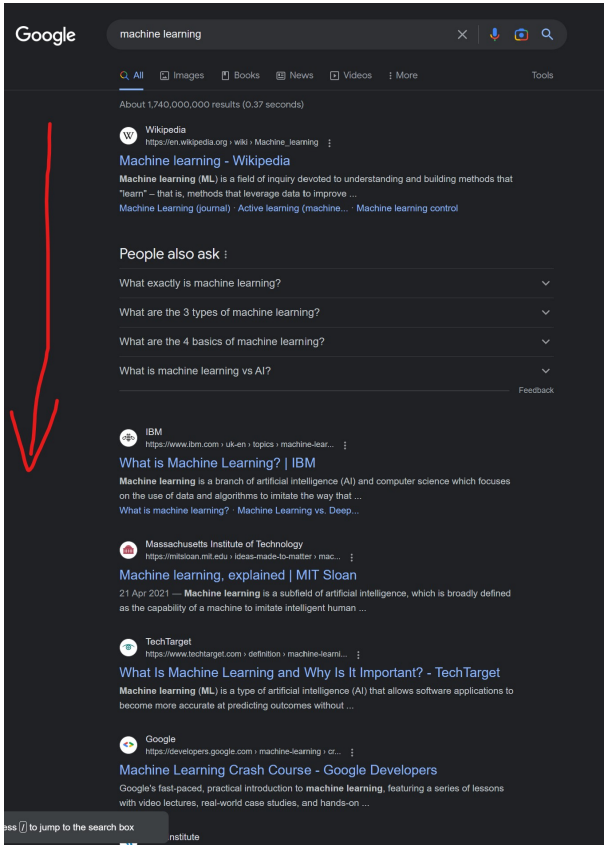
Pagerank: How Math and Algorithm Can Change the World

Motivation:

- The common problem of a internet search is: How to rank webpages that contain the same keywords?



Motivation:

- You want rank **popular content** before low quality contents.
- How do you determine the popularity of a webpage?
    - To make things harder, people may try to fool you into thinking some websites are more popular than it actually is.

Problem Setup

- Since webpages contain links pointing to each other, you can think the structure of internet as a "web",
    - where webpages are "nodes" on the web and links are "edges".
- Structure of the web can be represented by a matrix A:
    - $A_{i,j} = 1$ if website i contains a link **pointing to** j.
    - $A_{i,j} = 0$ if not.
    - $A_{i,i} = 0$ as we do not consider self-links.
- Suppose there are total N websites on internet, A is a $\{0,1\}^{n \times n}$ matrix.

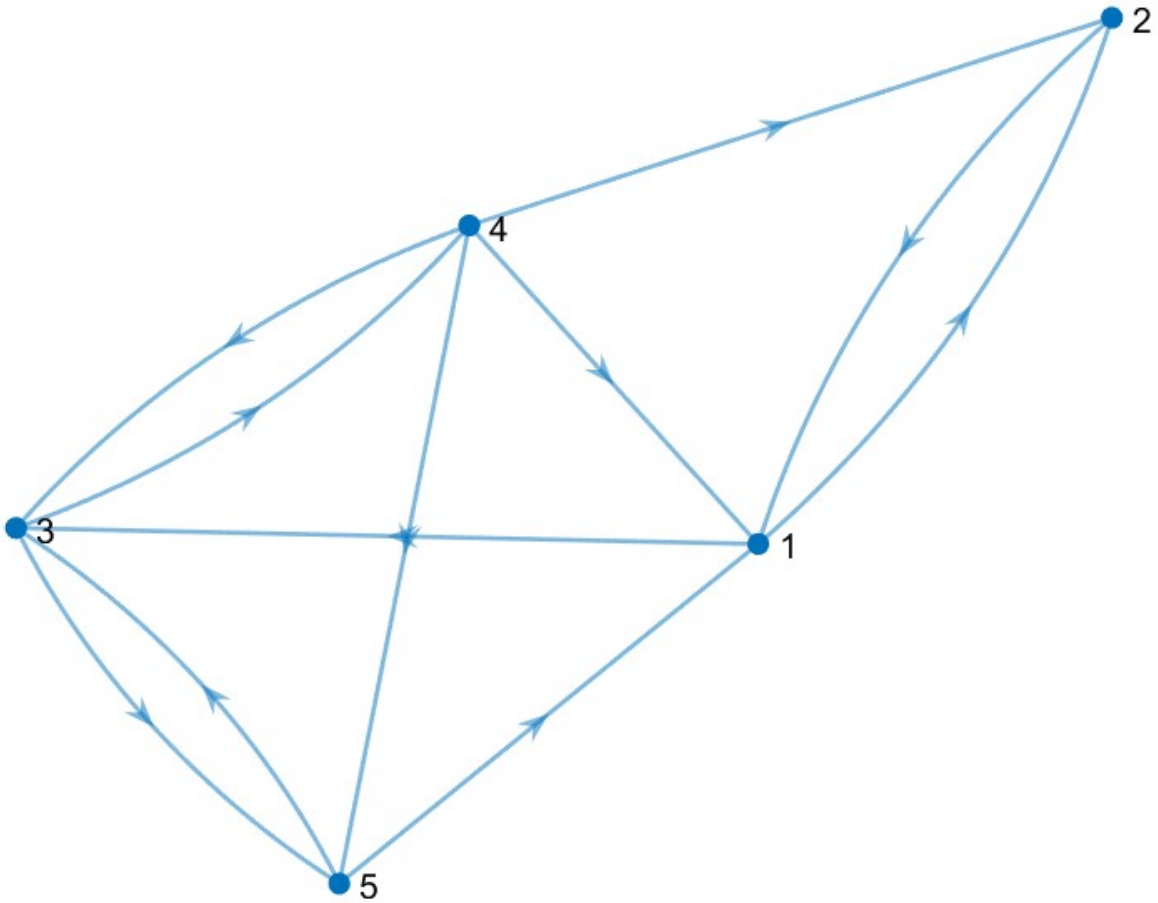
Problem Setup

The matrix A below

```
0 1 1 0 0
1 0 0 0 0
0 0 0 1 1
1 1 1 0 1
1 0 1 0 0
```

induces the following web structure:

Problem Setup



- An arrow from i to j means a webpage i contains a hyperlink points to j.

Popularity Contest

The idea behind Google's pagerank algorithm is a simple popularity contest:

1. If there exists a link from i to j, it means i votes for j in a popularity contest.
2. The webpage receives more votes, are deemed **more popular** than the ones receive less votes.

Popularity Contest

However, this naive algorithm can be easily fooled by junk websites called "link farms".

- Link farms are meaningless websites that contains links to other sites just to boost the popularity of those websites.
- One can easily boost the popularity of a website by creating many link farms pointing to that website.
- Therefore, to make our the algorithm work, we need to introduce an additional rule:

Popularity Contest

- **The vote from a popular website carries more weight than a vote from a less popular website.**
  - A vote from a reputable site (say, wikipedia) carries more weight than a vote from a personal blog.
  - The link farms are less popular as nobody would create a hyperlink to junk sites.

Three Principles of Pagerank

0. A link from i to j represents a vote from i to j.
1. Websites that receives more votes are more popular.
2. Votes from more popular websites carry **more weights**.

How do you design an algorithm that assign popularity according to these principles?

Design the Algorithm

Let us design an iterative algorithm. Denote $p_j^{(t)}$ as the popularity of webpage j at iteration t.

At iteration zero, $p_j^{(0)} \leftarrow 1/N, \forall j$.

- Assume all websites have equal popularity at the begining.

At iteration t, improve p_j by counting the weighted votes

- $q_j \leftarrow \sum_{i \in \{1 \dots N\}} A_{i,j} \cdot p_i^{(t)}$
- $p_j^{(t+1)} \leftarrow q_j / \left(\sum_{j \in \{1 \dots N\}} q_j \right)$
    - Votes of all webpages must sums up to one.

Stopping Criteria

- Stop if $p_j^{(t+1)} \approx p_j^{(t)}$.
- After the algorithm stops, $p_j^{(t+1)}$ is the pagerank for webpage j.

Algorithm Implementation

- In this CW, your code should print out $p_j, j = 1..5$ for the web structure given at the beginning of the slides.
  - I leave the details of implementation to you.
- However, you code must contain a class:

```
class Problem{
  // ... Your code here
public:
  // ... Your code here
  void solve(){
    // Prints out p_j, j from 1 to N
    // Two decimal places, with a space in between
    // e.g.
    // 0.10 0.10 0.20 0.40 0.20
  }
}
```

Algorithm Implementation

Your `main` function should look like this:

```
int main(){
  Problem cw3;
  cw3.solve();
}
//output (example):
//0.10 0.10 0.20 0.40 0.20
```

You are not allowed to modify the `main` function.

- Your program should not have memory leak.

Marking Criteria

- Submitting correct code (10%)
    - Submitting a C++ file with **the correct name**.
    - Your code compiles and runs **without major error** such as **crash**, **infinite loop**.
        - It will be tested using `g++` in the lab pack.
- Writing the correct code to print out pagerank of the five webpages (40%).
    - Do not print anything more than that.
- Good Algorithm Design (30%)
    - You reuse the `matrix` class in previous labs. (15%).
    - Your code is OOP (encapsulation?) (15%).
- Good Coding Practice (20%) (the same as before)