Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Licenciatura en Computación

Algoritmos y Programación
Noviembre 2019

Alumnos: Sebastian Russian

Proyecto#3

Para el desarrollo de este proyecto se nos solicitó trabajar con las estructuras de Árboles.

```
    if(!l.isEmpty())
    {//creación del segundo arbol
        huffman( *(l.get(l.first())), q);
        l.erase(l.first());

        CastingEst(q,ll);
        plantar(ll,deco);
    }


    while(!l.isEmpty())
    {
        string pTemp= *(l.get(l.first()));
        for(int i=0;i<pTemp.length();++i ){
            path(*(codi.getRoot()),pTemp[i],s );

            castingBinario(s, lc);
            way(*(deco.getRoot()), lc, sol );
            s="";
        }
        out.insert(sol,out.last());



        l.erase(l.first());
    }

    output(out);
    return 0;
}
```

Este es el orden por el cual se rige nuestro algoritmo.

Como siempre la parte inicial de un código es la declaración de variables y la inclusión de librerías justo por eso nos adelantamos directamente al punto en el que los datos principales ya están leídos desde el archivo y pasan a ser procesados por el resto del código.

Como se puede apreciar lo que inicia la construcción de los arboles es un condicional el cual verifica si hay información en la lista con la cual se pueda realizar la función Huffman la cual nos regresara la cantidad de veces que se repite cada letra y la propia letra, esto será almacenado en una estructura que contiene dichos datos.

```
void huffman(string &s,sebas::queue<letras> &q)
{//se consigue el codigo huffman para crear el arbol
  int c[50];
  string order="";
  for(int i=0;i<50;++i) //se iguala todo el arr a 0
    c[i]=0;

    for(int i=0;i<s.length();++i){//se verifica cuantas veces se repite cada letra
      if(s[i]<'a')
        ++c[s[i]-'A'];
      else
        ++c[s[i]-'a'];
    }
  letras aux;
  for(int i=0;i<s.length();++i){
    if(!isHere(order,s[i]))
      order+=s[i];
  }
  for(int ii=0;ii<order.length();++ii){
    if( q.size() < order.length() )
      for(int i=0;i<50;++i){
        if(c[i]>0){
          if(i<25){
            aux.l=i+'A';
            aux.c=c[i];
            c[i]=0;
          }else{
            aux.l=i+'a';
            aux.c=c[i];
            c[i]=0;
          }
          q.enqueue(aux);
        }
      }
  }
}
```

```
struct letras
{
    char l;
    int c=0;
};
```

Dicha estructura será utilizada por varias listas y colas para preservar el orden en el que se leyeron para luego ser creado el árbol.

```
void plantar(sebas::list<letras> &l, BST<string> &t){
    if(!l.isEmpty()){
        sebas::list<letras>::pos pos=l.first();
        int c=0;
        letras aux;
        while(pos!=l.last()){
            aux=w(l.get(pos));
            c+=aux.c;
            l.next(pos);
        }
        string node;
        if(l.size()>1){
        node="$" + to_string(c);
        t.insert(node);
        }

        aux=w(l.get(l.first()));
        node=aux.l + to_string(aux.c);

        t.insert(node);
        l.erase(l.first());
        plantar(l,t);
    }
}
```

Luego de eso entramos en un while el cual estará cada línea del archivo después de las dos (2) la cuales a través de las mismas se logra conseguir la codificación con la función “path” la cual se usa para indicar el camino a recorrer en el 2do árbol usando la función “way”, las palabras obtenidas se guardan en una lista la cual será escrita en el archivo de salida con “output”

```
template <class T>
void leer(sebas::list<T> &l)
{//lee el archivo
    string s;
    ifstream archivo_entrada("Entrada.txt");
    if(archivo_entrada.fail()){
        cout<<"Error en la apertura del archivo.\n";
         archivo_entrada.close();
    }
    while(getline(archivo_entrada, s))
        l.insert(s,l.last());
    archivo_entrada.close();
}
```

Función encargada de leer el archivo y guardar la información

```
void path(node<string> &Pnode, char ToFind, string &sol
){//codigo binario
    string aux= Pnode.getInfo();
    if(Pnode.left == NULL and Pnode.right == NULL and aux[0] != ToFind ){
        cout<< "\nKEOPS\n"<<endl;
    }else if(aux[0] == ToFind){
        sol=sol;
    }else if(Pnode.left!=NULL){
        path( *(Pnode.getLeftChild()), ToFind, sol+="0");
    }else if(Pnode.right!=NULL){
        path( *(Pnode.getRightChild()),ToFind, sol+="1");
    }
}
```

Función con la cual se consigue la codificación del 1er árbol

```
void way(node<string> &Pnode, sebas::list<char> &l,string &sol)
{//sigue el camino indicado por el codigo binario//convierte el codigo binario en una lista
    string aux= Pnode.getInfo();
    char ToFind= '/';
    if(!l.isEmpty())
        ToFind= *(l.get(l.first()));
    if( l.isEmpty() ){
        sol+=aux[0];
    }else if(Pnode.left!=NULL and ToFind == '0'){
        l.erase(l.first());
        way( *(Pnode.getLeftChild()), l, sol );
    }else if(Pnode.right!=NULL and ToFind == '1'){
        l.erase(l.first());
        way( *(Pnode.getRightChild()),l, sol );
    }
}
```

Función que recorre el segundo árbol usando código binario