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 *
          left, MinHeapNode *right){
16         this->left = left;
17         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
                visualization
11          if(root->data == '\n')
12              cout << "\\n" << " : " << str << "\n";
13          else
14              cout << root->data << " : " << str << "\n";
15
16      }
```

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

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

```
    // Transfor to bynary value
    bitset<8> y(letter);

    // Return respective char binary value
    return(char)y.to_ulong();
}
```

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

```
// Create table of binary
void createTable(struct MinHeapNode* root, string str){
    // Return if root is NULL
    if (!root)
        return;

    if (root->data != '\0')
        table[root->data] = str;

    // Pre order construction
    createTable(root->left, str + "0");
    createTable(root->right, str + "1");
}
```

Cria a tabéla em binário.

```
void encodeTrie(struct MinHeapNode* root){

    if(root->isLeaf()){
        // Add bit 1 if root is leaf
        trieCode += "1";

        // Get in binary char value
        bitset<8> y(root->data);

        // Transfor binary value to string
        trieCode += y.to_string();

        return;
    }

    // Add bit 0 if root is not leaf
    trieCode += "0";

```

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

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

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

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

```
    ifstream file(filename);

    // Vetors size
    int n;

    // Chars on file
    vector<char> carcteres;

    // Frequency of each char
    vector<int> freq;

    // Create vector with chars and each frequency
    create(&carcteres,&freq,&n);

    struct MinHeapNode *left, *right;
    // Compress all data
    Trie = HuffmanCodes(left, right, Trie, carcteres,
       freq, n);

    // Create table of values for each char
    createTable(Trie,"");

    // Create trie binary code
    encodeTrie(Trie);

    // Print trie chars codes
    printTrie(Trie, "");

    // Create compressed file
    createCompressedFile();

    // Closing file
    file.close();
}
```

Comprime o arquivo.

```

void decompress(){

  // Interaction with user
    printf("Input file name that want to decompress : ")
       ;
```

```
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.

compilation flags: Para compilar esse código, usamos no terminal o seguinte para gerar o executável 'compresser':
huff.cpp: g++ -o compresser huff.cpp -std=c++11

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
       onde ele termina, eh porque nao precisa ordenar
5    if(esq >= dir) return;
6
7    // Chama recursivamente o algoritmo ordenando do
       comeco ate a metade
8    mergeSort(v,aux,esq,(esq+dir)/2);
9
10   // Chama recursivamente da metade + 1 ate o final
11   mergeSort(v,aux,(esq+dir)/2 + 1,dir);
12
13   // Intercala tudo
14   intercalar(v,aux,esq,(esq+dir)/2 + 1,dir);
15 }
```

A função mergesort ordena o vetor de acordo com o conceito do algorítmo passado em sala de aula. Essa função avalia todos o 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
2  // Imprime os valores do vetor
3  void print_array(int *v, int n){
4
5    // Fazendo loop e andando pelos elementos do vetor
6    for(int i = 0; i < n; i++)
7      printf("%d ",v[i]);
8
```

```
    // Quebra de linha
    puts("");

}
```

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

```
int main()
{
    // Variavel para pegar o tamanho do vetor
    int n;

    // Iteracao com o usuario
    printf("Digite o tamanho do vetor :");

    // Lendo o tamanho do vetor
    cin >> n;

    // Criando o vetor de tamanho n e um auxiliar para
       poder ordenar
    int array[n], aux[n];

    // Lendo n numeros e salvando no vetor
    for (int i = 0; i < n; i++)
        cin >> array[i];

    // Chamando o algoritmo de ordenacao merge sort
    mergeSort(array, aux, 0, n - 1);

    // Imprimindo o vetor
    print_array(array, n);

    return 0;
}
```

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

compilation flags: Para compilar esse código, usamos no terminal o seguinte para gerar o executável 'mergesort':

mergesort.cpp: g++ -o mergesort mergesort.cpp