

Lecture 5-2

# PageRank Algorithm

The Google logo, featuring the word "Google" in its characteristic multi-colored font (blue, red, yellow, blue, green, red).

PageRank



# Plan

---

✓ 2D arrays: Basic concepts

✓ Example: Matrix operations

✓ Internal view of 2D arrays

➔ Aside topic: Reading data from a file and from the standard input

- PageRank algorithm (example of 2D array processing)

# Reading / processing a file

Example: Counts how many times integer values appear in a file.

OS function: displays the file

```
% more demo.dat
4
1 1 2
0 1 3 0 3
0 1

% java ReadFileDemo
0: 3
1: 4
2: 1
3: 2
```

```
/** Represents a standard input stream. Provides methods
    for controlling and reading values from this input */
public class In {

    /** Initializes a new input stream from the given file. */
    In(String fileName)

    /** Reads the next token from the input,
     *  parses it as an integer, and returns the integer. */
    public int readInt()

    /** Returns true if the input is empty. */
    public boolean isEmpty()

    // More In functions follow.
}
```

In class API

# Reading / processing a file

Example: Counts how many times integer values appear in a file.

```
/** Reads all the values in a file, skipping white space, and prints their
 * frequency. Precondition: The first value in the file, say N, indicates
 * that each value in the file is a non-negative int < N. */
public class ReadFileDemo {
    public static void main(String[] args) {
        // Creates a new input stream and sets in to refer to it
        In in = new In("demo.dat");
```

Example of *object-based programming*: **in** is an object of type **In**. It has data and methods

OS function: displays the file

```
% more demo.dat
4
1 1 2
0 1 3 0 3
0 1

% java ReadFileDemo
0: 3
1: 4
2: 1
3: 2
```

```
/** Represents a standard input stream. Provides methods
    for controlling and reading values from this input */
public class In {

    /** Initializes a new input stream from the given file. */
    In(String fileName)

    /** Reads the next token from the input,
     *  parses it as an integer, and returns the integer. */
    public int readInt()

    /** Returns true if the input is empty. */
    public boolean isEmpty()

    // More In functions follow.
}
```

In class API

# Reading / processing a file

Example: Counts how many times integer values appear in a file.

```
/** Reads all the values in a file, skipping white space, and prints their
 * frequency. Precondition: The first value in the file, say N, indicates
 * that each value in the file is a non-negative int < N. */
```

```
public class ReadFileDemo {
    public static void main(String[] args) {
        // Creates a new input stream and sets in to refer to it
        In in = new In("demo.dat");
        // Reads the upper-limit of the values
        int N = in.readInt();
        // Creates a frequency array
        int[] count = new int[N];
        // Reads and counts the values
        while (!in.isEmpty()); {
            int x = in.readInt();
            count[x]++;
        }
        // Prints the frequency array
        for (int i = 0; i < N; i++)
            System.out.println(i + ": " + count[i]);
    }
}
```

Example of *object-based programming*: **in** is an object of type **In**. It has data and methods

Main point of this exercise:  
Illustrates reading a text files using an object-based file reader (In object)

OS function: displays the file

```
% more demo.dat
4
1 1 2
0 1 3 0 3
0 1

% java ReadFileDemo
0: 3
1: 4
2: 1
3: 2
```

```
/** Represents a standard input stream. Provides methods
    for controlling and reading values from this input */
public class In {

    /** Initializes a new input stream from the given file. */
    In(String fileName)

    /** Reads the next token from the input,
     *  parses it as an integer, and returns the integer. */
    public int readInt()

    /** Returns true if the input is empty. */
    public boolean isEmpty()

    // More In functions follow.
}
```

In class API

# Standard output

---

```
public class RandomNumbers {  
    public static void main(String[] args) {  
        int N = Integer.parseInt(args[0]);  
        // Prints N random numbers in [0,1)  
        for (int i = 0; i < N; i++) {  
            System.out.println(Math.random());  
        }  
    }  
}
```

Terminal (aka shell / cmd):

```
% java RandomNumbers 5  
0.4234137005317864  
0.2984657006398488  
0.7456080688315734  
0.0038273723723723  
0.8734883483448448  
%
```

**Terminal:** An interactive OS program that enables managing files and executing programs, using textual commands

“Standard output”: The stream of characters that a program writes

print / println / printf functions write to standard output

By default: Standard output goes to the terminal.

# Standard input

---

Terminal (aka shell / cmd):

```
% java Average
```

# Standard input

---

Terminal (aka shell / cmd):

```
% java Average
```

```
10
```

```
20
```

```
30
```

```
Average: 20.0
```

```
%
```

In each line the user types something and presses “enter” (which produces an “end-of-line” character)

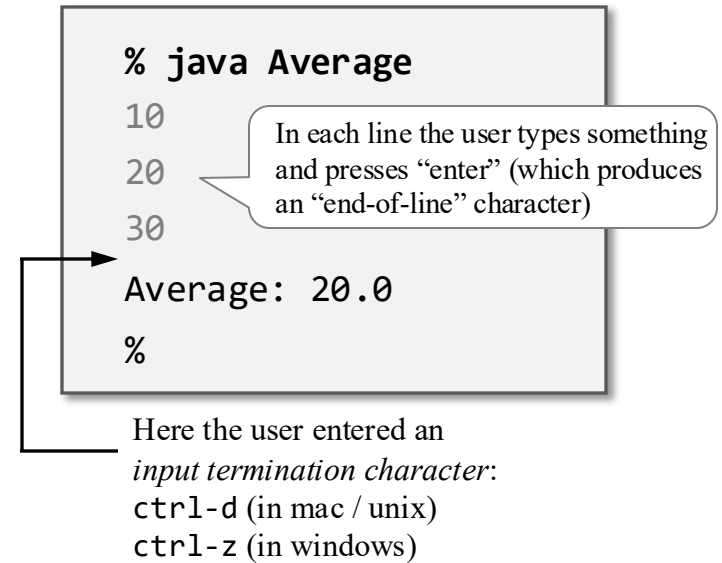
Here the user entered an  
*input termination character*:  
ctrl-d (in mac / unix)  
ctrl-z (in windows)

# Standard input

---

How to manage this  
human – machine  
interaction?  
(HW5...)

Terminal (aka shell / cmd):



# Standard input

```
/** Reads numbers from standard input and prints their average. */  
public class Average {  
    public static void main(String[] args) {  
        // Creates an In object for representing standard input  
        In in = new In();  
    }  
}
```

In: a class for  
reading inputs

Terminal (aka shell / cmd):

```
% java Average
```

```
10
```

```
20
```

```
30
```

```
Average: 20.0
```

```
%
```

In each line the user types something  
and presses “enter” (which produces  
an “end-of-line” character)

Here the user entered an  
*input termination character*:  
ctrl-d (in mac / unix)  
ctrl-z (in windows)

“Standard input”: The stream of characters that a program reads;  
By default, comes from the keyboard.

# Standard input

```
/** Reads numbers from standard input and prints their average. */
public class Average {
    public static void main(String[] args) {
        // Creates an In object for representing standard input
        In in = new In();
        // Assumption: At least one value
        double x = in.readDouble();
        int count = 1;
        double sum = x;
        while (!in.isEmpty()) {
            x = in.readDouble();
            sum += x;
            count++;
        }
        System.out.println("..." + sum / count);
    }
}
```

In: a class for reading inputs

Terminal (aka shell / cmd):

```
% java Average
```

```
10
```

```
20
```

```
30
```

```
Average: 20.0
```

```
%
```

In each line the user types something and presses “enter” (which produces an “end-of-line” character)

Here the user entered an *input termination character*:  
ctrl-d (in mac / unix)  
ctrl-z (in windows)

Remember to compile In.java!

“Standard input”: The stream of characters that a program reads;  
By default, comes from the keyboard.

Java provides no simple way to read inputs from standard input;

In: An open source class for handling standard input, used in this course.

# The `In` class

---

`In`: a library for handling standard input:

---

<code>boolean</code>	<code>isEmpty()</code>	<i>true if no more values, false otherwise</i>
<code>int</code>	<code>readInt()</code>	<i>read a value of type <code>int</code></i>
<code>double</code>	<code>readDouble()</code>	<i>read a value of type <code>double</code></i>
<code>long</code>	<code>readLong()</code>	<i>read a value of type <code>long</code></i>
<code>boolean</code>	<code>readBoolean()</code>	<i>read a value of type <code>boolean</code></i>
<code>char</code>	<code>readChar()</code>	<i>read a value of type <code>char</code></i>
<code>String</code>	<code>readString()</code>	<i>read a value of type <code>String</code></i>
<code>String</code>	<code>readLine()</code>	<i>read the rest of the line</i>
<code>String</code>	<code>readAll()</code>	<i>read the rest of the text</i>
<code>...</code>		

[In API](#) (click)

# Redirection

---

(Previous examples):

```
% java RandomNumbers 3
```

```
0.4234137005317864
```

```
0.2984657006398488
```

```
0.7456080688315734
```

```
% java Average
```

```
100
```

```
200
```

```
100
```

```
300
```

```
<ctrl-d> / <ctrl-z>
```

```
Average: 175.0
```

```
%
```

# Redirection

---

(Previous examples):

```
% java RandomNumbers 3
0.4234137005317864
0.2984657006398488
0.7456080688315734

% java Average
100
200
100
300
<ctrl-d> / <ctrl-z>
Average: 175.0
%
```

Redirection examples:

```
% java RandomNumbers 1000 > data.txt

% more data.txt      (more: OS command for viewing a file)
0.05589098017497873
0.5792882827583564
...

% java Average < data.txt
Average: 0.48881491492997764

% java RandomNumbers 1000000 | java Average
Average: 0.4999861238602861
%
```

## Redirection operators

**> *fileName*:** Directs standard output to a file

**< *fileName*:** Directs standard input from a file

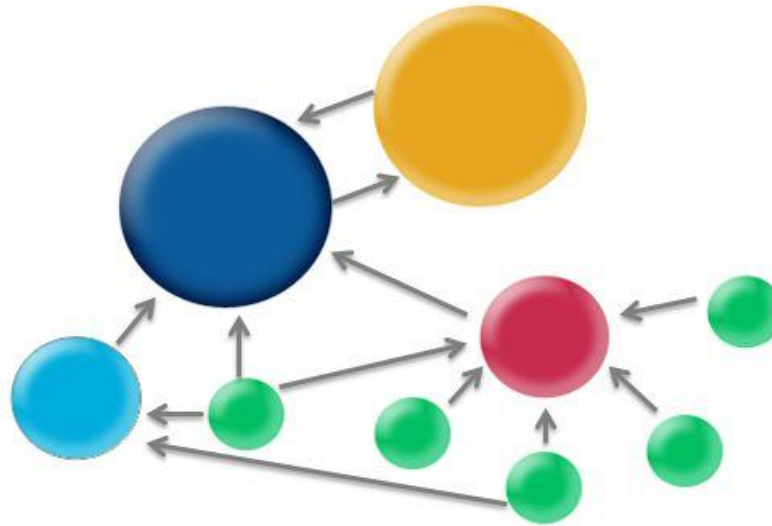
***process1* | *process2*:** Pipes the output of *process1* into the input of *process2*

# Plan

---

- ✓ 2D arrays: Basic concepts
- ✓ Example: Matrix operations
- ✓ Internal view of 2D arrays
- ✓ Aside topic: Reading data from a file and from the standard input

➡ PageRank algorithm (example of 2D array processing)



## PageRank Algorithm

How to Rank Web Pages According to their “Importance”

## The PageRank Citation Ranking: Bringing Order to the Web

January 29, 1998

### Abstract

The importance of a Web page is an inherently subjective matter, which depends on the readers interests, knowledge and attitudes. But there is still much that can be said objectively about the relative importance of Web pages. This paper describes PageRank, a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them.

We compare PageRank to an idealized random Web surfer. We show how to efficiently compute PageRank for large numbers of pages. And, we show how to apply PageRank to search and to user navigation.

---

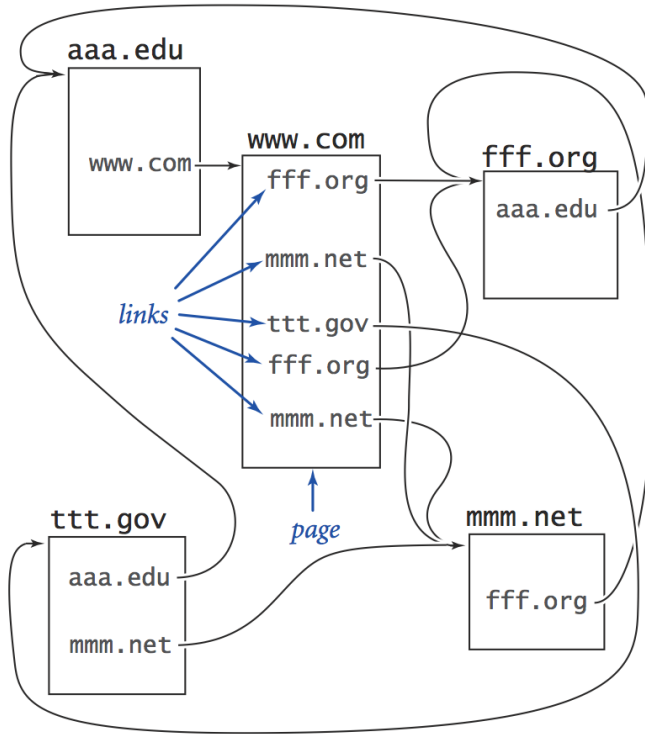


[https://en.wikipedia.org/wiki/History\\_of\\_Google#/media/File:Google\\_page\\_brin.jpg](https://en.wikipedia.org/wiki/History_of_Google#/media/File:Google_page_brin.jpg)

# PageRank

---

Model of the web (consisting of 5 pages with hyperlinks):

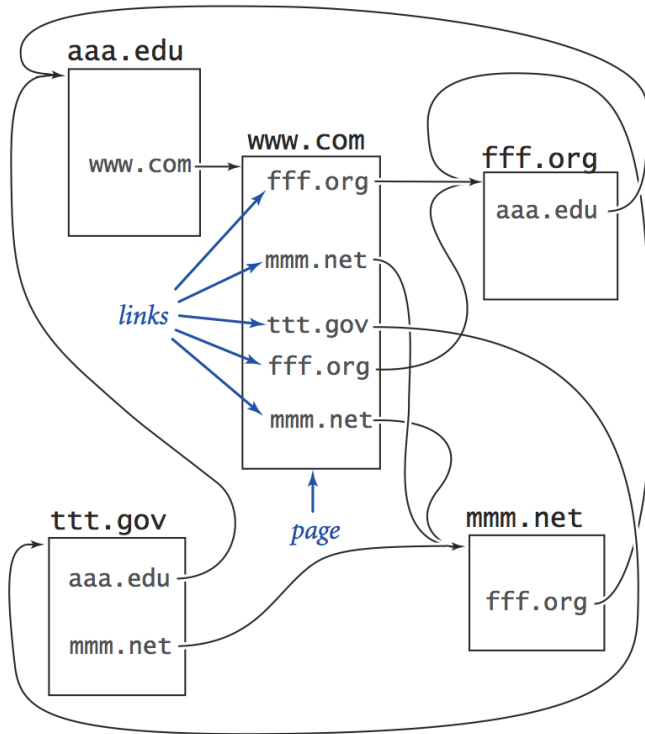


The challenge: Rank web pages according to their *importance*

# PageRank

---

Model of the web (consisting of 5 pages with hyperlinks):



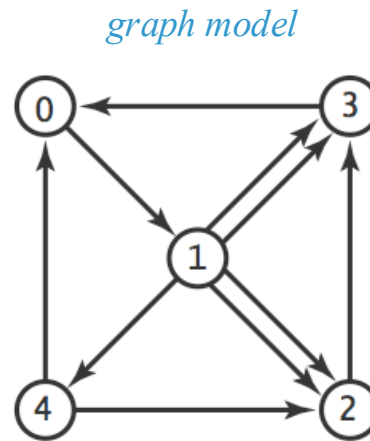
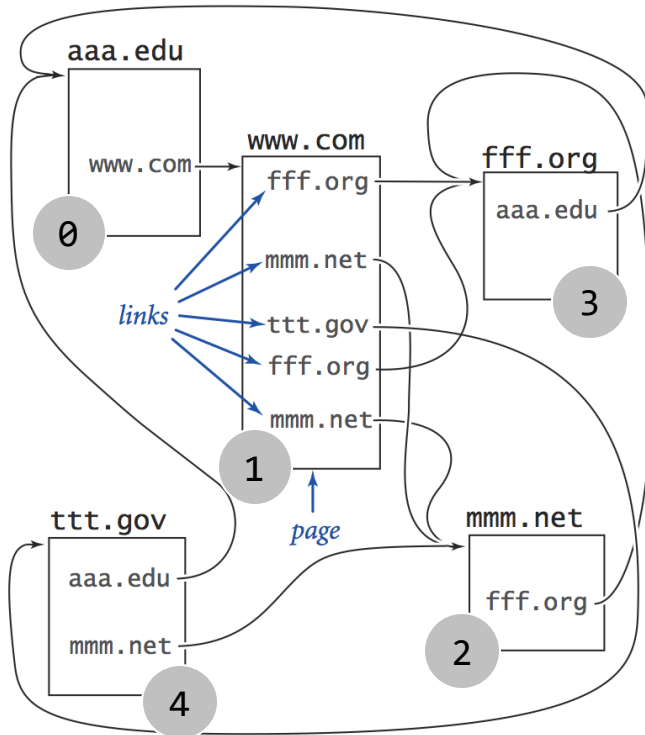
The challenge: Rank web pages according to their *importance*

*Importance*: (i) the more visits a page gets, the more important it becomes  
(ii) pages that get visits from important pages become more important

PageRank: The algorithm that Google uses to measure importance.

# Model

Model of the web (consisting of 5 pages with hyperlinks):



data file

5	← number of pages (N)			
0	1			
1	2	1 2		
1	3	1 3	1 4	
2	3			
3	0			
4	0	4 2		

} links

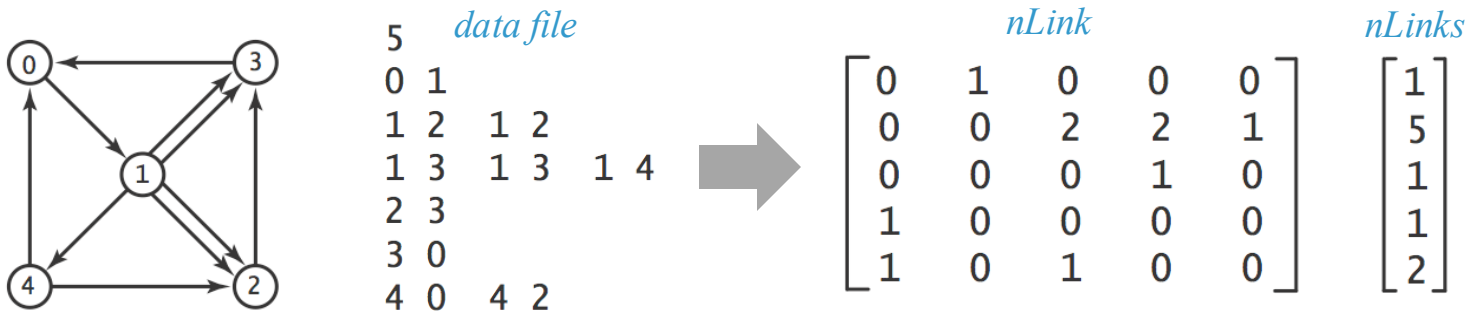
The challenge: Rank web pages according to their *importance*

*Importance*: (i) the more visits a page gets, the more important it becomes

(ii) pages that get visits from important pages become more important

PageRank: The algorithm that Google uses to measure importance.

# Data structures



```

public class PageRank {
    public static void main(String[] args) {
        // Creates an In object for representing the input (file)
        In in = new In("web.dat");
        int N = in.readInt();           // number of pages
        int[][] nLink = new int[N][N]; // nLink[i][j]: number of links from page i to page j
        int[] nLinks = new int[N];     // nLinks[i]: total number of outgoing links from page i

        // Reads the links data and computes the link counts
        while (!in.isEmpty()) {
            int row = in.readInt();
            int col = in.readInt();
            nLink[row][col]++;
            nLinks[row]++;
        }
        ...
    }
}
    
```

```

/** Represents a standard input stream. Provides methods
    for controlling and reading values from this input */
public class In {

    /** Initializes a new input stream from the given file. */
    In(String fileName)

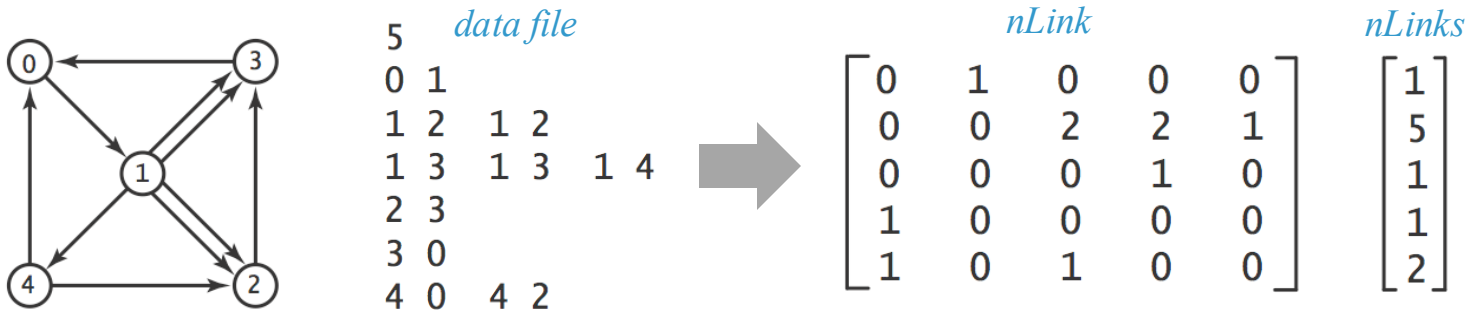
    /** Reads the next token from the input,
        * parses it as an integer, and returns the integer. */
    public int readInt()

    /** Returns true if the input is empty. */
    public boolean isEmpty()

    // More In functions follow.

}
    
```

# Data structures



*leap probabilities*

$$\begin{bmatrix} .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \end{bmatrix} + \begin{bmatrix} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .45 & 0 & .45 & 0 & 0 \end{bmatrix} = \begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$

$p1[i][j]$  = probability of “leaping” from page  $i$  to page  $j$

$p2[i][j]$  = probability of going from page  $i$  to page  $j$  by clicking a link

$p[i][j]$  = probability of going from page  $i$  to page  $j$

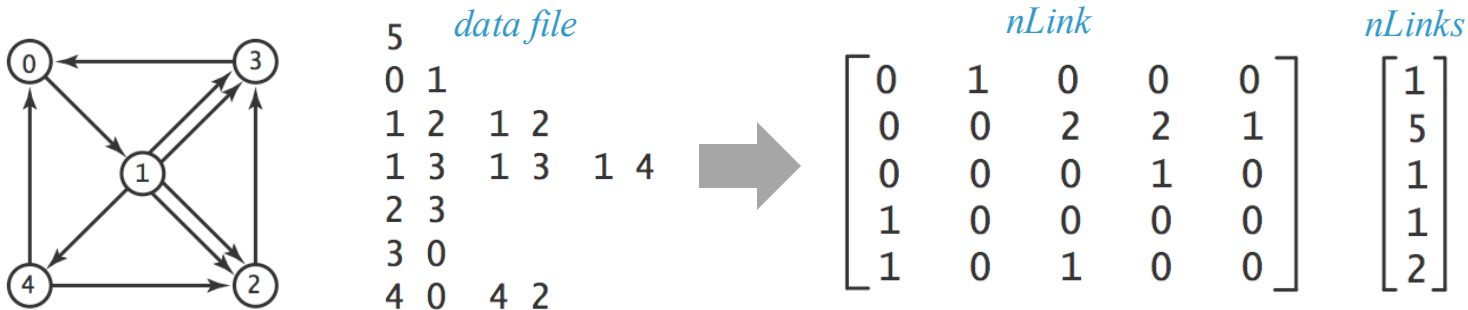
## Web surfing behavior (working assumption of PageRank):

When a user visits a page:

10% of the time the user “leaps” to some random page with equal probability

90% of the time the user clicks a random hyperlink within the current page

# Data structures



The transition matrix is calculated as the sum of the leap probabilities matrix and the link probabilities matrix:

$$\begin{bmatrix} .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \end{bmatrix} + \begin{bmatrix} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .45 & 0 & .45 & 0 & 0 \end{bmatrix} = \begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$

```

...
// Reads the data file and constructs the nLinks and nLink matrices (previous slide)
...
// Constructs the transition matrix
double[][] transition = new double[N][N];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        transition[i][j] = .10 / N +
            .90 * ((double) nLink[i][j] / nLinks[i]);
    }
}
    
```

# Algorithm

---

*transition matrix*

$$\begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$

$p[i][j]$  = probability that a random user  
goes from page  $i$  to page  $j$

## PageRank Algorithm

// Simulates a random user that makes  $T$  moves from one page to another.

// Counts how many times each of the  $N$  pages will be visited.

`int[] count = int [N]`      // stores how many times each page was visited so far

# Algorithm

---

*transition matrix*

$$\begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$

$p[i][j]$  = probability that a random user goes from page  $i$  to page  $j$

## PageRank Algorithm

// Simulates a random user that makes  $T$  moves from one page to another.

// Counts how many times each of the  $N$  pages will be visited.

`int[] count = int [N]`      // stores how many times each page was visited so far

`page = 0`      // starts the random walk at page 0

repeat  $T$  times:

    // Selects the next page, using the row probabilities (Monte Carlo)

`page = select a random integer from 0, ..., N-1 with probability  $p[page, 0], \dots, p[page, N-1]$`

`count [page]++`


// Normalizes the page counts

for  $i = 0, \dots, N-1$ :

`pageRank[i] = count [i] / T`

↑  
In order to perform this simulation, we first have to compute the Cumulative Distribution Functions

# Algorithm

<i>transition matrix (PDFs)</i>		<i>CDFs matrix</i>
$\begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & .92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$		$\begin{array}{l} \text{page 0:} \\ \text{page 1:} \\ \text{page 2:} \\ \text{page 3:} \\ \text{page 4:} \end{array} \begin{bmatrix} .02 & .94 & .96 & .98 & 1.00 \\ .02 & .04 & .42 & .80 & 1.00 \\ .02 & .04 & .06 & .98 & 1.00 \\ .92 & .94 & .96 & .98 & 1.00 \\ .47 & .49 & .96 & .98 & 1.00 \end{bmatrix}$

## PageRank Algorithm

// Simulates a random user that makes  $T$  moves from one page to another.

// Counts how many times each of the  $N$  pages will be visited.

`int[] count = int [N]`      // stores how many times each page was visited so far

`page = 0`      // starts the random walk at page 0

repeat  $T$  times:

    // Selects the next page, using the row probabilities (Monte Carlo)

`page = select a random integer from 0, ..., N-1 with probability p[page, 0], ..., p[page, N-1]`

`count [page]++`

// Normalizes the page counts

for  $i = 0, \dots, N-1$ :

`pageRank[i] = count [i] / T`

↑  
In order to perform this simulation, we first have to compute the Cumulative Distribution Functions

# Algorithm

*transition matrix (PDFs)*

.02	.92	.02	.02	.02
.02	.02	.38	.38	.20
.02	.02	.02	.92	.02
.92	.02	.02	.02	.02
.47	.02	.47	.02	.02



*CDFs matrix*

page 0:	.02	.94	.96	.98	1.00
page 1:	.02	.04	.42	.80	1.00
page 2:	.02	.04	.06	.98	1.00
page 3:	.92	.94	.96	.98	1.00
page 4:	.47	.49	.96	.98	1.00

...

// Computes the CDF's of the PDF's represented by the transition matrix

```
double[][] CDF = new double[N][N];
for (int i = 0; i < N; i++) {
    CDF[i] = MyRandom.CDF(transition[i]);
}
...
```

How to construct a CDF from a PDF,  
and how to generate random events  
using a CDF, was described in lecture 4-1.

/\*\* Library of statistical and random functions.\*/

public class **MyRandom** {

/\*\* Creates a CDF from a given PDF. \*/

public static double[] **CDF**(double[] p)

/\*\* Generates a random integer from a given CDF. \*/

public static int **rnd**(double[] P)

...

}

MyRandom API

# Implementation

## *CDFs matrix*

page 0:	.02	.94	.96	.98	1.00
page 1:	.02	.04	.42	.80	1.00
page 2:	.02	.04	.06	.98	1.00
page 3:	.92	.94	.96	.98	1.00
page 4:	.47	.49	.96	.98	1.00

```
...  
// Reads the data file and constructs the transition and CDF matrices (previous slides)  
...
```

```
/** Library of statistical and random functions.*/  
public class MyRandom {  
    /** Creates a CDF from a given PDF. */  
    public static double[] CDF(double[] p)  
    /** Generates a random integer from a given CDF. */  
    public static int rnd(double[] P)  
    ...  
}
```

MyRandom API

# Implementation

## *CDFs matrix*

page 0:	.02	.94	.96	.98	1.00
page 1:	.02	.04	.42	.80	1.00
page 2:	.02	.04	.06	.98	1.00
page 3:	.92	.94	.96	.98	1.00
page 4:	.47	.49	.96	.98	1.00

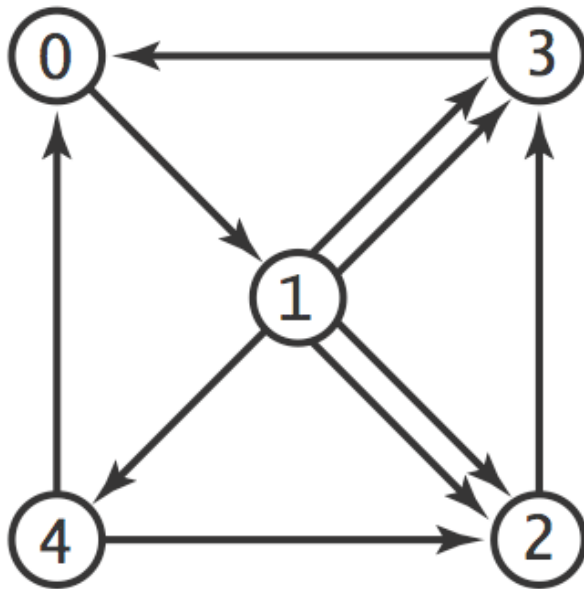
```
...
// Reads the data file and constructs the transition and CDF matrices (previous slides)
...
// Simulates the behavior of a user who makes T random moves (command-line argument).
// In each move, a random page is selected using the transition probability from the current page.
int[] count = new int[N];           // how many times each page was visited
int T = Integer.parseInt(args[0]);  // number of simulated moves from one page to another
int page = 0; // Starts at page 0
// Makes T moves
for (int t = 0; t < T; t++) {
    // Selects randomly which page to go to
    page = MyRandom.rnd(CDF[page]);
    count[page]++;
}
// End of simulation: Prints the pageranks the pages
for (int i = 0; i < N; i++)
    System.out.print((double) count[i] / T);
```

```
/** Library of statistical and random functions.*/
public class MyRandom {
    /** Creates a CDF from a given PDF. */
    public static double[] CDF(double[] p)
    /** Generates a random integer from a given CDF. */
    public static int rnd(double[] P)
    ...
}
```

MyRandom API

# Results

---



```
% more web.dat
```

```
5
```

```
0 1
```

```
1 2 1 2
```

```
1 3 1 3 1 4
```

```
2 3
```

```
3 0
```

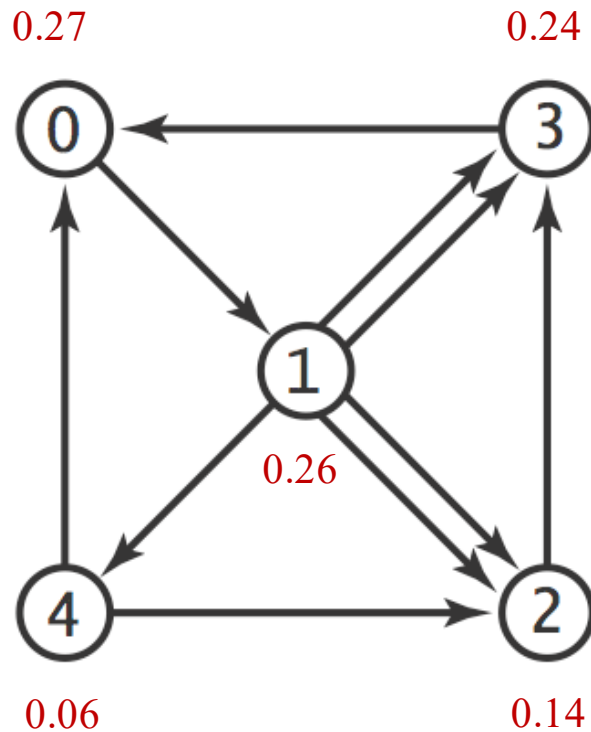
```
4 0 4 2
```

```
% java PageRank 10000000
```

```
Page ranks:
```

```
0.273 0.266 0.146 0.247 0.068
```

# Results



```
% more web.dat
```

```
5
```

```
0 1
```

```
1 2 1 2
```

```
1 3 1 3 1 4
```

```
2 3
```

```
3 0
```

```
4 0 4 2
```

```
% java PageRank 10000000
```

```
Page ranks:
```

```
0.273 0.266 0.146 0.247 0.068
```

# PageRank class (all the pieces together, nothing new in this slide)

```
public class PageRank {
    public static void main(String[] args) {
        // Creates an In object for representing the input (file)
        In in = new In("web.dat");
        int N = in.readInt();           // number of pages
        int[][] nLink = new int[N][N]; // nLink[i][j]: number of links from page i to page j
        int[] nLinks = new int[N];     // nLinks[i]: total number of outgoing links from page i

        // Reads the links data and computes the link counts
        while (!in.isEmpty()) {
            int row = in.readInt();
            int col = in.readInt();
            nLink[row][col]++;
            nLinks[row]++; }

        // Constructs the transition matrix
        double[][] transition = new double[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                transition[i][j] = .10 / N + .90 * ((double) nLink[i][j] / nLinks[i]);

        // Computes the CDF's of the PDF's represented by the transition matrix
        double[][] CDF = new double[N][N];
        for (int i = 0; i < N; i++)
            CDF[i] = MyRandom.CDF(transition[i]);

        // Simulates the behavior of a user who makes T random moves.
        // In each move, a random page is selected from the current page.
        // The random selection is made using the CDF of the current page.
        int[] count = new int[N];           // how many times each page was visited
        int T = Integer.parseInt(args[0]); // number of simulated moves from one page to another
        int page = 0; // Starts at page 0
        // Makes T moves
        for (int t = 0; t < T; t++) {
            // Selects randomly which page to go to
            page = MyRandom.rnd(CDF[page]);
            count[page]++;
        }

        System.out.println("Page ranks:");
        for (int i = 0; i < N; i++) { System.out.printf("%7.3f", (double) count[i] / T); }
    }
}
```

% more web.dat

5

0 1

1 2 1 2

1 3 1 3 1 4

2 3

3 0

4 0 4 2

% java PageRank 10000000

Page ranks:

0.273 0.266 0.146 0.247 0.068