Laboratorio 2 de Programación 3

Leandro Rabindranath León

Este laboratorio es evaluado y la calificación que obtengas cuenta para tu calificación definitiva.

1. Introducción

El fin de este laboratorio es aprender acerca del método de ordenamiento quicksort, especialmente la criticidad de seleccionar un buen pivote.

Para medir las bondades de selección del pivote, "contaremos" la cantidad total de comparaciones y el número de intercambios que se efectúan según algunos criterios de selección del pivote.

Las comparaciones que nos conciernen son las que ocurren entre los elementos de la secuencia que deseamos ordenar. Por ejemplo, dado que `(` a[]) es un arreglo, entonces en la siguiente sentencia ocurre una comparación:

```
if (a[i] < a[p])
          // ...
```

Nota que este tipo de comparación es de entrada más costosa que:

```
if (i < p)
          // ...
```

Pues, aparte de cada acceso `a[i]` requiere un cálculo de dirección, también se requiere cuando menos un acceso adicional a memoria (primero `i`, luego `a[i]`). Cuanto más complejos sean los datos que se almacenen el arreglo, más costosa es su comparación. Por ejemplo, un arreglo de cadenas de caracteres. Si crees que las cadenas no se usan mucho, entonces nunca vayas a programar web.

El número de intercambios (`swap()`) también es importante de conocer porque representan escrituras en memoria, las cuales tienden a ser más costosas que las lecturas.

Planteado lo anterior, debe estarte claro que cuantas menos comparaciones e intercambios efectúe un método de ordenamiento mejor tiende a ser.

En este laboratorio estudiaremos los siguientes criterios de selección del pivote:

1. El elemento más a la izquierda; o sea, el indizado por `l`.
2. El elemento más a la derecha; o sea, el indizado por `r`.

3. Selección del pivote al azar.

4. La mediana entre los extremos `l`, `r` y el centro $m = \frac{l+r}{2}$

5. La mediana entre `Num_Samples` elementos seleccionados al azar.

Aparte de lo anterior, requieres implementar la partición genérica que sea independiente del método de selección del pivote y el quicksort mismo.

1.1. Selección del pivote

Todo procedimiento para seleccionar el pivote tiene el siguiente prototipo:

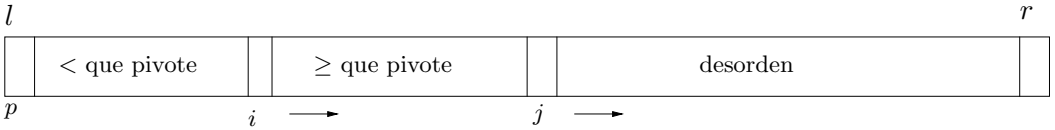
```
tuple<size_t, size_t>
select_pivot(const DynArray<long> & a, const size_t l, const size_t r)
```

La rutina selecciona un pivote en el arreglo `a[]` entre los índices `l` y `r`. El valor de retorno es una tupla cuyo primer elemento es la cantidad de comparaciones y segundo la cantidad de escrituras.

Un `swap(x, y)` se considera como tres (3) escrituras. Así, cada vez que en el quicksort se invoque a `swap()` se contarán 3 escrituras. Por simplicidad, contabilizaremos la cantidad de swaps.

1.2. Partición

En este laboratorio deberás implementar la partición poniendo el pivote en la posición `l`, el cual es el segundo método explicado en clase y puede pictorizarse del siguiente modo:



Como dijimos, el pivote debe encontrarse en la posición l. El proceso debe garantizar que los elementos a la izquierda de i (excepto l) sean menores que p (el pivote), mientras que los elementos entre i i j son mayores o iguales que el pivote. Entre j y r el arreglo aún no ha sido "visto" y por tanto se considera no particionado. Al final, cuando $j = r + 1$, se ejecuta un `swap()` entre l e $i - 1$, lo cual deja al arreglo definitivamente particionado.

Estudia cuidadosamente este método de partición y asegúrate de implementarlo correctamente.

1.3. Quicksort

En este laboratorio debes desarrollar un quicksort que minimice el consumo de la pila. Para ello, tal como se te indicó en clase, asegúrate siempre de mandar a ordenar recursivamente la partición de menor tamaño.

Atención: el evaluador impondrá un tamaña de pila pequeño ($\approx$ 10 Kb). Consecuentemente, si no consideras lo anterior, el evaluador muy probablemente abortará y no será evaluado.

2. Prelaboratorio

Para desarrollar exitosamente este laboratorio requieres tener instalada la biblioteca $\aleph_\omega$ (Aleph-ω); especialmente debes estar familiarizado con el tipo `DynArray`.

Para que el probador y otros utilitarios que se te proveerán compilen, deberás tener instalada las bibliotecas `TCLAP` y `nana`.

2.1. Generación de secuencias aleatorias

En este laboratorio se provee un generador de números aleatorios instrumentado en `rand.C`. Compílalo y ejecuta:

```
./rand -h
```

Para ver cómo funciona.

2.2. Probador

Para que puedas realizar tus pruebas se provee el archivo `qs.C` el contiene una batería de pruebas de todas las rutinas que deberás implementar. Este probador compila y funciona, pero puesto hasta que tus rutinas no estén correctamente implementadas sus resultados no serán correctos.

Puedes usar `qs` en combinación con el generador de secuencias aleatorias. Por ejemplo, la línea:

```
./rand -n 10000000 -i -1000000 -x 1000000 | ./qs-op -n 10000000
```

Genera 10^7 números al azar entre $[-10^6, 10^6]$ y luego prueba tus rutinas.

En `qs.C` hay dos rutinas que te podrían ser de ayuda:

1. `sorted()` la cual verifica si el arreglo está ordenado, y
2. `partition_ok()` la cual verifica si el arreglo está particionado.

3. Laboratorio

Las implementaciones de tu laboratorio las pondrás dentro del archivo `q.H`.

3.1. Pivote el elemento más a la izquierda

Escribe la función

```
    tuple<size_t, size_t>
select_left(const DynArray<long> &, const long l, const long)
```

La cual selecciona como pivote el elemento situado en `l`.

3.2. Partición

Escribe la función:

```
QSinfo qs_partition(DynArray<long> & a, const long l, const long r,
                    Sel_Pivot sel_pivot)
```

Esta rutina particiona el arreglo `a` entre los índices `l` y `r` según el método de selección de pivote `sel_pivot`.

Es crítico que comprendas la interfaz.

`QSinfo` se define así:

```
using QSinfo = tuple<size_t, size_t, size_t>;
```

Este es el tipo de retorno de la partición; o sea una "tripleta". El primer elemento es el índice del pivote. El segundo es la cantidad de comparaciones que se hicieron. El tercer elemento es la cantidad de swaps.

Por supuesto, es muy útil que instrumentes correctamente esta función.

Esta función se te evaluará (con bastante peso), pero su correctitud no incidirá en el resto de las rutinas.

3.3. Quicksort

Instrumenta:

```
tuple<size_t, size_t>
qs(DynArray<long> & a, const long l, const long r, Sel_Pivot sel_pivot)
```

La cual ordena por el quicksort usando la partición ya explicada y con selección de pivote `sel_pivot`.

La rutina retorna una tupla cuyo primer elemento es la cantidad de comparaciones y segundo la cantidad de swaps.

Ordena siempre de primero la partición más pequeña. Esto garantizará un consumo de pila mínimo. Es importante que hagas esto porque sino tu programa podría fallar para arreglos grandes que estén semi-ordenados. Una falla de este tipo probablemente causará un desborde de la pila del proceso- El evaluador no hará nada por detectar esta situación. Así pues, si el evaluador aborta entonces será imposible evaluarte.

3.3.1. Pivote el elemento más a la derecha

Programa:

```
 tuple<size_t, size_t>
select_right(const DynArray<long> &, const long, const long r)
```

La cual selecciona como pivote el elemento situado en `r`.

3.3.2. Pivote seleccionado al azar

Escribe la función

```
 tuple<size_t, size_t>
select_random_median(const DynArray<long> & a, const long l, const long r)
```

La cual selecciona como pivote un elemento al azar entre $[l..r]$.

El código que se te provee está configurado para usar un generador de números aleatorios global. Para seleccionar un número aleatorio usa la función:

```
gsl_rng_uniform_int(g, N)
```

La cual retorna un entero aleatorio comprendido en el intervalo $[0..N)$ (**ojo:** N **no está incluido**).

Para esta función sólo es necesaria una llamada al generador de números aleatorios.

Recuerda ajustar el aleatorio para que esté dentro de $[l..r]$.

3.3.3. Pivote como la mediana de los extremos

Escribe la función

```
 tuple<size_t, size_t>
select_median(const DynArray<long> & a, const long l, const long r)
```

Esta función retorna el índice de la mediana entre los elementos `a[l]`, `a[(l+r)/2]` y `a[r]`.

Hay varios métodos para seleccionar la mediana. Por esa razón, para asegurar reproducibilidad, use el siguiente algoritmo para seleccionar su mediana:

```
  const long & x = a(l);
  const long & y = a(m);
  const long & z = a(r);

  long median;
  if (x < y)
    {
      if (y < z)
        median = y;
      else
        {
          if (x < z)
            median = z;
          else
            median = x
        }
    }
  else if (x < z)
```

```
    median = x
  else
    {
      if (y < z)
        median = z;
      else
        median = y;
    }
```

Al final, la variable `median` contiene la mediana. Así, lo que debes hacer es emplear este algoritmo para contar la cantidad de comparaciones y calcular el índice de la mediana de los elementos (cuál de los índices entre `l, m, r` contiene la mediana).

Por simplicidad, aunque inexacto, asume que cada asignación es un swap.

Si el tamaño de la partición es menor o igual a dos (2), entonces este método no tiene sentido. En este caso debes retornar `r` como índice del pivote.

3.3.4. Pivote como la mediana de `Num_Samples` elementos al azar

Escribe la función:

```
   tuple<size_t, size_t>
select_random_median(const DynArray<long> & a, const long l, const long r)
```

Esta función selecciona como pivote la mediana entre `Num_Samples` elementos escogidos al azar. Este método sólo se ejecuta si la partición contiene más de `Random_Theshold` elementos. Si la partición contiene `Random_Theshold` o menos elementos, entonces la selección será mediante una llamada a `select_median(a, l, r)`.

Para escoger la mediana ejecutaremos exactamente `Num_Samples` sorteos para seleccionar los índices al azar. Por cada índice k escogido al azar, cargaremos en un arreglo `samples[]` el par `(a[k], k)`. Luego ordenaremos `samples[]` mediante el método de inserción dado. Una vez ordenado, la mediana es el elemento del centro.

4. Evaluación

La fecha de entrega de este laboratorio es el **lunes 20 de abril** hasta las 6 pm. Puedes entregar antes, recibir corrección e intentar cuantas veces prefieras (limitado por supuesto a nuestra capacidad computacional). Tu calificación será la de la última sumisión.

Para evaluarte debes enviar el archivo `q.H` a la dirección:

`leandro.r.leon@gmail.com`

Antes de la fecha y hora de expiración. El "subject" debe **obligatoria y exclusivamente** contener **exactamente** el texto **PR3-LAB-02**. 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 `q.H` y sólo envía ese archivo**.

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**.

Atención: si tu programa no compila, entonces el evaluador no compila. Si alguna de tus rutinas se cae, entonces el evaluador se cae. Si una rutina cae en un lazo infinito o demora demasiado, entonces el evaluador cae en un lazo infinito o se demora demasiado. Exactamente sucedería lo mismo si tu programa fuese parte de un sistema de aviónica: ¡se estrella el avión!. Por esa razón, en todos estos casos será imposible darte una nota, lo que simplemente se traduce en que tienes cero. Si tu rutina dispara una excepción inesperada, el evaluador no hará mucho por capturarla y reportártela.

Por favor, no envíes programas que no compilan o se caen. Hacen perder los tiempos de red, de cpu, el tuyo y el mío.

La puntuación se distribuirá en dos partes:

Cualidad	Porcentaje
Eficacia	80 %
Estilo y elegancia de código	20 %

El programa que entregues será compilado y ejecutado contra varios archivos de prueba. Para que una solución esté válida, debe corresponderse exactamente con nuestros casos de prueba. Si tu solución es incorrecta, entonces tu nota será 0.

5. Recomendaciones

1. Este laboratorio es "fácil". No es el adecuado para dejarlo como el ausente. Aprovéchalo para obtener una buena nota.

2. Diseña tú mismo un caso de prueba pequeño; unos 20 elementos aproximadamente. Ejecuta el quicksort y cuenta tu mismo la cantidad de comparaciones que se efectúa.

3. Usa el generador de números aleatorios la cantidad exacta de veces que se requiere; ni una más y ni una menos. Si lo haces pierdes la reproducibilidad.