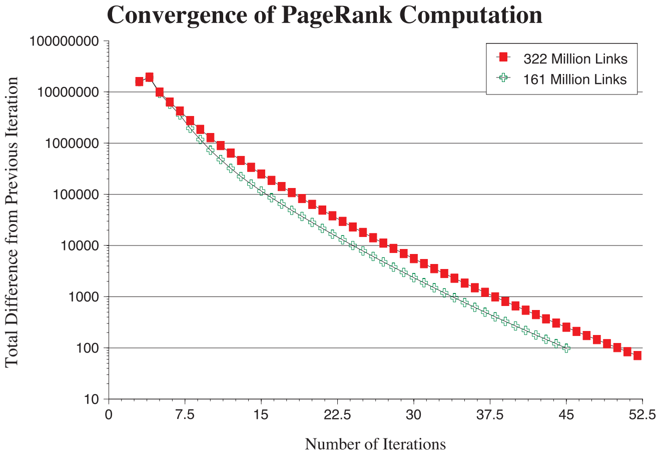
computation by power iteration method

- 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



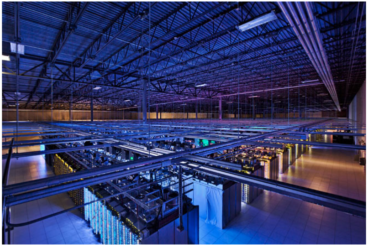
source: L. Page, et al. The pagerank citation ranking: Bringing order to the web. 1998.

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

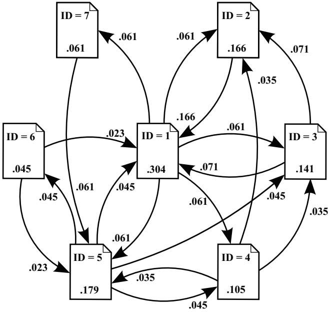
- note: this algorithm was introduced in 1998. the current algorithm used by Google must have evolved significantly since then.

google servers



google system in 1998 and a current data center

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
```

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)

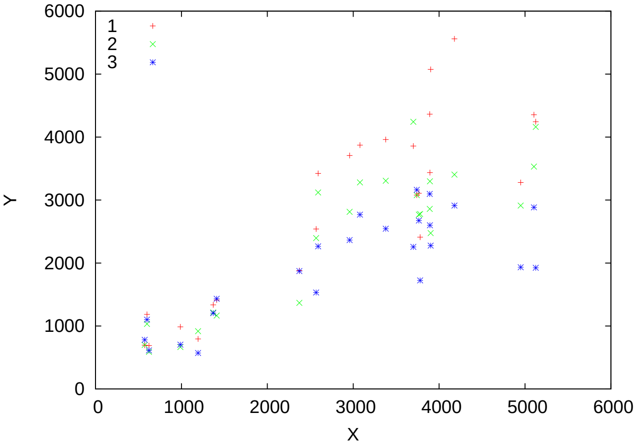
```
# compute pagerank by power method
k = 0 # iteration number
begin
  rank_sum = 0.0  # sum of pagerank of all pages: should be 1.0
  diff_sum = 0.0  # sum of differences from the last round
  last_rank = rank.clone  # copy the entire hash of pagerank

  # 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

    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
```

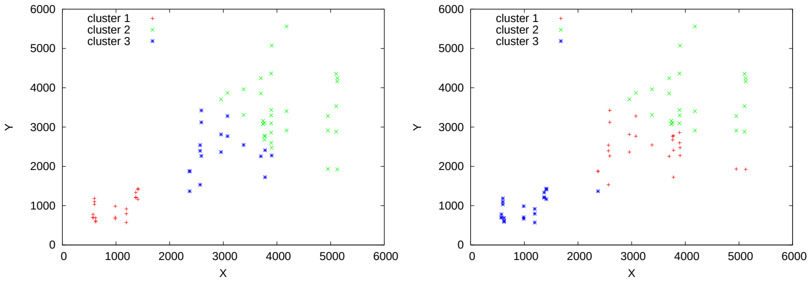
previous exercise: k-means clustering

```
% 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 dsistance 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
```

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
```

on the final report

- select A or B
  - A. Wikipedia pageview ranking
  - B. free topic
- up to 8 pages in the PDF format
- submission via SFC-SFS by 2015-01-29 (Thu) 23:59

final report topics

A. Wikipedia pageview ranking

- purpose: extracting popular keywords from real datasets and observing temporal changes
- data: pagecount datasets from Wikipedia English version
- items to submit
  - A-1 CCDF plot of the pagecount distribution
  - A-2 list of top 10 titles for each day and for the week
  - A-3 plot the changes of the daily ranking of the top 10 titles
  - A-4 other analysis (optional)
    - optional analysis of your choice
  - A-5 discussion on the results
    - describe what you observe from the data

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

more weight on the discussion for the final report

A. Wikipedia pageview ranking

data: pagecount datasets from Wikipedia English version

- original datasets provide by wikimedia
  - http://dumps.wikimedia.org/other/pagecounts-raw/
- pagecount dataset for the report: en-201412.zip (790MB, 2.4GB uncompressed)
  - hourly pagecounts of the week, Dec 1-7, 2014
  - only for English Wikipedia, only 4 hours (00-04 UTC) for each day (to reduce the data size)

data format

- project encoded_pagetitle requests size
  - project: wikimedia project name (all "en" in this dataset)
  - encoded_pagetitle: URI encoded page title
  - requests: the number of requests
  - size: the size of the content

```
$ head -n 10 pagecounts-20141203-030000
en !! 1 9295
en !!! 6 103994
en !!!_(album) 2 23644
en !%20(disambiguation) 1 10393
en !%EF%BF%BD%02 1 6645
en !Adios_Amigos! 1 15951
en !Alabadle! 1 10736
en !Bang! 1 15328
en !Ciauetistico! 2 21038
en !Hero 1 10938
```

a script to decode titles

- titles are percent-encoded
  - can be converted to UTF-8 by ruby's CGI.unescape()

```
#!/usr/bin/env ruby

require 'cgi'

re = /^([\w\.]+)\s+(\S+)\s+(\d+)\s+(\d+)/

ARGF.each_line do |line|
  if re.match(line)
    project, title, requests, bytes = $~.captures
    decoded_title = CGI.unescape(title)
    print "#{project} \"#{decoded_title}\" #{requests} #{bytes}\n"
  end
end
```

A. more on pagecount ranking

- A-1 CCDF plot of the pagecount distribution
  - aggregate all the datasets, sum up all requests for each title, and plot CCDF of the pagecount distribution
  - a log-log plot with request count on the X-axis, CCDF on Y-axis
- A-2 list of top 10 titles for each day and for the week total
  - create a table similar to the following

```
rank 12/1              12/2        12/3                  ...  12/7                  total
1    "Main_Page"      "Main_Page" "Main_Page"            ... "Main_Page"           "Main_Page"
2    "Ethernet_frame" "Cofferdam" "Special:HideBanners"  ... "Special:HideBanners" "Special:HideBanne
        ...
```

- A-3 plot the changes of the daily ranking of the top 10 titles
  - time on X-axis, ranking on Y-axis
  - come up with a good way by yourself to show the changes of ranking over the week

summary

Class 12 Search and Ranking

- Search systems
- PageRank
- exercise: PageRank algorithm

summary of the class: what you learned in the class

- how to understand statistical aspects of data, and how to process and visualize data
  - which should be useful for writing thesis and other reports
- programming skills to process a large amount of data
  - beyound what the existing package software provides
- ability to suspect statistical results
  - the world is full of dubious statistical results and infomation manipulations
  - (improving literacy on online privacy)

- programming and hands-on data analysis
  - just reading textbooks isn't enough
  - certain skills can be learned only through first hand experiences