Tarea #1 - Cálculo de intersecciones en Disco

23 de septiembre de 2020 - Gonzalo Navarro & Bernardo Subercaseaux

En esta tarea experimentarán con algoritmos para intersectar listas en memoria secundaria. El objetivo de la tarea es estudiar en la práctica la validez del modelo de memoria secundaria visto en clases, pero también aprender a experimentar y elaborar informes. Este enunciado, a diferencia de los enunciados posteriores, explica en detalle lo que se espera de un informe.

Como resultado de la tarea deberán entregar un informe y todo el código utilizado. La corrección estará centrada en el informe. Por medio de U-Cursos se le entregará un template de base para la elaboración de los informes. El informe NO debe responder directamente a los problemas planteados en este enunciado en formato *Respuesta 1*, *Respuesta 2*, *Respuesta 3*, ..., sino que debe escribirse como un artículo breve, que comienza con un abstract (describe el tema tratado en la tarea y resume los resultados y conclusiones obtenidos) y en cuyo desarrollo se encuentran respondidas todas las preguntas planteadas en el enunciado.

1. Problema a tratar - Intersección de listas

Dadas dos listas, P y T de enteros de 32 bits, se desea calcular $P \cap T$, es decir, cuáles elementos de P están también en T. Como motivación, este tipo de problemas es vital en la implementación de consultas a bases de datos, en particular en consultas con joins.

Trabajaremos en un contexto particular con las siguientes hipótesis:

1. $|P|$ es mucho menor que $|T|$.
2. P cabe en memoria RAM (sin embargo debe leerse desde un archivo).
3. T está ordenado.

Considere por simplicidad que en los archivos de P y T cada entero va en su propia línea, y tiene siempre 9 dígitos, con posibles ceros a la izquierda.

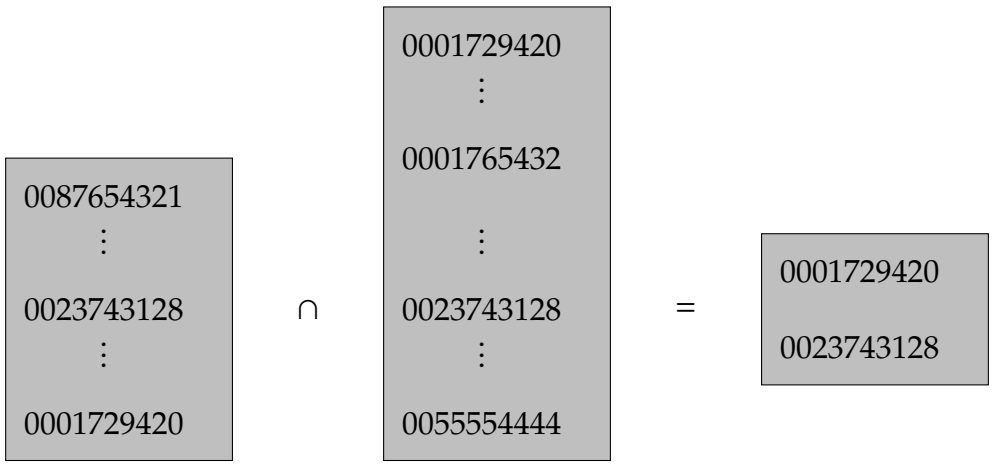

Figura 1: Intersección de archivos `P.txt` y `T.txt`, resultando en `output.txt`

Se estudiarán tres estrategias. A continuación se describe la idea general de cada una.

- (**Búsqueda binaria**) Se leen los elementos de P uno a uno, y por cada uno se realiza una búsqueda binaria en T (que está ordenado). Los elementos encontrados se van guardando en un arreglo, y finalmente se escriben en el archivo de salida.
- (**Búsqueda lineal**) Se cargan todos los elementos de P en memoria principal. Luego se lee T linealmente, cargando de a bloques (de tamaño B), y por cada elemento cargado se revisa si está en P (que ya se ha cargado a RAM).
- (**Búsqueda indexada**) Inicialmente se construye en memoria principal un arreglo S (que podemos interpretar como un *índice*), cuyos elementos son los primeros elementos de cada bloque de T. Es decir $S[i] \coloneqq T[i \cdot B]$ para $i = 0, \ldots, |T|/B - 1$. Esto asume que $|T|/B \leq M$ (siendo M el tamaño de la memoria principal). Luego se carga P en memoria tal como en la estrategia anterior, y por cada elemento en P se hace búsqueda binaria en S, para identificar en qué bloque de T se encuentra. Una vez hallado el bloque, se lee ese bloque de T para verificar si el elemento deseado se encuentra en T o no.

1. (0.5 pts.) Escriba el pseudo-código de las estrategias descritas, considerando una implementación en memoria principal (es decir, tratando P y T como si fueran arreglos). Utilice un paquete de LaTeXcomo algorithmicx para escribir el pseudo-código en su informe, utilizando un nivel de detalle similar al del enlace anterior.

2. (0.5 pts.) Argumente las siguientes complejidades de peor caso en términos de la cantidad de operaciones de I/O.

- (**Búsqueda binaria**) $O(|P| \log |T|)$
- (**Búsqueda lineal**) $O(|T|/B)$
- (**Búsqueda indexada**) $O(|P| + |T|/B)$

3. (1.5 pts.) Programe cada estrategia para memoria secundaria utilizando el lenguaje Python ($\geq$ 3.6). Su código debe poder ejecutarse de la siguiente forma:

```
> python binarySearch.py   P.txt T.txt
> python linearSearch.py   P.txt T.txt
> python indexedSearch.py  P.txt T.txt
```

Resultando en un archivo `output.txt` que contiene la intersección de ambos archivos, un entero por línea, cada entero usando 9 dígitos. En clase auxiliar se darán detalles sobre cómo programar en memoria secundaria.

2. Hipotésis

Habiendo programado las distintas estrategias, es momento de plantear hipótesis sobre su rendimiento. En particular, deberá plantear hipótesis en función del rango de los parámetros. Por ejemplo, *si* $|P| < |T|/7$, *la variante indexada será más eficiente*. Sus hipótesis deberán considerar 2 variables; el tiempo de ejecución y el número de operaciones de I/O realizadas.

	# de comparaciones	Tiempo de ejecución
Caso $\|P\| > \|T\|/5$	Las estrategias se comportan de manera similar.	La estrategia indexada es la más rápida, y la búsqueda binaria es la más lenta.
Caso $\|P\| < \|T\|/5$	La estrategia de búsqueda binaria realiza menos operaciones de I/Os.	Las estrategias se comportan de manera similar.

4. (0.3 pts.) Describa sus hipótesis sobre el rendimiento de las distintas estrategias para el problema de intersección (apartado 3). Utilice una tabla, como se muestra a continuación (a modo de ejemplo),

y **explique el razonamiento detrás de sus hipótesis**. No es necesario que las hipótesis sean correctas, solo que sean serias y honestas. Table Generator puede serle útil para generar tablas en LaTeX. Quizás le sea útil la siguiente referencia sobre control de ancho para tablas.

3. Consideraciones para experimentar en memoria secundaria

Para experimentar, deberán simular tener una memoria principal pequeña, de tamaño $M = 5 \cdot 10^6$. Esto quiere decir que en su código no deberá tener en memoria arreglos de tamaño mayor a M, pero por simplicidad no debe preocuparse por el espacio que gastan las variables auxiliares. Simularán además un tamaño de bloque $B = 500$. Para esto bastará con que cada vez que lea o escriba en un archivo, lo haga de a B bytes cada vez. Note que, convenientemente, dado que cada línea de los archivos tiene 10 bytes (9 dígitos + 1 salto de línea), cada lectura permite leer exactamente 50 líneas.

Para medir los tiempos de ejecución puede utilizar el módulo `time` de Python. La siguiente referencia discute formas elegantes de medir el tiempo de funciones de Python. Para contar el número de operaciones de I/Os puede simplemente llevar una variable de cuenta, a la cual suma 1 al momento de hacer una lectura/escritura. Puede ser cómodo mantener una variable global de cuenta y abstraer las lecturas/escrituras a una función que además de ejecutarlas incrementa la cuenta global.

4. Diseño Experimental

Habiendo realizado las hipótesis, el objetivo es diseñar una serie de experimentos que permitan ya sea refutar, verificar o al menos convencerse parcialmente de las hipótesis planteadas.

4.1. Generación aleatoria de instancias

Primero consideremos la generación de P. Llamemos $n = |P|$. Para cada valor de n, podemos definir una variable aleatoria D_n, que será una de las posibles instancias aleatorias de tamaño n. En esta tarea tomaremos $D_n = (U_1, \ldots, U_n)$, donde cada U_i es una variable aleatoria independiente que toma valores entre 1 y 10^9 con probabilidad uniforme. En palabras simples, cada elemento de P se elegirá uniforme e independientemente como un entero en $[1, 10^9]$.

La generación de T tiene una dificultad adicional, ya que debe ser un archivo ordenado. Para generar T puede generar cada elemento al azar uniforme e independiente (tal como en P), y luego ordenar el archivo usando la función sort de linux.

5. (0.3 pts.) Programe un generador de instancias aleatorias, que se ejecute recibiendo $|P|$ y $|T|$, y genere los archivos `P.txt` y `T.txt` (resultando en el archivo de T ya ordenado).

```
> python generator.py  <tamaño de P> <tamaño de T>
```

4.2. Experimentos

Recordemos que para un algoritmo $\mathcal{A}$ y una instancia I podemos llamar $c(\mathcal{A}, I)$ al costo (en este caso número de I/Os o tiempo I/Os) de ejecutar $\mathcal{A}$ sobre la instancia I. Cuando el tamaño de las instancias está fijo, digamos n, podemos definir $Q_{k,n}$ como el costo promedio sobre k instancias de tamaño n generadas aleatoriamente según el procedimiento descrito anteriormente. Sea $S_{k,n}$ un conjunto de tamaño k, cuyos elementos son instancias de tamaño n generadas aleatoriamente. Entonces podemos escribir

$$\text{(Caso promedio experimental)} \quad Q_{k,n}(\mathcal{A}) = \left(\sum_{I \in S_{k,n}} c(\mathcal{A}, I) \right) / k$$

En nuestro problema, sin embargo, n está compuesto de dos variables: el largo de P y el largo de T. Para simplificar la siguiente pregunta asumiremos $|P| = 10^6$ y $|T| = 10^8$, por lo que la variable Q dependerán solamente de k, ya que el tamaño de las instancias está fijo.

6. (0.3 pts.) El valor de k influirá en la varianza de la variables aleatoria Q; Entre mayor sea k, menor debiese ser la varianza. Experimente valores Q_k variando k entre 10 y 1000. Grafique Q_k usando barras de error para representar la desviación estándar. El paquete pgfplots le puede ser útil para graficar.

7. (0.5 pts.) Describa valores de $|P|$, $|T|$ y k con los que intentará verificar las hipótesis planteadas sobre los distintos algoritmos. Incluya en su informe (por más que repita lo escrito en este enunciado) la forma de generar las instancias aleatorias. (Hint: utilice un k fijo y pequeño, que parezca no generar tanta varianza).

5. Resultados

8. (0.5 pts.) Experimente sobre $Q_{|P|,|T|,k}$, según los valores de $|P|$ y $|T$ diseñados en el apartado anterior, para cada estrategia.

Tras correr los experimentos, deberá incluir los resultados a través de gráficos y texto que los explique.

9. (0.5 pts.) Grafique claramente los resultados de sus experimentos. No debería incluir más de 6 gráficos (seleccione los que mejor manifiestan las ideas que quiere comunicar) en el cuerpo de su informe, pero puede incluir gráficos menos importantes en el anexo. (Hint: Puede serle útil definir una variable $\rho = |T|/|P|$).

Evite referirse en términos de $Q_{n,k}$ en su informe. Opte en cambio por *Rendimiento promedio de la estrategia X*.

10. (0.2 pts.) Contraste los resultados obtenidos con las hipótesis planteadas.

6. Variantes

Una forma de entender qué parámetros y variables de un problema son realmente relevantes es estudiar variantes y ver cuánto efecto tienen en los resultados. En este apartado estudiaremos la relevancia de la implementación específica de la estrategia de búsqueda lineal. A continuación se describen 2 variantes de la estrategia de *Búsqueda Lineal*.

- (**Búsqueda lineal + búsqueda binaria**) Tal como en la búsqueda lineal inicialmente presentada, se comienza cargando P en memoria principal. Ahora se ordena P en memoria principal. Luego al leer T linealmente por bloques, cada elemento de un bloque leído se busca con búsqueda binaria en P (dado que P está ordenado).
- (**Búsqueda lineal + merge**) Idéntica a la estrategia anterior, solo que en lugar de buscar los elementos del bloque leído de T con búsqueda binaria en P, se realiza una etapa de *merge* (dado que tanto el bloque leído de T como P están ordenados) para buscar los elementos comunes. Para encontrar la intersección entre dos listas ya ordenadas por medio de una etapa de merge, se mantiene un puntero a cada una, inicialmente al inicio de cada lista. En cada paso si los elementos apuntados son iguales, se añade ese elemento a la intersección, y si difieren se avanza el puntero que apunte al menor elemento. La etapa acaba cuando en una lista el puntero llega al final.

11. (0.3 pts.) Argumente las siguientes complejidades en el peor caso en términos de *operaciones en CPU* para las diferentes variantes de la estrategia de búsqueda lineal.

- (**Búsqueda lineal inicial**) $O(|T||P|)$
- (**Búsqueda lineal + búsqueda binaria**) $O\left((|T|/B)|P|\log B\right)$
- (**Búsqueda lineal + merge**) $O(|P|\log|P| + |T| + |T||P|/B)$

12. (0.3 pts.) Compare experimentalmente las tres variantes, presentando un gráfico al respecto. ¿Alguna de las diferentes variantes para la segunda estrategia genera diferencias en la comparación de las tres estrategias?

13. (0.3 pts.) Imagine que en lugar de tener simplemente P y T, tuviese un archivo grande T y muchos archivos pequeños $P_1, P_2, \ldots, P_\ell$. ¿Qué ventaja proporciona la estrategia indexada en este caso? Compare su complejidad en el peor caso, en términos de I/Os, con la de la búsqueda lineal.

7. Conclusión

Tras contrastar los resultados con las hipótesis planteadas, concluya sobre los resultados obtenidos. Una conclusión no consiste en repetir lo ocurrido, sino en darle sentido y perspectiva. Idealmente la conclusión hace sentido a su lector *después* de haber leído el resto del documento.