Trabajo Práctico 3

El trabajo práctico número 3 tiene fecha de entrega para el día ** 8/12**.

Contenido

Introducción

El objetivo de este trabajo práctico es el de modelar Internet: las páginas web y sus interacciones. Nos interesa modelar cómo podemos navegar a través de Internet. Se quiere poder realizarse varias consultas para poder entender distintos aspectos de la red. Se trabajará con porciones ínfimas de la red.

Datos disponibles

Vamos a trabajar con una ínfima porción de Internet, del portal llamado “Wikipedia”. Es posible descargarse el contenido completo de Wikipedia en cualquier idioma desde aquí. Nosotros nos enfocaremos en la versión en español, si bien el trabajo es completamente compatible con cualquier otro idioma, hay idiomas con mucha mas carga que otros.

Además, les brindamos a ustedes un parser implementado en Python para que no sea necesario que ustedes lo implementen. En caso de preferir implementar algún cambio sobre el mismo, o utilizar uno propio, no hay inconvenientes. Igualmente, para que no sea necesario consumir tiempo en esta tarea se brinda un archivo de texto de salida ya parseado, que una vez descomprimido pesa 1.3GB. Dicho archivo proviene del dump de Wikipedia en español hasta el día 1/5/2016, pero ya es posible descargar versiones más actualizadas, o bien de otras wikis. En caso de no utilizar ese archivo, o simplemente para experimentar (con otro idioma, por ejemplo) pueden simplemente ejecutarlo haciendo:

```
    $ python wiki_parser.py <path_dump> <path_parsed>
```

El archivo parseado tiene el formato TSV (cada campo es separado por un tab):

```
TituloArticulo1 Link1   Link2   Link3   ... LinkN
TituloArticulo2 Link1   Link2   Link3   ... LinkM
...
```

Dichos links deben hacer referencia a otros Títulos de Artículos. Tener en cuenta que, por simpleza del parser, pueden haber links que referencien a artículos que no existan. Por ejemplo: se han filtrado las entradas de artículos referentes a años para hacer más liviano el archivo y más interesantes los caminos a recorrer por nuestra “Pequeña Internet”.

Considerar que el archivo completo cuenta con más de 3 millones de artículos. Por lo tanto, para que puedan también realizar pruebas más rápidas, les brindamos un archivo de una reducción de la primera red que cuenta con los primeros 75.000 artículos visitados resultantes al realizar un recorrido BFS desde ‘Argentina’ en el set de datos completo de Wikipedia.

Consigna

Dado este archivo parseado, se debe modelar Internet con una estructura Grafo considerando únicamente los títulos de las páginas y las conexiones entre ellas. Esto implica determinar todas las características necesarias para el grafo. Se pide implementar un

programa que cargue inicialmente el grafo recibiendo como parámetro la ruta del archivo parseado:

```
$ ./netstats wiki-reducido-75000.tsv
```

Una vez cargada la red, se deberán realizar acciones sobre la misma a partir de comandos ingresados desde entrada estándar.

Implementación

El trabajo puede realizarse en lenguaje a elección, siendo aceptados Python y C, y cualquier otro a ser discutido con el corrector asignado.

El trabajo consiste de 3 partes:

1. El TDA Grafo, con sus primitivas completamente agnósticas sobre su uso para modelar la red de Internet.
2. Una biblioteca de funciones de grafos, que permitan hacer distintas operaciones sobre un grafo que modela Internet, sin importar cuál es la red específica.
3. El programa `NetStats` que utilice tanto el TDA como la biblioteca para poder implementar todo lo requerido.

Es importante notar que las primeras dos partes deberían poder funcionar en cualquier contexto: El TDA Grafo para cualquier tipo de TP3 (o utilidad); la biblioteca de funciones debe funcionar para aplicar cualquiera de las funciones implementadas sobre cualquier grafo que tenga las características de las de este TP (particularmente, dirigido y no pesado). La tercera parte es la que se encuentra enteramente acoplada al TP en particular.

El programa debe recibir por parámetro, cargar en memoria el set de datos (`$ ./netstats wiki-reducido-75000.tsv`) y luego solicitar el ingreso de comandos por entrada estándar, del estilo `<comando> 'parametro'`. Notar que esto permite tener un archivo de instrucciones a ser ejecutadas (i.e. `$ ./netstats wiki-reducido-75000.tsv < entrada.txt`).

De todas las funcionalidades, pueden optar por implementar distintas. Algunas consideraciones:

1. Cada funcionalidad e implementación otorga distinta cantidad de puntos, basado en dificultad y el interés del curso en que implementen dicha funcionalidad o implementación particular de la misma (Cada estrella ★ corresponde a un punto).
2. Se deben conseguir al menos 7 puntos para poder aprobar el TP.
3. En caso de obtener menos de 10 puntos, la nota máxima será 7.
4. En caso de obtener menos de 12 puntos, la nota máxima será 9.
5. En caso de obtener 16 puntos o más la nota máxima del trabajo práctico puede llegar a 11.
6. El total de puntos entre todas las funcionalidades es 18.

A continuación se listarán los comandos junto a ejemplos de entrada y salidas para el caso de la red reducida. Recomendamos trabajar con este set de datos, puesto que el original cuenta con una enorme cantidad de datos, por lo que puede demorar mucho tiempo cada una de las pruebas a realizar.

Listar operaciones (obligatorio, sin puntos)

- Comando: `listar_operaciones`.
- Parámetros: ninguno.
- Utilidad: Dado que no todas las funcionalidades o implementaciones son obligatorias, debe existir un comando que nos permita saber cuáles son las funcionalidades disponibles. Debe imprimirse una línea por cada **comando** que esté implementado.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(1)$.
- Ejemplo: Entrada:

  ```
  listar_operaciones
  ```

  Salida:

  ```
  camino
  mas_importantes
  conectados
  ciclo
  en_rango
  ```

Camino más corto (★)

- Comando: `camino`.
- Parámetros: `origen` y `destino`. Origen y destino son **páginas**.
- Utilidad: nos imprime una lista con las **páginas** con los cuales navegamos de la página `origen` a la página `destino`, navegando lo menos posible.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(P + L)$, siendo P la cantidad de páginas, y L la cantidad de Links en toda la

red.

- Ejemplos: Entrada:

```
camino Argentina,Zinedine Zidane
camino Jeremy Irons,pejerrey
camino handball,Napoleón Bonaparte
```

Salida:

```
Argentina -> Francia -> Zinedine Zidane
Costo: 2
Jeremy Irons -> Reino Unido -> Patagonia argentina -> Río Grande (Tierra del Fuego) -> pejerrey
Costo: 4
No se encontro recorrido
```

Artículos más importantes (★★★)

Utlizaremos el algoritmo de PageRank para implementar este comando, dado que además fue pensado primordialmente para este escenario.

- Comando: `mas_importantes`.
- Parámetros: `n`, la cantidad de páginas más importantes a mostrar.
- Utilidad: nos muestra las `n` páginas más centrales/importantes del mundo según el algoritmo de pagerank, ordenadas de mayor importancia a menor importancia.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(K(P+L) + P\log(n))$, siendo K la cantidad de iteraciones a realizar para llegar a la convergencia (puede simplificarse a $\mathcal{O}(P \log n + L)$ (El término $\mathcal{O}(P \log n)$ proviene de obtener los Top-n luego de haber aplicado el algoritmo).
- Ejemplo: Entrada:

```
mas_importantes 20
```

Salida:

```
Argentina, Estados Unidos, Buenos Aires, España, Francia, Provincias de la Argentina, Magnoliophyta, Alemar
```

Importante: Considerar que esto podría pedirse varias veces por ejecución, y no se desea repetir el calculo (ya que no debería el valor de pagerank de cada artículo).

Conectividad (★★)

- Comando: `conectados`.
- Parámetros: `página`, la página que se le quiere obtener la conectividad.
- Utilidad: nos muestra todos las páginas a los que podemos llegar desde la `página` pasado por parámetro y que, a su vez, puedan también volver a dicha `página`.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(P+L)$. Considerar que a todas las páginas a las que lleguemos también se conectan entre sí, y con el tamaño del set de datos puede convenir guardar los resultados.
- Ejemplo: Entrada:

```
conectados Boca Juniors
conectados Argentina
```

Salida: En ambos casos la CFC está compuesta por los mismos artículos. Dejamos acá un archivo con la salida esperada (no necesariamente en ese orden, pero sí *esos* artículos), dado que la misma consta de 44764 de los 75000 artículos. Es importante notar que la segunda consulta debería obtener un resultado en tiempo constante.

Ciclo de n artículos (★★★)

- Comando: `ciclo`.
- Parámetros: `página` y `n`.
- Utilidad: permite obtener un ciclo de largo `n` que comience en la página indicada.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(P^n)$, pero realizando una buena poda puede reducirse sustancialmente el tiempo de ejecución.
- Ejemplo: Entrada:

```
ciclo Jeremy Irons,4
ciclo Polonia,20
```

Salida:

```
Jeremy Irons -> The Silence of the Lambs (película) -> Dances with Wolves -> Premios Óscar -> Jeremy Irons
Polonia -> Unión Europea Occidental -> Unión Europea -> Idioma italiano -> Florencia -> Cannes -> Festival
```

En caso de no haber un ciclo de dicho largo empezando desde la página mencionada, debe escribirse por salida estándar `No se encontro recorrido`.

Lectura a las 2 a.m. (★★)

- Comando: `lectura`.
- Parámetros: `página1`, `página2`, …, `página_n`.
- Utilidad: Permite obtener un orden en el que es válido leer las páginas indicados. Para que un orden sea válido, si `página_i` tiene un link a `página_j`, entonces es necesario **primero leer** `página_j`. Solo se debe tener en cuenta los artículos mencionados en los parámetros. Esto, por supuesto, puede implicar que no podamos cumplir con lo pedido por encontrarnos con un ciclo.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(n + L_n)$, siendo n la cantidad de páginas indicadas, y L_n la cantidad de links entre estas.
- Ejemplo: Entrada:

```
lectura Buenos Aires,Roma
lectura Hockey sobre hielo,Roma,Japón,árbol,Guerra,Dios,universo,Himalaya,otoño
```

Salida:

```
No existe forma de leer las paginas en orden
otoño, Himalaya, universo, Dios, Guerra, árbol, Hockey sobre hielo, Japón, Roma
```

Importante: considerar lo indicado en el enunciado. Si quiero saber un orden válido para leer `página1` y `página2`, y hay un link de `página1` a `página2`, un orden válido es `página2` y luego `página1`.

Diametro (★)

- Comando: `diametro`.
- Parámetros: ninguno.
- Utilidad: permite obtener el diámetro de toda la red. Esto es, obtener el camino mínimo más grande de toda la red. *Nota*: Puede haber más de uno de estos, pero todos tendrán el mismo largo.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(P(P + L))$.
- Ejemplo: Debido a la complejidad, aplicar sobre el grafo incluso reducido puede demorar muchas horas. Por lo tanto, les acercamos un ejemplo con 5.000 artículos. Entrada:

```
diametro
```

Salida:

```
Huésped (biología) -> Agente biológico patógeno -> Animalia -> Carlos Linneo -> Finlandia -> Unión Europea
Costo: 7
```

Todos en Rango (★)

- Comando: `rango`.
- Parámetros: `página` y `n`.
- Utilidad: permite obtener la cantidad de páginas que se encuenten a **exactamente** `n` links/saltos desde la `página` pasada por parámetro.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(P + L)$.
- Ejemplo: Entrada:

```
rango Tokio,8
rango Tokio,3
rango Perón,4
```

Salida:

```
0
39347
53346
```

Comunidades (★★)

- Comando: `comunidad`.
- Parámetros: `página`.
- Utilidad: permite obtener la comunidad dentro de la red a la que pertenezca la página pasada por parámetro. Para esto, utilizaremos el sencillo algoritmo de Label Propagation.
- Ejemplo: Entrada:

```
comunidad Chile
```

  Salida: Dado que la salida puede ser muy grande adjuntamos una salida de ejemplo. No es requisito que la salida sea tal cual está (ni en orden ni los artículos), y se tendrá consideración las diferencias que se puedan llegar a tener. La idea de implementar este comando es que tengan un primer acercamiento a un algoritmo muy sencillo para detectar comunidades. En este caso, pueden ver que muchas páginas relacionadas a Chile y ciudades de Chile han quedado en la misma comunidad, lo cual sería un resultado esperable.

A quién le interese este tema puede ver otro tipo de algoritmos, como por ejemplo el Algoritmo de Louvain.

Navegacion por primer link (★)

- Comando: `navegación`.
- Parámetros: `origen`.
- Utilidad: Se dice que si comenzamos en *cualquier* artículo de wikipedia, y navegamos únicamente utilizando el primer link, eventualmente llegaremos al artículo de *Filosofía*. Por lo tanto, queremos implementar un comando que navegue usando el primer link (de los que tengamos reportados) desde la página `origen` y navegando usando siempre el primer link. Debemos continuar accediendo al primer link hasta que la página ya no tenga links, o bien hasta que hayamos llegado a 20 páginas.
- Complejidad: este comando debe ejecutar en $\mathcal{O}(n)$.
- Ejemplo: Entrada:

```
navegacion Argentina
navegacion Alemania
navegacion queso
navegacion Bolivia
navegacion Brasil
```

  Salida:

```
Argentina -> polacos
Alemania -> Derecho penal -> complicidad
queso
Bolivia -> Guerra del Acre -> Hevea brasiliensis -> Guerra del Acre -> Hevea brasiliensis -> Guerra del Acr
Brasil -> Hermeto Pascoal -> Portugués brasileño -> caña de azúcar
```

Coeficiente de Clustering (★★)

El *Coeficiente de Clustering* es una métrica que nos permite entender cuán agrupados se encuentran los vértices de un grafo. Para explicarla de manera simplificada, es similar a plantear la proporción en la que se cumple al regla de transitividad: *Cuántos de mis adyacentes son adyacentes entre sí*.

El coeficiente de clustering de un vértice i en un grafo dirigido puede calcularse como:

$$C_i = \frac{|e_{ij} : v_j, v_k \in \text{adyacentes}(v_i), e_{ij} \in \mathcal{E}|}{k_i(k_i - 1)}$$

Lo cual quiere decir: por cada par de adyacentes al vértice en cuestión, si existe la arista yendo de uno al otro (si además está la recíproca, lo contamos otra vez). A esa cantidad de aristas lo dividimos por $k_i(k_i - 1)$ siendo k_i el grado de salida del vértice i. En caso de tener menos de 2 adyacentes, se define que el coeficiente de clustering de dicho vértice es 0. Considerar que el coeficiente de clustering es siempre un número entre 0 y 1.

El clustering promedio de toda la red será:

$$C = \frac{1}{n} \sum_{\forall v \in \text{grafo}} C_i$$

Importante: Es importante no contar los bucles que puedan existir al calcular el coeficiente.

- Comando: `clustering`
- Parámetros: `pagina`, opcional.

- Utilidad: Permite obtener el coeficiente de clustering de la página indicada. En caso de no indicar página, se deberá informar el clustering promedio de la red. En ambos casos, informar con hasta 3 dígitos decimales.
- Complejidad: En caso que se indique una página en particular, debe ejecutar en $\mathcal{O}(1)$ (considerando que la red es *muy* dispersa). En caso que no se indique ninguna página, deberá ejecutar en $\mathcal{O}((P+L)^2)$.
- Ejemplo: Entrada:

```
clustering River Plate
clustering Club Atlético River Plate
clustering Juan Domingo Perón
clustering Ámsterdam
# Ejemplo con la red de 5.000 artículos:
clustering
```

Salida:

```
0.000
0.030
0.018
0.065
# Ejemplo con la red de 5.000 artículos:
0.115
```

Entrega

Adicionalmente a los archivos propios del trabajo práctico debe agregarse un archivo `entrega.mk` que contenga la regla `netstats` para generar el ejecutable de dicho programa (sea compilando o los comandos que fueren necesarios). Por ejemplo, teniendo un TP elaborado en Python, podría ser:

```
netstats: netstats.py grafo.py biblioteca.py
    cp netstats.py netstats
    chmod +x netstats
```

Importante: En caso de recibir un error `FileNotFoundError: [Errno 2] No such file or directory: './netstats': './netstats'`, tener en cuenta que para el caso de enviar código escrito en Python es necesario además indicar la ruta del intérprete. Esto puede hacerse agregando como primera línea del archivo principal (en el ejemplo, sería `netstats.py`) la línea: `#!/usr/bin/python3`.

Criterios de aprobación

El código entregado debe ser claro y legible y ajustarse a las especificaciones de la consigna. Debe compilar sin advertencias y correr sin errores de memoria.

La entrega incluye, obligatoriamente, los siguientes archivos de código:

- el código del TDA Grafo programado, y cualquier otro TDA que fuere necesario.
- el código de la solución del TP.

La entrega se realiza en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.