|  |  |  |
| --- | --- | --- |
| **软件学院实验报告** | | |
| 姓名：胡逢彬 学号：2225060803 专业：网络工程 年级：2022级 | | |
| 课程名称 | | 数据结构 |
| 实验名称 | | 实验8、哈夫曼树的构建和编码 |
| 实验的准备阶段 | 实验内容 | （1）实验目的  通过该实验，使学生理解哈夫曼树的概念，掌握哈夫曼树及哈夫曼编码的构造过程，体会网络发送端和接收端编码和译码过程及其工作原理。  （2）实验内容  给定报文中26个字母a-z及空格的出现频率{64, 13, 22, 32, 103, 21, 15, 47, 57, 1, 5, 32, 20, 57, 63, 15, 1, 48, 51, 80, 23, 8, 18, 1, 16, 1, 168}，构建哈夫曼树并为这27个字符编制哈夫曼编码，并输出。模拟发送端，从键盘输入字符串，以%为结束标记，在屏幕上输出输入串的编码；模拟接收端，从键盘上输入0-1哈夫曼编码串，翻译出对应的原文。  （3）参考界面    （4）验收/测试用例   * 模拟发送端   输入：I love you  输出：01101111011110011100000010111100011100100001   * 模拟接收端 输入   输入：01101101111011000111111010111101101001100001  输出：it is a dog |
| 实验类型 | 验证性 |
| 实验的重点、难点 | 重点： 哈夫曼树的创建和编码  难点：哈夫曼树的创建和编码 |
| 实验环境 | VC++6.0 |
| 实验的实施阶段 | 实验步骤及完成任务情况 | 一、设计思想  设计思想：  1. 模块化思想：将整个实验分解为多个模块，如输入HuffmanTree参数、初始化HuffmanTree参数、创建HuffmanTree和编码表、输出编码表、输入编码并翻译为字符、输入字符并实现转码等模块，每个模块负责特定的功能，使它们相互独立、易于理解和维护。  2. 哈夫曼树构建思想：根据给定的26个字母a-z及空格的出现频率，构建哈夫曼树。哈夫曼树的构建过程是通过不断合并权重最小的节点来构建，直至形成根节点为止。这样可以保证权重较小的字符对应的编码较短，权重较大的字符对应的编码较长。  3. 哈夫曼编码生成思想：在哈夫曼树构建完成后，根据树的结构，生成每个字符的哈夫曼编码。通过递归地遍历哈夫曼树的路径，从根节点到叶子节点的路径上的0和1分别表示左子节点和右子节点，从而生成对应的编码表。编码表可以在编码和译码过程中进行查找和使用。  4. 模拟发送端和接收端思想：模拟发送端，从键盘输入字符串，并对字符串进行编码。编码过程是根据输入的字符串，查找对应的字符的哈夫曼编码，并输出编码结果。模拟接收端，从键盘输入0-1哈夫曼编码串，根据编码表进行查找，并翻译出对应的原文。  5. 用户界面交互思想：为了方便用户操作和理解，设计一个用户界面，在界面上显示菜单选项，用户可以选择不同的功能，如输入HuffmanTree参数、初始化HuffmanTree参数、输出编码表等。根据用户的选择，执行相应的功能并在界面上显示结果。  通过以上设计思想，可以将实验分解为多个模块，实现哈夫曼树的构建、哈夫曼编码的生成、编码和译码的操作，并提供友好的用户界面交互，使学生能够更好地理解哈夫曼树和哈夫曼编码的概念，以及网络发送端和接收端编码和译码的工作原理。   1. 主要源代码   #include <iostream>  #include <string>  #include <queue>  #include <map>  using namespace std;  // 哈夫曼树节点的结构定义  struct HuffmanNode {  char data; // 字符  int weight; // 权重  HuffmanNode\* left; // 左子节点  HuffmanNode\* right; // 右子节点  HuffmanNode(char c, int w) : data(c), weight(w), left(NULL), right(NULL) {}  };  // 用于比较HuffmanNode指针的大小（权重小的在前）  struct compare {  bool operator()(HuffmanNode\* a, HuffmanNode\* b) {  return a->weight > b->weight;  }  };  // 构建Huffman树  HuffmanNode\* buildHuffmanTree(map<char, int>& freq) {  priority\_queue<HuffmanNode\*, vector<HuffmanNode\*>, compare> pq;  for (auto& pair : freq) {  pq.push(new HuffmanNode(pair.first, pair.second));  }  while (pq.size() > 1) {  HuffmanNode\* left = pq.top(); pq.pop();  HuffmanNode\* right = pq.top(); pq.pop();  HuffmanNode\* parent = new HuffmanNode('-', left->weight + right->weight);  parent->left = left;  parent->right = right;  pq.push(parent);  }  return pq.top();  }  // 生成Huffman编码  void generateHuffmanCodes(HuffmanNode\* root, string code, map<char, string>& huffmanCodes) {  if (!root) {  return;  }  if (root->data != '-') {  huffmanCodes[root->data] = code;  }  generateHuffmanCodes(root->left, code + "0", huffmanCodes);  generateHuffmanCodes(root->right, code + "1", huffmanCodes);  }  // 输出Huffman编码表  void printHuffmanCodes(map<char, string>& huffmanCodes) {  cout << "Huffman编码表：" << endl;  for (auto& pair : huffmanCodes) {  cout << pair.first << " : " << pair.second << endl;  }  }  // 编码  string encode(map<char, string>& huffmanCodes, string input) {  string encoded;  for (char c : input) {  encoded += huffmanCodes[c];  }  return encoded;  }  // 译码  string decode(HuffmanNode\* root, string encoded) {  string decoded;  HuffmanNode\* curr = root;  for (char c : encoded) {  if (c == '0') {  curr = curr->left;  } else {  curr = curr->right;  }  if (curr->data != '-') {  decoded += curr->data;  curr = root;  }  }  return decoded;  }  int main() {  map<char, int> freq = {  {'a', 64}, {'b', 13}, {'c', 22}, {'d', 32}, {'e', 103}, {'f', 21}, {'g', 15},  {'h', 47}, {'i', 57}, {'j', 1}, {'k', 5}, {'l', 32}, {'m', 20}, {'n', 57},  {'o', 63}, {'p', 15}, {'q', 1}, {'r', 48}, {'s', 51}, {'t', 80}, {'u', 23},  {'v', 8}, {'w', 18}, {'x', 1}, {'y', 16}, {'z', 1}, {' ', 168}  };  HuffmanNode\* root = buildHuffmanTree(freq);  map<char, string> huffmanCodes;  generateHuffmanCodes(root, "", huffmanCodes);  printHuffmanCodes(huffmanCodes);  int choice;  while (true) {  cout << endl;  cout << "请选择功能：" << endl;  cout << "1. 输入HuffmanTree的参数" << endl;  cout << "2. 初始化HuffmanTree参数" << endl;  cout << "3. 创建HuffmanTree和编码表" << endl;  cout << "4. 输出编码表" << endl;  cout << "5. 输入编码，并翻译为字符" << endl;  cout << "6. 输入字符，并实现转码" << endl;  cout << "7. 退出" << endl;  cout << "请输入选择的功能编号：";  cin >> choice;  switch (choice) {  case 1:  break;  case 2:  break;  case 3:  break;  case 4:  printHuffmanCodes(huffmanCodes);  break;  case 5: {  string input;  cout << "请输入要编码的字符串（以%结束）：";  // cin >> input;  string temp;  getline(cin,temp);  getline(cin,input);  input += '%';  string encoded = encode(huffmanCodes, input);  cout << "编码结果为：" << encoded << endl;  break;  }  case 6: {  string encoded;  cout << "请输入要译码的编码串：";  //cin >> encoded;  string temp;  getline(cin,temp);  getline(cin,encoded);  string decoded = decode(root, encoded);  cout << "译码结果为：" << decoded << endl;  break;  }  case 7:  return 0;  default:  cout << "输入无效！请重新输入。" << endl;  break;  }  }  } |
| 实验结果的处理阶段 | 实验结果 | IMG_256 |
| 实验结果总结 |  |