DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Tirate un qué, tirate un ranking... [Primera entrega]

6 de octubre de 2014 — Métodos Numéricos

Grupo Autodenominado "Los Pichis"

Integrante	LU	Correo electrónico
De Sousa Bispo, Germán Edgardo	359/12	german_nba11@hotmail.com
De Sousa Bispo, Mariano Edgardo	389/08	marian_sabianaa@hotmail.com
Valdés Castro, Tobías	800/12	tobias.vc@hotmail.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

Introducción

Los talentosos y apuestos muchachos que instauraron la moda de los cabellos raros y los pasos bailables más utilizados en los últimos años han caído lentamente en el olvido. Con el fin de volver a su momento de gloria, el grupo tropical ha contactado al DC para brindarles una solución teniendo en cuenta el bajo presupuesto con el que cuentan en el momento.

Los motores de búsqueda son herramientas claves para explorar la red. Lo único que debemos hacer para utilizarlos es enviar un *query* con lo que querramos buscar, y allí el motor se encargará de darnos la información correspondiente a nuestra búsqueda. Para que la experiencia del usuario sea óptima, esta información *no puede estar presentada de cualquier forma*. Por lo tanto, una tarea vital del buscador será devolver al usuario un listado de páginas cuyo orden será impuesto por la *relevancia* de cada página.

El equipo de Métodos Númericos ha llegado entonces a la siguiente conclusión: para volver a todos los televisores nacionales e internacionales, este grupo de genios musicales debe figurar en páginas importantes en la web (y así encabezar todas las búsquedas). El problema ahora radica en determinar cuáles son las páginas importantes que posicionarán al grupo en los primeros links de cualquier búsqueda tropical.

De esta manera, el grupo autodenominado "Los Pichis" comenzó la labor de estudiar algunos de los más conocidos algoritmos de búsqueda: *In-Degree*, *PageRank* y *Hyperlink-Induced Topic Search*(HITS).

El primero se basa en generar el ranking de importancia de la web teniendo en cuenta solamente la cantidad de links que apuntan a la página. Mientras mayor sea la cantidad de links, más importante es la página.

PageRank es un algoritmo que luego de buscar en toda la red e indexar los datos obtenidos para realizar una búsqueda eficiente, rankea la importancia de la base de datos de manera que, cuando el usuario realiza una búsqueda, las páginas más importantes se presenten primero. Si una página u apunta a una página v, se puede decir que v es importante. Sin embargo, para no dejar que simplemente una página sea más importante por tener muchas páginas apuntadas a ella, se puede realizar un ponderado de los links utilizando la importancia de la página de origen (no es lo mismo ser apuntado por una página importante que por muchas no importantes). De esta manera, consideramos la importancia de una página v como la importancia de la página u (que apunta a v) e inversamente proporcional al grado de u (es decir, la cantidad de links que posee la página u). Entonces, si la página u contiene n_u links, uno de los cuales apunta a v, el aporte de ese link a la página v será x_u/n_u. Como para cada página pedimos la siguiente ecuación:

$$x_k = \sum_{j \in L_k} \frac{x_j}{n_j}, \qquad k = 1, \ldots, n. \tag{1}$$

Luego, el modelo planteado es equivalente a encontrar un $x \in \mathbb{R}^n$ tal que $Px = x$. Esto significa, encontrar un autovector asociado al autovalor 1 de una matriz cuadrada, tal que $x_i \geq 0$ y $\sum_{i=1}^{n} x_i = 1$.

Finalmente, la idea del método *HITS* se basa en una noción de autoridad que se trasmite de una página a otra mediante links. Es decir, se considera que existen páginas que cumplen rol de autoridad sobre un tema y se trata de modelar la relación entre estas páginas y aquellas que las apuntan, las cuales se denominan *hubs*. La relación entre ambos se establece matricialmente de la siguiente forma:

$$x = A^t y \tag{2}$$
$$y = Ax, \tag{3}$$

Siendo x un vector de peso de autoridad, y el de los hubs y A la matriz de adyacencia creada a partir de los links de una página a otra. Kleinberg propone en *Authoritative sources in a hyperlinked environment* con un y_0 inicial, al cual aplicarle esta ecuación iterativamente para que, bajo ciertas condiciones que nuestro problema cumple, el método converja. En base a este ranking obtenido

luego de realizar las iteraciones, se obtienen las páginas que son mejores autoridades y mejores hubs.

aplicando luego el paso de normalización correspondiente. Los autores proponen comenzar con un y_0 incial, aplicar estas ecuaciones iterativamente y demuestran que, bajo ciertas condiciones, el método converge. Finalmente, en base a los rankings obtenidos, se retorna al usuario las mejores t *autoridades* y los mejores t *hubs*.

1. Desarrollo

1.1. Ideas sobre la Implementación

1.1.1. Características generales del problema

1.2. Implementación

En la implementación, decidimos utilizar herencia y polimorfismo para simplificar el código. Este trabajo práctico es significativamente más grande que el anterior; una implementación descuidada necesariamente nos conduciría a muchos errores, complejos de encontrar y corregir.

Los puntos centrales tenidos en cuenta para el desarrollo fueron:

- **Algoritmos**: Debemos implementar tres algoritmos distintos para buscar prácticamente la misma información: *Page Rank, HITS, In Degree*
- **Parsing**: El parseo de los archivos de inputs deben ser distintos dependiendo de su instancia: *Standford, Toronto*
- **Declarativiad**: Nos es más fácil pensar el problema si tenemos una `WebNet` y un conjunto de `WebPage` para aplicar los algoritmos, las cuales poseen a su vez `Rank` . En el único momento que efectivamente necesitamos sus valores (por ejemplo cantidad de nodos con los que se conecta) es para aplicar el método de la potencia. De esta manera la implementación de las operaciones de matriz quedan ocultas dentro de cada algoritmo de *ranking*.

Escribiendo un pseudo-código, el punto de entrada *(main)* de nuestra aplicación es muy similar al siguiente:

```
PageRanking()
1 ParsingAlgorithm parsingAlgorithm = createParsingAlgorithmFromParameters(entryPointParameters)
2 WebNet net = parsingAlgorithm.parseFile(webDefinitionFile)
3 RankingAlgorithm rankingAlgorithm = createRankingAlgorithmFromParameters(entryPointParameters)
4 rankingAlgorithm.RankPages(net)
5 parsingAlgorithm.SaveRank(net)
```

Como se comentó anteriormente, es necesario para *Page Rank* y *HITS* aplicar un método iterativo para obtener el *ranking* que buscamos. En este trabajo, utilizamos el método de la potencia.

Por otro lado, sabemos que la red generada por la interconexión entre páginas genera un grafo donde el grado de cada nodo rara vez es n (siendo n la cantidad de páginas). Por lo tanto, generar la matriz de adyacencia no parece ser una buena solución: tendríamos muchos elementos cuyo valor es cero. La cátedra sugirió tres posibles implementaciones de matrices esparsas: *Dictionary of keys (DOK)*, *Compressed Sparse Row (CSR)* y *Compressed Sparse Column (CSC)*. Nosotros optamos por *CSR*, y la justificación la detallamos a continuación.

1.2.1. Implementación con matriz Column Sparse Row (CSR)

Definamos como n a la cantidad de columnas y filas de la matriz, *CSR* la almacena utilizando únicamente tres arreglos (sean: *rowPointers, colIndexes y values*), la dimensión de ellos son: n para *rowPointers*, cantidad de elementos no nulos para *colIndexes* e igual cantidad para *values*.

La semántica de estos arreglos es la siguiente, suponiendo que i hace referencia a filas y j a columnas:

- `rowPointers`: El valor de la i-ésima posición referencia el índice de colIndexes a partir del cual se encuentran los elementos de la i-ésima fila. En caso de no haber elementos en la i-ésima fila de la matriz, esta posición se completa con el mismo valor que la i-1 (simplemente por cuestiones implementativas).
- `colIndexes`: El valor contenido dentro de cada posición representa a j
- `values`: Contiene los valores de la matriz, cada posición se relaciona uno a uno con *colIndexes*.

Si queremos obtener el elemento a_{ij} podemos definir el siguiente pseudo-codigo:

```
ELEMENT(i, j)
1  int colIndexLower = rowPointers[i]
2  int colIndexUpper = rowPointers[i+1]
3  foreach col : colIndexLower to colIndexUpper
4  . if colIndexes[col] == j
5  . . return values[col]
```

De esta manera resulta muy sencillo efectuar una multiplicación entre la matriz y un vector: Iterando las filas multiplicamos solo las posiciones j del vector que tengamos en colIndexes por el valor almacenado en values. Si ninguna multiplicación es efectuada, el valor el cero. Por la naturaleza de la multiplicación de matriz por vector, resulta poco útil utilizar la estructura *CSC*. *CSC* es conveniente si necesitamos multiplicar por la traspuesta de una matriz, ya que el mismo algoritmo la recorrería de manera traspuesta. En el algoritmo de *HITS* necesitamos la traspuesta, sin embargo, utilizamos la misma estructura, cargando los índices de manera inversa.

Tanto *CSR* como *CSC* son complejos de crear si los datos no son suministrados de manera ordenada. Para eso, generamos una clase *CSRBuilder* al cual se le pasan los datos de manera desordenada y una vez que la carga de datos se completa, se encarga de ordenarlos, crear los índices e instanciar una *CSRMatrix*. Si bien no es idéntico, preserva el espíritu de un *DOK*, ya que termina utilizando de cláve *i, j* por la cual ordena la información antes de generar la matriz esparsa por filas.

Por la modularización generada, no sería complejo modificar el programa para incluir las dos implementaciones faltantes y hacer pruebas de *performance*. Lamentablemente por falta de tiempo, esto no fue posible.

1.2.2. Problemas en la Implementación

En comparación al trabajo anterior, tuvimos menor cantidad de problemas a la hora de la implementación.

Notamos dos como relevantes: para el algoritmo de Standford supusimos que los comentarios no se incluían en los tests. Sin embargo a la hora de testearlo estos los incluían. El segundo punto a destacar fue por desconocimiento del lenguaje: nos tomó bastante tiempo entender cómo implementar un método abstracto así como dada una instancia concreta, devolver su super clase. El compilador no nos ayudó con la descripción de los errores.

1.3. Experimentación

1.3.1. Convergencia de PageRank

1.3.2. Convergencia de HITS

1.3.3. Tiempo de Cómputo

1.3.4. Análisis Cualitativo

1.3.5. Casos de Ejemplo

2. Conclusión

3. Bibliografía y referencias

- **STL de C++**: `http://en.cppreference.com`.
- **Funciones de Métodos Numéricos:**

  Eliminación Gaussiana y *backward substitution*: Richard BURDEN, Numerical Analysis 9th Ed. Chapter 6
- **Contador de clocks**: `http://www.mcs.anl.gov/~kazutomo/rdtsc.html`