

Tarea 2:

Implementación y Análisis

de algoritmo Shell Sort

Profesor: Benjamín Bustos.

Profesor Auxiliar: Boris Romero

Sebastián Quezada

Alumno: Jorge Gutiérrez

Fecha: 3 de Septiembre de 2014

Introducción

Uno de los mayores avances de la humanidad ha sido la creación de maquinas capaces de hacer tareas productivas de forma rápida y eficiente. Así la humanidad ha sido capaz de aumentar sus esperanzas de vida y la calidad de esta utilizando estas maquinas, tanto para la fabricación de alimentos, ropa , autos, documentos, maquinas, en fin, la automatización de la producción de bienes. Por otro lado estas maquinas además de estar presente en la producción de bienes están participando activamente en el proceso de organización de la vida en la sociedad. Es fácil notar que la gente usa aparatos electrónicos comúnmente y casi como si fuera algo normal .

Ahora la procedencia de todos estos artefactos tiene un ancestro en común y en si son una forma mas avanzada de antepasado capaz de procesar información millones de veces mas rápido que la anterior, así es como podemos relacionar el ábaco con la computadora y dado esta relación, y pensando que ambos tienen un origen común, entonces esto significa que ambas pueden resolver el mismo problema, tal que el computador puede resolverlo mas rápido que el ábaco. Si nos preguntamos cual es este problema nos daremos cuenta rápidamente que es. “Contar y ordenar”.

Así es como el humano para poder ser capaz de resolver este problema mediante computadoras a creado distintas formas de abordarlo, llamadas algoritmos.

En este informe se probara una forma de ordenar elementos, en especifico números usando un método llamado Shell Sort el cual se pondrá a prueba y se discutirán sus resultados.

Diseño de la Solución

Implementación de Shell Sort:

La filosofía de esta solución esta dada por la h-ordenación, es decir, imagine que tengo una lista de n elementos:

5

4

3

2

1mtlmt 1n

n

Y elijo la 3-ordenacion, esto quiere decir que formo otro lista solo con los elementos de de la lista inicial, pero con solo los elementos que están a 3 espacios (h espacios) de distancia de mi posición inicial.

Es decir tendría la lista

3n+1

13

10

7

4

1mtlmt 1n

La cual debo ordenar y devolver los elementos ordenados a la lista original respetando que mi primer elemento de esta lista ordenada debe ir en la posición inicial donde me pare y luego el segundo a una distancia de 3 (h) desde donde puse el anterior y así hasta volver a completar la lista inicial.

Luego cambio mi punto de inicio por el siguiente, es decir

3n+2

14

11

8

5

2mtlmt 1n

Y vuelve a poner en la lista inicial de forma ordenada, al igual que para el caso anterior.

Así esto se repite para tantas posiciones iniciales como el valor de h sea , en este caso se haría para la posición inicial 1,2 y 3.

Cabe mencionar que ordenar cada una de estas listas se hará por ordenamiento por inserción el cual consiste en tomar un elemento y compararlos con todos los anteriores hasta que encuentre un lugar en donde el anterior es menor a mi elemento seleccionado y el de adelante mayor a este. Se deja como ejercicio al lector una mayor profundización en este tipo de ordenamiento.

Implementación

Ya conociendo el diseño de la solución planteada en el apartado anterior, entonces la implementación el lenguaje Java puede ser descrita de la siguiente manera:

**public** **void** Shellbyint (**int** h){

/\*Este metodo esta implementado de tal manera que solo modifica la lista que esta guardada en un parametro interno de la clase, por lo que no estrega nada VOID\*/

ArrayList aux= **new** ArrayList();

**int** i;

/\*Se require una lista auxiliary en donde guarder el extracto y un indice que vaya saltando de h en h o i\*h \*/

**for** (**int** pos=0;pos<h;pos++){ //Marca POs inicial

**for** (i=0;i<=**this**.Lista.size();i++){ //Marca el salto

**try**{

aux.add(**this**.Lista.get(pos+i\*h));

}**catch** (Exception IndexOutOfBounds){

aux=ordenar(aux);

**break**;}}

/\*Se completa la list hasta que se haya reccorrido entera o se salga de ella, luego de eso se ordena la lista\*/

**for** (**int** j=0;j<=**this**.Lista.size();j++){

**try**{

**this**.Lista.set(pos+j\*h, aux.get(j));

}**catch** (Exception IndexOutOfBounds){

aux.clear();

**break**; //Idem

}

}

}

}

Experimentación

La experimentación consiste en 2 principales experimentos para listas de tamaño con .

Experimento N°1: Ejecutar Shell Sort con la secuencia decreciente de números de la forma comenzando con el más cercano a n.

Experimento N°2: Ejecutar Shell Sort con la secuencia que se obtiene con < 1; 4; 10; 23; 57; 132; 301; 701 > y

los siguientes valores de la secuencia se calculan con la fórmula hasta llegar al más cercano a n.

Se analizara el tiempo requerido por cada uno para llevar a cabo el ordenamiento y la cantidad de comparaciones lógica que hace.

Resultados y Conclusiones

Los resultados obtenidos para el Experimeto N°1 se pueden expresar en el siguiente gráfico.

Los resultados del experimento N°2 se pueden ver en el siguiente Grafico



Anexos

1. Clase ShellSort

**package** Paquete;

**import** java.util.ArrayList;

**import** java.util.Random;

**public** **class** ShellSort {

ArrayList Lista = **new** ArrayList();

Random nr = **new** Random();

ArrayList auxlist=**new** ArrayList();

**long** contador=0;

**int** contadorif=0;

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

}

**public** ShellSort(){}

**public** ShellSort(**int** i){

**double** n=Math.*pow*(2,i);

**float** aux;

**while** (n>0){

aux=nr.nextFloat();

**if** (-1==**this**.Lista.indexOf(aux)){

**this**.Lista.add(aux);

n--;

}

}

}

**public** **void** Shellbyint (**int** h){

ArrayList aux= **new** ArrayList();

**int** i;

**for** (**int** pos=0;pos<h;pos++){

**for** (i=0;i<=**this**.Lista.size();i++){

**try**{

aux.add(**this**.Lista.get(pos+i\*h));

}**catch** (Exception IndexOutOfBounds){

aux=ordenar(aux);

**break**;}}

**for** (**int** j=0;j<=**this**.Lista.size();j++){

**try**{

**this**.Lista.set(pos+j\*h, aux.get(j));

}**catch** (Exception IndexOutOfBounds){

aux.clear();

**break**;

}

}

}

}

**public** **void** Shellbylist (ArrayList n){

**int** count=n.size()-1;

**while** (count >=0){

Shellbyint((Integer) n.get(count));

}

}

**public** ArrayList ordenar(ArrayList lista){

**float** aux;

**for** (**int** pos1=1;pos1<lista.size();pos1++){

aux=(**float**)lista.get(pos1);

**for** (**int** pos2=pos1-1;pos2>=0 && (**float**)lista.get(pos2)>(**float**)aux;pos2--){

**this**.contadorif++;

lista.set(pos2+1, lista.get(pos2));

lista.set(pos2, aux);

}

}

**return** lista;

}

**public** ArrayList secb(){

**int** num=**this**.Lista.size();

ArrayList list = **new** ArrayList();

Double a;

**int** b;

a=(Math.*log*(num+1)/Math.*log*(2));

b=a.intValue();

**for**(**int** i=b;i>=0;i--){

list.add(Math.*pow*(2, i)-1);

}

**return** list;}

**public** ArrayList secc(){

ArrayList list= **new** ArrayList();

list.add(1);

list.add(4);

list.add(10);

list.add(23);

list.add(57);

list.add(132);

list.add(301);

list.add(701);

**long** aux=(Long) list.get(list.size()-1)\*9/4;

**while** (aux<**this**.Lista.size()){

list.add(aux);

aux=(Long) list.get(list.size()-1)\*9/4;}

**return** list;

}

}

1. Clase Main