Laboratorio - 3

Alumno: Jackson Fernando Merma Portocarrero

Escuela Profesional: Ingeniería de Sistemas

CUI: 20202143

Correo: jmermap@unsa.edu.pe

Ejercicio 1: Búsqueda binaria

El desarrollo de este algoritmo es simple, ya que se inicializa con un arreglo ordenado de 8 datos y se hace simplemente un ciclo *while*, para encontrar una coincidencia de *num* en la estructura.

Algoritmo: Se inicializa un extremo inferior (*min*) y superior(*max*) que determinara la posición absoluta a buscar con $pos = min + (max - min)/2$;

```
//algoritmo
while(min<=max){
  pos=min+(max-min)/2;
  if(arr[pos]==num){
    cout<<"Encontrado"<<endl;
    break;
  }
  if(arr[pos]>num){
    max=pos-1;
  }else{
    min=pos+1;
  }
}
```

Gráfico 1: Codigo en Java (algoritmo de búsqueda binaria)

Ejercicio 2: Búsqueda binaria grafico para '*n*' casos

Clase 'G': Gráfico

La clase G, es básicamente un fragmento de código que ayudará a graficar la complejidad del algoritmo de búsqueda binaria con datos de tiempo en **nanosegundos**; para ello se hace uso de algunas librerías como *awt,sing* y *geom*. Entonces, se envía al constructor de este objeto, un arreglo de números que almacena los tiempos de ejecución en cada caso, para luego pasar a ser graficados mediante un bucle *for* con cada punto.

```
//graficando puntos
for(int i=0;i<coordinates.length;i++){
  double x1=mar+i*x;
  double y1=height-mar-scale*coordinates[i];
  g1.fill(new Ellipse2D.Double(x1-2,y1-2,4,4));
}
```

Gráfico 2: Código Java (bucle *for* para graficar puntos)

Algoritmo en clase principal

Para el desarrollo de un gráfico que proporcione la complejidad del algoritmo, se da como entrada una cantidad de **'n'** casos; para ello también se debe pensar en los datos con los que se trabajara en la búsqueda, entonces se creó un generador de datos que comienza en la generación de 800 datos y con un rango de 200 datos en el siguiente caso, es decir 800,1000,1200,1400, etc desde el caso 1 al n.

```
final int MIN=800;
final int RANGO=200;

//casos
for(int i=MIN,u=0;u<cant;i+=RANGO,u++){
  long nums[];
  long start,end,duration;
  int len=i;
  nums=new long[len];
```

Gráfico 3: Código Java (bucle *for* para generación de cantidad de datos por caso)

Luego, se consideró su creación de datos en cada caso de forma creciente para que el problema sea lo más posiblemente realista, de tal forma que se genere un número aleatorio de 10 en 10 para cada dato; eso implica que se crean k datos y que sean aleatorios entre 1 a 10*k.

```
//datos aleatorios ordenados de 10 en 10. Cantidad=len
//Ejemplo: len=100, entonces numeros ordenados y aleatorios de 1 a 1000
for(int k=0,minRango=1;k<len;k++,minRango+=10)
  nums[k]=(long)(Math.random()*(10))+minRango;
```

Gráfico 4: Código Java (bucle *for* para generación de cada dato en un caso)

Finalmente se implementó el mismo algoritmo del problema anterior y la captura de tiempo con la instrucción *System.nanoTime()*, que capturara datos en nanosegundos. Para luego crear un objeto de tipo G enviado los datos capturados en cada caso.

Ejemplo

En el siguiente ejercicio se generó 1000 casos, dando como resultado la siguiente gráfica:

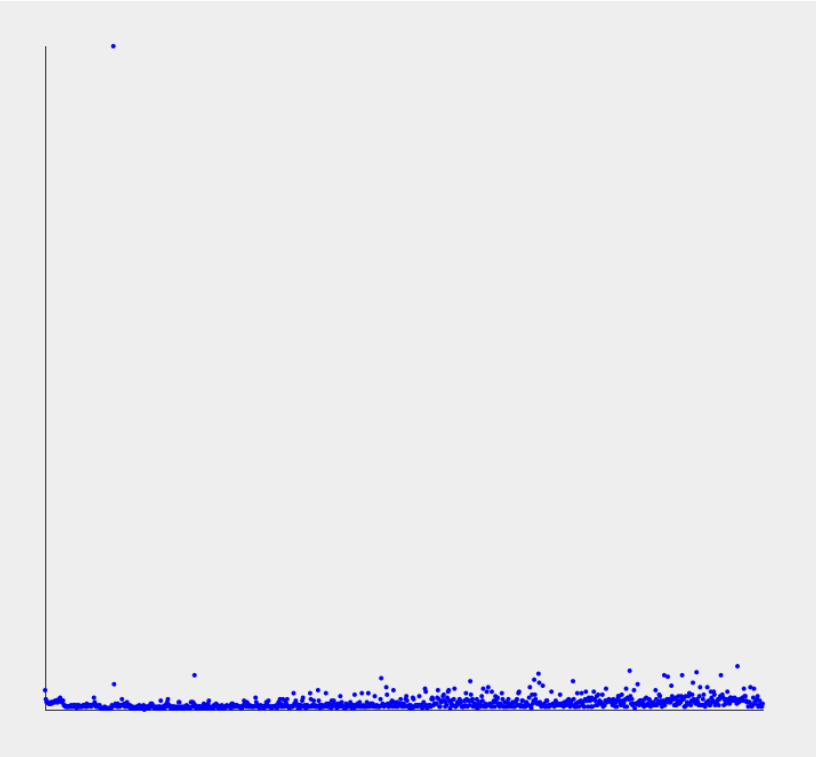

Gráfico 5: Gráfico de dispersión de ‘n’ casos y tiempo de ejecución en nanosegundos

Observación: La gráfica no tiene títulos, ni títulos en cada eje, pero esencialmente expresa una de: *casos por tiempo* en nanosegundos. Finalmente, como se puede observar, la gráfica trata de mostrar un tiempo logarítmico para el algoritmo de búsqueda binaria.

Ejercicio 3: Algoritmo de emparejamiento

Entrada y salida de datos

La entrada de datos es la cantidad de participantes en 1 grupo, luego los nombres del grupo A (n nombres), luego una matriz de n x n donde se indicará el orden de preferencia del sujeto i-ésimo del grupo A (filas) que tienen del grupo B (columnas), en números (1-n). En la siguiente línea la entrada será n nombres del grupo B y finalmente una matriz de n x n, donde se indicará el orden de preferencia del sujeto j-ésimo del grupo B (filas) que tienen del grupo A (columnas), sin embargo estas últimas estarán de forma ordenada.

Ejemplo:

```
2
joel luis
1 2
2 1
amanda rosa
2 1
1 2
```

En este primer caso, hay dos participantes por cada grupo, el grupo A conformado por joel y luis, en la matriz luego de sus nombres, se indica que joel (el primero), tiene más preferencia por el sujeto 1 del grupo B (amanda) y luego por el sujeto 2 (rosa), de esta misma forma se puede interpretar la segunda fila de la matriz A (sujetos en orden de preferencia para luis).
Sin embargo en la segunda tabla, se trabaja de manera diferente, ya que la fila 1, se interpreta como los sujetos del grupo A preferidos por amanda, pero en el orden corriente, es decir, el dato de la fila 1 y columna 1 (2), se interpreta como si el sujeto 1 (columna) tiene el orden 2 de preferencia para el sujeto de la fila 1, eso quiere decir que joel tiene orden de preferencia 2 para amanda, mientras que luis tiene orden de preferencia 1 para amanda (dato de fila 1 y columna 2); esto se decide así para generar un coste de complejidad O(1), al tener conflictos entre sujetos que prefieren mas a otros y rompimientos de relaciones.

Finalmente, la salida de datos es el emparejamiento con nombres de los sujetos del grupo A con el grupo B.

```
Emparejamientos
joel->amanda
luis->rosa
```

Algoritmo

Para el desarrollo del algoritmo, se usan 3 arreglos adicionales de tamaño n, que básicamente 2 de ellos seguirán el emparejamiento que tiene el grupo A con el grupo B, y el segundo, en viseversa; el tercer arreglo hace seguimiento al orden de preferencia(posición) del grupo A, lo cual simplifica el coste de ingreso de datos al aplicar una cola por prioridad.

Finalmente, se aplica el algoritmo emparejando y llevando conteos.

```java
for(int i=0;count<len;i++,i%=len){
  if(empA[i]==0){
    int opt=a[i][pri[i]]-1;
    //verificacion en B
    if(empB[opt]==0){
      empA[i]=opt+1;
      empB[opt]=i+1;
      count++;
    }else{
      //verificacion de la mejor opcion
      if(b[opt][i]<b[opt][empB[opt]-1]){
        empA[empB[opt]-1]=0;
        empB[opt]=i+1;
        empA[i]=opt+1;
      }else{
        pri[i]++;
        i--;
      }
    }
  }
}
```

Gráfico 6: Código Java (algoritmo de emparejamiento)

Ejemplo

En el siguiente ejemplo, se dan los siguientes datos

Victor	Bertha	Amy	Diane	Erika	Claire
William	Diane	Bertha	Amy	Claire	Erika
Xavier	Bertha	Erika	Claire	Diane	Amy
Yancey	Amy	Diane	Claire	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Claire

Amy	Zeus	Victor	William	Yancey	Xavier
Bertha	Xavier	William	Yancey	Victor	Zeus
Claire	William	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	William
Erika	Yancey	William	Zeus	Xavier	Victor

Luego, para la entrada de datos se tiene:

```
5
victor william xavier yancey zeus
2 1 4 5 3
4 2 1 3 5
2 5 3 4 1
1 4 3 2 5
2 4 1 5 3
amy bertha claire diane erika
2 3 5 4 1
4 2 1 3 5
5 1 2 3 4
1 5 4 3 2
5 2 4 1 3
```

Dando como resultado:

```
Emparejamientos
victor->amy
william->claire
xavier->bertha
yancey->erika
zeus->diane
```