哈夫曼树和哈夫曼编码

- 每个初始结点最终都成为叶结点，且**权值越小**的结点到根结点的**路径长度越大**;
- 构造过程中共新建 $n-1$个结点(**双分支结点(非叶子结点)**,每两个结点合并成一个),哈夫曼树的结点总数为 $n+n-1=2n-1$(**常考**);
- 哈夫曼树中不存在度为 1的结点;
- 哈夫曼树不唯一,但 $WPL=\sum_{i=1}^{n} w_i \times l_i$(其中 w_i表示第 i个叶节点所带的权值, l_i是该叶结点到根结点的路径长度(**经过的边**))必然相同且最优.
- 若没有一个编码是另一个编码的前缀,则称这样的编码为**前缀编码**.例如, 11是 1101的前缀,故不满足前缀编码.

并查集

在采用树的**双亲指针数组**表示作为并查集的存储表示时,集合元素的编号从 0到 $SIZE-1$,其中 $SIZE$是最大元素的个数.

```
#define SIZE 110
int UFSets[SIZE];
```

- 并查集的初始化操作

```
void Initial(int S[])//并查集的初始化
{
    for(int i=0;i<SIZE;i++)
        S[i]=-1;
}
```

- 并查集的查询操作,最坏时间复杂度为 $O(n)$

```
int Find(int S[],int x)//并查集的查询操作(主要)
{
    while(S[x]>=0)//循环找到x的根
        x=S[x];
    return x;//根的S[]小于0
}
```

- 并查集的合并操作, $Union$函数时间复杂度为 $O(1)$

```
void Union(int S[],int root1,int root2)//并查集的合并操作(主要)
{
    if(root1==root2)
        return ;
    S[root2]=root1;
}

void Join(int S[],int x,int y)//查询并合并操作
{
    int fx=Find(S,x),fy=Find(S,y);
    Union(S, fx, fy);
}
```

- 关于$Union$操作的优化

用**根结点的绝对值**表示树的结点总数,让小树合并到大树.该方法构造的树高不超过 $\lfloor \log_2 n \rfloor + 1$,当$Union$操作优化后,$Find$操作最坏时间复杂度为 $O(\log_2 n)$.

```
void Union_Optimize(int S[],int root1,int root2)//并查集的合并优化操作
{
    if(root1==root2)
        return ;
    if(S[root2]>S[root1])//root2所在树是小树,结点数少(注意是负数比较)
    {
        S[root1]+=S[root2];
        S[root2]=root1;
    }
    else//反之,roo1所在树是小树
    {
        S[root2]+=S[root1];
        S[root1]=root2;
    }
}

void Join(int S[],int x,int y)//查询并合并操作
{
    int fx=Find(S,x),fy=Find(S,y);
    //Union(S, fx, fy);
    Union_Optimize(S, fx, fy);
}
```

- 关于$Find$操作的优化(路径压缩)

先找到根结点,再将查找路径上的所有节点都挂到根结点下.

```
//非递归实现
int Find_Optimize(int S[],int x)//并查集的查询优化操作(主要)
{
    int root=x;
    while(S[root]>=0)//循环找到根
        root=S[root];
```

```
    while(x!=root)
    {
        int t=S[x];
        S[x]=root;
        x=t;
    }
    return root;
}

//递归实现
int Find_Optimize(int S[],int x)//并查集的查询优化操作(主要)
{
    if(S[x]>=0)
        return S[x]=Find_Optimize(S,S[x]);
    return x;
}
```

每次 $Find$操作,先找根,再**压缩路径**,可使树的高度不超过 $O(\alpha(n))$,其中 $\alpha(n)$是一个增长很缓慢的函数,对于常见的 n值,通常 $\alpha(n) \leq 4$,因此优化后并查集的 $Find$和 $Union$操作时间开销很低.

> 在有 n个叶结点的哈夫曼树中,非叶结点的总数是()
>
> A. $n-1$
>
> B. n
>
> C. $2n-1$
>
> D. $2n$

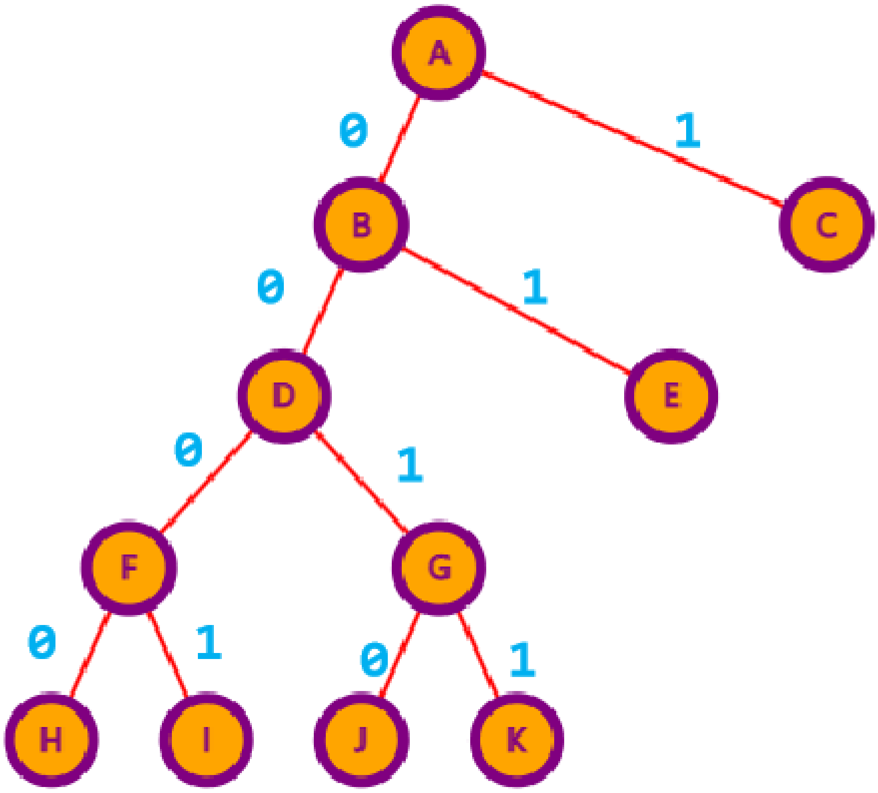
套结论:构造过程中共新建 $n-1$个结点(**双分支结点(非叶子结点)**,每两个结点合并成一个),哈夫曼树的结点总数为 $n+n-1=2n-1$(**常考**)

> 下列编码中,(B)不是前缀码.
>
> A. `{00,01,10,11}`
>
> B. `{0,1,00,11}`
>
> C. `{0,10,110,111}`
>
> D. `{10,110,1110,1111}`

若没有一个编码是另一个编码的前缀,则称这样的编码为**前缀编码**.在 A, C, D中均满足该条件,而在 B中, 0是 00的前缀, 1是 11的前缀,故不满足条件.

> 设哈夫曼编码的长度不超过 4，若已对两个字符编码为 1和 01，则还最多可对()个字符编码.
>
> A. 2
>
> B. 3
>
> C. 4
>
> D. 5

将该哈夫曼编码树构造出来,注意**一个编码不能是任何其他编码的前缀**.



其中 01和 1已使用,剩下 4种,分别是 $0000, 0001, 0010, 0011$.

> 一棵哈夫曼树共有 215个结点，对其进行哈夫曼编码，共能得到()个不同的码字.
>
> A. 107
>
> B. 108
>
> C. 214
>
> D. 215

构造过程中共新建 $n-1$个结点(**双分支结点(非叶子结点)**,每两个结点合并成一个),哈夫曼树的结点总数为 $n+n-1=2n-1$(**常考**)

设码字(叶子结点)数为 x,已知总结点数为 215,即 $2\times x-1=215$,故结果为 $x=108$.

> 若度为 m的哈夫曼树中，叶子结点个数为 n,则非叶子结点的个数为().
>
> A. $n-1$
>
> B. $\lfloor \frac{n}{m} \rfloor -1$
>
> C. $\lceil \frac{n-1}{m-1} \rceil$
>
> D. $\lceil \frac{n}{m-1} \rceil -1$

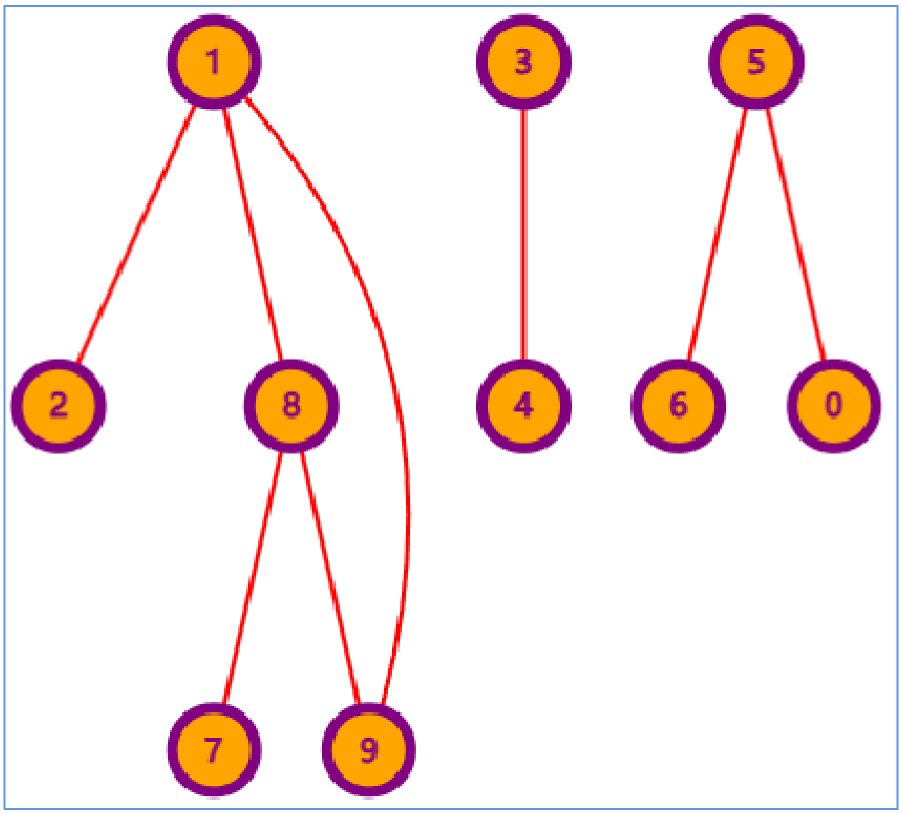
设非叶子结点的个数为 n_m, $总结点数=n+n_m$;

边数(度数)贡献主要由非叶子结点产生,且`边数(度数)=总结点数-1`,故 $总结点数-1=n_m\times m$;

联立两式得: $n_m\times m+1=n+n_m$,故 $n_m=\frac{n-1}{m-1}$,此处上取整的原因是主要进行比较操作(多合一),存在余数时亦需要进行处理.

> 并查集中最核心的两个操作是:①查找,查找两个元素是否属于同一个集合;②合并,如果两个元素不属于同一个集合,且所在的两个集合互不相交,则合并这两个集合.假设初始长度为 $10(0\sim 9)$的并查集,按 $1-2$、$3-4$、$5-6$、$7-8$、$8-9$、$1-8$、$0-5$、$1-9$的顺序进行查找和合并操作,最终并查集共有()个集合.
>
> A. 1
>
> B. 2
>
> C. 3
>
> D. 4

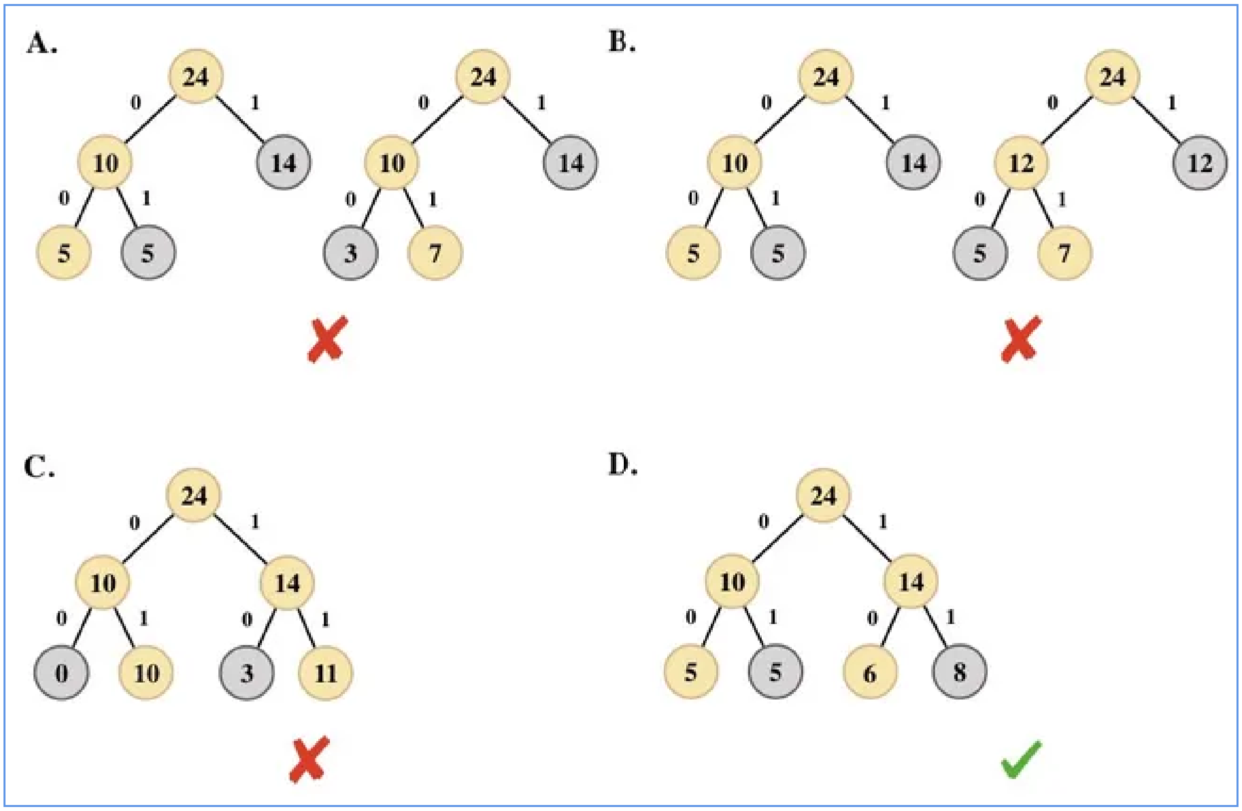
如下图所示,并查集被分成 3个集合.



> **2015统考真题**：下列选项给出的是从根分别到达两个叶结点路径上的权值序列，属于同一棵哈夫曼树的是(D).
>
> A. $24, 10, 5$和 $24, 10, 7$
>
> B. $24, 10, 5$和 $24, 12, 7$
>
> C. $24, 10, 10$和 $24, 14, 11$
>
> D. $24, 10, 5$和 $24, 14, 6$

- 选项 A的根结点的左孩子 10的左右孩子权值出现了两种情况，意味着这两条路径一定不会出现在同一棵哈夫曼树中.
- 选项 B的根结点 24的左右孩子权值出现了两种情况，意味着这两条路径一定不会出现在同一棵哈夫曼树中.
- 选项 C虽然成功构造出了一棵完满二叉树，但这棵完满二叉树**不是一棵哈夫曼树**,最小的两个数 0和 3最先结合在一起.

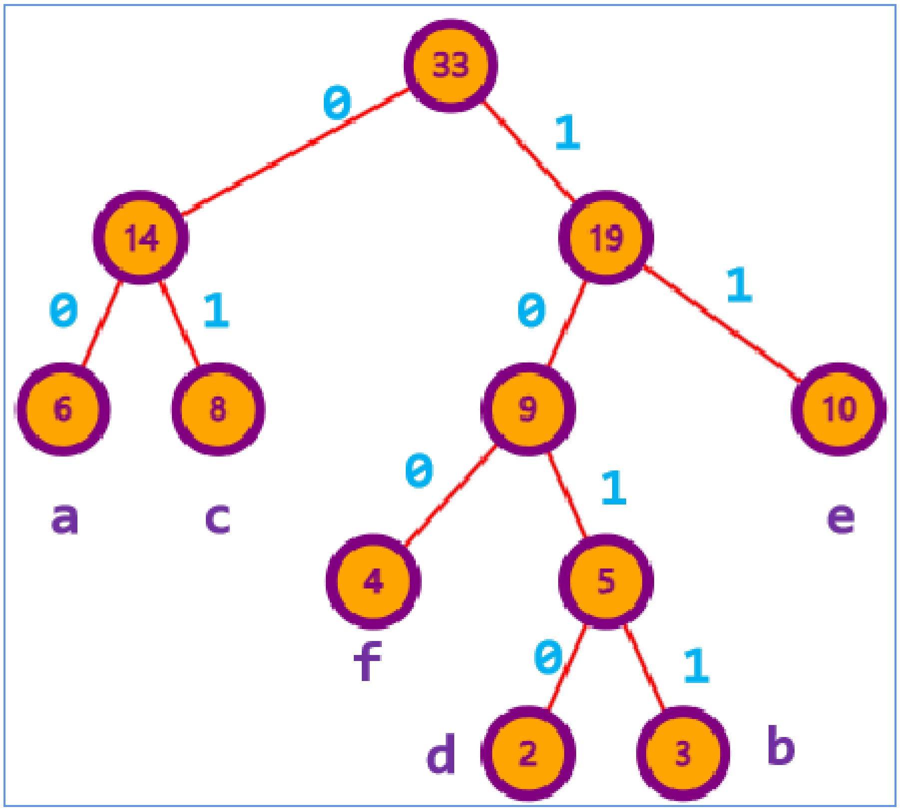
- 选项 D的这棵完满二叉树是一棵哈夫曼树



> **2017统考真题**：已知字符集`{a,b,c,d,e,f,g,h}`,若各字符的哈夫曼编码依次是 $0100, 10, 0000, 0101, 001, 11, 0001$,则编码序列 0100011001001011110101的译码结果是()
>
> A. $acgabfh$
>
> B. $adbagbb$
>
> C. $afbeagd$
>
> D. $afeefgd$

> **2017统考真题**：已知字符集`{a,b,c,d,e,f}`,若各字符出现大的次数分别为 $6, 3, 8, 2, 10, 4$,则对应字符集中各字符的哈夫曼编码可能是(A)
>
> A. $00, 1011, 01, 1010, 11, 100$
>
> B. $00, 100, 110, 000, 0010, 01$
>
> C. $10, 1011, 11, 0011, 00, 010$
>
> D. $0011, 10, 11, 0010, 01, 000$

首先构造出对应的哈夫曼树,到底是 0还是 1,具体代入测试即可.



> **2019统考真题**：对 n个互不相同的符号进行哈夫曼编码。若生成的哈夫曼树共有 115个结点，则 n的值是().
>
> A. 56
>
> B. 57
>
> C. 58
>
> D. 60

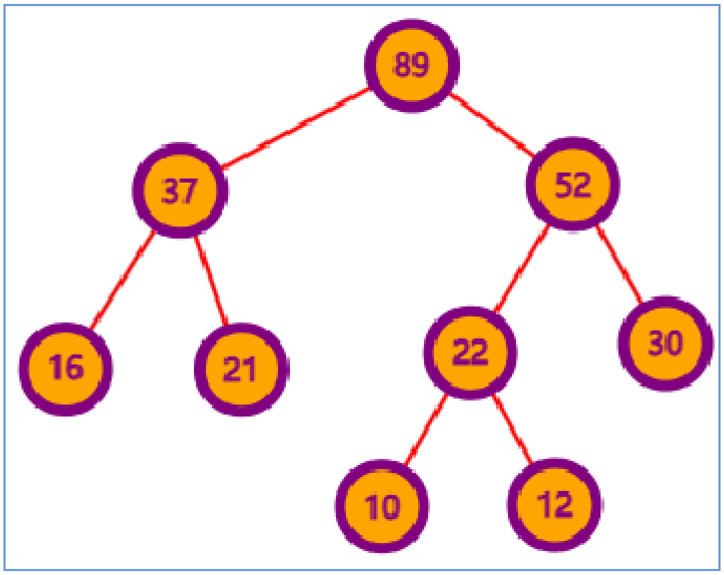
构造过程中共新建 $n-1$个结点(**双分支结点(非叶子结点)**,每两个结点合并成一个),哈夫曼树的结点总数为 $n+n-1=2n-1$(**常考**)

已知设码字(叶子结点)数为 n,总结点数为 115,即 $2\times n-1=115$,故结果为 $n=58$.

> **2021统考真题**：若某二叉树有 5个叶结点,其权值分别为 $10,12,16,21,30$,则其最小的带权路径长度(WPL)是()
>
> A. 89
>
> B. 200
>
> C. 208
>
> D. 289

如下图所示,计算得 $WPL=\sum_{i=1}^{n} w_i\times l_i$

$=(10+12)\times 3+(16+21+30)\times 2=66+134=200$



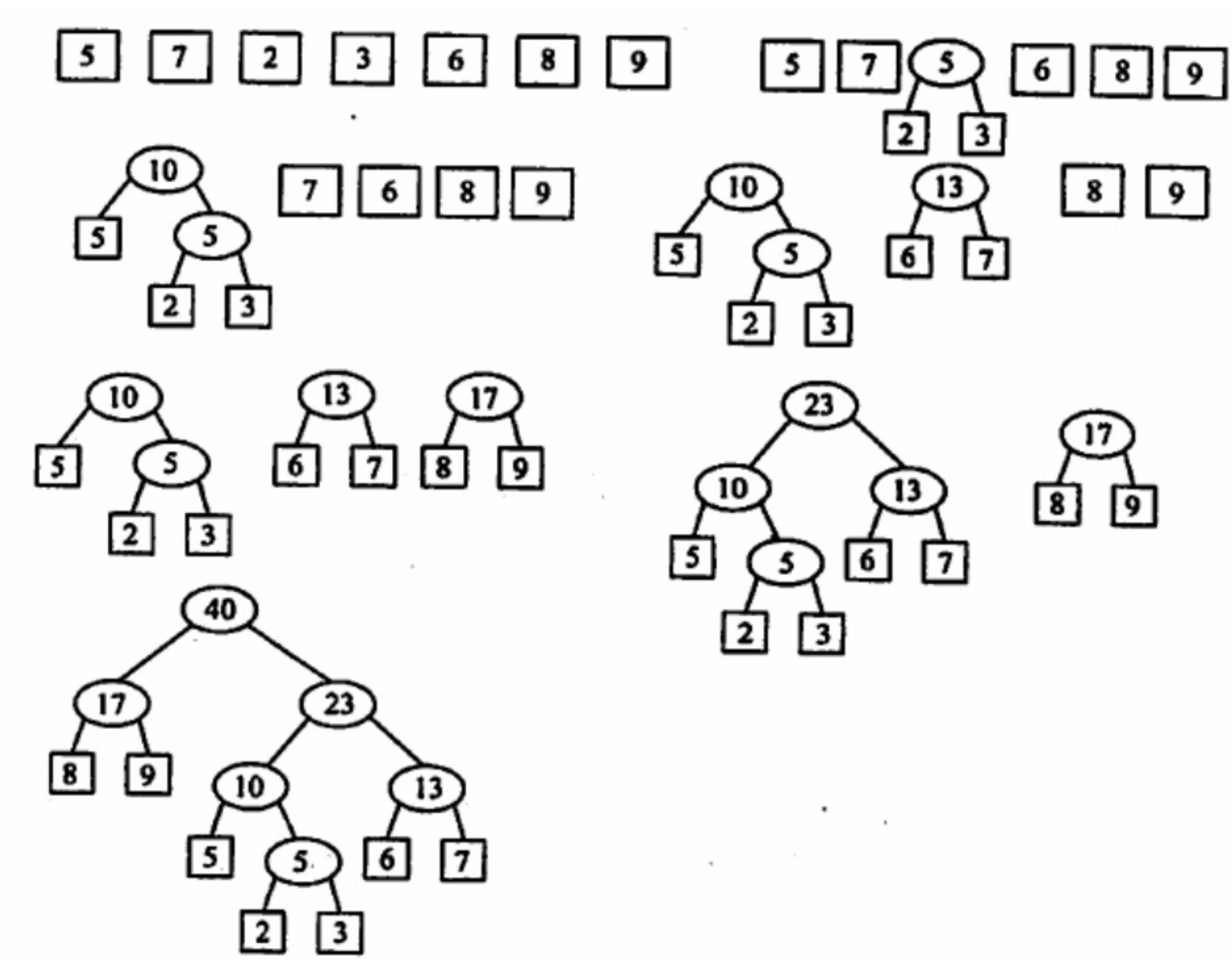
5.5.1

> 设给定权集 `w={5,7,2,3,6,8,9}`,试构造关于 w的一棵哈夫曼树,并求其加权路径长度 WPL.

如下图所示,计算得 $WPL=\sum_{i=1}^{n} w_i \times l_i$

$=(2+3)\times 4+(5+6+7)\times 3+(8+9)\times 2$

$=20+54+34=108$



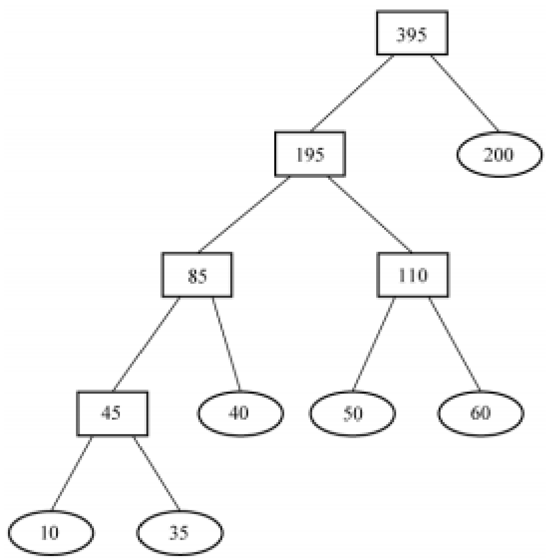
5.5.2

> **2012统考真题**：设有 6个有序表 A,B,C,D,E,F，分别含有 $10,35,40,50,60$和 200个数据元素，各表中的元素按升序排列。要求通过 5次两两合并、将 6个表最终合并为 1个升序表，并使最坏情况下比较的总次数达到最小。请回答下列问题：
>
> 1)给出完整的合并过程,并求出最坏情况下比较的总次数;
>
> 2)根据你的合并过程,描述 n($n\geq 2$)个不等长升序表的合并策略,并说明理由.

1)

6 个表的合并顺序如下图所示，实际上相当于以各有序表的长度为**权值**，构建一棵哈夫曼树。

合并两个长度分别为 m和 n的有序表，最坏情况下需要比较 $m+n-1$次.



- 第 1次合并：表 A与表 B合并，生成含有 $10+35=45$个元素的表 AB,最多比较次数为 $10+35-1=44$;
- 第 2次合并：表 AB与表 C合并，生成含有 $45+40=85$个元素的表 ABC,最多比较次数为 $45+40-1=84$;
- 第 3次合并：表 D与表 E合并，生成含有 $50+60=110$个元素的表 DE,最多比较次数为 $50+60-1=109$;
- 第 4次合并：表 ABC与表 DE合并，生成含有 $85+110=195$个元素的表 $ABCDE$,最多比较次数为 $85+110-1=194$;
- 第 5次合并：表 $ABCDE$与表 F合并，生成含有 $195+200=395$个元素的表 $ABCDEF$,最多比较次数为 $195+200-1=394$;

比较的总次数为 $44+84+109+194+394=825$

2)

在对多个有序表进行两两合并时，若表长不同，则最坏情况下总的比较次数依赖于表的合并次序，可以借用**哈夫曼树**的构造思想，依次选择最短的两个表进行合并，可以获得最坏情况下最佳的合并效率。

5.5.3

> **2020统考真题**：若任意一个字符的编码都不是其他字符编码的前缀，则称这种编码具有前缀特性。现有某字符集(字符个数 ≥ 2)的不等长编码，每个字符的编码均为二进制的 0、 1序列，最长为 L位，且具有前缀特性。请回答下列问题：
>
> 1)哪种数据结构适宜保存上述具有前缀特性的不等长编码?
>
> 2)基于你所设计的数据结构，简述从 $0/1$串到字符串的译码过程。
>
> 3)简述判定某字符集的不等长编码是否具有前缀特性的过程。

1)

二叉树或**哈夫曼树**.普通的二叉树也可以设计前缀编码,哈夫曼树会使总长最小,是对前者的优化.

2)

将所有的字符信息存储到二叉树的叶子结点上,且约定左分支表示字符 0,右分支表示字符 1,则可将**根结点到叶子结点的路径上分支字符组成的字符串**作为该叶子结点字符的编码.从根结点出发将 $0/1$串沿着分支探查下去,遇到带有信息的叶子结点即为一个字符,然后再从根结点出发,依次类推直至 $0/1$串全部译码为字符串.

3)

只需判定**存储有字符信息的结点是否全部为叶子结点**即可.若存储有某个字符信息的结点是非叶子结点,那么它的 $0/1$编码一定是它孩子结点 $0/1$编码的前缀,违反前缀特性.