PageRank

# Introducción

Cuando los motores de búsqueda como [Google](https://www.google.com/?hl=es) muestran resultados, intentan colocar las páginas más “importantes” y de más alta calidad más arriba que las páginas menos importantes.

Pero ¿Cómo hacen esto?

Una heurística podría ser que una página “importante” es aquella que es referencia por muchas otras páginas, es decir, muchas otras páginas tienen enlaces a la página en cuestión, pues es razonable imaginar que la mayoría de los sitios apuntarán a páginas web de mejor calidad que otras. Por lo tanto, podríamos imaginar un sistema donde a cada página se le de una valoración (un rango, en inglés rank) de acuerdo al número de páginas que tengan enlaces a esta y, los rangos más altos, serían una señal de una importancia más alta.

Esta definición no es perfecta, pues si alguien quiere hacer que su página siempre aparezca en el tope de la búsqueda, podría crear muchas otras páginas que tengan enlaces a la página de interés para que sun rango se incremente. Por esa razón, para el algoritmo PageRank, que fue creado por los co-fundadores de Google, una página web es más importante si es apuntada por muchas otras páginas que también son importantes. Esta definición pareciera un poco circular, pero hay varias estrategias para calcular estos rangos.

El objetivo de esta práctica es que aprehendas cómo se pueden utilizar los modelos de grafos para resolver este tipo de problemas.

## Modelo de navegación aleatorio

Una manera de pensar en PageRank es con el [modelo de navegación aleatorio](https://www.appyweb.es/diccionario/modelo-navegacion-aleatorio/) (en inglés: [random surfing model](https://en.wikipedia.org/wiki/Random_surfing_model)) el cual considera el comportamiento de un navegante hipotético en la Internet que hace clicks en los enlaces de manera aleatoria. Considera el siguiente corpus de páginas web en el que una flecha entre dos páginas indica la existencia de un enlace.



El modelo de navegación aleatorio simula un navegante que comienza con una página al azar y luego elige enlaces aleatoriamente para seguir navegando. Por ejemplo, si el navegante está en “Página 2”, podría elegir aleatoriamente entre “Página 1” y “Página 3” para visitar (los enlaces duplicados en una página son tratados como un solo enlace y los enlaces de una página a sí misma son ignorados). Si se encontrara en “Página 3”, entonces el navegante podría elegir ir a “Página 2” o “Página 4”.

El PageRank de una página puede ser descrito como la probabilidad de que un navegante aleatorio esté en una página en cualquier momento del tiempo. Después de todo, si hay más enlaces a una página particular, es más probable que un navegante aleatorio termine en esta. Además, hay más probabilidad de dar click a un enlace de una página más importante que aquellas menos importantes que son referenciadas por menos páginas.

Una manera de interpretar este modelo es mediante un grafo, donde un nodo representa una página y un arco representa que la página que está al origen (nodo del que sale el arco) tiene un enlace a la página que está en el destino (nodo al que llega el arco).

Muestreando nodos del grafo aleatoriamente, podemos estimar el PageRank de cada página. Podemos empezar eligiendo una página al azar, entonces mantenernos eligiendo enlaces aleatoriamente llevando un trazo de cuántas veces hemos visitado cada página. Después de haber reunido toda nuestra muestra (basado en un número que elegimos antes), la proporción de la cantidad de veces que estuvimos en cada página puede ser una estimación del rango de esta.

Sin embargo, esta definición de PageRank resulta un poco problemática, si consideramos una red de páginas como la siguiente.



Si eligiésemos aleatoriamente comenzar en “Página 5”, no tendremos otra alternativa que movernos a “Página 6”, de allí no tendremos otra opción que movernos a “Página 5” de nuevo y estaríamos dando vueltas en círculo. Estaríamos estimando un rango de 0.5 para cada una de esas dos páginas y 0 para el resto dado que las otras nunca serán visitadas.

Para asegurarnos de que siempre podamos llegar a otro lugar en el corpus de páginas web, introduciremos a nuestro modelo un factor de amortiguamiento d. Con probabilidad d (donde d es 0.85 usualmente), el navegante aleatorio eligierá uno de los enlaces en la página actual aleatoriamente. Por en otro caso (con probabilidad 1- d), el navegante aleatorio elige una de todas las páginas del corpus al azar (incluida aquella en la que se encuentra actualmente).

Nuestro navegante aleatorio ahora empieza eligiendo una página al azar y, entonces, por cada muestra adicional que quisieramos generar, elegimos un enlace de la página actual aleatoriamente con probabilidad d, y elegimos cualquier otra página al azar con probabilidad 1- d. Si podemos llevar un trazo de cuántas veces hemos visitado cada página en la muestra, podemos tratar la proporción de nodos en los que hemos estado como el PageRank de la página que representa.

## Algoritmo iterativo

También podemos definir el PageRank de una página usando una expresión matemática recursiva. Sea PR(p) el PageRank de una página dada p: la probabilidad de que un navegante aleatorio termine en esa página. ¿Cómo podemos definir PR(p)? Sabemos que hay dos maneras de que un navegante aleatorio alcance esa página.

1. Con probabilidad 1 - d, el navegante la eligierá al azar y terminará en p.
2. Con probabilidad d, el navegante sigue un enlace desde la página i hasta la página p.

La primera condición es bastante sencilla de expresar matemáticamente: Esto es 1 - d dividido por N, donde N es la cantidad de páginas total en el corpus. Esto es porque la probabilidad 1 - d de elegir una página al azar se divide por igual entre todas las N páginas posibles.

Para la segunda condición, necesitamos considerar cada posible página i que enlace a la página p. Para cada una de esas páginas entrantes, sea NumLinks(i) el número de enlaces en la página i. Cada página i que enlaza con p tiene su propio PageRank, PR(i), que representa la probabilidad de que estemos en la página i en un momento dado. Y dado que desde la página i viajamos a cualquiera de los enlaces de esa página con la misma probabilidad, dividimos PR(i) por el número de enlaces NumLinks(i) para obtener la probabilidad de que estuviéramos en la página i y eligiéramos el enlace a la página p.

Esto nos da la siguiente definición para el PageRank de una página p.

En esta fórmula, d es el factor de amortiguamiento, N es el número total de páginas en el corpus, i abarca todas las páginas que enlazan con la página p y NumLinks(i) es el número de enlaces presentes en la página i.

Entonces, ¿cómo haríamos para calcular los valores de PageRank para cada página? Podemos hacerlo a través de la iteración: comience asumiendo que el PageRank de cada página es 1 / N (es decir, tiene la misma probabilidad de estar en cualquier página). Luego, use la fórmula anterior para calcular nuevos valores de PageRank para cada página, en función de los valores de PageRank anteriores. Si seguimos repitiendo este proceso, calculando un nuevo conjunto de valores de PageRank para cada página en función del conjunto anterior de valores de PageRank, eventualmente los valores de PageRank convergerán (es decir, no cambiarán más que un pequeño umbral con cada iteración).

# Práctica

## Modelo de datos

Para este problema utilizaremos un modelo de grafo basado en listas de adyacencia. Comúnmente este modelo se representa mediante una lista de nodos, siendo un nodo un concepto abstracto que se ajusta a las necesidades del modelo. Como estamos hablando de listas de adyacencia, es indispensable que un nodo tenga una lista de los arcos que salen de este.

En un modelo bastante general de grafos, normalmente se tiene una estructura para representar los arcos que tiene (al menos) un apuntador para cada nodo que conecta, en ocasionses se puede almacenar cualquier tipo de información en los arcos. Se tiene un modelo que representa un arco adyacente cuyo contenido podría ser un apuntador al arco y un nodo con cualquier información requerida y una lista de arcos adyacentes. De esa manera podr´iamos garantizar la existencia de un arco una sola vez y, en los nodos solo se tienen apuntadores a estos y evitamos la repitencia de datos en memoria. La biblioteca [DeSiGNAR](https://github.com/R3mmurd/DeSiGNAR) (por ejemplo), tiene una representación bastante genérica para grafos y digrafos adaptada para resolver una amplia gama de problemas reprensetados por estos modelos. Pueden verla [aquí](https://github.com/R3mmurd/DeSiGNAR/blob/master/include/graph.H). Sin embargo, existen problemas para los que, dada la sencillez del modelo, es preferible crear una abstracción del grafo adaptada a la necesidad.

En el caso que nos compete en esta práctica, solo requerimos que un nodo almacene el nombre de una página y, como no se requiere que los arcos almacenen información, tener una lista de los nodos a los cuales hace referencia la página que este alberga. Un código fuente rápido y sencillo para representar este grafo podría ser algo como lo que sigue:

struct Node

{

string page\_name;

vector<string> linked\_page\_names;

};

using Graph = vector<Node>;

Dado que los nombres de las páginas suelen tener una cantidad pequeña de caracteres, no sería grave que, si varias páginas tienen enlaces a una misma página, se repita el nombre en las diferentes listas de adyacencia. Dado que no es un modelo que vaya a estar siendo modificado mediante eliminación de nodos o arcos, usamos vector que nos da un modelo de memoria compacto. Aún así nos enfrentaríamos a un problema que es que, cuando estamos construyendo el grafo, podríamos encontrarnos intentando añadir un nodo, bien sea a la lista de nodos o en alguna lista de adyacencia, que ya exista (enlaces repetidos de una página a otra por ejemplo), entonces tendriamos que lidiar con buscar si ya existe antes de insertar.

Para resolver esto, podríamos utilizar una estructura de datos que ignore las entradas repetidas al momento de insertar, esto podría ser mediante un set (estructura de datos en tiempo O(N lg N)) o mediante un unordered\_set (estructura de datos en tiempo O(1)) y nuestro modelo podría representarse de la siguiente manera:

struct Node

{

string page\_name;

unordered\_set<string> linked\_page\_names;

};

using Graph = unordered\_set<Node, NodeHash, NodeEqual>;

Dado que nuestro tipo Node es definido por nosotros, debemos entonces lidiar con el cálculo de la función hash para un nodo y la forma de comparar la igualdad de un nodo. Esta ´ultima podría resolverse sobrecargando el operador == en el nodo e ignoramos el argumento plantilla NodeEqual. Si usásemos la clase set, o creamos el functor para comparación “menor que” o solo sobrecargamos el operador < dentro de la clase Node.

Ya esto suma cierta complejidad a este modelo. Por esa razón, en pro de la sencillez, el modelo en esta práctica se representa mediante un mapa de cadenas. Es decir, utilizando bien sea la clase map (O(N lg N)) o la clase unordered\_map (O(1)), nos darían exactamente el modelo requerido. Dicho lo anterior, nuestro grafo en esta práctica está representado mediante la siguiente estructura:  
  
unordered\_map<string, unordered\_set<string>> graph;

En este modelo/home/alex/tmp/Sources/Courses/ULA/Codes/dynamic-programming/viterbi/viterbi.cpp

/home/alex/tmp/Sources/Courses/ULA/Codes/dynamic-programming/viterbi/test.cpp

/home/alex/tmp/Sources/Courses/ULA/Codes/dynamic-programming/viterbi/viterbi.hpp, sabiendo que el primer parámetro plantilla de una “Key” y el segundo es el valor asociado a esto, entonces tenemos que “Key” representa un nodo (el nombre de una página) y el segundo parámetro es nuestra lista de adyacencia representada mediante un conjunto de nombres de las páginas a las que se enlaza.

Si consideramos el primer corpus mostrado en este documento, la estructura de datos de arriba quedaría como sigue (el orden escrito acá no es exactamente el orden real que tendrá la estructura de datos en memoria, de allí el prefijo unordered en los nombres de las clases, no hay un orden garantizado):

graph = {

“Página 1”: { “Página 2” },

“Página 2”: { “Página 1”, “Página 3” },

“Página 3”: { “Página 2”, “Página 4” },

“Página 4”: { “Página 2” }

}

## Distribución del proyecto

Se te provee un paquete de proyecto con una estructura básica, algunos códigos ya resueltos y otros que debes resolver.

El proyecto está estructurado de la siguiente manera:

**lib/include/IOUtils.hpp y lib/src/IOUtils.cpp**: Contienen la implementación de operadores para imprimir diversas estructuras de datos que manejaremos en el proyecto, también encontrarás una macro LOG que sirve para imprimir en pantalla cualquier mensaje con el que se quiera hacer seguimiento del comportamiento. Estos mensajes solo se verán si al compilar se hace con la bandera -DDEBUG. La forma general de usarla es similar a printf, es decir, el primer parámetro es una cadena formateada y luego vendrían las variables requeridas. Estudia estos archivos, puedes agregar cualquier cosa que requieras pero no modifiques lo existente.

**lib/include/TestUtils.hpp y lib/src/TestUtils.cpp**: Contienen algunas operaciones básicas para hacer pruebas unitarias de nuestras soluciones. Estudialos, puedes agregar cualquier cosa que requieras pero no modifiques lo existente.

**lib/include/Crawler.hpp y lib/src/Crawler.cpp**: Contienen la implementación de la clase Crawler, esta se encarga de navegar sobre un directorio que contiene un corpus de páginas web y construir el digrafo de enlaces entre estas.

**lib/include/PageRank.hpp y lib/src/PageRank.cpp**: Contienen la definición de la clase PageRank la cual debes resolver.

**resources**: es un directorio que contiene tres directorios y, cada uno de estos, contiene un corpus de páginas web diferente.

**main/src**: Directorio que contiene un programa que recibe la ruta de un corpus y ejecuta las operaciones de PageRank y muestra los resultados. Un ejemplo de ejecución es: ./main/bin/PageRank resources/corpus0.

**test/src**: Directorio que contiene un programa con pruebas unitatiras para la clase Crawler y otro que contiene pruebas unitarias para la clase PageRank.

**Makefile**: Contiene reglas de compilación para todo el proyecto: make main compilara todos los programas en main/src y generar los ejecutables en main/bin. make test hace algo similar con todo lo que está en test/bin y make all compila todos los main y todos los test.

**run-tests.sh**: es un script bash que ejecuta todos los tests.

## Tu trabajo

Tu trabajo consiste en resolver los métodos de la clase PageRank en el archivo PageRank.cpp. Los métodos que debes resolver son los siguientes:

**transition\_model**: debe retornar un mapa que representa la distribución de probabilidad sobre cuál página, un navegante aleatorio, visitaría. El método recibe dos parámetros, el nombre de la página (page) que se está visitando y el factor de amortiguamiento (damping\_factor). El retorno deberá contener como “key” el nombre de la posible página a visitar y a cada una de estas, se le debe asociar el valor de probabilidad de que el navegante aleatorio la visite luego. Ten en cuenta que, por ser una distribución de probabilidad, la sumatoria de todos los valores que están allí debe ser 1.

Con probabilidad 1 - damping\_factor el navegante aleatorio debería elegir aleatoriamente una de las páginas de todo el corpus con igual probabilidad. Por ejemplo, si el corpus fuese {"1.html": {"2.html"}, "2.html": {"1.html", "3.html"}, "3.html": {"2.html", "4.html"}, "4.html": {"2.html"}}, la página fuese "2.html" y damping\_factor fuese 0.85, entonces la salida de este método debería ser {"1.html": 0.4625, "2.html": 0.0375, "3.html": 0.4625, "test4.html": 0.0375}. Esto es porque, con probabilidad de 0.85, elegimos aleatoriamente ir desde la página 2 hasta la página 1 o la página 3 (cada una de estas tiene probabilidad de 0.425), pero cada página obtiene un adicional de 0.0375 debido a que, con probabilidad de 0.15, las elegimos aleatoriamente entre las 4 páginas del corpus. Si la página en cuestión no tiene enlaces, entonces retornaremos una distribución de probabilidad para todas las páginas del corpus, todas ellas con la misma probabilidad, la probabilidad de cada una se calcula fácilmente mediante 1 / N donde N es la cantidad de páginas en el corpus.

**sample\_pagerank:** Este método retorna un mapa cuyas “keys” son las páginas del corpus y el valor asociado a cada una de estas es el PageRank estimado y recibe como parámetro el factor de amortiguamiento (damping\_factor) y la cantidad de muestras (n) que deberían ser generadas para estimar los valores PageRank. La sumatoria de los valores en el mapa que se retorna deberá ser 1. La primera muestra deberá ser generada eligiendo una página aleatoriamente (para esto, usa el método get\_random\_page que se te provee resuelto) y el resto de las muestras deberán ser generadas a partir de la muestra anterior basada en el “transition model” de esa muestra, en otras palabras, una vez que obtengas la primera muestra al azar, le calcularás el modelo de transición y, a partir de ese modelo, obtendras la siguiente página al azar basadas en la probabilidad del modelo (para esto se te provee un método resuelto llamado weighted\_choice que recibe como parámetro el modelo de transición). Finalmente, el PageRank de una página será la proporción de veces que la visites, esto es la cantidad de veces visitada dividida por la cantidad de muestras.

**iterate\_pagerank:** Este método recibe como parámetro el factor de amortifguamiento (damping\_factor) y calcula los PageRanks basados en la fórmula descrita anteriormente. Debe retornar los PageRanks de cada página con una precisión de 0.001. El retorno cumple las mismas especificaciones del método anterior. Esta operación debería comenzar asignando a cada página un PageRank de 1 / N donde N es la cantidad de páginas en el corpus, luego debería calcular repetidamente los nuevos valores de rango con base en todos los valores de rango actuales de acuerdo a la fórmula descrita anteriormente (esto es, calcular el PageRank basado en los PageRanks de todas las páginas que la apuntan. Esto debería repetirse hasta que ninguno de los valores cambien por una diferencia mayor a 0.001 entre los valores actuales y los nuevos calculados.  
  
No deberías modificar los métodos ya provistos. Solo los tres que se te indicaron.

# Evaluación

Esto lo puedes resolver en pareja (elige bien a quien te acompaña).

Tienes todo el semestre para entregar este proyecto y podrás hacer un envío una vez por semana. Si envías y fallas, entonces podrás enviar de nuevo la siguiente semana y así sucesivamente hasta terminar el semestre.

Para poder evaluarte, debes enviar el archivo PageRank.cpp, al correo [alejandro.j.mujic4@gmail.com](mailto:alejandro.j.mujic4@gmail.com) con el subject: [AYDA]-03-PageRank. El correo deberá tener los nombres y números de identificación de la pareja participante.