# AMS 562 Homework #4
# C++ Template Programming

Dr. Joseph Schuchart

Due: November 5, 2025

## 1 Objective

The primary objective of this assignment is to delve deep into C++ template programming while implementing the PageRank algorithm. By the end of this assignment, students should gain proficiency in:

- Designing and implementing templated classes and functions in C++.
- Understanding the mathematical basis and computational aspects of the PageRank algorithm.
- Representing matrices in C++ and performing matrix operations.

## 2 Description

The PageRank algorithm, initially used by Google Search, measures the importance of website pages. It counts the number and quality of links to a page to determine the website's importance. In this assignment, students will implement the PageRank algorithm using C++ templates, emphasizing both the mathematical aspects of web search algorithms and advanced C++ template programming.

For simplicity, we will represent the graph using a dense matrix. While there are more efficient sparse matrix representations for large graphs, the dense matrix approach allows us to focus on the core algorithm and C++ template mechanics.

This assignment will primarily be header-only, meaning the main implementations will reside in `.hpp` files. This approach is common for templated libraries in C++.

### 2.1 Test Case: PageRank of Small Web Structure

Consider a small web of 6 pages. The links between these pages are as follows:

1. Page A links to pages B and C.
2. Page B links to pages C and D.
3. Page C links to page A.
4. Page D links to pages B and E.
5. Page E links to pages B, D, and F.
6. Page F links to page E.

**Adjacency Matrix Representation**:

In the context of the PageRank algorithm, the adjacency matrix $A$ is defined such that $A_{ij} = 1$ if there is a link from page $j$ to page $i$, and $A_{ij} = 0$ otherwise. This means that the columns represent the source pages, and the rows represent the destination pages.

Based on the links provided, the adjacency matrix $A$ (before normalization) is:

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

Where rows and columns correspond to pages A, B, C, D, E, and F respectively.

## 2.2 Templated Graph Representation

Design a templated class `Matrix<T>` where `T` can be any numeric type (e.g., `int`, `float`, `double`). The matrix may be represented using `std::vector<std::vector<T>>` or a `std::vector<T>` of size $N \times N$ with either manual index computation or using `std::mdspan`. Implement member functions to set values, normalize columns, and retrieve values. Ensure that the columns are normalized so that they sum to 1, handling dangling nodes (columns with all zeros) by setting their entries to $\frac{1}{N}$, where $N$ is the number of pages.

## 2.3 Templated Vector Representation

Design a templated class `Vector<T>` where `T` can be any numeric type (e.g., `int`, `float`, `double`). The vector should be represented using `std::vector<T>`. Implement member functions to set values, retrieve values, and perform basic vector operations as needed for the PageRank algorithm, such as scalar multiplication, vector addition, dot product, and computation of norms (e.g., the 1-norm).

## 2.4 Templated PageRank Algorithm

Design a templated function `pageRank<T>(const Matrix<T>& M, Vector<T>& r)` that computes the PageRank values for a given transition probability matrix $M$ and initializes the rank vector `r`. Implement the PageRank iteration within this function, ensuring it works for different numeric types.

## 2.5 Implementation

1. **Initialization**:
   - Normalize the columns of the adjacency matrix $A$ to create the transition probability matrix $M$, ensuring that each column sums to 1.
   - If there exists a dangling node (i.e., a column in $A$ are all zeros), then set the entire column to $\frac{1}{N}$, where $N$ is the total number of pages. (In the particular test case, we do not have dangling nodes.)
2. **PageRank Iteration**:
   - Initialize a rank vector $r$ of size 6 with all entries equal to $\frac{1}{6}$. This represents the initial rank of the pages.
   - Define a damping factor $\alpha = 0.85$.
   - Define a teleportation vector $s$ of size 6 with all entries equal to $\frac{1}{6}$.
   - Update the rank vector using the formula:

$$r' = \alpha M r + (1 - \alpha) s$$

   - Iterate the above step until convergence is achieved. Convergence is defined when the 1-norm of the difference between consecutive rank vectors is below a threshold of $1 \times 10^{-6}$:

$$\|r' - r\|_1 = \sum_{i=1}^{N} |r'_i - r_i| < 1 \times 10^{-6}$$

- After each iteration, update $r$ with the new values from $r'$.

3. **Unit Testing**:
   - Implement unit tests for each class and method to validate their correctness.
   - Tests should include both edge cases and normal scenarios.
4. **Verification**:
   - Ensure that the columns of your matrix are normalized so that they sum to 1.
   - For the given test case, verify the PageRank results by comparing them with known solutions or by checking that the rank vector sums to 1. To obtain a known result, you can use the fact that when $\alpha = 1$, then the PageRank vector is the principal eigenvector of the transition probability matrix $M$ (the vector is $[0.230769, 0.230769, 0.230769, 0.153846, 0.1153846, 0.038462]^T$ for the specific example above).

# 3 Submission Guidelines

## 3.1 GitHub Repository

1. **Create a Private Repository:**
   - Sign in to your GitHub account and create a private repository named `AMS562_Homework4`.
2. **Push Your Code:**
   - Include all C++ header (`.hpp`) files, source (`.cpp`) files, and any associated test files. Make sure to follow the project structure:

```
project_root/
|-- Matrix.hpp
|-- Vector.hpp
|-- PageRank.hpp
|-- main.cpp
|-- Makefile
|-- README.md
```

   - Add a `README.md` file that includes:
     - **Project Overview:** A brief description of the project and its objectives.
     - **File Structure:** Explanation of the file organization.
     - **Compilation Instructions:** Step-by-step guide on how to compile the project using the provided Makefile.
     - **Usage Instructions:** How to run the tests and any example commands for running the PageRank algorithm.
3. **Share the Repository:**
   - Add the TA (`yuxuanye1`) as a collaborator to your private repository.
   - Ensure that your last commit is made before the project deadline.

### 3.2 Code Documentation

- **Commenting:**
  - Use Doxygen-style comments for documenting classes and methods.
- **Code Quality:**
  - Follow consistent coding standards and naming conventions.
  - Ensure proper indentation and code formatting for readability.

### 3.3 Report Submission

1. **Report Document:**
   - Create a 1–2 page report.
   - Format the report with Times New Roman, font size 12, single line spacing, and 1-inch margins.
2. **Report Content:**
   - **Repository URL:** Include the URL of your GitHub repository.
   - **Classes and Templates:** Provide a brief description of each class and template programming concepts demonstrated.
   - **Testing and Verification:** Summarize the testing process and results.
   - **Lessons Learned and Challenges:** Reflect on the project and challenges faced.
3. **Submission:** Upload the report as a PDF to Brightspace before the deadline.

### 3.4 Deadlines

- **Project Deadline:** November 5, 2025, by 11:59 PM EDT.

## 4 Grading Rubric

- **Code Structure (30%)**:
  - Organization, Naming Conventions, C++ Best Practices.
- **Implementation (30%)**:
  - Functionality, Error Handling, Efficiency, Code Quality.
- **Unit Tests (20%)**:
  - Coverage, Effectiveness, Implementation.
- **Documentation (10%)**:
  - Doxygen Comments, Report Quality.
- **Submission (10%)**:
  - GitHub Repository, Report Submission.