D.S.复习提纲

第1章：

数据，数据结构，基本类型，抽象数据类型，Java语言的面向对象编程、递归的概念与实现。

- 主要能用递归思想写出算法

例子： **ppt-----递归例1 求n!**

作业--------- 例2 求数组中的最大值

例3 求数组元素的平均值

例2,例3如果用链表来实现呢？

复习例题----例4 统计二叉树中的叶结点个数

例5 交换每个结点的左子女和右子女

例1.　　求n!

factorial function f(n)=n!

$$f(n)=\begin{cases} 1 & n<=1 \quad \text{(base) //递归终结条件} \\ n*f(n-1) & n>1 \text{ (recursive component) //递归部分} \end{cases}$$

f(5)=5*f(4)=5*4f(3)=5*4*3f(2)=5*4*3*2f(1)=120

```
static long factorial (int n)
{   if ( n <= 1)
          return  1;
    else return n* factorial( n-1 )
 }
```

例2. 求数组中的最大值

```
public static int findMax(int[] a, int n){
    //n表示n个元素，它们在数组a中
        if(n= =1){
            return a [0];
         }
         else{
            int temp=findMax(a,n-1);
            return temp>a [n-1]?temp:a [n-1];
         }
    }

int max(int a[],int n)
 {  if(n = = 1) return a[0];
    int m = max(a,n-1);
    if( m > a[n-1] )
         return m;
    else
         return a[n-1];
}
```

例2. 求数组中的最大值

如果用链表来实现表:

求链表中的最大值

```
int GetMaxInt( ListNode f )
{  if( f.link = = NULL ) return f .data ;
    else
        {   int i = GetMaxInt( f.link );
            if ( i > f .data ) return i ;
            else  return f .data ;
        }
 }
```

或 else return (f . data) > (GetMaxInt(f . link)) ? f . data : GetMaxInt(f . link);

例3．求数组元素的平均值

```
float average( int a[],int n)
{  if(n = = 1)
        return a[0];
    else
        return (average(a,n-1)*(n-1)+a[n-1])/n;
 }
```

如果用链表：

```
 float Average( ListNode f, int n )
{   if( f.link = = NULL ) return f.data;
    else return ( Average ( f.link, n-1 ) * ( n-1 ) + f.data ) / n;
}
```

例4. 统计叶子结点个数

```
int leafNum ( BinTreeNode <Type> * root )
{
    if ( root = = NULL ) return 0 ;
    if ( root->leafchild = = NULL && root->rightchild = = NULL )
            return 1;
    else return leafNum( root-> leftchild ) + leafNum ( root-> rightchild ) ;
}
```

例5. 交换左右子树

```
void Swapchild ( BinTreeNode * p )
{  if ( p = = NULL ) return ;
   BinTreeNode * temp = p -> left ;
   p ->left = p -> right ;
   p -> right = temp;
   Swapchild ( p ->left );
   Swapchild (p ->right );
}
```

第2章　算法分析

最佳、最差和平均情况下的复杂度差异;

大O、Ω和 θ 符号

1）分析某个语句的执行次数（频度）

2）分析某个程序段执行的时间复杂度（用大O表示，要求写出推导过程）

ppt:-----对排序算法与查找算法的分析

例1----
```
for (int i = 1; i <= n;  i++)
    for (int j = 1; j<=n; j++)
    {   c[i][j] = 0.0;
        for ( int k = 1; k <= n; k++)
            c[i][j] = c[i][j]+a[i][k]*b[k][j];
    }
```

次数为:　n*n*n

第2章　算法分析

例2.
```
x = 0; y = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
        for (int k = 1; k <= j; k++)
            x = x+y;
```
次数为:　n*(n+1)*(n+2)/6

例3.
```
int x = 91;  int y = 100;
while(y>0)
{   if(x>100) { x -= 10;  y--; }
    else x++;
}
```
1100次

第3章 表、栈和队列

表、栈和队列的（基本概念，顺序存储结构，链式存储结构，应用），

表：

逻辑-----（e_1, e_2, …..e_n)

物理------数组实现

链表实现------单链表

循环链表

双向链表

表头结点

cursor

操作------查找、插入、删除等

ppt----多项式相加

约瑟夫问题

双链表的插入、删除

用链表实现

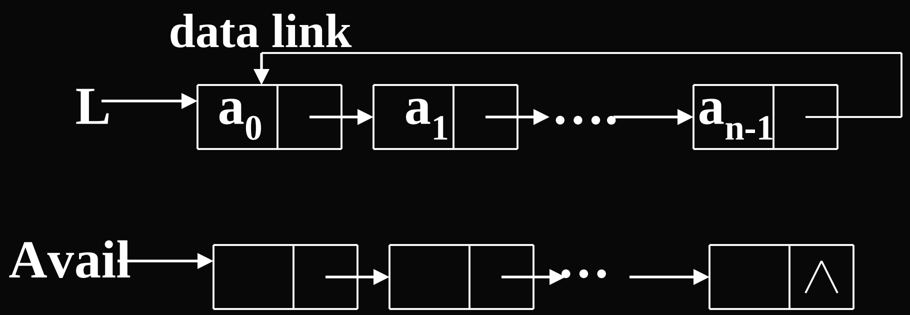
例题----逆转链表等题

例1. 逆转链表（假设不带表头结点）

```
public  void  inverse( ListNode f )
{  if ( f = = NULL ) return;
   ListNode  p = f . link ;  pr = NULL;
   while ( p ! = NULL )
  {   f . link = pr ;
      pr = f ;
      f = p ;
      p = p . link ;
  }
   f . link = pr ;
 }
```

第3章

例2. 设有如下结构的循环链表和可利用空间表



请在常数时间内实现将L链表中的所有结点归还到可利用空间表

```
ListNode p = L.link;
L.link = Avail;
Avail = p;
```

栈、队列

定义-----栈的定义， 队列的定义

机内实现------数组 （循环队列）

单链表

应用

栈-----对表达式求值。中缀----后缀----对后缀表达式求值

递归函数的实现。

PPT：第4章中用非递归实现中序,后序遍历(在第4章中讲)

队列---循环队列的补充题：已知队尾元素的位置与元素的个数，

求队头元素的位置。

中缀到后缀：

(a+b)*((c-d)/2*e)-----→ ab+cd-2/e**

用了什么栈？

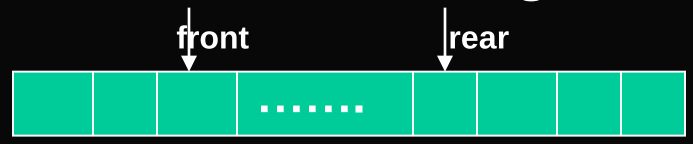
对后缀表达式求值：
用了什么栈

例2. 队列---循环队列的补充题

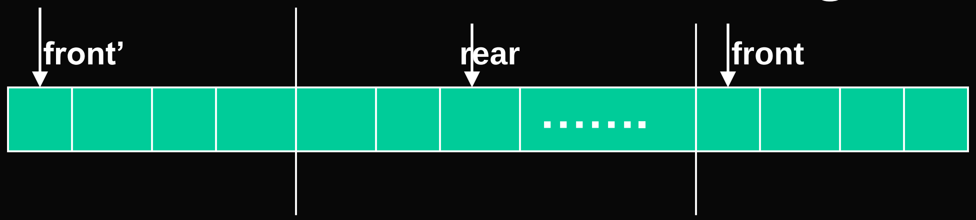
已知队尾元素的位置与元素的个数， 求队头元素的位置。

先用实例来分析，然后归结到一般情况。

情况一: front=rear-length+1



情况二: front=rear-length+1+m



合并: front=(rear-length+1+m)%m

特殊矩阵的压缩存储

Arrays and Matrix

1D-Array

1. One-dimensional array

1D-array is a limited sequence composed of n ($n \geq 0$) elements which are of the same data type.

For example:

	0	1	2	3	4	5	6	7	8	9
a	35	27	49	18	60	54	77	83	41	02
							i			Size-1

Location of the element

$$Loc(a[i])=Loc(a[0])+i$$

2D-Array

Two-dimensional arrays are composed of n rows and m columns.

$$A[n][m]=\begin{pmatrix} a_{00} & a_{01} & a_{02}\ldots\ldots a_{0\ m-1} \\ a_{10} & a_{11} & a_{12}\ldots\ldots a_{1\ m-1} \\ a_{20} & a_{21} & a_{22}\ldots\ldots a_{2\ m-1} \\ \ldots\ldots\ldots\ldots\ldots & & \\ a_{n-10} & a_{n-11} & a_{n-12}\ldots\ldots a_{n-1\ m-1} \end{pmatrix}$$

2D-Array

There are three ways to implement a 2D array

1) mapping the 2D-array to a 1D-array

$$
\begin{pmatrix}
a_{00} & a_{01} & a_{02} \cdots\cdots a_{0\ m-1} \\
a_{10} & a_{11} & a_{12} \cdots\cdots a_{1\ m-1} \\
a_{20} & a_{21} & a_{22} \cdots\cdots a_{2\ m-1} \\
\cdots\cdots\cdots\cdots & & \\
a_{n-10} & a_{n-11} & a_{n-12} \cdots\cdots a_{n-1\ m-1}
\end{pmatrix}
\xrightarrow{\text{Row major order}}
$$

a_{00}
a_{01}
…
$a_{0\ m-1}$
a_{10}
a_{11}
….
$a_{n-1\ m-1}$

2D-Array

Location mapping:

a) row-major order

$Loc(a[i][j])=Loc(a[0][0])+[i*m+j]*l$

b) column-major order

$Loc(a[i][j])=Loc(a[0][0])+[j*n+i]*l$

2D-Array

An 3D-Array:

int a[m_1][m_2][m_3]

Location mapping

Loc(a[i][j][k])=Loc(a[0][0][0])+i*m_2*m_3+j*m_3+k

Special Matrix

A square matrix has the same number of rows and columns.

Some special forms of square matrix that arise frequently are:

- Diagonal. M(i,j)=0 for i!=j;
- Tridiagonal. M(i,j)=0 for|i-j|>1;
- Lower triangular. M(i,j)=0 for i<j;
- Upper triangular. M(i,j)=0 for i>j;
- Symmetric.M(i,j)=M(j,i);

Special Matrix

For example:

$$\begin{matrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{matrix} \qquad \begin{matrix} 2 & 1 & 0 & 0 \\ 3 & 1 & 3 & 0 \\ 0 & 5 & 2 & 7 \\ 0 & 0 & 9 & 0 \end{matrix} \qquad \begin{matrix} 2 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 4 & 2 & 7 & 0 \end{matrix}$$

(a)Diagonal (b) Tridiagonal © Lower Triangular

$$\begin{matrix} 2 & 1 & 3 & 0 \\ 0 & 1 & 3 & 8 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 2 & 4 & 6 & 0 \\ 4 & 1 & 9 & 5 \\ 6 & 9 & 4 & 7 \\ 0 & 5 & 7 & 0 \end{matrix}$$

(d)Upper Triangular (e)Symmetric

Special Matrix

1)Lower Triangular

$$\begin{pmatrix} a_{11} \\ a_{21}\ a_{22} \\ a_{31}\ a_{32}\ a_{33} \\ \cdots\cdots \\ a_{n1}\ a_{n2}\ \cdots\cdots\cdots a_{nn} \end{pmatrix}$$

Location mapping in row-major order:

$$Loc(a(i,j))=Loc(a(1,1))+[(1+2+3+\ldots\ldots+i-1)+(j-1)]*l$$

$$=Loc(a(1,1))+(i*(i-1)/2+j-1)*l$$

Special Matrix

2)Upper Triangular

$$\begin{pmatrix} a_{11} & a_{12} & \cdots\cdots\cdots & a_{1n} \\ & a_{22} & \cdots\cdots\cdots & a_{2n} \\ & & \cdots\cdots\cdots\cdots & \\ & & & a_{nn} \end{pmatrix}$$

Location mapping in row-major order:

$$Loc(a(i,j))=Loc(a(1,1))+[\sum_{K=1}^{i-1}(n-k+1)+j-i]*l$$

Special Matrix

3) Tridiagonal

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & a_{32} & a_{33} & a_{34} & \\ \cdots\cdots & & & & \\ & & & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

Location mapping in row-major order:

Loc(a(i,j))=Loc(a(1,1))+[(i-1)*3-1+(j-i+1)]*l

Sparse Matrices

1.Definition:

An m*n matrix is said to be sparse if "many" of its elements are zero.

number of zero elements>>number of non-zero elements

Sparse Matrices

An example of sparse matrix:

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 6 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 \end{pmatrix}$$

Sparse Matrices

2.Array representation

- The nonzero entries of an sparse matrix may be mapped into a 1D array in row major order.
- The structure of each element is:

row	col	value

Sparse Matrices

For example :

$$\begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 6 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 \\ 0 & 4 & 5 & 0 & 0 & 0 \end{pmatrix} \longrightarrow$$

a:

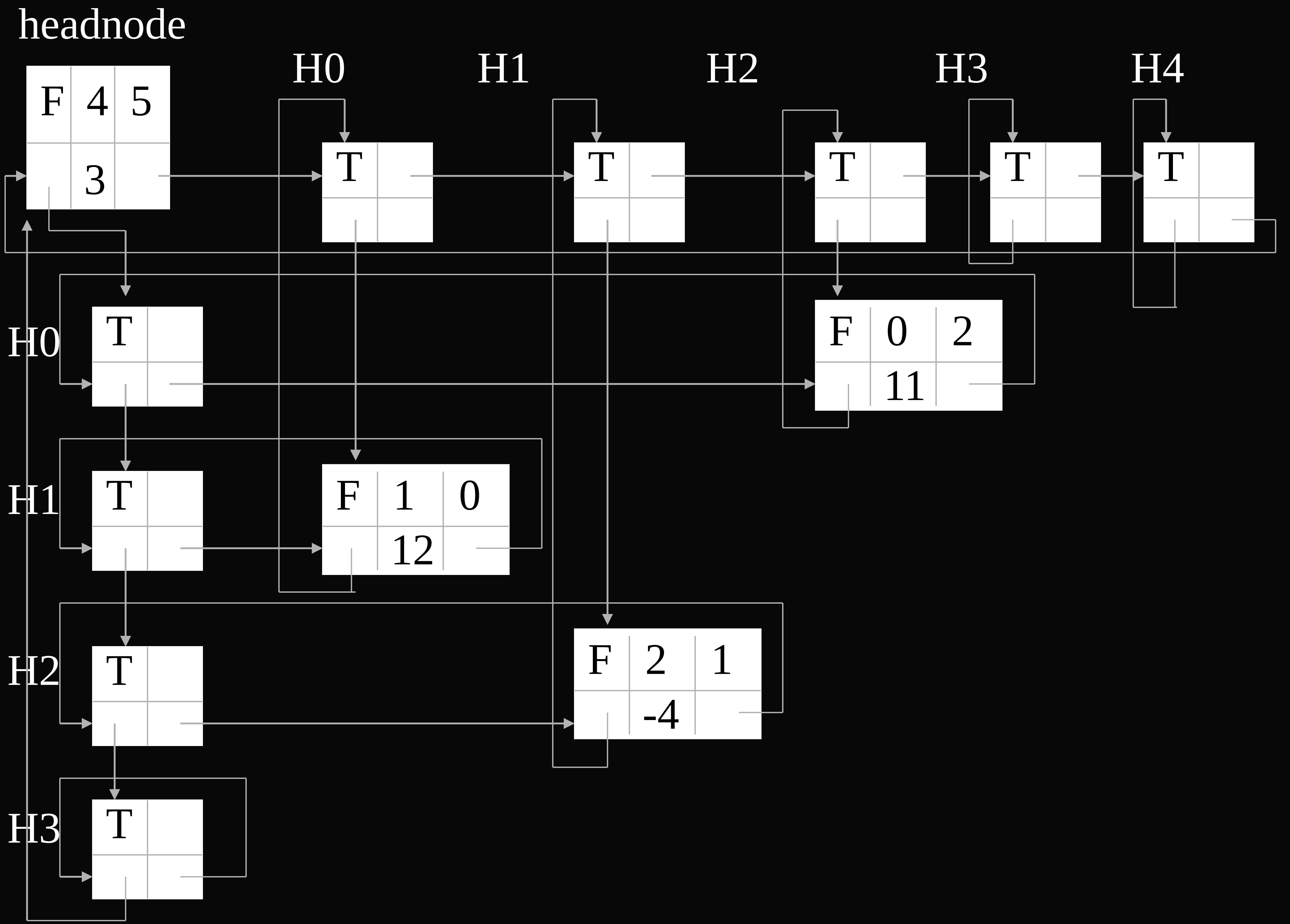
	row	col	value
0	1	4	2
1	2	2	6
2	2	5	7
	3	4	9
	4	2	4
MaxTerms-1	4	3	5

Sparse Matrices

3. Linked Representation

例子：

$$\begin{array}{cc} & \text{五 列} \\ \text{四行} & \begin{pmatrix} 0 & 0 & 11 & 0 & 0 \\ 12 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$


headnode
F
4
5
3
H0
H1
H2
H3
H4
T
T
T
T
T
H0
T
F
0
2
11
H1
T
F
1
0
12
H2
T
F
2
1
-4
H3
T

习题：

设有一个n*n的对称矩阵A，如下图(a)所示。为了节约存储，可以只存对角线及对角线以上的元素，或者只存对角线或对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。我们把它们按行存放于一个一维数组B中，如图(b)和图(c)所示。并称之为对称矩阵A的压缩存储方式。试问：

1）存放对称矩阵A上三角部分或下三角部分的一维数组B有多少元素？

2）若在一维数组B中从0号位置开始存放，则如图(a)所示的对称矩阵中的任一元素a_{ij}在只存上三角部分的情形下(图(b))应存于一维数组的什么下标位置？给出计算公式。

3）若在一维数组B中从0号位置开始存放，则如图(a)所示的对称矩阵中的任一元素a_{ij}在只存下三角部分的情况下*(图(c))应存于一维数组的什么下标位置？给出计算公式。

$$\begin{pmatrix} a_{00} & a_{01} & \ldots a_{0\,n-1} \\ a_{10} & a_{11} & \ldots a_{1\,n-1} \\ & \cdots\cdots\cdots\cdots & \\ a_{n-10} & a_{n-11} & \ldots a_{n-1n-1} \end{pmatrix} \qquad \begin{pmatrix} a_{00} & a_{01} & \ldots a_{0n-1} \\ & a_{11} & \ldots a_{1n-1} \\ & \cdots\cdots\cdots\cdots & \\ & & a_{n-1n-1} \end{pmatrix} \qquad \begin{pmatrix} a_{00} & & \\ a_{10} & a_{11} & \\ & \cdots\cdots\cdots & \\ a_{n-10} & a_{n-11} & \ldots a_{n-1n-1} \end{pmatrix}$$

(a) (b) (c)

答案：

1) $1+2+3+\ldots+n = \frac{1}{2}*(1+n)*n$

2) $loc(A[i,j]) = loc(B[0]) + (n+n-1+\ldots+n-i+2 + j-i)$

$$\begin{cases} t = \frac{1}{2}*(2*n-i+1)*i + j-i & i<=j \\ t = \frac{1}{2}*(2*n-j+1)*j + i-j & i>j \end{cases}$$

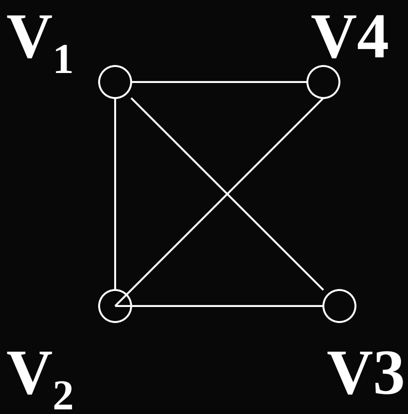
$$\begin{cases} t = \frac{1}{2}*(2*n-i+2)*(i-1)+j-i \\ t = \frac{1}{2}*(2*n-j+2)*(j-1)+j-i \end{cases}$$

3) $loc(A[i,j] = loc(B[0]) + (1+2+3+\ldots+i-1+j-1)$

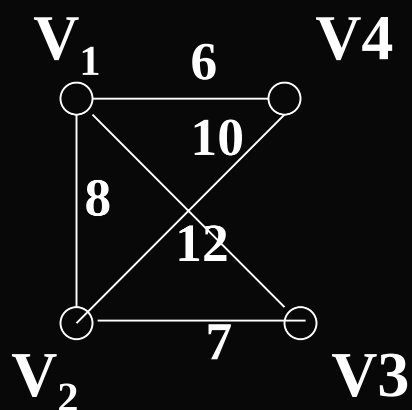
$$\begin{cases} t = \frac{1}{2}*i*(i+1) + j & i>=j \\ t = \frac{1}{2}*j*(j+1) + i & i<j \end{cases}$$

$$\begin{cases} t = \frac{1}{2}*i*(i-1)+j-1 \\ t = \frac{1}{2}*j*(j-1)+i-1 \end{cases}$$

对角线元素的地址：$t = i*(i+3)/2$



$$A(i,j)=\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$



$$A(i,j)=\begin{pmatrix} 0 & 8 & 12 & 6 \\ 8 & 0 & 7 & 10 \\ 12 & 7 & 0 & \infty \\ 6 & 10 & \infty & 0 \end{pmatrix}$$

第4章　树

1.二叉树的定义、性质

2.满二叉树与完全二叉树的概念

3.二叉树的机内存储：

数组表示（完全二叉树）、左---右拉链表示、cursor

4.先序、中序、后序遍历

递归

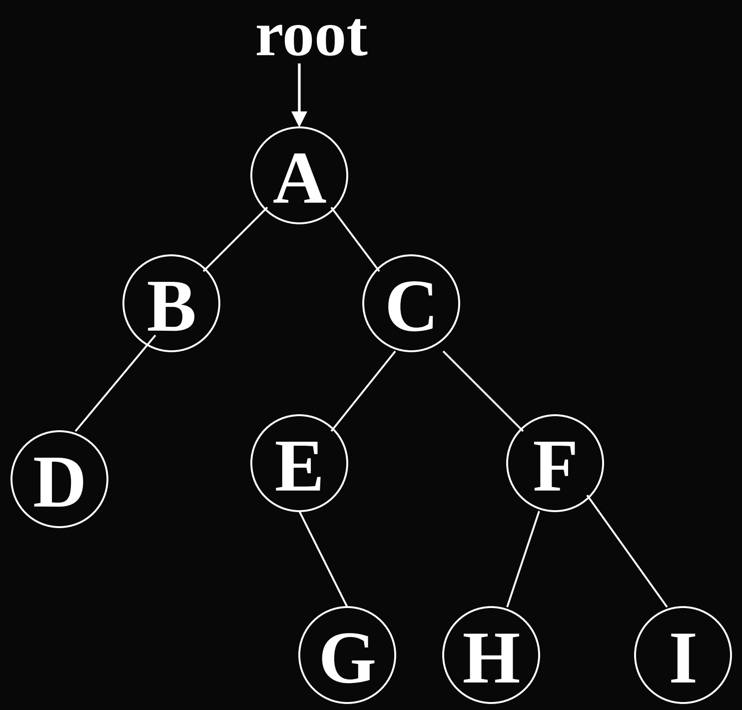
非递归

层次遍历-----用到队列

例1. 第4章中用非递归实现中序,后序遍历

Inorder, Postorder non-recursive algorithm

- Inorder non-recursive algorithm



```
template<class T>
    void InOrder(BinaryNode<T>* t)
    { if(t){  InOrder(t→Left);
                visit(t);
                InOrder(t→Right);
              }
    }
```

Inorder non-recursive algorithm

```
void Inorder(BinaryNode <T> * t)
{  Stack<BinaryNode<T>*> s(10);
    BinaryNode<T> * p = t;
    for (  ;  ;  )
    { 1) while(p!=NULL)
          { s.push(p);   p = p->Left; }
      2)  if (!s.IsEmpty( ))
          {  p = s.pop( );
             cout << p->element;
             p = p->Right;
          }
          else return;
    }
}
```

5. 利用先序、中序可唯一构造一棵树

先序：ABDCEGFHI

中序：DBAEGCHFI

利用中序、后序可唯一构造一棵树

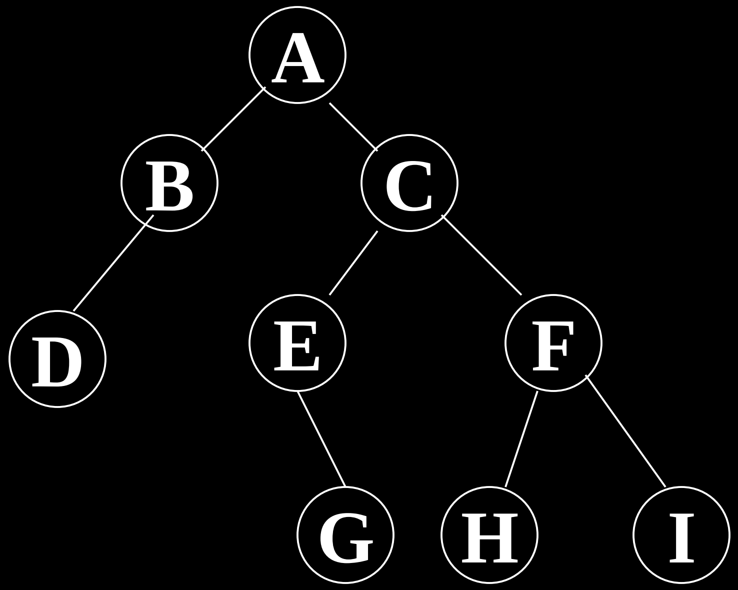
手工画出一棵树

利用算法生成一棵树

Create BinaryTree recursive algorithm

preorder:ABDCEGFHI

inorder: DBAEGCHFI



*6. 利用广义表表示来构造一棵树

7. 应用

树的机内表示：

广义表表示、双亲表示、左子女---右兄弟表示

树的存储方式：三种

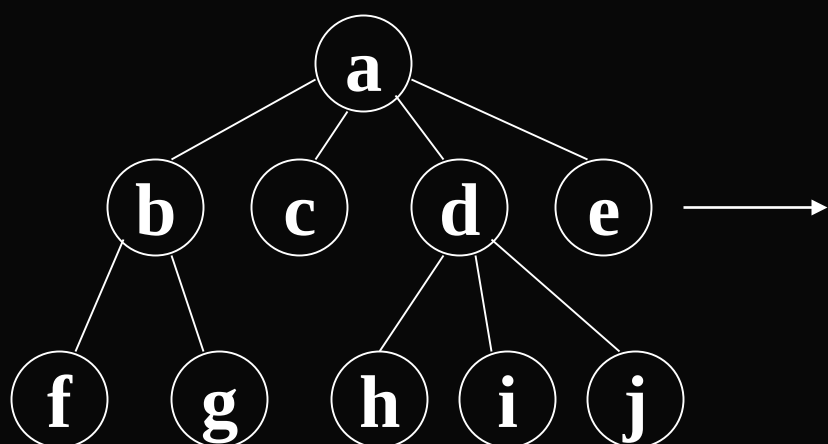
•广义表表示：a(b(f,g),c,d(h,i,j),e)

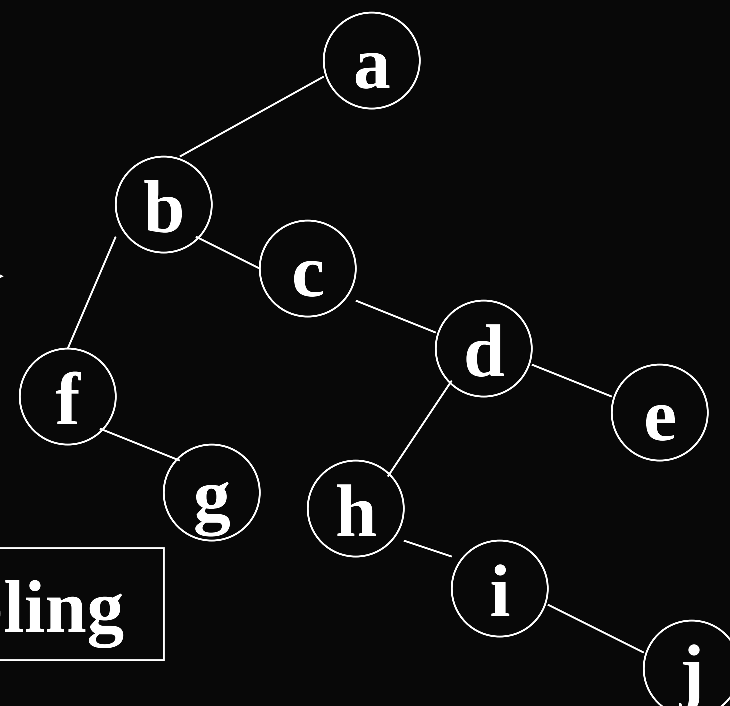
•双亲表示法

	a	b	f	g	……….		
	0	1	2	2			

•左子女—右兄弟表示法

1) Take a tree as a binary tree





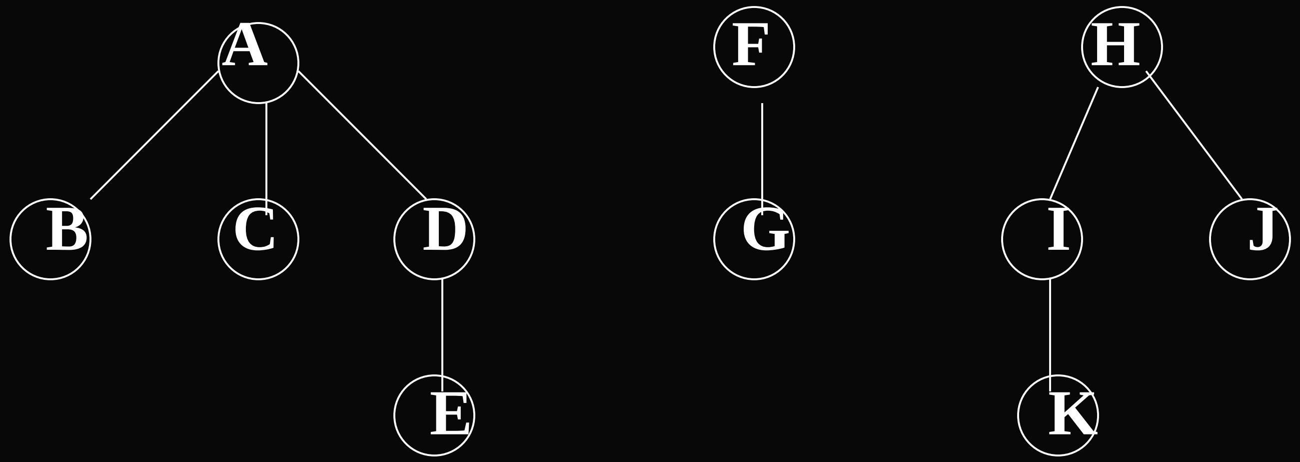
firstchild	data	nextsibling

```
class TreeNode:
    T data;
    TreeNode *firstchild, *nextsibling;
 class Tree:
     TreeNode * root,  *current;
```

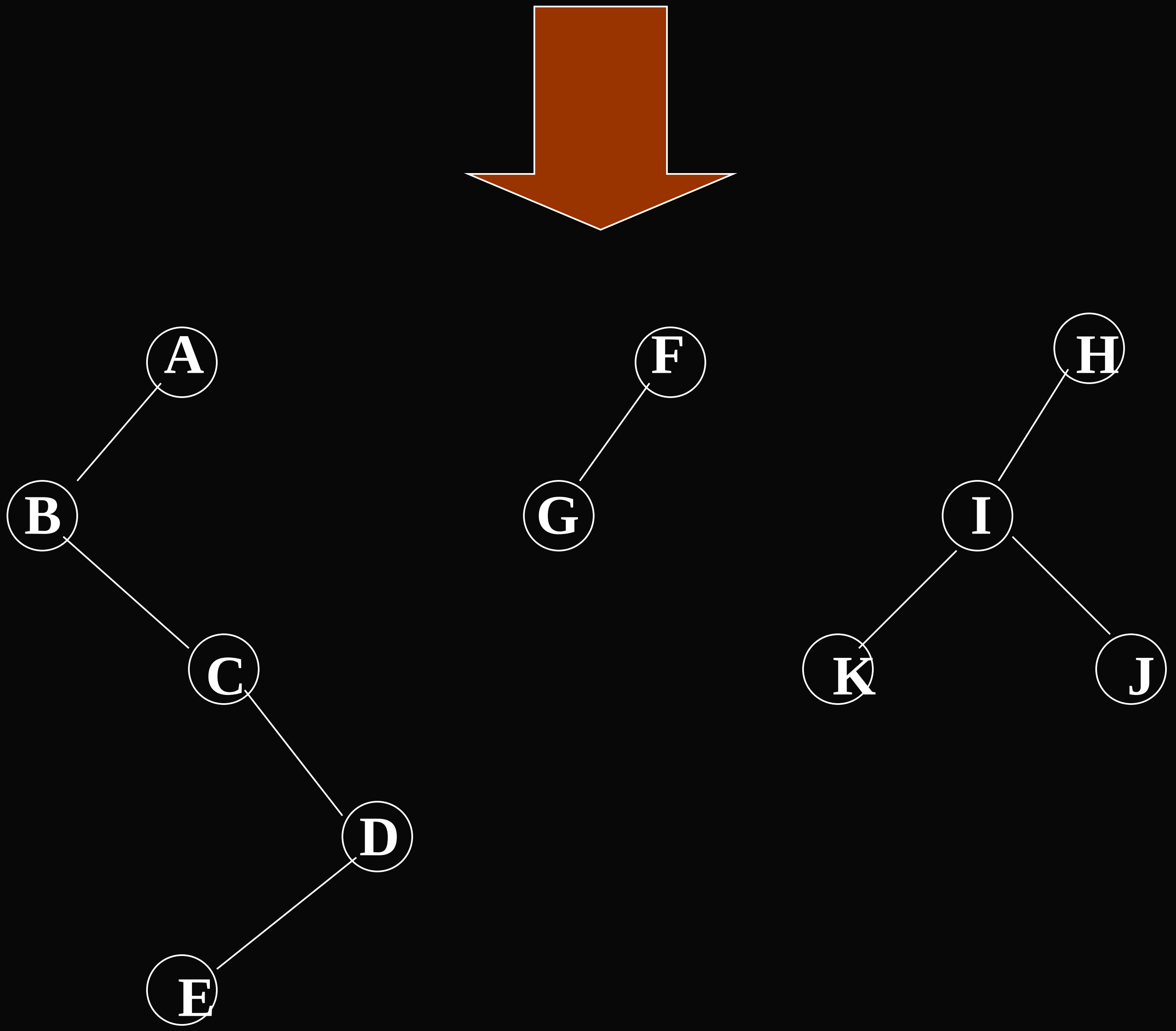
树-----二叉树的转换

Forest ⟷ Binary tree

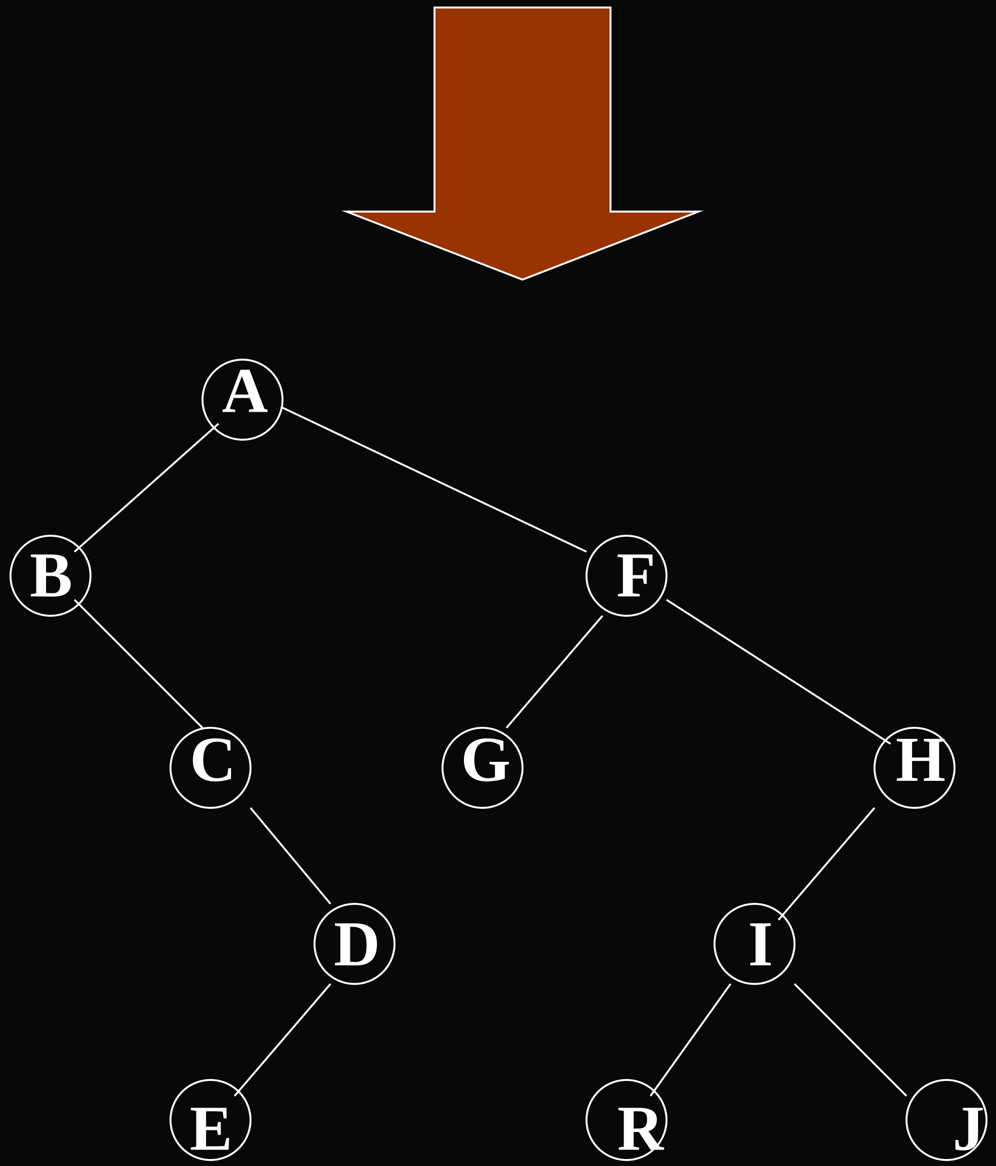
- **Forest ⟶ Binary tree**



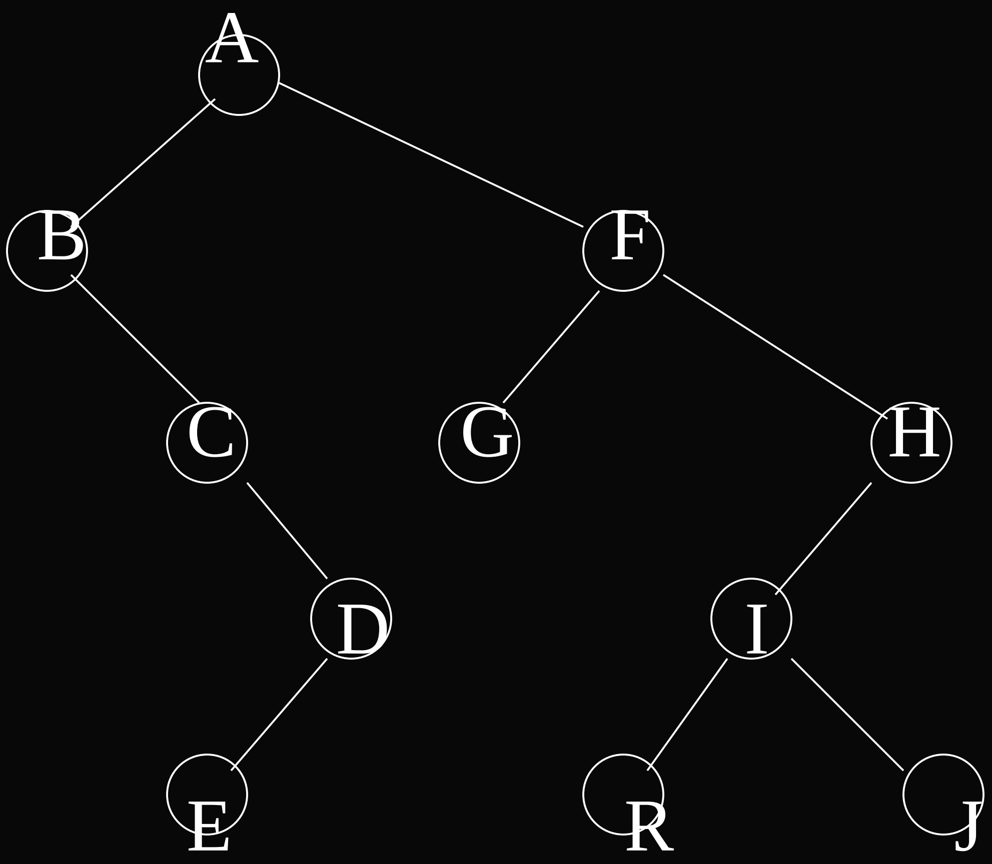
每棵树转为二叉树



把每棵二叉树根用右链相连



- **Binary tree ⟶ Forest**



树与森林的遍历

树的遍历：深度优先遍历，广度优先遍历

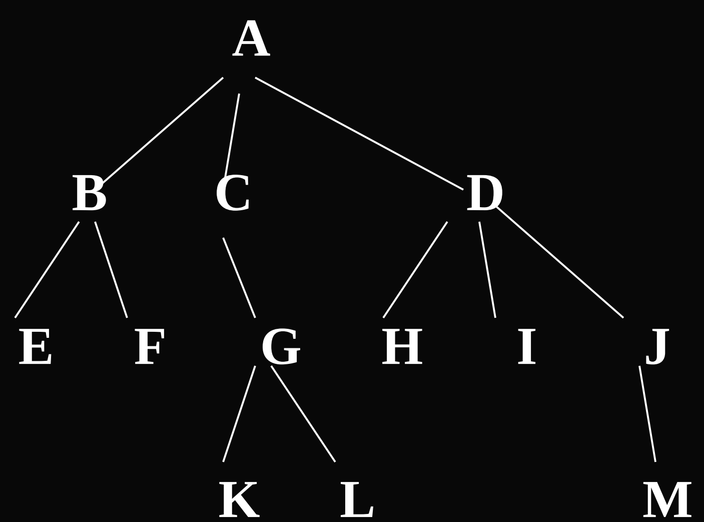
- 深度优先遍历

先序次序遍历（先序）

访问树的根 → 按先序遍历根的第一棵子树，第二棵子树，……等。

后序次序遍历（后序）

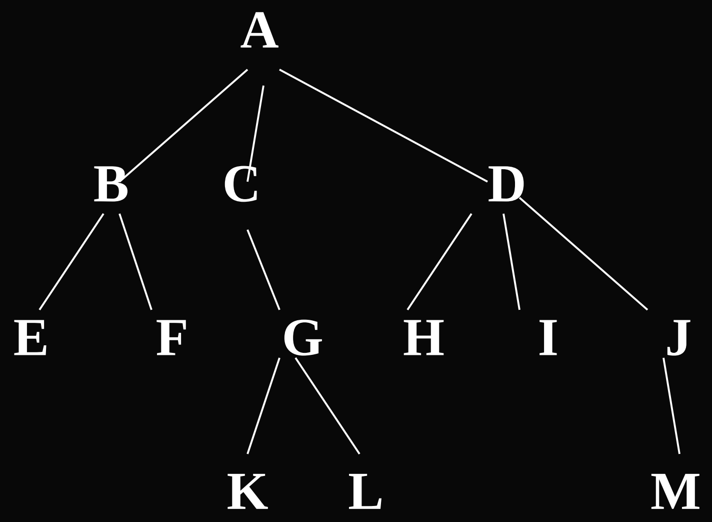
按后序遍历根的第一棵子树，第二棵子树，……等 → 访问树的根。



先根：ABEFCGKLDHIJM与对应的二叉树的先序一致

后根：EFBKLGCHIMJDA与对应的二叉树的中序一致

- 广度优先遍历



分层访问：**ABCDEFGHIJKLM**

森林的遍历

深度优先遍历

* 先根次序遍历
  - 访问F的第一棵树的根
  - 按先根遍历第一棵树的子树森林
  - 按先根遍历其它树组成的森林

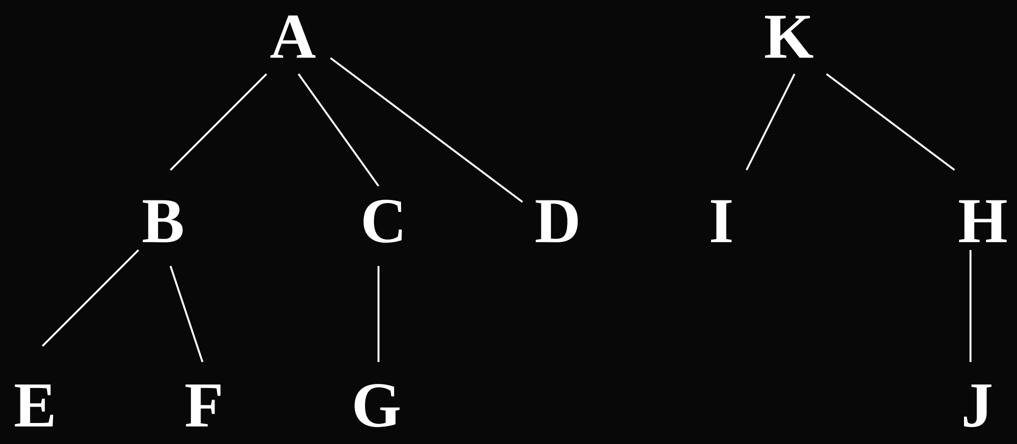
  二叉树的先序

* 中根次序遍历
  - 按中根遍历第一棵树的子树森林
  - 访问F的第一棵树的根
  - 按中根遍历其它树组成的森林

  二叉树的中序

* 后根次序遍历
  - 按后根遍历第一棵树的子树森林
  - 按后根遍历其它树组成的森林
  - 访问F的第一棵树的根

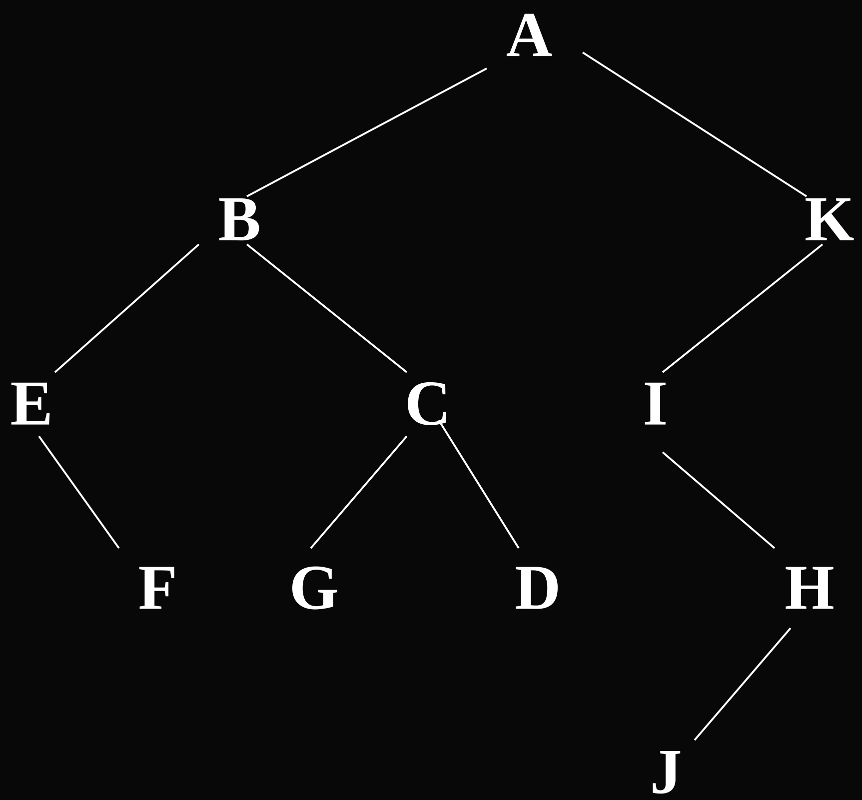
  二叉树的后序



先根：ABEFCGDKIHJ

中根：EFBGCDAIJHK

后根：FEGDCBJHIKA



广度优先遍历（层次遍历）

AKBCDIHEFGJ

补充:

线索树

Thread Tree

1.Purpose:

2. Thread Tree Representation

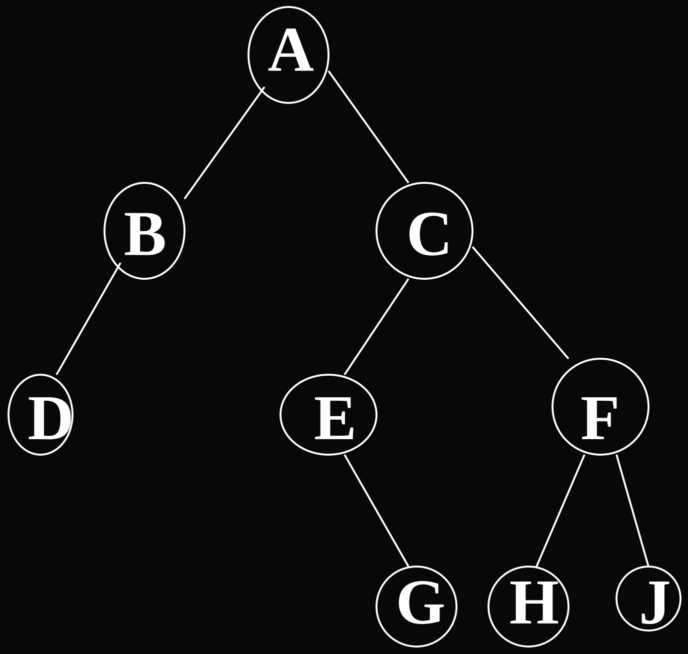
left Thread Tree and right Thread Tree

3.Thread Tree class

1.Purpose:

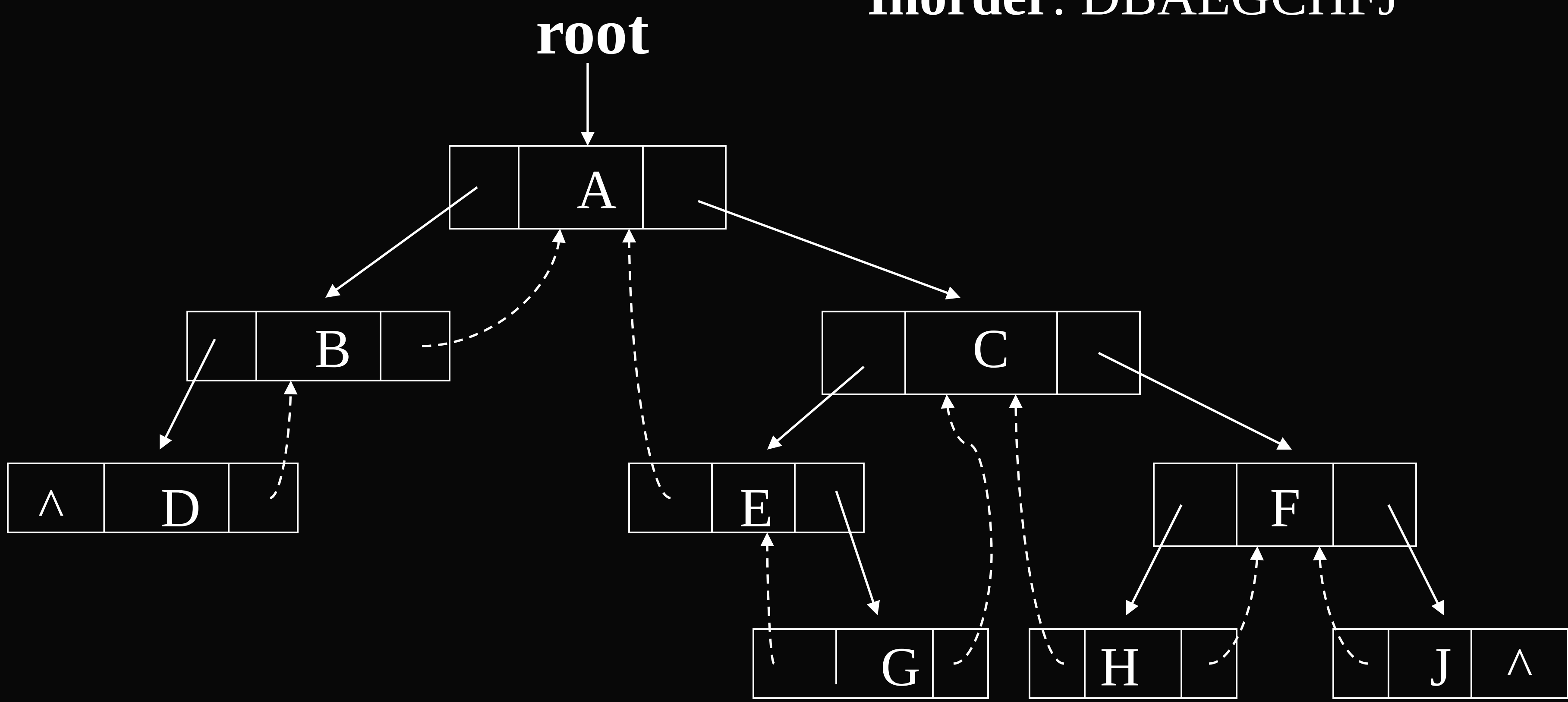
Thread Tree

Example:



Thread Tree

Inorder: DBAEGCHFJ



2. 机内如何存储

一个结点增加两个标记域：

leftchild	leftthread	data	rightthread	rightchild

leftThread = = { 0 leftchild 指向左子女; 1 leftchild 指向前驱（某线性序列）}

rightThread = = { 0 rightchild 指向右子女; 1 rightchild 指向后继 }

Thread Tree

left threadTree

right threadTree

8. 哈夫曼树

哈夫曼树的构造

哈夫曼编码

扩充的二叉、三叉、….、t叉树

15, 3, 14, 2, 6, 9, 16, 17　构造扩充的三叉树。

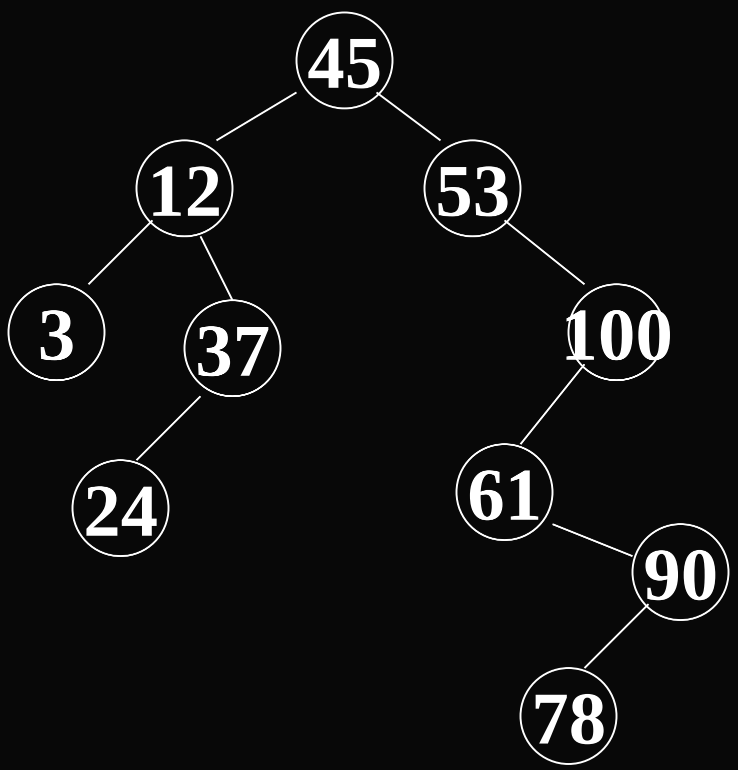
第4.1章：二叉搜索树

1.二叉搜索树的概念
2.带索引的二叉搜索树的概念
3. AVL树-----平衡的二叉搜索树
4. B-树

1.二叉搜索树的概念

二叉搜索树

Example:

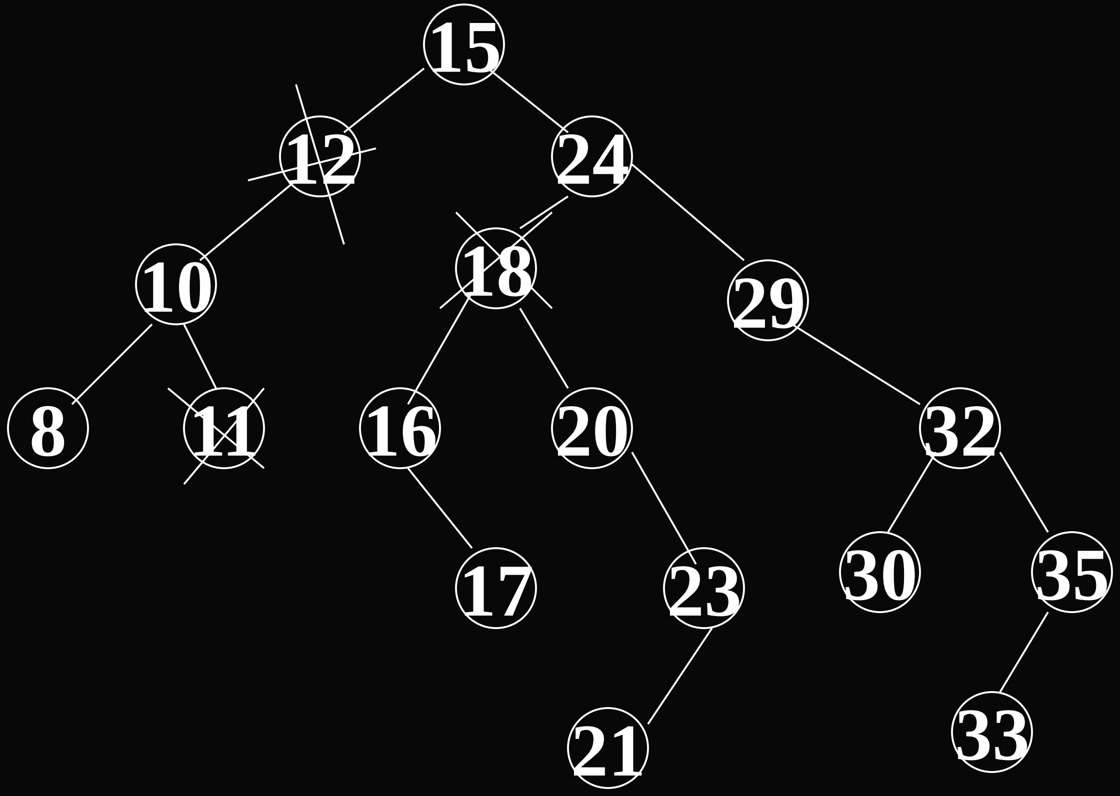


left	**element**	**right**

二叉搜索树

主要操作：

查找、插入、删除

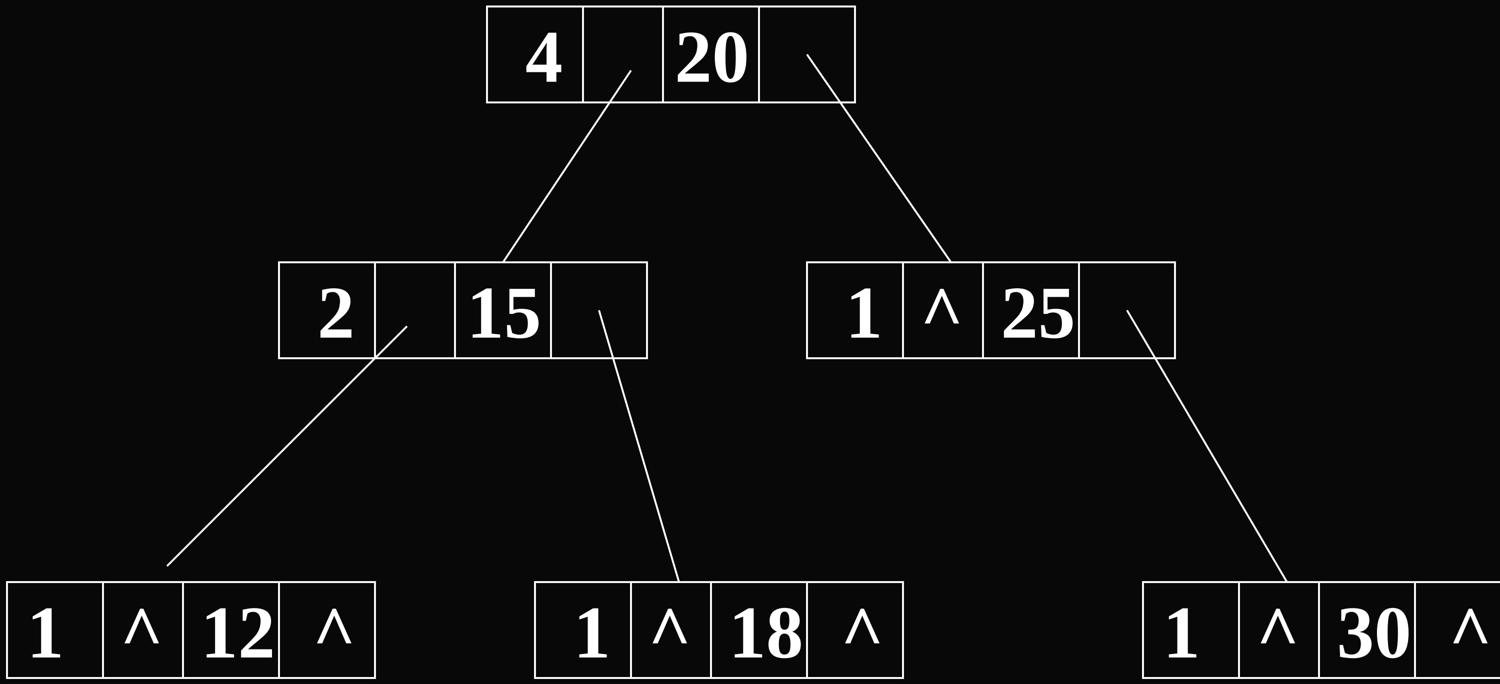


2.带索引的二叉搜索树的概念

- **An indexed binary search tree is derived from an ordinary binary search tree by adding the field leftSize to each tree node.**
- **Value in Leftsize field=number of the elements in the node's left subtree +1**

leftSize	**left**	**element**	**right**

Example:



例子：

写一递归函数实现在带索引的二叉搜索树（IndexBST)中查找第k个小的元素。

```
public Comparable findK( BinaryNode root, int k)
{
    if( root==null) return null;//空
    if( k< root. leftSize) //在左子树
        findK( root. left, k);
    else if( k>root. leftSize) //在右子树
        findK( root. right, k-root. leftSize);//注意减去
    else return root.element;
}
```

3.AVL树----平衡的二叉搜索树

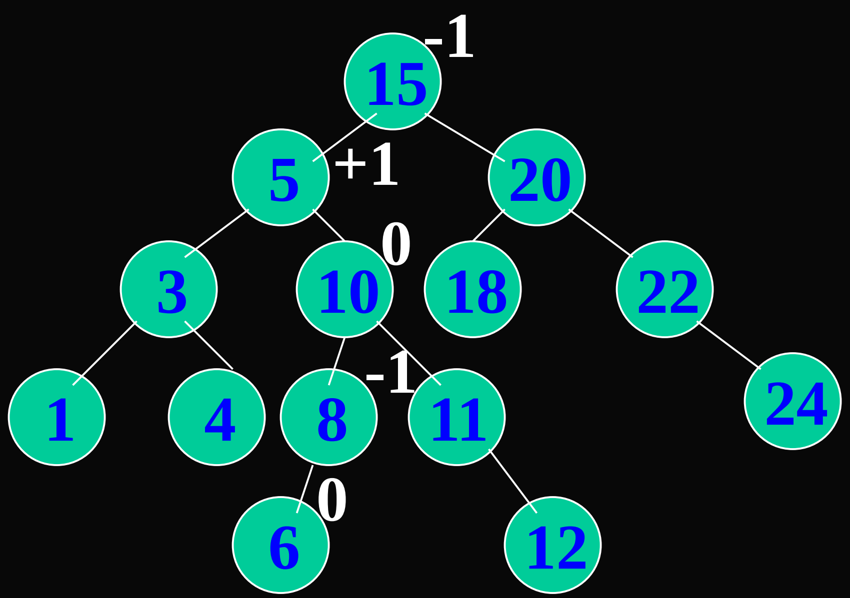
Definition of an AVL tree:

(1) is a binary search tree

(2) Every node satisfies

$|h_L - h_R| <= 1$ where h_L and h_R are the heights of T_L(left subtree) and T_R(right subtree),respectively.

例子



AVL Tree

- Height of an tree:

  the longest path from the root to each leaf node

- Balance factor *bf(x)* of a node x :

  height of right subtree of x – height of left subtree of x

Each node:

Left	data	Right	balance(height)

AVL Tree

The height of an AVL tree with n elements is $O(\log_2 n)$, so an n-element AVL search tree can be searched in $O(\log_2 n)$ time.

AVL Tree

- 插入

  左外侧， 右外侧-----一次旋转

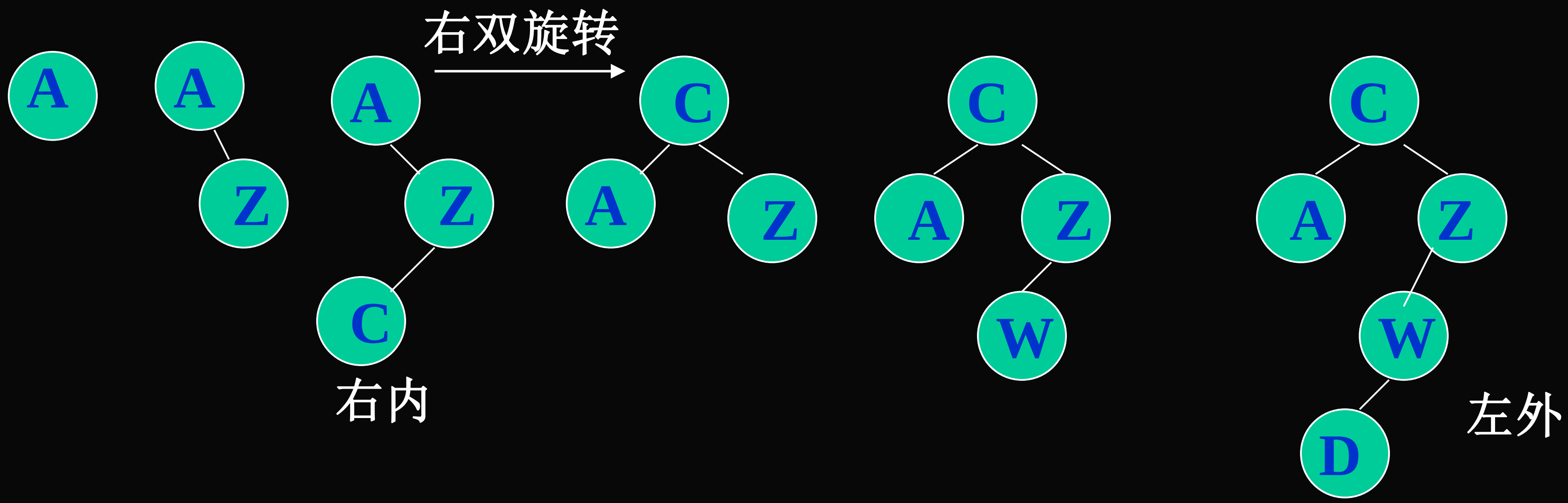
  左内侧，右内侧------二次旋转

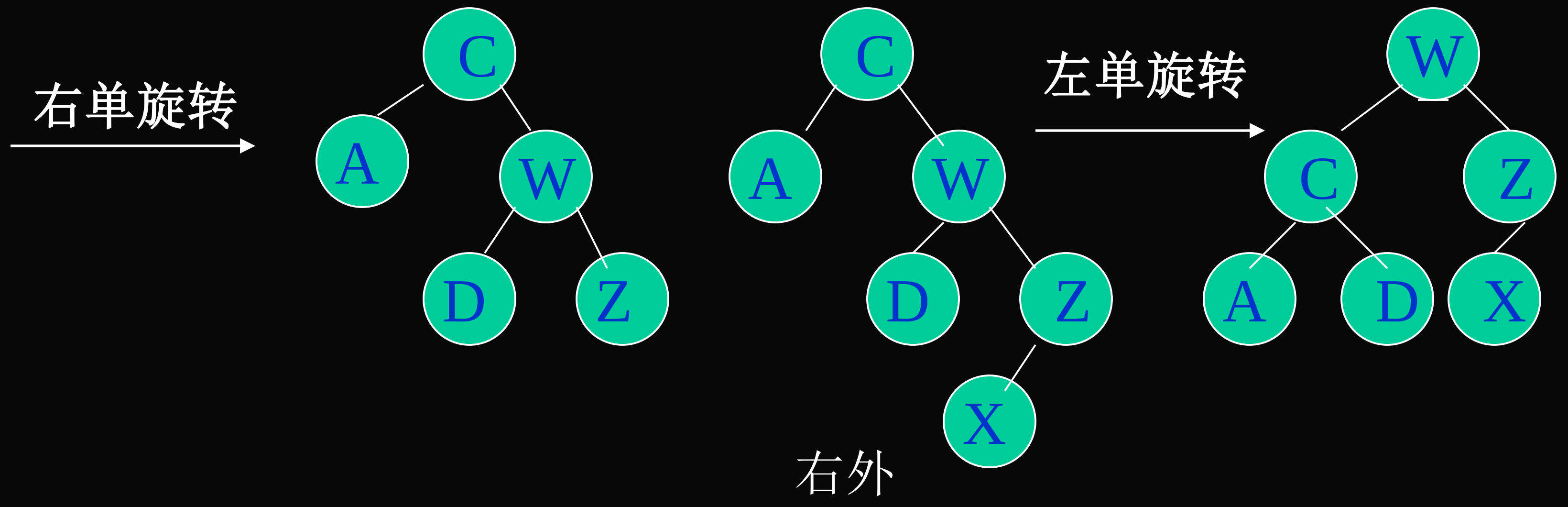
AVL树的插入:

1. 首先要正确地插入
2. 找到有可能发生的最小不平衡子树
3. 判别插入在不平衡子树的外侧还是内侧
4. 根据3的判别结果,再进行单旋还是双旋

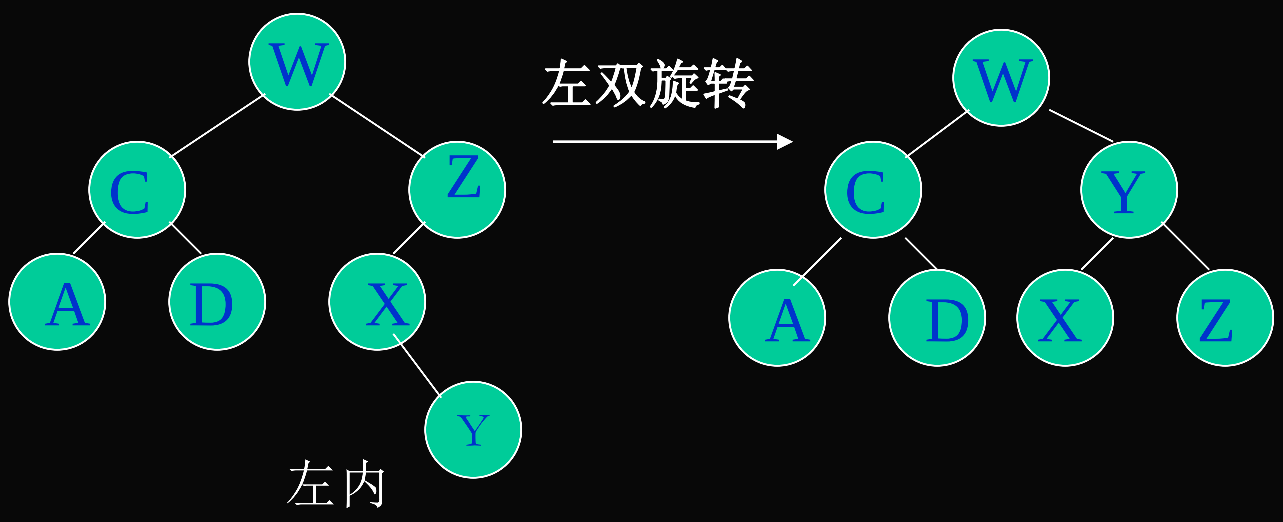
从空的AVL树建树的算法。一个例子：

7个关键码发生四种转动　A，Z，C，W，D，X，Y





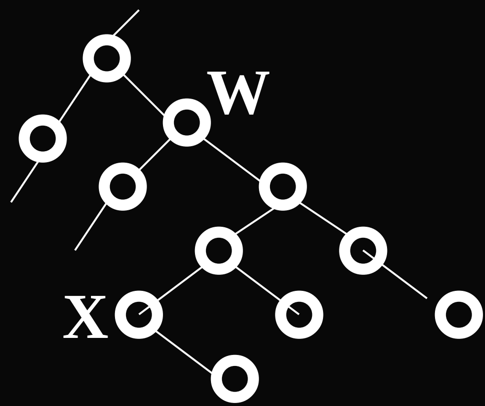
右单旋转
C
A
W
D
Z
C
A
W
D
Z
X
右外
左单旋转
W
C
Z
A
D
X



W
C
Z
A
D
X
Y
左内
左双旋转
W
C
Y
A
D
X
Z

- AVL树的删除：

方法： 与二叉搜索树的删除方法一样。



假设被删除结点为W, 它的中序后继为X, 则用X代替W,并删除X. 所不同的是:删除X后,以X为根的子树高度减1,这一高度变化可能影响到从X到根结点上每个结点的平衡因子,因此要进行一系列调整。

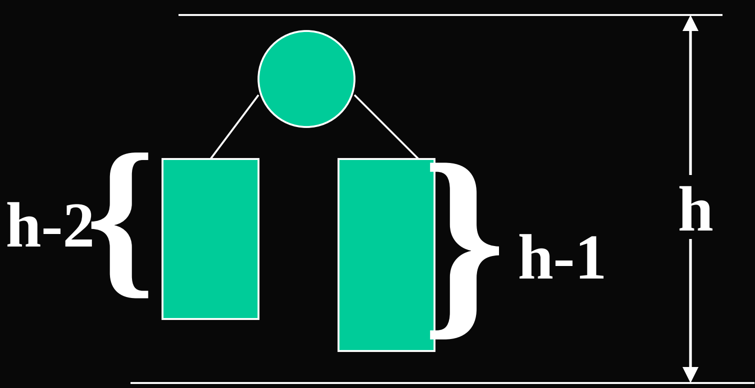
- **AVL树的算法分析**

具有n个结点的平衡二叉树（AVL），进行一次插入或删除的时间最坏情况≦O（$\log_2 n$)

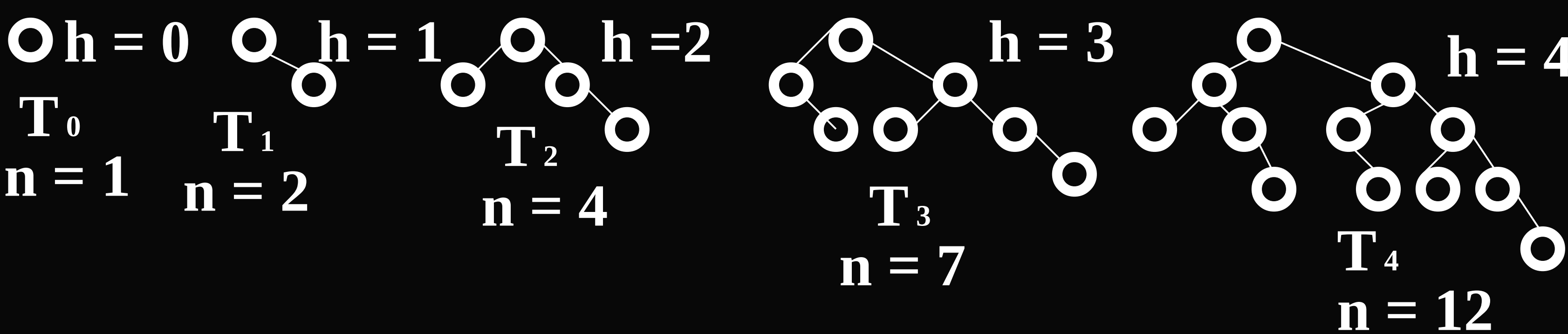
证明：实际上要考虑n个结点的平衡二叉树的最大高度

$$\leqq (3/2)\log_2 (n+1)$$

设T_h为一棵高度为h，且结点个数最少的平衡二叉树。



假设右子树高度为**h-1**
因结点个数最少,∴左子树高度
只能是**h-2**
这两棵左子树,右子树高度分别
为**h-2, h-1**,也一定是结点数最少的:



AVL Tree

例子：

对一棵空的AVL树，分别画出插入关键码为{ 16，3，7，11，9，28，18，14，15}后的AVL树。

4. B-树（外查找）

B-Trees of order m

70年 R.Bayer提出的。

Definition： A B-tree of order m is an m-way search tree. If the B-tree is not empty, the corresponding extended tree satisfies the following properties:

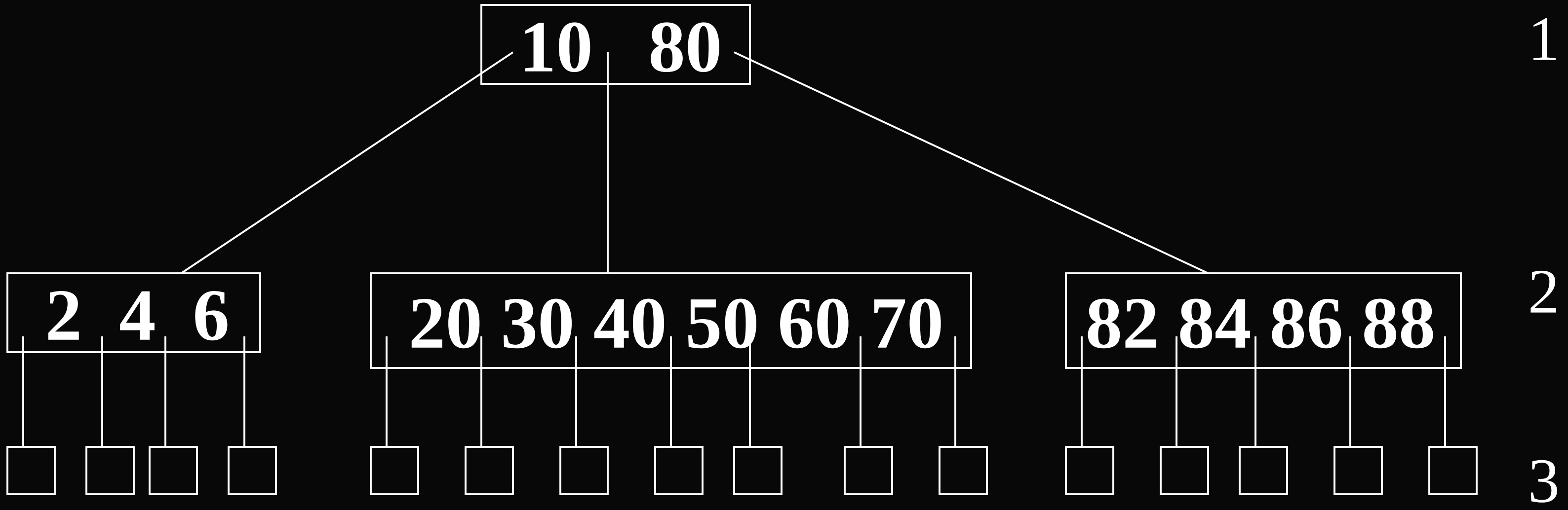
1) the root has at least two children

2) all internal nodes other than the root have at least $\lceil m/2 \rceil$ children

3) all external nodes are at the same level

B-trees

example



a B-tree of order 7

B树的插入： （注意分支数的上界**m)**

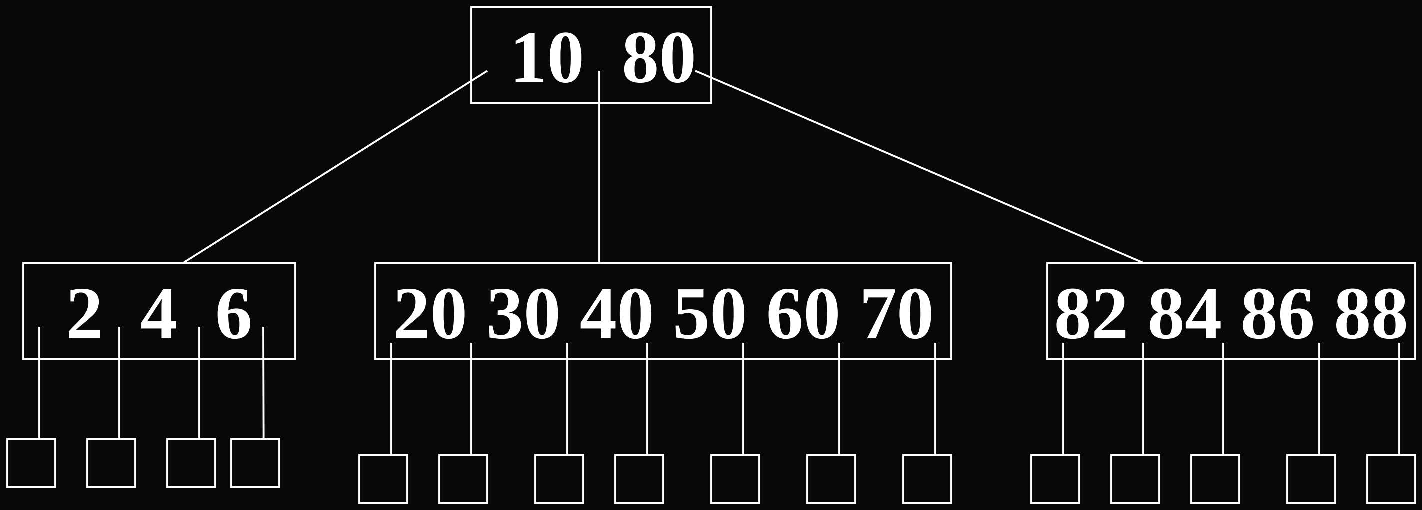
一定只发生在外部结点的上一层。

1. 能插。按序插入

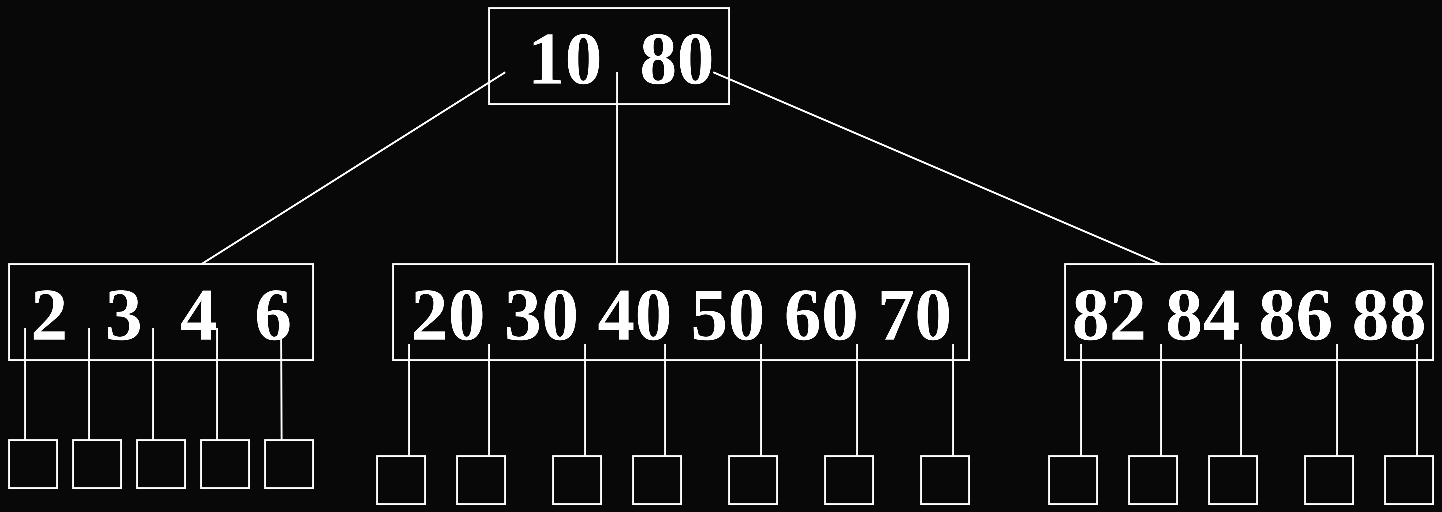
2. 不能插。将关键码按序插入后，把该结点分为两个结点，并把中间的关键码上提到父亲结点，可能引起再一次分裂，依次向上传递。可能引起树升高一层。

能插

Insert 3



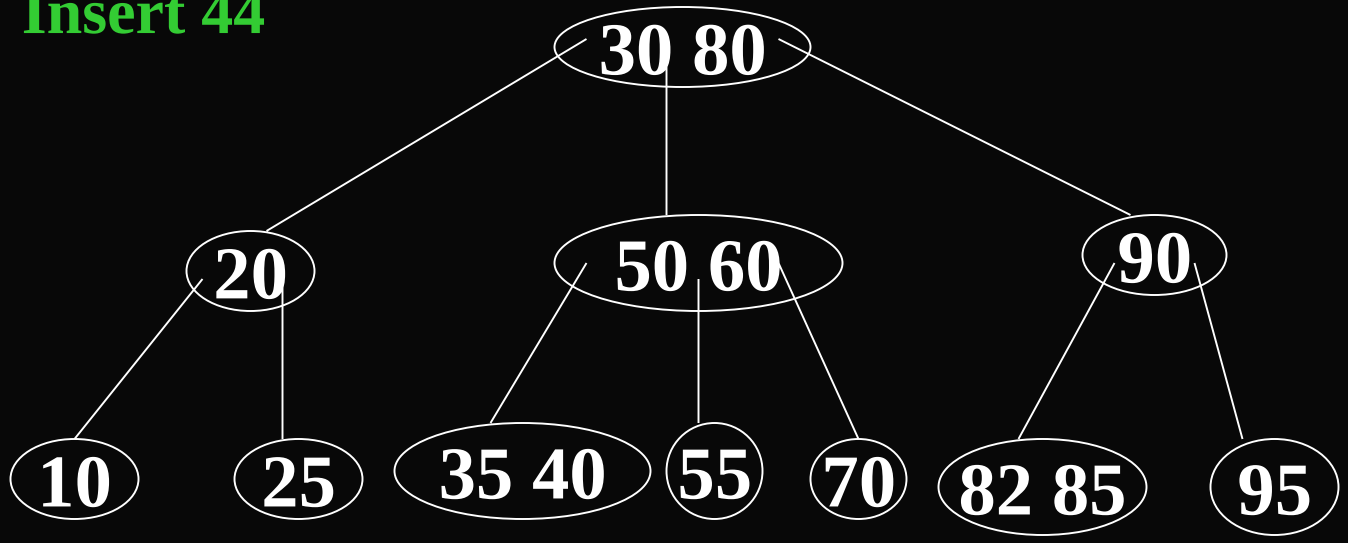
A B-Tree of order 7


10 80
2 3 4 6
20 30 40 50 60 70
82 84 86 88

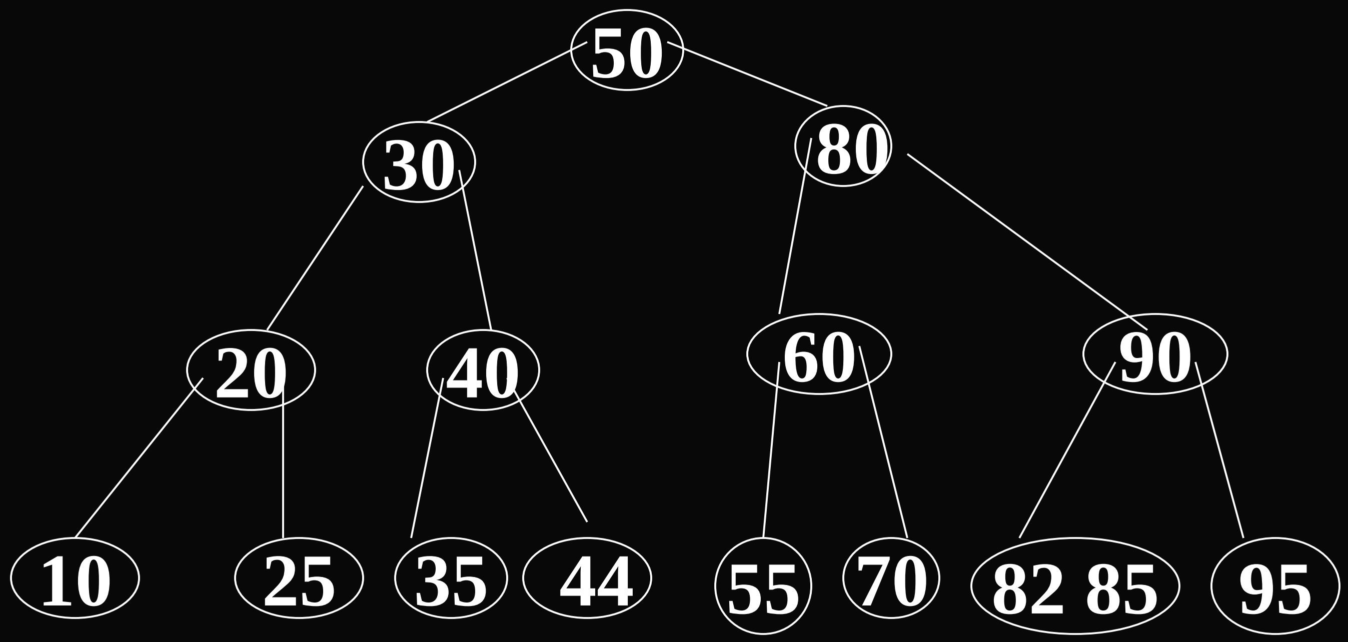
不能插

Example:

Insert 44



A B-Tree of order 3


50
30
80
20
40
60
90
10
25
35
44
55
70
82 85
95

删除：（注意分支数的下界⌈m/2⌉）

有两种情况:

1. 发生在外部结点的上一层（与插入情况一样）

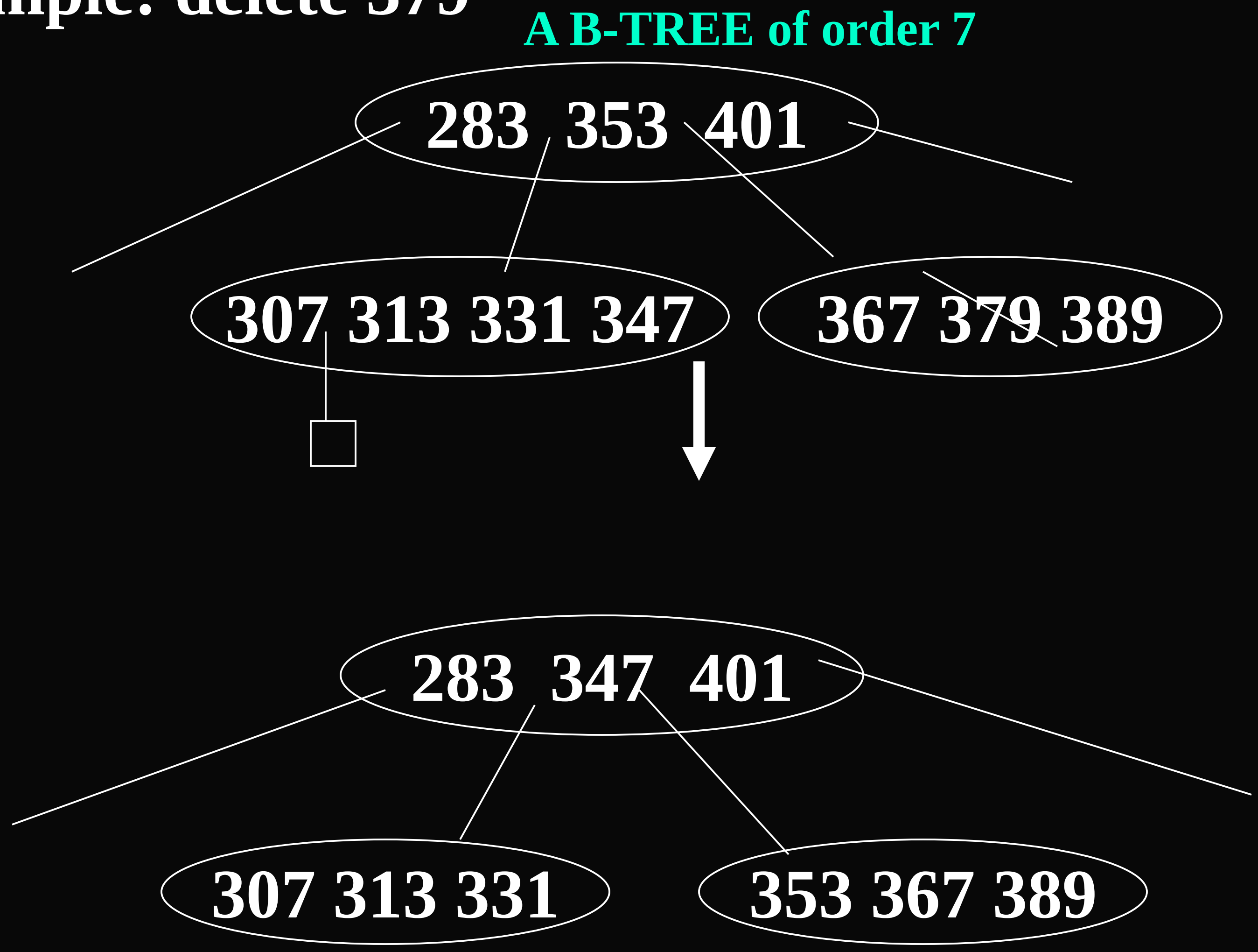
1）能删则删

2) 不能删（关键码要删除，但分支数也减少1）

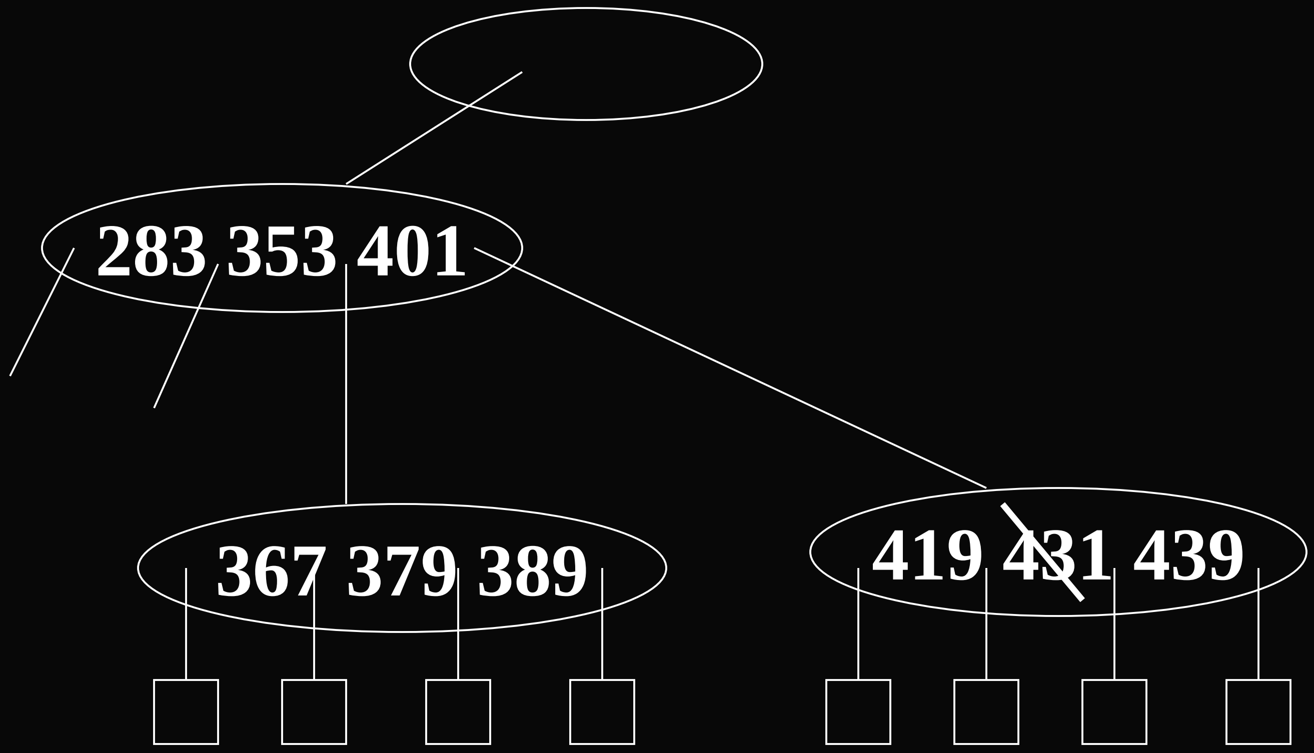
a. 能借则借，但要作关键码的适当调整;

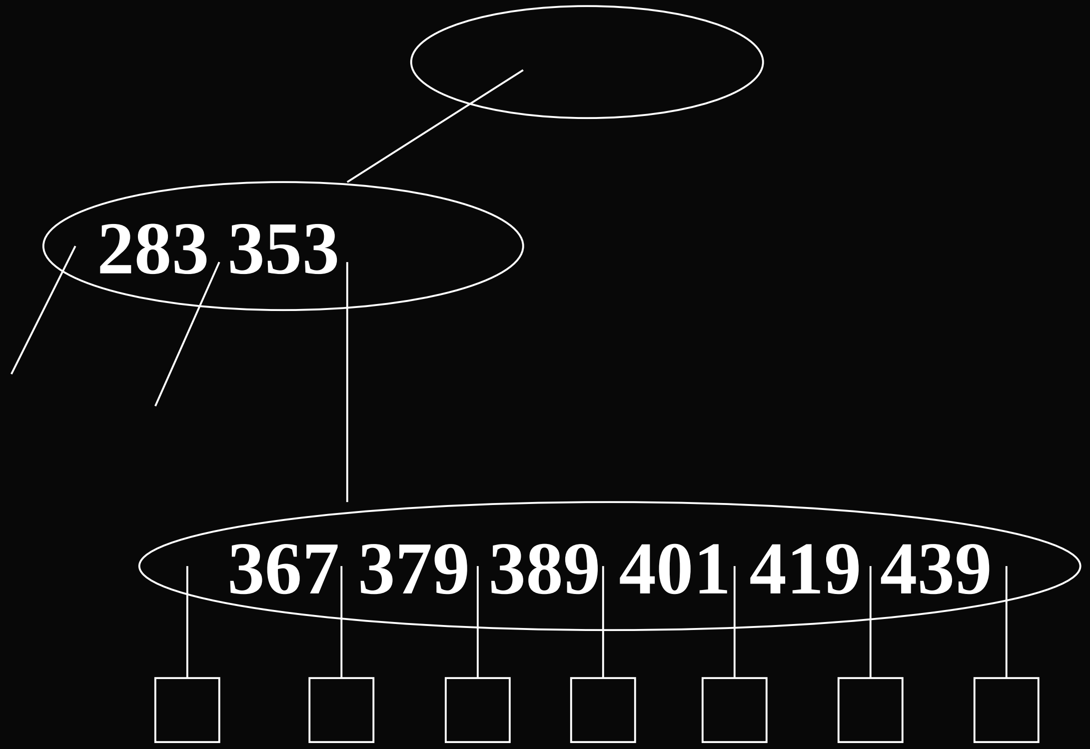
b. 不能借，则与邻近的一个结点合并，相应的父结点的一个关键码要下放，如果引起父结点的不平衡，则还是能借则借，不能借，则合并，这样会引起更上一层的不平衡，依次传递上去，可能会引起树高降低一层。

Example: delete 379



Example:a B-Tree of order 7 ,delete 431

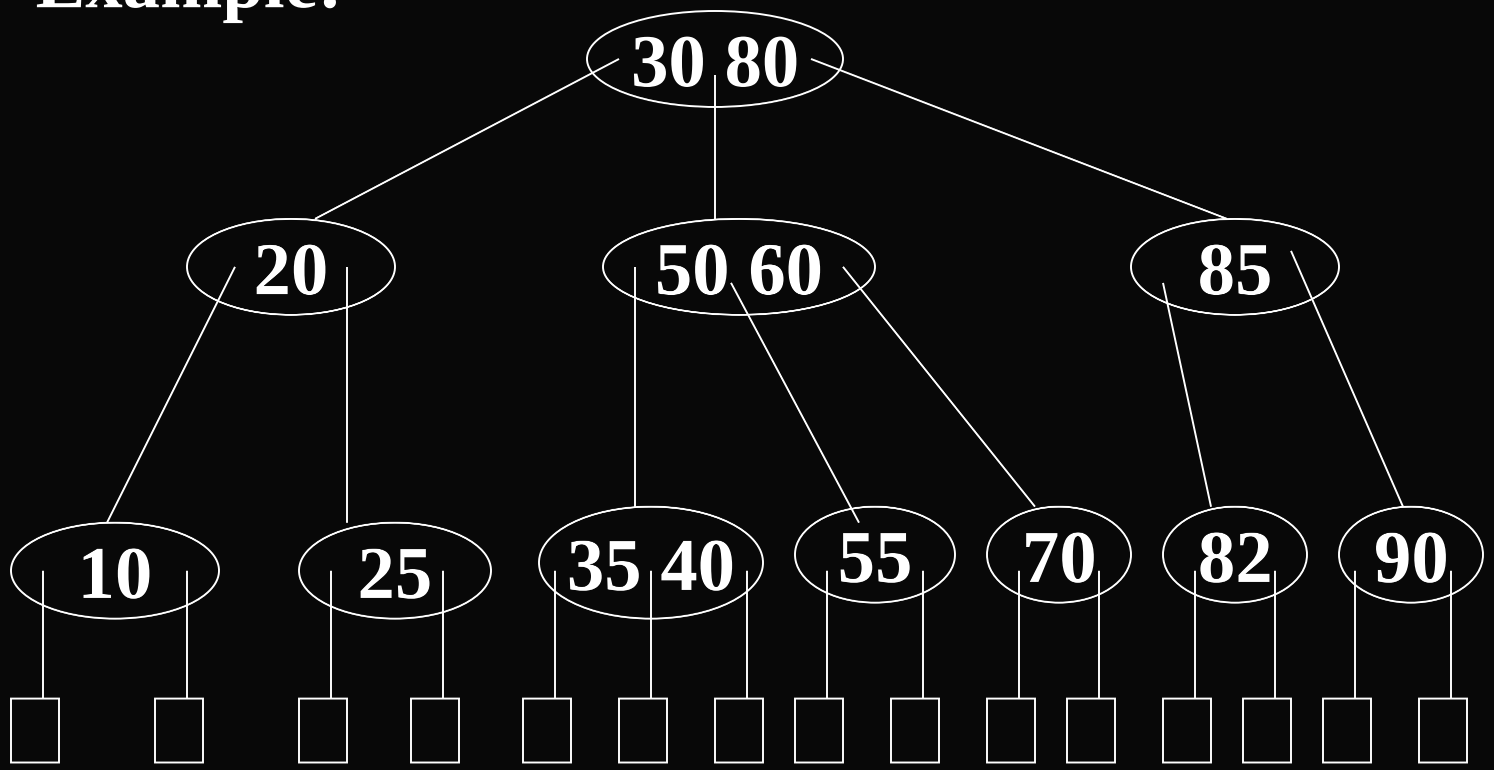



283 353
367 379 389 401 419 439

2. 删除发生在以上各层

- 删除它
- Replace it with the smallest key in the right subtree or the largest key in the left subtree
- 这样就变成1的情况

Example:



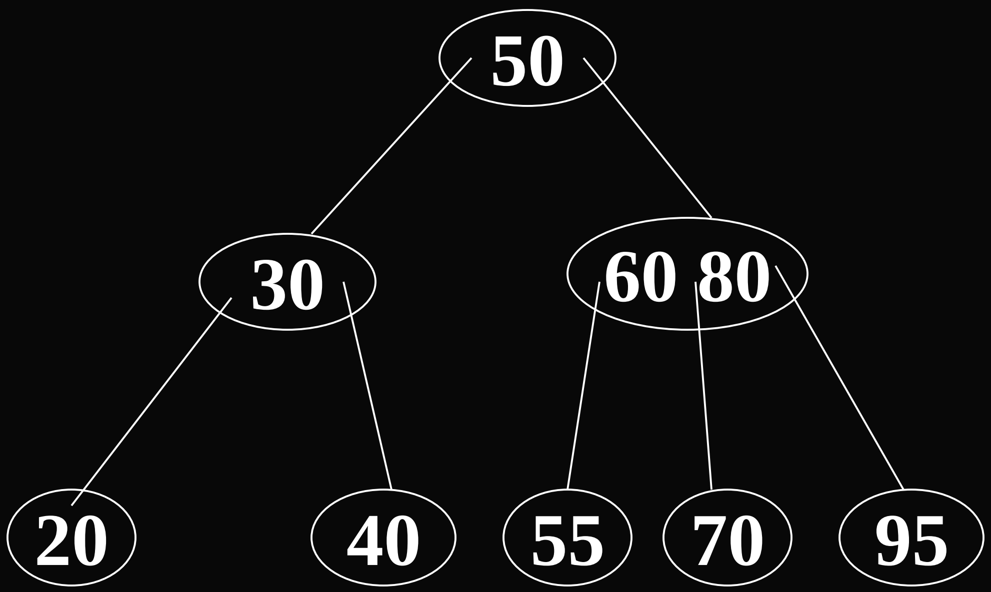
A B-TREE of order 3

Delete 80, then replace it with 82 or 70, delete 82 or 70 at last

B-tree

例子:

1. 分别 delete 50 ,40 in the following 3阶B-树.



第5章：散列

1.散列函数的选择

2.解决冲突的方法

开地址法：线性探查法

平方探查法

二次散列

链地址法

Hash Function

1. 散列函数的选择
    1. 计算简单
    2. 地址分布比较均匀

2. solve a collision

- **Open Addressing**

1) linear Probing

If hash(key)=d and the bucket is already occupied then we will examine successive buckets d+1, d+2,……m-1, 0, 1, 2, ……d-1, in the array

solve a collision

example

keys： Burke, Ekers, Broad, Blum, Attlee, Alton, Hecht, Ederly

hash(key) = ord(x) – ord(‘A’)

x为取key第一个字母在字母表中的位置。例如：

hash(Attlee) = 0

H(Burke) = 1 , H(Ekers) = 4 , H(Broad) = 1, H(Blum) = 1,
H(Attlee) = 0 , H(Hecht) = 7 , H(Alton) = 0 , H(Ederly) = 4,

设散列表长 m = 26 (0~25)

0	1	2	3	4	5	6	7	8		25
Attlee	Burke	Broad	Blum	Ekers	Alton	Ederly	Hecht	……	……	
1	1	2	3	1	6	3	1			

分析比较次数：

搜索成功的平均搜索长度

$$1/8*(1 + 1 + 2 + 3 + 1 + 1 + 6 + 3) = 18/8$$

* 搜索不成功的平均搜索长度

$$1/26*(9+8+7+6+5+4+3+2+ 1+1+1+\ldots\ldots+1) =$$

$$(9+8+7+6+5+4+3+2+ 18) = 62/26$$

solve a collision

2) Quadratic probing

If hash(k)=d and the bucket is already occupied then we will examine successive buckets d+1, d+2^2, d+3^3……, in the array

example :

(89,18, 49, 58, 69)

hash(k) = k % 10;

0	1	2	3	4	5	6	7	8	9
49		58	69					18	89
2		3	3					1	1

ASVsucc=(1+1+2+3+3)/5

solve a collision

3) Double Hashing

If $hash_1(k)=d$ and the bucket is already occupied then we will counting $hash_2(k)=c$, examine successive buckets d+c, d+2c, d+3c……, in the array

example:

hash1(k) = k % 10;

hash2(k) = R – (k % R);

其中 R < TableSize的质数

(89, 18, 49, 58, 69)

hash2(49)=7-49%7=7 (R=7)

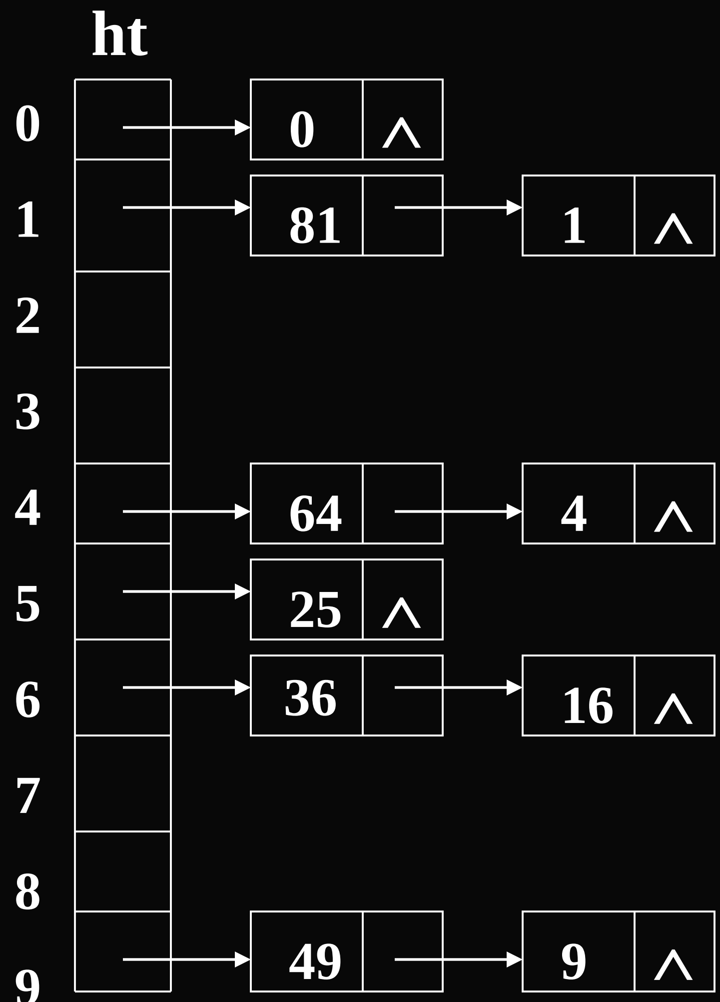
hash2(58)=7-58%7=5

hash2(69)=7-69%7=1

0	1	2	3	4	5	6	7	8	9
69			58			49		18	89

solve a collision

- Separate Chaining



0, 1, 4, 9, 16, 25, 36, 49, 64, 81

$$Hash(x) = x \% 10$$

Chapter 5

例子：

设散列表为HT[13]，散列函数为

H(key) = key % 13 。用线性开地址法解决冲突，对下列关键码序列 12,23,45,57,20,03,78,31,15,36：

1) 画出其散列表。

2）计算等概率下搜索成功的平均搜索长度。

3）如果采用链表散列解决冲突，画出该链表。

2010年统考题

综合应用题(10分)

将关键字序列(7, 8, 30, 11, 18, 9, 14) 散列存储到散列表中, 散列表的存储空间是一个下标从0开始的一个一维数组中, 散列函数为:

H(key) = (key*3) MOD T

处理冲突采用线性探测法,要求装载因子为0.7

问题:

1). 请画出所构造的散列表;

2). 分别计算等概率情况下,查找成功和查找不成功的平均查找长度.

注: 所谓查找不成功的平均查找长度是指: 在表中所有可能散列到的位置上,要插入新元素时为找到空桶的探查次数的平均值.

解答:

1). 由装载因子0.7, 数据总数7个,得到存储空间长度为10, 所以

H(hey) = (key*3) MOD 10　　　(7, 8, 30, 11, 18, 9, 14)

散列表为:

0	1	2	3	4	5	6	7	8	9
30	7	14	11	8	18		9		
1	1	1	1	1	2		1		

2). 查找成功的ASL= (1+1+1+1+1+2+1)/7 = 8/7

查找不成功的ASL= (7+6+5+4+3+2+1+2+1+1)/10 = 3.2

注: 所谓查找不成功的平均查找长度是指: 在表中所有可能散列到的位置上,要插入新元素时为找到空桶的探查次数的平均值.

第6章：优先队列

1.优先队列的概念

2.优先队列的实现

用无序的线性表来实现

用堆来实现----堆的定义

初始化一个堆

堆排序

优先队列

1.优先队列的概念

- A priority queue is a collection of zero or more elements. Each element has a priority or value.
- Operations:

  1)find an element

  2)insert a new element

  3)delete an element

Heaps

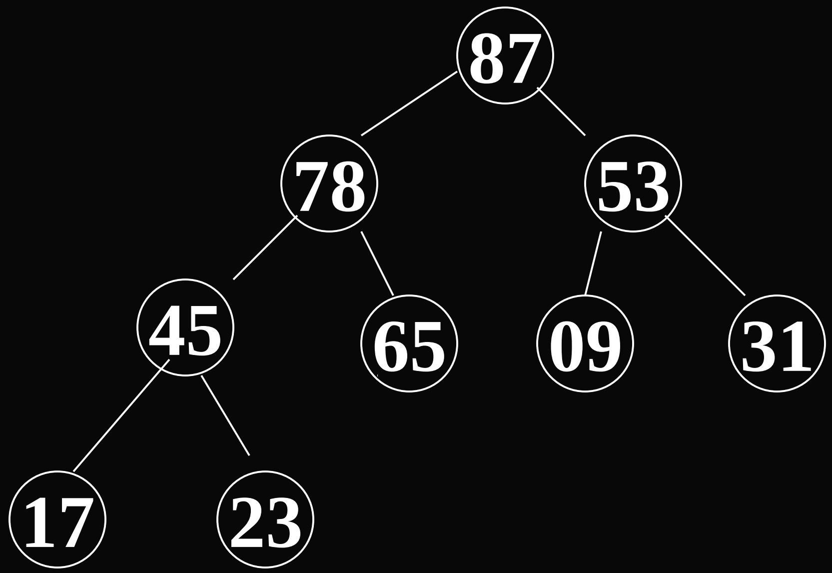
2.优先队列的实现(用堆）

A max heap(min Heap)

- **is A complete binary tree**
- **The value in each node is greater(less) than or equal to those in its children(if any).**

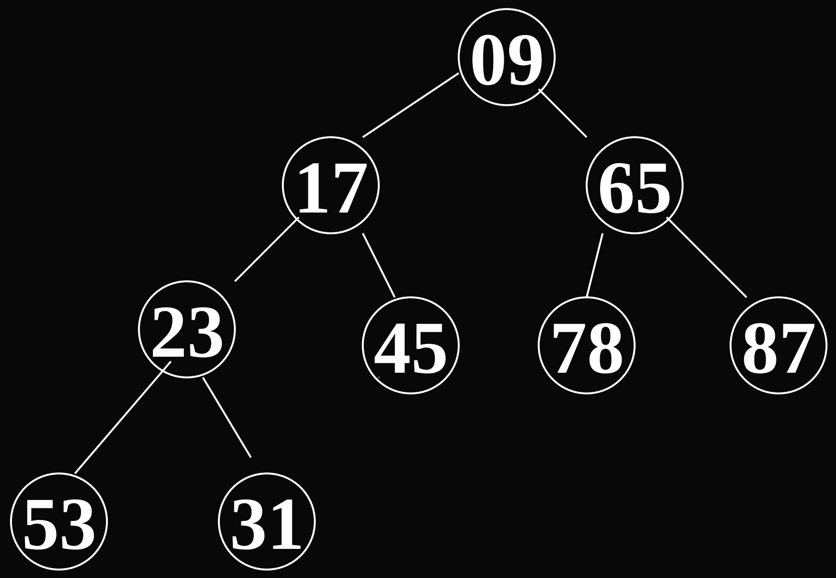
Example of a max heap

k={87,78,53,45,65,09,31,17,23}



Example of a min heap

k={09,17,65,23,45,78,87,53,31}



考纲上的题：

判别以下序列是否是堆？如果不是，将它调整为堆。

1) { 100, 86, 48, 73, 35, 39, 42, 57, 66, 21 }

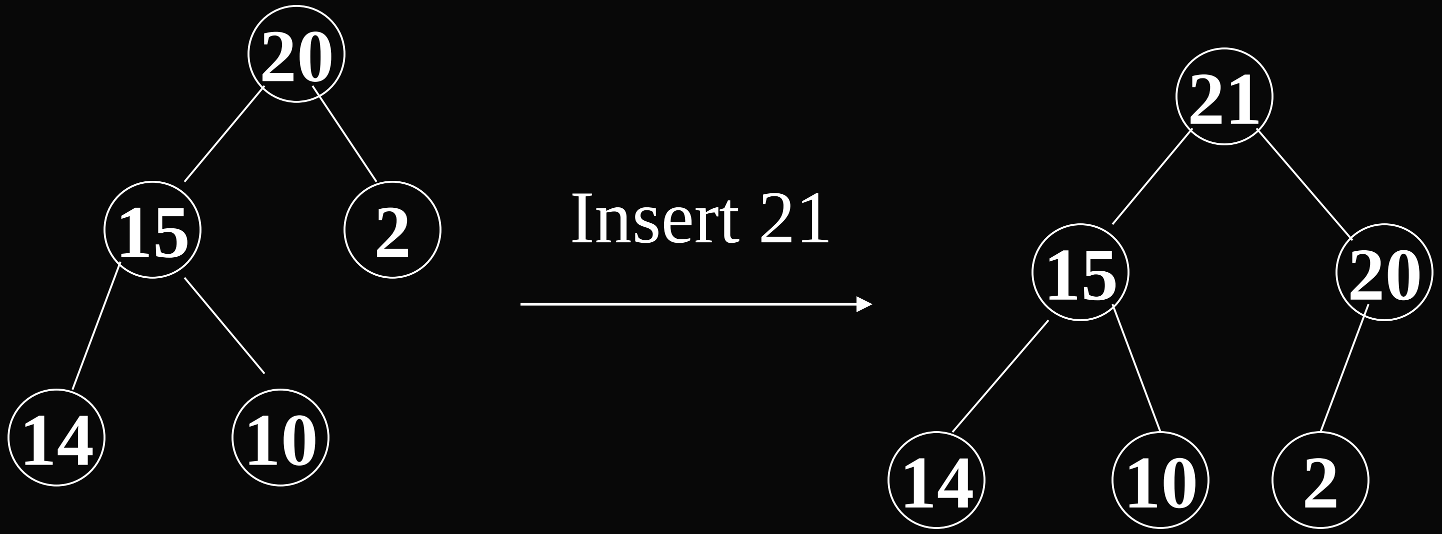
2) { 12, 70, 33, 65, 24, 56, 48, 92, 86, 33 }

3) { 103, 97, 56, 38, 66, 23, 42, 12, 30, 52, 06, 20 }

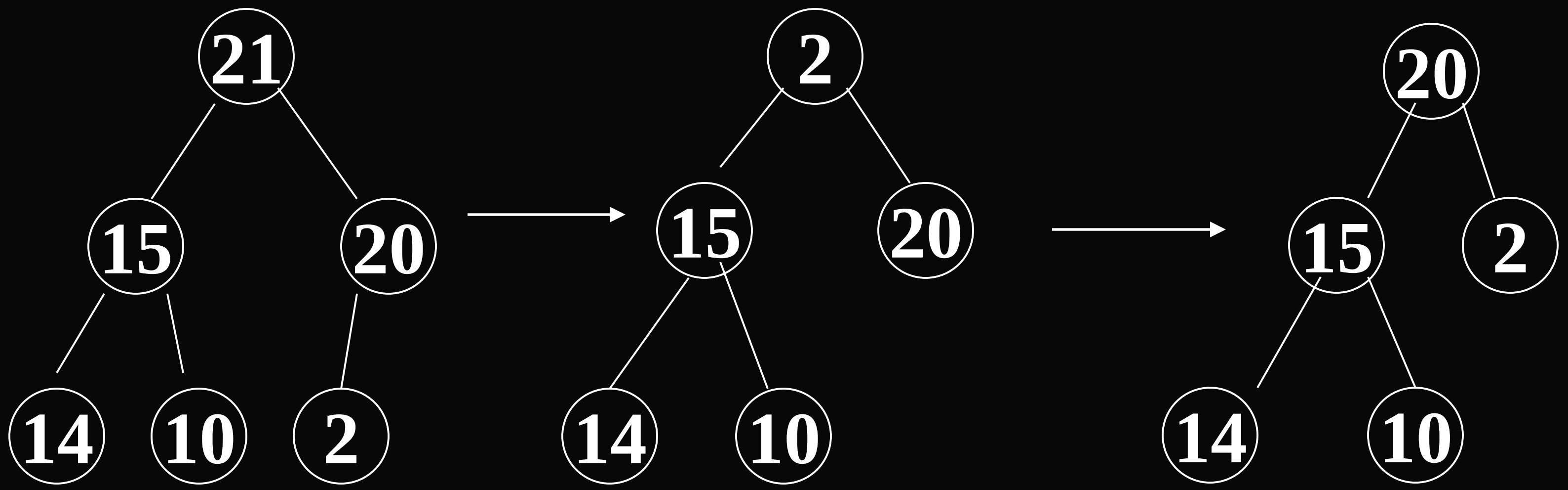
4) { 05, 56, 20, 23, 40, 38, 29, 61, 35, 76, 28, 100 }

Insertion

Example:



deletion

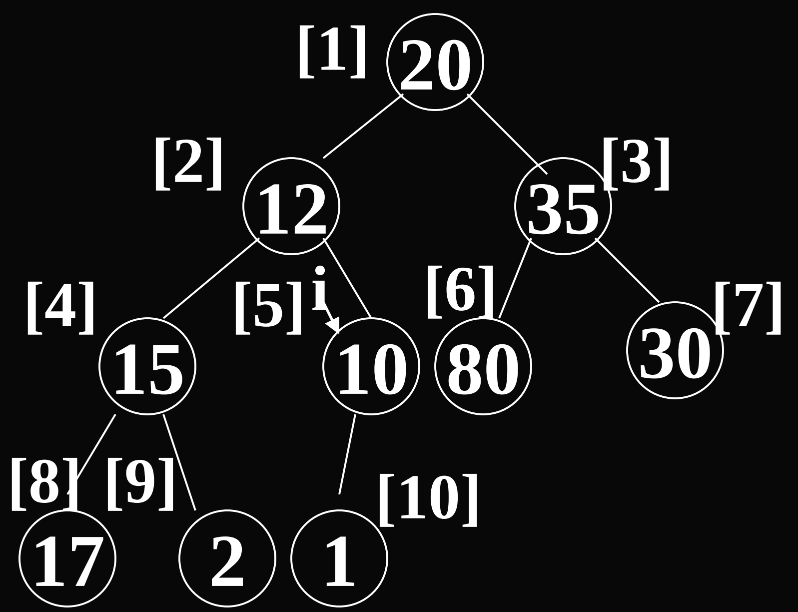


Initialize a nonempty max heap

有两种方法建堆:

- 将数据依次放入一棵完全二叉树, 然后由下而上调, 如下例. O(n)
- 输入一个数据,就调整一下,即由上而下调. O(nlogn)

Example: {20,12,35,15,10,80,30,17,2,1}

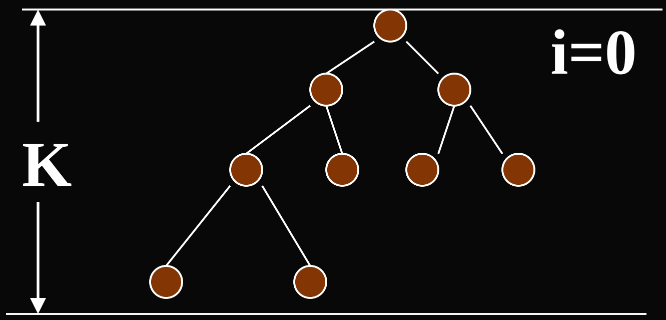


$$\underbrace{i=[n/2],\ [n/2]-1\ ,\ldots,\ 1}$$

Turn into max heap from these subtree roots

Create Heap time complexity:

初始建堆：n个结点，$K=\lfloor \log_2 n \rfloor$，从0层开始



第i层交换的最大次数为k-i

第i层有2^i个结点

$$\text{总交换次数：} \sum_{i=0}^{k-1} 2^i \cdot (k-i) = \sum_{j=1}^{k} j \cdot 2^{k-j} = \sum_{j=1}^{k} j(2^k \cdot 2^{-j})$$

令k-i=j

$$= 2^k \cdot \sum_{j=1}^{k} j \cdot 2^{-j} \le 2^k \cdot 2 \le 2^{\log n} \cdot 2 = 2n = O(n)$$

heap sort

Method:

1)initialize a max heap with the n elements to be sorted $O(n)$

2)each time we delete one element, then adjust the heap $O(\log_2 n)$

Time complexity is $O(n)+O(n*\log_2 n)=O(n*\log_2 n)$

heap sort

例子:

Example :{21,25,49,25*,16,08}

Chapter 6

设待排序的关键码序列为{ 12, 2, 16, 30, 28, 10, 16*, 20, 6, 18 }, 使用堆排序方法进行排序。写出建立的初始堆，以及调整的每一步。

第7章：排序

- 各种排序方法的算法思想与时间复杂度的分析

1.排序的有关概念

稳定性

2.插入排序(直接插入排序，二分法插入排序，shell排序)

3.交换排序(起泡排序，快速排序）

4.选择排序（直接选择排序，堆排序）

5.归并排序

6.基数排序

第7章：排序

1.排序的有关概念

内排序：对内存中的n个对象进行排序。

外排序：内存放不下，还要使用外存的排序。

排序算法的稳定性：

如果待排序的对象序列中，含有多个关键码值相等的对象，用某种方法排序后，这些对象的相对次序不变的，则是稳定的, 否则为不稳定的。

例： $35 \quad 8_1 \quad 20 \quad 15 \quad 8_2 \quad 28$

$8_1 \quad 8_2 \quad 15 \quad 20 \quad 28 \quad 35$

稳定的

第7章：排序

2. 插入排序(直接插入排序，二分法插入排序，表插入排序，shell排序)

- **直接插入排序**

例子

v_0 i=1

8 3 2 5 9 1 6

3 8

2 3 8

2 3 5 8

…

算法分析

1)n个对象已有序

比较总次数KCN＝n－1＝O（n）

移动次数　RMN＝2*(n－1)＝O（n）

2）n个对象逆序

KCN＝1+2+3+…+(n-1)＝n（n－1）/2=$O(n^2)$

RMN=(1+2)+(2+2)+…+(n-1+2)= n（n－1）/2+2(n-1)

=$(n^2+3n-4)/2=O(n^2)$

- 折半插入排序（Binary Insert Sort）
  也称二分法插入排序

1.思想

0	1	2	3	4	5	6	7
28	13	72	85	39	41	6	20
6	13	28	39	41	72	85	20

算法分析

折半查找所需比较次数与初始排序无关，仅依赖于对象个数

比较次数： v_0， v_1， $v_2,\ldots,v_{i-1}$， $v_i,\ldots,v_{n-1}$

$\uparrow$

- 设 $n=2^k$，插入第 i 个对象时，需要经过 $\lfloor \log_2 i \rfloor + 1$
- 次关键码比较
- ∴折半查找所需的关键码比较次数为：
- $\sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 1) = \underbrace{1}_{2^0\text{个}1} + \underbrace{2+2}_{2^1\text{个}2} + \underbrace{3+3+\ldots+3}_{2^2\text{个}3} + \underbrace{4+\ldots+4+4}_{2^3\text{个}4} + \ldots + \underbrace{k+k+\ldots+k}_{2^{k-1}\text{个}k}$
- $= 2^0+2^1+2^2+\ldots+2^{k-1}$
- $+ 2^1+2^2+\ldots+2^{k-1}$
- $+2^2+\ldots+2^{k-1}$
- $\ldots\ldots$
- $+2^{k-2}+2^{k-1}$
- $+2^{k-1}$

$$=\sum_{i=1}^{k}(2^k-2^{i-1})=k\cdot 2^k-\sum_{i=1}^{k}2^{i-1}=k\cdot 2^k-2^k+1$$

$$=n\cdot\log_2 n - n+1\approx$$

$$n\cdot\log_2 n=O(n\cdot\log_2 n)$$

稳定性：稳定

- 希尔排序

例子

	0	1	2	3	4	5	6	7	8
gap＝4	72	73	71	23	94	16	05	68	10
gap=2	10	16	05	23	72	73	71	68	94
gap=1	05	16	10	23	71	68	72	73	94
	05	10	16	23	68	71	72	73	94

稳定性：不稳定

算法分析：与选择的缩小增量有关，但到目前还不知如何选择最好结果的缩小增量序列。

平均比较次数与移动次数大约$n^{1.3}$左右。

3.交换排序(起泡排序，快速排序）

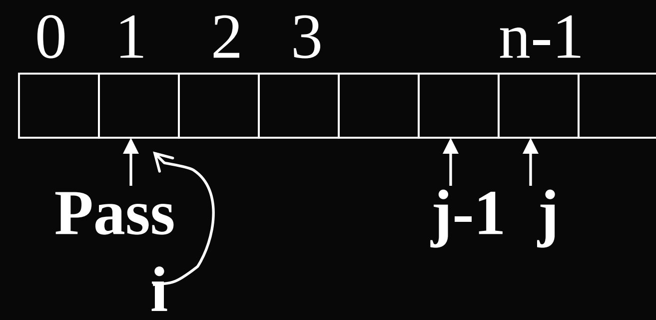
- 起泡排序

方法： 1）从头到尾做一遍相邻两元素的比较，
有颠倒则交换，记下交换的位置。一
趟结束，一个或多个最大（最小）元
素定位。

2）去掉已定位的的元素，重复1，直至一
趟无交换。

例子

67		67	
54		54	
46		46	
38		38	
20		32	
15	t	20	
32	t	15	t
25		25	t
8		8	



∴不一定要做**n-1**趟且每次比
不一定要比整个数组。

4.算法分析

最小比较次数

有序：**n-1**次比较，移动次数为**0**

最大比较次数

逆序：**(n-1)+(n-2)+…+1=n(n-1)/2≈O(n²)**

(比较次数)

$$3\sum_{i=1}^{n-1} i=(3/2)n(n-1)$$

(移动次数)

5.稳定性

起泡排序是稳定的

- 快速排序（分划交换排序）

  1962年Hoare提出的。

1. 方法：

   1）在n个对象中，取一个对象（如第一个对象——基准pivot），按该对象的关键码把所有≤ 该关键码的对象分划在它的左边。>该关键码的对象分划在它的右边。

   2） 对左边和右边（子序列）分别再用快排序。

2. 例子

```
 i                                                  j
 ↓                                                  ↓
46    13   55   42    94    05   17   70   82   100
[17   13   05   42]   46    [94  55   70   82   100]
[05   13]  17   [42]  46    [94  55   70   82   100]
05    13   17   42    46    [94  55   70   82   100]
05    13   17   42    46    [82  55   70]  94   100]
05    13   17   42    46    [70  55]  82   94   100
05    13   17   42    46    55   70   82   94   100
```

3. 算法分析

1）最差的情况（当选第一个对象为分划对象时）

如果原对象已按关键码排好序

$$\left.\begin{array}{l} K_1\ [\qquad\qquad\qquad\qquad\qquad\qquad] \\ \qquad K_2\ [\qquad\qquad\qquad\qquad\quad] \\ \qquad\qquad K_3\ [\qquad\qquad\qquad] \\ \qquad\qquad\qquad\qquad\qquad \cdots\cdots \end{array}\right\} \quad O(n^2)$$

2) 最理想的情况

每次分划第一个对象定位在中间

2^0:n

2^1:2*(n/2)

2^2:4*(n/4)

…….

2^k:

设$n=2^k$,一共做了K趟 $K=\log_2 n$

$$T(n)\le n+2T(n/2)\le n+2(n/2+2T(n/4))=2n+2^2T(n/2^2)$$
$$\le 2n+2^2(n/2^2+2T(n/2^3))=3n+2^3T(n/2^3)\le\ldots$$
$$\le kn+2^kT(n/2^k)=n\log_2 n+nT(1)=O(n\log_2 n)$$

可以证明Quicksort的平均计算时间也是O（ $n\log_2 n$ ）

4．选择排序

方法：1.直接选择排序
　　　2.堆排序

- 直接选择排序

  思想：首先在n个记录中选出关键码最小（最大）的记录，然后与第一个记录（最后第n个记录）交换位置，再在其余的n-1个记录中选关键码最小（最大）的记录，然后与第二 个记录（第n-1个记录）交换位置，直至选择了n－1个记录。

	0	1	2	3	4	5
例子:	21	25	49	25^{*}	16	08
	08	[25	49	25^{*}	16	21]
	08	16	[49	25^{*}	25	21]
	08	16	21	[25^{*}	25	49]
	08	16	21	25^{*}	25	49]
	08	16	21	25^{*}	25	49

算法分析：比较次数$n-1+n-2+\ldots+1=n(n-1)/2=O(n^2)$

与原始记录次序无关。

稳定性　：不稳定的。

堆排序（由J.W.J.Willman提出的）

1.思想：第一步，建堆，根据初始输入数据，利用堆的调整算法FilterDown（），形成初始堆。（形成最大堆）

第二步，一系列的对象交换和重新调整堆

2.例子：书中的例子{21 25 49 25* 16 08}

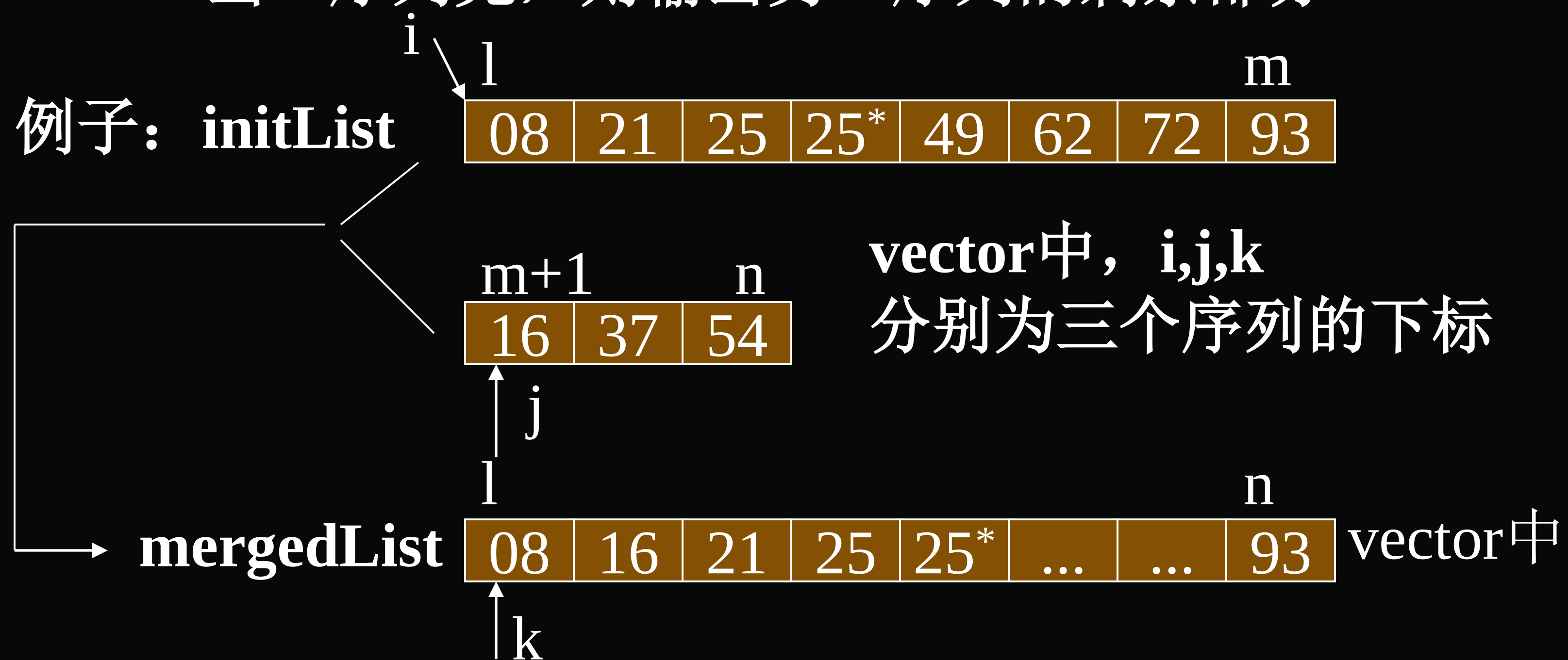
i=⌊（n-1）/2⌋=⌊5/2⌋=2、1、0进行FilterDown（）

5 归并排序（merge sort）

一、归并：两个（多个）有序的文件组合成一个有序文件

方法：每次取出两个序列中的小的元素输出之；

当一序列完，则输出另一序列的剩余部分



二、迭代的归并排序算法

1.方法:

n个长为1的对象两两合并，得n/2个长为2的文件

n/2个长为2……………………得n/4个长为4的文件

⋮

2个长为n/2的对象两两合并,得 1个长为n的文件

n个initList.vector[]

0 1 2 3 4 5 6 7 8 9 10

2.例子:[21][25][49][25*][93][62][72][08][37][16][54] len=1

一趟

[21 25][25* 49][62 93][08 72][16 37][54] len=2

二趟

[21 25 25* 49][08 62 72 93][16 37 54] len=4

三趟

[08 21 25 25* 49 62 72 93][16 37 54] len=8

四趟 [08 21 25 25^* 49 62 72 93][16 37 54] len=8

[08 16 21 25 25^* 37 49 54 62 72 93] len=16

3.算法:

主程序（多趟）→ 一趟 → 多次merge

4.算法分析：合并趟数$\log_2 n$,每趟比较n次，所以为$O(n\log_2 n)$

5. 稳定性：稳定。

6. 基数排序

链式基数排序（Radix Sort）

1)例子

288　371　260　531　287　235　56　299　18　23

因为十进制数字的范围在[0，9]，所以分配10只桶或盒子

先从个位开始排

	531							18	
260	371		23		235	56	287	288	299
0	1	2	3	4	5	6	7	8	9

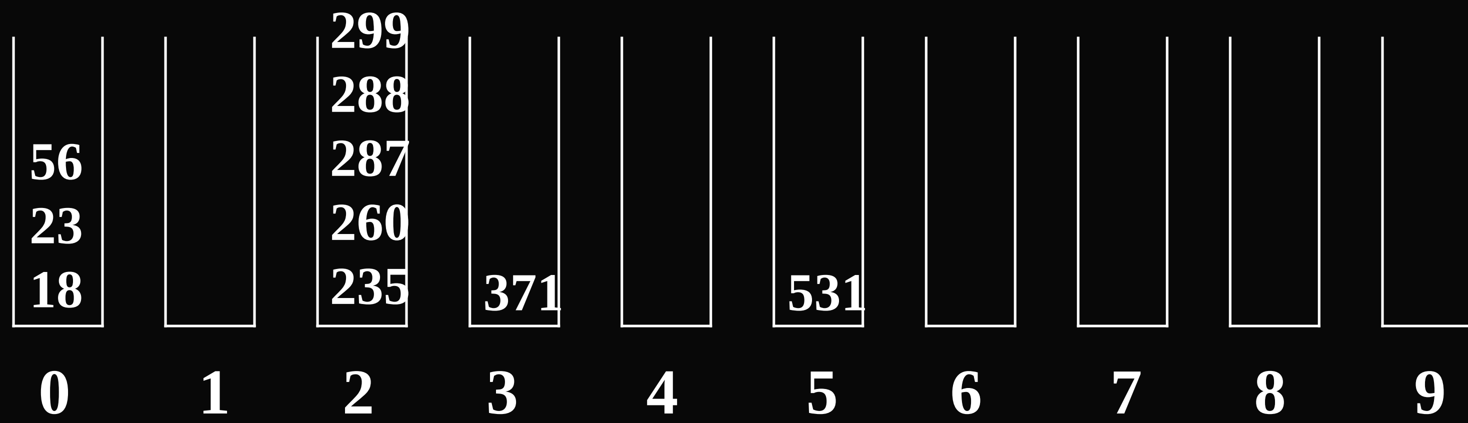
收集 260 371 531 23 235 56 287 288 18 299

再对十位数字排

			235					288	
	18	23	531		56	260	371	287	299
0	1	2	3	4	5	6	7	8	9

收集18 23 531 235 56 260 371 287 288 299

再对百位数字排



收集18 23 56 235 260 287 288 299 371 531

3)算法分析

初始化桶 O（ radix）
分配桶 O（n）
收集桶 O（radix）

$\left.\begin{array}{l}\text{初始化桶 } O(radix)\\ \text{分配桶 } O(n)\\ \text{收集桶 } O(radix)\end{array}\right\}$ O(n+radix)

循环d次，所以，总执行时间为O(d.(n+radix)

附加时间：O(n+2radix)

指针 桶指针

4)稳定的

第7章：排序

复习例题---在O(n)时间内实现将负数排在所有非负数之前。

```
void sort ( float [ ] a, int n )
 {   int i = 0 ,  j = n-1 ;
      while ( i != j )
      {   while ( a[j] >= 0.0 && i < j ) j-- ;
           while ( a[i] < 0 && i < j ) i++ ;
           float temp = a[i] ;  a[i] = a[j]; a[j] = temp;
           j-- ; i++ ;
       }
 }
```

第7章：排序

考纲上的题目：

下列排序算法中，时间复杂度为$O(n\log_2 n)$且占有额外空间最少的是

A. 堆排序　　B. 起泡排序

C. 快速排序　　D. 希尔排序

第9章：图

1.无向图、有向图的有关概念

2.图的机内存储

　　邻接矩阵

　　邻接表

3.图的若干算法

　1) 图的遍历----DFS

　　　　　　　　　BFS

　2) 最小代价生成树---Prime算法

　　　　　　　　　　　Kuscal算法

　3) 最短路径-----Dijkstra算法

　　　　　　　　　Floyed算法

　4) 活动网络----AOV——拓扑排序

　　　　　　　　　AOE——关键路径

4. 例1, 例2

2.图的机内存储

邻接矩阵

邻接表

Representation of graphs and digraphs

1. Adjacency Matrix

$G=(V,E)$, $V=\{V_1,V_2,\ldots\ldots,V_n\}$

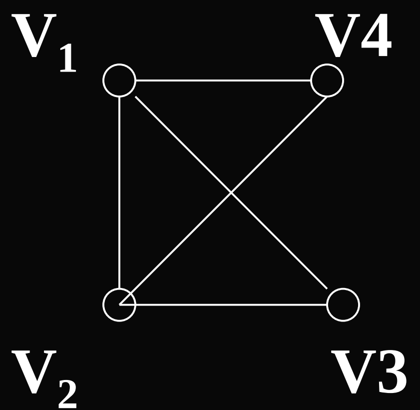
then the adjacency matrix of graph G:

$$A(i,j)=\begin{cases} 1 & \text{if } \langle i,j\rangle,\langle j,i\rangle \in E \text{ or } (i,j)\in E \\ 0 & \text{otherwise} \end{cases}$$

Representation of graphs and digraphs

For example:

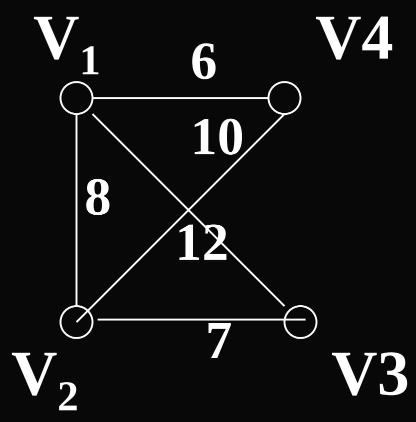
graph



$$A(i,j)=\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

1)Adjacency matrix of graph is a symmetric matrix

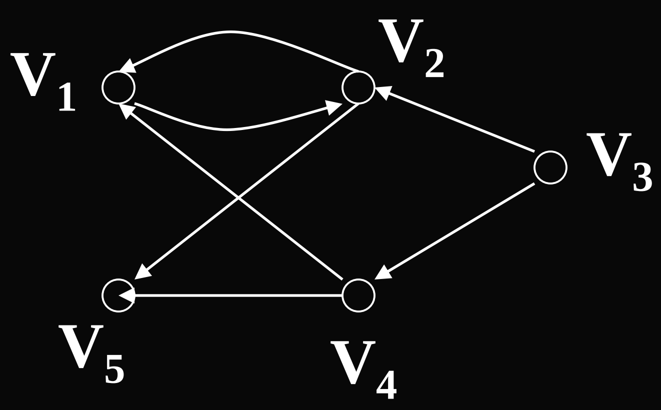
2) $\sum_{j=1}^{n} A(i,j)=\sum_{j=1}^{n} A(j,i)=d_i$ **(degree of vertex i)**



$$A(i,j)=\begin{pmatrix} 0 & 8 & 12 & 6 \\ 8 & 0 & 7 & 10 \\ 12 & 7 & 0 & \infty \\ 6 & 10 & \infty & 0 \end{pmatrix}$$

Representation of graphs and digraphs

Digraph



$$A(i,j)=\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\sum_{j=1}^{n} A(i,j) = d_i^{out} \qquad \sum_{j=1}^{n} A(j,i) = d_i^{in}$$

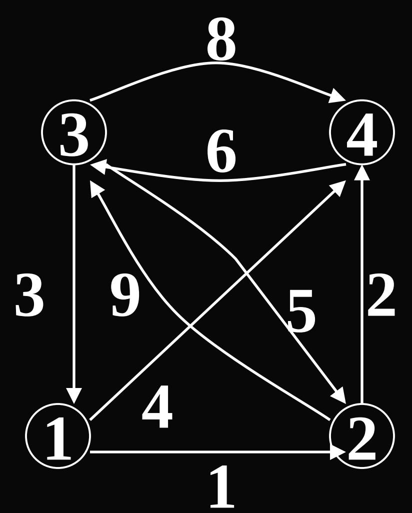
Representation of graphs and digraphs

Representation of networks, replace 1 with weights, others with ∞

$$A(i,j)=\begin{cases} W(i,j) & \text{if } i!=j \text{ and } \langle i,j\rangle,\langle j,i\rangle \in E \text{ or} (i,j)\in E \\ \infty & \text{otherwise} \end{cases}$$

Representation of graphs and digraphs

For example: 除了邻接矩阵外,还要顶点信息表.

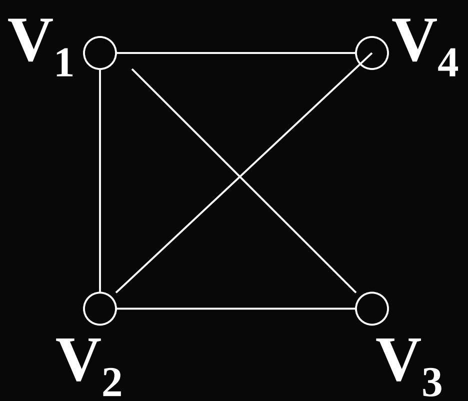


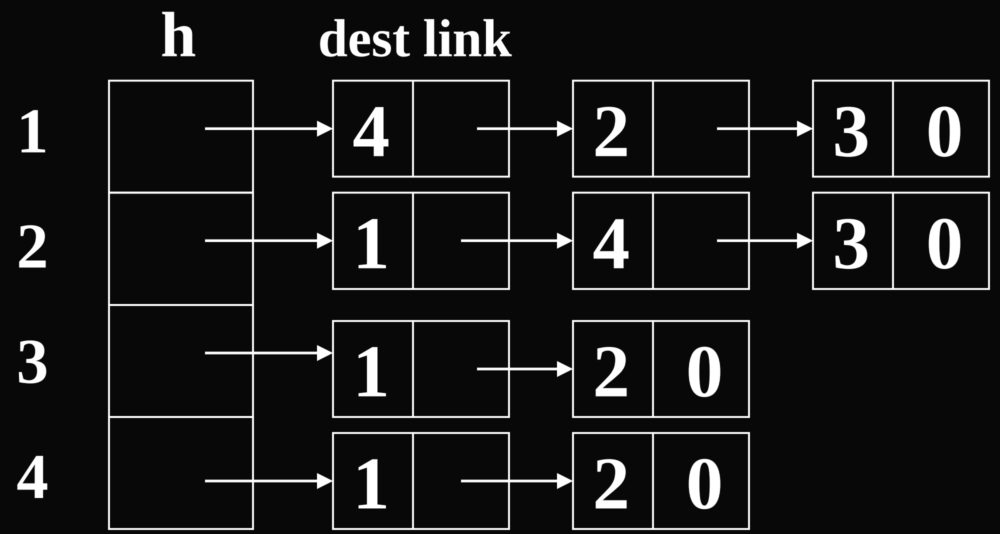
$$A(i,j)=\begin{pmatrix} \infty & 1 & \infty & 4 \\ \infty & \infty & 9 & 2 \\ 3 & 5 & \infty & 8 \\ \infty & \infty & 6 & \infty \end{pmatrix}$$

Representation of graphs and digraphs

2. Linked-adjacency Lists

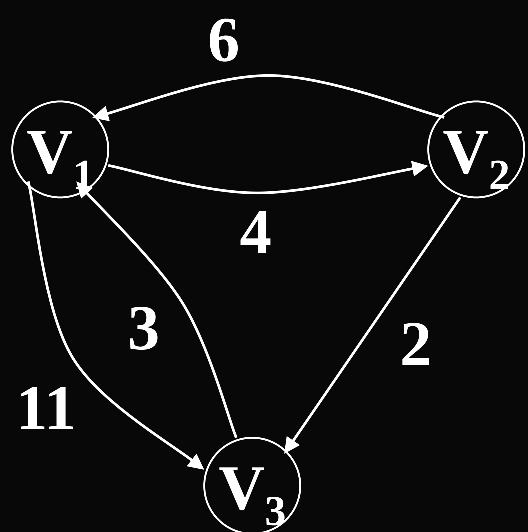
reduce the storage requirement if the number of edges in the graph is small.

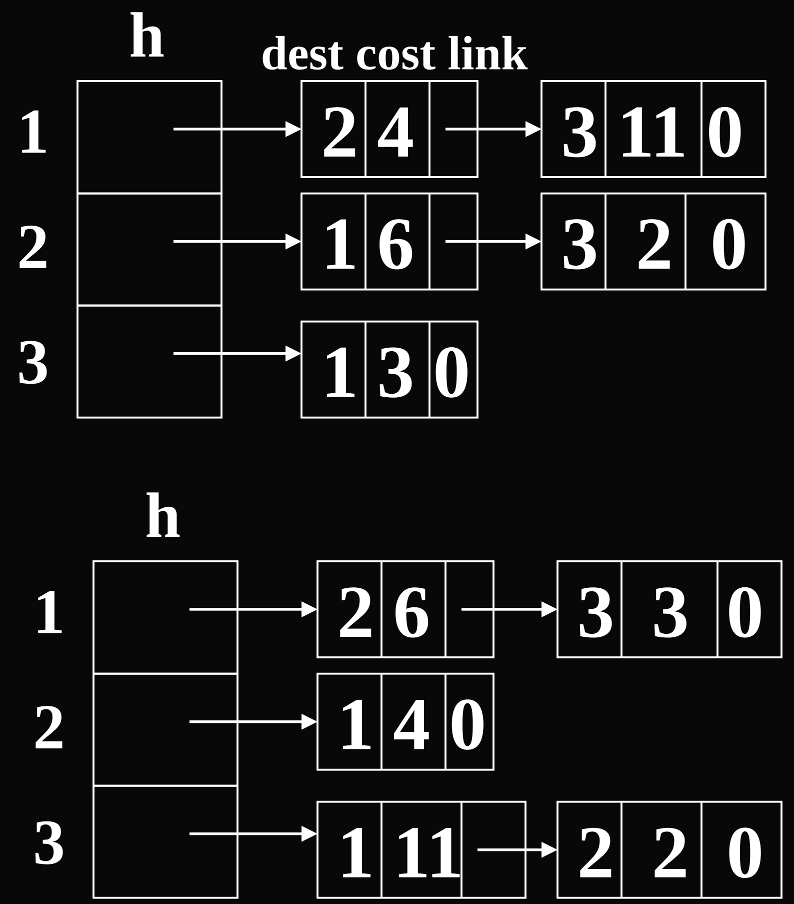




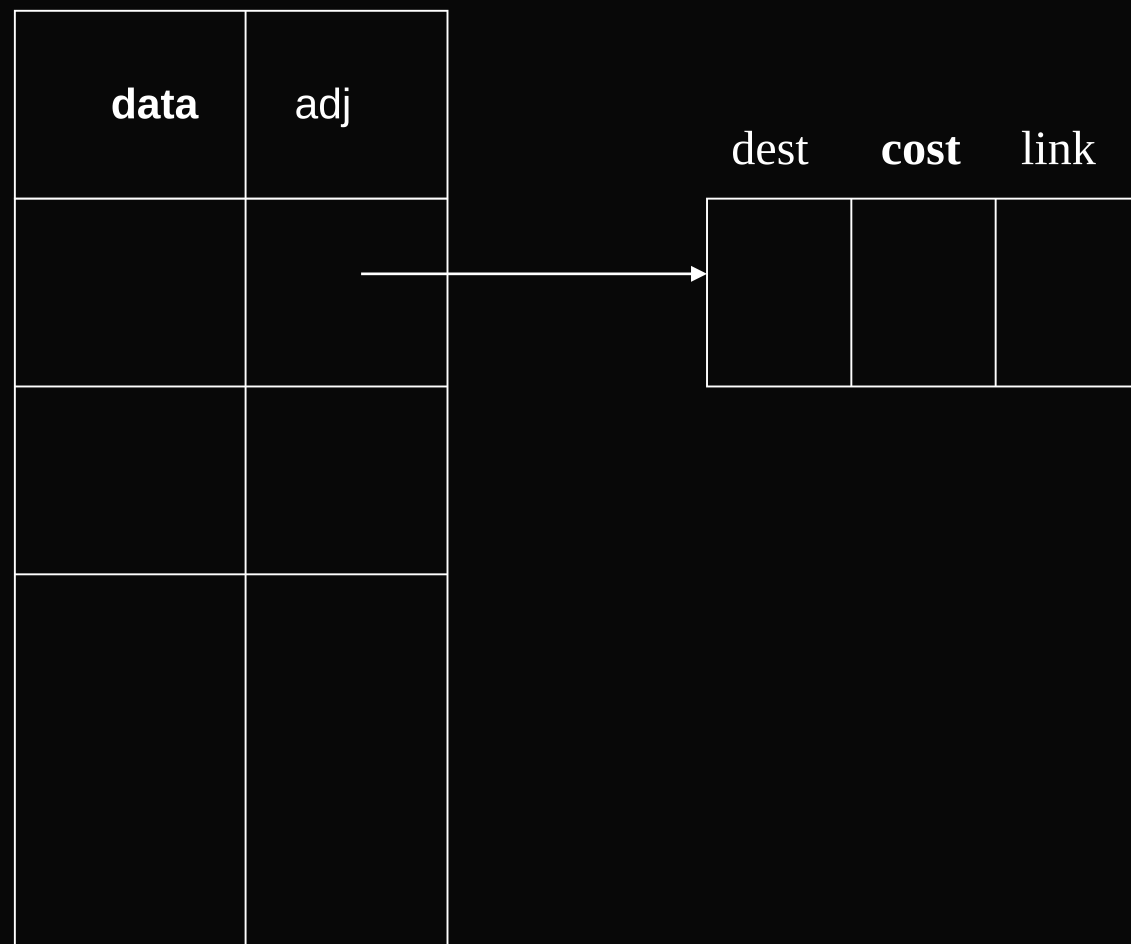
Representation of graphs and digraphs

Digraph:





Node Table:



第9章：图

3.图的若干算法

1) 图的遍历----DFS

BFS

DFS:

思想：　从图中某个顶点V0出发,访问它,然后选择一个V0 邻接到的未被访问的一个邻接点V1出发深度优先遍历图,当遇到一个所有邻接于它的结点都被访问过了的结点U时,回退到前一次刚被访问过的拥有未被访问的邻接点W,再从W出发深度遍历,……直到连通图中的所有顶点都被访问过为止.

BFS:

思想：从图中某顶点V0出发，在访问了V0之后依次访问v0的各个未曾访问过的邻接点，然后分别从这些邻接点出发广度优先遍历图，直至图中所有顶点都被访问到为止.

2) 最小代价生成树---Prime算法

Kuscal算法

最小代价生成树（minimun-cost spanning tree）

问题的提出：如何找到一个网络的最小生成树，即各边权的总和为最小的生成树

Kuscal算法: 实现思想
具体实现
数据结构: 邻接矩阵
堆、并查集----实现技巧

Prime算法： 实现思想
具体实现
数据结构: 邻接矩阵
辅助数据结构：开辟两个附加数组

对于具体实现，还可以有其他一些实现方法

算法结构为：

n

n

求最小的 n

修改 n

时间复杂度：$O(n^2)$

思考题：这两种算法分别适合那种情况？

3) 最短路径-----Dijkstra算法

Floyed算法

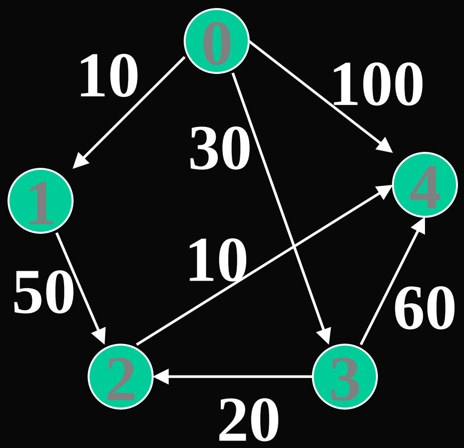
两种算法:

1)边上权值为非负情况的从一个结点到其它各结点的最短路径（单源最短路径）（Dijkstra算法）

2)边上权值为非负情况的所有顶点之间的最短路径

1.含非负权值的单源最短路径（Dijkstra)

- 问题



V_0		V_1	10
V_0	V_3	V_2	50
V_0		V_3	30
V_0	V_3 V_2	V_4	60

如果按距离递增的顺序重新排列一下

	经过		终止	距离
V_0			V_1	10
V_0			V_3	30
V_0	V_3		V_2	50
V_0	V_3	V_2	V_4	60

距离值数组:dist

0	0			
1	10 0-1			
2	∞ 0-2	60 0-1-2	50 0-3-2	
3	30 0-3	30 0-3		
4	100 0-4	100 0-4	90 0-3-4	60 0-3-2-4

路径path

0				
1	0	0	0	0
2	-1	1	3	3
3	0	0	0	0
4	0	0	3	2

每次放由v_0到达该顶点的前一顶点

算法分析：

n

n

n

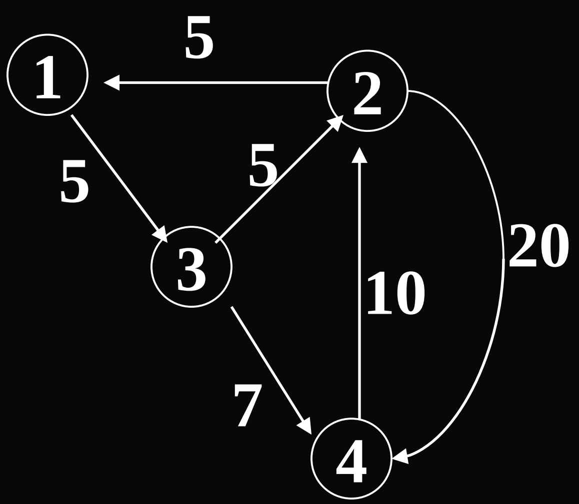
n

$O(n^2)$

2.所有顶点之间的最短路径（floyed)

$O(n^3)$

例子：



$$A=\begin{pmatrix} 0 & \infty & 5 & \infty \\ 5 & 0 & \infty & 20 \\ \infty & 5 & 0 & 7 \\ \infty & 10 & \infty & 0 \end{pmatrix}$$

floyed算法：在矩阵A上作n-1次迭代，设每次迭代结果分别为

$$A^{(0)},A^{(1)},A^{(2)},\ldots A^{(n)}$$

$A^{(0)}$=源矩阵，认为v_i->v_j的直接弧为它们的min路径

$A^{(1)}=A^{(1)}[i,j]=\min(A^{(0)}[i,j], A^{(0)}[i,1]+A^{(0)}[1,j])$

此时 $A^{(1)}[i,j]$可能已换成v_i-v_1-v_j

$A^{(2)}=A^{(2)[}[i,j] =\min(A^{(1)}[i,j], A^{(1)}[i,2]+A^{(1)}[2,j])$

即考虑经过顶点2，它可能是 v_i-v_j, v_i-v_1-v_j, v_i-v_1-v_2-v_j, v_i-v_2-v_1-v_j, v_i-v_2-v_j的min者

⋮

$A^{(k)}=A^{(k)[}[i,j] =\min(A^{(k-1)}[i,j],{}^{A(k-1)}[i,k]+A^{(k-1)}[k,j])$

⋮

4) 活动网络----AOV——拓扑排序

AOE——关键路径

- 用顶点表示活动的网络
  （拓扑排序—**topological sort)**

算法思想：

1）从图中选择一个入度为0的结点输出之。

（如果一个图中，同时存在多个入度为0的结点，则随便输出那一个结点）

2）从图中删掉此结点及其所有的出边。

3）反复执行以上步骤：a）直到所有结点都输出了，则算法结束

b）如果图中还有结点，但入度不为0，则说明有环路

算法分析：**n**个顶点，**e**条边

建立链式栈**O**（**n**）

每个结点输出一次，每条边被检查一次**O**（**n**＋**e**）

所以为:**O**（**n**＋**n**＋**e**）

- 用边表示活动的网络（AOE网络，Activity On Edge Network）

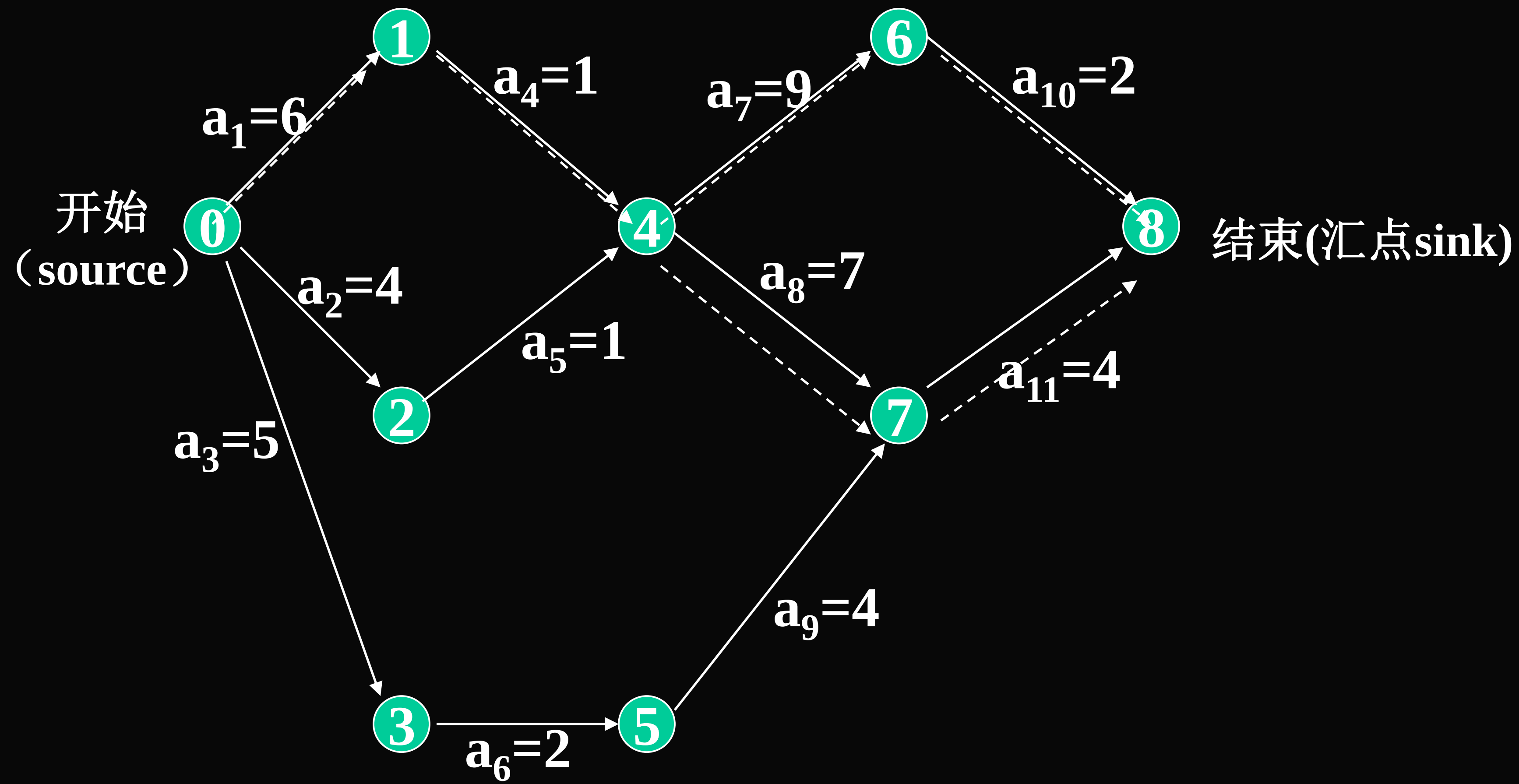
  又称为事件顶点网络

  顶点：表示事件（event）

  事件——状态。表示它的入边代表的活动已完成，它的出边代表的活动可以开始，如下图v_0表示整个工程开始，v_4表示a_4，a_5活动已完成a_7，a_8活动可开始。

  有向边：表示活动。

  边上的权——表示完成一项活动需要的时间



开始
（source）
0
1
2
3
4
5
6
7
8
结束(汇点sink)
a1=6
a2=4
a3=5
a4=1
a5=1
a6=2
a7=9
a8=7
a9=4
a10=2
a11=4

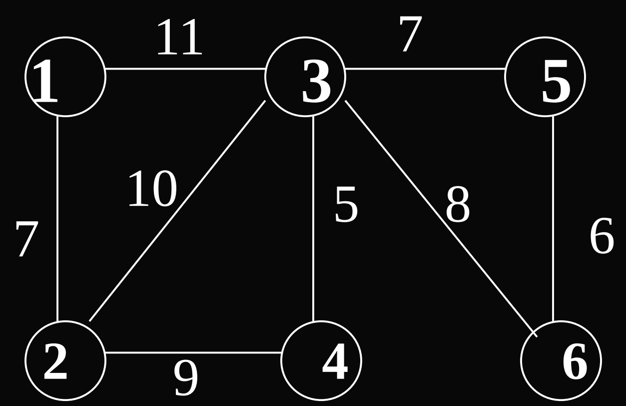
关键路径（**critical path**）

1)目的 **:** 利用事件顶点网络，研究完成整个工程需要多少时间
加快那些活动的速度后，可使整个工程提前完成。

2)关键路径：具有从开始顶点**(**源点）→完成顶点（汇点）的
最长的路径

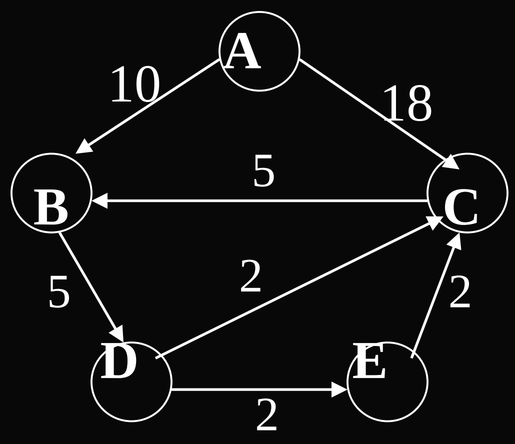
$$\text{算法分析：}\left.\begin{array}{l}\left.\begin{array}{l}\text{按拓扑排序求}Ve[i]\\ \text{按逆拓扑排序求}Vl[i]\end{array}\right\} O(n+e)\\ \text{求各活动}e[k]\text{和}l[k] \qquad O(e)\end{array}\right\} O(n+e)$$

1. 对下列无向图：



分别用Prim算法与Kruscal算法，求出最小代价生成树（要求写出构造生成树的每一步）。

2. 对下列有向图：



用Dijkstra算法求从顶点A到其它各顶点的最短路径。

3.考纲上的题：

（10分）设无向图 G = （ V， E）， 其中 V = { 1, 2, 3, 4, 5 },

E = { (1,2,4) , (2,5,5) , (1,3,2) , (2,4,4) , (3,4,1) , (4,5,3) , (1,5,8) } , 每条边由一个三元组表示，三元组中前两个元素为与该边关联的顶点，第三个元素为该边的权。请写出图 G 中从顶点1到其余各点的最短路径的求解过程。要求列出最短路径上的各顶点，并计算路径长度。