

**《数据结构与算法实践》实践报告**

|  |
| --- |
| 学 院： 虚拟现实学院  年级 专业 班级： 18级 软件工程 APP开发技术1班  课程（环节）名称： 《数据结构与算法实践》 |
| 学 生 学 号： 11827041  学 生 姓 名： 叶欣宇  指 导 教 师： 张亮 |

2019年6月24日

|  |  |
| --- | --- |
| **实践题目** | 哈夫曼应用实践 |
| **实践时间** | 2019.6.24-2019.6.28 |
| **一、实践目的**  1.熟练应用哈夫曼编码方法。  2.熟练掌握哈夫曼编码、译码思想。  3.能够基于哈夫曼编码、译码思想形式下，设计合理算法解决最优树应用问题。 | |
| **二、实践内容**  基于哈夫曼编码、译码思想形式下，设计合理算法，根据统计字符串中每种字符串出现频率，利用哈夫曼编码原理，对每个字符进行（0,1）编码，并输出每种字符编码。  。 | |
| **三、实践步骤**  （一）算法的伪代码  1.选取出两个权值最小的结点；  1.1初始记录最小权值min为0；  1.2遍历全部结点，找出单节点，如果此结点的父亲没有，那么把结点号赋值给 min，跳出循环；  1.3继续遍历全部结点，找出权值最小的单节点，如果此结点的权值比 min 结点的权值小，那么更新 min 结点，否则就是最开始的 min；  1.4 s1指针指向权值最小的叶子结点；  1.5重复1.3的步骤找到第二个min；  1.6 s2指针指向第二个权值最小的叶子结点；   1. 创建哈夫曼树；   2.1初始化叶子结点，结构数组来初始化每个叶子结点，初始的时候看做一个个单个结点的二叉树；  2.2初始化非叶子结点；  2.3在(\*huffmanTree)[1]~(\*huffmanTree)[i-1]的范围内选择两个parent为0且weight最小的结点，其序号分别赋值给s1、s2，选出的两个权值最小的叶子结点，组成一个新的二叉树，根为 i 结点；   1. 求每个叶子结点对应的哈夫曼编码   3.1初始化编码起始指针  3.2从叶子到根结点求编码，从右到左的顺序编码入数组内，左分支标0，右分支标1  3.3打印编码序列   1. 算法的C语言描述   void elect(HuffmanTree \*huffmanTree, int n, int \*s1, int \*s2){  int i = 0;  int min = 0;  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0){  min = i;  break;  }  }  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0){  if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){  min = i;  }  }  }  \*s1 = min;  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0 && i != (\*s1)){  min = i;  break;  }  }  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0 && i != (\*s1)){  if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){  min = i;  }  }  }  \*s2 = min;  }  void createHuffmanTree(HuffmanTree \*huffmanTree, int w[], int n){  int m = 2 \* n - 1;  int s1,s2,i;  \*huffmanTree = (HuffmanTree)malloc((m + 1) \* sizeof(Node));  for(i = 1; i <= n; i++){  (\*huffmanTree)[i].weight = w[i];  (\*huffmanTree)[i].lChild = 0;  (\*huffmanTree)[i].parent = 0;  (\*huffmanTree)[i].rChild = 0;  }  for(i = n + 1; i <= m; i++){  (\*huffmanTree)[i].weight = 0;  (\*huffmanTree)[i].lChild = 0;  (\*huffmanTree)[i].parent = 0;  (\*huffmanTree)[i].rChild = 0;  }  printf("HuffmanTree:\n");  for(i = n + 1; i <= m; i++){  elect(huffmanTree, i-1, &s1, &s2);  (\*huffmanTree)[s1].parent = i;  (\*huffmanTree)[s2].parent = i;  (\*huffmanTree)[i].lChild = s1;  (\*huffmanTree)[i].rChild = s2;  (\*huffmanTree)[i].weight=(\*huffmanTree)[s1].weight+(\*huffmanTree)[s2].weight;  printf("%d (%d, %d)\n", (\*huffmanTree)[i].weight, (\*huffmanTree)[s1].weight, (\*huffmanTree)[s2].weight);  }  printf("\n");  }  void createHuffmanCode(HuffmanTree \*huffmanTree, HuffmanCode \*huffmanCode, int n){  int i,start,p;  unsigned int c;  huffmanCode=(HuffmanCode \*)malloc((n+1) \* sizeof(char \*));  char \*cd = (char \*)malloc(n \* sizeof(char));  cd[n-1] = '\0';  for(i = 1; i <= n; i++){  start = n - 1;  for(c=i, p=(\*huffmanTree)[i].parent;p!=0;c=p,p=(\*huffmanTree)[p].parent){  if( (\*huffmanTree)[p].lChild == c){  cd[--start] = '0';  }  else{  cd[--start] = '1';  }  }  huffmanCode[i] = (char \*)malloc((n - start) \* sizeof(char));  strcpy(huffmanCode[i], &cd[start]);  }  free(cd);  for(i = 1; i <= n; i++){  printf("%c %d 的哈夫曼编码是 %s\n", C[i],(\*huffmanTree)[i].weight, huffmanCode[i]);  }  printf("\n");  }   1. 时间复杂度分析   选权值最小的节点：问题规模为n，基本语句为if((\*huffmanTree)[i].parent == 0){if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){min = i;}}以及if((\*huffmanTree)[i].parent == 0 && i != (\*s1)){if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){min = i;}}，时间复杂度为n；  创建哈夫曼树：问题规模为n-1～m，即n-1～2n\*-1，即n，基本语句为elect(huffmanTree, i-1, &s1, &s2);在for循环中调用了另一个函数，时间复杂度为n2；  求哈夫曼编码：问题规模为n，基本语句为if( (\*huffmanTree)[p].lChild == c){cd[--start] = '0';}else{cd[--start] = '1';}，时间复杂度为n2   1. 编译   编译通过。  （五）测试  **WeChata073367bfdf9336d6d1b37a3827394e7 WeChatee1c03be9dc0af435d0b4d21f4795aa9** | |
| **四、实践项目成果**  #include <stdio.h>  #include <stdlib.h>  #define maxSize 100  typedef struct{  unsigned int weight;  unsigned int parent;  unsigned int lChild;  unsigned int rChild;  } Node, \*HuffmanTree;  char C[100];  typedef char \*HuffmanCode;  //选取两个权值最小的结点s1,s2  void elect(HuffmanTree \*huffmanTree, int n, int \*s1, int \*s2){  int i = 0;  int min = 0;  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0){  min = i;  break;  }  }  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0){  if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){  min = i;  }  }  }  \*s1 = min;  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0 && i != (\*s1)){  min = i;  break;  }  }  for(i = 1; i <= n; i++){  if((\*huffmanTree)[i].parent == 0 && i != (\*s1)){  if((\*huffmanTree)[i].weight < (\*huffmanTree)[min].weight){  min = i;  }  }  }  \*s2 = min;  }  //创建哈夫曼树  void createHuffmanTree(HuffmanTree \*huffmanTree, int w[], int n){  int m = 2 \* n - 1;  int s1,s2,i;  \*huffmanTree = (HuffmanTree)malloc((m + 1) \* sizeof(Node));  for(i = 1; i <= n; i++){  (\*huffmanTree)[i].weight = w[i];  (\*huffmanTree)[i].lChild = 0;  (\*huffmanTree)[i].parent = 0;  (\*huffmanTree)[i].rChild = 0;  }  for(i = n + 1; i <= m; i++){  (\*huffmanTree)[i].weight = 0;  (\*huffmanTree)[i].lChild = 0;  (\*huffmanTree)[i].parent = 0;  (\*huffmanTree)[i].rChild = 0;  }  printf("HuffmanTree:\n");  for(i = n + 1; i <= m; i++){  elect(huffmanTree, i-1, &s1, &s2);  (\*huffmanTree)[s1].parent = i;  (\*huffmanTree)[s2].parent = i;  (\*huffmanTree)[i].lChild = s1;  (\*huffmanTree)[i].rChild = s2;  (\*huffmanTree)[i].weight=(\*huffmanTree)[s1].weight+(\*huffmanTree)[s2].weight;  printf("%d (%d, %d)\n", (\*huffmanTree)[i].weight, (\*huffmanTree)[s1].weight, (\*huffmanTree)[s2].weight);  }  printf("\n");  }  //逆向求每个叶子结点对应的哈夫曼编码  void createHuffmanCode(HuffmanTree \*huffmanTree, HuffmanCode \*huffmanCode, int n){  int i,start,p;  unsigned int c;  huffmanCode=(HuffmanCode \*)malloc((n+1) \* sizeof(char \*));  char \*cd = (char \*)malloc(n \* sizeof(char));  cd[n-1] = '\0';  for(i = 1; i <= n; i++){  start = n - 1;  for(c=i, p=(\*huffmanTree)[i].parent;p!=0;c=p,p=(\*huffmanTree)[p].parent){  if( (\*huffmanTree)[p].lChild == c){  cd[--start] = '0';  }  else{  cd[--start] = '1';  }  }  huffmanCode[i] = (char \*)malloc((n - start) \* sizeof(char));  strcpy(huffmanCode[i], &cd[start]);  }  free(cd);  for(i = 1; i <= n; i++){  printf("%c %d 的哈夫曼编码是 %s\n", C[i],(\*huffmanTree)[i].weight, huffmanCode[i]);  }  printf("\n");  }  int main(int argc, char \*argv[]) {  HuffmanTree HT;  HuffmanCode HC;  int \*w,i,n,weight;  char string[maxSize];  printf("任意输入字符串：" );  scanf("%s",string);  printf("n = " );  scanf("%d",&n);  for (i=1; i<=n; i++) {  C[i]=string[i-1];  }  w=(int \*)malloc((n+1)\*sizeof(int));  printf("输入这%d个元素的权值:\n",n);  for(i=1; i<=n; i++){  printf("%c: ",string[i-1]);  fflush(stdin);  scanf("%d",&weight);  w[i]=weight;  }  createHuffmanTree(&HT, w, n);  createHuffmanCode(&HT,&HC,n);  return 0;  } | |
| **五、教师评语**  **报告成绩： 教师签字：** | |