INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

Hernández Hernández Alejandro

Boleta:2016630177

Análisis de Algoritmos

M. En C. Luz María Sanchez García

Practica 2

Fecha de entrega: 11-03-2018

INTRODUCCIÓN

En esta práctica, se analizó el costo temporal y espacial de algunas estructuras de datos, eligiendo 3 tipos de ellos, para observar el costo de busqueda de algun dato, existente o no, en dicha estructura de n elementos.

Las estrucutras a elegir son 5: Pila, Cola, Lista, Lista doblemente Enlazada, y Árbol.

El tamaño de la estructura será definida por el usuario, y los datos elegios, fueron enteros, los cuales fueron elegidos al azar y puestos en cada espacio para cada estructura.

PLANTEAMIENTO DEL PROBLEMA

Conocer el costo temporal y espacial de la busqueda de 1 o mas elementos en 3 TAD(Tipo Abstracto de Datos) a elegir, y comparar los resultados, y programando dichos TAD en el lenguaje C.

Descripción de la práctica

- Hacer un programa que realice la ***búsqueda lineal*** de un elemento dentro de 3 estructuras de datos distintas (pila, lista, árbol, etc.) considerando que los n datos no se encuentran ordenados (peor caso o caso promedio).

- Diseñar los algoritmos para calcular de cada una de las estructuras de datos:
  - La función de complejidad temporal: *f(n)*
  - La función de complejidad espacial: *f(n)*

Solución

Para calcular el costo de busqueda del algoritmo, en la funcion de busqueda de cada tipo de dato, agregar una variable que se autoincremente cada vez que se ejecute una instrucción del propio algoritmo, y al final de su ejecucion, mosrar en pantalla el costo total que realizo el algoritmo de busqueda.

Lo primero, es elegir los TADs a utilizar, en mi caso, serán: Cola, Pila y Lista.

Luego, declarar cada estructura, por buenas practicas, en un archivo *.h*, para importarlo en el archivo *.c*.

En los algoritmos de busqueda, agregar la variable mencionada al inicio de este apartado para conocer el costo de ejecucion del algoritmo de busqueda.

Códigos

Pila

pila.h

```h
#ifndef _pila_
#define _pila_

typedef struct Pila{
    int dato;
    int pos;
    struct Pila *sig;
}Pila;

typedef Pila *pila;

pila creapila();
pila push(pila p,int e,int pos);
pila pop(pila p);
void buscar(pila p,int dato);

#endif
```

pila.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include "pila.h"

int main(){
  srand(time(NULL));
  pila p=creapila();
  int n=-1,busc=0;
   while(n<0){
     printf("ingrese tamaño pila: ");
     scanf("%d",&n);
     if(n<0){
       printf("ERROR: es negativo.\n");
     }
  }
```

```
17.    for(int i=0;i<n;i++){
18.      p=push(p,rand()%50,i+1);
19.    }
20.    while(busc>=0){
21.
22.      printf("ingrese dato a buscar(entre 0 y 50): ");
23.      scanf("%d",&busc);
24.      if(busc<=-1) {
25.        while(p!=NULL){
26.          p=pop(p);
27.        }
28.        exit(0);
29.      }
30.      buscar(p,busc);
31.
32.    }
33. }
34.
35. pila creapila(){
36.    pila p=NULL;
37.    return p;
38. }
39.
40. pila push(pila p, int e,int pos){
41.    pila elem=(pila)malloc(sizeof(Pila));
42.    if(elem==NULL){
43.      printf("ERROR: no hay memoria\n");
44.      exit(0);
45.    }
46.    elem->dato=e;
47.    elem->pos=pos;
48.    elem->sig=NULL;
49.
50.    if(p==NULL){
51.      p=elem;
52.    }
53.    else{
54.      pila aux=p;
55.      while(aux->sig!=NULL){
56.        aux=aux->sig;
```

```
57.      }
58.      aux->sig=elem;
59.    }
60.
61.    return p;
62.
63. }
64.
65. pila pop(pila p){
66.  pila aux2=p;
67.    if(p->sig==NULL){
68.      free(aux2);
69.      p=NULL;
70.    }else{
71.      pila aux=p->sig;
72.      while(aux->sig!=NULL){
73.        aux=aux->sig;
74.        aux2=aux2->sig;
75.      }
76.      aux2->sig=NULL;
77.      free(aux);
78.    }
79.    return p;
80. }
81.
82. void buscar(pila p, int dato){
83.    pila aux=p;
84.    int costo=0;
85.    while(aux!=NULL){costo+=1;
86.      if(aux->dato==dato){costo+=1;
87.        printf("el dato se encuentra en la posicion %d\n",aux->pos);
88.        break;
89.      }else{
90.        costo+=1;
91.      }
92.      aux=aux->sig;costo+=1;
93.    }
94.    if(aux==NULL){
95.     printf("No se encuentra el dato\n");
96.   }
```

```
97.    printf("Costo total del algoritmo de busqueda: %d\n",costo);
98. }
```

Cola

cola.h

```
1. #ifndef _cola_
2. #define _cola_
3.
4. typedef struct Cola{
5.  int dato;
6.  int pos;
7.  struct Cola *sig;
8. }Cola;
9.
10. typedef Cola *cola;
11.
12. cola crearcola();
13. cola agregar(cola c,int e,int pos);
14. cola quitar(cola c);
15. void buscar(cola c, int dato);
16.
17. #endif
```

cola.c

```
1. #include <stdio.h>
2. #include<stdlib.h>
3. #include<time.h>
4. #include "cola.h"
5.
6. int main(){
7.  srand(time(NULL));
8.  cola c;
9.  c=crearcola();
10.  int tam;
11.  printf("Ingrese tamaño de la cola: ");
12.  scanf("%d",&tam);
13.  for(int i=0;i<tam;i++){
14.   c=agregar(c,rand()%50,i+1);
15.  }
```

```
 cola aux=c;

 while(tam>-1){
  printf("Ingrese dato a buscar(entre 0 y 50): ");
  scanf("%d",&tam);
  if(tam<=-1) {
    while(c!=NULL){
     c=quitar(c);
    }
    exit(0);
   }
  else{
   buscar(c,tam);
  }
 }
}

cola crearcola(){
 cola c=NULL;
 return c;
}

cola agregar(cola c, int e,int pos){
 cola elem=(cola)malloc(sizeof(Cola));
 if(elem==NULL){
  printf("Error: no hay memoria\n");
  exit(0);
 }
 elem->dato=e;
 elem->pos=pos;
 elem->sig=NULL;
 if(c==NULL){
  c=elem;
 }else{
  cola aux=c;
  while(aux->sig!=NULL){
   aux=aux->sig;
  }
  aux->sig=elem;;
 }
```

```
 return c;
}

cola quitar(cola c){
 cola aux=c;
 if(c->sig==NULL){
  free(aux);
  c=NULL;
 }else{
  c=c->sig;
  free(aux);
 }
 return c;
}

void buscar(cola c, int dato){
 cola aux=c;
 int costo=0;
 while(aux!=NULL){costo+=1;
  if(aux->dato==dato){costo+=1;
   printf("El dato se encuentra en la posicion %d\n",aux->pos );
   break;
  }else{
   costo+=1;
  }
  aux=aux->sig;costo+=1;
 }
 if(aux==NULL){
  printf("El dato no se encuentra\n");
 }
 printf("el costo del algoritmo es: %d\n",costo);
}
```

Lista

lista.h

```
#ifndef _lista_
#define _lista_

typedef struct Lista{
```

```
5.  int dato;
6.  int pos;
7.  struct Lista *sig;
8.  struct Lista *ant;
9. }Lista;
10.
11.  typedef Lista *lista;
12.
13.  lista crearlista();
14.  lista agregar(lista l, int dato, int pos);
15.  lista quitar(lista l);
16.  void buscar(lista l, int dato);
17.
18.  #endif
```

lista.c

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include<time.h>
4.  #include "lista.h"
5.
6.  int main(){
7.
8.   srand(time(NULL));
9.   lista l;
10.   l=crearlista();
11.   int tam;
12.   printf("Ingrese tamaño de la lista: ");
13.   scanf("%d",&tam);
14.   for(int i=0;i<tam;i++){
15.    l=agregar(l,rand()%50,i+1);
16.   }
17.   lista aux=l;
18.
19.   while(tam>=0){
20.    printf("Ingrese dato a buscar(entre 0 y 50): ");
21.    scanf("%d",&tam);
22.    if(tam<=-1) {
23.     while(l!=NULL){
```

```
   l=quitar(l);
  }
  exit(0);
 }
 else{
  buscar(l,tam);
 }
}

}

lista crearlista(){
 lista l;
 l=NULL;
 return l;
}

lista agregar(lista l,int dato,int pos){
 lista elem;
 elem=(lista)malloc(sizeof(Lista));
 if(elem==NULL){
  printf("Error: no hay memoria\n");
  exit(0);
 }
 elem->dato=dato;
 elem->pos=pos;
 elem->sig=NULL;
 if(l==NULL){
  l=elem;
  elem->ant=NULL;
 }else{
  lista aux=l;
  while(aux->sig!=NULL){
   aux=aux->sig;
  }
   aux->sig=elem;
   elem->ant=aux;
 }
 return l;
}
```

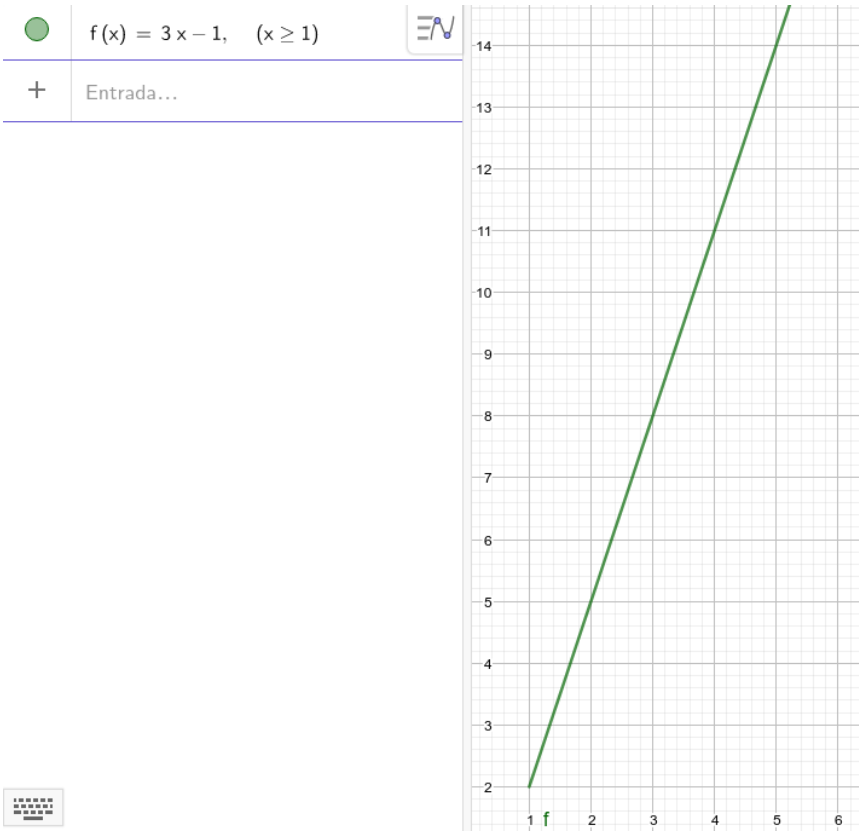
```
64.
65. lista quitar(lista l){
66.  lista aux=l;
67.  if (l->sig==NULL){
68.   free(l);
69.   l=NULL;
70.  }else{
71.   while(aux->sig!=NULL){
72.    aux=aux->sig;
73.   }
74.   aux->ant->sig=NULL;
75.   free(aux);
76.  }
77.  return l;
78. }
79.
80. void buscar(lista l, int dato){
81.  lista aux=l;
82.  int costo=0;
83.  while(aux!=NULL){costo+=1;
84.   if(aux->dato==dato){costo+=1;
85.    printf("El elemento esta en la posicion %d\n",aux->pos );
86.    break;
87.   }else{costo+=2;
88.    aux=aux->sig;
89.   }
90.  }
91.  if(aux==NULL){
92.   printf("No existe el elemento\n");
93.  }
94.  printf("Costo del algoritmo: %d\n",costo );
95. }
```

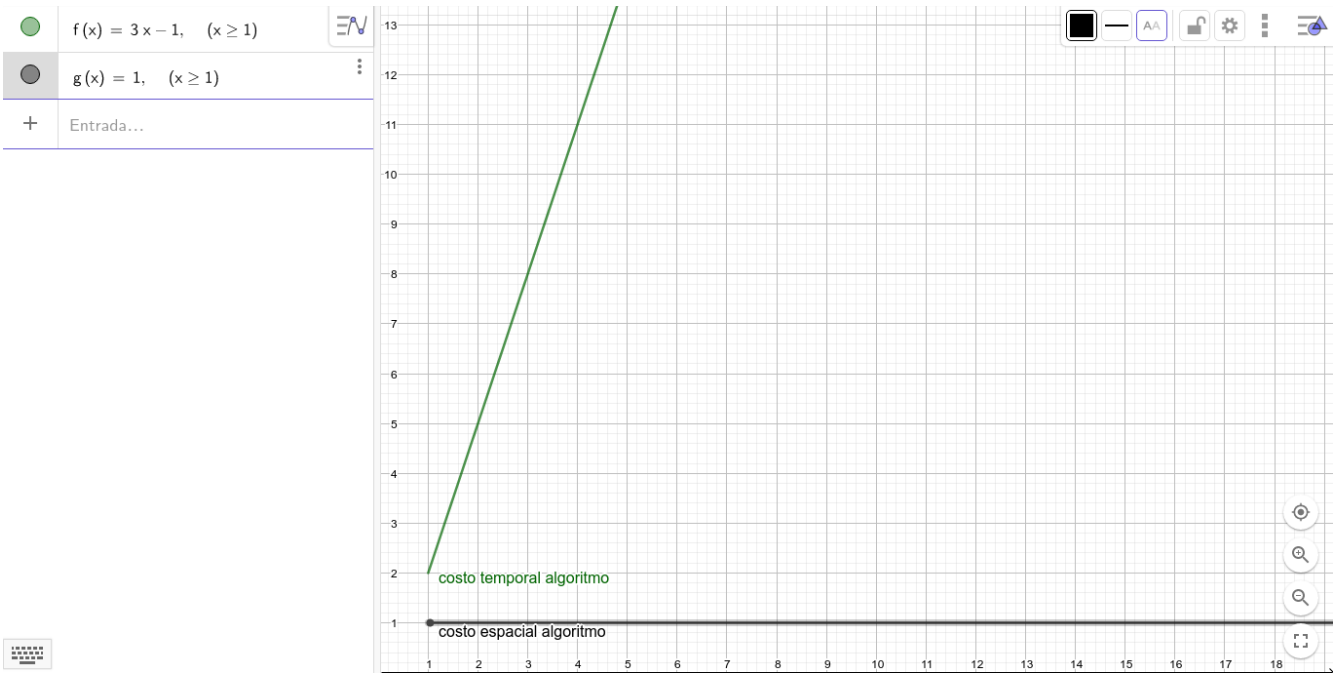
Salidas: los programas se ejecutaron con el compilador gcc de linux, en Arch Linux como sistema operativo, y Sublime Text como editor del codigo, en una laptop marca Asus modelo X540SA, con 2gb de memoria RAM, y procesador Intel Celeron N3050 de 1.60GHz

Entrada(n)	Pila		Cola		Lista	
	Mejor caso	Peor caso	Mejor caso	Peor caso	Mejor caso	Peor caso
100	2	300	2	300	2	300

1000	2	3000	2	3000	2	3000
5000	2	15000	2	15000	2	15000
10000	2	30000	2	30000	2	30000
50000	2	150000	2	150000	2	150000
100000	2	300000	2	300000	2	300000
200000	2	600000	2	600000	2	600000

Con numeros mas grandes, se comienza a trabar la maquina.





La funcion de costo temporal del algoritmo se expresa: *f(x)=3x-1*, con *x*=posicion donde se encuentra un elemento, y siendo *f(x)=3n* , el costo temporal si el elemeto no se encuentra, con *n*=tamaño de elementos en la estructura.

Conclusión

Debido a que las 3 estructuras eran similares, los algoritmos de busqueda tenian el mismo costo temporal, de manera que no se podian optimizar, y por los recursos del hardware, era mas propenso a trabarse con entradas n muy grandes, como por ejemplo, a partir de la prueba *n=50000*, la maquina se tardaba en crear la lista, aguantando solamente hasta la entrada *n=200000*, despues de esos numeros, se trababa y calentaba la maquina.

Ahora bien, como dije, las estrucutras eran similares: de la lista, salen la cola y la pila. Si hubiese usado árbol, otro costo hubiese tenido.