哈夫曼树 (Huffman Tree)

张晓平

2016 年 11 月 7 日

1 定义

哈夫曼树是一种带权路径长度最短的二叉树，也称最优二叉树。

2 构造

构造过程如下：

1. 将所有左、右子树都为空的结点作为根结点；
2. 选出两颗根结点的权值最小的树作为一棵新树的左、右子树，且置新树的根结点的权值为其左、右子树上根结点的权值之和。注：左子树的权值应小于右子树的权值。
3. 从森林中删除这两棵树，同时把新树加入到森林中。
4. 重复 2、3，直到森林中只有一棵树为止，此树便是 Huffman 树。

3 Huffman 编码

利用 Huffman 树求得的用于通信的二进制编码称为 Huffman 编码。树中从根到每个叶子结点都有一条路径，对路径上各分支作如下约定：指向左子树的分支表示 0 码，指向右子树的分支表示 1 码。取每条路径上的 0 或 1 的序列作为各叶子结点对应的字符编码，即是 Huffman 编码。

4 程序实现

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
typedef char ElemType;

/* Huffman tree’s node */
typedef struct HuffNode {
  ElemType data;
  struct HuffNode * rchild;
  struct HuffNode * lchild;
  int weight;
  ElemType code[20];
} HuffNode, * HuffTree;

/* Queue */
typedef struct QueueNode {
  HuffNode * data;
  struct QueueNode * next;
} QueueNode;

typedef struct {
  QueueNode * front;
  QueueNode * rear;
} Queue;

Queue * Create_Empty_Queue();
int EnterQueue(Queue * head, HuffNode * data);
HuffNode * DeleteQueue(Queue * head);
int Is_Empty_Queue(Queue * head);

int Is_Empty_OrderQueue(Queue * head);
int EnterOrderQueue(Queue * head, HuffNode * p);

HuffNode * Create_Huffman_Tree(Queue * head);
int HuffmanCode(HuffNode * root);

HuffNode * MakeNode(ElemType item, HuffNode * lchild, HuffNode * rchild, int weight);
```

```
int GetDepth(HuffTree tree);
```

```
#include ”Huffman.h”

Queue * Create_Empty_Queue()
{
  QueueNode * QNode;
  Queue * HQueue;

  QNode = (QueueNode *) malloc(sizeof(QueueNode));
  QNode->next = NULL;

  HQueue = (Queue *) malloc(sizeof(Queue));
  HQueue->front = HQueue->rear = QNode;

  return HQueue;
}

int EnterQueue(Queue * head, HuffNode * data)
{
  QueueNode * temp;

  temp = (QueueNode *) malloc(sizeof(QueueNode));
  temp->data = data;
  temp->next = NULL;

  head->rear->next = temp;
  head->rear = temp;

  return 0;
}

HuffNode * DeleteQueue(Queue * head)
{
  QueueNode * temp;
  temp = head->front;
  head->front = temp->next;
  free(temp);
  temp = NULL;
  return head->front->data;
}

int Is_Empty_Queue(Queue * head)
{
  if(head->front == head->rear)
    return 1;
  else
    return 0;
}

int EnterOrderQueue(Queue * head, HuffNode * p)
{
  QueueNode * m = head->front->next;
  QueueNode * n = head->front;
  QueueNode * temp;

  while(m) {
    if (m->data->weight < p->weight) {
      m = m->next;
      n = n->next;
    } else
      break;
  }

  if(m == NULL){
    temp = (QueueNode *) malloc(sizeof(QueueNode));
    temp->data = p;
    temp->next = NULL;

    n->next = temp;
    head->rear = temp;
    return 0;
  }

  temp = (QueueNode *) malloc(sizeof(QueueNode));
  temp->data = p;
  n->next = temp;
  temp->next = m;
  return 0;
}

int Is_Empty_OrderQueue(Queue * head)
{
  if(head->front->next->next == NULL)
    return 1;
  return 0;
}


HuffNode * Create_Huffman_Tree(Queue *head)
{
  HuffNode * right, * left, * current;

  while (!Is_Empty_OrderQueue(head)) {
    left  = DeleteQueue(head);
    right = DeleteQueue(head);
    current = (HuffNode *) malloc(sizeof(HuffNode));
    current->weight = left->weight + right->weight;
    current->rchild = right;
    current->lchild = left;
    EnterOrderQueue(head, current);
  }

  return head->front->next->data;
}
```

```
//Huffman Code
int HuffmanCode(HuffNode * root)
{
  HuffNode * current = NULL;
  Queue * queue = Create_Empty_Queue();
  EnterQueue(queue, root);

  while(!Is_Empty_Queue(queue)){
    current = DeleteQueue(queue);

    if(current->rchild == NULL && current->lchild == NULL)
      printf(”%c:%d␣%s\n”, current->data, current->weight, current->code);

    if(current->lchild){
      strcpy(current->lchild->code, current->code);
      strcat(current->lchild->code, ”0”);
      EnterQueue(queue, current->lchild);
    }

    if(current->rchild){
      strcpy(current->rchild->code, current->code);
      strcat(current->rchild->code, ”1”);
      EnterQueue(queue, current->rchild);
    }
  }
  return 0;
}

/* Generate a node */
HuffNode * MakeNode(ElemType item, HuffNode * lchild, HuffNode * rchild, int weight)
{
  HuffNode * pnode = (HuffNode *) malloc(sizeof(HuffNode));
  if (pnode){
    pnode->data = item;
    pnode->lchild = lchild;
    pnode->rchild = rchild;
    pnode->weight = weight;
    /* pnode->code = code; */
  }
  return pnode;
}

/* Return a BiTree’s depth */
int GetDepth(HuffTree tree)
{
  int cd, ld, rd;
  cd = ld = rd = 0;
  if(tree) {
    ld = GetDepth(tree->lchild);
    rd = GetDepth(tree->rchild);
    cd = (ld > rd ? ld : rd);
    return cd+1;
  }else
    return 0;
}
```

```
// input characters a-g
#include ”Huffman.h”
int main(void){
  Queue * head;
  HuffNode * root;
  HuffNode * node[100];
  ElemType ch, cc[100];
  int weight[100] = {0};
  int i, k = 0;
  printf(”input␣character:\n”);
  while(1) {
    scanf(”%c”, &ch);
    if(ch == ’\n’){
      break;
    }
    else {
      cc[k++] = ch;
    }
  }

  for(i = 0; i < k; i++)
      weight[cc[i]-’a’]++;

  k = 0;
  for(i = 0; i < 7; i++){
    if(weight[i] > 0) {
      node[k++] = MakeNode(’a’+i, NULL, NULL, weight[i]);
    }
  }

  head = Create_Empty_Queue();
  for(i = 0; i < k; i++)
    EnterOrderQueue(head, node[i]);

  root = Create_Huffman_Tree(head);
  printf(”\nDepth␣of␣Huffman␣Tree␣is␣%d\n”, GetDepth(root));

  printf(”\nHuffman␣Codes␣are:\n”);
  HuffmanCode(root);
}
```

运行结果

```
input character:
aaaabbbcccddddddeeeeeffffffggggggg

Depth of Huffman Tree is 5

Huffman Codes are:
d:6 00
```

```
g:7 01
a:4 100
e:5 101
f:6 111
c:3 1100
b:3 1101
```