5.7 Huffman Coding Trees

106

问题引入：

设给出一段报文 CAST CAST SAT AT A TASA

字符集合是 { C, A, S, T }，各个字符出现的频度(次数)是 W={ 2, 7, 4, 5 }

若给每个字符以等长编码

A : 00 T : 10 C : 01 S : 11

则报文总编码长度为 (2+7+4+5) * 2 = 36 bits

字符平均长度= 36/18 = 2 bits

若按各个字符出现的概率不同而给予不等长编码，可望减少总编码长度

A : 0 T : 10 C : 110 S : 111

它的总编码长度为

7*1+5*2+(2+4)*3 = 35 bits 。

字符平均长度= 35/18 = 1.944 bits 比等长编码的情形要短

哈夫曼编码

107

- 结点间路径长度(Path Length)

  连接两结点的路径上的分支数

- 结点的路径长度(又称结点的深度)

  从根结点到该结点的路径上分支的数目

- 叶子的加权路径长度（weighted path length of a leaf)

  叶子的深度与叶子的权值之积

- 树的外部路径权值 (External path weight)，也称树的加权路径长度(Weighted Path Length of a tree, WPL)

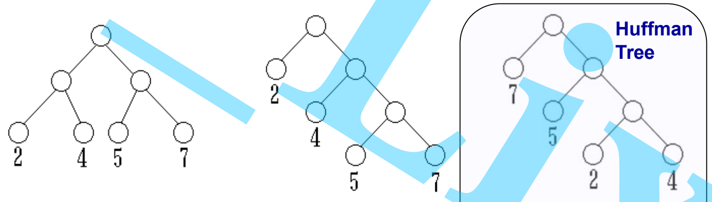
  树的所有叶结点的加权路径长度之和

$$WPL = \sum_{i=0}^{n_0-1} (w_i * l_i)$$

108

5.7.1 哈夫曼树定义 (Huffman Tree)

在所有 含n_0个带确定权值叶子结点的 二叉树中，必存在一棵加权路径长度最小的树，称该树为“最优树”,或“哈夫曼树” (Huffman Tree)



(a) $WPL=36$ (b) $WPL=46$ (c) $WPL=35$

1）叶结点的权值越小，离根越远

2）叶结点的权值越大，离根越近

哈夫曼树的特点之一

109

哈夫曼树 (Huffman Tree)

(c) $WPL = 35$　　WPL=37

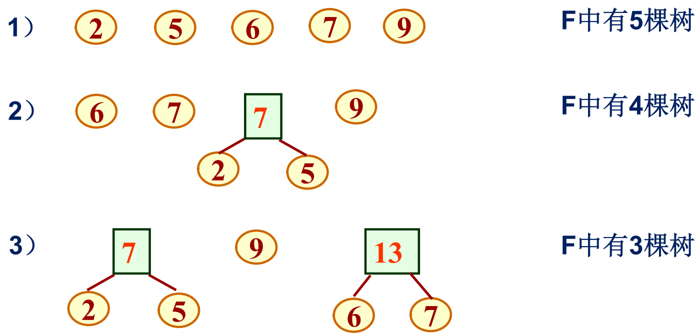
Huffman Tree 特点一:
1）叶结点的权值越小，离根越远
2）叶结点的权值越大，离根越近
特点二:
满二叉树，FBT

5.7.2 Huffman Tree Construction

1. 根据给定的 n 个符号权值对 { {s_1, w_1}, {s_2, w_2},, …, {s_n, w_n}, }，造 n 棵二叉树的集合 $F = \{T_1, T_2, \ldots, T_n\}$，其中每棵二叉树中均只含一个符号权 值对为 (s_i, w_i)的根结点，其左、右子树为 空树；
2. 在 F 中选取根结点权值最 小的两棵二叉树，分别作为左(最小)、右(次小)子树构造一棵新的二叉树，并置这棵新的二叉树根结点的权值 为其左、右子树根结点的权值之 和；
3. 从F中删去这两棵树，同时加入刚生成的新树；
4. 重复 (2) 和 (3) 两步，直至 F 中只含一棵树为止。

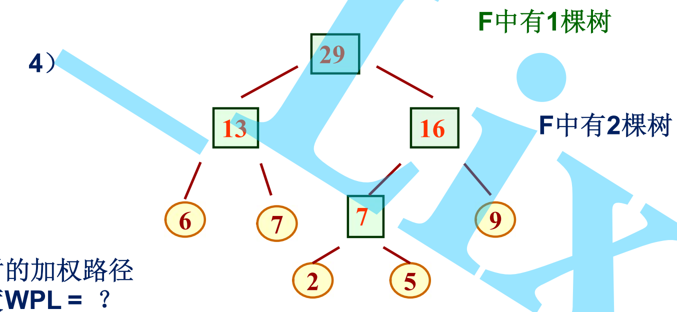
Huffman Tree Construction (2)

例1：已知字符集 { A, C,V, E, K }的出现频数为 { 5, 6, 2, 9, 7 }，以频数为权值构造哈夫曼树



Huffman Tree Construction (3)

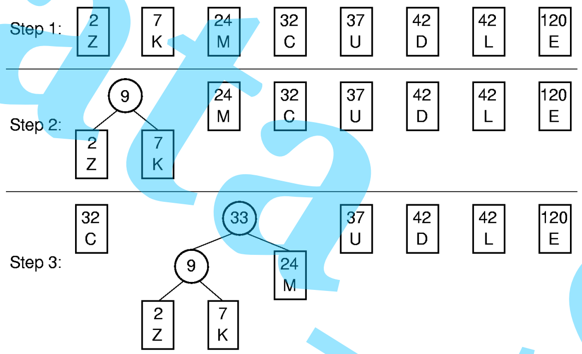
例1：已知字符集 { A, C,V, E, K }的出现频数为 { 5, 6, 2, 9, 7 }，以频数为权值构造哈夫曼树



该树的加权路径长度WPL = ？

Huffman Tree Construction (4)

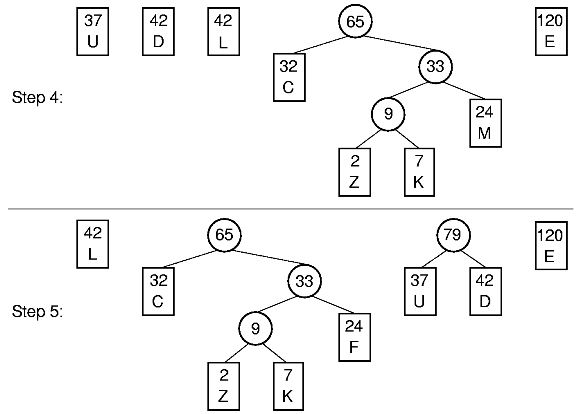
example2:



114

Huffman Tree Construction (5)

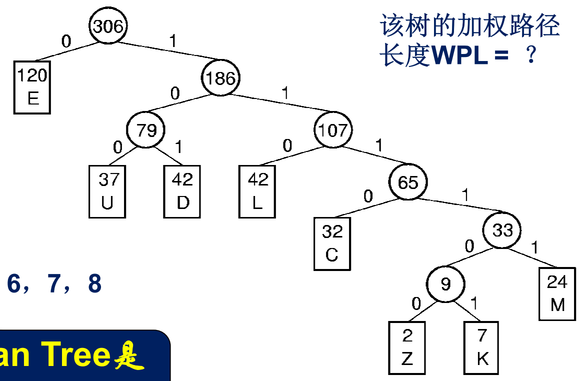
example2:



115

Huffman Tree Construction (6)

example2:



该树的加权路径长度WPL = ？

step 6，7，8

Huffman Tree是full(满)二叉树

116

请 思 考！

1. 若所有叶子权值相同，在huffman树中是否深度也相同呢？
2. 若叶子权值不同，在huffman树中是否深度就一定不同呢？

字符集合：{ A, B, C, D, E }
权值*W*: { 1, 1, 1, 1, 1}

字符集合：{ A, B, C, D }
权值*W*: { 3, 4, 5, 6}

字符集合：{ A, B, C, D, E, F, H, I }
权值*W*: { 1, 1, 1, 1, 1, 1, 1, 1 }

字符集合：{ A, B, C, D }
权值*W*: { 4, 4 5, 9}

若叶结点个数为2的整数次方且所有叶节点权值相同(近)，则对应的Huffman Tree是一个叶子节点均在最底层(即深度相同)的完全满二叉树

117

HuffmanTree class implement (pointer based)

需要定义3个class： FreqPair，HuffNode，HuffTree

FreqPair class

```
template <class Elem>
class FreqPair {
private:
        Elem symbol;
        int  freq;
public:
        FreqPair( Elem s, int f) {  symbol = s;  freq = f;}
        int weight() { return freq;  }
        Elem  val() { return symbol;  }
};
```

118

118

HuffNode class(1)

```
template <class Elem>
class HuffNode {      // Abstract base class， 可变类型结点
public:
          virtual  int weight()=0;
          virtual bool isLeaf() = 0;
};
template <class Elem>      // Leaf  of huffman tree
class LeafNode : public HuffNode<Elem> {    //<Elem>
private:
           FreqPair <Elem>  * it;            // freq pair
public:
          LeafNode(  Elem val, int freq)
          {  it = new FreqPair<Elem>(val, freq);  }     // Constructor
           int weight() {return it->weight();}
          bool isLeaf() { return true; }
          FreqPair<Elem>*  val() { return it; }
};
```

119

119

HuffNode class(2)

```
template <class Elem>   // Internal node of huffman tree
class IntlNode : public HuffNode<Elem> {
private:
          HuffNode<Elem>* lc;     // Left child
          HuffNode<Elem>* rc;    // Right child
          int  wgt;        // weight value
public:
           IntlNode( HuffNode<Elem>* l, HuffNode<Elem>* r)
          { wgt = l->weight()+ r->weight(); lc = l; rc = r; }
           int weight() {return wgt; }
           bool isLeaf() { return false;  }
          HuffNode<Elem>* left() { return lc; }
          void setLeft(HuffNode<Elem>* l)  {  lc = (HuffNode*)l;  }
          HuffNode<Elem>* right()  { return rc;   }
          void setRight(HuffNode<Elem>* r)   {  rc = (HuffNode*)r;   }
};
```

120

120

HuffTree class(1)

```
template <class Elem>
class HuffTree {
private:
   HuffNode<Elem>* myroot;
   void printhelp(HuffNode<Elem>* subroot, int level) const {  //相当于中序遍历
        FreqPair<Elem>* s1;
        if (subroot==NULL)  return;
       if (subroot->isLeaf()) {     // Do leaf node
         for(int i=0; i<level; i++)   cout << "*";
         cout << "Leaf: ";
         s1=((LeafNode<Elem> *)subroot)->val ();
         cout<<s1->val()<<"  "<<s1->weight()<< endl;   }
      else {
         printhelp(((IntlNode<Elem> *)subroot)->left(),level+1);  //打印左树
         for(int i=0; i<level; i++)    cout << "*";         //打印根
          cout << "Internal: ";
         cout<< ((IntlNode<Elem> *)subroot)->weight()<<endl;
         printhelp(((IntlNode<Elem> *)subroot)->right(),level+1); //打印左树
      }
  }
```

121

121

HuffTree class(2)

```
public:
   HuffTree(Elem val, int freq)  {
         myroot = new LeafNode<Elem>(val,freq);
   }
   HuffTree(HuffTree<Elem>* l, HuffTree<Elem>* r) {
         myroot = new IntlNode<Elem>(l->root(),r->root());
   }

   ~HuffTree( ) { ......  }   //可参考BST class 补写
   clear( ) {  ...... }        //可参考BST class 补写
   HuffNode<Elem>*  root()  { return myroot;   }
   int weight() {  return myroot->weight(); }
   void print( ) const {     //相当于中序遍括历
        if (myroot == NULL)  cout << "The huffTree is empty.\n";
        else   printhelp(myroot, 0);
   }
   ... ...
};
```

122

5.7.3 哈夫曼编码

利用哈夫曼树可以构造一种不等长的二进制编码，并且构造所得的哈夫曼编码是一种最优前缀编码，即所传电文的总长度最短。

前缀编码：任何一个字符的编码都不是同一字符集中另一个字符的编码的前缀。

Letter	code
A :	0
T :	10
C :	110
S :	111

Letter	Freq	Code	Bits
C	6	00	2
K	7	01	2
E	9	10	2
A	5	110	3
V	2	111	3

Letter	code
C:	1110
D:	101
E:	0
M:	11111
K:	111101
L:	110
U:	100
Z:	111100

123

哈夫曼编码(2)

----步骤

给定待编码字符集，哈夫曼编码步骤如下：

1. 将各个字符的频度作为权值，构造哈夫曼树；

2. 给每个内部结点左右两条边赋0和1的权值；

3. 将每个叶结点从根开始路径上的权值按顺序排列，即得该叶结点对应字符的编码码字。

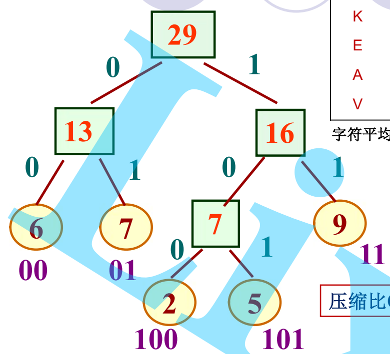
124

哈夫曼编码(3)---example1

Huffman编码(不等长)

Letter	Freq	Code	Bits
C	6	00	2
K	7	01	2
E	9	11	2
A	5	101	3
V	2	100	3

字符平均编码长度= 65/29=2.24



等长编码

Letter	Code	Bits
C	000	3
K	001	3
E	010	3
A	100	3
V	101	3

字符平均编码长度= 3

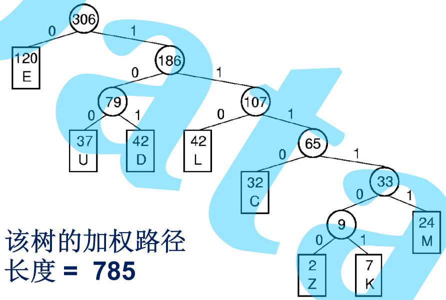
压缩比CR= 1.33

若有字符串KEAC，其huffman编码结果：0111101100

等长编码结果：001010100000

125

哈夫曼编码(4) ---example2



Letter	Freq	Code	Bits
C	32		
D	42		
E	120		
M	24		
K	7		
L	42		
U	37		
Z	2		

该树的加权路径
长度 = 785

Huffman编码：字符平均编码长度= 785/306 = 2.565

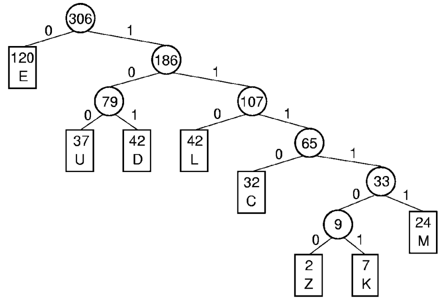
等长编码： 字符平均编码长度 = log(8)=3

CR = 3/2.565=1.17

126

126

哈夫曼编码(5) ---example2



Letter	Freq	Code	Bits
C	32	1110	4
D	42	101	3
E	120	0	1
M	24	11111	5
K	7	111101	6
L	42	110	3
U	37	100	3
Z	2	111100	6

Encode for EEEE: 0000

Encode for DEED: 10100101

Encode for ZZKZ: 111100111100111101111100

127

127

请 思 考

- 若各叶子权值相同，哈夫曼编码码字的bit数是否相同呢？
- 若叶子权值不同，哈夫曼编码码字的bit数是否一定不同呢？

字符集合: { A, B, C, D, E }　　{ A, B, C, D }
权值*W:* { 1, 1, 1, 1, 1}　　{ 3, 4, 5, 6 }

字符集合: { A, B, C, D, E, F, H, I }　　{ A, B, C, D }
权值*W:* { 1, 1, 1, 1, 1, 1, 1, 1 }　　{ 4, 4 5, 9}

若叶结点个数为2的整数次方且各叶节点权值相同(近)，则对应的Huffman Tree是一个叶子节点均在最底层(码字长度相同)完全满二叉树，此时编码结果等同于等长编码

否则呢？ Huffman编码比等长编码有优势吗？

128

本章我们学到了

- 树的基本概念和定义
- 二叉树的定义和性质
- 二叉树的遍历， 前序， 中序， 后序， 广度优先
- 二叉树结点的执行： 基于指针、 基于数组
- 二叉搜索树(BST)： 定义， 基本操作
- 堆(heap)/优先队列： 定义， 基本操作
- huffman编码树： 定义， 构建， 编码

129

129

本章作业三

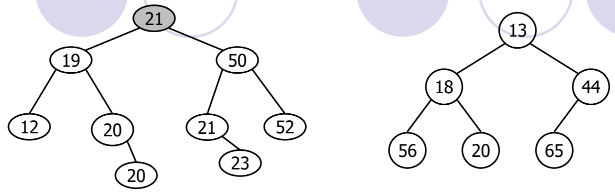
- 5.26 (b)
- 5.28
- 5.32

130

130

课堂测验

1 给出从左下图BST中删除根结点处21后的结果



2. 给出对右上图小堆执行RemoveFirst 操作的具体过程及最终结果。

3. 给定字符权值对，

1）构建哈夫曼树，

2）写出每个字符对应的码字，并求平均码字长度

3）对字符串 DEEMKLE 进行哈夫曼编码

4) what is the expected number of bits required by the Huffman code for a message containing 2000 characters

D	22
E	120
M	24
K	12
L	18

131

131

Chapter5 end

132

132