INTRODUÇÃO

problema a ser resolvido:

O problema a ser resolvido é pegar um arquivo de tamanho razoavelmente grande e transformá-lo em um arquivo menor utilizando o algoritmo de huffman.

conceito geral:

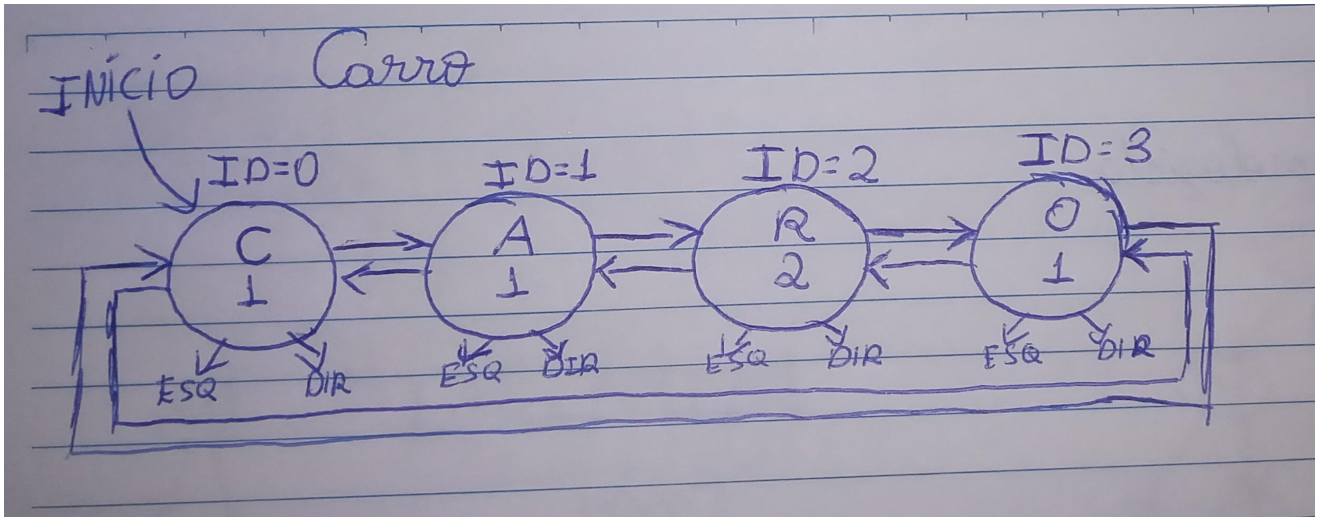
O algoritmo de huffman usa a frequência das letras do arquivo para formar uma árvore, através dessa árvore cria um código de binário que por meio desta mesma árvore usa como localização da palavra na árvore, os processos feitos até a criação da árvore no meu algoritmo são os seguintes: inicialmente ele coloca todos os caracteres do arquivo em um vetor depois busca pelo vetor as palavras e sua frequência,após isso ele percorre a lista e vê se a palavra já foi colocada na lista caso ele não encontre ele adiciona a nova palavra e sua frequência a lista caso encontre passa para a próxima palavra para que não haja repetições na lista, após isso é criada a árvore a partir da lista que é feita pegando os dois nós que possuem as duas menores frequência e cria-se um nó ligando ambos a sua esquerda e a sua direita, depois coloca-se esse nó a lista repetindo esse mesmo processo até que hajam apenas dois valores a lista esse dois últimos serão ligados pela raiz, com a árvore feita pega-se a sequência binária através do caminho feito da raiz às folhas(1 para direita e 0 para esquerda), depois basta trocar as palavras pelo seu referente binário junto com isso são guardados os dados necessários para escrever a árvore sem escrever as palavras que estão no arquivo normal por isso escrevi as palavras da lista junto a frequência(que não se repetem sendo assim menor que o arquivo inicial) junto com o tamanho da lista, através desses dados é refeito toda a árvore, prosseguindo isso basta transformar os 0 e 1 nas em suas respectivas palavras, para isso pega-se o número que está no arquivo caminhando para esquerda ao ler 0 e para direita ao ler 1 assim que chegar a uma folha onde as letras estão guardadas escreve a letra e volta para raiz para iniciar o processo até ter transformado todo o arquivo de 0 em 1 no texto escrito do arquivo inicial.

DESENVOLVIMENTO

```c
typedef struct no{
    int id;
    int frequencia;
    char bin[30];
    char letra;
    struct no* dir;
    struct no* esq;
    struct no* pai;
    struct no* prox;
    struct no* ant;
} NO;
int localizadorId = 0;
int tamLista = 0;
NO* inicio = NULL;
```

implementação

Tendo em vista que a posição do caractere na lista não tinha importância acreditei que seria melhor fazer de uma lista duplamente encadeada circular com um ponteiro de início, sendo assim na estrutura o código faz uso das variáveis letra,frequência,bin(guarda o valor binário equivalente a letra),ID(para que eu possa identificar todos os nós criados inclusive os que não possuem letra) para saber em qual ID o código estar usa-se uma variável global localizadorID, voltando a falar do nó ele possui os ponteiros de prox, ant, pai, esq e dir então ao trabalhar com a palavra de exemplo "carro" a lista formada é a seguinte.



por ser uma lista duplamente encadeada circular e todo valor ser sempre adicionado no “fim” essa ação é sempre constante independente de quantos valores serão adicionados, para adicionar são feitos 3 passos:

PASSO 1

```c
//pega o todo arquivo enviado e trasnforma em um vetor de caracteres
void pegandoArquivo(FILE* arq,char palavra[]){
    char auxTexto[1];
    while(fgets(auxTexto,2,arq) != NULL){
        strncat(palavra,auxTexto,1);
    }
}
```

PASSO 2

```c
//busca na lista encadeada a letra retornando 1 se não achar e 0 se achar
int buscarPalavra(char letra){
    NO* aux = inicio;
    for(int i = 0;i<tamLista;i++){
        if (aux->letra == letra){
            return 0;
        }
        aux = aux->prox;
    }
    return 1;
}
//percorre a palavra e põe na lista os caracteres que não são repetidos
void percorrerPalavra(char palavra[]){
    char letra;
    for(int i = 0;i < strlen(palavra);i++){
        letra = palavra[i];
        //contador de frequencia
        int contador = 0;
        for(int j = 0;j<strlen(palavra);j++){
            if(palavra[j] == letra){
                //se ao percorrer a palavra a letra reapareceu incrementa
                contador++;
            }
        }
        //testa se a letra ja está na lista encadeada circular não estiver adciona ela
        //função buscar palavra está acima
        if(buscarPalavra(letra) == 1){
            addLista(letra,contador);
        }
    }
}
```

PASSO 3

```c
//adiciona um nó na lista e adciona um ID para identifica-lo e incrementa o ID para que os valores n se repitam
void addLista(char letra,int frequencia){
    NO* novo = malloc(sizeof(NO));
    novo->letra = letra;
    novo->id = localizadorId;
    novo->frequencia = frequencia;
    //para que não de erro na arvore e as letras serão folhas então esq e dir é NULL
    novo->esq = NULL;
    novo->dir = NULL;
    if(tamLista == 0){
        inicio = novo;
        novo->ant = novo;
        novo->prox = novo;
    }else{
        inicio->ant->prox = novo;
        novo->ant = inicio->ant;
        inicio->ant = novo;
        novo->prox = inicio;
    }
    tamLista++;
    localizadorId++;
}
```

Após ter a lista montada inicia-se o processo de formação da árvore, essa formação é feita pegando os dois nós de menor frequência da lista e criando um novo nó que por sua vez recebe a frequência como sendo a soma da frequência dos dois nós ao mesmo tempo que seu esq e dir apontam para os nós sendo assim os nós recebem como pai esse novo nó que é devolvido a lista para que o processo continue e seja repetido até sobrar apenas dois valores(que serão sempre o início e o próximo depois dele) a qual serão ligados pela raiz.

```
//começa a criar a arvore a partir da folha feita pegando os valores com as 2 menores frequencia
void criarArvore(){
    while(tamLista > 2){
        NO* menor1 = buscarMenor();
        removerLista(menor1->id);
        NO* menor2 = buscarMenor();
        removerLista(menor2->id);
        //enquanto a lista for maior que dois ele vai pegando os dois nos com menores frequencias da lista e os remove
        //e adiciona um no na arvore ao mesmo tempo que adciona esse no a lista e prossegue
        addNoLista(addArvore(menor1,menor2));
    }if(tamLista == 2){
        //ao final cria a raiz com os dois ultimos valores que serão sempre o inicio e o proximo depois dele
        criarRaiz(inicio,inicio->prox);
    }
}
```

```
//essa função perccore a lista buscando os menores valores dela
NO* buscarMenor(){
    NO* aux = inicio;
    NO* menor = inicio; //o menor começa do inicio
    for(int i = 0;i < tamLista ; i++){
        if(aux->frequencia <= menor->frequencia){ //testa se o valor é realmente o menor
            menor = aux;
        }
        aux = aux->prox;
    }
    return menor;
}
```

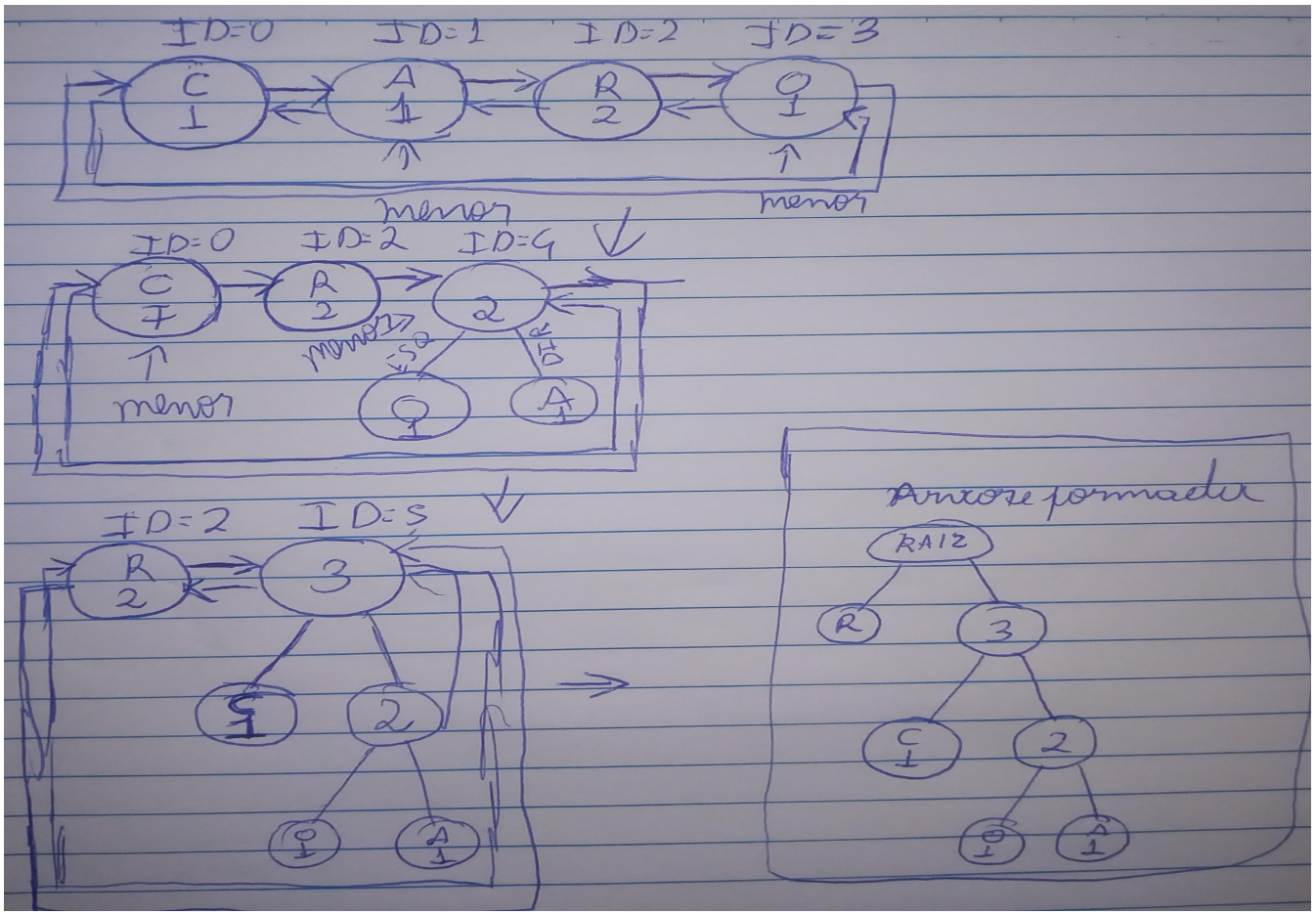
```
//isso "remove" da lista repare que ele não da free no lixo pois o valor removido é colocado na arvore
NO* removerLista(int id){
    NO* aux = inicio;
    if(inicio->id == id){
        //caso o valor que será removido seja o inicio faz com que o proximo seja o novo inicio
        inicio = inicio->prox;
    }else{
        while(aux->id != id){
            //percorre o vetor atras do ID que será removido
            aux = aux->prox;
        }
    }
    //faz com que os nos anteriores e posteriores ao aux ignorem ele veja que eu não dei free no ponteiro pois ele será usado na arvore
    aux->ant->prox = aux->prox;
    aux->prox->ant = aux->ant;
    tamLista--;
    return aux;
}
```

```
//essa função cria o nó na arvore que liga os valores com 2 menores frequencia e retorna ele para que seja adcionado a lista
NO* addArvore(NO* menor1, NO*menor2){
    NO* novo = malloc(sizeof(NO));
    novo->esq = menor1;
    novo->dir = menor2;
    //eu ultilizei o conceito de pai na arvore para pegar o binario
    menor1->pai = novo;
    menor2->pai = novo;
    novo->frequencia = menor1->frequencia + menor2->frequencia;
    novo->id = localizadorId;
    localizadorId++;
    return novo;
}
```

```
//adiciona um NO sem letra na lista no caso as junções da arvore a pós se ligarem aos 2 menores devem entrar na lista para presseguir
//o processo
void addNoLista(NO* novo){
    inicio->ant->prox = novo;
    novo->ant = inicio->ant;
    inicio->ant = novo;
    novo->prox = inicio;
    tamLista++;
}
```

```
//ao final do precesso é necessario criar a raiz da arvore onde essa função é chamada
void criarRaiz(NO* no1,NO* no2){
    NO* novo = malloc(sizeof(NO));
    novo->esq = no1;
    novo->dir = no2;
    no1->pai = novo;
    no2->pai = novo;
    novo->frequencia = no1->frequencia+no2->frequencia;
    //o pai da raiz é NULL
    novo->pai = NULL;
    novo->id = localizadorId;// da um ID para o novo no
    localizadorId++;//incrementa para que não haja ID repetido
    raiz = novo;//define o no como raiz
}
```

Agora para fique mais claro aqui vai o exemplo “carro”



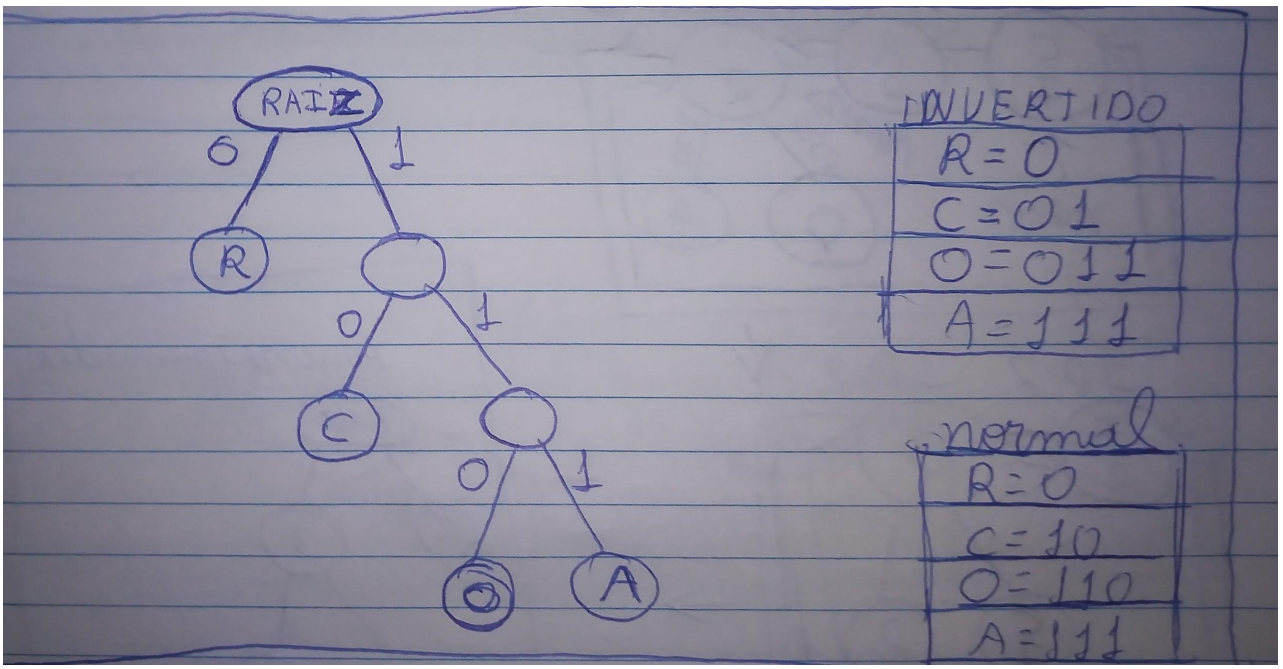
Após ter criado a árvore é necessário percorrer a árvore da raiz a folha onde para esquerda equivale a 0 e para direita equivale a 1 no entanto o meu algoritmo usou algo diferente, ao invés de ir da raiz à folha ele foi da folha até a raiz utilizando a ideia de pai e gerando assim uma variável char binário ao contrário da qual deveria ser, devido a isso foi necessário chamar uma função que inverte a string tornando o binário correto basicamente desinvertendo o binário e junto com isso ele mostra o binário formado junto com sua letra ao final.

```c
//inverte string minha ja que a ideia é ler a partir da folha até a raiz depois inverter a ordem do valor binario criado
void inverter(char palavra[]){
    int tam = strlen(palavra);
    for(int i = 0; i < tam/2; i++) {
        char aux = palavra[i]; //armazena o character inicial
        palavra[i] = palavra[tam-1-i]; //Troca o character da ponta oposta -1 pois o ultimo valor da string é '\0'
        palavra[tam-1-i] = aux; //o ultimo recebe o primeiro
    }
}
```

```c
//percorre a arvore criando o codigo binario de cada letra
void guardarHuffman(NO* aux ,NO* aux2){
    //aux: variavel quer irá a percorrer o array
    //aux2:guarda a variavel inicial que vai guardar o valor binario
    if(aux->esq == NULL && aux->dir == NULL){
        printf("%c:",aux->letra);
    }
    if(aux->pai != NULL){
        //guardo o valor binario criado a cada caminho feito
        //adcionar 2 apenas porque ele espera que seja 2 strings
        if(aux->pai->esq == aux){
            strncat(aux2->bin," 0 ",2);
        }
        if(aux->pai->dir == aux){
            strncat(aux2->bin," 1 ",2);
        }
        guardarHuffman(aux->pai,aux2);
    }else{
        //ao final o binario gerado vem ao contrario por isso é necessario inverte-lo
        inverter(aux2->bin);
    }

}
```

Para que a ideia fique mais clara, novamente irei mostrar o exemplo da palavra carro



Tendo todos esses dados já é possível transformar o arquivo em um código de binário, no entanto antes disso é necessário escrever os dados necessários para montar a árvore, para isso basta percorrer a lista encadeada circular criada logo no começo e antes de transformá-la em árvore a escrevemos no arquivo.

```c
//escreve os dados necessarios para reconstruir a arvore
void escrever_cabecalho(FILE* saida){
     NO* aux = inicio;
     for(int i = 0;i<tamLista;i++){
          fprintf(saida,"%c",aux->letra);
          fprintf(saida,"%d",aux->frequencia);
          aux = aux->prox;
     }
}
```

```c
void escrever_arvore_corpo(FILE* entrada,FILE* saida){
    char letra[1];
    //como eu ja tinha pego o arquivo e percorrido ele todo tive que chamar ele pro começo atravez do rewind
    rewind(entrada);
    fprintf(saida,"\n");
    while(fgets(letra,2,entrada) != NULL){
        buscarNo(letra[0],raiz,saida);
    }
}
```

```c
//busca o NO equivalente a letra
void buscarNo(char letra,NO* aux,FILE* saida){
     if(aux->esq != NULL){
          buscarNo(letra,aux->esq,saida);
     }
     if(aux->dir != NULL){
          buscarNo(letra,aux->dir,saida);
     }
     if(aux->esq == NULL && aux->dir == NULL && aux->letra == letra){
          //escreve o valor binario referente a letra
          fwrite(aux->bin,1,strlen(aux->bin),saida);
     }
}
```

.

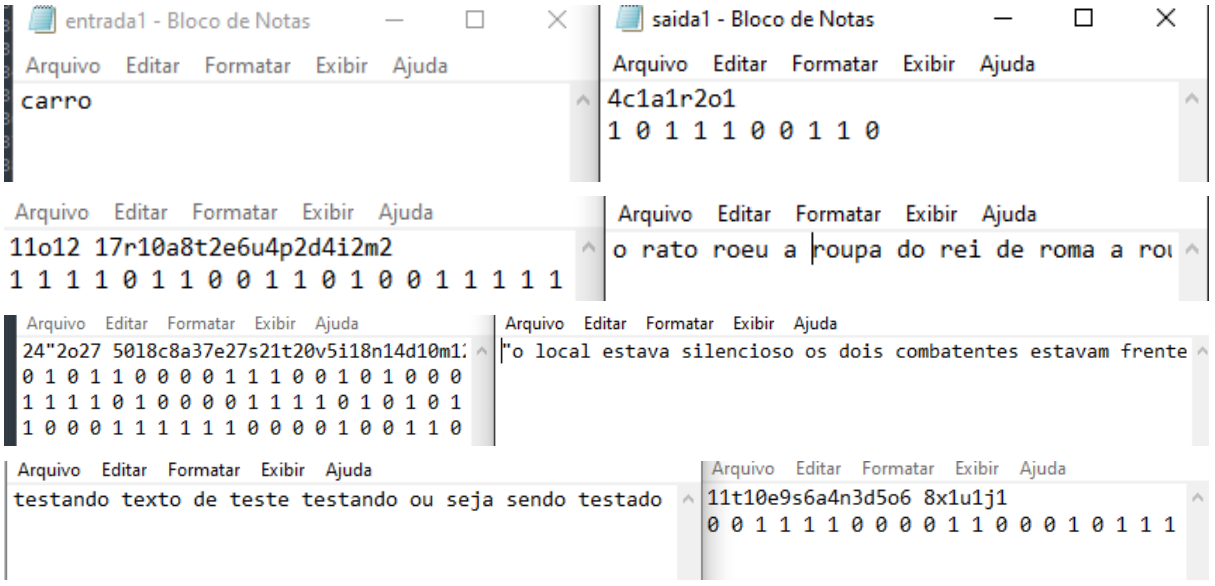
Com isso todo o arquivo foi decodificado agora para a decodificação é feita a remontagem de todo a árvore utilizando dos dados da listas e a montando chamando todas as funções anteriormente a partir da parte do “addlista” e logo após isso percorre-se o arquivo escrevendo as letras referente ao seu código binário

```c
void percorrerArq(FILE* entrada){
    int tam = 0;
    //inicia pegando o tamanho do arquivo
    fscanf(entrada,"%d",&tam);
    for(int i = 0;i < tam ; i++){
        char letra;
        int frequencia;
        fscanf(entrada,"%c",&letra);
        fscanf(entrada,"%d",&frequencia);
        addnoLista(frequencia,letra);
    }
}
```

```c
//após pegar a arvore bastar remontar o arquivo
void descompactar(FILE *entrada,FILE* saida){
    char letra[1];
    //o while estava interferindo com uso de variaveis locais dentro da função por isso tive que usar auxiliar global
    auxiliar = raiz;
    while(fgets(letra,2,entrada) != NULL){
        if(letra[0] == '0'){
            auxiliar = auxiliar->esq;
        }
        if(letra[0] == '1'){
            auxiliar = auxiliar->dir;
        }
        if(auxiliar->esq == NULL && auxiliar->dir == NULL){
            //ao chegar na folha escreve a letra e volta para raiz e reinicia todo processo até decodificar todo arquivo
            fprintf(saida,"%c",auxiliar->letra);
            auxiliar = raiz;
        }
    }

}
```

Ademais vou mostrar alguns exemplos do código rodando no entanto ele não compacta e sim codifica o arquivo por isso o tamanho é até maior do arquivo inicial apesar de que em caso de teste com arquivos que utiliza de caracteres especiais ou formatação específica fique menor isso é unicamente pelo fato de que esses caracteres não são escritos corretamentes e o arquivo gerado é ilegível até depois da “descompactação”.



conclusão

o trabalho foi de fato uma tarefa árdua no entanto sua conclusão(meio que incompleta)foi possível devido a um vídeo compartilhado por um dos discentes(link na biografia) e através desse vídeo eu criei a logica por tras adaptando para o que eu acreditava ser a melhor forma de fazê-lo munindo ele a algumas “artimanhas”, junto a isso o maior dos desafios foi escrever bits em um arquivo por esse motivo o arquivo não pode ser compactado no mais eu diria que foi um grande trabalho com pequenos no entanto visíveis defeitos

BIBLIOGRAFIA

vídeo:link do vídeo usado de base
site usado para aprender arquivos:link do site
site para função inverter:link do site