Practica 2

Los utlimos

4 de octubre de 2021

Analsis de casos por codigo

Busqueda con Arbol Binario de busqueda

Operaciones basicas consideradas: comparacion entre la llave a buscar y el elemento del nodo, la funcion isempty que es una comparacion entre el ABB y el valor nulo.

```
void search(struct node* root, int key)
{
    struct node* aux = root;
    while(!isempty(aux) && key != aux->data)
        aux = (key < aux->data) ? aux->left : aux->right;
    if(isempty(aux))
        printf("El elemento %d no se encontro en el arbol", key);
    else
        printf("El elemento %d se encontro en el arbol", key);
}
```

Condiciones del mejor caso: que el elemento a buscar este en la raiz.
Condiciones del peor caso: un arbol de tamaño de problema n que tenga n niveles y el elemento este en el nodo hoja.
Funcion de complejidad temporal del mejor caso $= 1$
Funcion de complejidad temporal del peor caso $= 2n$
Funcion de complejidad temporal del caso medio $= \frac{1+2n}{2}$

Busqueda Binaria

Operaciones basicas consideradas: comparaciones de la llave con elementos del arreglo, actualizacion de limite inferior y limite superior y la asignacion de h.

```
int binaria(int array[], int size, int key)
{
    int l = 0, r = size - 1;
    int h;
    while(l <= r)
    {
        h = l + (r - l)/2;
        if(key < array[h])
            r = h - 1;
        else if(key > array[h])
            l = h + 1;
        else
            return h;
    }
    return -1;
}
```

Condiciones del mejor caso: cuando el numero a buscar esta en medio del arreglo.
Condiciones del peor caso: cuando el numero a buscar esta al final del arreglo.
Funcion de complejidad temporal del mejor caso = 3
Funcion de complejidad temporal del peor caso = $4log_2(n)$
Funcion de complejidad temporal del caso medio = $\frac{4log_2(n)+3+3log_2(n)}{3}$

Busqueda exponencial

Operaciones consideradas: comparacion entre el elemento buscado y los elementos del arreglo, comparacion entre i y el size, operacion de asignacion de i, la busqueda binaria que se acotara como log_2 n.

```
int exponencial(int array[], int size, int key)
{
    if(array[0] == key)
        return 0;
    int i = 1;
    while(i < size && array[i] < key)
        i *= 2;
    return binaria(array, i/2, min(i, size-1), key);
}
```

Condición de mejor caso: El número buscado está en la primera posición.
Condiciones Peor caso: El numero esta en la ultima posicion del arreglo siempre y cuando el tamaño del arreglo no sea: potencia de 2 + 1, en ese caso seria la posicion anterior.
Función de complejidad temporal del mejor caso : 1.
Función complejidad temporal del peor caso: $4log_2(n)$.
Función complejidad caso medio: $\frac{1}{2} + 2log_2(n)$.

Busqueda de fibonacci

Operaciones consideradas: Comparaciones entre el elemento buscado y los elementos del arreglo, asignaciones a las variables fibonacci dentro del while de analisis.

```
        auxf2 = auxf1;
        auxf1 = auxfm;
        auxfm= auxf2 + auxf1;
    }
    rango = -1;// rango toma el valor de -1
    //el ciclo entra hasta que ya no queden numeros a buscar ,compararemos elind
    while (auxfm > 1) {
        // verifica  si auxfm2 es no es vacio
        i = min(rango + auxf2, n - 1);
        //Si numbuscar es mayor que el valor de  auxfm2  el arreglo se dezplaza
        if (arr [i] <x) {
            auxfm= auxf1;
            auxf1 = auxf2;
            auxf2 = auxfm- auxf1;
            rango = i;
        }
        //en cambio si  es mayor que el valor en arr[auxfm2],  dezplazamos el ar
        else if (arr [i]> x) {
            auxfm= auxf2;
            auxf1 = auxf1 - auxf2;
            auxf2 = auxfm- auxf1;
        }
        // si el elemnto es encontrado se retorna la posicion i
        else
            return i;
    }
 //se comprara el ultimo elemento con numbusc
    if (auxf1 && arr [rango + 1] == x){
        return rango + 1;
    }
    //si no se encuentra se retorna -1
    return -1;
}
```

Condición de mejor caso: El número buscado está en la primera posición.
Condiciones Peor caso: Cuando el número que buscamos se encuentra en el sub-arreglo que tiene mayor tamaño por lo que hará un mayor número de instrucciones.
Función de complejidad temporal del mejor caso : 1.
Función complejidad temporal del peor caso: $6log(n)$.
Función complejidad caso medio: $6log(n)$.

Busqueda lineal

Operaciones consideradas: Comparaciones entre el elemento buscado y los elementos del arreglo.

```
int busquedaLineal(int array[], int size, int key)
{
    int i;
    for (i = 0; i < size; i++)
        if (array[i] == key)
            return i+1;
    return -1;
}
```

Condición de mejor caso: El número buscado está en la primera posición.
Condiciones Peor caso: El número buscado está al final del areglo.
Función de complejidad temporal del mejor caso : 1.
Función complejidad temporal del peor caso: n.
Función complejidad caso medio: $\frac{n(n+3)}{2(n+1)}$.