

**Universidade de Brasília**

**Departamento de Ciência da Computação**



## **Lista de Exercício 5 - OA**

### **Autores:**

Gabriel Bessa 16/0120811

Thiago Veras 16/0146682

### **Disciplina:**

Organização de Arquivos

### **Turma:**

A

### **Professor:**

Oscar Gaidos

Brasília  
20 de Maio de 2018

## Exercícios:

1- Implemente o método de Huffman para compressão de dados.

## Resposta:

A compressão de dados pelo método de Huffman, consiste na compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo.

## Explicação do Código:

```
1
2 #include <iostream> // Use cin and cout functions
3 #include <bitset>
4 #include <map>
5 #include <vector>
6 #include <fstream> // Use open() function
7 #include <queue>
8
9 // Used to omit ::std syntax
10 using namespace std;
```

Cabeçalho do programa, onde estão as bibliotecas usadas para o programa.

```
1
2 // A Huffman tree node
3 struct MinHeapNode {
4
5     // One of the input characters
6     char data;
7
8     // Frequency of the character
9     unsigned freq;
10
11     // Left and right child
12     MinHeapNode *left, *right;
13
14     // Constructor
15     MinHeapNode(char data, unsigned freq, MinHeapNode *
16         left, MinHeapNode *right){
17         this->left = left;
18         this->right = right;
```

```

18     this->data = data;
19     this->freq = freq;
20 }
21
22 // Check if node is leaf
23 bool isLeaf(){
24     return !left and !right;
25 }
26 };

```

Estrutura usada para gerar o nó da árvore de huffman.

```

1
2 // Print trie chars and values
3 void printTrie(struct MinHeapNode* root, string str){
4     if (!root)
5         return;
6
7     // If root is leaf, so have a char
8     if (root->isLeaf()){
9
10        // For in case char are \n, to improve
11        // visualization
12        if(root->data == '\n')
13            cout << "\\n" << " : " << str << "\\n";
14        else
15            cout << root->data << " : " << str << "\\n";
16    }

```

Função que printa os chars e os valores dentro dos nós raízes.

```

1 // Read 8 bits in chars and transform to binary char
2 // value
3 char readChar(int idx, string trieCode){
4
5     // String that will recieve bits in binary
6     string letter;
7
8     // Get 8 binary chars
9     for(int i = idx; i <= idx+8; i++)
10        letter += trieCode[i];

```

```

11 // Transfor to bynary value
12 bitset<8> y(letter);
13
14 // Return respective char binary value
15 return(char)y.to_ulong();
16 }

```

Lê até 8 caracteres e os convertem pra char em binário.

```

1 // Create table of binary
2 void createTable(struct MinHeapNode* root, string str){
3     // Return if root is NULL
4     if (!root)
5         return;
6
7     if (root->data != '\0')
8         table[root->data] = str;
9
10    // Pre order construction
11    createTable(root->left, str + "0");
12    createTable(root->right, str + "1");
13 }

```

Cria a tabela em binário.

```

1 void encodeTrie(struct MinHeapNode* root){
2
3     if(root->isLeaf()){
4         // Add bit 1 if root is leaf
5         trieCode += "1";
6
7         // Get in binary char value
8         bitset<8> y(root->data);
9
10        // Transfor binary value to string
11        trieCode += y.to_string();
12
13        return;
14    }
15
16    // Add bit 0 if root is not leaf
17    trieCode += "0";
18 }

```

```

19 // Pre order call
20 encodeTrie(root->left);
21 encodeTrie(root->right);
22
23 }

```

Essa função adiciona bits '1' quando a raiz é igual a folha. Depois disso ela transforma esse valor binário gerado em char, pra uma string.

```

1 // Create frequency and set of chars used on file
2 void create(vector<char> *carcteres, vector<int> *freq,
3             int *n){
4
5     ifstream file(filename);
6
7     // String to read file
8     string s;
9
10    // Count freq
11    map<char,int> mp;
12
13    // Walk through file and walk through chars
14    mp['\n'] = -1;
15    while(getline(file,s)){
16        for(auto x: s)
17            mp[x]++;
18        mp['\n']++;
19    }
20    mp[4] = 1;
21
22    // Walk through map and set vector of char and freq.
23    for(auto x: mp){
24        carcteres->push_back(x.first);
25        freq->push_back(x.second);
26    }
27
28    // Set carcteres size
29    *n = mp.size();
30
31    // close file
32    file.close();
33 }

```

Cria os caracteres, e a quantidade de caracteres que serão utilizadas em arquivos.

```
1
2 void printMsg(struct MinHeapNode* root, int idx){
3     // Stop recursion if don't have more binary string
4     // to read
5     if(idx > compressedMessage.size())return;
6
7     // If root is leaf then we achieved a char
8     if(root->isLeaf()){
9         if(root->data == 4) return;
10
11         uncompressedMessage += ((int)root->data == 13 ?
12                                 '\n':root->data);
13         printMsg(Trie, idx);
14     }
15     // If is not a root and is 1, so go right
16     else if(compressedMessage[idx] == '1')
17         printMsg(root->right, idx+1);
18
19     // Go left otherwise
20     else
21         printMsg(root->left, idx+1);
22 }
```

Printa a raiz na tela, caso ela seja uma folha.

```
1
2 string binaryString2bits(string total){
3     string ret, aux;
4
5     // Walk through all bits
6     for(int i = 0; i < total.size(); i++){
7
8         // Add bit to string aux
9         aux += total[i];
10
11         // Work only with 8 bits number
12         if( aux.size() != 8) continue;
13
14         // Transform to binary
```

```

15         bitset<8> y(aux);
16
17         // Get respective char according to binary value
18         // and save on compressed file
19         ret += (char)y.to_ulong();
20
21         // Reset numer;
22         aux.clear();
23     }
24
25     return ret;
26 }
27
28 string char2binaryString(string message){
29
30     // string to return
31     string ret;
32
33     for(int i = 0; i < message.size(); i++){
34         // Get binary value from byte value
35         bitset<8> y(message[i]);
36         // Add binary to the final string
37         ret += y.to_string();
38     }
39
40     // Return binary string
41     return ret;
42 }

```

Ambas as funções possuem características iguais, só que com funcionalidades antônimas. Uma transforma a string em binário para bits, e a outra transforma os caracteres para a string em binário. Retornando a string binária no final.

```

1
2 void createCompressedFile(){
3
4     // Open message code
5     ifstream file(filename);
6
7     // Open output file compressed
8     ofstream saida("compressed.txt");
9

```

```

10 // Strings to read file and final result
11 string line, total;
12
13 // Adding Trie binary code to the beginnig o
    compressed file
14
15 // Walktrough file lines and create binary code with
    table
16 while(getline(file,line)){
17     for(auto x: line)
18         total += table[x];
19     total += table['\n'];
20 }
21 total += table[4];
22
23 total = trieCode + total;
24
25 // Total binary form need to be multiple of 8
26 string final = binaryString2bits(total);
27
28
29 // Adding compressed to file
30 saida << final;
31
32 // Close message file
33 file.close();
34
35 // Close compressed file
36 saida.close();
37 }

```

Cria o arquivo que será comprimido.

```

1
2 void compress(){
3
4     // User interaction
5     printf("Input file name to compress : ");
6
7     // Read filename
8     cin >> filename;
9
10    // open message file

```



```

11     ifstream file(filename);
12
13     // Vetors size
14     int n;
15
16     // Chars on file
17     vector<char> carcteres;
18
19     // Frequency of each char
20     vector<int> freq;
21
22     // Create vector with chars and each frequency
23     create(&carcteres,&freq,&n);
24
25     struct MinHeapNode *left, *right;
26     // Compress all data
27     Trie = HuffmanCodes(left, right, Trie, carcteres,
28         freq, n);
29
30     // Create table of values for each char
31     createTable(Trie, "");
32
33     // Create trie binary code
34     encodeTrie(Trie);
35
36     // Print trie chars codes
37     printTrie(Trie, "");
38
39     // Create compressed file
40     createCompressedFile();
41
42     // Closing file
43     file.close();
44 }

```

Comprime o arquivo.

```

1
2 void decompress(){
3
4     // Interaction with user
5     printf("Input file name that want to decompress : ")
6     ;

```

```

6
7 // Read filename
8 cin >> filename;
9
10 // open compressed file
11 ifstream compressed(filename);
12
13 // Get compressed message
14 getline(compressed, compressedMessage);
15
16 // Idx to walk in compressedMessage
17 int idx = 0;
18
19 // Transform bits message in string message
20 compressedMessage = char2binaryString(
    compressedMessage);
21
22 // Build trie with this binary string
23 Trie = buildTrie(compressedMessage, &idx);
24
25 // Print trie
26 printTrie(Trie, "");
27
28 // Print compressed message
29 printMsg(Trie, idx);
30
31 // Open final result file
32 ofstream out("uncompressed.txt");
33
34 // Write message
35 out << uncompressedMessage;
36
37 // Close file;
38 out.close();
39
40 // Close compressed file
41 compressed.close();
42 }

```

Descomprime o arquivo.

```

1
2 int main(){

```

```

3
4 // Interaction with user
5 printf("1 - Compress file\n");
6 printf("2 - Read compressed file\n");
7
8 // Operation variable
9 int op;
10
11 // Read operation
12 cin >> op;
13
14 // Go to huffman compression
15 if(op == 1)
16     compress();
17
18 // Decompress
19 else if(op == 2)
20     decompress();
21
22 return 0;
23 }

```

Principal função do programa, a qual será rodada durante a execução do mesmo.

**Obs:** Alguns trechos de código foram omitidos se não o relatório ia ficar quase que ilegível. Apesar de ter as descrições em cada trecho de código, tentamos ser breves nas explicações das funções como o senhor disse no LINF.

**2-** Descreva e implemente um método de ordenação dentre os seguintes: quick sort, shell sort ou merge sort.

**Resposta:**  
**Explicação do Código:**

```

1
2 #include <bits/stdc++.h> //Including all C++ Libraries
3
4 // Used to omit ::std syntax
5 using namespace std;

```

Começo do código com todas as bibliotecas do c++ para uso de funções.

```

1 // Print menu on terminal
2 void intro(){
3     printf("\n");
4     printf("#####\n");
5     printf("#                               #\n");
6     printf("#                MERGE SORT                #\n");
7     printf("#                               #\n");
8     printf("#####\n");
9     printf("#                               #\n");
10    printf("#        Aluno: Thiago Veras Machado        #\n");
11    printf("#                M: 16/0146682                #\n");
12    printf("# Aluno: Gabriel Cunha Bessa Vieira #\n");
13    printf("#                M: 16/0120811                #\n");
14    printf("#                               #\n");
15    printf("#####\n");
16    printf("\n");
17    printf("How many files you want to read ? : ");
18 }

```

Essa função apenas printa o menu de abertura no terminal

```

1 // Function which inter calls both halves called by
  merge sort
2 void intercalar (int *v,int *aux,int ini1, int ini2,int
  fim2){
3
4     int in1 = ini1, in2 = ini2, fim1 = in2-1, au = 0;
5
6     // Colocando os elementos ordenados no vetor aux
7     while(in1 <= fim1 and in2 <= fim2){
8
9         if (v[in1] < v[in2])
10            aux[au++] = v[in1++];
11        else
12            aux[au++] = v[in2++];
13    }
14    // Colocando os resto dos elementos caso nao tenha
      terminado de pegar todos os elementos do vetor v
15    while(in1 <= fim1)
16        aux[au++] = v[in1++];
17    while(in2 <= fim2)
18        aux[au++] = v[in2++];

```

```

19     for(int i = 0; i < au; i++)
20         v[i + ini1] = aux[i];
21 }

```

Esse trecho de código é da função intercalar, que pega as duas metades que foram separadas pelo mergesort e junta. E também cria um vetor auxiliar caso o resultado tenha sido encontrado, mas ainda tenha elementos sobrando para jogar no principal.

```

1 // Funcao de ordenacao do vetor
2 void mergeSort (int *v, int *aux, int esq, int dir){
3
4     // Se a posicao onde o vetor comeca eh maior ou igual
5     // onde ele termina, eh porque nao precisa ordenar
6     if(esq >= dir) return;
7
8     // Chama recursivamente o algoritmo ordenando do
9     // comeco ate a metade
10    mergeSort(v,aux,esq,(esq+dir)/2);
11
12    // Chama recursivamente da metade + 1 ate o final
13    mergeSort(v,aux,(esq+dir)/2 + 1,dir);
14
15    // Intercala tudo
16    intercalar(v,aux,esq,(esq+dir)/2 + 1,dir);
17 }

```

A função mergesort ordena o vetor de acordo com o conceito do algoritmo passado em sala de aula. Essa função avalia todos os casos possíveis do mergesort. Nos casos normais de ordenação, no pior caso, onde tem que ordenar todo o vetor, e no caso que o vetor está ordenado, se for o caso, ela já retorna.

```

1 // Imprime os valores do vetor
2 void print_array(int *v, int n){
3
4     // Fazendo loop e andando pelos elementos do vetor
5     for(int i = 0; i < n; i++)
6         printf("%d ", v[i]);
7
8     // Quebra de linha
9     puts("");
10
11 }

```

12 }

Essa função apenas percorre o vetor, imprimindo os seus respectivos valores.

```
1  int main()
2  {
3      // Variavel para pegar o tamanho do vetor
4      int n;
5
6      // Iteracao com o usuario
7      printf("Digite o tamanho do vetor :");
8
9      // Lendo o tamanho do vetor
10     cin >> n;
11
12     // Criando o vetor de tamanho n e um auxiliar para
        poder ordenar
13     int array[n], aux[n];
14
15     // Lendo n numeros e salvando no vetor
16     for (int i = 0; i < n; i++)
17         cin >> array[i];
18
19     // Chamando o algoritmo de ordenacao merge sort
20     mergeSort(array, aux, 0, n - 1);
21
22     // Imprimindo o vetor
23     print_array(array, n);
24
25     return 0;
26 }
```

Essa é a função principal do programa, a qual vai executar corretamente o mergesort