DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 1: PageRank

September 9, 2022 Métodos Numéricos

Grupo 18

Integrante	LU	Correo electrónico
Vekselman, Natán	338/21	natanvek11@gmail.com
Arienti, Federico	316/21	fa.arienti@gmail.com
Barcos, Juan Cruz	463/20	juancruzbarcos@hotmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

RESUMEN

El Ranking de Page, *PageRank* [1], es un método propuesto por Sergey Brin y Larry Page —co-fundadores de Google—, para establecer la importancia de una página web dentro del internet, o dentro de un subconjunto de las páginas que lo componen. Holísticamente, la relevancia de un sitio se interpretará como la fracción de tiempo, al largo plazo, que un navegante permanecerá en él [2].

Desde una perspectiva algorítmica, PageRank busca resolver un sistema lineal $\mathbf{A}x = x$, donde $\mathbf{A}$ es una matriz estocástica en columnas [2] y cada una de sus posiciones a_{ij} representa la probabilidad que un usuario situado en la página j decida navegar a la página i.

Este trabajo propone una implementación eficiente del ranking a través del uso de una estructura de matríz acorde al problema, y el empleo de iteradores específicos, para reducir el costo espacial y temporal de la eliminación gaussiana, método utilizado para su resolución.

Se buscará dar una presentación teórica y una evaluación cuantitativa y cualitativa de los resultados de tanto el método propuesto, como de PageRank en si.

Palabras clave: *Ranking de Page, Eliminación Gaussiana, Matrices ralas*

Contenidos

1. Introducción Teórica

1.1. **Aridad.** Consideremos primero el dominio y la imágen de PageRank.

Dominio: 1. un conjunto de páginas web interconectadas a través de hipervínculos. Podemos considerar este conjunto como un grafo direccionado, donde los nodos son los sitios y los ejes, los links. 2. un parámetro de entrada $p \in (0,\ 1)$, que representa la probabilidad que un usuario decida navegar aleatoriamente a otra página en el grafo. Se puede interpretar como el parámetro de un variable aleatoria de Bernoulli.

Imágen: un vector $x \in [0,\ 1]^n$, donde x_i representa el Ranking de Page para la i-ésima página del conjunto de entrada, donde x satisface que $x_i \geq 0\ \forall i : 0\ ...\ n$ y $\sum_{i=1}^{n} x_i = 1$.

Tenemos entonces:

$$PageRank :\ G_n \times (0,\ 1) \longrightarrow [0,\ 1]^n \qquad \forall n \in \mathbb{N} \tag{1}$$

donde G_n refiere al conjunto de conjuntos de páginas web, interconectadas a través de hipervínculos, con cardinalidad n.

1.2. **El sistema.** PageRank propone resolver un sistema de ecuaciones para encontrar la relevancia de cada página i $(i : 1\ ...\ n)$ en $g \in G_n$:

$$x_i := \sum_{j=1}^{n} x_j \cdot Pr(j \longrightarrow i) \tag{2}$$

donde $Pr(j \longrightarrow i)$ es la probabilidad que un usuario situado en la página j decida ir a la página i. Se define de la siguente manera:

$$Pr(j \longrightarrow i) := \begin{cases} (1-p) \cdot \frac{1}{n} + p \cdot \frac{I_{ij}}{c_j} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si no} \end{cases} \tag{3}$$

con $I_{ij} = 1$ si y sólo si existe un hipervínculo de j a i, con $j \neq i$ —y nulo en caso contrario—, y $c_j = \sum_{i=1}^{n} I_{ij}$, la cantidad de links salientes de j. La restricción $j \neq i$ será para evitar que se consideren auto-referencias en el ranking.

Notemos que $Pr(j \longrightarrow i)$ se puede interpretar de la siguente manera: un navegante situado en la página j decidirá con probabilidad p acceder a uno de los links del sitio y con probabilidad $1 - p$ saltar a otra página del conjunto. En ambos casos, deberá luego decidir uniformemente sobre el total disponible, y terminará eligiendo a i con una probabilidad de $\frac{I_{ij}}{c_j}$ ó $\frac{1}{n}$, respectivamente, acorde a la primer decisión. Si no hay links en la página, siempre saltará de manera uniforme a otra página del conjunto, y eligirá a i con probabilidad $\frac{1}{n}$.

x_i, por su parte, también recibe una interpretación particular: es la probabilidad que para algún momento $k > K$, el navegante se encuentre situado en la página i. Para un K lo suficientemente grande, esta probabilidad es única [4]. Notar que esto es equivalente a decir que x_i representa la fracción de tiempo, al largo plazo, que un navegante permanecerá en la página i.

A este modelo se lo conoce como el *modelo del navegante aleatorio*. El mismo asume lo siguiente: Dado un link de la página j a la página i, la página j le conferirá importancia a la página i de manera proporcional a la relevancia de j e inversamente proporcional al número de páginas a las que apunta j [4]. Esto se puede ver en que x_i es la suma de toda otra importancia x_j ponderada por $Pr(j \longrightarrow i)$, que incluye en su definición una división sobre la cantidad de links salientes c_j.

1.3. **Representación matricial.** Dado que estamos trabajando con un sistema lineal, será de utilidad considerar la matriz asociada '$\boldsymbol{A}$' y resolver, equivalentemente, $\mathbf{A}x = x$, donde $x = (x_1, \ \dots \ , \ x_n)^t$. Definimos entonces:

$$a_{ij} := Pr(j \longrightarrow i) \tag{4}$$

y proponemos que[1]:

$$\mathbf{A} = p\mathbf{WD} + ez^t \tag{5}$$

donde, $\forall i, \ j : \ 1 \ ... \ n$, se satisface que:

$$e_i = 1$$

$$z_j = \begin{cases} (1-p)/n & \text{si} \quad c_j \neq 0 \\ 1/n & \text{si no} \end{cases}$$

$$w_{ij} = \begin{cases} 1 & \text{si} \quad i \neq j \ \wedge \ j \xrightarrow{l} i \\ 0 & \text{si no} \end{cases}$$

$$d_{ij} = \begin{cases} 1/c_j & \text{si} \quad i = j \ \wedge \ c_j \neq 0 \\ 0 & \text{si no} \end{cases}$$

La notación $j \xrightarrow{l} i$ representa que existe un link de la página j a la página i, y las filas y columnas de $\mathbf{W}$, denominada *matriz de conectividad*, representan —indexadas por posición— las páginas de una web $g \in G_n$.

[1] Una demostración de esta equivalencia se encuentra en 5.1.

A partir de la ecuación (5) podemos ver que:

$$\begin{aligned} \mathbf{A}x &= x \\ (p\mathbf{WD} + ez^t)x &= x \\ p\mathbf{WD}x + ez^t x &= x \\ x - p\mathbf{WD}x &= ez^t x \\ (\mathbf{I} - p\mathbf{WD})x &= \gamma e \end{aligned}$$

donde $\gamma = z^t x$ es un escalar.

Dado que nuestro resultado deberá ser normalizado para cumplir con los requerimientos de la imágen (con la intuición que, si cada x_i representa una probabilidad disjunta, entonces las suma de las x_i deberá ser igual a uno, ya que su unión representa la probabilidad del espacio entero), podemos asumir un γ conveniente [3], $\gamma = 1$, tal que el sistema a resolver sea:

$$(\mathbf{I} - p\mathbf{WD})x = e \tag{6}$$

Notar que La matriz $\mathbf{I} - p\mathbf{WD}$ permite la aplicación de la eliminación gaussiana sin permutación [2].

[2]Una demostración de este enunciado se encuentra en 5.2.

2. Desarrollo

2.1. **Implementación.** A partir del sistema planteado en la ecuación (6), proponemos el siguiente método para la resolución de PageRank:

1) Construir la matriz $\mathbf{I} - p\mathbf{WD}$ a partir de alguna representación de $g \in G_n$ (recordar que g es un conjunto de páginas web con cardinalidad n).
2) Triangular la matriz extendida $(\mathbf{I} - p\mathbf{WD} \mid e)$ mediante eliminación gaussiana, sin pivoteo [3].
3) Resolver la matriz triangular resultante mediante el algoritmo de sustitución inversa.
4) Normalizar el resultado.

Tenemos entonces:

```
1 proc PageRank(in g: G<n>, in p: real, out x: vector<n>) {
2     /* Pre: 0 < p < 1 */
3     B := construir(g, p) // B = I - pWD
4     e := vector(n, 1)    // e = 1 de dimension n
5     eliminacion_gaussiana(B, e)
6     x := sustitucion_inversa(B, e)
7     normalizar(x)
8     /* Post: |Bx - x| < 1e-4 */
9 }
```

Algoritmo 1. Pseudocódigo para *PageRank*.

De este algoritmo surgen las siguientes preguntas: ¿Cómo representamos las matrices[4]? ¿Cómo construimos $(\mathbf{I} - p\mathbf{WD} \mid e)$? ¿Cómo implementamos la eliminación gaussiana? y ¿Cómo implementamos la sustitución inversa?

2.1.1. *Matríz.* Una representación eficiente de matríz será una que permita aprovechar las cualidades de $(\mathbf{I} - p\mathbf{WD} \mid e)$, que tenga un costo mínimo de mantenimiento respecto a sus operaciones elementales y que sea eficiente en el uso de la memoria.

Lo primero a notar son las operaciones fundamentales de la estructura. Desde un punto de vista abstracto, estas son aquellas que la definen como un espacio vectorial y, en el caso de $\mathbb{R}^{n \times n}$, como un álgebra:

[3] Es decir, nuestra solución no buscará reducir el error numérico de la aritmética de punto flotante —en particular por cancelación catastrófica— por medio de las técnicas de pivoteo parcial o pivoteo completo que se utilizan en algunas implementaciones del algoritmo.

[4] Para este trabajo asumiremos que existe una representación de vector. Sin embargo, notar que un vector puede ser representado por una matríz $n \times 1$ ó $1 \times n$.

$$\mathbf{A} + \mathbf{B} := (a+b)_{ij} = a_{ij} + b_{ij} \qquad \forall i : 1 \ldots n,\ j : 1 \ldots m$$

$$\mathbf{A} \cdot \mathbf{B} := (ab)_{ij} = \sum_{k=1}^{m} a_{ik}\ b_{kj} \qquad \forall i : 1 \ldots n,\ j : 1 \ldots q$$

$$\lambda \cdot_{\mathbb{R}} \mathbf{A} := (\lambda a)_{ij} = \lambda a_{ij} \qquad \forall i : 1 \ldots n,\ j : 1 \ldots m$$

Más allá de las posibles implementaciones —que no se requerirán—, importa destacar el fuerte carácter iterativo (las operaciones y la acción actúan sobre todas las posiciones) y observar que cuando $a_{ij} = 0$ ó $b_{ij} = 0$, el resultado es invariante respecto a al menos uno de los operandos. Desde un punto de vista algorítmico, esto significa que estos casos son redundantes y posiblemente se los pueda omitir (por ejemplo, iterando sólo sobre los elementos no nulos).

Esto es particularmente importante para PageRank, donde se espera que una página web tenga pocos links salientes (en relación al total de los sitios) y donde se puede demostrar una tendencia a la localidad de las relaciones [3]. Entonces, para un conjunto lo suficientemente grande, podemos suponer que nuestra matríz estará principalmente vacía.

A partir de este breve análisis, proponemos las siguientes estructuras[5]:

$$(7) \quad \text{matriz} := \begin{cases} \text{n, m} : \mathbb{N} & \\ \text{datos} : \textit{vector} < \textit{vector} < \text{par} := \left\{ \begin{array}{l} \text{posición} : \mathbb{N}_0, \\ \text{elemento} : \mathbb{R} \end{array} \right. >> & \\ \text{at} : \mathbb{N}\ i \times\ \mathbb{N}\ j\ \longrightarrow\ \mathbb{R} & \{0 \leq i < n \wedge 0 \leq j < m\} \\ \text{set} : \mathbb{N}\ i \times\ \mathbb{N}\ j\ \times \mathbb{R} \longrightarrow\ \text{matriz} & \{0 \leq i < n \wedge 0 \leq j < m\} \end{cases}$$

$$(8) \quad \text{iterador} := \begin{cases} \text{i, j, pos} : \mathbb{N}_0 & \\ \text{p} : *\textit{matriz} & \\ \text{at} :\longrightarrow\ \mathbb{R} & \{\text{en\_rango}()\} \\ \text{set} : \mathbb{R} \longrightarrow\ \text{iterador} & \{\text{en\_rango}()\} \\ \text{proximo\_fila} :\longrightarrow\ \text{iterador} & \\ \text{proximo\_columna} :\longrightarrow\ \text{iterador} & \\ \text{en\_rango} :\longrightarrow\ \text{bool} & \end{cases}$$

donde *vector* se refiere a un arreglo de tamaño variable en memoria contigua y $*$ designa un puntero.

[5] Una evaluación en más detalle de la implementación y las alternativas consideradas se encuentra en (5.3).

Esta estructura de matríz garantizará que el tamaño del vector externo siempre equivaldrá a n, que el tamaño de cada vector interno estará acotado por m, que los pares internos estarán ordenados por posición y que para cada uno se satisfará que $0 \leq \text{posición} < n$ y que el elemento es no nulo.

La estructura asociada al iterador, por su parte, satisfará que el iterador es válido si y sólo si $0 \leq i < n$ y $0 \leq pos < largo(datos[i])$, y que un iterador válido siempre estará sobre un elemento no nulo[6].

Entre ambas podremos iterar por los elementos no nulos de cada fila en $\Theta(1)$ —ya que estarán en orden sucesivo en el vector interno— y será eficiente en espacio[7]. Como contrapartida, no será eficiente en la inserción: en el peor caso requerirá mover todos los elementos en una fila (por un costo en $\Theta(m)$).

2.1.2. *Construir(g, p).* Dada nuestra representación de matríz, el siguiente paso será construir, a partir de un conjunto de páginas web g y un valor p, la matríz $\mathbf{I} - p\mathbf{WD}$. Para ello, vamos a adoptar una repesentación particular de g:

$$g \in G_n := \begin{cases} \#\text{páginas} : n, \\ \text{relaciones} : vector < \text{eje} := \{ \text{ i, j} : \mathbb{N} \ > \end{cases} \tag{9}$$

donde cada eje representa un hipervínculo de la página j a la página i y se satisface que $0 \leq i,\ j < n$.

Proponemos el siguiente algoritmo:

```
1 proc construir(in g: G<n>, in p: real, out B: matriz<n, n>) {
2     n := g.#paginas
3     B := identidad(n)    // B = I de dimension n x n
4     ponderar(B, g, p)    // B = I - pWD
5 }
```

Algoritmo 2. Pseudocódigo para *construir*.

Notamos que la matríz $\mathbf{D}$ es diagonal y, por ende, la multiplicación a derecha con $\mathbf{W}$ equivaldrá a escalar los elementos de cada columna W_j por d_{jj}. En el caso de los elementos nulos, la operación será invariante —algo que también sucederá con la multiplación escalar por $-p$—.

[6]Sobre el iterador, debemos aclarar que éste se situará en la columna no nula más cercana a la pedida, que el método PROXIMO_COLUMNA situará al iterador sobre la columna no nula más próxima (en relación a la que se usó para inicializar el iterador originalmente) de la siguiente fila, o continuará si esta es vacía, y PROXIMO_FILA situará al iterador sobre el próximo elemento no nulo en la misma fila.

[7]En relación a nuestras expectativas sobre la ralidad de PageRank.

Tenemos entonces:

```
proc identidad(in n: natural, out B: matriz<n, n>) {
    B := matriz(n, n, 0) // B = 0 de dimension n x n
    i := 0
    while i < n {
        B.set(i, i, 1)
        i := i + 1
    }
}
```

ALGORITMO 3. Pseudocódigo para *identidad*.

```
proc ponderar(inout B: matriz<n, n>, in g: G<n>, in p: real) {
    grado := vector(n, 0) // grado = 0 de dimension n
    i = 0
    while i < largo(g.relaciones) {
        eje := g.relaciones[i]
        grado[eje.j] := grado[eje.j] + 1
    }
    while i < largo(g.relaciones) {
        eje := g.relaciones[i]
        if eje.i != eje.j {
            B.set(eje.i, eje.j, -p / grado[eje.j])
        }
    }
}
```

ALGORITMO 4. Pseudocódigo para *ponderar*.

De ambos algoritmos podemos notar que la complejidad de CONSTRUIR estará en $\Theta(n+r\cdot c)$ donde r representa la cantidad total de relaciones y c es el costo de SET. Para nuestra representación, esto equivaldrá a $\Theta(n + r \cdot n)$ y, como $r \leq n^2$, tendremos un peor caso en $\Theta(n^3)$. Sin embargo, si $r << n^2$, se puede esperar un comportamiento mejor que el método 'directo' (cuyo costo fijo es $\Theta(n^3)$ por el producto de matrices).

2.1.3. *Eliminación gaussiana.* El siguiente paso será triangular $(\mathbf{I} - p\mathbf{WD} \mid e)$ para obtener un sistema fácil de resolver. Proponemos una versión optimizada de la eliminación gaussiana —para nuestro uso— que se vale de las siguientes observaciones:

- Dada una matríz $\mathbf{A}$ que permite la eg., el paso i-ésimo del algoritmo, que modifica la fila j $(j > i)$, es necesario sólo si $a_{ji} \neq 0$, ya que sino la operación $A_j \leftarrow A_j - \frac{a_{ji}}{a_{ii}} \cdot A_i$ es invariante.
- Similarmente, $A_j \leftarrow A_j - \frac{a_{ji}}{a_{ii}} \cdot A_i$, requiere operar sólo sobre los elementos no nulos de A_i.

```
proc eliminacion_gaussiana(inout A: matriz<n, n>, inout b: vector<n>) {
    i := 0
    while i < A.n - 1 {
        jt := iterador(A, i + 1, i)
        while jt.en_rango() {
            if (jt.j == i) { // solo si B.at(i + 1, i) != 0
                mij := jt.at() / A.at(i, i)
                sumar_fila(A, b, i, jt.i, mij)
            }
            jt.proximo_columna()
        }
        i := i + 1
    }
}
```

ALGORITMO 5. Pseudocódigo para *eliminacion_gaussiana*.

```
proc sumar_fila(inout A: matriz<n, n>, inout b: vector<n>,
                in f1, f2: natural, in c: real) {
    it := iterador(A, f1, f1)
    while it.en_rango() {
        val := B.at(f2, it.j) - c * it.at()
        B.set(f2, it.j, val)
        it.proximo_fila()
    }
    b[f2] := b[f2] - mij * b[f1]
}
```

ALGORITMO 6. Pseudocódigo para *sumar_fila*.

Para no complicar el análisis, podemos estimar que el algoritmo costará $O(n^2 \cdot k^2)$ en el caso promedio. El n^2 corresponde a la iteración de cada fila j sobre cada fila i, dado $0 \leq i < j < n$, y k corresponde a la cantidad de elementos no nulos en la matríz[8]. Dado $k \leq n$, el costo de peor caso será $O(n^4)$. Sin embargo, mientras $k < \sqrt{n}$ este método funcionará mejor que la forma tradicional de la eliminación (cuyo costo teórico es $\Theta(n^3)$).

2.1.4. *Sustitución inversa.* Al igual que en la eliminación gaussiana, aprovecharemos el hecho que $A_j \leftarrow A_j - \frac{a_{ji}}{a_{ii}} \cdot A_i$ requiere operar sólo sobre los elementos no nulos de A_i, para lograr un algorítmo en $\Theta(n \cdot k)$, con k la cantidad de elementos no nulos de la matríz:

[8]Se estima k^2 por SUMAR_FILA, que itera sobre los elementos no nulos de una fila e incluye una operación de SET.

```
proc sustitucion_inversa(in A: matriz<n, n>, in b: vector<n>,
                         out x: vector<n>) {
    x := vector(A.n, 0) // x = 0 de dimension n
    i := A.n - 1
    while i >= 0 {
        s  := 0
        it := iterador(A, i, i + 1)
        while it.en_rango() {
            s := s + it.at() * x[it.j]
            it.proximo_fila()
        }
        x[i] := (b[i] - s) / A.at(i, i)
        i := i + 1
    }
}
```

Algoritmo 7. Pseudocódigo para *sustitucion_inversa*.

Nuevamente, el peor caso se dará cuando $k = n$.

2.1.5. *normalizar*. Por último, presentaremos un algóritmo estándar para normalizar un vector de tamaño n en $\Theta(n)$:

```
proc normalizar(inout x: vector<n>) {
    s := 0
    i := 0
    while i < largo(x) {
        s += x[i]
        i := i + 1
    }
    i := 0
    while i < largo(x) {
        x[i] = x[i] / s
        i := i + 1
    }
}
```

Algoritmo 8. Pseudocódigo para *normalizar*.

2.2. **Análisis cuantitativo.** Se procederá a evaluar una implementación de PageRank en C++ acorde a los algoritmos propuestos. Se realizarán los siguientes experimentos:

1) Se medirá el tiempo de ejecución y el error relativo: $|Ax' - x'|$ a través del resultado promedio de cien experimentos sobre una web de mil páginas en la que se variará la cantidad de links entre 0 (sin links) y $1000^2 - 1000$ (toda página conecta con toda

otra) —Los mismos se seleccionarán de manera aleatoria en cada experimento— y el valor p también será aleatorio.

2) Se evaluará el error absoluto y relativo de los resultados de PageRank respecto a los tests provistos por la cátedra.

3. Resultados y Discusión

4. Conclusiones

5. Apéndice

5.1. **A:** $A = pWD + ez^t$.

demostración. Recordemos que:

$$e_i = 1$$

$$z_j = \begin{cases} (1-p)/n & \text{si} \quad c_j \neq 0 \\ 1/n & \text{si no} \end{cases}$$

$$w_{ij} = \begin{cases} 1 & \text{si} \quad i \neq j \ \wedge \ j \xrightarrow{l} i \\ 0 & \text{si no} \end{cases}$$

$$d_{ij} = \begin{cases} 1/c_j & \text{si} \quad i = j \ \wedge \ c_j \neq 0 \\ 0 & \text{si no} \end{cases}$$

A partir de estas definiciones, vemos que, como **D** es diagonal, el producto a derecha **WD** escala cada columna w_j por el factor d_{jj}, tal que:

$$(\mathbf{WD})_{ij} = \begin{cases} w_{ij}/c_j & \text{si} \quad c_j \neq 0 \\ 0 & \text{si no} \end{cases}$$

Como p es un escalar, sigue entonces que:

$$(p\mathbf{WD})_{ij} = \begin{cases} p \cdot w_{ij}/c_j & \text{si} \quad c_j \neq 0 \\ 0 & \text{si no} \end{cases}$$

Además, $e \in \mathbb{R}^{n\times 1} \ \wedge \ z^t \in \mathbb{R}^{1\times n} \implies ez^t \in \mathbb{R}^{n\times n}$, y:

$$(ez^t)_{ij} := \sum_{k=1}^{1} e_{ik} \cdot z^t_{kj} = e_i \cdot z^t_j = 1 \cdot z^t_j = z_j$$

Por lo que:

$$(p\mathbf{WD} + ez^t)_{ij} = \begin{cases} p \cdot w_{ij}/c_j + z_j & \text{si} \quad c_j \neq 0 \\ z_j & \text{si no} \end{cases}$$

$$= \begin{cases} (1-p) \cdot \frac{1}{n} + p \cdot \frac{w_{ij}}{c_j} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si no} \end{cases}$$

pero:

$$a_{ij} := Pr(j \longrightarrow i) = \begin{cases} (1-p)\cdot\frac{1}{n} + p\cdot\frac{I_{ij}}{c_j} & \text{si} \quad c_j \neq 0 \\ \frac{1}{n} & \text{si no} \end{cases}$$

Como $I_{ij} = 1$ si y sólo si existe un hipervínculo de j a i, con $j \neq i$ —y nulo en caso contrario—, entonces $I_{ij} = w_{ij}$ y concluímos que $a_{ij} = (p\mathbf{WD} + ez^t)_{ij}$, $\forall i, j : 1 \ldots n$, lo que implica que:

$$\mathbf{A} = p\mathbf{WD} + ez^t$$

■

5.2. **B: $I - pWD$ permite la eliminación gaussiana.** Definimos la eliminación Gaussiana de la siguiente manera:

$$\mathbf{A} := \mathbf{I} - p\mathbf{WD}$$

$$\begin{aligned} \mathbf{Eg}_1(\mathbf{A}) &= \mathbf{A} \\ \mathbf{Eg}_{k+1}(\mathbf{A})_{ij} &= \begin{cases} \mathbf{Eg}_k(\mathbf{A})_{ij} & \text{si} \quad i < k+1 \ \vee \ j < k+1 \\ \mathbf{Eg}_k(\mathbf{A})_{ij} - \frac{\mathbf{Eg}_k(\mathbf{A})_{ik} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}} & \text{si no} \end{cases} \end{aligned}$$

$$HI : (\forall j \in \mathbb{N})(k \ \leq \ j \ \leq \ N) \longrightarrow_L |\mathbf{Eg}_k(\mathbf{A})_{jj}| \geq \sum_{i=k,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ij}|$$

$$CB : (k = 1)$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L |\mathbf{A}_{jj}| \geq \sum_{i=1,\ i\neq j}^{N} |\mathbf{A}_{ij}|$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L 1 \geq \sum_{i=1,\ i\neq j}^{N} |(1/C_j \vee 0)|$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L 1 \geq (|p/C_j| \vee 0) \cdot C_j$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L 1 \geq |p| > (|p/C_j| \vee 0) \cdot C_j$$

$$\text{sabemos} : \ 0 < p < 1 \rightarrow 1 \geq |p|$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L \ \text{True}$$

$$\text{QVQ:}(\forall j \in \mathbb{N})(k+1 \ \leq \ j \ \leq \ N) \longrightarrow_L |\mathbf{Eg}_{k+1}(\mathbf{A})_{jj}| \geq \sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_{k+1}(\mathbf{A})_{ij}|$$

$$\longrightarrow_L |\mathbf{Eg}_k(\mathbf{A})_{jj} - \frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq \sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ij} - \frac{\mathbf{Eg}_k(\mathbf{A})_{ik} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}|$$

$$\longrightarrow_L |\mathbf{Eg}_k(\mathbf{A})_{jj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq \sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ij}| + \sum_{i=k+1,\ i\neq j}^{N} |\frac{\mathbf{Eg}_k(\mathbf{A})_{ik} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}|$$

$$\sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ij}| = \sum_{i=k,\ i\neq j}^{N} (|\mathbf{Eg}_k(\mathbf{A})_{ij}|) - |\mathbf{Eg}_k(\mathbf{A})_{kj}|$$

$$\text{Por HI: } |\mathbf{Eg}_k(\mathbf{A})_{jj}| - |\mathbf{Eg}_k(\mathbf{A})_{kj}| \geq \sum_{i=k,\ i\neq j}^{N} (|\mathbf{Eg}_k(\mathbf{A})_{ij}|) - |\mathbf{Eg}_k(\mathbf{A})_{kj}|$$

$$|\mathbf{Eg}_k(\mathbf{A})_{jj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq |\mathbf{Eg}_k(\mathbf{A})_{jj}| - |\mathbf{Eg}_k(\mathbf{A})_{kj}| + \sum_{i=k+1,\ i\neq j}^{N} |\frac{\mathbf{Eg}_k(\mathbf{A})_{ik} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}|$$

$$|\mathbf{Eg}_k(\mathbf{A})_{kj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq \frac{|\mathbf{Eg}_k(\mathbf{A})_{kj}|}{|\mathbf{Eg}_k(\mathbf{A})_{kk}|} \sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ik}|$$

$$\sum_{i=k+1,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ik}| = \sum_{i=k+1}^{N} (|\mathbf{Eg}_k(\mathbf{A})_{ik}|) - |\mathbf{Eg}_k(\mathbf{A})_{jk}|$$

$$\text{Por HI: } |\mathbf{Eg}_k(\mathbf{A})_{kk}| - |\mathbf{Eg}_k(\mathbf{A})_{jk}| \geq \sum_{i=k+1}^{N} (|\mathbf{Eg}_k(\mathbf{A})_{ik}|) - |\mathbf{Eg}_k(\mathbf{A})_{jk}|$$

$$|\mathbf{Eg}_k(\mathbf{A})_{kj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq \frac{|\mathbf{Eg}_k(\mathbf{A})_{kj}|}{|\mathbf{Eg}_k(\mathbf{A})_{kk}|}(|\mathbf{Eg}_k(\mathbf{A})_{kk}| - |\mathbf{Eg}_k(\mathbf{A})_{jk}|)$$

$$|\mathbf{Eg}_k(\mathbf{A})_{kj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}| \geq |\mathbf{Eg}_k(\mathbf{A})_{kj}| - |\frac{\mathbf{Eg}_k(\mathbf{A})_{jk} \ \cdot \ \mathbf{Eg}_k(\mathbf{A})_{kj}}{\mathbf{Eg}_k(\mathbf{A})_{kk}}|$$

$$(\forall j \in \mathbb{N})(1 \ \leq \ j \ \leq \ N) \longrightarrow_L \ \text{True}$$

QED que $(\forall j \in \mathbb{N})(k \ \leq \ j \ \leq \ N) \longrightarrow_L |\mathbf{Eg}_k(\mathbf{A})_{jj}| \geq \sum_{i=k,\ i\neq j}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ij}|$
Por ende el elemento $|\mathbf{Eg}_k(\mathbf{A})_{kk}| = 0 \leftrightarrow \sum_{i=k+1}^{N} |\mathbf{Eg}_k(\mathbf{A})_{ik}| = 0$ O sea que el elemento de la diagonal en cada paso de la eliminacion gaussiana es 0 si y solo si todos los elementos que tiene por debajo en su columna son 0 o sea que se podria saltear ese paso

5.3. **C: Estructuras alternativas.**

Referencias

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems, 30 (1-7):107–117*, 1998.

[2] Kurt Bryan and Tanya Leise. The $25,000,000,000 eigenvector: The linear algebra behind google. *T SIAM review, 48 (3):569–581*, 2006.

[3] Amy N Langville and Carl D Meyer. Deeper inside pagerank. *Internet Mathematics, 1(3):335–380*, 2004.

[4] Sepandar D Kamvar; Taher H Haveliwala; Christopher D Manning and Gene H Golub. Extrapolation methods for accelerating pagerank computations. *In Proceedings of the 12th international conference on World Wide Web, pages 261-270. ACM*, 2003.