DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

16 de abril de 2018 — Métodos numéricos

Integrante	LU	Correo electrónico
Luis Arroyo	913/13	luis.arroyo.90@gmail.com
Carlos Humpiri	942/14	jhumpiri1599@gmail.com
Matías Millassón	131/13	matiasmillasson@gmail.com
Eric Peker	548/13	epeker.135@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

En este trabajo práctico vamos a hacer una versión simplificada de PageRank. Este algoritmo, como su nombre lo indica, sirve para ordenar un conjunto de páginas web según la probabilidad de que un usuario las visite, o lo que es lo mismo, cuan importante es. PageRank consiste en resolver el siguiente sistema de ecuaciones:
$Ax = x$ dónde $A \in \mathbb{R}^{nxn}$, $x \in \mathbb{R}^n$ y n es la cantidad de páginas.
La matriz A tiene en el elemento a_{ij} la probabilidad de que un usuario navegue desde la página i hacia la página j. Se espera que la matriz A sea rala por la naturaleza del problema. Para resolver el sistema de manera eficiente propusimos una representación para las matrices aprovechando la gran cantidad de ceros que tienen, basada en un vector de diccionarios. Por último experimentamos sobre el tiempo de ejecución variando los parámetros de entrada.

Palabras claves: matriz rala, matriz diagonal dominante, PageRank

2. Desarrollo

2.1. Resolución del problema

Para explicar como resolvimos el sistema $Ax = x$ antes debemos comentar formalmente cómo está definida esa matriz.
Primero definimos la matriz de conectividad W de la siguiente manera:

$$W_{ij} = \begin{cases} 1 & \text{si la página i tiene un link saliente a la página j} \\ 0 & \text{si no} \end{cases} \tag{1}$$

Así, se espera que la matriz W tenga muchos elementos cuyo valor sea cero ya que no es común que un sitio posea muchos vínculos hacia otros como así tampoco es normal que haya un sitio tal que todos los demás lo apunten.
Para una página se define su grado como la cantidad de enlaces que salen de ella:

$$c_j = \sum_{i=1}^{n} w_{ij} \tag{2}$$

Ahora definimos la importancia de un sitio de la siguiente forma:

$$x_i = \sum_{j=1}^{n} \frac{x_j}{c_j} w_{ij} \tag{3}$$

Luego definimos la matriz de puntajes $R = WD$ dónde D es la matriz diagonal con los inversos de los grados en caso de que sean distintos de cero. En caso de que no haya enlaces para esa página el valor va a ser cero.

Pero si queremos hacer un modelo más fiel a la realidad no nos podemos quedar con esa matriz tan naïf ya que los usuarios de las páginas web no las recorren siguiendo siempre los links sino que pueden pasar a otras páginas por más que no estén relacionadas con la actual.

Por esto mismo vamos a involucrar la probabilidad, llamésmosla p, de que un usuario no siga por un link hacia otro sitio sino que salte a una no linkeada. Entonces tenemos tres casos:

- El usuario sigue un link (probabilidad = $\frac{1}{c_j}$).
- El usuario elije una página cualquiera a pesar de tener links para continuar (probabilidad = $\frac{1}{n}$).

- El sitio carece de enlaces con lo cual elije el siguiente al azar (probabilidad = $\frac{1}{n}$).

Dadas estas condiciones podemos juntar todos los casos en la siguiente matriz que vamos a denominar A:

$$a_{ij} = \begin{cases} \frac{1-p}{n} + \frac{p*w_{ij}}{c_j} & \text{si c}_\text{j} \neq 0 \\ \frac{1}{n} & \text{si no} \end{cases}$$

Como queremos ver la importancia de cada página x_i el sistema de ecuaciones queda:

$$Ax = x \tag{4}$$

Pero para resolver este sistema de manera eficiente vamos a descomponer la matriz A de la siguiente manera:

$$A = pWD + ez^t \tag{5}$$

D es una matriz diagonal,

$$D_{jj} = \begin{cases} \frac{1}{c_j} & \text{si c}_\text{j} \neq 0 \\ 0 & \text{si no} \end{cases}$$

$e \in \mathbb{R}^n, e_i = 1 \forall i \in [1, n]$ y $z \in \mathbb{R}$

$$z_j = \begin{cases} \frac{1-p}{n} & \text{si c}_\text{j} \neq 0 \\ \frac{1}{n} & \text{si no} \end{cases}$$

Reemplazando 5 en 4 nos queda:
$(pWD + ez^t)x = x \Leftrightarrow$
$(I - pWD)x = ez^t x$
Por simplicidad se tomó $z^t x = 1$, con lo que nos queda la siguiente ecuación:

$$(I - pWD)x = e \tag{6}$$

La ecuación 6 es un sistema de ecuaciones y para resolverlo vamos a usar el algoritmo clásico, es decir **eliminación gaussiana**. En particular vamos a descomponer la matriz $I - pWD$ en una matriz triangular inferior L y una matriz triangular superior U. Pero para no realizar pivoteo parcial es condición necesaria que la matriz $I - pWD$ sea estrictamente diagonal dominante por columnas[1].

2.2. Demostraciones de las afirmaciones realizadas

2.2.1. $A = pWD + ez^t$

Ahora se va a probar que la ecuación 5 es válida Tomemos un elemento de la matriz, vale la siguiente igualdad

$$a_{ij} = p \sum_{r=1}^{n} w_{ir} d_{rj} + z_j$$

Separemos en dos casos, cuando $c_j = 0$ y cuando no. Reemplazando cada parte por su definición obtenemos:

- $c_j = 0$:
  $\frac{1}{n} = p \sum_{r=1}^{n} w_{ir} 0 + \frac{1}{n} \Leftrightarrow$
  $\frac{1}{n} = p \sum_{r=1}^{n} 0 + \frac{1}{n} \Leftrightarrow$
  $\frac{1}{n} = 0 + \frac{1}{n} \Leftrightarrow$
  $\frac{1}{n} = \frac{1}{n}$

- $c_j \neq 0$:
  $\frac{1-p}{n} + \frac{p*w_{ij}}{c_j} = p\sum_{r=1}^{n} w_{ir}d_{rj} + \frac{1-p}{n} \Leftrightarrow$
  $\frac{p*w_{ij}}{c_j} = p\sum_{r=1}^{n} w_{ir}d_{rj} \Leftrightarrow$
  D al ser una matriz diagonal solo tiene un valor no nulo en la diagonal por lo que:
  $\sum_{r=1}^{n} w_{ir}d_{rj} = w_{ij} * \frac{1}{c_j}$
  Reemplazando en la ecuación anterior se verifica la igualdad trivialmente. □

2.2.2. Demostración de que la matriz es diagonal dominante

A es estrictamente diagonal dominante por columna si cumple la siguiente propiedad:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ji} \tag{7}$$

Sea nuestra matriz $A = I$ - pWD vamos a demostrar que es estrictamente diagonal dominante por columna.

Sea A la siguiente matriz:

$$A = I - pWD = \begin{pmatrix} 1 & -pd_{22}w_{12} & \cdots & -pd_{nn}w_{1n} \\ -pd_{11}w_{21} & 1 & \cdots & -pd_{nn}w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -pd_{11}w_{n1} & -pd_{22}w_{n2} & \cdots & 1 \end{pmatrix}$$

Nosotros sabemos que $|a_{ii}| = 1 \forall i = 1, ..., n$

Tomemos una columna cualquiera de la matriz A:

$$Col(A)_i = \begin{pmatrix} -pd_{ii}w_{21} \\ \vdots \\ -pd_{ii}w_{i-1i} \\ 1 \\ -pd_{ii}w_{i+1i} \\ \vdots \\ -pd_{ii}w_{ni} \end{pmatrix}$$

Voy a probar lo siguiente:

$$1 > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ji}| \tag{8}$$

Primero reemplazamos a_{ji} por $-pd_{ii}w_{ji}$ y aplicamos propiedades de los módulos.

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ji} = \sum_{\substack{j=1 \\ j\neq i}}^{n} |-pd_{ii}w_{ji}| = \sum_{\substack{j=1 \\ j\neq i}}^{n} |p||d_{ii}||w_{ji}| = |p||d_{ii}| \sum_{\substack{j=1 \\ j\neq i}}^{n} |w_{ji}| \tag{9}$$

Acá dividimos en 2 casos según el grado del nodo i:

Si grado del nodo i es 0 entonces

$$c_i = 0 \Rightarrow d_{ii} = 0 \tag{10}$$

Reemplazando esto tenemos:

$$|p||d_{ii}|\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| = |p|0\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| = 0 \tag{11}$$

Y como $0 < 1$, cumple que A es estrictamente diagonal dominante por columna.

Si grado del nodo i es distinto de 0 entonces:

$$c_i = \sum_{k=1}^{n}|w_{ki}| \Rightarrow d_{ii} = \frac{1}{\sum_{k=1}^{n}|w_{ki}|} \tag{12}$$

Reemplazando esto tenemos:

$$|p||d_{ii}|\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| = |p|\frac{1}{\sum_{k=1}^{n}|w_{ki}|}\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| = |p|\frac{\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}|}{\sum_{k=1}^{n}|w_{ki}|} \tag{13}$$

Acordémonos que $w_{ii} = 0$ entonces tenemos:

$$|p|\frac{\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}|}{\sum_{k=1}^{n}|w_{ki}|} = |p|\frac{\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| + 0}{\sum_{k=1}^{n}|w_{ki}|} = |p|\frac{\sum_{\substack{j=1\\j\neq i}}^{n}|w_{ji}| + w_{ii}}{\sum_{k=1}^{n}|w_{ki}|} = |p|\frac{\sum_{j=1}^{n}|w_{ji}|}{\sum_{k=1}^{n}|w_{ki}|} \tag{14}$$

Pero $\frac{\sum_{j=1}^{n}|w_{ji}|}{\sum_{k=1}^{n}|w_{ki}|} = 1$, reemplazando esto tenemos:

$$|p|\frac{\sum_{j=1}^{n}|w_{ji}|}{\sum_{k=1}^{n}|w_{ki}|} = |p| \tag{15}$$

Y como p ∈ (0,1) entonces p < 1, por lo tanto A es estrictamente diagonal dominante por columna. □

2.2.3. Admisión de factorización LU

Si A es estrictamente diagonal dominante por columna entonces tiene factorización LU. Veamos la prueba
Como A es estrictamente diagonal dominante por columna entonces $|a_{11}| > \sum_{i=2}^{n}|a_{i1}|$ entonces $a_{11} > 0$, puedo hacer el primer paso de la eliminación gaussiana.

$$A^{(1)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

Donde usamos $F_i = F_i - \frac{a_{i1}}{a_{11}}F_i$, para $i = 2, ..., n$
Entonces los $a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}$, para $i = 2, ..., n$ y $j = 1, ..., n$

Ahora voy a probar que la matriz A_{22} es estrictamente diagonal dominante por columnas, donde la matriz A_{22} es:

$$A_{22} = \begin{pmatrix} a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

quiero probar que:

$$|a_{jj}^{(1)}| > \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ji}^{(1)}| \tag{16}$$

Vamos a reemplazar $a_{ij}^{(1)}$ por $a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}$

$$\sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ji}^{(1)}| = \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}| \tag{17}$$

Aplicamos propiedades de los módulos:

$$\sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}| < \sum_{\substack{i=2 \\ i\neq j}}^{n} \left(|a_{ij}| + |\frac{a_{i1}}{a_{11}} a_{1j}| \right) = \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ij}| + \frac{|a_{1j}|}{|a_{11}|} \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{i1}| \tag{18}$$

Ahora vamos acotar las dos sumatorias de la ecuación (17):

$$\sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ji}| < |a_{jj}| - |a_{1j}| \tag{19}$$

$$\sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{i1}| < |a_{11}| - |a_{j1}| \tag{20}$$

Reemplazando la ecuación (18) y (19) en la ecuación 17 tenemos:

$$\sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{ij}| + \frac{|a_{1j}|}{|a_{11}|} \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{i1}| < |a_{jj}| - |a_{1j}| + \frac{|a_{1j}|}{|a_{11}|} \sum_{\substack{i=2 \\ i\neq j}}^{n} |a_{i1}| < |a_{jj}| - |a_{1j}| + \frac{|a_{1j}|}{|a_{11}|} (|a_{11}| - |a_{j1}|) \tag{21}$$

Resolviendo tenemos:

$$|a_{jj}| - |a_{1j}| + \frac{|a_{1j}|}{|a_{11}|} (|a_{11}| - |a_{j1}|) = |a_{jj}| - \frac{|a_{1j}|}{|a_{11}|} |a_{j1}| \tag{22}$$

Aplicando propiedades de los módulos tenemos:

$$|a_{jj}| - \frac{|a_{1j}|}{|a_{11}|} |a_{j1}| \leq \left| a_{jj} - \frac{a_{1j}}{a_{11}} a_{j1} \right| = |a_{jj}^{(1)}| \tag{23}$$

Por lo tanto acabamos de demostrar que la matriz $A^{(k)}$, donde $k = 1, ..., n-1$ de lo que queda de aplicar la eliminación gaussiana son estrictamente diagonal dominante por columnas, esto quiere decir que nuestro $|a_{ii}^{(k)}|$ nunca va hacer cero, por lo tanto va tener factorización LU □.

¿Cómo se garantiza la aplicabilidad de EG?
Como la matriz $A = I$ - pWD es estrictamente diagonal dominante por columna tiene factorización LU esto quiere decir que nunca nuestro $|a_{ii}^{(k)}|$ de la eliminación gaussiana va a ser 0, por lo tanto podemos garantizar que su aplicabilidad nunca se va a interrumpir.

¿La matriz (I - pWD) esta bien Condicionada?
Si la matriz $A = I$ - pWD estuviera mal condicionada entonces para poder mejorar su numero de condición lo que hacemos es aplicar el pivote parcial al algoritmo de eliminación gaussiana, pero como nuestra matriz $A = I$ - pWD es estrictamente diagonal dominante por columna, el pivoteo parcial nunca se va aplicar a nuestro $a_{ii}^{(k)}$ ya que es estrictamente mayor que $a_{ji}^{(k)}$ donde $j = i + 1...n$, por lo tanto nuestra matriz $A = I$ - pWD esta bien condicionada.

2.3. Implementación de la solución

2.3.1. Representación de las matrices

Las matrices utilizadas en el trabajo práctico son ralas, es decir, poseen una gran cantidad de ceros como dato. Por esto, decidimos implementar una estructura que guarde la matriz en memoria de manera eficiente, sin ocupar espacio de más. La idea detrás de esta implementación es guardar únicamente los valores no nulos e inferir que, si un valor no está definido en la estructura, debe ser cero. Considerando una matriz de tamaño *n*n*, la estructura elegida se basa en tener un **vector** de tamaño n. El elemento i del vector representa la fila i y cada uno de estos estará populado por un **diccionario** (implementado por la clase *map* de c++).
Veamos algunos ejemplos de nuestra representación:

Matriz	**Representación**
$W = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	{ [0] = diccionario vacío,[1] =diccionario(<0,1>)}
$Y = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$	{[0] = diccionario(<1,1>), [1] =diccionario(<0,1>),[2] = diccionario(<1,1>) }

2.3.2. Funciones implementadas

2.3.2.1 Generación de la matriz D

Ya que la matriz D es diagonal se decidió que lo mejor era representarla como un vector de floats, dónde el elemento i representaba al elemento d_{ii}. Sencillamente se recorrió la matriz por columnas y se realizó la sumatoria.

```
1: procedure suma_columnas(Matriz W)
2:     solucion
3:     for j ∈ [0..cantidad de columnas de la matriz) do
4:         acumulador ← 0
5:         for i ∈ [0..cantidad de filas de la matriz) do
6:             valorPosicion_ij ←Buscar en el diccionario filas[i] la clave j
7:             if valorPosicion_ij no es nulo then                ▷ Si W_ij no es cero
8:                 acumulador ← acumulador + valorPosicion_ij
9:         if acumulador ≠ 0 then
10:            Agregar al final de solucion el valor acumulador^-1
11:        else
12:            Agregar al final de solucion el valor 0
13:    return solucion
```

2.3.2.2 Multiplicación de matrices

Para realizar la multiplicación de matrices se siguió el algoritmo clásico, en el cual se hace el producto interno de la fila i-ésima de la matriz de la izquierda con la columna j-ésima de la matriz de la derecha para obtener el elemento correspondiente a la fila i y columna j del producto. Pero también es importante comentar que se aprovechó que la matriz de la derecha es diagonal, buscando solamente el primer elemento en la primer fila, el segundo en la segunda fila, etc. Se modifica la matriz izquierda.

```
1: procedure multiplicacion(Matriz W, vector D)
```

```
2:      for j ∈ [0..cantidad de filas de la matriz W) do                ▷ Recorro las filas
3:          for i ∈ [0..cantidad de columnas de la matriz) do
4:              valorPosicion_ij ←Buscar en el diccionario filas[i] la clave j
5:              if valorPosicion_ij no es nulo then                     ▷ Si W_ij no es cero
6:                  if D[j] ≠ 0 then
7:                      valor(filas[i], j) ← valor(filas[i], j) ∗ D[j]
8:                  else
9:                      Eliminar la clave j del diccionario filas[i]
10:         if acumulador ≠ 0 then
11:             Agregar al final de solucion el valor acumulador^−1
12:         else
13:             Agregar al final de solucion el valor 0
```

2.3.2.3 Multiplicación por un escalar

Este algoritmo se basa en recorrer todos los elementos de cada diccionario de cada vector para cambiarles el valor. Notar que se modifica la matriz.

```
1: procedure multiplicacion_escalar(Matriz W, entero n)
2:     for i ∈ [0..cantidad de filas de la matriz W) do
3:         for j ∈ claves(filas[i])) do
4:             valor(filas[i], j) ← valor(filas[i], j) ∗ n
```

2.3.2.4 Restar la matriz identidad

En este algoritmo, al igual que en la multiplicación por una matriz diagonal se usa fuertemente que los únicos valores no nulos de la matriz I son los de la diagonal. Notar que se modifica la matriz W

```
1: procedure restar_identidad(Matriz W)
2:     for i ∈ [0..cantidad de filas de la matriz W) do
3:         if i ∈ claves(filas[i]) then
4:             valor(filas[i], i) ← valor(filas[i], i) − 1
5:         else
6:             valor(filas[i], i) ← −1
```

2.3.2.5 Descomposición LU

Se modifica la primer matriz pasada por parámetro para que sea la matriz triangular inferior. Se modifica la matriz segunda matriz pasada por parámetro para que sea la matriz triangular superior. En la línea 6 se escribió de manera compacta que se multiplica toda la fila por el escalar y luego se la resta otra fila coeficiente a coeficiente.

```
1: procedure eliminacion_gausiana(Matriz W, Matriz L)
2:     for j ∈ [1..cantidad de filas de la matriz W) do
3:         for i ∈ [j..cantidad de columnas de la matriz W) do
4:             if W[i][j] != 0 then
5:                 L[i][j] ← W[i][j]/W[j][j]
6:             W[i] ← W[j] ∗ W[i][j]/W[j][j] − W[i]
```

2.3.2.6 Resolución de sistema Ly=b

Para resolver el sistema de ecuaciones con una matriz triangular inferior comienzo por la primer fila ya que tiene una sola incógnita, lo agrego al vector solución y lo uso para resolver la ecuación de la siguiente fila. Se asume que el vector b es una columna de unos, por eso no es parámetro de la función. Devuelve el vector x.

```
1: procedure solucion_lower(Matriz L)
2:     solucion
3:     solucion[0] ← L[0][0]
4:     for i ∈ [2..cantidad de filas de la matriz L) do
5:         acumulador ← 0
6:         for j ∈ [2..longitud solución) do
7:             acumulador ← acumulador + L[i − 1][j − 1] ∗ solucion[j − 1]
8:         solucion[i] ← 1 − acumulador
9:     return solucion
```

2.3.2.7 Resolución de sistema Ux=y

Este algoritmo es muy similar al anterior pero la matriz es triangular superior y l vector b es pasado por parámetro. Otra diferencia es que al ser U una matriz superior se comienza a resolver la última ecuación. Esto hace que en el vector solución se guarden los valores al revés, por lo que lo invertimos antes de retornarlo.

```
1: procedure solucion_upper(Matriz U, vector b)
2:     solución
3:     tamanio ← longitud(b)
4:     solución[0] ← b[tamanio − 1]/U[tamanio − 1][tamanio − 1]
5:     for i ∈ [2..cantidad de filas de la matrizU) recorriendose decrecientemente do
6:         acumulador ← 0
7:         for j ∈ [1..longitud solución) do
8:             acumulador ← acumulador + U[i − 1][i + j − 1] ∗ solucion[tamanio − j]
9:         solucion[tamanio − i] ← 1 − acumulador
10:    return reverso(solucion)
```

3. Experimentación

3.1. Análisis de relación tiempo tamaño dimensión

3.1.1. Introducción

En este experimento nos interesa analizar cual es la relación que existe entre el tamaño de la matriz, la cantidad de elementos que cada una posee, que esto se verá reflejado en la ralidad o sea en la densidad de elementos que tendrá la matriz, y el tiempo que tarda en ejecutarse. Nos interesa realizar este experimento ya que optamos por utilizar una representación en la que no se guarden los ceros para optimizar la eliminación gaussiana.

3.1.2. Metodología de la experimentación

Nuestros experimento consta de matrices de tamaño entre 50 y 1500 aumentando sucesivamente cada matriz de a 50 elementos, así también para cada matriz vamos a evaluarla en 4 instancias de ralidad, para 5 %, 10 % , 20 % y 30 %. Creímos que no es interesante aumentar la densidad ya que, nuestro problema está basado en matrices ralas y consideramos que una matriz que tenga el 40 % o mas de sus elementos no se la puede considerar una matriz rala.

La medición del tiempo se realizará con la biblioteca ctime , en donde evaluamos la función que calcula el ranking completo no solo a eliminación gaussiana

Para cada instancia de tamaño y de densidad, corremos el experimento 10 veces y tomamos la media de estos, para reducir posibles errores debido a outliers.

3.1.3. Resultados

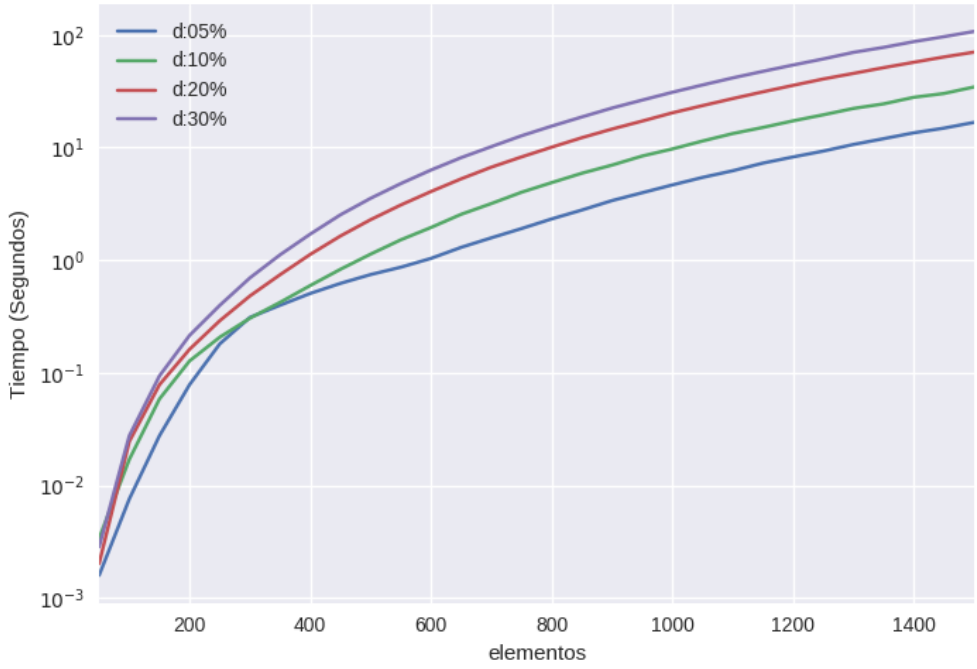


Figura 1: Relación tamaño densidad y tiempo

Decidimos usar una gráfica en escala logarítmica ya que los elementos muy pequeños se veían aplastados con la otra escala y esto nos permite ver dos fundamentales, la primera es que el tiempo que tarda nuestro algoritmo se ve afectado directamente por la cantidad de elementos no así tanto por el tamaño, ya que para cada instancia de tamaño de la matriz observando la densidad de cada matriz aumenta significativamente el tiempo entre cada una instancia de ellas. Por lo que nos dice que estamos optimizando el tiempo ya que no evaluamos instancias que son ceros, cosa que si hubiéramos optado por elegir una estructura en donde guardáramos dichos valores, en cada paso de la eliminación gaussiana deberíamos evaluarlos y operar con estos ceros por lo que nos quedaría una gráfica en donde la densidad no afectara ya que el algoritmo seria $\theta(n^3)$

Por otro lado, vemos que con la densidad de 5 % y %10 a los 300 elementos, en tiempo se igualan pero luego continúan con una diferencia clara. Hemos mirado los resultados de la salida de nuestro experimento y comprobamos que, en este punto sucedió que ambas densidades tuvieron aproximadamente la misma cantidad de elementos en particular la que debería ser del 5 % tenia casi la misma cantidad de elementos que el de 10 % por lo que se puede decir que fue un outlier a la hora de crear la matriz de manera aleatoria.

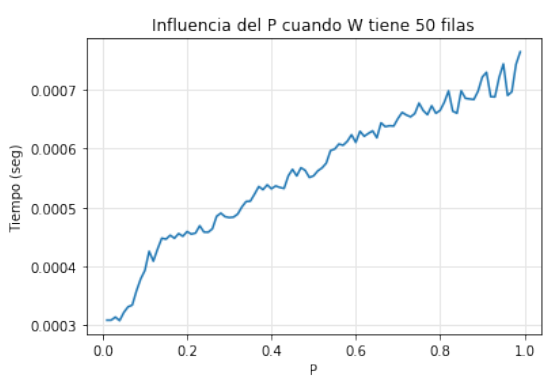
3.1.4. Resumen:

Podemos decir que el uso de una estructura especial para las matrices ralas da buen resultado, ya que el tiempo que tarda el algoritmo está ligado directamente con la cantidad de elementos distintos de ceros y no con el tamaño de la matriz, de todas formas el algoritmo seguirá siendo $\theta(n^3)$ pero el hecho de que trabajamos con matrices ralas, reduce mucho el tiempo de procesamiento.

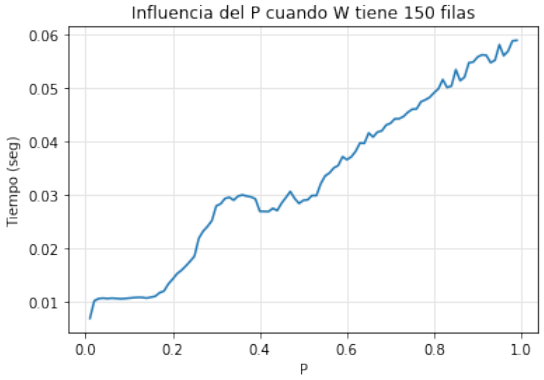
3.2. Variación del P

Para ver el impacto de la probabilidad de que un usuario continúe en otra página decidimos experimentar variando el el P en distintos grafos. La matriz W tenía muy pocos elementos distintos de cero (5 % para ser exactos). La decisión de tomar matrices muy ralas fue para que no influyera la cantidad de elementos y, entonces, se pueda apreciar correctamente la importancia de la probabilidad.

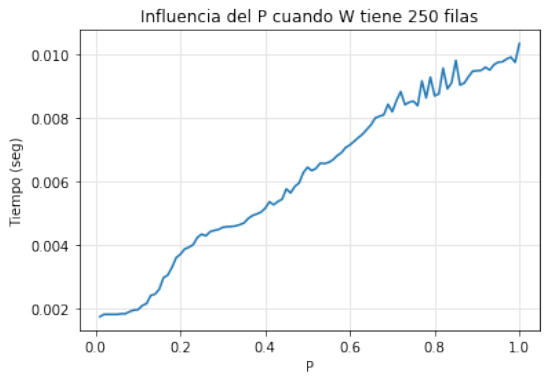
Hipótesis: cuando el p es muy chico el tiempo de ejecución es menor que cuando está más cerca de uno.



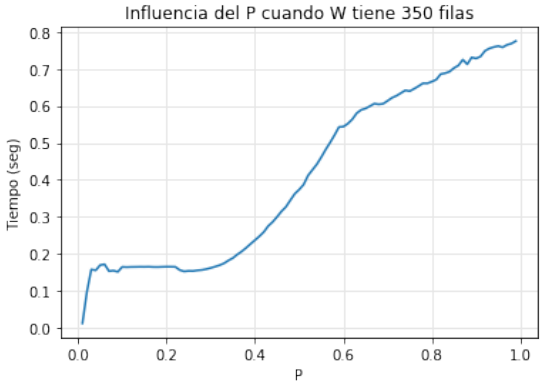
(a) Resultados del experimento donde varía p, la matriz de conectividad tiene 50 filas.



(b) Resultados del experimento donde varía p, la matriz de conectividad tiene 150 filas.



(a) Resultados del experimento donde varía p, la matriz de conectividad tiene 250 filas.



(b) Resultados del experimento donde varía p, la matriz de conectividad tiene 350 filas.

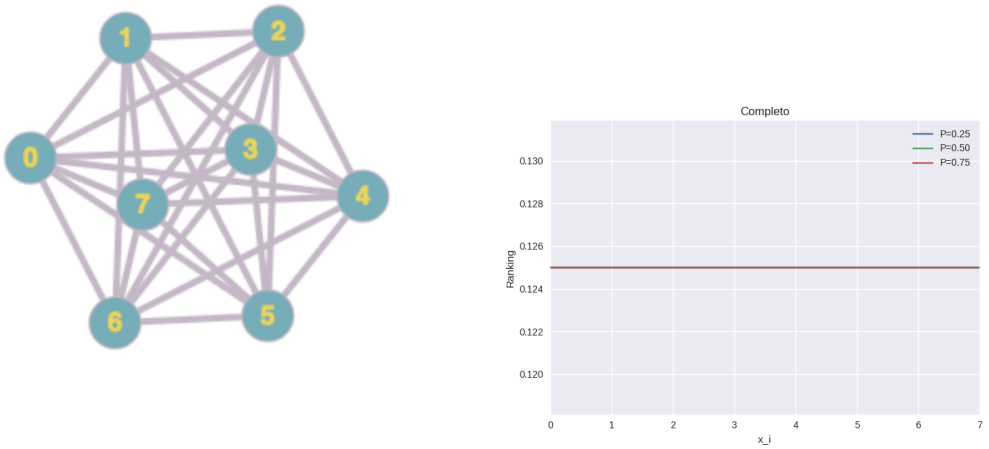
Como se ve en los gráficos la hipótesis se verifica. Creemos que una posible causa es que si se multiplica cada coeficiente de la matriz WD por p y éste es muy pequeño puede hacer que el valor resultante sea menor que el epsilon utilizado para aproximar cero.
También podemos notar que en la matriz de 50 filas y en la de 250 se disparó el tiempo de ejecución cuando el p fue de aproximadamente 0.1, mientras que para el caso de 150 filas y el de 350 el crecimiento rápido comenzó con 0.3. Podemos concluir que a partir de $p = 0{,}4$ es muy probable que la gráfica esté creciendo con una gran pendiente y que en $p = 0{,}5$ tendremos un tiempo de ejecución del doble o del triple a comparación del tiempo que tardó con $p = 0{,}1$.

3.3. Análisis cualitativo

Para realizar el análisis cualitativo decidimos estudiar los siguientes grafos:

- Grafo completo.
- Grafo completo con el agregado de un camino simple.
- Árbol de altura dos.
- Grafo en el que existe un nodo tal que casi todos apuntan a él.
- Grafo no conexo, con dos componentes conexas muy similares.

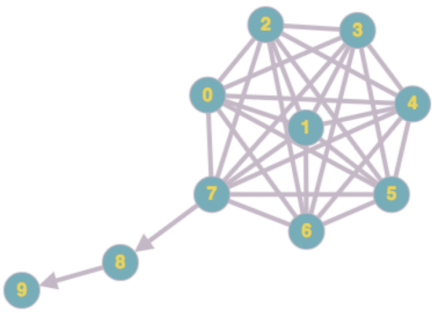
3.3.1. Grafo completo



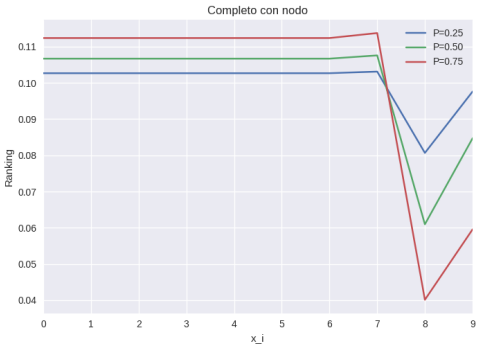
(a) Representación gráfica del grafo completo (b) Resultados de PageRank para el grafo completo

En este caso podemos ver que el ranking de todos las páginas es equivalente. Esto tiene sentido, ya que todas las páginas están conectadas entre si, por lo que no debería existir una con más peso que la otra. Además, que el valor de p sea más chico o más grande no modifica el ranking. Tener más chance de continuar por los links de una propia página o saltar hacia otra de las restantes no modifica el ranking si todos los nodos son equivalentes entre sí.

3.3.2. Grafo completo con el agregado de un camino simple



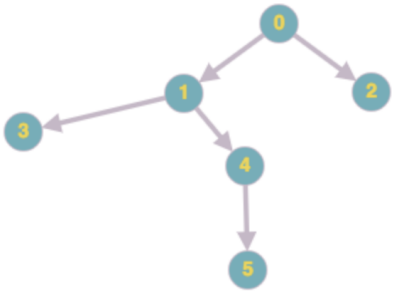
(a) Representación gráfica del grafo completo con dos nodos agregados



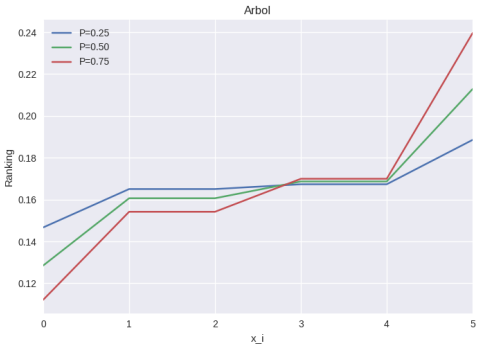
(b) Resultados de PageRank para el grafo completo con dos nodos agregados

En este caso todas las páginas tienen links a las otras, excepto por dos: una a la que apunta sólo una de las páginas más visitadas (8), y otra apuntada por esta última (9). Se nota que estas dos últimas páginas tienen menos puntaje debido a la poca incidencia de links en estos.

3.3.3. Árbol de altura dos



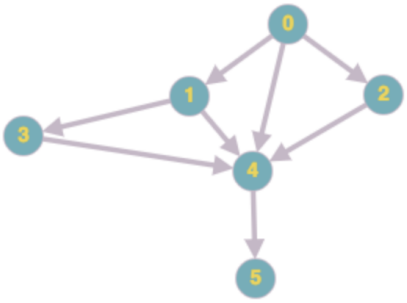
(a) Representación gráfica del árbol estudiado



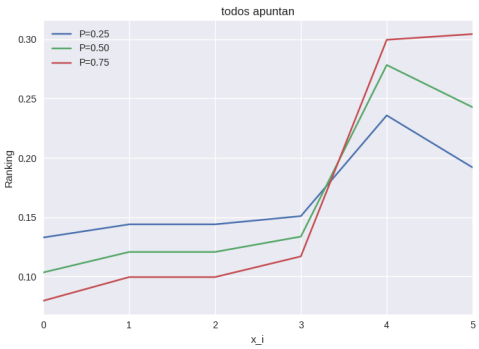
(b) Resultados de PageRank para el árbol

En este caso, las páginas están organizadas de forma tal que se genera una jerarquía: las páginas "padres.ªpuntan a páginas "hijas", pero estas últimas no apuntan a las páginas que las referencian. Además, cada página tiene sólo un padre. De esta manera, a medida que vamos recorriendo el árbol desde la raíz hasta las hojas, el ranking va aumentando por página; cada una va acumulando el peso de su "padreż de las páginas que apuntan a este último, De esta forma, al no tener links que vuelvan hacia .ªtrás", las hojas del árbol serán las que más puntaje tendrán.

3.3.4. Grafo en el que existe un nodo tal que casi todos apuntan a él



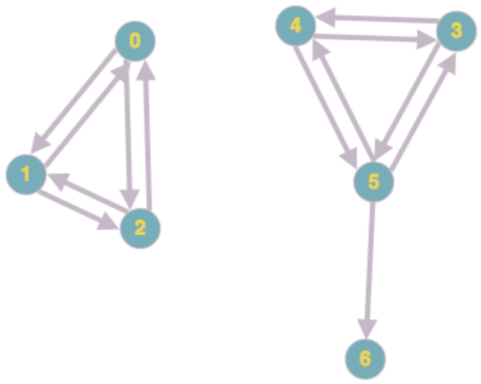
(a) Representación gráfica del grafo con un vértice con todos apuntando a él



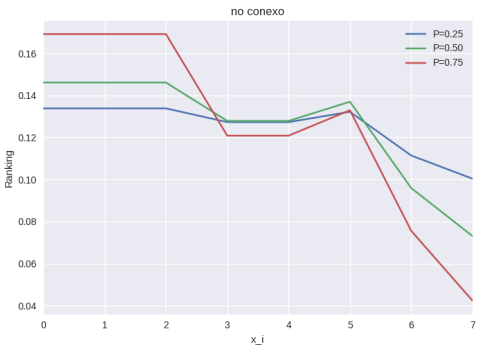
(b) Resultados de PageRank para el grafo con un vértice con todos apuntando a él

Este caso es parecido al último, en el sentido de que a medida que vamos recorriendo.[el] grafo, es decir, recorriendo las páginas, no tenemos forma de volver a una página anteriormente visitada. La diferencia radica en que en el caso de una página, la (4), muchas otras páginas desembocan en ella. Eso se refleja en su puntaje, el cual es mucho más grande que el de las otras, debido a su gran incidencia de links. Además, esta página sólo apunta a una, la (5). Esta no alcanza el puntaje de la (4) ya que sólamente es apuntada por esta (tiene poca incidencia de links). Sin embargo, esta página tiene más puntaje que las anteriores a (4) ya que está siendo apuntada por una página de mucho peso.

3.3.5. Grafo no conexo, con dos componentes conexas muy similares



(a) Representación gráfica del grafo no conexo estudiado



(b) Resultados de PageRank para el grafo no conexo

Este caso esta diseñado para probar la independencia entre conjuntos de páginas cuando éstos no se están apuntando mutuamente. Tenemos dos grupos separados, uno de páginas que se apuntan todas entre sí (0, 1, 2) y otras que poseen otra estructura en la que todas se apuntan entre sí menos una página es sólo apuntada por otra (3, 4, 5, 6). Para el primer grupo, se observa que el puntaje es igual y consistente con el hecho de que todos las páginas poseen el mismo peso. En cambio, las páginas del segundo grupo muestran variaciones que muestran que algunas páginas tienen más peso que otras, debido a este desbalance de links. De esta manera podemos apreciar que, al no

haber ningún link compartido entre ambos grupos, no se influencia el cálculo del puntaje entre ellos.

3.3.6. Análisis de p

Para todos estos casos definidos se observa que, cuando el p es mayor, el puntaje de las páginas con más links sube, mientras que las de menor puntaje bajan todavía más. Eso es lógico, ya que al aumentar la probabilidad de que se sigan los links definidos dentro de cada página y no se salte a otra de manera random, hay menor chance de llegar a las páginas apuntadas por menor cantidad de links. Es decir, las páginas con menor puntaje son favorecidas por el hecho de que el usuario salte a ellas de manera random, porque cuesta llegar a ella atravesando links definidos por otras.

3.3.7. Calidad del algoritmo

El hecho de que los rankings obtenidos hayan coincidido bastante bien con la intuición de cuanto peso tendría cada página de acuerdo a los links que la apuntaban nos hace pensar que *PageRank* es un buen algorítmo para calcular los puntajes.

4. Conclusiones

Un aspecto complejo del problema fue buscar que propiedades cumple la matriz $I - pWD$, para poder simplificar los algoritmos utilizados para la resolución del problema (Ej: multiplicación de matrices, eliminación gausiana, búsqueda de la solución por medio de factorización LU, etc). Otro aspecto interesante fue buscar las estructuras necesarias para implementar matrices ralas, para poder simplificar el código de la resolución al problema, una vez detectado la estructura mas eficiente vimos que su impacto se vio reflejado al minimizar la complejidad algorítmica de la solución, lo cual nos fue útil para poder realizar casos de experimentación con grafos bastante grandes con distintas cantidades de aristas. También pudimos observar que cualitativamente nuestro modelo represente de manera bastante fiel la realidad, ya que los rankings realizados fueron como esperabamos.

5. Anexo

5.1. Enunciado

Métodos Numéricos
1^o Cuatrimestre 2018
Trabajo Práctico 1: PageRank

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Sistemas de ecuaciones lineales

Introducción

El motor del buscador de Google, desde hace más de 20 años, utiliza el denominado *ranking de Page* o *PageRank* [1] como uno de los criterios para ponderar la importancia de los resultados de cada búsqueda. Calcular este ranking requiere simplemente resolver un sistema de ecuaciones lineales donde la cantidad de ecuaciones e incógnitas del sistema es igual al número de páginas consideradas. ¿Simplemente?

Modelado del problema

Para un determinado conjunto de n páginas web definamos la *matriz de conectividad* $\mathbf{W}$ poniendo $w_{ij} = 1$ si la página j tiene un link a la página i y $w_{ij} = 0$ si no. Además $w_{ii} = 0$ pues ignoramos los *autolinks*. De esta forma, la matriz $\mathbf{W}$ puede resultar extremadamente rala y muy grande de acuerdo al tamaño del conjunto.

Para cada página $j = 1 \dots n$, definimos su *grado* como:

$$c_j = \sum_{i=1}^{n} w_{ij} \tag{1}$$

Es decir, la cantidad de links *salientes* de j. Típicamente c_j es un número mucho menor que n.

Se busca que el ranking sea mayor en las páginas *importantes*. Heurísticamente, una página es importante cuando recibe muchos "votos" de otras páginas, es decir, links. Pero no todos los links pesan igual: los links de páginas más importantes valen más. Pocos links de páginas importantes pueden valer más que muchos links de páginas poco importantes. Y los links de páginas con muchos links valen poco. Por lo tanto, una forma de calcular la importancia o *puntaje* x_i de la página i es:

$$x_i = \sum_{j=1}^{n} \frac{x_j}{c_j} w_{ij} \tag{2}$$

Es decir, la página j le aporta a i su puntaje ponderado por cuántos links salientes tiene. También, se puede definir la matriz de puntajes $\mathbf{R} = \mathbf{WD}$, donde $\mathbf{D}$ es una matriz diagonal con elementos d_{jj} de la forma:

$$d_{jj} = \begin{cases} 1/c_j & \text{si} \quad c_j \neq 0 \\ 0 & \text{si} \quad c_j = 0 \end{cases},$$

Lo cual nos permite calcular el ranking de todas las páginas como:

$$\mathbf{R}\,\mathbf{x} = \mathbf{x} \tag{3}$$

donde $\mathbf{x} = (x_1, \dots, x_i, \dots, x_n)^T$. Luego, la ecuación 2 corresponde al elemento i-ésimo $(\mathbf{R}\,\mathbf{x})_i$.

Este modelo tiene un problema y es que no logra capturar el comportamiento errático del usuario mientras surfea la red redes.

[1] Por Larry Page, uno de los fundadores de Google, otrora joven científico actualmente devenido multimillonario. Ver artículo original del 1998 con más de 16500 citas [1]

Modelo del navegante aleatorio

Un enfoque alternativo es considerar el modelo del *navegante aleatorio*. El navegante aleatorio empieza en una página cualquiera del conjunto, y luego en cada página j que visita elige con probabilidad $p \in (0,1)$ si va a seguir uno de sus links, o con probabilidad $1-p$, si va a pasar a otra página cualquiera del conjunto. Una vez tomada esa decisión, si decidió seguir un link de la página j elige uno al azar con probabilidad $1/c_j$, mientras que si decidió pasar a otra página cualquiera entonces elige una al azar con probabilidad $1/n$). Cuando la página j no tiene links salientes, es decir $c_j = 0$, elige al azar una página cualquiera del conjunto. Por lo tanto, se espera que luego de mucho surfear el navegante aleatorio va a estar en páginas importantes con mayor probabilidad.

Formalmente, la probabilidad de pasar de la página j a la página i es:

$$a_{ij} = \begin{cases} (1-p)/n + (p\ w_{ij})/c_j & \text{si} \quad c_j \neq 0 \\ 1/n & \text{si} \quad c_j = 0 \end{cases}, \tag{4}$$

y sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ a la matriz de elementos a_{ij}. Entonces el *ranking de Page* es la solución del sistema:

$$\mathbf{A}\ \mathbf{x} = \mathbf{x} \tag{5}$$

que cumple $x_i \geq 0$ y $\sum_i x_i = 1$.

Por lo tanto, el elemento i-ésimo $(\mathbf{Ax})_i$ es la probabilidad de encontrar al navegante aleatorio en la página i sabiendo que x_j es la probabilidad de encontrarlo en la página j, para $j = 1 \ldots n$;

Luego, la matriz $\mathbf{A}$ puede reescribirse como:

$$\mathbf{A} = p\ \mathbf{W}\ \mathbf{D} + \mathbf{e}\ \mathbf{z}^T,$$

donde $\mathbf{D}$ es una matriz diagonal de la forma

$$d_{jj} = \begin{cases} 1/c_j & \text{si} \quad c_j \neq 0 \\ 0 & \text{si} \quad c_j = 0 \end{cases},$$

$\mathbf{e}$ es un vector columna de unos de dimensión n y $\mathbf{z}$ es un vector columna cuyos componentes son:

$$z_j = \begin{cases} (1-p)/n & \text{si} \quad c_j \neq 0 \\ 1/n & \text{si} \quad c_j = 0 \end{cases}.$$

Así, la ecuación (5) puede reescribirse como

$$(\mathbf{I} - p\ \mathbf{W}\ \mathbf{D})\ \mathbf{x} = \gamma\ \mathbf{e}, \tag{6}$$

donde $\gamma = \mathbf{z}^T\ \mathbf{x}$ funciona como un factor de escala.

Cálculo del ranking de Page

De esta manera, un procedimiento para calcular el ranking de Page consiste en:

1. Suponer $\gamma = 1$.
2. Resolver el sistema lineal de la ecuación (6).
3. Normalizar el vector $\mathbf{x}$ de manera que $\sum_i x_i = 1$.

Enunciado

Se debe implementar un programa en `C` o `C++` que realice el cálculo del ranking de Page según el procedimiento descripto anteriormente.

Como parte **obligatoria** se pide implementar lo siguiente:

1. El método de Eliminación Gaussiana (EG)
2. Una estructura de matrices ralas que sea eficiente en espacio y en tiempo para la tarea que se busca realizar.
3. Herramientas de entrada/salida para la lectura de archivos con los datos de los conjuntos de páginas y escritura del ranking de Page. Se debe respetar el formato que se describe más abajo.

Previamente, **deberán** estudiar las características de la matriz involucrada y responder a lo siguiente:

1. ¿Por qué la matriz $\mathbf{A}$ definida en (4) es equivalente a $p\ \mathbf{W}\ \mathbf{D} + \mathbf{e}\ \mathbf{z}^T$? Justificar adecuadamente.
2. ¿Cómo se garantiza la aplicabilidad de EG? ¿ La matriz $(\mathbf{I} - p\ \mathbf{W}\ \mathbf{D})$ está bien condicionada? ¿Cómo influye el valor de p ?

Además, **se pide** realizar un informe utilizando como guía las pautas de laboratorio de la materia conteniendo la experimentación pedida en la siguiente sección.

Es importante incluir en el informe del trabajo práctico, en la sección desarrollo, aquellas decisiones tomadas en función de las estructuras de datos utilizadas y las alternativas consideradas y descartadas para los métodos utilizados.

Experimentación

Se deberá realizar tanto un análisis cualitativo como cuantitativo de los métodos vistos en el trabajo.

Para el análisis cuantitativo, se pide, como mínimo, estudiar los tiempos de procesamiento en función del tamaño del grafo de páginas y de la densidad del mismo.

Para el análisis cualitativo se deberán estudiar los rankings obtenidos, en función de la estructura del grafo, y del valor de p.

Para esto, el grupo **deberá** proponer al menos 3 instancias de prueba no triviales, que crean pertinentes para analizar el método (entregando además los archivos correspondientes).

Para el análisis, guiarse y responder las siguientes preguntas:

1. ¿Cómo es el ranking obtenido en cada caso de acuerdo a la estructura del grafo páginas?
2. ¿Qué conclusiones pueden sacar de la interpretación de los resultados?
3. Respecto del ranking de Page: ¿Funciona cómo era esperado? ¿Hubo sorpresas? ¿Qué pueden concluir sobre su significado? ¿Dirían que es un buen ranking?
4. ¿Cómo es la calidad de los rankings?

Datos de entrada/salida

Los archivos de entrada y salida serán de texto plano, y deberán respetar el siguiente formato:

archivo entrada: En la primera línea un entero N, la cantidad total de páginas. En la segunda línea un entero M, la cantidad total de links. Luego siguen M líneas, cada una con dos enteros i j separados por un espacio ($1 \leq i, j \leq N$), indicando que hay un link de la página i a la página j.

archivo de salida: La primera línea deberá contener el valor de p utilizado. Luego, deberán seguir N líneas, donde la línea i contiene el valor del ranking de Page para la página i.

El programa ejecutable deberá cumplir con el siguiente formato de uso:

```
./tp1 archivo p
```

Donde `archivo` es la ubicación del archivo de entrada a leer, y `p` es el valor de p a usar. El archivo de salida deberá ser escrito en la ubicación `archivo.out`.

La cátedra provee además un conjunto de tests con archivos de entrada y salida esperada, los cuales deberán funcionar correctamente en sus implementaciones para aprobar el trabajo.

Fechas de entrega

- *Formato Electrónico:* hasta el Domingo 15 de Abril de 2018, a las 23:59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP1]` seguido de la lista de apellidos de los integrantes del grupo separados por punto y coma `;`.
- Formato físico: Lunes 16 de Abril de 2018, 17 hs. en la clase de laboratorio.
- Pautas de laboratorio: `http://www-2.dc.uba.ar/materias/metnum/homepage.html`

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.

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.

5.2. Código

```
float EPSILON = 0.001;


// constructor de una matriz de tamanio n con unos en los indices de los links

matriz::matriz(unsigned int n, link links[], int m){
  tamanio = n;
  filas.resize(n);
  for (int i = 0; i < m; i++) {
    filas[get<1>(links[i])-1].insert(make_pair(get<0>(links[i])-1,1));
  }
}

void matriz::eliminacion_gausiana(matriz &L) {
    for (unsigned int j = 1; j <= tamanio - 1; j++) {
        for (unsigned int i = j + 1; i <= tamanio; i++) {
            // como la matriz es estrictamente diagonal dominante
            // nuestro a_j_j nunca va a ser cero
            float a_i_j = dame_elem_matriz(i, j);
            if (abs(a_i_j) > EPSILON){
              float a_j_j = dame_elem_matriz(j, j);
              float cociente = (a_i_j / a_j_j);
              L.agregar_elemento(i, j, cociente);
              //resta de diccionarios
              resta_filas(dame_fila(i), dame_fila(j), cociente);
            }
        }
    }
}

vector<float> matriz::solucion_upper(vector<float>& y) {
    vector<float> x;
    x.push_back(y[tamanio - 1]/this->dame_elem_matriz(tamanio, tamanio));
    for (unsigned int i = tamanio - 1; i >= 1; i--) {
        float suma_parcial = 0;
        for (unsigned int j = 1; j <= x.size(); j++) {
            float elem_matriz = this->dame_elem_matriz(i, i+j);
            elem_matriz = elem_matriz * x[x.size() - j];
            suma_parcial = suma_parcial + elem_matriz;
        }
        float resultado = (y[i-1] - suma_parcial)/this->dame_elem_matriz(i, i);
        x.push_back(resultado);
    }
    std::vector<float> reverse_x;
    for (unsigned int i = 1; i <= tamanio; i++) {
      reverse_x.push_back(x[tamanio-i]);
    }
    return reverse_x;
}


vector<float> matriz::rankear(float p) {
  // calculo la matriz diagonal
  vector<float> D = suma_columnas();
  // hago W*D
```

```
  this->multiplicacion(D);
  // hago p*W*D
  this->multiplicacion_escalar(p);
  // hago p*W*D-I
  this->restar_identidad();
  // hago I-p*W*D
  this->multiplicacion_escalar(-1);
  // hago la matriz L y la U
  matriz L(tamanio);
  L.crear_identidad();
  this->eliminacion_gausiana(L);
  // resuelvo LU x = e
  vector<float> Y = L.solucion_lower();
  return this->solucion_upper(Y);
}

void matriz::multiplicacion(vector<float> &matriz_D) {
    for (unsigned int i = 0; i < tamanio; i++) {
        for (unsigned int j = 0; j < tamanio; j++) {
            std::map<Columna, Valor>::iterator it = filas[j].find(i);
            if (it != filas[j].end()) {
                //la clave esta en la matriz
                if (matriz_D[i] != 0) {
                    // El elemento de la diagonal es =! a cero
                    if (abs(it->second * matriz_D[i]) > EPSILON) {
                        it->second = it->second * matriz_D[i];
                    }
                } else {
                    filas[j].erase(it);
                }
            }
        }
    }
}
```

6. Bibliografía

[1] Burden, R. and Faires, J. (2011). Numerical analysis. [Boston, MA]: Brooks/Cole, Cengage Learning.

[2] Infolab.stanford.edu. (2018). The Anatomy of a Search Engine. [online] Available at: http://infolab.stanford.edu/ backrub/google.html [Accessed 15 Apr. 2018].