Estruturas utilizadas da STL

vector

Header: <vector>

Exemplo de declaração: vector< T > vetor;

Definição: vetor de elementos T redimensionável.

Porque: Esta estrutura foi utilizada porque permite acesso em tempo constante aos elementos, ao contrário de um heap (como o map por exemplo). Em contrapartida foi necessário incluir elementos "extras" para que cada símbolo do código ASCII tivesse seu elemento correspondente no vetor, mesmo que este não estivesse presente no texto a ser compactado/descompactado. Mas isto não representou um custo adicional excessivo já que o código ASCII considerado tem 256 símbolos.

Esta classe também foi utilizada para armazenar vetor de bits dos códigos do texto compactado.

string

Header: <string>

Exemplo de declaração: string linha;

Definição: array de caracteres redimensionáveis.

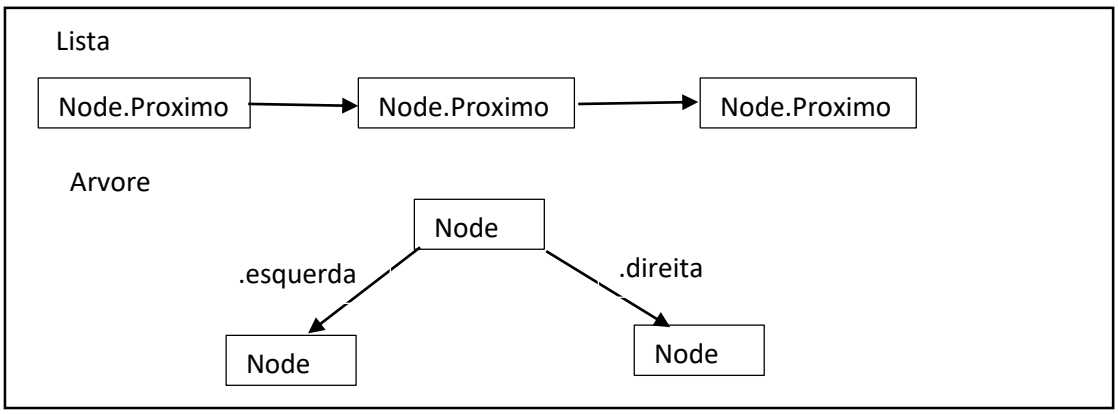
Porque: esta classe foi principalmente utilizada para armazenar o texto lido dos arquivos até que ele fosse processado pelo compactador. Também foi utilizada para gravar a ordem dos símbolos na arvore.

Estrutura implementada

Node

Definição: nó de uma árvore ou lista.

Porque: Esta classe é utilizada na implementação da arvore de Huffman. Cada elemento tem um ponteiro para os seus nó filhos da direita e da esquerda. Também foi utilizada para representar uma lista dos símbolos do alfabeto ordenada de forma crescente de acordo com a frequencia do símbolo no texto.



Principais pontos do código

Primeiro é analisado o comando passado como primeiro parâmetro, que pode ser “c” ou “d”. Se for “c” o arquivo indicado será compactado, se for “d” será descompactado.

```
if(argv[1][0] == 'c'){
   Compactar(argv[2], argv[3]);
}else if(argv[1][0] == 'd'){
   Descompactar(argv[2], argv[3]);
}
```

Compactar

Para contar a frequencia dos símbolos, o arquivo é lido linha por linha. Para cada caractere encontrado é somando 1 ao respectivo elemento do vetor frequência. Por exemplo se for encontrada a letra “A” o elemento 65 do vetor incrementado.
Exemplo: frequencia[‘A’]++ -> frequencia[65]++

```
//-------------------------------------------------------------------------|
//Etapa 1. Contar a frequencia dos simbolos                                |
//-------------------------------------------------------------------------|
getline(entrada, linha);//Lê uma linha do entrada
do{
   //Para cada caracter da linha
   for(unsigned int aux = 0; aux < linha.size(); aux++){
      T++;//Soma mais um ao tamanho total do entrada
      (frequencia[linha[aux]])++;//soma mais um a frequencia do caracter
   }
   //Lê a proxima linha
   getline(entrada, linha);
}while(!(entrada.eof()));//Enquanto não alcançar o fim do entrada
```

Com a frequência calculada, os simbolos com frequencia maior que 0 (todos que aparecem no texto) são inseridos em uma lista ordena de forma crescente de acordo com a frequência.

```
for(unsigned int aux = 0; aux < MAX_ASCII; aux++){
    if(0 != frequencia[aux]){
        K++;
        node = new Node(aux, frequencia[aux]);
        Inserir_na_Lista(node);
    }
}
```

Para montar a árvore os dois primeiros nós da lista (os nós com menor frequência), são retirados da lista, e um novo nó é criado com os dois nós retirados como filho. O novo nó é inserido na lista, mantendo ela ordenada. Este processo é repetido até que na lista tenha apenas um nó.

```
//-------------------------------------------------------------------------|
//Etapa 2.1.Montar a arvore de Huffman                                      |
//-------------------------------------------------------------------------|
while(NULL != primeiro->proximo){
    node = new Node(primeiro->freq + primeiro->proximo->freq);
    node->esquerda = primeiro;
    node->direita = primeiro->proximo;
    primeiro = primeiro->proximo->proximo;
    Inserir_na_Lista(node);
}
```

O arquivo é lido novamente substituindo os os caracteres pelos respectivos códigos. A concatenação desses códigos é inserida em um vetor de booleam.

```
getline(entrada, linha);
do{
    for(unsigned int aux = 0; aux < linha.size(); aux++){
        texto.insert(texto.end(), mapa[linha[aux]].begin(), mapa[linha[aux]].end());
    }
    getline(entrada, linha);
}while(!(entrada.eof()));
```

Em um próximo passo, o vetor de código é lido e convertido em char. Na medida que os caracteres são formados, eles são armazenados no arquivo de saída. Se for necessário, são acrescentados bit extras para completar o último byte.

```
char t;
unsigned int aux;
//Para cada "bit" no arquivo
for(aux = 0; aux <= (texto.size() - 8);){
    //Juntar 8 casas do vetor para formar um byte
    for(int bi = 0; bi < 8; bi++){
        t = (t << 1) + (texto[aux++] ? 1 : 0);
    }
    //Gravar o caracter formado no arquivo de saida
    saida << t;
}

//Se for nessesário, preenche o ultimo byte com bits excedentes
if(aux != texto.size()){
    saida << (t << (texto.size() - aux));
}
```