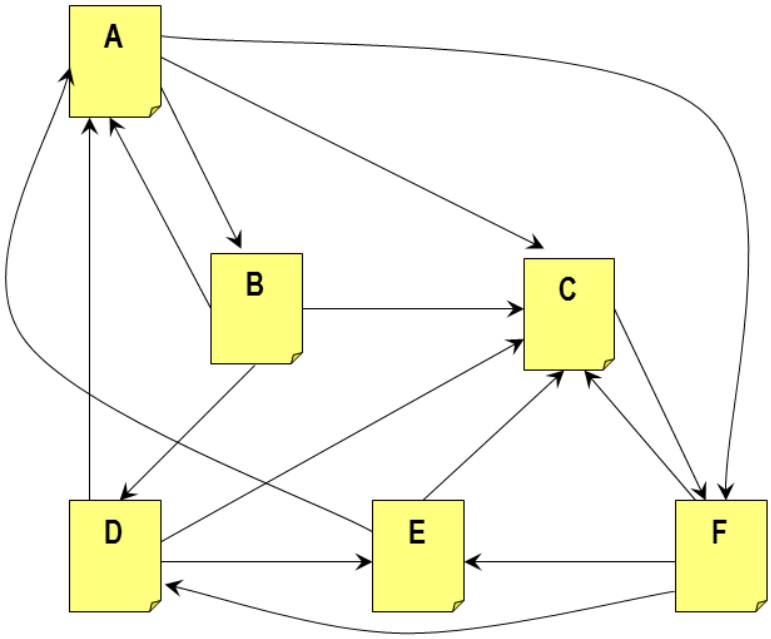
7ª práctica de laboratorio de SIW
“PageRank”

Motivación

Los primeros buscadores web aplicaron todas las técnicas conocidas de recuperación de información para resolver búsquedas en la Web y, aún así, fallaron. No sería hasta la invención del ranking basado en hiperenlaces que los buscadores se convirtieron en la principal vía de acceso a la Web haciéndola verdaderamente útil.

Se recomienda **encarecidamente** la lectura de los artículos de Marchiori (1997) y de Page *et al.* (1999). En el primero se sugiere por primera vez la explotación de la naturaleza hipertextual de la Web para las búsquedas pero se señala que no hay una forma eficiente de hacerlo; en el segundo se muestra que sí hay una forma eficiente y se presenta una aplicación práctica: *PageRank*, semilla de Google.

Puesto que una lectura pausada de dichos artículos requerirá un tiempo del que no se dispone en la sesión de laboratorio puede optarse por examinar el correspondiente artículo de la Wikipedia para, a continuación, examinar una implementación de PageRank para un grafo concreto (ver figura) mediante hojas de cálculo.



¡Atención! El grafo de la figura tiene dos características fundamentales para una implementación simplista de *PageRank* (ver primera ecuación). En primer lugar se trata de un grafo conexo y en segundo todos los nodos tienen enlaces salientes—esto es, no hay sumideros.

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Existen dos opciones para aplicar *PageRank* a un grafo no conexo y/o con sumideros:

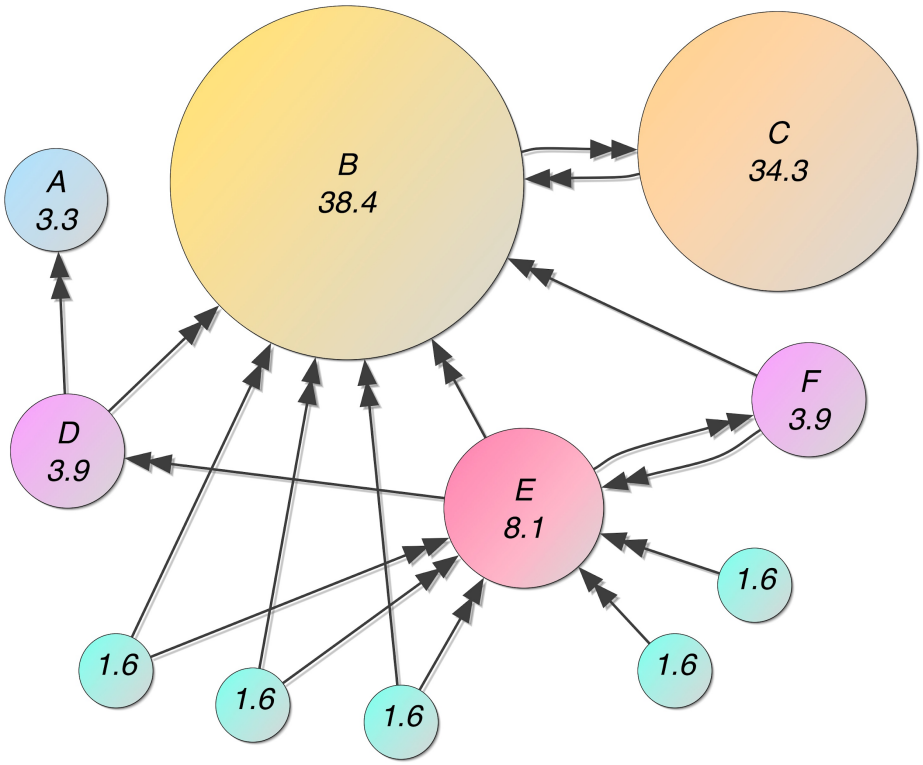
1. Añadir un nodo ficticio apuntado desde todos los nodos reales y que apunta a todos los nodos reales. El inconveniente es que dicho nodo va a "acaparar" una parte del *PageRank* global; no obstante, se trata de un problema menor puesto que no afecta al orden inducido sobre el resto de nodos.
2. Incorporar un *damping factor* (o factor de "teletransporte") y suponer que el *PageRank* de los nodos sumideros se reparte equitativamente entre el resto de nodos (ver segunda ecuación). Ese *damping factor* se suele establecer en 0.85 de tal modo que nuestro "random surfer" tiene un 85% de probabilidades de seguir haciendo click en los enlaces que se encuentra y un 15% de "saltar" a una página aleatoria en la web.

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Descripción del ejercicio y del entregable

El objetivo del ejercicio es implementar en Python el algoritmo *PageRank*; dicho algoritmo recibirá como entrada un grafo (podría ser un diccionario donde las claves sean nombres de los nodos y los valores listas de nombres de nodos) y producirá como salida un diccionario asociando a cada nodo su valor *PageRank*. La salida del script se dará por pantalla y deberá estar ordenada por valor *PageRank*.

Para verificar el funcionamiento del algoritmo se aplicará sobre el siguiente grafo (fuente: Wikipedia).



El *script* recibirá como entrada un fichero de texto que indicará las aristas en el siguiente formato: `origen, destino`.

El siguiente ejemplo corresponde al grafo de la primera figura, no al de la Wikipedia:

```
A,B
A,C
A,F
B,A
B,C
B,D
C,F
D,A
D,C
D,E
E,A
E,C
F,C
F,D
F,E
```

Se entregará en el campus virtual un archivo comprimido con el código fuente, un documento indicando las decisiones tomadas para la implementación, y un archivo de texto que representará al grafo sobre el cual se debe probar la implementación.

Ejercicio de entrega opcional

Leer el trabajo de Mihalcea y Tarau (2004) sobre la adaptación de *PageRank* para la extracción de palabras y sentencias clave de un texto.

Reflexionar sobre el modo en que podría transformarse un texto en un grafo donde las palabras serían nodos y los enlaces representarían co-ocurrencias de términos.

Trata de responder a las siguientes preguntas:

- ¿Cómo maneja tu implementación de *PageRank* los enlaces salientes repetidos?
- ¿Deberían tenerse en cuenta en la Web? ¿Y en un texto?
- ¿Cómo podría segmentarse un texto en sentencias? Consulta la sección *"3.8 Segmentation"* de Bird *et al.* (2009) **pero** te recomiendo usar el `SentenceTokenizer` de *textblob*.

Implementa un *script* en Python que reciba como parámetro el nombre de un fichero de texto; dicho fichero contendrá texto plano (podría tener saltos de línea o no pero no se usarán como separadores de sentencias) y para el mismo se obtendrá internamente su representación gráfica. Sobre dicho grafo se aplicará la adaptación de *PageRank* para: (1) obtener los 10 términos más relevantes y (2) las 10 sentencias más relevantes pero en orden de aparición.

¡Atención! ¿Qué papel juegan las palabras vacías? ¿Deben eliminarse? De ser así, ¿en qué momento?

Se entregará un archivo comprimido con el código fuente, el archivo de entrada usado en la demostración y un documento explicando las decisiones tomadas en la implementación.

Referencias bibliográficas

- Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".
- Marchiori, M. (1997). The quest for correct information on the web: Hyper search engines. *Computer Networks and ISDN Systems*, *29*(8), 1225-1236.
- Mihalcea, R., & Tarau, P. (2004). Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*.

- Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The PageRank citation ranking: Bringing order to the web*. Stanford InfoLab.