Laboratorio 1 de Diseño y análisis de algoritmos

Leandro Rabindranath León

1. Introducción

1.1. Cálculo y suma de los m menores números de un conjunto de n elementos

En este laboratorio calcularás los m menores números de un conjunto que contiene n números. Por ejemplo, para el conjunto $N = \{71, 55, 74, 66, 62, 93, 80, 16, 84, 28\}$ de $n = 10$ números. Los tres menores números son $\{16, 28, 55\}$.

A efectos de validar el laboratorio, mantendrás la suma en línea de los m menores números. Por ejemplo, para el conjunto anterior N, luego de leer los $n = 3$ primeros números, se tiene:

```
{ 55 71 74 } = 200
```

Es decir, $\{55, 71, 74\}$ son los tres menores actuales en línea y la suma en línea es 200. Después cuando se lee el 66 debemos sacar el 74 del conjunto de los menores y meter el 66. Así, después de leído el 66 tenemos:

```
out 74 in 66 = { 55 66 71 } = 266
```

La suma en línea es $200 + 66 = 266$; es decir, la suma anterior más el número ingresado en el conjunto de los tres menores leídos. Análogamente, cuando leemos el 62, ocurre:

```
out 71 in 62 = { 55 62 66 } = 328
```

O sea, sale el 71, entra el 62 y se actualiza la suma a 328.

Cuando se lee los siguientes 93 y 80 estos se ignoran pues ambos son mayores que los tres menores vistos hasta el presente.

El resto de las iteraciones se muestra como:

```
out 66 in 16 = { 16 55 62 } = 344
out 62 in 28 = { 16 28 55 } = 372
```

Al terminar de leer los 10 números tenemos el conjunto $\{16, 28, 55\}$ y el valor de 372 como la suma en línea final. Adicionalmente, la suma de los tres menores números es 99

1.2. Cálculo y suma de los m mayores números de un conjunto de n elementos

El mismo razonamiento expresado para los m menores puede plantearse para los m mayores. Así, para el conjunto ejemplo anterior N, lo siguiente muestra las iteraciones:

```
{ 55 71 74 } = 200
out 66 in 66 = { 66 71 74 } = 266
out 71 in 93 = { 71 74 93 } = 359
out 74 in 80 = { 74 80 93 } = 439
out 80 in 84 = { 80 84 93 } = 523
```

La suma en línea e inclusive la suma de los m mayores puede desbordarse. No te preocupes por ello. Si ello ocurriese, nuestra solución también se desbordaría y sería la misma.

1.3. La plantilla

Te será provisto el código fuente de un programa llamado `heap.C`.

También te provee un simple `Makefile`. Edítalo y revisa y eventualmente modifica las variables `CLANG` y `ALEPH` según tu instalación.

1.3.1. El tipo `SumExtremes`

`heap.C` contiene la siguiente especificación de clase:

```
struct SumExtremes
{
  DynList<Ulong> extremes;
  long long sum;

  SumExtremes() {}

  SumExtremes(DynList<Ulong> && items, const long long __sum)
    : extremes(std::move(items)), sum(__sum)
  { }
};
```

`SumExtremes` es un par consistente de una lista de los m menores (o mayores) números leídos y del valor final de la suma en línea.

1.3.2. La rutina `compute_extremes()`

Tu trabajo en este laboratorio es programar la rutina:

```
std::pair<SumExtremes, SumExtremes>
compute_extremes(ifstream & in, const size_t num_lines, const size_t m)
```

`compute_extremes()` está parcialmente implementada; sólo la lectura del archivo.

`compute_extremes()` lee del archivo abierto `in` `num_lines` números y va calculando en línea las sumas de los `m` menores y los `m` mayores leídos. La rutina retorna un par de objetos de tipo `SumExtremes`, donde el atributo `first` corresponde al los `m` menores números y `second` a los `m` mayores (recuerda que `SumExtremes` contiene una lista y el valor de la suma en línea.)

1.4. El programa principal

La sintaxis del programa es la siguiente:

```
./heap input-file n m
```

Donde

- `input-file` es el nombre del archivo de entrada.
- `n` es la cantidad de números que debes leer del archivo.
- `m` es el tamaño de los conjuntos en línea de los menores y mayores leídos y cuyas sumas en línea vas a calcular.

El archivo de entrada contiene una secuencia arbitraria de números aleatorios. Cada número está en una línea distinta. Se garantiza que el parámetro `num-lines` es menor o igual a la cantidad de números (o líneas) que contiene el archivo. También se garantiza que `num_lines` $<$ `m`.

El programa que se te provee no contiene validaciones para la línea de comandos. Puedes suponer que la línea estará completa y los valores correctos. **No olvides que la línea de comandos debe contener siempre 4 parámetros (incluido el nombre del programa)**.

1.5. El formato de salida

A condición de que la rutina `compute_extremes()` esté correcta, la salida del programa ya está escrita por ti. Dicho de otro modo, **no tienes que modificar la función** `main()`.

En el ejemplo anterior un ejemplo de invocación podría ser:

```
2049,0$ ./heap rang-10.txt 10 3
```

Cuya salida sería:

```
Smallers            = 16 28 55
sum                 = 99
online sum smallers = 372
Biggers             = 80 84 93
sum                 = 257
online sum biggers  = 523
```

La primera línea imprime los `m` menores números de entre todos los `num_lines` números leídos del archivo[1]. La segunda línea es la suma de los anteriores `m` menores. La tercera línea es el valor final de la suma de los `m` menores en línea. La cuarta línea son los `m` mayores de entre todos los `num_lines` números leídos del archivo. La quinta línea es la suma de los `m` anteriores mayores. Finalmente, la última línea es el valor final de la suma en línea de los `m` mayores.

2. Comentarios sobre la solución

Hay varias maneras de resolver este problema, variando desde ordenamiento, árboles binarios de búsqueda y heaps.

2.1. Heaps

En la opinión de este suscrito, la manera más eficiente y, después de entendido, más simple de resolver este problema es mediante el uso de heaps.

Para el caso de los m mayores, puedes mantener un heap limitado a m números. Cuando lees un nuevo número, lo comparas con el menor entre los m mayores leídos (Â¡que están en el heap!). Según el resultado de esta comparación debes decidir si actualizas el heap y la suma en línea o lo ignoras.

Con esta técnica puedes resolver el problema en $\mathcal{O}(n \lg m)$ y con un consumo de espacio de $\mathcal{O}(m)$.

$\aleph_\omega$ (Aleph-ω) tiene tres implementaciones de heaps. Probablemente la más rápida sea la basada en arreglos estáticos, la cual está contenida en `tpl_arrayHeap.H`, pero no debe haber mucha diferencia de desempeño entre una y otra.

2.2. Otros enfoques

El enfoque de solución cándido es ordenar los números y mantener una secuencia ordenada de m números. Igual que con el heap, cada vez que lees un número debes decidir si meterlo en el conjunto de los m o ignorarlo. No obstante, es importante que notes que este método requiere insertar en una secuencia ordenada, lo que es $\mathcal{O}(m)$. Por consiguiente, podría ser muy lento si m es grande.

Otro enfoque más sofisticado, cuyo rendimiento en tiempo y espacio también son $\mathcal{O}(n \lg m)$ y $\mathcal{O}(m)$, respectivamente, consiste en guardar los m elementos en un árbol binario de búsqueda, preferiblemente equilibrado. Puedes usar este enfoque si te parece más cómodo. Sin embargo, se te advierte que probablemente -aún no se ha probado- sea más lento que usar heaps, pues conocer el menor o mayor elemento con árboles te consumirá $\mathcal{O}(\lg m)$, mientras que con un heap será $\mathcal{O}(1)$. Además, el heap es un árbol perfectamente equilibrado, mientras que los árboles binarios de búsqueda no.

[1] En este ejemplo `m` = 3.

2.3. Ayudas

2.3.1. Heaps de $\aleph_\omega$ (Aleph-ω)

$\aleph_\omega$ (Aleph-ω) tiene tres heaps: `ArrayHeap`, `DynArrayHeap` y `DynBinHeap`. Cualquiera de los tres te puede servir. Este suscrito recomienda usar `ArrayHeap`.

2.3.2. Heap inverso

Si usas un heap para calcular los m menores, entonces las prioridades deben estar invertidas; es decir, la raíz debe contener el mayor de todos los elementos del heap. Hay dos maneras de hacer esto. La primera es negar el valor antes de insertarlo. La segunda es especificar que la comparación es "mayor que". Por ejemplo, para `DynBinHeap`, esto se declararía del siguiente modo:

```
DynBinHeap<Ulong, std::greater<Ulong>> heap;
```

Parecido con con los otros heaps de $\aleph_\omega$ (Aleph-ω).

2.3.3. Árboles binarios

Si optas por resolver el problema con árboles binarios, entonces se te recomienda el tipo `DynSetTree`, cuya sintaxis de declaración podría ser la siguiente:

```
DynSetTree<Ulong, Treap> tree;
```

Que especifica un conjunto implementado con árboles binarios de búsqueda de tipo Treap. Si deseas usar AVL, entonces puedes declararlo así:

```
DynSetTree<Ulong, Avl_Tree> tree;
```

3. Evaluación

La fecha de entrega de este laboratorio es el viernes 22 de julio de 2016 hasta las 6 pm. Puedes entregar antes, recibir corrección e intentar hasta un máximo de tres intentos. Se te adjudicará la nota del último intento.

Para evaluarte debes enviar el archivo `read-sum.C` a la dirección:

```
leandro.r.leon@gmail.com
```

Antes de la fecha y hora de expiración. El "subject" debe **obligatoria y exclusivamente** contener el texto **AYDA-LAB-01**. Si fallas con el subject entonces probablemente tu laboratorio no será evaluado, pues el mecanismo automatizado de filtrado no podrá detectar tu trabajo. **No comprimas `test.H` y sólo envía ese archivo**. La nota que se te adjudicará será la de tu último intento.

En el cuerpo del mensaje pon los nombres, apellidos y cédula de la pareja que somete.

Por favor, cuando sometas varias veces, no hagas reply de un correo previo, pues el sistema lo asume como un hilo.

En el mensaje y en el inicio del fuente del programa debes colocar los nombres, apellidos y números de cédula de la pareja que somete el laboratorio. **Por favor, no uses otros medios para enviar la solución ni escribas otros comentarios adicionales en el email**.

La evaluación se distribuirá en cinco partes.

Tarea	Puntuación
Cálculo de los m menores	15 %
Suma en línea de los m menores	25 %
Cálculo de los m mayores	15 %
Suma en línea de los m mayores	25 %
Desempeño	20 %

4. Recomendaciones

1. Lee y estudia el fuente que se te provee.

2. Resuelve un problema a la vez. Por ejemplo, primero calcula los m mayores. No avances hasta estar seguro de que tu solución esté correcta. Luego resuelve la suma en línea. Cuando estés seguro de está correcta, pasa a resolver los m menores y luego su suma en línea.

3. Calcular los m mayores podría ser más fácil porque basta con un heap tradicional. En cambio, los m menores requieren que las prioridades del heaps estén invertidas.

4. Usa el foro para plantear tus dudas de comprensión. Tienes plena libertad para discutir algoritmos, diseño y otras consideraciones, pero de ninguna manera compartas código, pues es considerado **plagio**.

   Por favor, **no uses el correo electrónico ni el correo del facebook para plantear tus dudas. ¡Usa el foro!**

5. No te preocupes por el desempeño hasta no haber resuelto completamente el problema y tener una solución completamente operativa.

   Cuando tengas resuelto el problema, pregúntate ¿cómo lo puedo hacer más rápido?

6. Usa invariantes para verificar que tus suposiciones son correctas y se están cumpliendo.

7. Resuelve algunos ejemplos tú mismo con entradas pequeñas y números manejables. Por ejemplo, para $n = 20, 30, 50$ y $m = 2, 4, 7, 11$. Usa los archivos de prueba que se proveen. Aparte de que esto te ayudará a encontrar la solución te planteará tus casos de prueba.

8. No modifiques el `main()`, pues te arriesgas a distorsionar la salida. Y en general remítete sólo a programar `compute_extremes()` .

9. Usa el tipo `unsigned long` o `Ulong` (sinónimo) para leer los números del archivo. Usa `unsigned long long` o `U64` (sinónimo) para calcular las sumas en línea. De esa manera mitigarás el desborde.