

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 数据结构 | | | | |
| 实验编号： | 实验八 | | | | |
| 实验名称： | 哈夫曼树与哈夫曼编码 | | | | |
| 实验人员： | 学号 | 18111303044 | | | |
| 姓名 | 邵一波 | | | |
| 班级 | 2018级计算机类1班 | | | |
| 实验日期： | 2019-11-28 | | | | |
| 实验室： | 2号实验楼202室 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

|  |
| --- |
| **注：具体内容可根据专业特点和实验性质略作调整，页面不够可附页。**   1. **需求分析** 2. 描述：熟练掌握 huffman 树的构建方法以及 huffman 编码。 3. 输入的形式和输入值的范围   从键盘输入一个字符串（字符范围 a – z）     1. 输出的形式   所输入字符的权值    哈夫曼表    哈弗曼编码     1. 程序所能达到的功能 2. 从键盘接收任意一个字符串，以字符串中某字符出现的次数，作为该字符的 权值。利用得到的权值构造 huffman 树、并输出每个字符对应的 huffman 编码。 3. 测试数据   asfagfagaawrfgsryhws  aaabbbcccdddeffghjjkkl  eerrdfsfgshgshsafr   1. **概要设计** 2. 抽象数据类型的定义   **CommonDef.h**  #include<string.h>  #include<stdlib.h>  #include<stdio.h>  int m,s1,s2;  typedef struct  {  int weight;  int parent,lchild,rchild;  }HTNode,\*HuffmanTree; // 动态分配数组存储哈夫曼树  typedef char \*HuffmanCode; // 动态分配数组存储哈夫曼编码表   1. 函数的定义   **huffman.h**  #include "CommonDef.h"  void Select(HuffmanTree HT,int n);  void HuffmanCoding(HuffmanTree HT, HuffmanCode HC[], int \*w, int n);  int \* Input(int \* num);  void printCode(int n,HuffmanCode HC[]);   1. 主程序的流程图      1. **详细设计**   **HuffmanTestApp.c**  #include "huffman.h"  int main()  {  HuffmanTree HT = NULL;  HuffmanCode \*HC;  int n = 0;  int \*w;  HC = (HuffmanCode \*)malloc(n\*sizeof(HuffmanCode));  w = Input(&n);  HuffmanCoding(HT,HC,w,n);  printCode(n,HC);  return 0;  }  **Huffman.c**  #include "Huffman.h"  void Select(HuffmanTree HT,int n)  {  int i,j;  for(i = 1;i <= n;i++)  if(!HT[i].parent)  {  s1 = i;  break;  }  for(j = i+1;j <= n;j++)  if(!HT[j].parent)  {  s2 = j;  break;  }  for(i = 1;i <= n;i++)  if((HT[s1].weight>HT[i].weight)&&(!HT[i].parent)&&(s2!=i))  s1=i;  for(j = 1;j <= n;j++)  if((HT[s2].weight>HT[j].weight)&&(!HT[j].parent)&&(s1!=j))  s2=j;  }  void HuffmanCoding(HuffmanTree HT, HuffmanCode HC[], int \*w, int n)  {  int i, j;  char \*cd;  int p;  int cdlen;  if (n<=1)  return;  m = 2 \* n - 1;  HT = (HuffmanTree)malloc((m+1) \* sizeof(HTNode)); //0号单元未用  for(i=1; i<=n; i++)  {  HT[i].weight=w[i-1];  HT[i].parent=0;  HT[i].lchild=0;  HT[i].rchild=0;  }  for (i=n+1; i<=m; i++)  {  HT[i].weight=0;  HT[i].parent=0;  HT[i].lchild=0;  HT[i].rchild=0;  }  for (i=n+1; i<=m; i++)  {  // 建哈夫曼树  // 在HT[1..i-1] 中选择parent 为0 且weight 最小的两个结点，  // 其序号分别为s1 和s2。  Select(HT, i-1);  HT[s1].parent = i; HT[s2].parent = i;  HT[i].lchild = s1; HT[i].rchild = s2;  HT[i].weight = HT[s1].weight + HT[s2].weight;  }  printf("Huffman表如下\n");  for(j=1; j<=m; j++)  printf("<%2d>%3d%3d%3d%3d\n",j,HT[j].weight,  HT[j].parent,HT[j].lchild, HT[j].rchild);  //------ 无栈非递归遍历哈夫曼树，求哈夫曼编码  cd = (char \*)malloc(n\*sizeof(char)); // 分配求编码的工作空间  p = m; cdlen = 0;  for(i=1; i<=m; ++i)//遍历哈夫曼树时用作结点状态标志  HT[i].weight = 0;  while (p)  {  if(HT[p].weight==0)  {//向左  HT[p].weight = 1;  if(HT[p].lchild != 0)  {  p = HT[p].lchild;  cd[cdlen++] ='0';  }  else if(HT[p].rchild == 0)  {//登记叶子结点的字符的编码  HC[p] = (char \*)malloc((cdlen+1) \* sizeof(char));  cd[cdlen] ='\0'; strcpy(HC[p], cd); // 复制编码(串)  }  }  else if (HT[p].weight==1)  { // 向右  HT[p].weight = 2;  if (HT[p].rchild != 0) { p = HT[p].rchild; cd[cdlen++] ='1'; }  }  else  {  HT[p].weight = 0;  p = HT[p].parent;  --cdlen;  }  }  }  void printCode(int n,HuffmanCode HC[])  {  int i;  puts("对应的哈夫曼编码如下:");  for(i = 1;i <= n;i++)  printf("第%2d个字符对应的Huffman编码%s\n",i,HC[i]);  }  **Input.c**  #include "huffman.h"  int \* Input(int \* num)  {  int \*w;  char a[100];  int result[26] = {0};  int i,j;  \*num = 0;  puts("输入要编码的字符串");  scanf("%s",a);  for(i=0;i<strlen(a);i++)  result[a[i]-'a']++;  puts("字符对应的权值如下");  for(i=0;i<26;i++)  {  if(result[i]>0)  {  printf("%c %d\n",'a'+i,result[i]);  (\*num) ++;  }  }  w = (int \*)malloc((\*num)\*sizeof(int));  for(i=0,j=0;i<26;i++)  {  if(result[i]>0)  {  \*(w+j) = result[i];  j++;  }  }  return w;  }   1. **调试分析** 2. 调试过程中遇到的问题是如何解决的以及对设计与实现的回顾讨论和分析   运算结果出错，经检查是函数参数问题   1. 算法的时空分析，改进设想   尝试使用链式储存结构重新完成实验   1. **测试数据与结果**   **（1）**    **（2）**    **（3）** |