Algoritmo de Busqueda Binaria Recursiva

Grupo Alias

Universidad Tecnológica Metropolitana

`https://github.com/Lanceconan/AnalisisDeAlgoritmos`

4 de mayo de 2014

Manuel Irarrázabal
Rafael Vivar
Daniel Gutiérrez
Juan Cid

Introducción

Definicion

La búsqueda binaria en un vector ordenado de datos se realiza comprobando el elemento que está en el centro del vector y mirando si el elemento buscado es mayor o menor.

Proposito búsqueda binaria

Se utiliza cuando el vector en el que queremos determinar la existencia de un elemento está previamente ordenado.
Este algoritmo reduce el tiempo de búsqueda considerablemente, ya que disminuye exponencialmente el número de iteraciones necesarias.

Funcion Binaria Recursiva en C/C++

```
#include <iostream>
#include <vector>

bool busqueda_binaria(const vector<int> &v, int principio, int fin, int &x){
    bool res;
    if(principio <= fin){
        int m = (principio + fin)/2;
        if(x < v[m]) res = busqueda_binaria(v, principio, m-1, x);
        else if(x > v[m]) res = busqueda_binaria(v, m+1, fin, x);
        else res = true;
    }else res = false;
    return res;
}
/*{Post: Si se encuentra devuelve true, sino false}*/
```

Figura: Codigo

Calculo de complejidad

Mejor Caso

En base al árbol podemos postular que su altura esta expresada en la siguiente formula:

$$n <= 2^{(h+1)} - 1 \quad (1)$$

Obteniendo que:

$$h >= \log_2(n) \quad (2)$$

Esta es la característica principal de un árbol, la que permite que las operaciones sean tan rápidas, el único inconveniente es asegurar este mínimo valor para la altura ...

Peor Caso

Si sumamos las operaciones de inserción y recorrido tendriamos lo siguiente para el mejor caso:

$$O(n) = n \log_2 (n) + n \tag{3}$$

O esto para el peor caso:

$$O(n) = n^2 + n \tag{4}$$

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Sobre el Código

- IDE utilizado: Aptana Studio 3
- Programación en Ruby, version 1.9.3
- Una única clase con sus inherentes métodos
  - Métodos que permiten facil manipulación y entendimiento del código
  - Variables Nemotécnicas
  - Búsqueda binaria recursiva
- Estructura utilizada: Un arreglo de largo N-1
- Metodo de Ordenamiento Burbuja (alternativo)

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Caracteristicas

- Permite distintos poblamientos de los datos
  - Ordenado automatico (de entrada)
  - Aleatorio
  - Manual
- Facil Manipulacion del Codigo
- Utiliza el ordenamiento Burbuja Complejidad

$$O(n) = n^2 \tag{5}$$

Pruebas

Probemos el programa

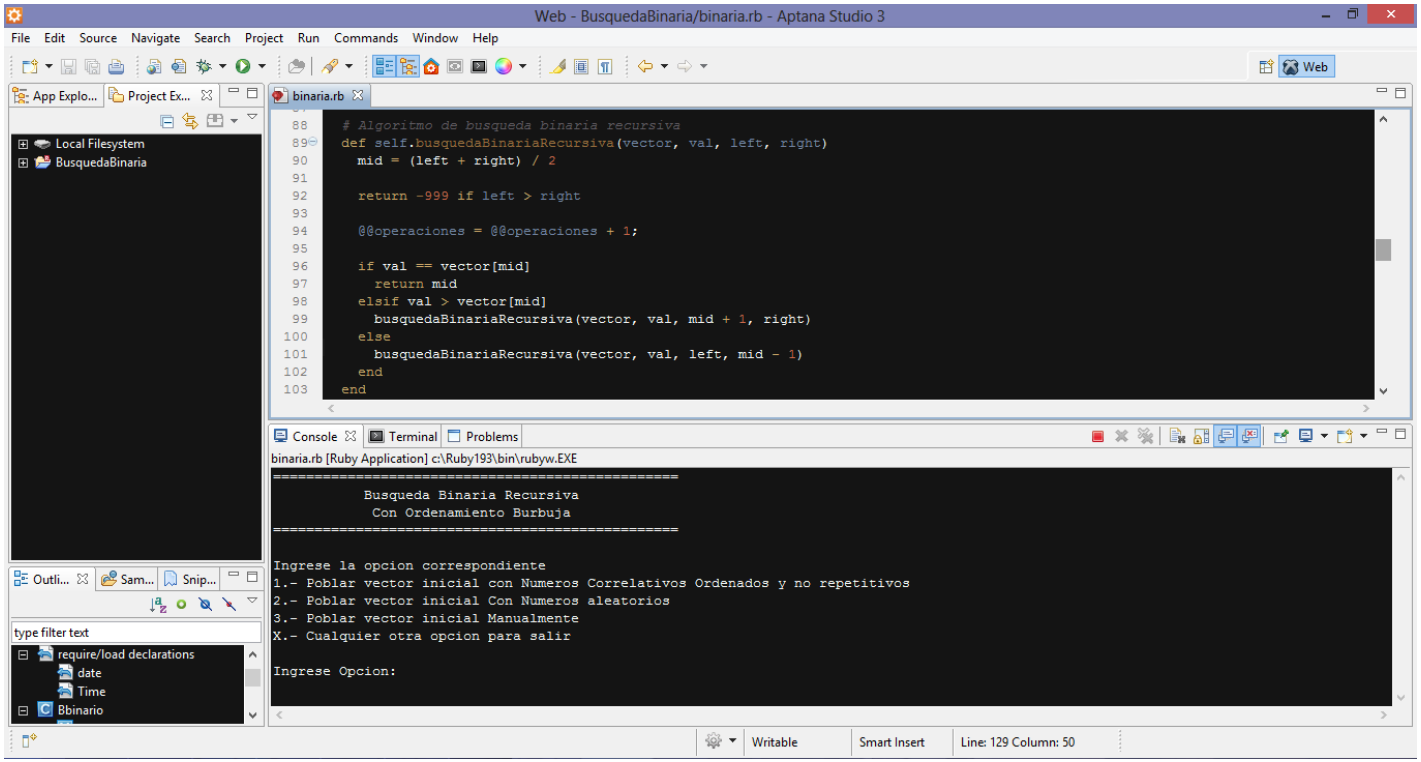

Figura: Pantallazo Aptana

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Pruebas

- Peor Caso: Que el valor no esté en el vector unidimensional
  - 1.000 : 10 llamados
  - 10.000 : 14 llamados
  - 100.000 : 17 llamados
  - 1.000.000 : 20 llamados
  - 10.000.000 : 24 llamados
  - 100.000.000 : 27 llamados
- Mejor Caso: Que el valor buscado está en el vector unidimensional justo en el centro estimado
  - 1.000 : 1 llamado
  - 10.000 : 1 llamado
  - 100.000 : 1 llamado
  - 1.000.000 : 1 llamado
  - 10.000.000 : 1 llamado
  - 100.000.000 : 1 llamado

Conclusiones

- Se puede mejorar usando arboles autobalanceables, aunque estar asegurando minuciosamente puede añadir un costo considerable.
- No se nota la tendencia logaritmica de las inserciones debido a la complejidad lineal del recorrido.

Conclusiones

- Se puede mejorar usando arboles autobalanceables, aunque estar asegurando minuciosamente puede añadir un costo considerable.
- No se nota la tendencia logaritmica de las inserciones debido a la complejidad lineal del recorrido.