EIE
Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA

TAREA 2: STACK-QUEUE Y ÁRBOLES BST

ESTUDIANTES:
DENNIS CHAVARRÍA SOTO (B82097)

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

II CICLO 2019

Índice

Índice de figuras **1**

1. Solución **3**

1.1. Dada una hilera de caracteres que contenga pares de parentesis, corchetes o boquillas, con letras, sea capaz de decir si dicha hilera esta bien escrita en el sentido de los pares de par ´ entesis; o sea, ´ que siempre que se abra un parentesis, se cierre antes de cerrar uno de otro tipo 3

1.2. Dada una cantidad de colas y una proporcion de prioridad, construir un sistema de colas de ´ prioridad, donde de manera aleatoria, se agreguen elementos a las colas y basada con la proporcion de prioridades, se eliminen los elementos del sistema. . 4

1.3. Utilizando el modelo de arbol de búsqueda binaria realice las siguientes operaciones en orden: ´ a) creacion del árbol ´ b) inserciones: 4, 7, 2, 9, 5, 6, 1, 0, 15, 3 c) borrados: 2, 6, 0, 15 d) balanceo 5

2. Anexos **9**

Índice de figuras

1. Solución

La premisa fundamental al resolver los problemas propuestos fue, no cambiar los demás archivos que no fueran el Main.cpp que invoca toda la lógica del programa, así pues, por cada enunciado relacionado con la implementación de código, se referirá principalmente a los main.

1.1. Dada una hilera de caracteres que contenga pares de parentesis, corchetes o boquillas, con letras, sea capaz de decir si dicha hilera esta bien escrita en el sentido de los pares de par ´ entesis; o sea, ´ que siempre que se abra un parentesis, se cierre antes de cerrar uno de otro tipo

Para el primer programa se instancia un stack y, luego, establecen, al inicio, tres identificadores. El programa cuenta con un condicional para determinar si la cadena ingresada está vacía, en dado caso, se termina el programa y no se ejecuta lógica alguna.

Si la cadena ingresada cumple con lo esperado, los identificadores se establecen como 0 apenas ingresan a un ciclo for que depende de la cantidad de argumentos ingresados (La cantidad de hileras), esto es necesario para pasar por cada hilera que el usuario ingresa, luego otro ciclo, de tipo for, se utiliza para recorrer cada hilera comparando si los elementos son aperturas de paréntesis, corchetes o boquillas, en dado caso aumenta el valor de id2, que corresponde a la cantidad de aperturas, luego compara si se encuentra un elemento de cerrado, y aumenta id3, que indica esta cantidad de terminos. Después de estos condicionales, hay uno tercero; al detectar un elemento de cerrado, compara si su homólogo de apertura está guardado en la última posición del stack, si esto es así, aumenta el valor de id. Si al terminar todos los recorridos, los id´s no son iguales, se concluye que la hiler es inválida. Esto se consigue con el condicional que está al final del método main.

Es importante destacar, cuando se está recorriendo por los términos de las hileras y se revisa el último elemento del stack, esto se consigue gracias a que, en el archivo Stack.h se encuentran diferentes métodos, entre ellos el de peek y el de pop; el primero permite revisar el último elemento de la pila, si coincide con el homólogo de cerrado que se encuentra en la hilera, gracias al recorrido que logra el ciclo for se usa el método del stack llamado pop, el cuál elimina el elemento más reciente agregado. De esta forma, apenas se abre un paréntesis, se esperaría que este sea el primer en ser cerrado, de forma que los más viejos son los últimos. Tal lógica se implementó y permitió el funcionamiento del código.

El resultado que se consigue es el deseado, el usuario introduce las hileras con los paréntesis, por ejemplo, y si no se tiene el respectivo elemento de cerrado, se indica *cuál cola entre las proporcionadas* es la incorrecta.

El archivo main depende de Element.h y Stack.h, que definen alguno métodos para la obtención de valores y formato, en el caso del primero; para el segundo, se define la clase que consta de todos los métodos para agregar, borrar o visualizar elementos en el stack.

1.2. Dada una cantidad de colas y una proporcion de prioridad, construir un sistema de colas de ´ prioridad, donde de manera aleatoria, se agreguen elementos a las colas y basada con la proporcion de prioridades, se eliminen los elementos del sistema.

La solución para este problema se consiguió con el código contenido en el archivo a.cpp; para ellos se requiere de bibliotecas como Chrono y Random, así como las cabeceras Queue.h y Element.h; este programa tiene un método que permite obtener números aleatorios.

En el método main se encuentra toda la lógica del programa y, por medio de variables como elements, rand-num, num-cola y nuevo-elemento, se consigue guardar la cantidad de colas que el usuario desea, luego, la cantidad de items que las colas almacenarán. Entre lo primero se instancia un arreglo de colas, pues, se desea que sea dinámico. El programa debe asignar la cantidad de elementos que guardará cada cola como 0, al inicio; luego, con ayuda de un ciclo for, se añaden elementos en colas aleatorias, estos también lo son y cada numero pseudoaleatorio generado se obtiene gracias al método mencionado inicialmente. Sin embargo, al asignar los elementos, se tienen dos condicionales, para garantizar, al máximo posible, que todas las colas tengan al menos un elemento; es decir, si el numero de cola generado aleatoriamente lleva a que una de ellas sea discriminada y no guarde elementos, estos detectores le asingarán el id de cola, para que le almacenen datos. Después de todo esto, se guardan.

El procedimiento de almacenamiento de datos en cada cola es complicado y susceptible a fallas, por ello, se incluye un ciclo que imprime al usuario los datos que componen las colas. Seguido a tal muestreo, se definen las variables k, p, elemento, prioridad y posición, estas se utilizan en un ciclo para determinar en cuál cola se encuentra, la máxima cantidad de elementos que puede eliminar de ella según la proporción de prioridad establecida. En tal ciclo, hay un primer condicional que realiza lo anteriormente mencionado, luego se aumenta un contador para saber cuando se alcanza el tope por prioridad y otro condicional verifica esta condición; este último tiene otro, el cuál identifica si ya se alcanzó el máximo de prioridad; osea, cuando se debe empezar otra vez con la primera relación de proporción que introduce el usuario. El valor de k tiene la función de que, se utiliza para navegar a través de los valores que definen esta proporción; la variable p indica la cola en la que se encuentra.

Al ejecutar toda la lógica anterior, el resultado es que, el usuario introduce la cantidad de colas, el programa define una cantidad máxima de 17 elementos que serán los que se almacenarán en la cola, luego, a partir de la prioridad (3:2:1 por ejemplo) indicada, se empiezan a eliminar los elementos contenidos y se muestran en la pantalla.

Este programa también incluye el archivo element.h con los mismos métodos que el stack, así como el Queue.h que define los métodos para agregar y eliminar elementos. Como consideración importante, la cola introduce los valores y usa el orden FIFO, el primero agregado es el primero que se elimina, contrario al stack.

1.3. Utilizando el modelo de arbol de búsqueda binaria realice las siguientes operaciones en orden: ´ a) creacion del árbol ´ b) inserciones: 4, 7, 2, 9, 5, 6, 1, 0, 15, 3 c) borrados: 2, 6, 0, 15 d) balanceo

En las siguientes figuras se encuentra la descripción del procedimiento que se debe tomar para cumplir con los requisitos del problema, sin embargo, para complementar, se destaca que para el inciso a, se debe instanciar o crear un árbol que tendrá un primer elemento, la raíz, tal y como se explica, sin embargo, esta apuntará a las primeras ramas, que serán el hijo mayor y el menor, a partir de aquí se debe considerar para el inciso b que, los hijos menores al valor de la raíz (el 4) siempre van a la izquierda, y, siempre, los hijos mayores al valor agregado a la izquierda, inmediatamente después de la raíz, van a la derecha; a la izquierda de este irán los menores; lo mismo ocurre con el lado derecho, los hijos del nodo con valor igual a 7. Esta relación debe mantenerse entre todos los descendientes.

Al borrar los elementos, para el inciso b, se tiene que, el puntero que indicaba la dirección de memoria del nodo eliminado, debe cambiar. Al hacer el diagrama se consigue borrando lo que se desea, pero una implementación computacional implica que el puntero, o la dirección de siguiente elemento, indique un nuevo nodo, para ello, como se explica en las siguientes figuras, se debe elegir entre el nodo que contiene el mayor valor que se encuentre en la raíz izquierda, sino, se elige el menor a la derecha y al que se elige, se apunta con el puntero del hijo borrado para el nodo inmediatamente anterior.

Considérese que los signos de pregunta en la parte c, indican que ahí se ha borrado un nodo, para el caso de la rama izquierda, se ve que hay una flecha que ahora apunta al nodo con el valor 3, pues, ese será el que rebalanceará tal rama. Entonces el árbol c indica

3) Utilizando el modelo de árbol de búsqueda binaria, realice las siguientes operaciones en orden.

a)

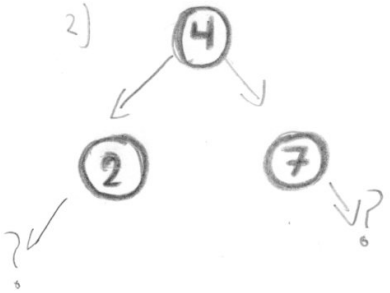
Se establece el primer nodo (La raíz) con condicionales se dice que esta almacenará las direcciones de memoria de sus hijos izquierdo y derecho, pero, como no existen, se asignan los punteros como 0x0. (estos nodos guardan un valor numerico)

b) 1)

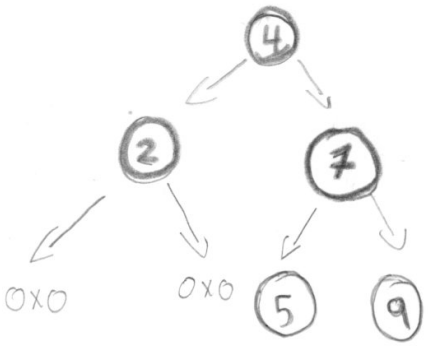
Inserciones (4,7,2,9,5,6,1,0 15,3)

Para las inserciones, se compara con la raíz, su valor, con el de los elementos que van a agregarse; el mayor se agrega a la derecha, el menor a la izquierda. (Así inserta 2 y 7)

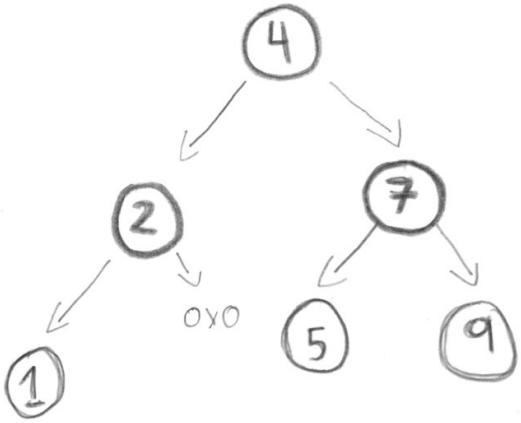
2)



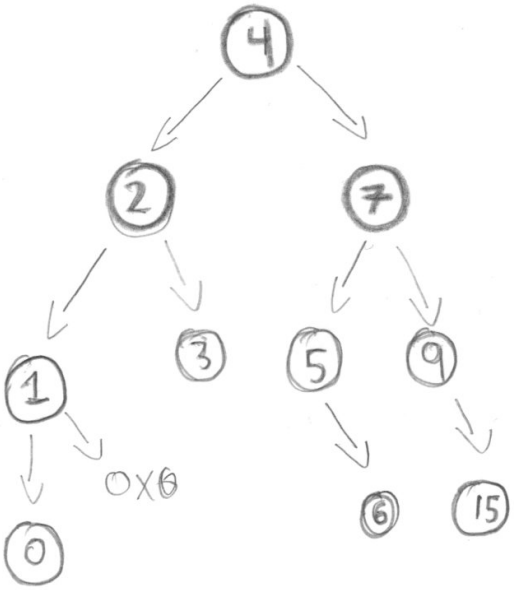
Para los hijos de los hijos, se compara, primero, si son menores o mayores que cuatro (El valor de la raíz) basado en ello, se envían a un lado u otro; si son menores o mayores, por ejemplo, 9 y 5 (son los que siguen en la lista de inserciones) estos son mayores que 4, pero el $5 < 7$ y $9 > 7$, entonces, el primero va a la izquierda, el segundo a la derecha.



Tal como se muestra en el diagrama izquierdo.
El método consiste en asignar nuevos espacios de memoria que sean nodos; los hijos 0x0 (dirección de los punteros) pasan a apuntar a estos nuevos espacios de memoria.



Para el valor 1 de la lista de inserción, se compara con la raíz, se concluye que va a la izquierda, así mismo, es menor que 2, entonces, es hijo izquierdo, la dirección del hijo pasa de 0x0 a la del nuevo espacio de memoria



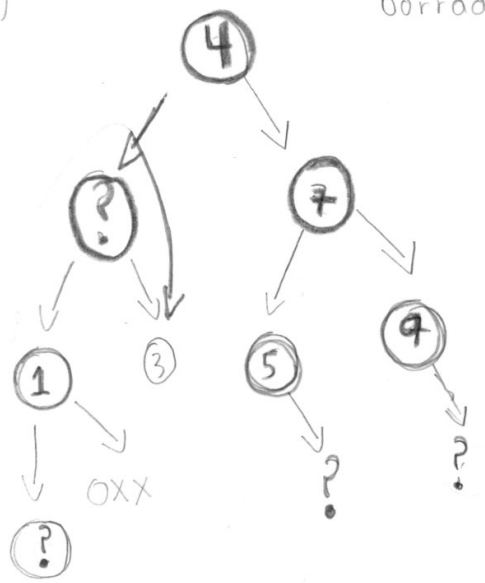
Para el 0; es menor que 4 y menor que 2; a su vez, es menor que 1, por ende, va a extremo inferior izquierdo del árbol.

Para el 3; 3<4 -> hijo a la izquierda, 3>2 -> el nodo con el valor "2" tiene como hijo derecho un nodo con valor = 3

Para el 15: 15>4 ∧ 15>7 ∧ 15>9, entonces, el hijo derecho del 9 es 15; su puntero apunta ahora al nodo con valor 15

Para el 6: (6>4, 6<7 ∧ 6>5) => el 6 es hijo derecho (mayor) del nodo con valor 5; así, la dirección de tal puntero es hacia el espacio de memoria del nodo con valor = 6.

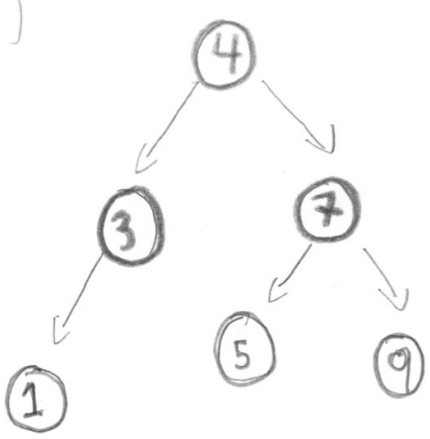
c) borrados: 2, 6, 0, 15



Al borrar los nodos, el puntero de la raíz o el nodo anterior al que se borra, apuntará al más pequeño a la derecha de los descendientes del elemento eliminado, a su vez, puede ser el más grande del lado izquierdo.

Así, la raíz apunta al más grande de la izquierda, de forma que rebalancea el árbol.

d)



Al efectuar el cambio de puntero anterior, se rebalancea el arbol y se obtiene el árbol izquierdo

2. Anexos

```cpp
#include <iostream>
#include <cstring>
#include "Element.h"
#include "Stack.h"

using namespace std;
#define Data Element<char> // vagancia

int main(int argc, char **argv)
{
    Stack<char> s;
    int id=0;
    int id2=0;
    int id3=0;
    if (argc==1){
        cout<<"Cadena vacia";
        return(0);
    }
    for (int k=1; k<argc; k++){
        id=0;
        id2=0;
        id3=0;

        for (int i=0; i<(strlen(argv[k])); i++) {
            if ((argv[k][i]=='(') || (argv[k][i]=='{') || (argv[k][i]=='[')|| (argv[k][i]=='<')){
                s.push(argv[k][i]);
                id2++;
            }
            if ((argv[k][i]==')') || (argv[k][i]==']') || (argv[k][i]=='}')|| (argv[k][i]=='>')){
                id3++;
            }
            if((argv[k][i]==')') || (argv[k][i]=='}') || (argv[k][i]==']')|| (argv[k][i]=='>')){
                if ((argv[k][i]==')' && s.peek()=='(')||(argv[k][i]=='}' && s.peek()=='{')||(argv[k]
                    s.pop();
                    id++;
                }
            }
        }
        if ((id==id2)&&(id2==id3)){
            cout<<"Cadena: "<<k<<" balanceada "<<endl ;
        }
        else{
            cout<<"Cadena: "<<k<<" no balanceada "<<endl ;
        }

    }

}
```

Listing 1: mainBP.cpp

```cpp
#include <iostream>
#include "Element.h"
#include "Queue.h"
#include <chrono>
#include <random>
#include <string>

#define Data Element<double> // vagancia
using namespace std;

int rand_gen(int elements){
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator (seed);
    std::uniform_int_distribution<int> distribution(0, elements-1);
    int number=distribution(generator);
    return(number);
}

int main(int argc, char* argv[])
{
    int elements;
    int rand_num;
    int num_cola;
    int nuevo_elemento;

    elements=int(argv[1][0])-48;
    int rem_itms[elements];

    Queue<int> q[elements];

    for (int i=0; i<elements; i++){
        rem_itms[i]=0;
        q[i]=Queue<int>();
    }

    for (int i=0; i<17; i++){
        num_cola=rand_gen(elements);
        if (i>=8){
            for (int k=0; k<elements; k++){
                if (rem_itms[k]==rem_itms[k-1]){
                    num_cola=k-1;
                }
                if (rem_itms[k]<=2 || rem_itms[k]==0){
                    num_cola=k;
                }
            }
        }
        nuevo_elemento=rand_gen(40);
        cout<<"                    ";
        q[num_cola].enqueue(nuevo_elemento);
        rem_itms[num_cola]++;
    }
        for (int c=0; c<elements; c++){
        cout<<endl;
        cout<<"Cola "<<c+1<<":";
        for(int i=0; i<rem_itms[c]; i++){
            int elemento = q[c].dequeue();
            q[c].enqueue(elemento);
            cout<<elemento<<" ";
```

```
60          }
61      }
62
63      int k=0;
64      int p=0;
65      int elemento;
66      int counter=0;
67      char prioridad;
68      int posicion=0;
69      cout<<endl;
70      cout<<endl;
71
72      for (int i=0; i<17; i++){
73          elemento=q[p].dequeue();
74          if (rem_itms[p]>0){
75              cout<<elemento<<" ";
76              rem_itms[p]--;
77              i--;
78          }
79          counter++;
80          prioridad=argv[2][k];
81          posicion=int(prioridad)-'0';
82
83          if (counter==posicion){
84              k=k+2;
85              p++;
86              counter=0;
87              string Dato2=argv[2];
88
89              if (k>Dato2.size()){
90                  k=0;
91                  p=0;
92              }
93
94          }
95
96      }
97  }
```

Listing 2: a.cpp

```
1
2
3  #include <vector>
4
5  template <typename D>
6  class Stack
7  {
8
9  public:
10     Stack(){};
11
12     Stack(const Stack &orig){};
13
14     ~Stack(){};
15
16     void push(D d)
17     {
18         this->a.push_back(d);
19     }
20
21     D pop()
22     {
23         if (this->a.size() > 0)
24         {
25             D d = this->a[a.size()-1];
26             a.pop_back();
27             return d;
28         }
29         else
30         {
31             D d;
32             return d;
33         }
34     }
35
36     D peek()
37     {
38         if (this->a.size() > 0)
39         {
40             D d = this->a.at(0);
41             return d;
42         }
43         else
44         {
45             D d;
46             return d;
47         }
48     }
49
50 private:
51     vector<D> a;
52 };
```

Listing 3: Stack.h

```
1
2
3  #include <vector>
4
5  template <typename D>
6  class Queue
7  {
8
9  public:
10     Queue(){};
11
12     Queue(const Queue &orig){};
13
14     ~Queue(){};
15
16     void enqueue(D d)
17     {
18         this->a.push_back(d);
19     }
20
21     D dequeue()
22     {
23         if (this->a.size() > 0)
24         {
25             D d = this->a.at(0);
26             auto it = this->a.begin();
27             this->a.erase(it);
28             return d;
29         }
30         else
31         {
32             D d;
33             return d;
34         }
35     }
36
37 private:
38     vector<D> a;
39 };
```

Listing 4: Queue.h