Ejercicios Varios :)

CC4102 - Diseño y Análisis de Algoritmos

Profesor: Gonzalo Navarro
Auxiliares: Joshimar Cordova, Jorge Bahamonde
Ayudante: Sebastián Ferrada

1 El Ejercicio Pendiente

Considere un almacenamiento en disco para grafos $G(V, E)$, con $|V| = n$ y $|E| = e$, como una secuencia de pares (i, j) para cada $(i, j) \in E$ en cualquier orden (pero sin permitir repetidos). Llamemos $\texttt{Sort}(e) = (e/B)\log_{M/B}(e/B)$ al costo óptimo de ordenar e elementos en disco, con B el tamaño del bloque y M el tamaño de la memoria.

En el enunciado original faltó lo siguiente: Puede suponer que $M \geq n$, pero no que $M \geq e$.

El problema está *malo*, en el sentido de que la solución tradicional está mala. De todas formas, incluyo cómo era el razonamiento y en qué se cae.

1.1 Obtener G^2

Muestre cómo obtener $G^2(V, E')$ en tiempo $O(\texttt{Sort}(e) + e'/B)$, donde $e' = |E'|$. Una arista $(u, v) \in E'$ si existe $w \in V$ tal que (u, w) y (w, v) están en E.

Respuesta: Consideraremos el grafo como dirigido; el caso de un grafo no dirigido puede simularse copiando todos los arcos en orden inverso, ordenando y escaneando para eliminar duplicados.

La idea es, dado que necesitamos encontrar dos aristas en G para definir una en G^2, tener dos copias de G; luego, dado un par de elementos (u, v) y (w, z) en la primera y segunda copia, respectivamente, chequear si $v = w$; si se cumple, agregar esto a la salida (es decir, G^2). A grandes rasgos, lo que haremos será lo siguiente:

- Crear una copia G' de G.
- Ordenar G y G'.
- Escanear G y G'; si, al ver (u, v) en G y (w, z) en G', $v = w$, escribir (u, z) en la salida.

Esto se ve muy bonito; sin embargo, es necesario que el algoritmo cumpla lo siguiente (además del tiempo de ejecución):

- Todos los elementos necesarios quedan en G^2.
- No hay elementos repetidos en G^2.

¿Cómo podemos estar seguros de que estos dos puntos se cumplan? Va a depender de cómo especifiquemos los pasos de **ordenar** y **escanear**. El primer paso está ambiguo porque es necesario definir qué criterio se usa para el orden de cada archivo. El segundo paso está ambiguo ya que

se escanean dos archivos de forma simultánea: dado que habrá un puntero para cada archivo, es necesario definir claramente cómo avanza cada uno de éstos.

Una forma de ver G^2 es como el grafo que tiene directamente conectados aquellos nodos originalmente conectados mediante un nodo que servía de "puente". Un intento, entonces, para resolver el problema, es intentar identificar estos puentes.

Ordenar: Ordenamos lexicográficamente ambos archivos: en el caso de G, ordenamos los pares (i, j) primero por el *segundo* elemento de cada par y por el *primer* elemento del par en caso de empate. En el caso de G' hacemos lo opuesto (se ordena en primer lugar por el primer elemento). La idea es que al poner G y G' uno al lado del otro se vería algo así:

$$\begin{array}{cc} G & G' \\ (h,a) & (a,b) \\ (x,a) & (a,d) \\ \vdots & \\ (n,z) & (z,f) \end{array}$$

Escanear: En este punto aprovechamos que $M \geq n$. Al escanear, podemos cargar en memoria tanto contenido de G como sea posible: si consideramos $M \in \Omega(n)$ podemos cargar todos los elementos de G cuyo segundo elemento tome un cierto valor (lo mismo para G', con sus segundos elementos); llamaremos a estos elementos un *trozo*. En el ejemplo anterior, podemos estar seguros de ser capaces de cargar todos los pares que tengan a como segunda componente de G, y todos los elementos de G' que tengan a como primera componente. Con esto, podemos determinar (¡en memoria!) todos los pares de vértices que están conectados por a, y pasar al siguiente trozo.

Notemos que si bien esto nos asegura que todos los elementos son entregados en la salida, no nos evita los duplicados.

De todas formas, el algoritmo queda así:

- Mientras queden elementos en G y en G' sin leer:
  - Sean $(x_1, y_1), (x_2, y_2)$ los elementos bajo los punteros de G y G'.
  - Mientras $y_1 < x_2$, avanzar el puntero de G; mientras $y_1 > x_2$; avanzar el puntero de G'.
  - Cuando se llega a $y_1 = x_2 = c$, cargar las líneas de G y G' que cumplan eso (a lo más n entradas de cada archivo).
  - En memoria, determinar (por ej., con un doble loop) todos los pares (x_1, y_2) de las líneas leídas con $y_1 = x_2 = c$ e imprimirlas a un archivo, en orden lexicográfico.
  - Mover los punteros de G y G' a los primeros pares (x_1, y_1) y (x_2, y_2) tales que $y_1 \neq c$ y $x_2 \neq c$.
- Combinar los archivos generados en orden, eliminando duplicados en el proceso.

El primer paso toma $O(e/B)$ lecturas de disco, ya que consiste simplemente en iterar sobre dos copias de G.

Hay dos problemas con esta solución. Primero, si nos restringimos a $e \gg M > n$, no podemos asegurar leer de ambos archivos simultáneamente (sólo nos cabría en memoria, por ejemplo, el trozo de G). En ese caso, se agregaría un factor n/B al escaneo (cargar el trozo de G' por bloques).

Más grave es lo que sucede con los duplicados. En el peor caso (por ejemplo, el grafo completo) todos los arcos se repiten n veces. A nuestro favor, los arcos salen en orden relativo (es decir, los arcos conectados por a estarán en orden; luego los conectados por b estarán en orden, etc), con lo que la eliminación de duplicados puede hacerse a través de un merge n-ario, que es lineal en la entrada. Luego, en el peor caso, esta parte toma $O(n \cdot e/B)$ (el tamaño del archivo de salida dividido en B). Esto es ineficiente, y se había pasado por alto en la formulación de la solución tradicional. Si encuentran una solución que sí alcance la cota, cuéntenme :)

1.2 Obtener G^*

Muestre cómo obtener $G^*(V, E^*)$, la clausura transitiva de G, donde $(u, v) \in E^*$ si existe un camino de u a v en G. Obtenga G^* en tiempo $O(\texttt{Sort}(e)\log n)$, donde $e' = |E'|$.

Respuesta: Suponiendo que se pudiera hacer la parte anterior en el tiempo que se pide, podemos hacer lo siguiente:

- Crear G^2 a partir de G y unirlos mediante un sort + merge (que elimine los repetidos). Esto genera el grafo de "caminos de largo igual o menor a 2".
- Repetir esto $\log n$ veces.

El grafo obtenido representa "los caminos de largo menor o igual a 2^k", que simbolizaremos como G_k, con k el número de iteraciones realizadas. Usando inducción (el caso base ya está enunciado arriba), es necesario mostrar el paso $k \to k + 1$.

Suponiendo que después de k iteraciones se llega a G_k, queremos demostrar que $G_k \cup G_k^2 = G_{k+1}$ Notemos que cualquier camino de largo menor o igual a 2^{k+1} puede ser descompuesto en dos caminos de largo menor o igual a 2^k, que deben entonces estar incluidos en G^k. De esta forma, estarán incluidos en G_k^2 (pues unirá todas las combinaciones de caminos incluidos en G_k). Luego estos caminos estarán en G_{k+1}.

1.3 Reportar caminos

Modifique su algoritmo para reportar los caminos más cortos entre todo par de nodos entre los que exista un camino.

Respuesta: La primera idea aquí es seguir el algoritmo anterior, pero reportar los caminos "nuevos". Esto se puede hacer en el merge. Notemos que en cada paso combinamos G con G^2; los caminos de G^2 Los nuevos caminos siempre estarán en G^2. obviamente. Luego, durante el merge de G y G^2, puede verificarse cada vez que aparece un elemento de G^2 que no esté en G y reportarlo en ese caso.

Queda, sin embargo, reportar el largo del camino. Una idea para esto es anotar los largos de camino: es decir, partir con G almacenando arcos de la forma $(u, v, 1)$; al momento de conectar (u, v, n) y (v, w, m), generar $(u, w, n+m)$. Esto requiere tratar varios puntos (en particular, ordenar según la tercera componente si hay empates en las otras dos) y el manejo de duplicados al hacer merges (no considerar la tercera componente para el concepto de duplicados). Con esto, reportar el largo del camino más corto no es complicado.

2 Problemas: Cotas Inferiores

1. Demuestre que determinar si un grafo no dirigido es o no conexo toma, en el peor caso, $\binom{n}{2}$ consultas del tipo "¿existe un arco entre los vértices u y v?", si la cantidad total de vértices es n.

2. Sea T un árbol ternario perfecto de altura l. Suponga que cada hoja está etiquetada con un bit, y que cada nodo interno debe ser etiquetado con un bit cuyo valor corresponda al de la *mayoría* de las etiquetas de sus hijos. Suponga que, en un principio, sólo las hojas están etiquetadas. Demuestre que todo algoritmo determinístico que encuentre el valor de la raíz debe examinar todas las hojas en el peor caso.

3. El problema de *búsqueda aproximada en texto* consiste en, dado un string $T[1, n]$, llamado *texto*, otro string más corto $P[1, m]$, llamado patrón, y un entero $0 \leq k < m$, determinar si P ocurre en T permitiendo a lo más k *errores* (inserciones, borrados o reemplazos de caracteres en P (o T)). Ambos T y P son secuencias de caracteres de un alfabeto $[1, \sigma]$.

   Yao demostró en 1976 que no es posible resolver el problema de búsqueda exacta ($k = 0$) en menos de $\Omega(n\log_\sigma(m)/m)$ inspecciones de caracteres de T *en promedio* (el promedio supone que los caracteres de T se eligen al azar, uniformemente en $[1, \sigma]$).

(a) Demuestre que un adversario impide resolver el problema general inspeccionando menos de $k+1$ caracteres en cualquier ventana posible de T de largo m.

(b) Deduzca de lo anterior una cota inferior de la forma $\Omega(kn/m)$ para el problema, incluso en el caso promedio.

(c) Deduzca la cota para el problema $\Omega(n(k+\log_\sigma m)/m)$ en promedio, demostrada por Chang y Marr en 1994. Ellos también diseñaron un algoritmo con esa complejidad promedio, $O(n(k+\log_\sigma m)/m)$. ¿Qué puede decir entonces de la complejidad promedio del problema?

3 Problemas: Memoria Secundaria

1. Dada una relación binaria $\mathcal{R} \subset \{1,...,N\} \times \{1,...,N\}$, representada como un archivo secuencial de sus pares (i,j), construya algoritmos tan eficientes como le sea posible (considerando que $N \gg M$) para determinar si la relación es:

   (a) Refleja

   (b) Transitiva

   (c) Antisimétrica

2. Considere una lista de N elementos, y una memoria principal de tamaño $M \in \Theta(1)$. Diseñe algoritmos eficientes para:

   - Encontrar un elemento que tenga la mayoría absoluta (es decir, que aparezca más de $\frac{N}{2}$ veces), y reportar su frecuencia. Si no existe tal elemento, debe reportarse `no`.
   - Encontrar k elementos que aparezcan más de $\frac{N}{k+1}$ veces, con $k < M$, suponiendo que existen.

3. Suponga se tiene un archivo que contiene una secuencia de números positivos y negativos $a_1,...,a_N$. Describa un algoritmo que en $O(\frac{N}{B})$ operaciones de disco encuentre j_1, j_2 tales que la suma $\sum_{i=j_1}^{j_2} a_i$ alcanza su máximo valor.

4. Se nos entrega un conjunto P de N puntos en $\mathbb{R}^d$ en *posición general*; es decir, ningún par de puntos comparte el mismo valor en alguna dimensión. Considere que $N \gg M$. Dado un punto $p \in \mathbb{R}^d$, llamaremos $p[i]$ a su i-ésima coordenada. Un punto p_1 *domina* a otro punto p_2 (denotado como $p_1 \prec p_2$) si se cumple $p_1[i] < p_2[i] \forall i = 1,...,d$.

   Se nos pide calcular el *skyline* de P, denotado como $SKY(P)$, que incluye todos los puntos de P que no son dominados por ningún otro:

   $$SKY(P) = \{p \in P |\ \nexists p' \in P, p' \prec p\}$$

   (a) Resuelva el problema para $d = 2$ usando $O(\frac{N}{B}\log_{M/B}\frac{N}{B}))$ operaciones de disco.

   (b) Resuelva el problema, ahora para $d = 3$. Llegue a la misma cota para la cantidad de operaciones de disco que en el caso anterior.

4 Problemas: Propuestos anteriores

1. En este problema daremos una cota inferior de $\Omega(n\log n)$ para el problema de determinar si existe un par de elementos idénticos en un conjunto de n, en el modelo de comparaciones.

   (a) Suponga que los elementos son $a_0 \leq a_1 ... \leq a_{n-1}$. Definimos las comparaciones *adyacentes* como las comparaciones entre a_i y a_{i+1}, para cada $i \in [0,n]$. Muestre, utilizando un argumento de adversario, que cualquier algoritmo que determine si existe un par de elementos idénticos debe realizar todas las comparaciones adyacentes.

(b) Muestre que para problema de ordenar n elementos $a_0 \leq a_1 ... \leq a_{n-1}$ es suficiente conocer el resultado de las comparaciones adyacentes. Es decir, muestre que existe un algoritmo que recibe n elementos, un conjunto de resultados (representados, por ejemplo, de la forma $(a, b, v), v \in \{-1, 0, 1\}$) de comparaciones que incluya todas las adyacentes, y que ordena los elementos.

(c) Concluya, notando que el paso anterior es una forma de *reducción*.

2. Se tienen dos arreglos A y B de largo n almacenados en memoria secundaria. Si bien son diferentes, ambos contienen los enteros entre 1 y n. Se desea construir el arreglo C, dado por $C[i] = A[B[i]]$. Diseñe un algoritmo eficiente para esto, y analícelo.

3. Dadas dos listas crecientes de largos m y n, y suponiendo que $m < n$, diseñe y analice un algoritmo para unirlas en una única lista ordenada, que tome tiempo $O(m + m\log \frac{n}{m})$.

Hints

2.1.a El adversario mantiene dos grafos: uno conexo y uno no conexo. Partan del grafo vacío y el grafo completo.

3.4.a Partan por ordenar los puntos de P por una de sus coordenadas.

3.4.b De nuevo, partan por ordenar los puntos de P por una de sus coordenadas. Luego piensen en un algoritmo recursivo en el que el caso base se da si $N \leq M$. En el caso recursivo, particionen la lista (ordenada) de puntos, encuentren $SKY(P_i)$ para cada conjunto de la partición y construyan $SKY(P)$ a partir de estos resultados.

4.1.a Si a_k y a_{k+1} no se comparan, el adversario podría establecer otra relación entre éstos...

4.1.b Una forma es ver los elementos como vértices en un grafo, y un arco que va de a a b si una comparación determina que $a < b$. ¿Qué pueden hacer con el grafo si tienen todas las comparaciones adyacentes?

4.2 Si dado un arreglo con elementos a_i ordeno el arreglo de elementos (i, a_i) según su segunda componente, ¿qué queda en cada componente? Vean los arreglos como *permutaciones*. Si ordenar es aplicar una permutación a los elementos, ¿qué estamos haciendo al ordenar una permutación?

4.3 Consideren dividir la lista de largo n en trozos de largo $\frac{n}{m}$.