Algoritmos y Programación II (75.41)

Trabajo Práctico N°3

Segundo cuatrimestre 2014

Informe

Fecha: viernes 5 de diciembre

Alumno: Murck, Pablo Martín
Padrón: 96694

Alumno: Ordoñez, Francisco
Padrón: 96478

Ayudante asignado: Pablo Najt

Diseño

El lenguaje de programación elegido para este trabajo practico es Python. Por ser un lenguaje orientado a objetos se trató de tomar el mayor provecho de esta característica, creando clases para definir el comportamiento y los atributos de diferentes objetos.

Diseño de cada archivo

Grafo:

Contiene el TDA Grafo genérico. Se implemento una clase Grafo para representar grafos opcionalmente dirigidos, opcionalmente ponderados y con libertad para asociar datos con vértices y aristas. Su única restricción es la prohibición de aristas paralelas para no complicar el uso del grafo, especialmente porque en grafos de redes (los que se tratarán en este TP) son innecesarios.

Importante del TDA Grafo es la existencia de la clase Vértice. Las modificaciones al grafo se hacen utilizando los métodos del Grafo, pero la búsqueda de información y la manipulación del grafo se hace a través de los Vértices.

El grafo se implementó tomando como base la representación por listas de adyacencia. Específicamente se utiliza un diccionario de diccionarios (técnicamente diccionarios a Vértices, los cuales tienen diccionarios también) para maximizar eficiencia en cuestiones de obtener Vértices, listas de adyacentes y verificación de conexiones.

Tiene excepciones propias que puede llegar a levantar si el usuario lo utiliza de forma inadecuada o no hace cumplir los requisitos del grafo.

BFS: (hace uso de los TDAs Grafo y Cola (Python collections.deque))

La idea del BFS es tener una clase que implementa el recorrido BFS recolectando múltiples informaciones para responder las consultas del usuario sobre el grafo. El recorrido se implementa a partir de un vértice raíz que impondrá el usuario en la inicialización.

Utiliza la estrategia de programación dinámica ya que, en caso de múltiples consultas, las respuestas pueden haber sido resueltas anteriormente, completa- o parcialmente, por lo tanto disminuyendo el tiempo de ejecución. Pero el beneficio que se ve en tiempo no se ve en memoria: se utilizan cuatro diccionarios para guardar la información y una cola para el BFS.

Las consultas que se le pueden hacer a la clase son todos los beneficios que provee BFS en grafos no dirigidos y no pesados: distancias entre vértices, caminos mínimos, vértices y cantidades a diferentes niveles, y un iterador por niveles.

Sistema (de comandos): (hace uso del TDA Grafo)

El Sistema es una clase que implementa el “programa principal” para las consultas. En su inicialización se encarga de crear el grafo a partir del archivo asociado a la red.

Cuando se ejecuta el Sistema (es un método de la clase y técnicamente la única primitiva que debería usar el usuario) este se encarga de leer la entrada del usuario y de hacer las conversiones y comprobaciones necesarias para llamar a las funciones del archivo “Comandos”.

Para llamar a las funciones de “limpieza” que luego llaman a las funciones principales de “Comandos” se utiliza un diccionario con estas funciones como los valores asociados a las claves que son la cadena a esperar del usuario.

El sistema tiene sus propias excepciones, algunas de uso interno (Errores del comando y de los parámetros) y otra (Error del archivo del grafo) que se levanta y debe manejar el usuario del Sistema.

Comandos: (hace uso de BFS y los TDAs Grafo, Cola (Python collections.deque), Heap (Python))

Contiene las funciones principales para los comandos del sistema.
(Ver diseño y análisis de las operaciones, sección Comandos)

TP3: (hace uso del Sistema)

Es el script inicial que se debe ejecutar con la ruta del archivo pasada por parámetro en la linea de comandos. Inicializa y ejecuta el Sistema manejando las excepciones que pueda llegar a levantar este último (errores en el archivo principalmente).

Diseño y Análisis de las operaciones

***N**: cantidad de elementos en un diccionario especifico.*
***K**: cantidad de resultados requeridos en algunos comandos.*
***V**: cantidad de vértices en el grafo.*
***E**: cantidad de aristas en el grafo.*
***Gr(x)**: grado de un vértice. (equivalente a su cantidad de adyacentes)*
Se consideran todas las conversiones y comparaciones como acotadas por una constante, por lo tanto siendo de O(1).

Ejecutar comando:

- Ejecutar función de “limpieza”, valor asociado a la clave esperada: O(1).
- Comprobaciones y, si es necesario, conversiones: O(1).

✳ Es constante.

En espacio se utiliza el diccionario de funciones asociadas a los comandos esperados y el de musico a id de vértice.

Obtener los mayores K de un diccionario de N ítems en orden decreciente:

Se basa en construir un heap de mínimo de los primeros K elementos, recorrer el resto del diccionario y mantener en el heap los mayores elementos. Luego desencolarlos y devolver en orden decreciente. Utilizado en los comandos recomendar y centralidad.

- Crear lista de tuplas a partir del diccionario: O(N).
- Crear heap a partir de los primeros k elementos de la lista: O(K) ya que:

1. Copia de la lista hasta k: O(K).
2. Heapify de la lista: O(K).

- Iterar el resto de la lista y, si es necesario, desencolar y encolar:
  O((N-K)(logK + logK)) = O((N-K)(2logK)) = O((N-K)logK).
- Vaciar el heap en una cola para mantener el orden decreciente: O(KlogK)

- Devolver la lista formada por la cola: O(K)
- ✳ Por lo tanto es: O(N + K + (N-K)logK + KlogK + K) = O(N + K + NlogK)
- ✳ Ya que K <= N, entonces: O(N+NlogK).

En espacio es necesario el heap de mínimo, la lista de tuplas y la cola.

Comandos

Recomendar:

DFS hasta nivel 2 calculando en cuantos arboles DFS diferentes están los del nivel 2 e ir guardando ese cálculo en un diccionario de vértices a nivel 2.

- Iterar los adyacentes de cada vecino del vértice: O(Gr(Vértice)*Gr(Vecino.i))
- Comprobación de nivel del adyacente e incorporación o actualización del diccionario: O(1).
- Seleccionar los mayores K en orden decreciente: O(N+NlogK) con N: Vértices a nivel 2.
- ✳ Por lo tanto es O(Gr(Vértice)*Gr(Vecino.i) + N+NlogK).

En espacio implicaría el diccionario de vértices a nivel 2 y lo necesario para obtener los mayores K (ver arriba).

Difundir:

Para cada difusor iterar los adyacentes e incrementarle su contador de compartidos (en un diccionario), verificar al mismo tiempo para cada adyacente si ya lo pueden compartir. Repetir para todos los difusores originales y para todo otro vértice que lo difunda posteriormente.

- Crear lista de vértices difusores a partir de los ids de los difusores: O(D). D: cant. difusores.
- Para cada difusor iterar sus vecinos y actualizar el diccionario: O(D*Gr(D.i)).
- Verificar al mismo tiempo si pueden compartir la información: O(1).
- Si lo hacen se agregan a la lista de nuevos y se elimina del diccionario: O(1).
- Repetir el proceso con los nuevos difusores: O(X*Gr(X.i) con X: nuevos difusores.
- Si llegan a difundir todos, entonces las repeticiones equivalen a: O(V*Gr(V.i)).
- ✳ Por lo tanto: O(D + V*Gr(V.i))

En espacio se utiliza la lista de nuevos y primeros difusores y además un diccionario de posibles difusores.

Centralidad:

Recorrido BFS de todos los vértices contra todos los otros (sin repetir) para calcular los caminos mínimos incrementando la cantidad de veces por las que se pasa por un vértice especifico en cada camino.

Nota: Por la implementación del grafo con diccionarios, este comando en diferentes ejecuciones puede llevar a resultados fluctuantes. Igualmente, los vértices verdaderamente centrales se mantendrán prácticamente constantes.

- Inicializar un diccionario de pasadas para todos los vértices: O(V).
- Caminos mínimos de todos contra todos, actualizando el diccionario: O(V*(V+E)).

(Gracias a la clase BFS se calcula solamente un árbol BFS → O(V+E) por cada vértice raíz.)

- Seleccionar los mayores K en orden decreciente: O(V+VlogK)

✳ Por lo tanto es O(V + V*(V+E) + V+VlogK) = O(V + V*(V+E) + VlogK)

En espacio se utiliza el diccionario, lo necesario para BFS (múltiples veces) y lo usado para obtener los mayores K.

Camino:

Recorrido BFS de un vértice al otro.

- Calcular camino mínimo usando BFS: O(V+E).

✳ Por lo tanto es O(V+E).

En espacio se utiliza lo necesario para la clase BFS.

Diámetro:

Recorrido BFS de todos los vértices contra todos los otros (sin repetir) para calcular los caminos mínimos manteniendo el camino máximo y actualizando si es necesario.

- Caminos mínimos de todos contra todos, actualizando el máximo: O(V*(V+E)).
  (Gracias a la clase BFS se calcula solamente un árbol BFS → O(V+E) por cada vértice raíz.)

✳ Por lo tanto es O(V*(V+E))

En espacio se utiliza lo necesario para la clase BFS. (múltiples veces)

Agrupamiento:

Calcular coeficiente de clustering de cada vértice y sumarlos para luego dividirlos por la cantidad de vértices del grafo.

Nota: Los vértices con clustering incalculable (1 o ningún vecino) no se cuentan en la suma y se descuentan de la cantidad de vértices totales para el promedio final.

- Calcular clustering para un vértice: O(Gr(Vértice)*Gr(Vecino.i)) ya que:

1. Contar aristas entre vecinos por duplicado: O(Gr(Vértice)*Gr(Vecino.i)).
2. Cálculo del coeficiente: O(1).

- Para todo vértice del grafo calcular el clustering: O(V * (Gr(V.i)*Gr(Vecino.i))).
- Hacer el promedio: O(1).

✳ Por lo tanto es O(V * (Gr(V.i)*Gr(Vecino.i))).

En espacio es constante.

Distancias:

Recorrido BFS por toda la componente conexa en la que se encuentra el vértice. Las distancias son los diferentes niveles del árbol que forma el recorrido BFS.

- BFS: O(V+E).

✳ Por lo tanto es O(V+E).

En espacio se utiliza lo necesario para la clase BFS.

Subgrupos:

Recorrido DFS por cada componente conexa visitando los vértices y contándolos para guardar la cantidad en un diccionario asignándole un identificador del componente.

- DFS contando vértices: O(V+E).
- Devolver el resultado en orden decreciente con sort: O(SlogS). Con S: cant de subgrupos.

✳ Por lo tanto es O(V+E+SlogS).

En espacio se utiliza el diccionario de visitados y de los subgrupos.

Articulación:

Algoritmo de Tarjan visto en clase basado en un recorrido DFS. La comprobación de si un vértice es punto de articulación se hace en el mismo recorrido. Ejecutarlo para cada componente conexa.

- DFS y verificación si un vértice es de articulación: O(V+E).

✳ Por lo tanto es O(V+E).

En espacio se necesitan diccionarios de: visitado, orden, inferior y padre, y aparte una lista donde guardar los puntos de articulación.