第2章 程序性能
一.$O(n^2)$的一些排序
二.渐进符号定义
三.排序总结
第3章 线性表
一.公式化描述（顺序存储）
3.基本操作复杂度
二.链式描述
2.基本操作复杂度
3.优化操作的实现方式:
3.1 单向循环链表：最后一个节点指向第一个节点
3.2 带哑元的链表：链表前部附加一个头节点
3.3 双向链表：next指针变为left和right两个指针
三.间接寻址
1.结构:
2.基本操作复杂度
3.优点
四.桶排序和基数排序
五.课本中扩展知识
1. 凸包
2. dsu
第4章 数组和矩阵
一.数组
二.矩阵
4.稀疏矩阵(sparse matrix)
5.十字链表表示稀疏矩阵(linked matrix)
第5章 栈
一.公式化描述
二.链表描述
三.课本中扩展知识
1.括号匹配
2.火车车厢重排
3.开关盒布线
第6章 队列
一.公式化描述
1.队首始终在数组第一个位置
2.队首也移动
3.循环数组
二.链表描述
1.应选择front->near的指针方向
三.课本中扩展知识
1.火车车厢重排
2.电路布线
3.图元识别
第7章 跳表和散列
一.字典(有序表)
1.定义
2.线性表表述
3.跳表表述
4.散列表(hashing)描述
4.1 直接定址法
4.2 数字分析法
4.3 平方取中法
4.4 折叠法
4.5 除留余数法

二.散列表(hashing)表示法中处理冲突
1.线性探测法
2.二次探测法
3.双散列法
4.链式散列法
三.课本中扩展知识
1. LZW压缩
第8章 二叉树和其他树
一.基本概念
14.“度”定理
15.二叉树和树的区别
16.二叉树的特性
17.满二叉树
18.完全二叉树
19.在完全二叉树中任意节点有记其序号为$i(1\le{}i\le{}n)$：
二.二叉树描述
1.公式化描述：根节点i=1，左孩子i<<1，右孩子i<<1|1，父亲i>>1
2.链表描述
三.二叉树的遍历
四.树和二叉树互转，森林和二叉树互转
1.树和二叉树互转
2.森林和二叉树互转
五.线索二叉树
第9章 优先队列
一.基本概念
1.大根树（小根树）
2.大根堆（小根堆）
3.父子节点位置关系
二.堆的基本操作(以最大堆为例)
1.插入($O(logn)$):**从下向上调整**
2.删除($O(logn)$):**从上向下调整**
3.创建堆 (两种实现方法)
3.1 (直接插入n次) $O(nlogn)$
3.2 (重整) $\theta(n)$
3.3 重整方法的复杂度推导
三.堆排序
四.哈夫曼树($Huffman$ tree)
1.一些定义
2.构造过程 $O(nlogn)$
3.一些性质
第10章 BST AVL
一.BST
1.定义
2.索引二叉搜索树
方便解决的问题：求第k大元素
3.搜索 $O(logn)$
4.插入 $O(logn)$
5.删除 $O(logn)$
二. AVL
1.定义
2.一个问题：高度为h的AVL的最少节点数
3.搜索
4.插入
5.删除
第11章 BT RBT
一. m叉搜索树
二. m阶$B-$树
1.搜索

2.插入
3.删除
三. RBT
1.定义
2. RBT和$2-3-4$树之间的关系
3.搜索 O(logn)
4.插入
第12章 图
一.基本概念
二.图的表示
三.最小生成树
1. Kruskal
2. Prim
四.最短路
1. Dijkstra
2. Floyd
五. Topology Sort
六.关键路径
七.应用
EX_1 排序和查找

第2章 程序性能

一.$O(n^2)$的一些排序

- **计数排序**

> 遍历所有元素，将它和左边所有元素比较，大的名次+1，之后再调换到相应的位置上。显然时间复杂度$O(n^2)$

- 选择排序

> 进行n轮操作，对第i轮，遍历没有移动过的$n-i+1$元素，把最小(大)的放在第1(n)个。显然时间复杂度$O(n^2)$

- 冒泡排序

> 对相邻元素进行比较，如果左边的元素大于右边的元素，则交换。进行n次本操作。显然时间复杂度$O(n^2)$

- 插入排序

> 从无序部分依次取一元素插入有序部分的正确位置上。显然时间复杂度$O(n^2)$　稳定排序!

- 稳定排序定义：**对于任意的$i<j, a_i=a_j$，若排序后a_i, a_j的位置$i^{'}, j^{'}$一定满足$i^{'}<j^{'}$,则该排序算法为稳定的，否则为不稳定的。**

二.渐进符号定义

- $f(n)=O(g(n))$，当且仅当存在正常数c和n_0，使对于所有$n \geq n_0$，有$f(n) \leq cg(n)$
- $f(n)=\Omega(g(n))$，当且仅当存在正常数c和n_0，使得对所有$n \geq n_0$，有$f(n) \geq cg(n)$
- $f(n)=\Theta(g(n))$，当且仅当存在正常数c_1、c_2和n_0，使得对所有$n \geq n_0$，有$c_1 g(n) \leq f(n) \leq c_2 g(n)$
- $f(n)=o(g(n))$，当且仅当$f(n)=O(g(n))$，且$f(n) \neq \Omega(g(n))$
- 大O松散上界，大Ω松散下界，Θ同为上下界，小o严格上界

三.排序总结

名称	平均时间复杂度	最坏时间复杂度	稳定性
冒泡	$O(n^2)$	$O(n^2)$	稳定
选择	$O(n^2)$	$O(n^2)$	**不稳定**
直接插入	$O(n^2)$	$O(n^2)$	**稳定**
归并	$O(nlogn)$	$O(nlogn)$	稳定
快速	$O(nlogn)$	$O(n^2)$	不稳定
堆	$O(nlogn)$	$O(nlogn)$	不稳定
希尔	$O(nlogn)$	$O(n^2)$	不稳定
基数	$O(NM)$	$O(NM)$	稳定

- 对于选择排序稳定性，比如序列 5 8 5 2 9，第一个5会换到2的后面。

第3章 线性表

一.公式化描述（顺序存储）

1.基本内容略

2.多个表，元素个数和$\leq n$，但每个表各自元素$\leq n$，为了提高空间效率，可以将多个表共用一个数组，使用两个额外数组，f[i],l[i]分别记录第i个表的第一个元素和最后一个元素在数组中的位置

3.基本操作复杂度

isempty length find $O(1)$

search delete insert output $O(n)$

二.链式描述

1.和公式化描述的本质区别：数据结构中的关联元素不保证存储空间上的关联 $O(n)$

2.基本操作复杂度

ADT	顺序表	单向链表
Destroy	O(1)	O(n)
isempty	O(1)	O(1)
length	O(1)	
find	**O(1)**	**O(k)**
search	**O(n)**	**O(n)**
delete		**O(k)**
insert		**O(k)**
output	O(n)	O(n)

3.优化操作的实现方式：

3.1 单向循环链表：最后一个节点指向第一个节点

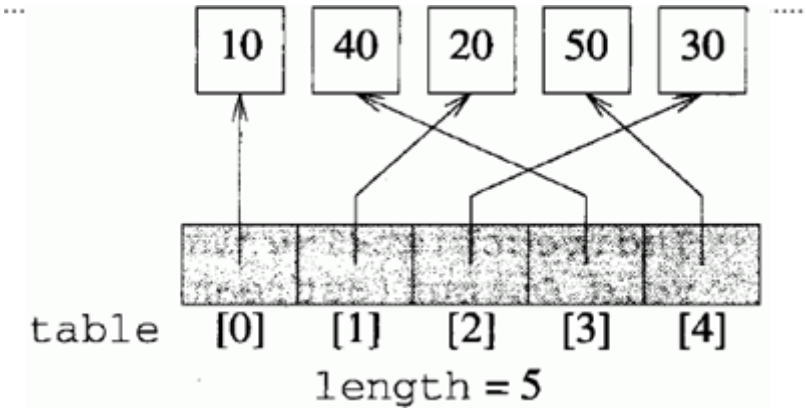
3.2 带哑元的链表：链表前部附加一个头节点

3.3 双向链表：next指针变为left和right两个指针

三.间接寻址

1.结构:

元素的存储为链接方式，通过指针访问，连续元素存储不联系；指针的组织通过公式化方式，连续元素的指针在存储上是连续的。



2.基本操作复杂度

ADT	公式化	单向链表	间接寻址
删除插入	O(length)	O(length)	O(length)
索引访问	O(1)	O(length)	O(1)

3.优点

- 插入删除元素时只需要移动指针，不需要移动数据。

四.桶排序和基数排序

将所有数切位c位r进制数，进行c个步骤，每个步骤对分解后的每一位进行桶排序，在这里，桶排序的稳定性非常重要，需要保证稳定性才能保证基数排序的正确性。

五.课本中扩展知识

1. 凸包

先处理退化情况(所有点共线)，然后极角排序，对排序后的所有相邻三元对进行检查，如果相邻三元对满足逆时针夹角小于等于180度，则舍弃第二个点；反之，遍历下个相邻三元对，最后剩下的所有点即为凸包。

2. dsu

```
//并查集，需要注意询问k所在的组，不是f[k]，而是find(k)
int f[maxn];
for(int i=1;i<=n;i++)
    f[i]=i;

int find(int k){//路径压缩寻找
    if(f[k]==k)return k;
    return f[k]=find(f[k]);
}
void merge(int x,int y){//y合并到x上面
    int a=find(x),b=find(y);
    f[b]=a;
}
```

第4章 数组和矩阵

一.数组

可以重载，为了方便进行向量的运算以及下标越界检测

二.矩阵

1.对角矩阵：可以只用一个一维数组来储存对角线上的非零元素。

2.三对角矩阵：3n-2个元素，考虑逐行映射、逐列映射，**对角线映射（下对角线、主对角线、上对角线）**

3.三角矩阵(对称矩阵)：也是计算出对应的映射函数即可，对称矩阵和三角矩阵差不多。

4.稀疏矩阵(sparse matrix)

数组描述，只保存非0元素。

0	0	0	2	0	0	1	0
0	6	0	0	7	0	0	3
0	0	0	9	0	8	0	0
0	4	5	0	0	0	0	0

a[]	0	1	2	3	4	5	6	7	8
row	1	1	2	2	2	3	3	4	4
col	4	7	2	5	8	4	6	2	3
value	2	1	6	7	3	9	8	4	5

使用稀疏矩阵进行**转置**操作，就是一个行主次序存储变为列主次序存储的过程。假设进行行主->列主，先$O(n)$(n为非零元素个数)统计以下每一列都有多少元素，这样就能知道每一列的一开始的元素要往哪里放，成为$next$数组。进行$O(n)$的放置，每列的元素放过一个就在对应的$next$数组中++即可。

系数矩阵相加类似于两容器按照**行主次序归并**，注意计算结果为0的，不要放在结果矩阵里，其余$O(m+n)$遍历即可

5.十字链表表示稀疏矩阵(linked matrix)

见书P170

注意，每行至少有一个非0元素，才会建立该行的行链表，行链表的节点按列号升序连接在一起。头节点链表的节点按行号升序的顺序连接在一起。

第5章 栈

一.公式化描述

略

二.链表描述

注意**栈顶应该在链表的首节点**，使用单向链表，栈顶在首节点和尾节点的比较如下表：

ADT	PUSH	POP
首节点	等价于Insert(0,x) $O(1)$	等价于Delete(1,x) $O(1)$
尾节点	$O(n)$	$O(n)$

三.课本中扩展知识

1.括号匹配

遇到（就push，遇到）就pop。从左扫描到右，若栈空且需要pop时栈内有未pop出的括号则匹配成功。

2.火车车厢重排

对出轨规则，发现栈底应尽量保留最大值。对于每个缓冲轨，先放入inf方便比较，记录一个now，初始为1。$O(n)$遍历所有火车，如果$a_i = now$，直接出轨，然后$now++$；若$a_i \neq now$，找一个栈顶元素大于a_i的缓冲轨，将其放入即可；若放入的时候，无合适的缓冲轨可放入，则失败。

3.开关盒布线

> 题意：给定圆上n个点及点间的m条连线，问这m条连线是否无交叉？

> 按照统一的顺时针或者逆时针遍历，遍历到连线的起点1先压入栈，遍历到起点2若栈顶不是连线的另一端点，则连线失败，否则，将连线的起点1出栈。

第6章 队列

一.公式化描述

1.队首始终在数组第一个位置

```
队首：front=0,
队尾：rear记录队尾，
队长：rear+1,
判断队列空：if(rear==-1)
push O(1); pop O(n);
```

2.队首也移动

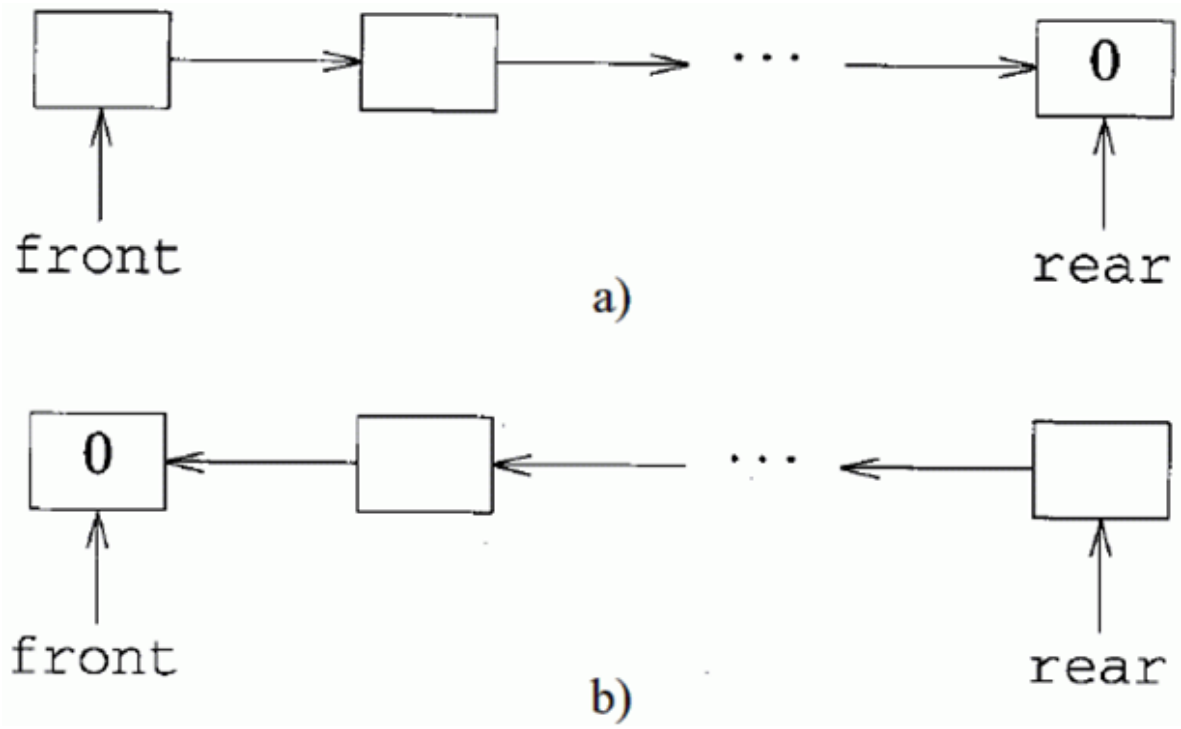
```
loc[i]=front+i-1;
判断队列空：if(front>rear)
pop O(1); push O(n)(不停向右移动，如果触碰到了a[maxsize-1]就会退化)
```

3.循环数组

```
loc[i]=(loc[1]+i-1)%Maxsize;
front=loc[1]-1;
rear=最后一个元素在数组中的位置
push O(1);pop O(1)
这里涉及到一个front==rear无法表达队列空还是队列满的二义性问题，故最简单的解决办法即只允许存放Maxsize-1个元素即可。
判断队列空：front==rear
判断队列满：front==(rear+1)%Maxsize
```

二.链表描述

1.应选择front->near的指针方向



	pop	push
front->rear	O(1)	O(1)
rear->front	O(n)	O(1)

三.课本中扩展知识

1.火车车厢重排

缓冲轨$fifo$，类似$lifo$操作。

2.电路布线

> 题解：裸bfs　$O(n^2)$

3.图元识别

> 题解：对于单色图像上每个标记为1的点跑bfs　$O(n^2)$

第7章 跳表和散列

一.字典(有序表)

1.定义

由一些形如(k,v)的数对所组成的集合，其中k是关键字，v是与关键字k对应的值。任意两个数对，其关键字都不等。

2.线性表表述

```
公式化描述：
搜索：O(logn),朴素二分
插入、删除：O(n)
链表描述：
搜索、插入、删除：O(n)
```

3.跳表表述

见书P240-246

```
搜索、插入、删除：O(logn),平均复杂度；O(n),最差复杂度
```

4.散列表(hashing)描述

$Address = Hash(key)$

4.1 直接定址法

> Hash(key)=a*key+b;

对关键字做一个线性计算，把计算结果当作散列地址

4.2 数字分析法

根据各位不同符号出现的频率特征选择出分布均匀（冲突较少）的位进行哈希映射。

4.3 平方取中法

取关键字平方后的中间几位为哈希地址，取的位数由表长决定。

4.4 折叠法

将关键字分割为位数相同的几部分（最后一部分可以位数不同），然后去这几部分的叠加和（舍去进位）作为哈希地址：

- 可以从左向右分割，也可以从右向左分割；
- 适用于关键字位数很多的情况；
- 一般分割出的位数将于散列表地址位数相同。

> 移位叠加法：把各部分最后一位对其相加；
>
> 间接叠加法：各部分不折断，沿各部分分界来回折叠，最后对其相加。

4.5 除留余数法

$H(key) = key \, mod \, p \quad , p \leqslant m$

> p一般取：最接近m的质数，或者不包含小于20的质因数的合数

二.散列表(hashing)表示法中处理冲突

1.线性探测法

使用hash函数计算出初始散列地址，一旦发生冲突，在表中顺次向后寻找下一个空闲位置。

$H_i = (H_{i-1} + d)\%m$，$d = 1$

注意！插入和查找操作按照上述步骤做，但删除操作，不能简单删除，因为插入的时候如果引起了冲突，简单删除会影响后续搜索操作！

U_n :一次不成功搜索平均检查的桶的数目；

S_n : 一次成功搜索平均检查的桶的数目。

$U_n = 1/2[1 + 1/(1-\alpha)^2]$

$S_n = 1/2[1 + 1/(1-\alpha)]$

$\alpha = n/b$：记为负载因子，表示hash表满的程度(长为b的表中有n个记录)。

2.二次探测法

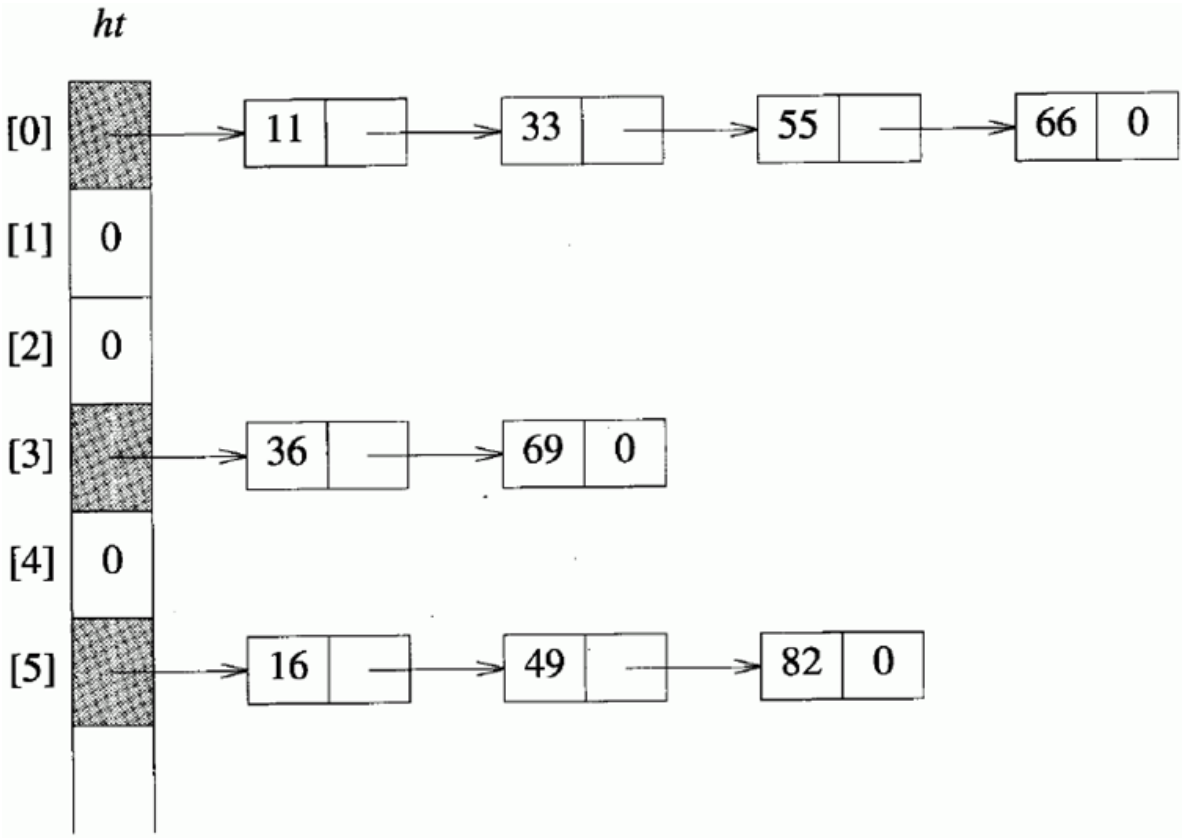
$H_i = (H_{i-1} + d)\%m$，$d = i^2$

解决了局部聚集问题，但在表不满的情况下，也不能保证插入成功。

3.双散列法

需要两个hash函数，第一个hash函数计算出关键字的首选地址；一旦发生冲突，用第二个hash函数计算出到下一地址的增量，或者直接计算下一个地址。

4.链式散列法



对于每一个桶，保留一个链表。

> 一些优化：在每个链表的结尾加上一个尾节点，关键字用inf表示。

$U_n = (\alpha + 1)/2, \alpha \geq 1$(不成功平均搜索次数)

$S_n = 1 + \alpha/2$

$\alpha = n/m$(平均链表长度)

三.课本中扩展知识

1. LZW压缩

见书P260-P268

第8章 二叉树和其他树

一.基本概念

1.树：是一个**非空的**有限元素的集合，其中一个元素为根，其余的元素构成t的若干子树。

2.元素：节点；

3.边：根节点与其子树根节点的关系；

4.边的两端：父母和孩子；

5.兄弟：相同父母的一对节点；

6.叶节点：没有孩子的节点；非叶节点：非终端节点。

7.级：树根是1级（也有的规定树根为0级），以此类推；

8.树的高度(或深度)：树中级的个数；

9.元素的度：**该元素孩子的个数**；

10.树的度：书中**所有元素**的度的最大值。

11.自由树：若G是一个无向图，下面的描述是等价的：

- G 是自由树；
- G 中任何两顶点由唯一简单路径相连；
- G 是连通的，但从图中移除任意一条边得到的图均不连通；
- G 是连通的，且 $|E| = |V| - 1$；
- G 是无环的，且 $|E| = |V| - 1$；
- G 是无环的，但如果向 E 中添加任何一条边，均会使得图中包含一个环；

12.森林：树的集合，通常认为是有根树的集合；有序森林：有序树的有序集合；

13.有根(有序)树去掉根节点->(有序)森林；(有序)森林添加父节点->有根(有序)树

14.“度”定理

若一棵树有n_1个度为1的节点，有n_2个度为2的节点，...，有n_m个度为m的节点，求有多少个度为0的节点?

$n_0 = (\sum_{i=1}^{m}(i-1)\cdot n_i) + 1$

15.二叉树和树的区别

- 二叉树是有限元素集合，**或者为空**；（而树是非空的）
- 二叉树每个节点都恰好有两棵子树（可以为空），树中每个节点可有若干子树；
- 二叉树每个节点的子树是有序的，树的子树间是无序的。

16.二叉树的特性

- 包含$n(n>0)$个节点的二叉树边数为$n-1$
- 若二叉树的高度为h,$h \geq 0$,则它最少有h个节点，最多有$2^h - 1$个节点
- 包含n个节点的二叉树高度最大为n，最小为$ceil(log_2(n+1))$

17.满二叉树

当高度为h的二叉树恰好有$2^h - 1$个元素时，被称为满二叉树。

18.完全二叉树

对高度为h的满二叉树的元素的最后一层，删去k个末尾元素，得到的树即为完全二叉树。
$(1 \leq i \leq k < 2^k)$

19.在完全二叉树中任意节点有记其序号为$i(1 \leq i \leq n)$：

当$i = 1$时，为二叉树的根；若$i > 1$，$i/2$为该元素父节点编号；

当$2i > n$，无左孩子；否则，编号为$2i$；

当$2i + 1 > n$，无右孩子；否则，编号为$2i + 1$；

二.二叉树描述

1.公式化描述：根节点i=1，左孩子i<<1，右孩子i<<1|1，父亲i>>1

相对完全二叉树缺少的元素比较少，才不会造成较大的空间浪费。

2.链表描述

```
template<class T>
struct btnode
{
    T element;
    btNode<T> *leftchild,rightchild;
}
```

三.二叉树的遍历

1.先序遍历(preorder)：根左右

2.中序遍历(inorder)：左根右

3.后序遍历(postorder)：左右根

4.注意：

去还原一棵树，给定先+后+中，可以唯一还原；先+中，后+中也可以，但先+后不可以(左链和右链的反例易知)。

对于表达式树的遍历，先序遍历的能得到前缀表达式；后序遍历能得到后缀表达式；中序遍历即常用的中缀表达式。其中前缀和后缀表达式没有歧义，不需要加括号，计算时，模拟一个辅助栈就可以。中缀表达式受到优先级影响，需要加适量括号，用中序遍历可以得到完全括号化的中缀表达式。

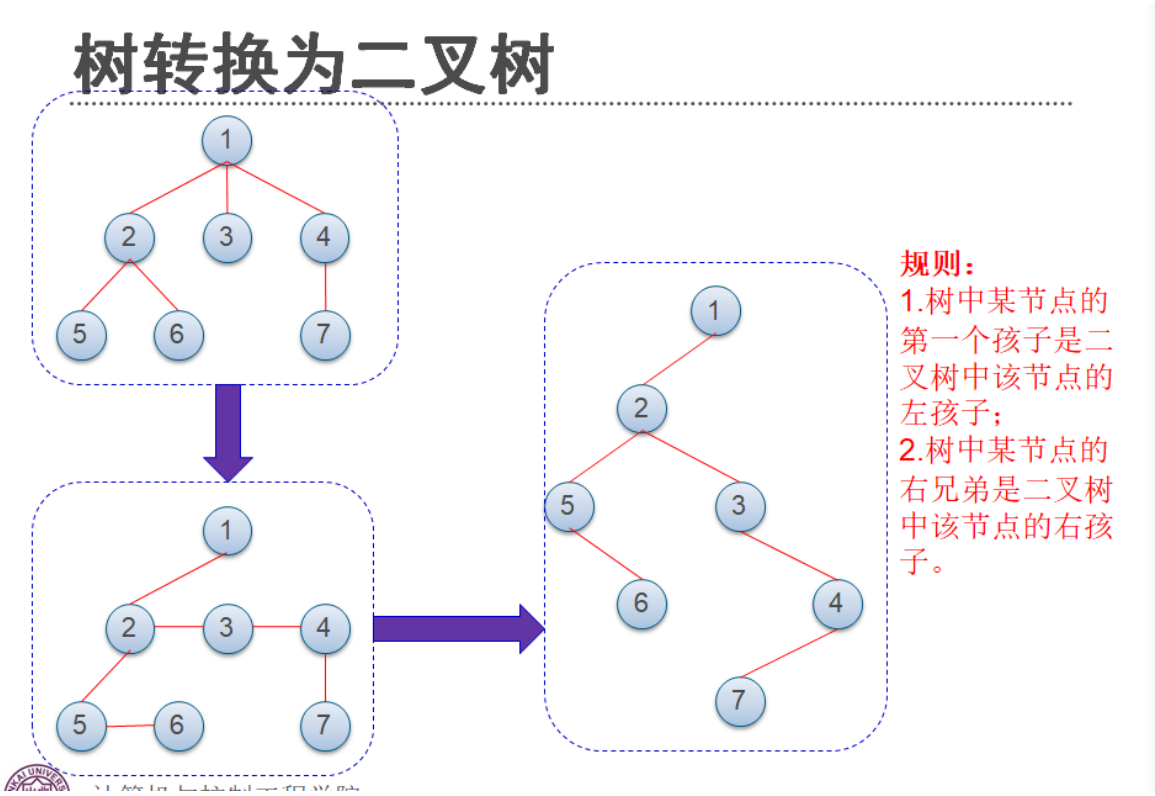
5.确定树高：递归遍历即可

```
h=max(tl.h,rl.h)+1;
```

四.树和二叉树互转，森林和二叉树互转

树的父指针表示法->树的左孩子右兄弟表示法

1.树和二叉树互转



2.森林和二叉树互转

先将森林中的每一棵树转化为二叉树；

- 再将第一棵二叉树的根作为转换后二叉树的根；

- 第一棵二叉树的左子树作为转换后二叉树的左子树；

- 第二棵二叉树作为转换后二叉树的右子树；

- 第三棵二叉树作为转换后二叉树根的右孩子的右子树

- 依此类推

五.线索二叉树

对于先序遍历线索二叉树，前驱不好找；后序遍历线索二叉树，后继不好找。

建什么样的线索树，就用什么样的遍历遍历一遍即可。

左指针指的有可能是左孩子或者前驱；右指针指的有可能是右孩子或者后继。

第9章 优先队列

一.基本概念

1.大根树（小根树）

一棵树，其中每个节点的值都大于(小于)或等于其子节点(如果有子节点的话)的值。注意，不一定是二叉树，节点的子节点个数可以任意。

2.大根堆（小根堆）

一个大根堆（小根堆）既是大根树（小根树）也是**完全二叉树。**

3.父子节点位置关系

(当前节点记为i)	左孩子	右孩子
从0开始编号	$2i$	$2i+1$
从1开始编号	$2i+1$	$2i+2$

二.堆的基本操作(以最大堆为例)

1.插入($O(logn)$):从下向上调整

```
由于是完全二叉树，插入操作一定一开始放在第一个非空位置，但是显然，不一定满足最大堆性质。
这时要从该叶节点向根节点递归，检查该节点的父节点是否满足ele[now>>1]>=ele[now]，
    如果满足，停止；
    不满足，则交换，即swap(ele[now>>1],ele[now]);
然后更新now=now>>1;
直至更新到根节点结束。
```

2.删除($O(logn)$):从上向下调整

大根堆只能删除最大的元素，即根节点。这时先保留一下最后一个元素(不一定是最小的)，记为$temp=ele[max_size]$，移除根节点元素后，**判断temp插入该节点能否构成最大堆**：如果可以就执行$ele[now]=temp$;，并结束循环；如果不满足，就把两个孩子中大的移上来，更新新空位，直至到一个叶节点。

```
if(ele[now>>1]>ele[now>>1|1]) now=now>>1;
else now=now>>1|1;
```

3.创建堆 (两种实现方法)

3.1 (直接插入n次) $O(nlogn)$

3.2 (重整) $\theta(n)$

从所有非叶节点检查(因为所有的叶节点没有子树，就不用检查了！)，以该节点为根的子树是否为最大堆，即直接询问

```
if(ele[now]>max(ele[now>>1],ele[now>>1|1])) f(1)...
else f(2)...
```

其中，f(1)即为break，说明以该节点为根的子树已经为最大堆；

其中，f(2)为交换该节点元素和两个孩子中大的元素，**接着要再去向下判断交换过的子节点为根节点的子树是否为最大堆。**

3.3 重整方法的复杂度推导

见书P305 $O(n),\Omega(n)=>\theta(n)$

三.堆排序

先建堆$O(n)$，pop n次即可。

四.哈夫曼树($Huffman$ tree)

1.一些定义

- 对于一棵具有n个外部节点的扩展二叉树，且外部节点标记为1,2,...,n，其对应的编码位串长度为：$WEP=\sum_{i=1}^{n}L(i)*F(i)$

(其中$L(i)$为从根到外部节点i的路径长度；$F(i)$为字符x的出现频率)

- 一棵二叉树，如果对一组给定的频率，其WEP最小，则这颗二叉树成为哈夫曼树。

2.构造过程 $O(nlogn)$

见书 $P319$，其中用到$priority_queue$来维护动态最大值。

3.一些性质

- 任何一个编码都不是其他编码的前缀
- 从根到叶的路径(左0右1)即是该叶的编码
- 哈夫曼树中度为0的节点有n个，度为1的有0个，度为2的有$n-1$个。

第10章 BST AVL

一.BST

1.定义

- 是一颗二叉树，可能为空，如果非空应该满足：

  （1）每个元素有一个关键值，并且没有任意两个元素有相同的关键值，所有的关键值都是唯一的；

  （2）根节点左子树的关键值小于根节点的关键值；

  （3）根节点右子树的关键值大于根节点的关键值；

  （4）根节点的左右子树都是二叉搜索树

2.索引二叉搜索树

每个节点记录一个索引值：该节点左子树节点个数+1

方便解决的问题：求第k大元素

3.搜索 $O(logn)$

- 从根节点开始将key与节点值比较：如果key小于节点值，进入左子树；如果key大于节点值，进入右子树；直至找到或子树为空时停止。
- $ASL_{成功}$ = **根节点到所有内节点长度的平均值**；
- $ASL_{不成功}$ = **根节点到所有外部节点长度的平均值。**

4.插入 $O(logn)$

- 需要先进行搜索操作，若成功，表明有重复关键字，则插入失败；若失败，搜索结束的位置即为插入位置。
- 新插入的节点一定是一个**叶子节点**，并且是查找不成功时路径上访问的最后一个节点的左孩子或右孩子节点。

5.删除 $O(logn)$

- 若删除的为叶节点，直接丢弃：父节点指向它的指针置为NULL
- 若删除的不是叶节点，且有且只有一个非空子树t，其根为q，直接丢弃p，用q取代p的位置。这里给p的父节点的孩子指针修改的时候，**注意判断p是否为根**，若为根，把q作为新的根；若不是根，就把p的父节点指向p的指针变为指向q
- 若删除的不是叶节点，且两个子树都不为空，找直接前驱（左子树的最右元素）或直接后继（右子树的最左元素）做key值替换，改为删去直接前驱或者直接后继。 **（一般惯例选用直接前驱）**

二. AVL

1.定义

- 可能为空，若非空，T_l, T_r分别为左子树和右子树，满足：T_l, T_r也是AVL；且$|h_l - h_r| \leqslant 1$。
- 平衡因子：某个节点左子树的高度—右子树的高度（在稳定的树中只可能是$-1, 0, 1$）

2.一个问题：高度为h的AVL的最少节点数

$|F_h| = |F_{h-1}| + |F_{h-2}| + 1$

将等式两遍均加1，可以得到标准$fibonacci$数列。

3.搜索

同BST搜索

4.插入

- 插入导致的不平衡树的特性：

（1）不平衡树中的平衡因子值限于-2，-1，0，1，2

（2）平衡因子为2的节点，在插入前平衡因子为1；平衡因子为-2的，插入前为-1

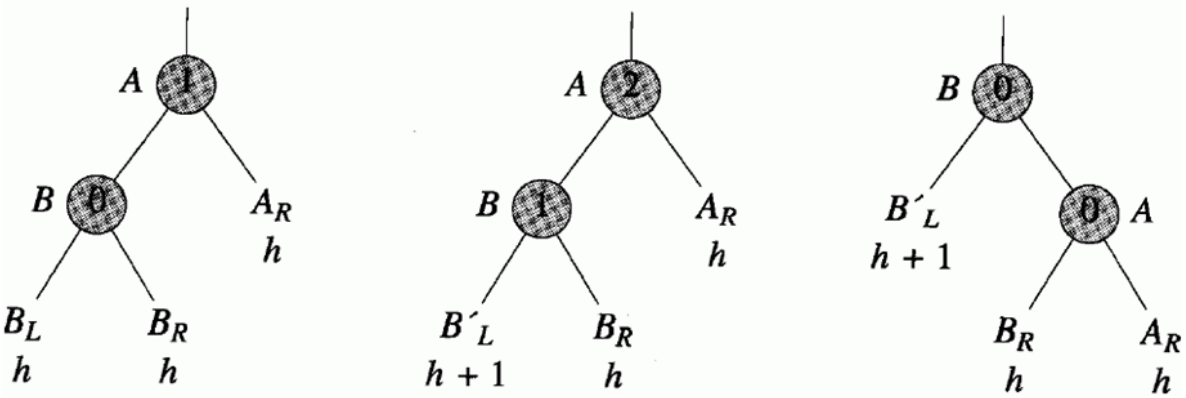
（3）只有从根到新插入节点路径上的节点，平衡因子才会在插入操作后改变

（4）假设A是距离新插入节点最近的，平衡因子为-2或2的祖先节点，则在插入前，从A到新插入节点的路径上，所有节点的平衡因子均为0

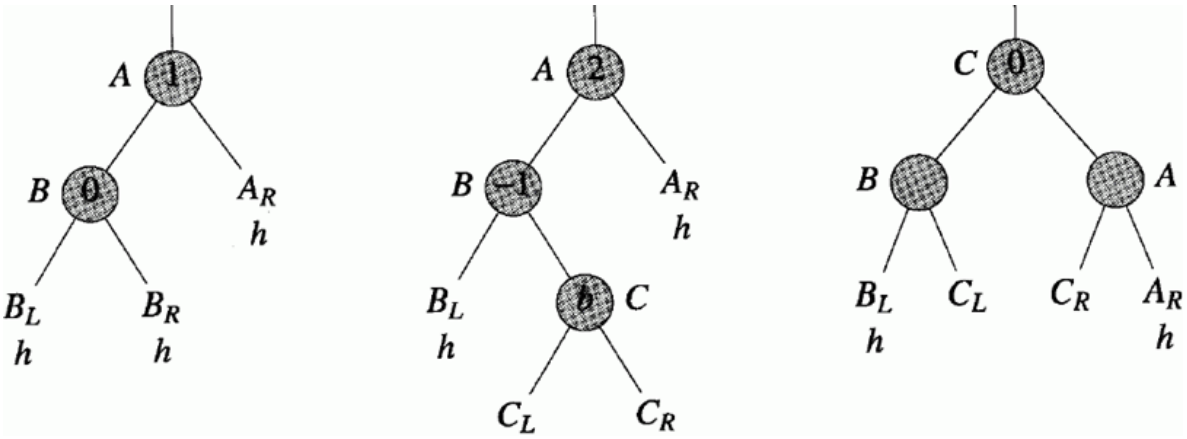
- X未变为A：插入后仍然平衡；
- X变为A：

（1）L型不平衡：左子树较高，新节点插入左子树：

LL：新节点插入左子树的左子树；（一次右旋）



LR：插入左子树的右子树；（对B一次左旋，对A一次右旋）



（2）R型不平衡：右子树较高，新节点插入右子树：

RL：插入右子树的左子树；

RR：插入右子树的右子树。

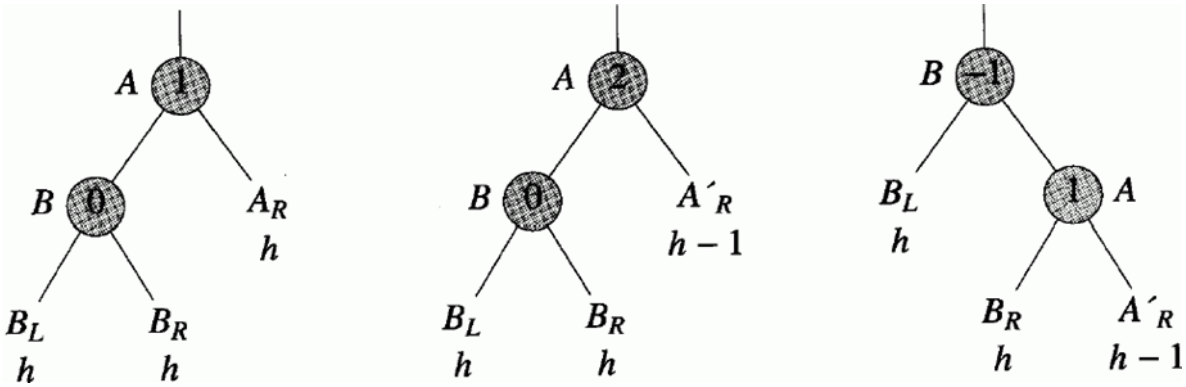
5.删除

- 先按照BST的删除方式，如果不平衡再按照如下规则进行调整：

记q为被删除节点的父亲，如果删除后bf(q)=0，子树平衡，无需调整，但高度改变，需要改变祖先节点平衡因子；若bf(q)=+1/-1,子树平衡，高度不变，无需调整；若bf(q)=+2/-2，子树不平衡。

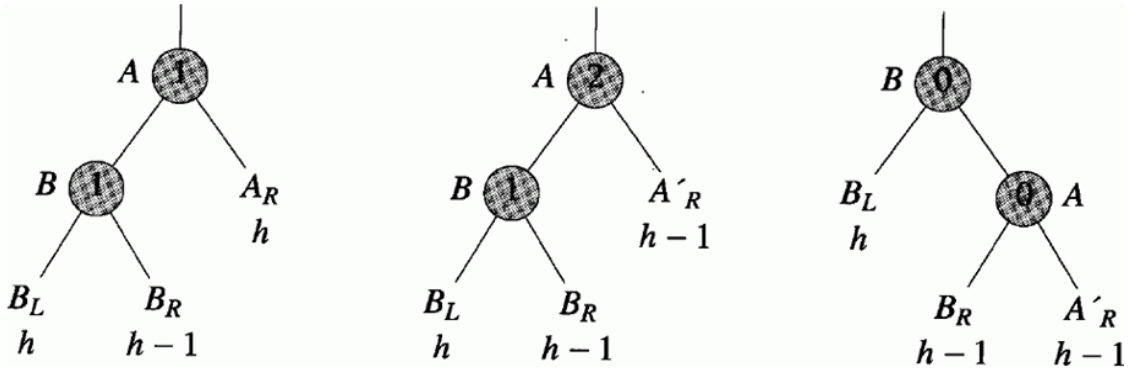
记从q到根路径上第一个新平衡因子为+2/-2的节点为A，假定删除发生在q的右子树，R型不平衡。记q的左子树为p。

- 三种不平衡情况的唯一差异：**不平衡节点的另一颗子树高度!**
- $bf(p)=0$,R_0不平衡



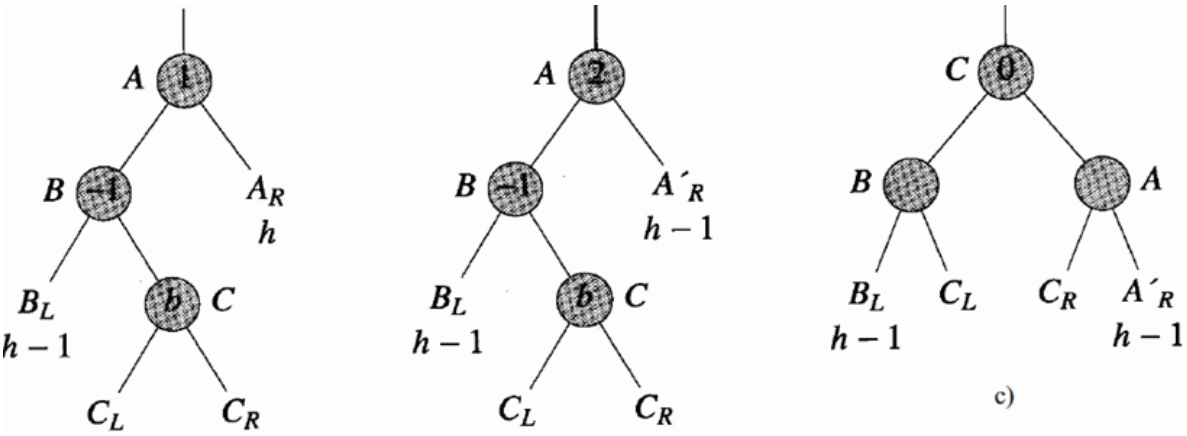
高度不变，祖先平衡因子不用调整，一次右旋。

- $bf(p)=1$,R_1不平衡



高度减少1，继续修正过程。一次右旋。

- $bf(p)=-1$,R_{-1}不平衡



高度减少1，继续修正过程，对B一次左旋，对A一次右旋。

- 注意R_1,R_{-1}都需要修改新根到根路径上所有点的平衡因子，L_1,L_{-1}同理，只不过旋转次数不同（R_{-1}和L_1需要双旋转）。

第11章 BT RBT

一. m叉搜索树

- 可以是一棵空树，如果非空，则：
- 在相应的扩充搜索树中(用外部节点替换零指针)，每个**内部节点最多可以有m个子女及1~$m-1$个元素**(外部节点不含元素和子女)；
- 每个含p个元素的节点，有$p+1$个子女

- 对于含p个元素的任意节点，$k_1, k_2, \ldots, k_p$是这些元素的关键字，元素按关键字**升序**排列，有$k_1 < k_2 < \ldots < k_P$，在以c_i为根的子树中的元素关键值均大于k_i而小于k_{i+1}，对于$1 \leq i \leq p$成立。
- 对于插入，如果待插入节点元素数<m-1，直接插入即可；若$\geq m-1$，则生成新孩子节点。对于插入，可以从左非空子树的最右或右非空子树的最左元素替换后删除。

二. m阶$B-$树

> m阶B-树是一棵**m叉搜索树**，如果B-树非空，那么相应的扩充树满足下列特征：
>
> 1.根节点至少有2个孩子
>
> 2.除根节点，所有内部节点至少有$ceil(m/2.0)$个孩子；
>
> 3.**所有外部节点位于同一层上**

二阶B-树：满二叉树；

三阶B-树：2-3树；

四阶B-树：2-3-4树。

1.搜索

同m叉搜索树，从根到外部节点路径上所有的内部节点都有可能被搜索到，所以，磁盘访问次数最多为h(h为$B-$树的高度)。

2.插入

首先检查树中是否已经有该key，若已有，则插入失败。

否则，寻找要插入的位置，一定在叶节点上，如果该节点插入前有$m-1$个key，则直接插入即可；

否则，先插入，则该节点拥有m个key，但这样该节点就有$m+1$个孩子了，不满足B-树性质！这时候需要提取该节点最中间的key(记为mid_key)上升到它的父亲节点，然后，把mid_key左侧的key值组成的节点和mid_key右侧的key值组成的节点分裂成两个，作为父节点的两个新孩子。此操作之后，该节点的父节点元素个数+1，有可能又不满足B-树基本性质，直至递归修改到一个合法节点，停止。

最差情况会进行h(读取搜索路径上的节点)+$2s$(回写分裂出的两个新节点)+1(回写新的根节点或插入后没有导致分裂的节点)，即，最多可以达到$3h+1$次磁盘访问。

3.删除

所有的删除可分为删除在叶节点上的元素或非叶节点上元素(直接按照BST删除方式转化为删除为叶节点上的元素)

下面只解释删除叶节点上的元素。

如果删除前，元素数目$> d-1$，直接删除即可；

如果删除前，元素数目$= d-1$，删除后小于限制值；

(1) 先考虑能否借用兄弟节点的key，可以就直接借用；

(2) 否则，将该节点和其中一个兄弟节点的所有key，**以及父亲节点位于这两个节点之间的那个key值**，合并成一个节点，层数位于原节点层。这样的新节点不会超过$m-1$个key值的最大限度。但是，这样显然会使原节点的父亲节点减少一个key值，导致父亲节点有可能也不满足B-树基本性质，最多需要从操作节点到根节点操作h次。**直至遇到一个操作后，合法的节点，或根节点被删除后为空，这时需要删除空的根节点，特别地，此时树高-1。**

最差情况磁盘操作次数为$3h$次

三. RBT

1.定义

- 特殊的二叉搜索树，对扩充的二叉搜索树满足：

R_1:根节点和所有外部节点的颜色是黑的；

R_2:根至外部节点的路径上没有连续红节点；

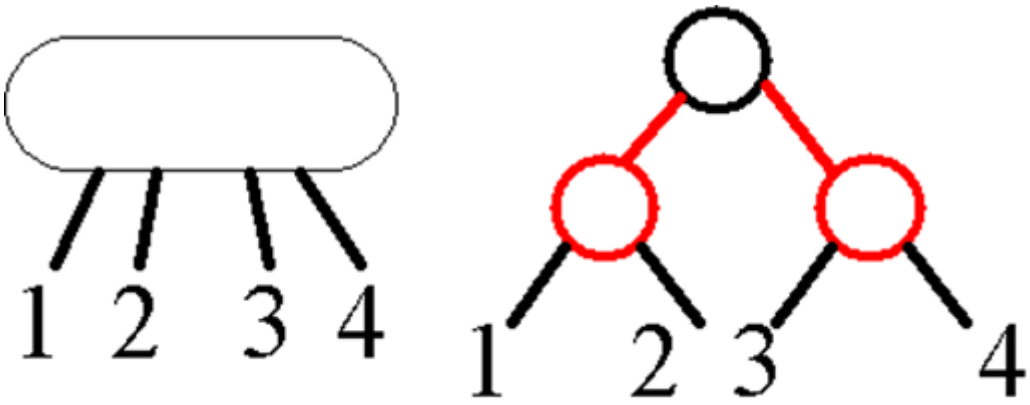
R_3:所有根至外部节点的路径具有相同数目的黑节点。

R:黑指针指向的孩子节点是黑的；红指针指向的孩子节点是红的。

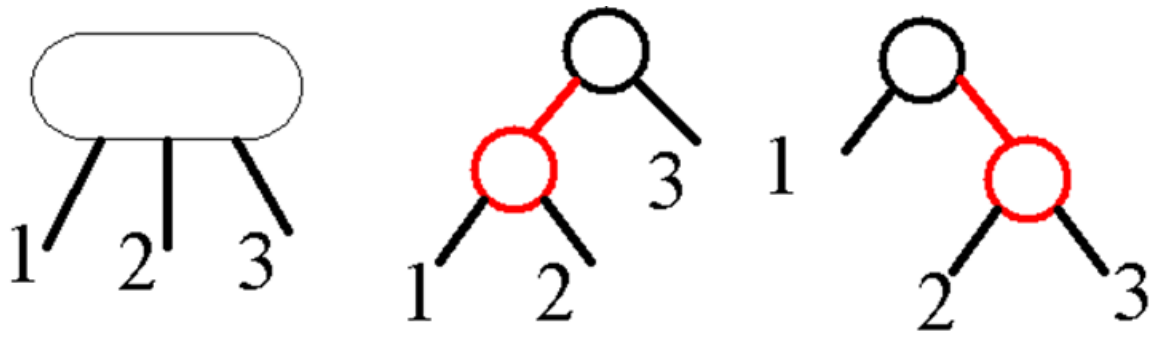
- 节点的阶：从该节点到其子树中任一外部节点的路径上的黑色指针的数目。

2. RBT和$2-3-4$树之间的关系

> 四节点的转换：



> 三节点的转换：

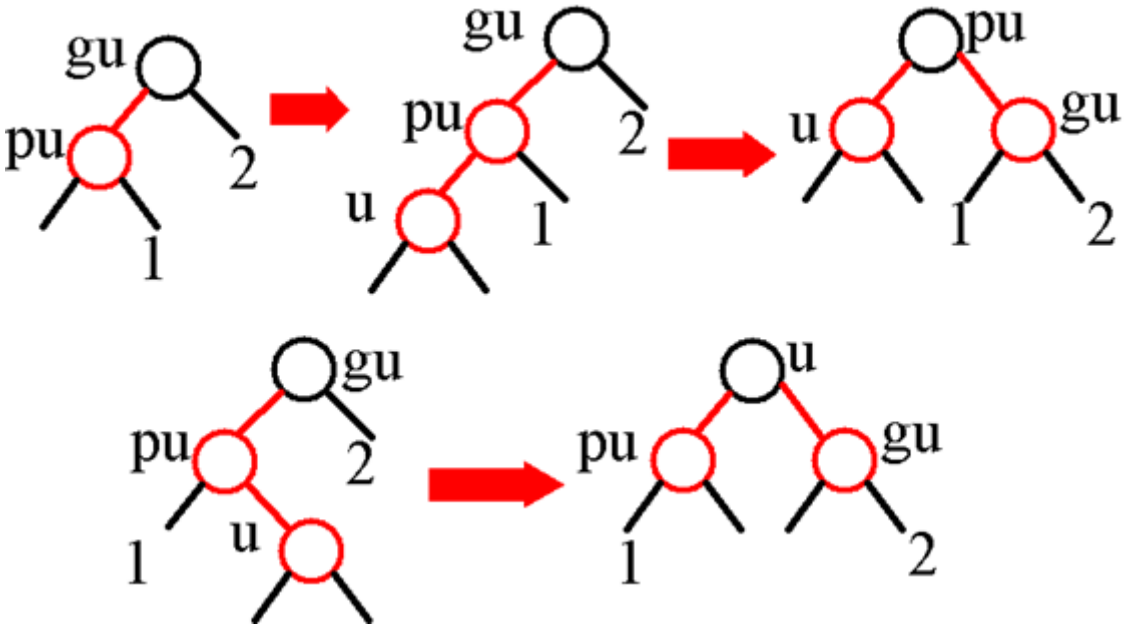


3.搜索 O(logn)

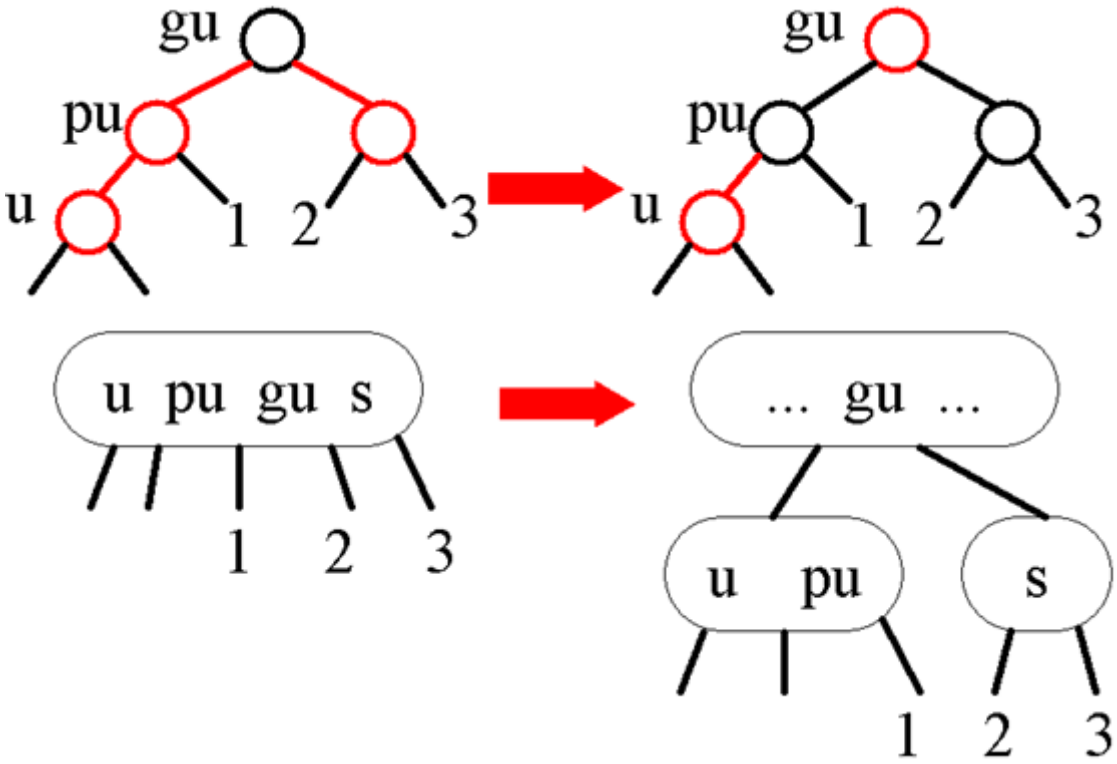
同BST

4.插入

> 连续红边情况：XYb，祖父的另一个孩子是黑色，旋转变色：根黑、子红



> 连续红边情况：XYr，祖父的另一个孩子是红色，不转变色：根红、子黑



第12章 图

一.基本概念

1.当且仅当(i,j)是图的边，称顶点i和j是**邻接**的，**边**(i,j)**关联与顶点**i**和**j

2.有向边(i,j)是**关联至**j，而**关联于**顶点i；顶点i邻接至顶点j，顶点j邻接于顶点i

3.根据定义，**一个图不能有重复的边，无向图两个顶点间最多只有一条边，有向图两个顶点也最多只有两条相反方向边。图不包括自连边，也没有环。**

4.简单路径：一条路径，除第一个和最后一个顶点之外，其余顶点均不同

5.无向图联通：当且仅当G的每一对顶点之间都有一条路径。

6.子图：图H的顶点和边的集合分别是图G的顶点和边的集合的子集。

7.环路：一条始点和终点相同的简单路径称为环路。

8.树：没有环路的无向连通图。

9.生成树：G的子图，若包含G的所有顶点，且是一棵树，则为G的生成树。

10.二分图：若A和B构成点集的全集，每条边都有一个点在A中，另一个点在B中。

11.顶点v的度：与v关联的边的数目。

12.有向图中顶点v的入度：关联至v的边的数目；出度：关联于v的边的数目。

13.完全图：一个具有n个顶点和n(n-1)/2条边的无向图是一个完全图。

14.连通分量：无向图中的极大联通子图。

15.强连通图：若有向图中任意两个顶点v_i, v_j，从v_i到v_j和从v_j到v_i均有路径，则G是强连通图。

16.强连通分量：有向图中的极大联通子图。

二.图的表示

1.邻接矩阵：

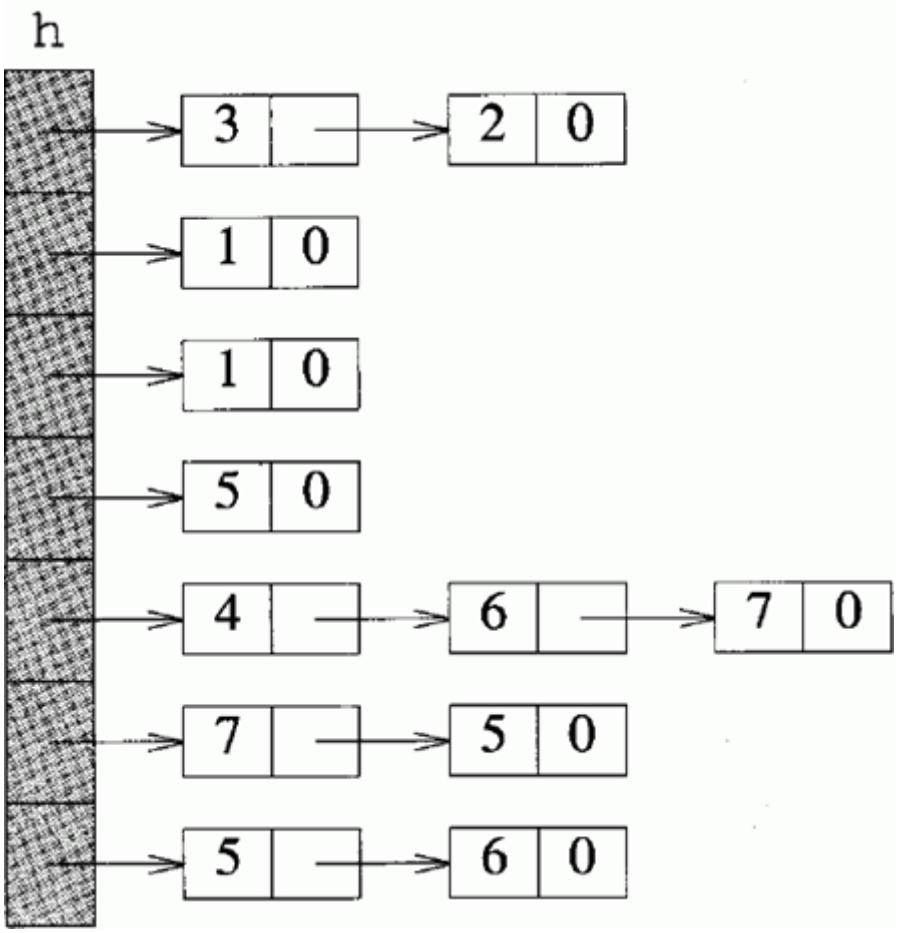
ADT	复杂度
求入度、出度或邻接节点集合	$O(n)$
求图中总边数	$O(n^2)$
增、删边	$O(1)$

2.邻接压缩表：

压到一位数组里，记录每个节点边开始的索引（节省了一些空间）

ADT	复杂度
求入度、出度	$O(1)$
增、删边	$O(n+e)$

3.邻接链表：



求入度不方便，求出度方便。

三.最小生成树

1. Kruskal

对所有边排序，费用从小到大选边，如果产生环路则舍弃，不产生环路直接加入生成树边集中。

2. Prim

从一个起始顶点开始，维护一个min数组，每次加入新的边都去更新这个min数组，从min数组中选取两端点未完全被访问过的边去更新。注意，每次加入的边，一定是一个顶点被访问，而另一个没有。实现这个可以额外记录一个vis数组。

四.最短路

1. Dijkstra

需要返回，从源点到每个终点的最短距离和具体路径。

全过程需要维护两个数组$d[]$(当前源点到各点的最短距离), $pre[]$(当前点最短路径的前驱)。首先初始化$d[i]=a[s][i](1\leq i\leq n)$，对于邻接于s的所有顶点i，置$pre[i]=s$,其余点置为$pre[i]=0$，对$pre[i]!=0$的所有顶点建立L；若L为空，则停止；否则从L中删除d值最小的顶点，更新与被删掉的顶点相连的边，$d[j]=min(d[j],d[i]+a[i][j])$,若d[j]发生了变化并且不再L中，则修改前驱$pre[j]=i$，并把j放入L，继续判断L是否为空。

2. Floyd

话不多说，直接for,for,for

```
for (k=0;k<n;k++)
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
        if (min[i][k]+min[k][j]<min[i][j])
        min[i][j]=min[i][k]+min[k][j],pre[i][j]=pre[k][j];
```

五. Topology Sort

1.在AOV(顶点活动网络，用顶点表示活动，用箭头表示活动间优先关系)网络中，顶点i在顶点j之前，意味着活动i是活动j的先决条件。不能出现有向环。

从有向图中选出一个没有前驱的顶点输出，从图中删除该点和所有从该点输出的边，重复前两步，直至所有顶点已输出，或图中不存在无前驱的顶点为止。

维护入度为0的点用栈实现。

使用邻接链表把复杂度降到$O(n+e)$

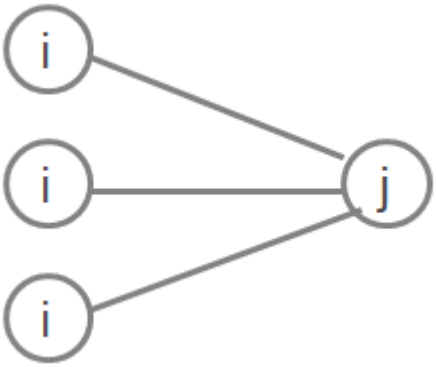
六.关键路径

1.在AOE(带权的有向无环图，其中顶点表示事件，边表示活动，权表示活动持续的时间)中：

（1）关键路径：从源点到汇点长度（权值）最长的路径，长度是完成总工程的最短时间。

（2）最早开始时间：e(i)

> 由$ve(0)=0$向前递推 $ve(j)=max(ve(i)+dut(<i,j>)),<i,j>\in E$
>
> 事件j的开始依赖于所有活动<i,j>的完成，显然应该取其中“最差”者：
>
> j再早也不会比最慢的那个早
>
> 再通过$e(i)=ve(j)$计算每个活动的$e(i)$.



（3）最迟开始时间：l(i) （前提：不影响工程进度）

> 由$vl(n)=ve(n)$向前：$vl(i)=min(vl(j)-dut(<i,j>)),<i,j>\in E$
>
> 所有的j都依赖于i，i再迟也不能影响j的启动，不能影响工期，应该取其中最早的。
>
> 再通过$l(i)=vl(k)-dut(<j,k>)$计算每个活动的$l(i)$.

（4）关键活动：$l(i)=e(i)$

Step1：从源点开始计算ve(i)，考察指向顶点i的所有边，寻找最大值ve(j)=max{ve(i)+dut(<i，j>)}，<i，j>∈E
Step2：从汇点开始计算vl(i)，考察顶点i发出的所有边，寻找最小值vl(i)=min{vl(j)-dut(<i，j>)}，<i，j>∈E

Step3：求每个活动的e(i)，等于活动发出顶点的ve值
Step4：求每个活动的l(i)，等于活动指向顶点的vl值减去活动本身的持续时间

Step5：找到那些l(i)=e(i)的活动，即为关键活动；构成的路径即为关键路径。关键路径可能不止一条。

七.应用

1.构件标记问题：

在一个无向图中，从一个顶点i可到达的顶点集合C与连接C的任意两个顶点的边称为连通构件。

用邻接矩阵描述图时，复杂性为$O(n^2)$，用邻接链表只为$O(n+e)$

方法就是：保存一个vis数组，对于当前所有vis数组中没有标记过的点做标记，对于一个连通构件使用bfs或dfs。

EX_1 排序和查找
