哈夫曼树算法及C++实现

一、相关概念

1、**叶子结点的权值**（weight）是对叶子结点赋予的一个有意义的数值量。

2、设二叉树有n个带权值的叶子结点，从根节点到各个叶子结点的路径长度与相应叶子结点权值的乘积之和叫做**二叉树的带权路径长度。**

3、给定一组具有确定权值的叶子结点，可以构造出不同的二叉树，将其中带权路径长度最小的二叉树称为**哈夫曼树。**

二、哈夫曼算法基本思想

(1) 以权值分别为W1,W2．．．Wn的n各结点，构成n棵二叉树T1,T2,．．．Tn并组成森林F={T1,T2,．．．Tn},其中每棵二叉树 Ti仅有一个权值为 Wi的根结点；

(2) 在F中选取两棵根结点权值最小的树作为左右子树构造一棵新二叉树，并且置新二叉树根结点权值为左右子树上根结点的权值之和（根结点的权值=左右孩子权值之和，叶结点的权值= Wi）；

(3) 从F中删除这两棵二叉树，同时将新二叉树加入到F中；

(4) 重复(2)、(3)直到F中只含一棵二叉树为止，这棵二叉树就是Huffman树。

三、哈夫曼算法的存储结构

考虑到对于有n个叶子结点的哈夫曼树有2n-1个结点，并且进行n-1次合并操作，为了便于选取根节点权值最小的二叉树以及合并操作，设置一个数组haftree[2n-1],保存哈夫曼树中的各个结点的信息，数组元素的结点结构如下图所示：

weight	lchild	rchild	parent

哈夫曼树的结点结构

其中，weight保存结点权值；

lchild保存该节点的左孩子在数组中的下标；

rchild保存该节点的右孩子在数组中的下标；

parent保存该节点的双亲孩子在数组中的下标。

可以用C++语言中的结构体类型定义上述结点，如下：

```
// 哈夫曼树的结点结构
struct element
{
    int weight;        // 权值域
    int lchild, rchild, parent;  // 该结点的左、右、双亲结点在数组中的下标
};
```

四、哈夫曼算法C++实现

为了判定一个结点是否已经加入哈夫曼树中，可通过parent域的值来确定。初始时parent的值为-1，当某结点加入到树中时，该节点parent域的值为其双亲结点在数组中的下标。

构造哈夫曼树时，首先将n个权值的叶子结点存放到数组haftree的前n个分量中，然后不断将两棵子树合并为一棵子树，并将新子树的根节点顺序存放到数组haftree的前n个分量的后面。

哈夫曼算法用伪代码描述为：

```
1、数组haftree初始化，所有数组元素的双亲、左右孩子都置为-1；
2、数组haftree的前n个元素的权值置给定权值；
3、进行n-1次合并
   3.1 在二叉树集合中选取两个权值最小的根节点，其下标分别为i1，i2；
   3.2 将二叉树i1、i2合并为一棵新的二叉树k。
```

哈夫曼算法完整代码描述如下；

```
#include<iostream>
#include <iomanip>//这个头文件是声明一些“流操作符”的
//比较常用的有:setw(int);//设置显示宽度，left//right//设置左右对齐。 setprecision(int);//设置浮点数的精确度。
using namespace std;
// 哈夫曼树的结点结构
struct element
{
    int weight;        // 权值域
    int lchild, rchild, parent;  // 该结点的左、右、双亲结点在数组中的下标
};
// 选取权值最小的两个结点
void selectMin(element a[],int n, int &s1, int &s2)
{
    for (int i = 0; i < n; i++)
    {
        if (a[i].parent == -1)// 初始化s1,s1的双亲为-1
        {
            s1 = i;
            break;
        }
    }
    for (int i = 0; i < n; i++)// s1为权值最小的下标
    {
        if (a[i].parent == -1 && a[s1].weight > a[i].weight)
            s1 = i;
    }
    for (int j = 0; j < n; j++)
    {
        if (a[j].parent == -1&&j!=s1)// 初始化s2,s2的双亲为-1
        {
            s2 = j;
            break;
        }
    }
    for (int j = 0; j < n; j++)// s2为另一个权值最小的结点
    {
        if (a[j].parent == -1 && a[s2].weight > a[j].weight&&j != s1)
            s2 = j;
    }
}
// 哈夫曼算法
// n个叶子结点的权值保存在数组w中
void HuffmanTree(element huftree[], int w[], int n)
{
    for (int i = 0; i < 2*n-1; i++)     // 初始化，所有结点均没有双亲和孩子
    {
        huftree[i].parent = -1;
        huftree[i].lchild = -1;
        huftree[i].rchild = -1;
    }
    for (int i = 0; i < n; i++)     // 构造只有根节点的n棵二叉树
    {
        huftree[i].weight = w[i];
    }
    for (int k = n; k < 2 * n - 1; k++) // n-1次合并
    {
        int i1, i2;
```

计算机内存(2)
计算机网络(2)
更多

随笔档案
2018年8月 (1)
2018年5月 (2)
2018年4月 (11)
2018年3月 (19)
2018年2月 (2)
2018年1月 (9)
2017年12月 (10)

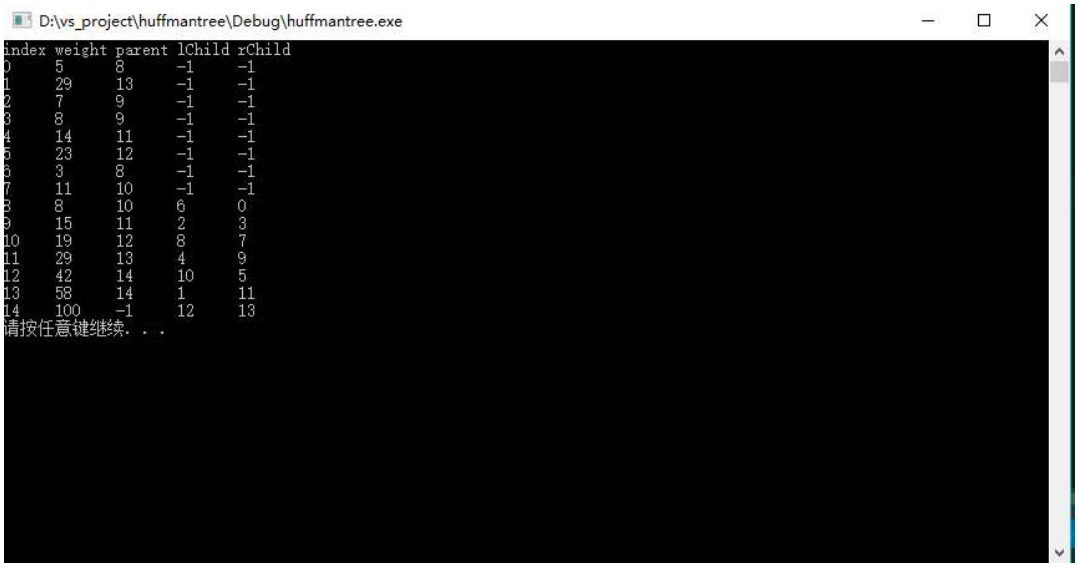
阅读排行榜
1. C++ int与string的
实现)(36342)
2. 最短路径-Dijkstra算
(18829)
3. 图的存储结构(邻接矩
C++实现(8390)
4. 哈夫曼树算法及C++
5. 二叉树的二叉链表存
现(5179)

推荐排行榜
1. OSI七层模型与TCP/
2. 图的存储结构(邻接矩
C++实现(2)
3. 《C++ Primer Plus
——处理数据(2)
4. 一道算法题-八皇后问
(1)

```
        selectMin(huftree, k, i1, i2); // 查找权值最小的俩个根节点，下标为i1,i2
        // 将i1, i2合并，且i1和i2的双亲为k
        huftree[i1].parent = k;
        huftree[i2].parent = k;
        huftree[k].lchild = i1;
        huftree[k].rchild = i2;
        huftree[k].weight = huftree[i1].weight + huftree[i2].weight;
    }

}
   // 打印哈夫曼树
void print(element hT[],int n)
{
    cout << "index weight parent lChild rChild" << endl;
    cout << left;    // 左对齐输出
    for (int i = 0; i < n; ++i)
    {
        cout << setw(5) << i << " ";
        cout << setw(6) << hT[i].weight << " ";
        cout << setw(6) << hT[i].parent << " ";
        cout << setw(6) << hT[i].lchild << " ";
        cout << setw(6) << hT[i].rchild << endl;
    }
}
int main()
{
    int x[] = { 5,29,7,8,14,23,3,11 };         // 权值集合
    element *hufftree=new element[2*8-1];     // 动态创建数组
    HuffmanTree(hufftree, x, 8);
    print(hufftree,15);
    system("pause");
    return 0;
}
```

最后的输出结果为：



参考文献：

[1]王红梅, 胡明, 王涛. 数据结构(C++版)[M]. 北京：清华大学出版社。

2018-01-03

标签： C++， 数据结构， 算法， 二叉树

~君莫笑~
关注 - 17
粉丝 - 7
+加关注

1 0

5. C++ 源代码到可执行
(1)