期中部分题目讲解

1、现有一个由单链表和循环单链表构成的特殊链表，单链表的头元素是head，末尾元素为tail，head->…->tail即是一条普通的链表，同时tail元素还是另一条循环单链表的头元素。如A->B->C->D->E->F->G->H->E就是一个特殊链表，其中head为A，tail为E。

现在你只知道head，但是你想知道这个循环链表中有多少元素，但同时你的剩余空间十分的小，所以你想用尽可能小的空间来解决问题，请问你会怎么做？

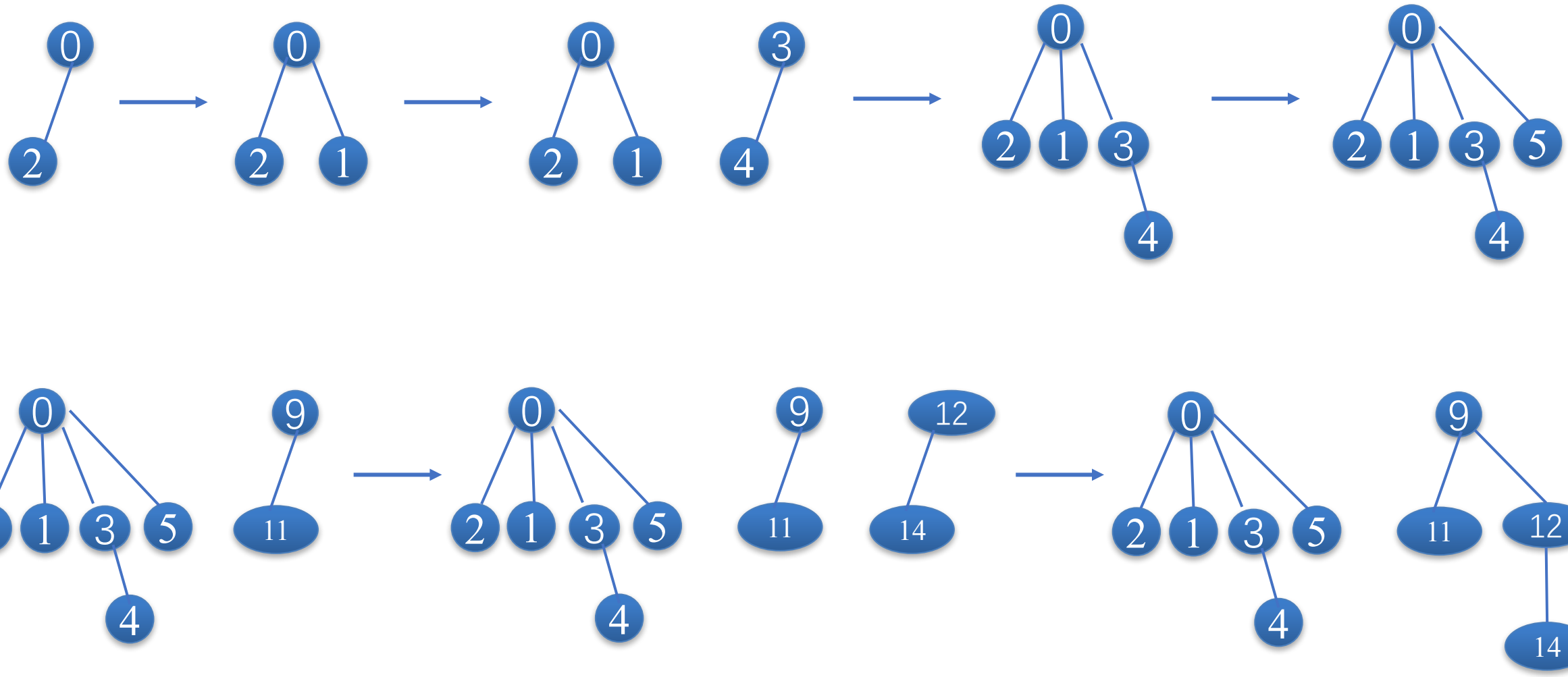
（1）朴素：从head开始遍历，使用一个链表记录已访问过的节点，然后每次到新的节点就去查找该节点是否已经访问过，若访问过，就找到了tail节点。 空间占用O(n)。

（2）快慢指针：同时记录两个节点s和t，从头结点开始，每次s走一步，t走两步，当s和t第一次相遇时，s就是循环链表中的元素，且如果在第n次相遇，循环链表中元素个数就是n

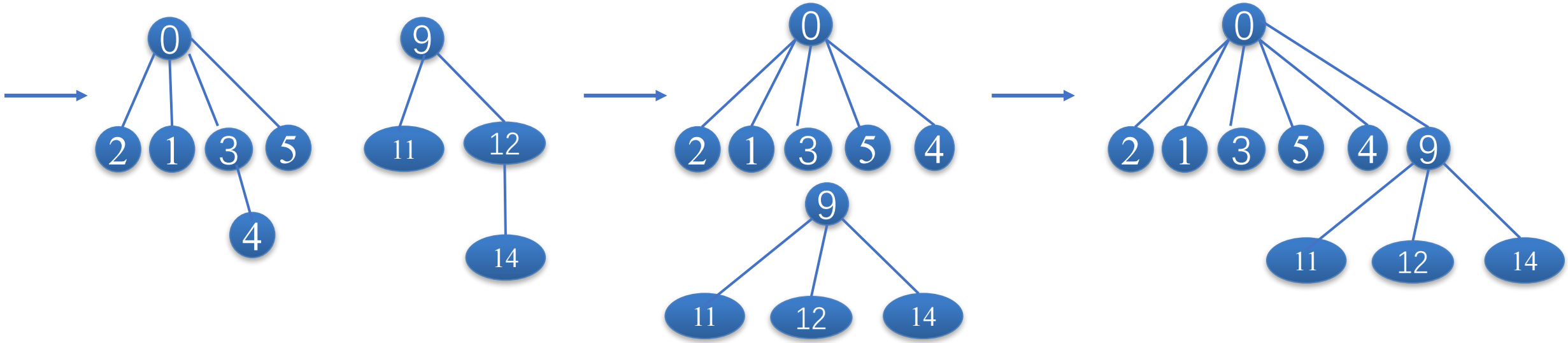
2、对下列15个等价对进行合并，给出所得等价类树的图示。在初始情况下，集合中的每个元素分别在独立的等价类中。使用重量权衡合并规则，合并时子树结点少的并入结点数多的那棵（多的那个作为新树根，少的那个根作为新根的直接子结点）；若两棵树规模同样大，则把根值较大的并入根值较小（新树根取值小的）。同时，采用路径压缩优化。

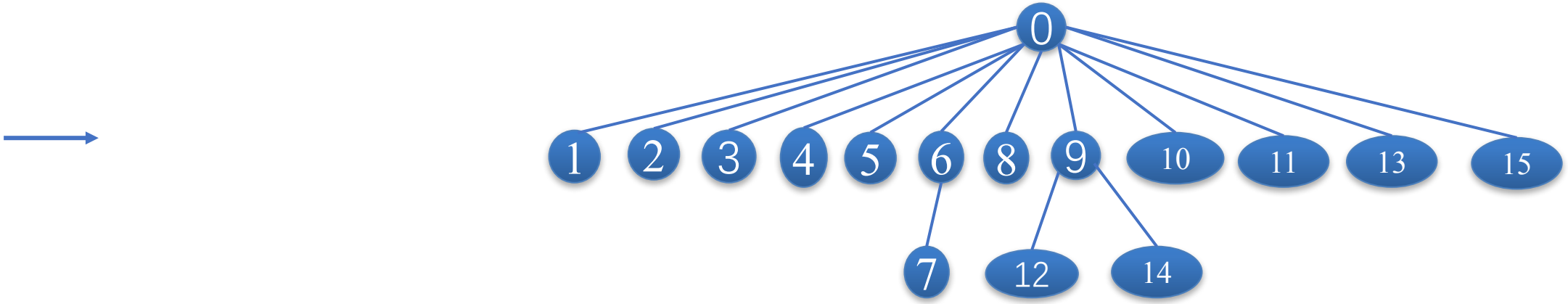
(0,2) (1,2) (3,4) (3,1) (3,5) (9,11) (12,14) (12,9) (4,14) (6,7) (8,10) (8,7) (7,11) (10,15) (10,13)

(0,2) (1,2) (3,4) (3,1) (3,5) (9,11) (12,14) (12,9) (4,14) (6,7) (8,10) (8,7) (7,11) (10,15) (10,13)



(0,2) (1,2) (3,4) (3,1) (3,5) (9,11) (12,14) (12,9) (4,14) (6,7) (8,10) (8,7) (7,11) (10,15) (10,13)





3、下面的算法利用一个栈将一给定的序列从小到大排序。长度为n的序列由1, 2, …, n这n个连续的正整数组成。请补全下面的代码段，使其可以判断给定的序列是否可以利用一个栈进行排序，如果可以，输出相应的操作。

```
Stack<int> s;                                  //  s为栈
int sequence[maxLen],  len;                    // sequence为待排序序列，  len为序列长度
bool islegal()  {                              // 判断序列是否可以排序
    int i, j, k;
    for (i = 0; i < len; i++) {
        for (j = i+1; j < len; j++) {
            for (k = j+1; k < len; k++) {
                if ( sequence[k] < sequence[i] && sequence[i] < sequence[j] )
                 return false;
            }
        }
    }
    return true;
}
```

i	j		k
2	3	4	5 1

```
void transform() {                                  // 输出转换操作
    int cur = 1, i = 0;          //cur表示排好序输出的元素个数
    while ( //填空2 ) {
        while (s.isEmpty() || //填空3 ) {
            //填空4
            cout << "push ";
            if (cur == s.top()) break;
        }
        s.pop();
        cout << "pop ";
        ++cur;
    }
}
```

① sequence[k] < sequence[i] && sequence[i] < sequence[j]
② cur <= len
③ cur < s.top() 或 cur ! = s.top()　　sequence[i]<s.top() X
④ s.push(sequence[i++]);

第四次书面作业

1、假设某文件经内部排序得到100个初始归并段，试问:

(1)若要使多路归并三趟完成排序，则应取归并的路数至少为多少？

- 假设有m个初始顺串,每次对k个顺串进行归并,归并趟数为$[\log_k m]$
- k^3>=100, k>=5 ;至少取5路

(2)假若操作系统要求一个程序同时可用的输入、输出文件的总数不超过13，则按多路归并至少需几趟可完成排序？

需要1个输出缓冲区，故每次最多12路进行归并，$\log_{12}100$ 向上取整，至少需要2趟

2、有一个长度为n的数组A 由k个不重复的元素组成，其中k<$\sqrt{n}$，现要对此数组进行排序，要求其算法复杂度低于$\Omega(n \log n)$。为了完成这个任务，我们分两步完成。第一步先用辅助数组B来对A中不重复的k个元素进行排序，得到长度为k已排好序的数组B，第二步利用数组B来对A进行排序。

例子：假设我们有一个数组A

A = [5,$10^{10}, \Pi, \frac{128}{279}, 10^{10}, \Pi, 5, 10^{10}, \Pi, \frac{128}{279}, \Pi, \Pi, 5, 5, \Pi, 5, 10^{10}$]

辅助数组B = [$\frac{128}{279}, \Pi, 5, 10^{10}$]

最后得到的结果为：

A = [$\frac{128}{279}, \frac{128}{279}, \Pi, \Pi, \Pi, \Pi, \Pi, \Pi, 5, 5, 5, 5, 5, 10^{10}, 10^{10}, 10^{10}, 10^{10}$]

问题1：请设计一个算法完成第一步操作，得到数组B，并分析算法的时间复杂度

问题2：请设计一个算法完成第二步操作，写明算法思想即可

问题1：请设计一个算法完成第一步操作，得到数组B，并分析算法的时间复杂度

①从A中取一个数，查看是否存在于数组B

②若不存在，则加入数组B中，利用插入排序完成该步骤。

(保证数组B一直有序)

复杂度分析:

查询：利用二分，每次查询复杂度为$O(\log k)$，因为数组B中最多有k个元素，因此查询的复杂度为$O(\mathrm{nlog}k)$

插入：一共会有k次插入，因此复杂度为$O(1+2+\cdots+k)=$ $O(k^2)$

因此总复杂度为： $O(n\log k+k^2)$

又因 k< $\sqrt{n}$ 所以总复杂度为 $O(n\log k)$

问题2：请设计一个算法完成第二步操作，并分析算法的复杂度

利用计数排序

建立与B大小相同的数组C，清0

然后对A中的元素进行遍历，利用对应B中元素的位置，在C中记录此数出现的次数

再利用计数输出最后的结果

问题2：请设计一个算法完成第二步操作，并分析算法的复杂度

```
Counting-Sort(A)
For i = 1 to k do C[i] = 0
For i =1 to n do
        Location = Binary-Search(B,A[i])
        C[Location] = C[location]+1
For i = 1 to k do
        For j=0 to C[i]
                output(B[i])
```

3、给定一个关键字序列｛24，19，32，43，38，6，13，22｝

请写出快速排序第一趟的结果（以序列第一个数为轴值）

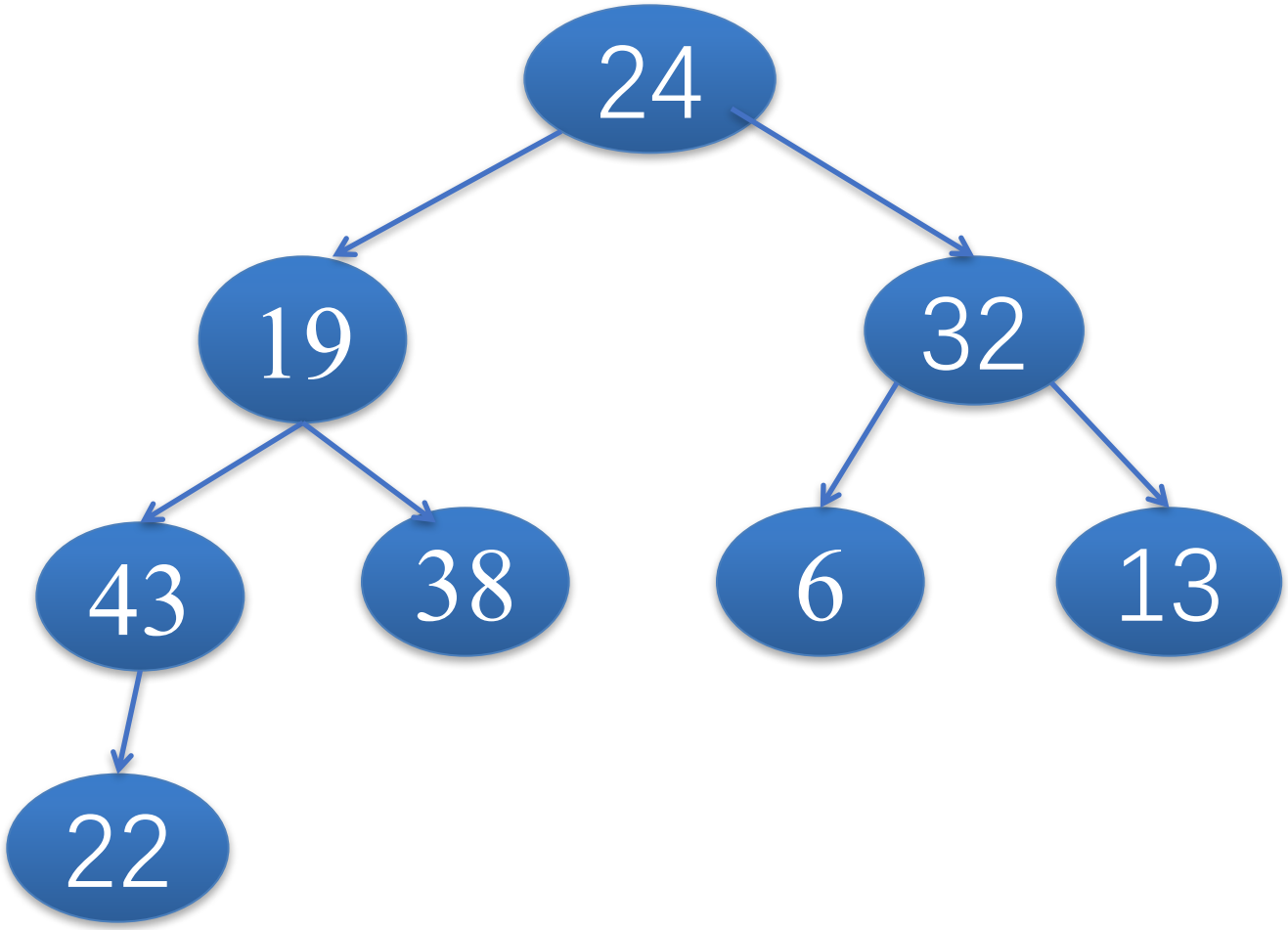
若用堆排序，所建的初始堆（最大堆);

在最坏情况下上面两种方法，哪种方法的时间复杂度最差？

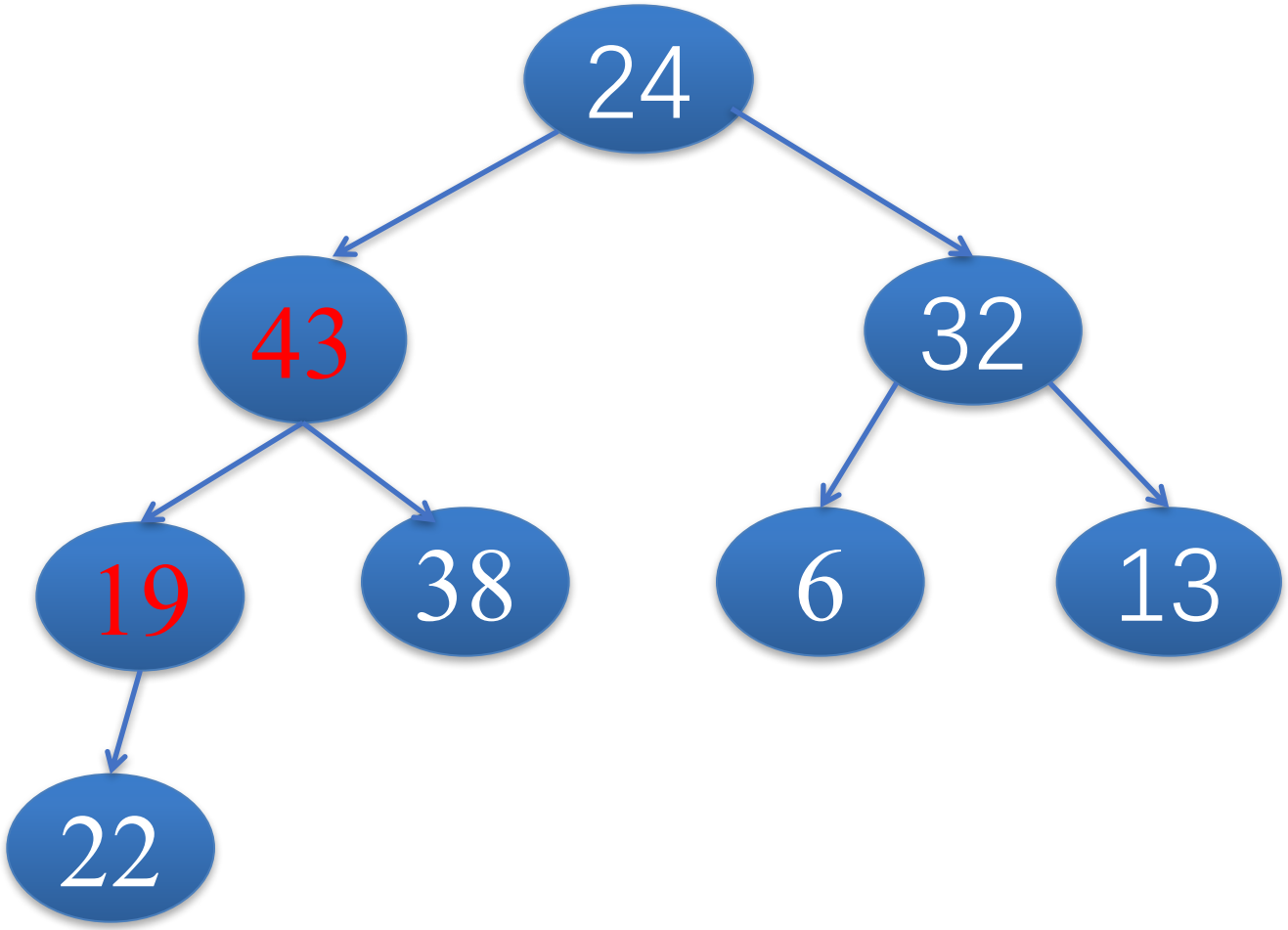
- 选择轴值并存储轴值
- 最后一个元素放到轴值位置
- 初始化下标i, j，分别指向头尾
- i递增直到遇到比轴值大的元素，将此元素覆盖到j的位置；
  j递减直到遇到比轴值小的元素，将此元素覆盖到i的位置
- 重复上一步直到i==j，将轴值放到i的位置，完毕

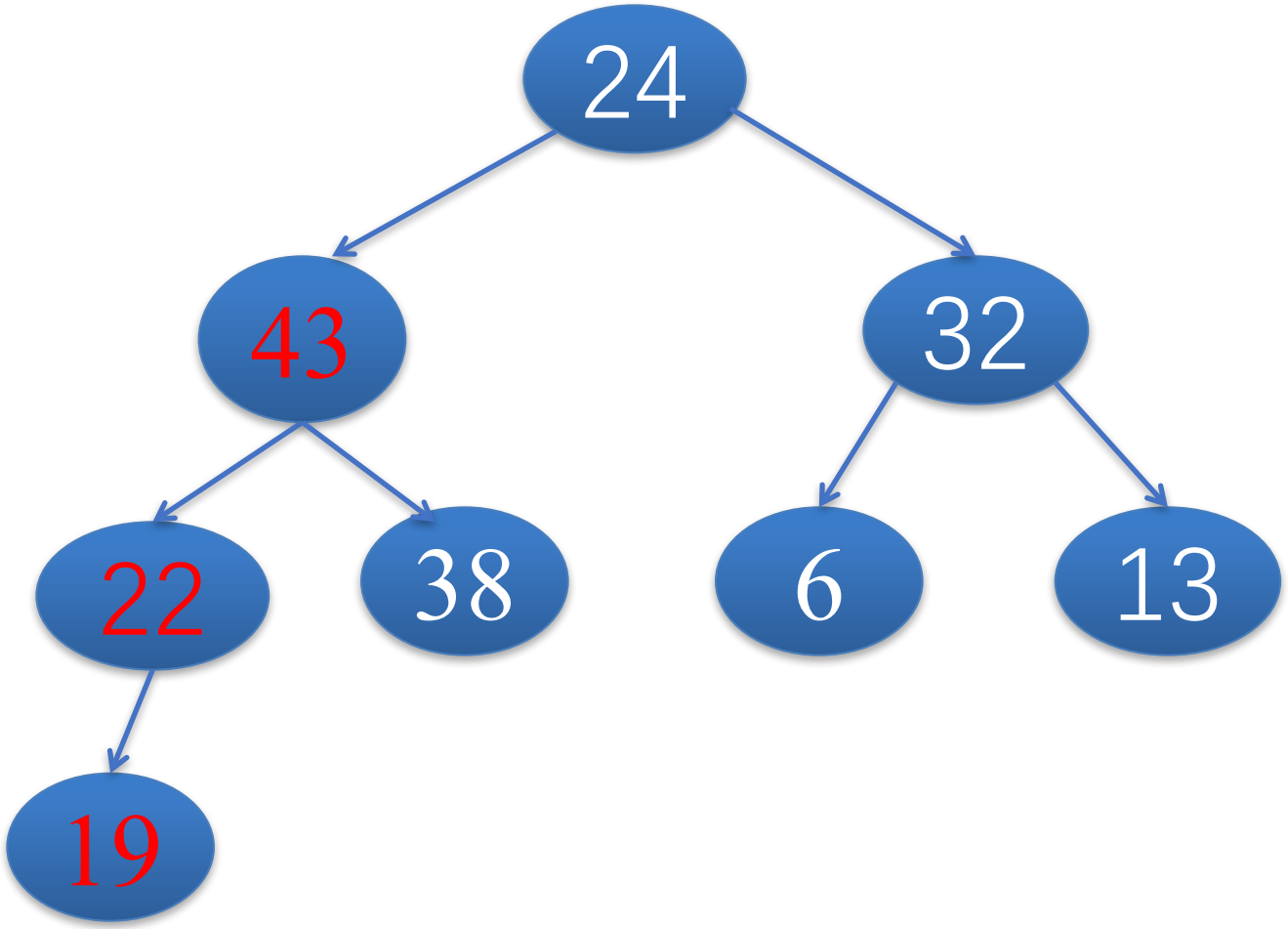
- 初始序列: 24 19 32 43 38 6 13 22 轴值=24
  L R
- 最后一个元素放到轴值位置: 22 19 32 43 38 6 13 22
  L R
- 移动L直到遇到比轴值大的元素:22 19 32 43 38 6 13 32
  L R
- 移动R直到遇到比轴值小的元素:22 19 13 43 38 6 13 32
  L R
- 移动L直到遇到比轴值大的元素:22 19 13 43 38 6 43 32
  L R
- 移动R直到遇到比轴值小的元素:22 19 13 6 38 6 43 32
  L R
- 移动L直到遇到比轴值大的元素:22 19 13 6 38 38 43 32
  L R
- 移动R,LR两者相等 该替换为轴值:22 19 13 6 24 38 43 32

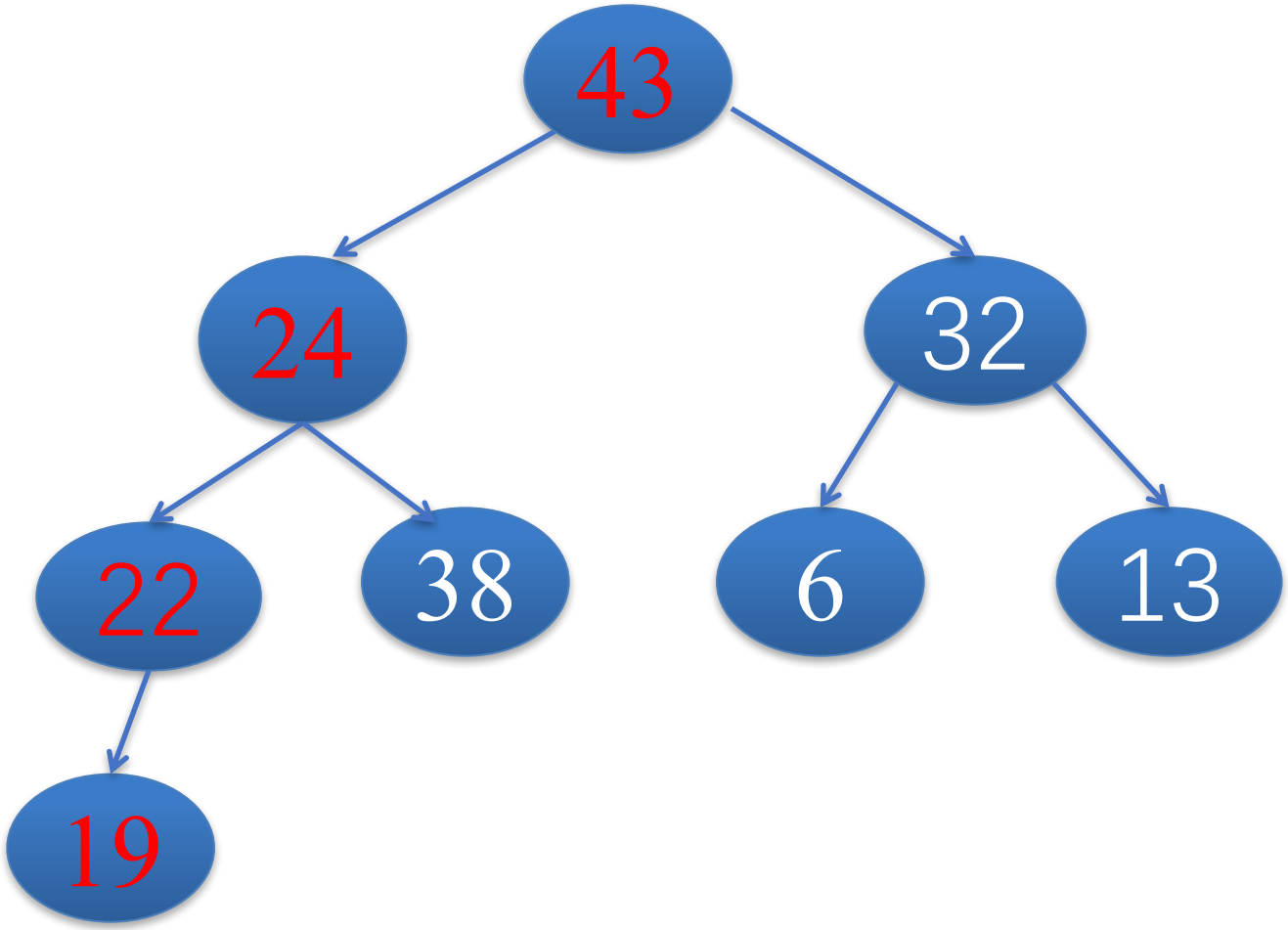
建堆-填充

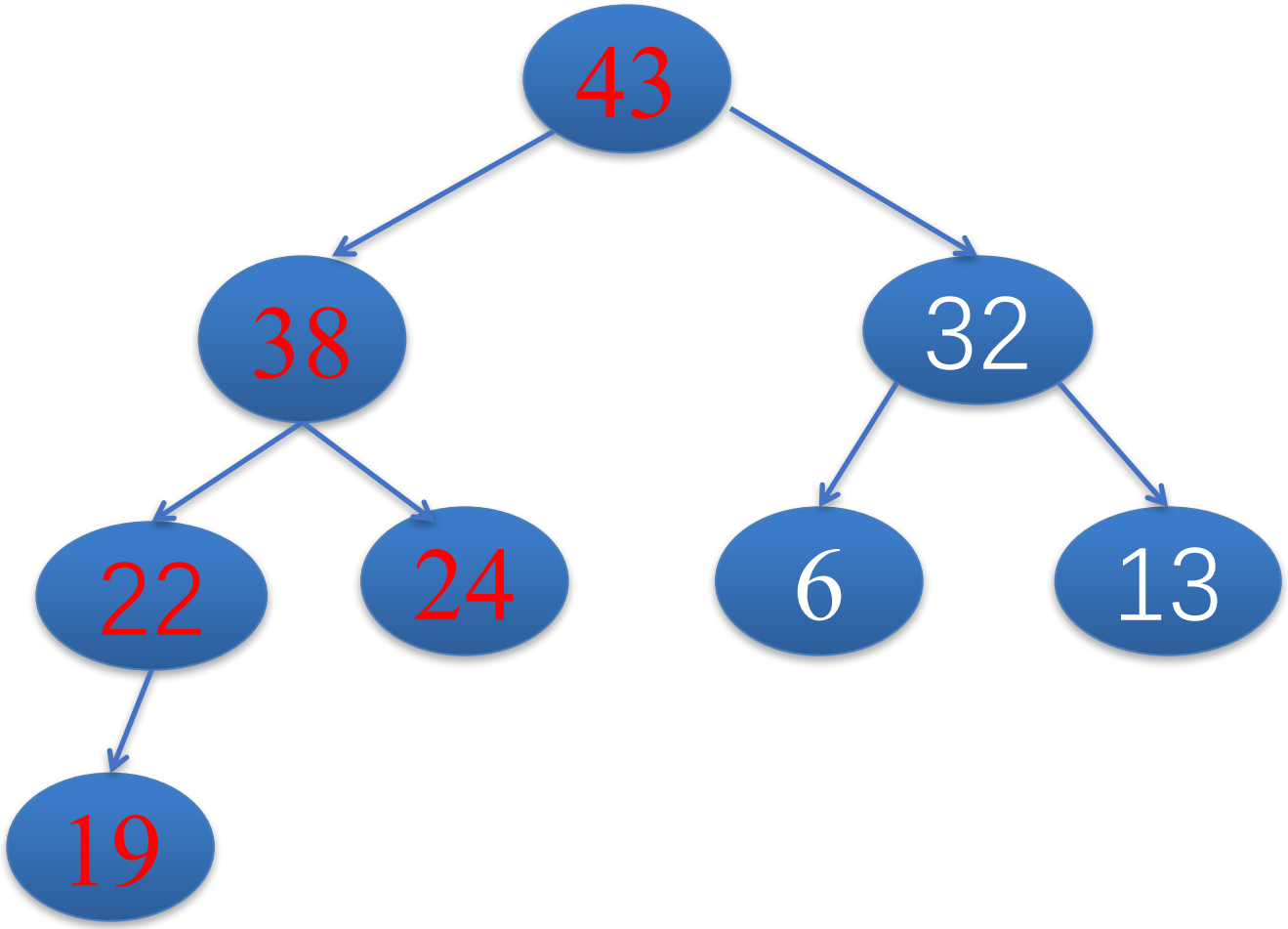


建堆-调整



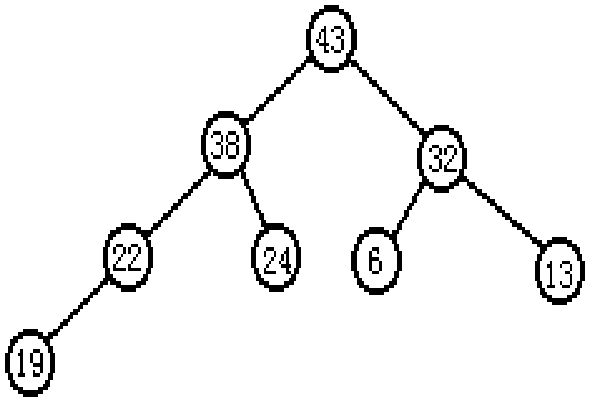

24
43
32
22
38
6
13
19


43
24
32
22
38
6
13
19


43
38
32
22
24
6
13
19

（1）快速排序的第一趟结果为｛22,19,13,6,24,38,43,32｝;

（2）堆排序时所建立的初始大顶堆如所图所示：



（3）在最坏情况下两种排序方法所需时间：堆是 O(nlogn)，快速排序是 $O(n^2)$，所以，可见在最坏情况下快速排序时间复杂度最差。

注意：快速排序的平均时排序速度最快，但在最坏情况下不一定比其他排序方法快。

3．以 10 为基数，采用低位优先的基数排序处理下列数字：{346, 22, 31, 212, 157, 102, 568, 435, 8, 14, 5}，则有

第 1 遍：_____

第 2 遍：_____

第 3 遍：_____

第一遍：31, 22, 212, 102, 14, 435, 5, 346, 157, 568, 8

第二遍：102, 5, 8, 212, 14, 22, 31, 435, 346, 157, 568

第三遍：5, 8, 14, 22, 31, 102, 157, 212, 346, 435, 568

5、给定元素值互异的数组A[1,…,n]，尝试在O(n)时间内找到第k小的数，设计出相应算法并给出给出算法复杂度（提示：以快速排序算法为模型，递归查找第k小的数）。

主要思想：利用快速排序中的轴值，可将数组分成两个部分。若轴值恰好是第k大的数，则返回轴值，否则若k在左半部分，在左半部分寻找，若k在右半部分，在右半部分寻找。

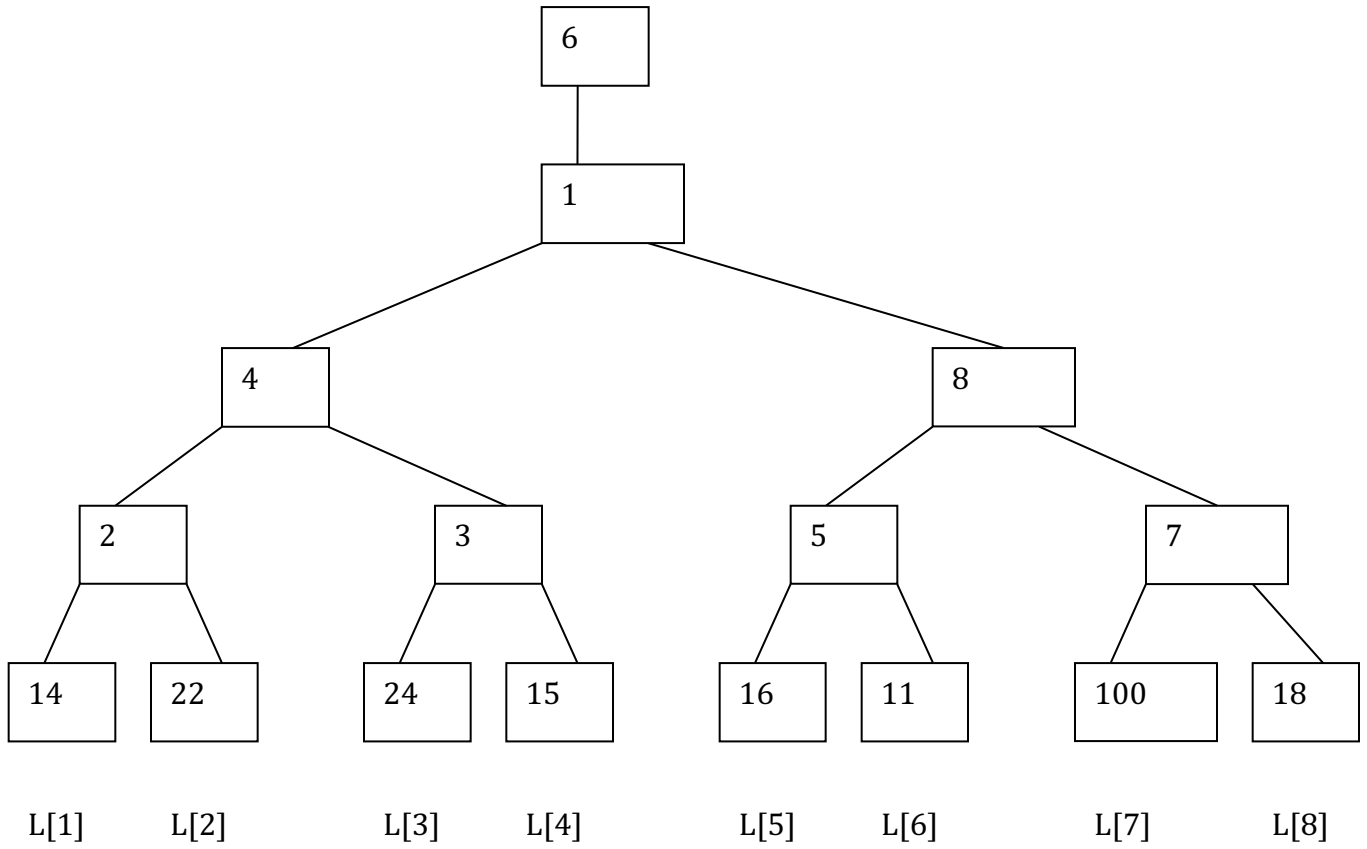
时间复杂度：O(n) [n+n/2+n/4+……]

附主要函数代码：

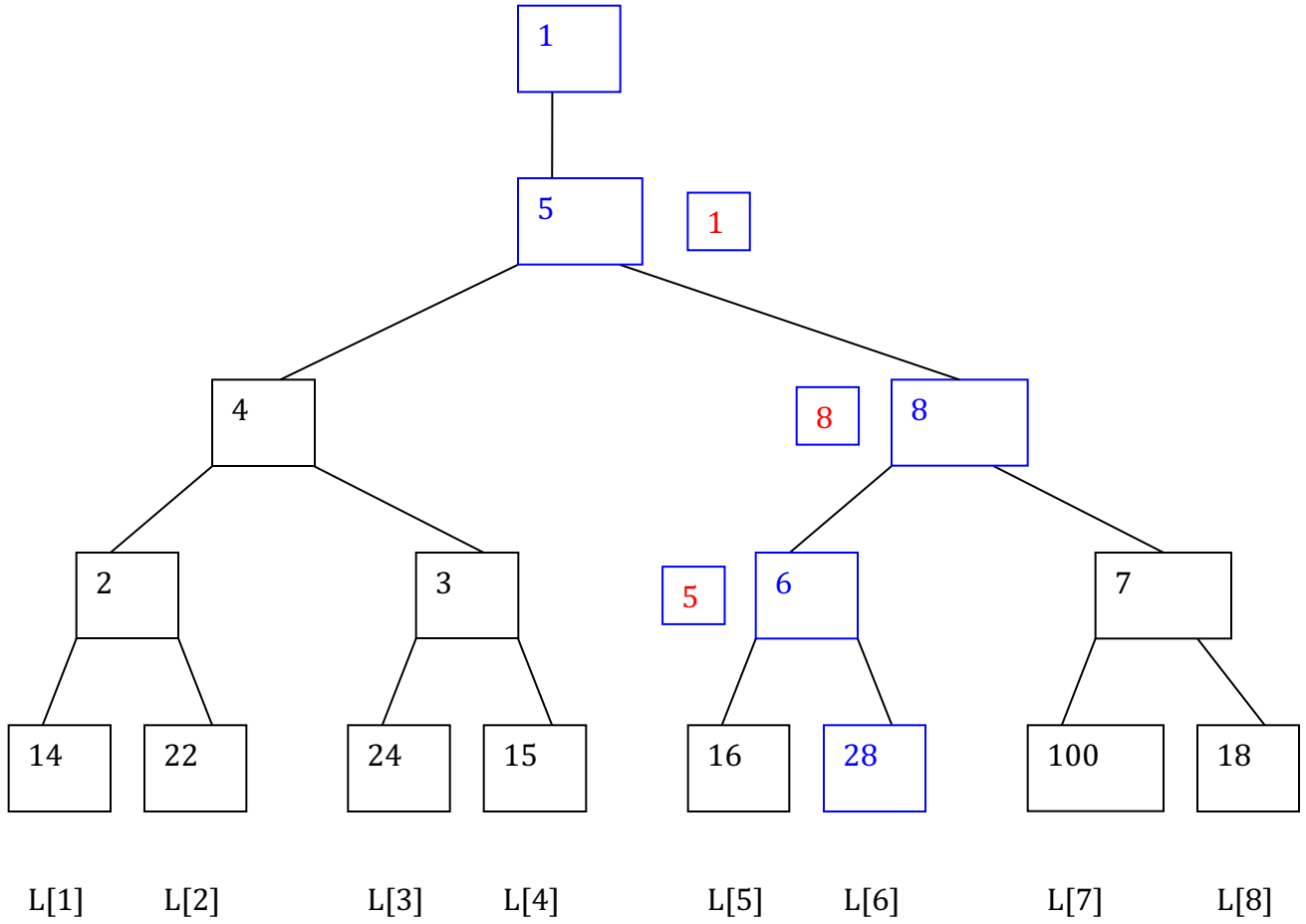
```
int search(int l, int r, int key)  //传入参数表示数组下标标号,假设数组从1开始标号
{
        if(l == r)  return a[r];
        int p;
        p = Partition(l,r);
        if(key == p) return a[p];
        if(key < p) return search(l, p-1, key);
        else return search(p+1, r , key);
}
```

6、有8个顺串，每个顺串的第一个记录的关键码分别为14，22，24，15，16，11，100，18，而第二个记录的关键码分别为26，38，30，26，50，28，110，40。请画出对顺串开始8路合并时的败者树。从败者树输出一个全局优胜者(并有相应的一个记录进入败者树)后需对败者树进行重构，请画出输出第1个全局优胜者并进行重构后的败者树。

初始时的败者树：



重建的败者树



索引部分课堂作业

1. 设磁盘页块大小为4096（=4K）字节，存储每个记录对象需要占用128 字节（其中关键码占用4字节），且所有记录均已按关键码有序地存储在磁盘文件中。若主存中有512K空间可以用来存储索引结构，请简要回答以下问题:

（1）使用线性索引，数据文件中最多可以存放多少字节数据记录？(每个索引项8字节，其中关键码4 字节，地址4 字节)

（2）使用二级索引，数据文件中最多可以存放多少字节的数据记录？

答案:

(1) 内存大小 512K，内存中可以存储的索引项为: 512K/8 = 64K（个）
一个索引项对应着一个磁盘块：
4096 * 64K = 256M 字节数据

(2) 二级索引在内存中，一级索引放在磁盘中。
内存中可以存放的二级索引数量: 512K/8 = 64K(个)
在每一个索引项对应的磁盘块中存放一级索引，共有索引项:
64K * (4096/8) = 32M （个）
共存放的字节数据: 32M*4096 = 128G

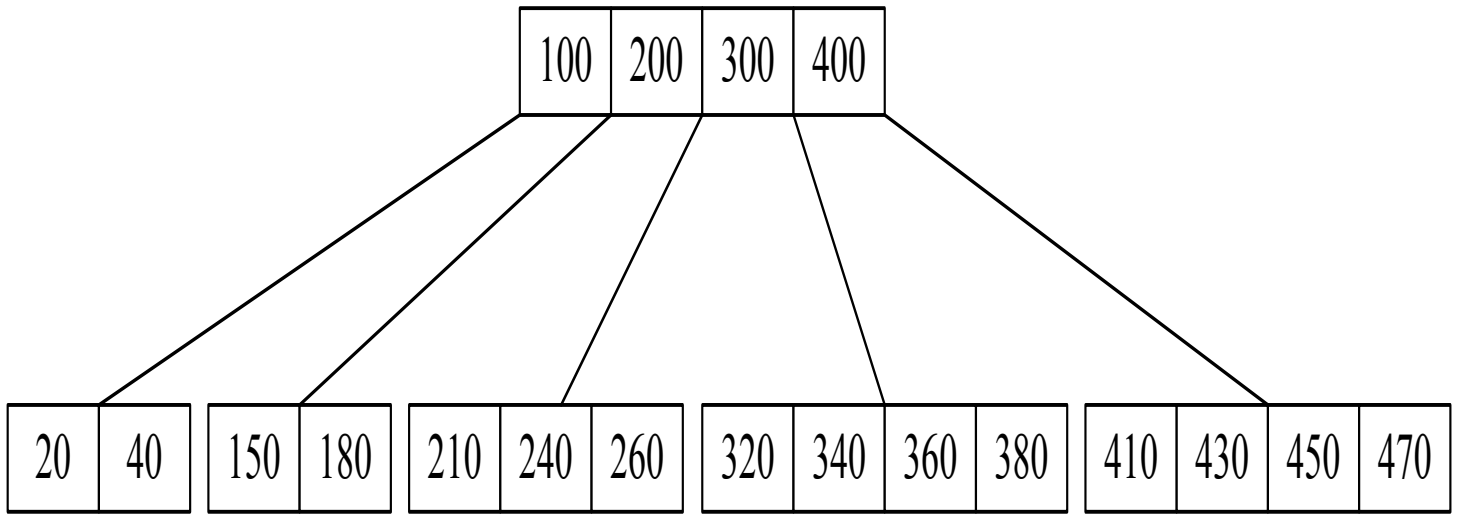
2. 64阶的B树中，除根结点和叶结点外，每个结点至少包含__31___个关键码，最多包含__63__个关键码，设B树中包含的关键码个数为64K（64*1024），B树结点均存放在磁盘中，每次只能从磁盘往内存中读入一个结点，查找一个给定关键码的记录时，最多需要进行_5___次访问操作。

$$N+1 \geq 2 \cdot \lceil m/2 \rceil^{k-1}, \quad k \leq 1+\log_{\lceil m/2 \rceil}(\frac{N+1}{2})$$

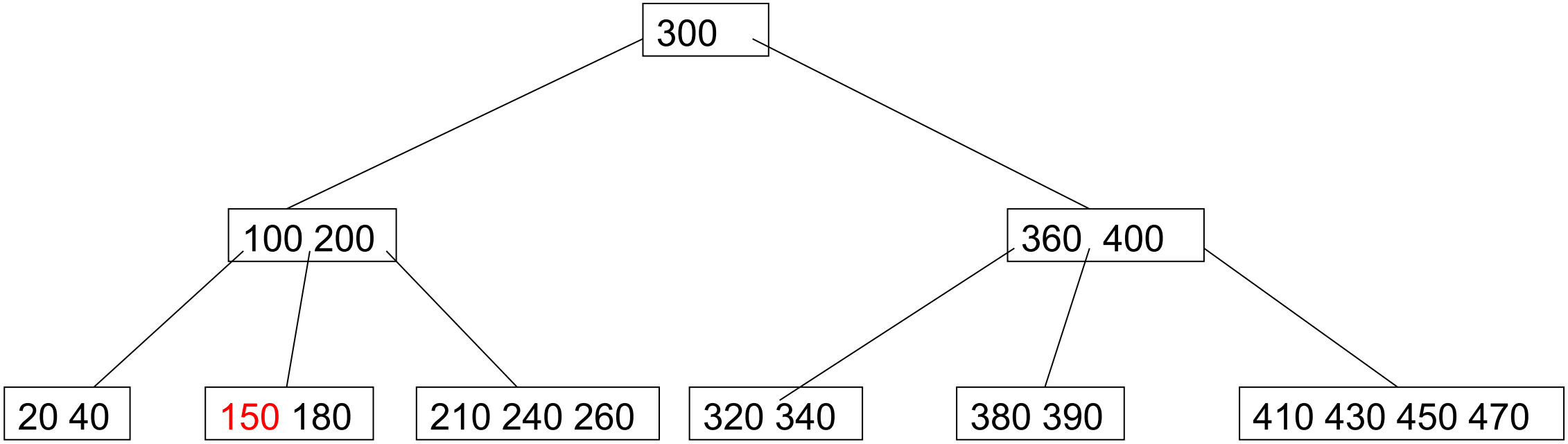
关键码个数N = 64*1024　　阶数m=64

层数k为4　　高度为 K+1

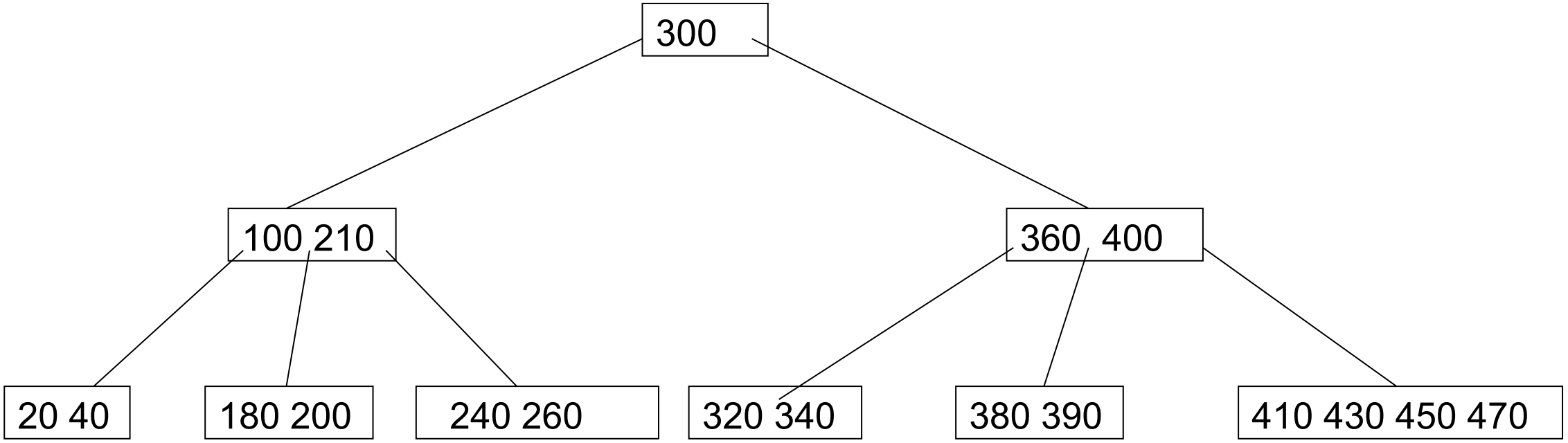
3. 请画出往下图的5阶B树中插入一个关键码390后得到的B树，以及再删除关键码150后得到的B树。并请分别分析插入和删除时的读写页块次数。（假设在操作之前B树所有的结点页块均在外存，删除关键码时，如果需要从兄弟结点借关键码的话，先查看左边的、再查看右边的兄弟。）



- 找插入节点时向下读盘次数：2次
- 一次非根节点分裂+一次根节点分裂 写次数：5次



- 读盘次数：5次(向下找到该节点加查看左右兄弟节点)
- 写次数：3次



期中考试最后两道证明题：

二叉树的内部路径长度是指所有节点的深度的和。假设将N个互不相同的随机元素插入一棵空的二叉搜索树。请证明得到的二叉搜索树的内部路径长度期望为O(NlogN)。

设 D(N)为 N 个结点的二叉搜索树的内部路径长度，如果根的左子树有 i 个结点，则

D(N)=D(i)+D(N-i-1)+N-1。

如果根的左子树有 i 个节点，则插入序列的第一个元素是 N 各元素中的 i+1 小的元素，

由于序列的随机性，根的左子树有 i 个节点(0<=i<=N-1)概率分别为 1/N。

则期望内部路径长度

$$D(N) = \left(\sum_{i=0}^{N-1}(D(i) + D(N-i-1))\right)/N + N - 1 = (2/N)\left(\sum_{i=0}^{N-1} D(i)\right) + N - 1 = O(NlogN)$$

为什么是O(NlogN)？？《数据结构与算法分析C语言描述》

$$D(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + N - 1$$

在第7章将遇到并求解这个递归关系，得到的平均值为 $D(N) = O(N \log N)$。因此任意节点的期望深度为 $O(\log N)$。作为一个例子，图4-26所示随机生成的500个节点的树的节点

$$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN \tag{7.14}$$

如果用 N 乘以方程(7.14)，则有

$$NT(N) = 2\left[\sum_{j=0}^{N-1} T(j)\right] + cN^2 \tag{7.15}$$

我们需要除去求和符号以简化计算。注意，我们可以再套用一次方程(7.15)，得到

$$(N-1)T(N-1) = 2\left[\sum_{j=0}^{N-2} T(j)\right] + c(N-1)^2 \tag{7.16}$$

若从(7.15)减去(7.16)，则得到

$$NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c \tag{7.17}$$

移项、合并并除去右边无关紧要的项 $-c$，我们得到

$$NT(N)=(N+1)T(N-1)+2cN \tag{7.18}$$

现在有了一个只用 $T(N-1)$表示 $T(N)$的公式。再用叠缩公式的思路，不过方程(7.18)的形式不适合。为此，用 $N(N+1)$除方程(7.18)：

$$\frac{T(N)}{N+1}=\frac{T(N-1)}{N}+\frac{2c}{N+1} \tag{7.19}$$

现在可以进行叠缩

$$\frac{T(N-1)}{N}=\frac{T(N-2)}{N-1}+\frac{2c}{N} \tag{7.20}$$

$$\frac{T(N-2)}{N-1}=\frac{T(N-3)}{N-2}+\frac{2c}{N-1} \tag{7.21}$$

$$\vdots$$

$$\frac{T(2)}{3}=\frac{T(1)}{2}+\frac{2c}{3} \tag{7.22}$$

将方程(7.19)到(7.22)相加，得到

$$\frac{T(N)}{N+1}=\frac{T(1)}{2}+2c\sum_{i=3}^{N+1}\frac{1}{i} \tag{7.23}$$

该和大约为 $\log_e(N+1)+\gamma-\frac{3}{2}$，其中 $\gamma\approx0.577$ 叫做欧拉常数(Euler's constant)，于是

$$\frac{T(N)}{N+1}=O(\log N) \tag{7.24}$$

从而

$$T(N)=O(N\log N) \tag{7.25}$$

2、证明按先根次序周游树获得的序列与其对应二叉树的前序周游获得的序列相同

不妨设先根次序周游得到的序列为 S_1，前序周游得到的序列为 S_2，则要证明 $S_1 = S_2$，

即证明对$\forall(x1,x2) \in S_1$，则一定有$(x1',x2') \in S_2$，其中$x1'$和$x2'$分别是树中结点x1和x2

对应二叉树的结点。

若$(x1,x2) \in S_1$，那么分以下两种情况：

若x2在x1的子树，那么对应二叉树中，$x2'$在$x1'$的左子树中，一定有$(x1',x2') \in S_2$。

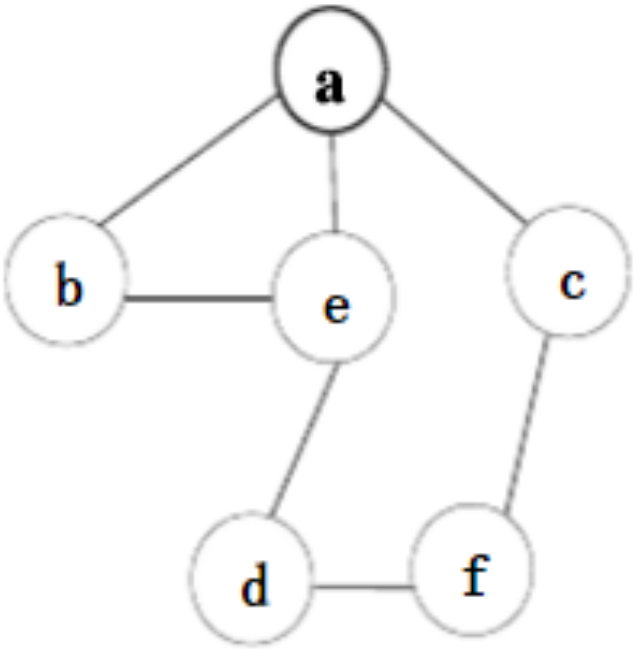
若x2在x1的兄弟树中，那么对应二叉树中，$x2'$在$x1'$的右子树中，一定有$(x1',x2') \in S_2$。

所以得证$\forall(x1,x2) \in S_1$，则一定有$(x1',x2') \in S_2$，所以两个序列相同。

作业三

1、选择填空题

（1）如图所示，若从a点出发进行遍历，若按深度优先搜索法，则可能得到的一种顶点序列为___D___若按宽度优先搜索法，则可能得到的一种顶点序列为___B__



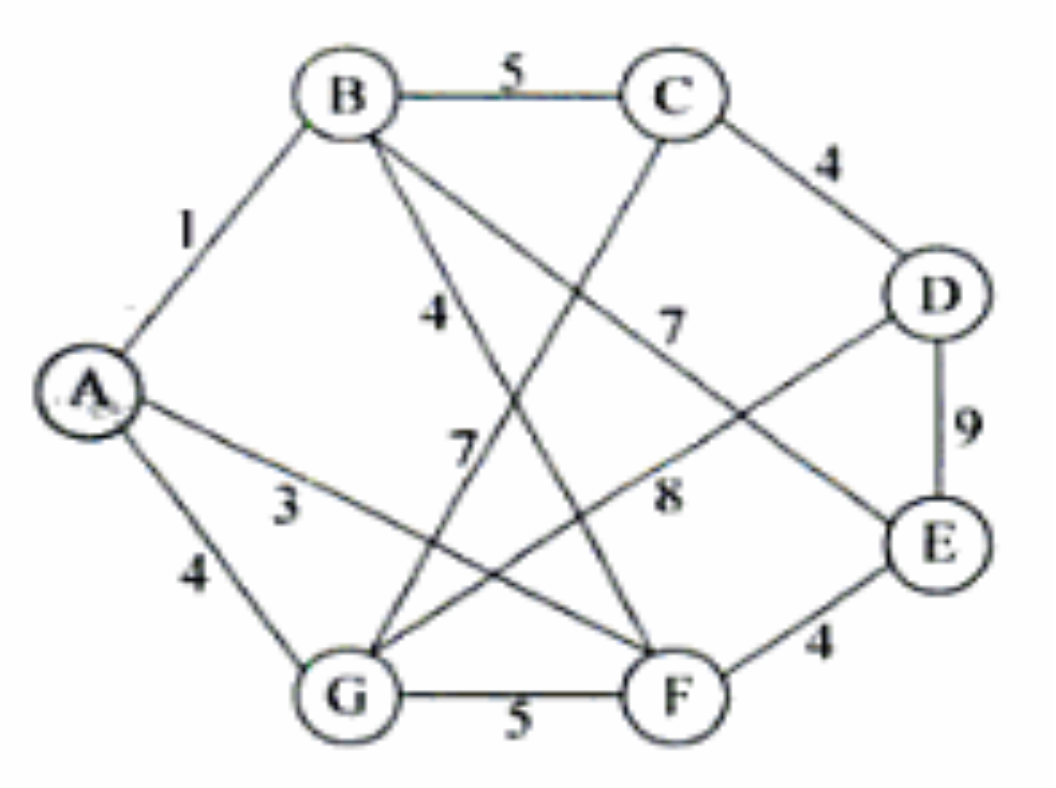
① A. a, b, e, c, d, f　　B. e, c, f, e, b, d
C. a, e, b, c, f, d　　D. a, e, d, f, c, b

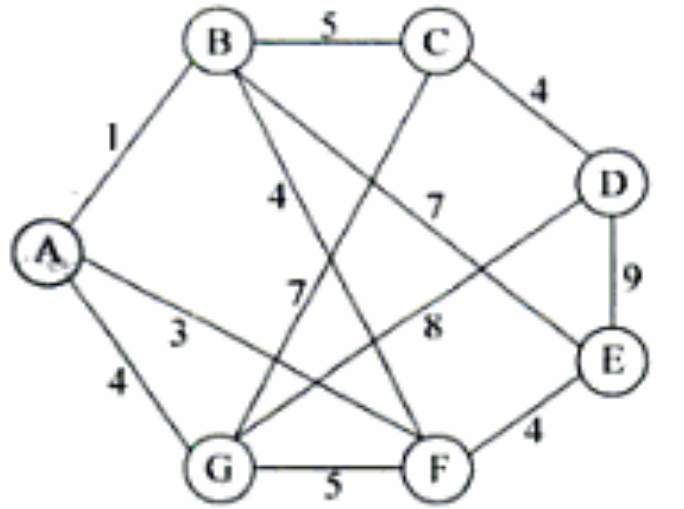
② A. a, b, c, e, d, f　　B. a, b, c, e, f, d
C. a, e, b, c, f, d　　D. a, c, f, d, e, b

（2）有29 条边的无向连通图，至少有 9 个顶点，至多有 30 个顶点；

完全图：任何两顶点间都有边相关联的图称为完全图 (complete graph)，完全图显然具有最大的边数
n 个顶点的无向完全图有 n(n-1)/2 条边

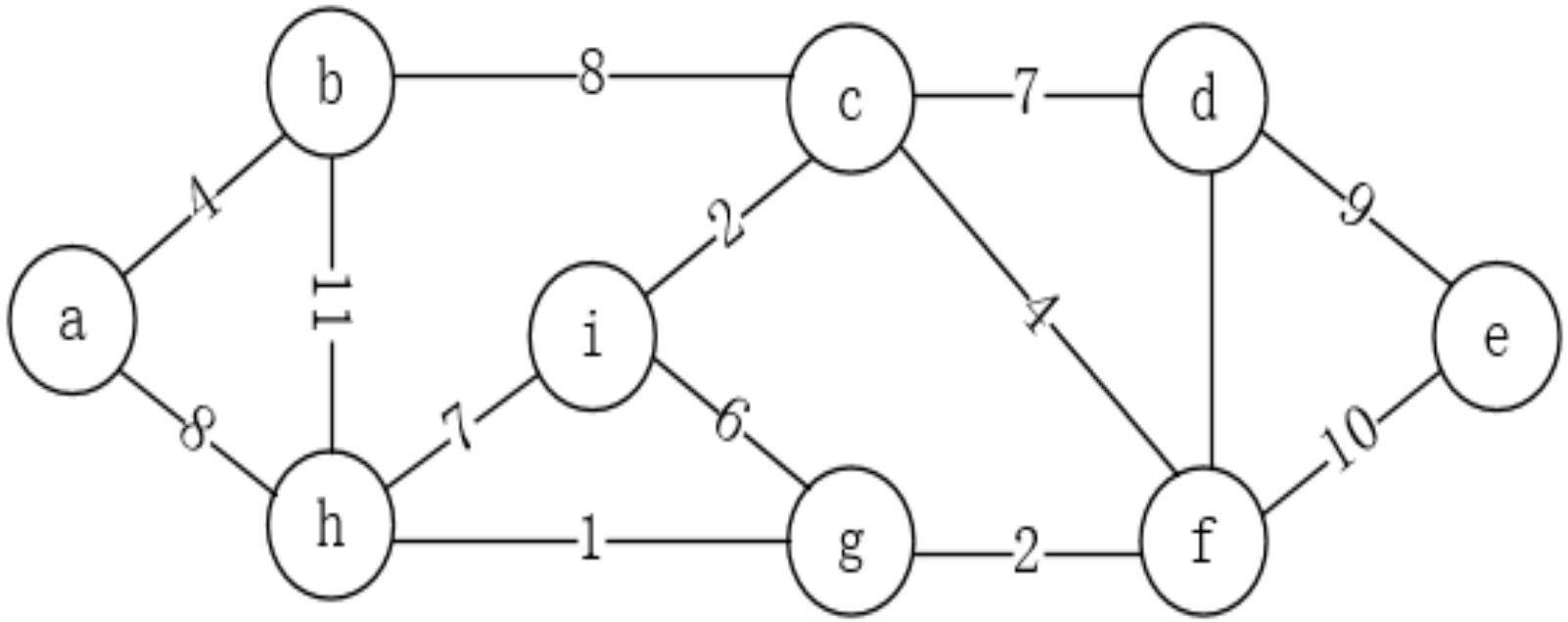
2、如下图，请用Dijkstra算法计算由源节点A到图中其他各结点的最短路径，写出计算过程（建议使用PPT中的表格形式）。

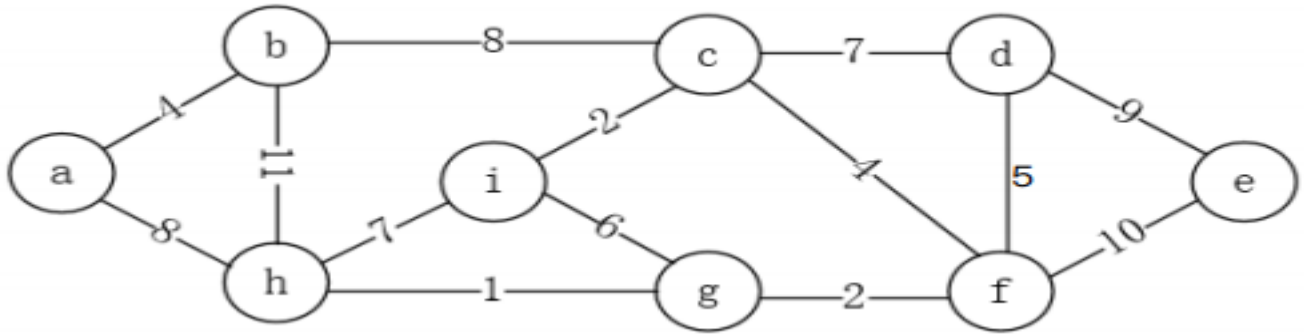




迭代	集合 N	结点 B	结点 C	结点 D	结点 E	结点 F	结点 G
初始化	{A}	(B,1)	(no,∞)	(no,∞)	(no,∞)	(F,3)	(G,4)
1	{A,B}	(B,1)*	(B,6)	(no,∞)	(B,8)	(F,3)	(G,4)
2	{A,B,F}		(B,6)	(no,∞)	(F,7)	(F,3)*	(G,4)
3	{A,B,F,G}		(B,6)	(G,12)	(F,7)		(G,4)*
4	{A,B,F,G ,C}		(B,6)*	(C,10)	(F,7)		
5	{A,B,F,G ,C,E}			(C,10)	(F,7)*		
6	{A,B,F,G ,C,E,D}			(C,10)*			

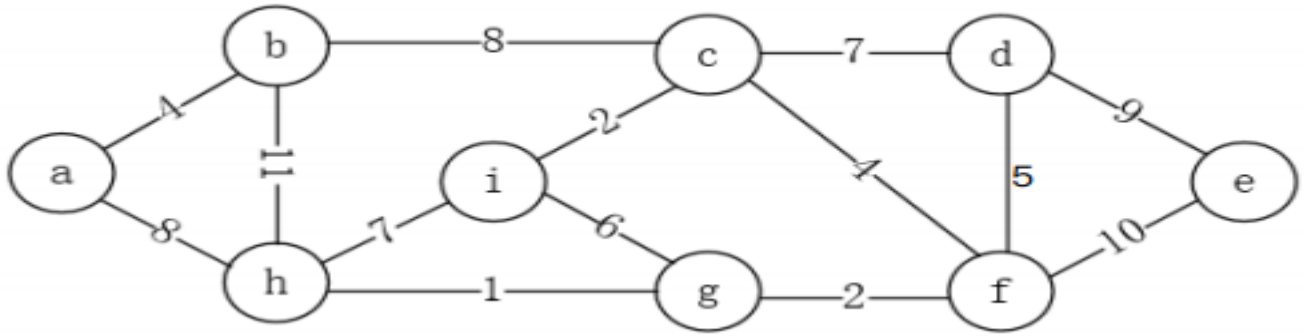
3、如下图，请分别使用Kruskal和Prim算法构造最小生成树，画出构造过程。使用Prim算法时以a结点为初始节点。





Kruskal：

第一步：判断权值为1的连接边hg是否属于集合，由于开始集合为空，则将两个节点加入到集合，且连接该边，并判断是否属于回路；
第二步：判断权值为2的连接边ic是否属于集合，由于集合只有节点h和g，则将节点i和c加入到集合中，且连接该边，并判断是否属于回路；
第三步：判断权值为2的连接边gf是否属于集合，根据判断规则，把f加入到集合，且连接该边；
第四步：根据判断规则，将节点a和b加入到集合，且连接该边；
第五步：根据判断规则，连接该边cf；
第六步：根据判断规则，连接fd；将d加入集合；
第七步：根据判断规则，连接节点i和g时形成回路，则放弃，继续下次判断；
第八步：判断权值为7的连接边；cd和hi都会形成回路，放弃；
第九步：判断权值为8的连接边；根据判断规则，连接节点a和h；放弃bc
第十步：判断权值为9的连接边；将节点e加入到集合，并连接该边；此时集合已经包含所有节点，则终止循环；



Prim：

第一步：以a为初始节点，并找到权值为最小的边，即为ab；
第二步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；即权值最小边为bc或者ah；这里选取bc；
第三步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；即选择ci；
第四步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；即选择cf；
第五步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；即选择fg；
第六步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；即选择gh；
第七步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；必须保证不能形成回路，即选择cd；
第八步：在所有已选取结点与未选取结点构成的边中，找一条权值最小的边；必须保证不能形成回路，即选择de；由于树包含了所有节点，且满足，则在此终止，为的最小生成树。

4、求证：只要适当地排列顶点的次序，就能使有向无环图邻接矩阵中主对角 线以下的元素全部为 0。

解析：

任意n个结点的有向无环图都可以得到一个拓扑序列。设拓扑序列为v0v1v2…vn-1，我们来证明此时的邻接矩阵A为上三角矩阵。证明采用反证法。

假设此时的邻接矩阵不是上三角矩阵，那么，存在下标i和j（i>j），使得A[i][j]不等于零，即图中存在从vi到vj的一条有向边。由拓扑序列的定义可知，在任意拓扑序列中，vi的位置一定在vj之前，而在上述拓扑序列v0v1v2…vn-1中，由于i>j，即vi的位置在vj之后，导致矛盾。因此命题正确。

5、请设计一个基于深度优先搜索的拓扑排序算法 ，使之能够具备发现图中环的功能 。（给出伪代码 ，并给予适当的注释 ）

【思路】针对有回路的拓扑排序，应该将标志位设为三种情况：UNVISIT、VISIT和PUSHED(表明已放入后序深度周游的结果序列中)。
当遇到一个结点，从它所能邻接到的结点中若有已被标志为VISIT但不是PUSHED的时，表明出现了环。
为了方便处理，增加一个tag标记，一旦发现环则层层退出，并报告“存在环”。

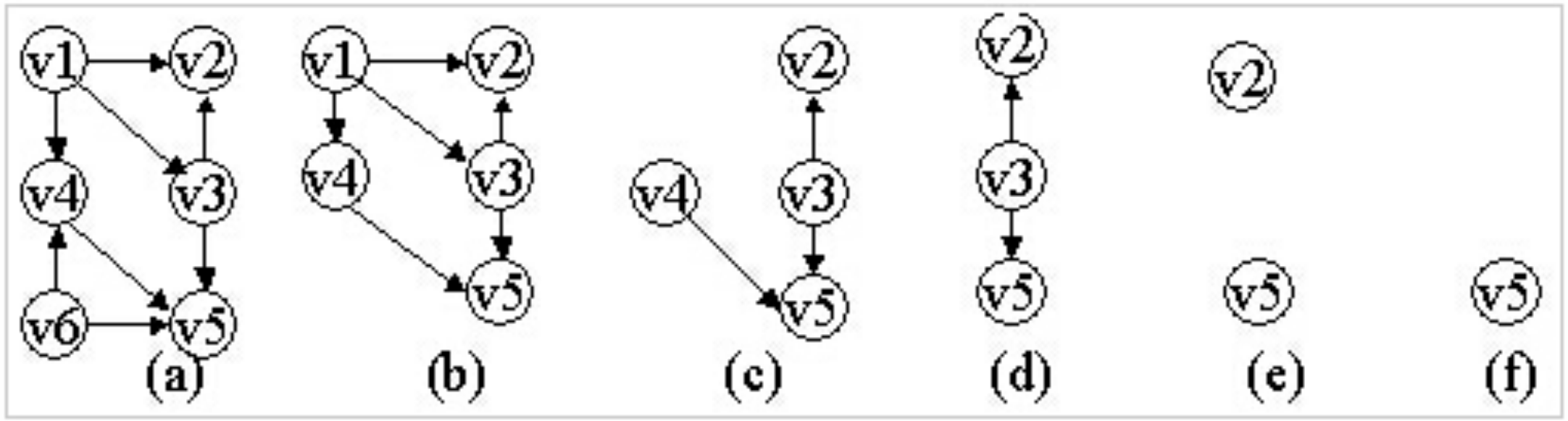
Kahn算法：

需要维护一个入度为0的顶点的集合：

每次从该集合中取出(没有特殊的取出规则，随机取出也行，使用队列/栈也行，下同)一个顶点，将该顶点放入保存结果的List中。

紧接着循环遍历由该顶点引出的所有边，从图中移除这条边，同时获取该边的另外一个顶点，如果该顶点的入度在减去本条边之后为0，那么也将这个顶点放到入度为0的集合中。然后继续从集合中取出一个顶点……

v6 - v1 - v4 - v3 - v2 - v5



拓扑排序和深度优先搜索存在这样的关系：

经过拓扑排序后的顶点是按深度优先搜索后各顶点结束递归时间的降序排列的。

```
void DFS(int u)
{
        vis[u] = 1;    //标记当前节点被访问
        for(int v=1; v<=n; v++)
                if(a[u][v] && vis[v]! =2)
                 DFS(v);

        Vis[u] = 2;  //标记当前节点已结束递归
        timef++;
        f[u] = timef;
}

void DFS_main()     {
        timef = 0;

        for(int i=1; i<=n; i++)
        {
                if(vis[i] == 0)
                        DFS(i);
        }
}
```

Vis[i]=0:节点没被访问过
Vis[i]=1:节点被访问过但还没结束递归
Vis[i]=2:节点结束递归

【算法】基于DFS的拓扑排序

```
void TopsortbyDFS_Circle(Graph& G) {
      for (int i=0; i<G.VerticesNum(); i++)  //对图所有顶点的标志位初始化为UNVISIT
            G.Mark[i] = UNVISITED;
      int *result=new int[G.VerticesNum()];
      int tag = NOTCIRCLED, index = 0;

      for (i=0; i<G.VerticesNum(); i++)
            if ((G.Mark[i] == UNVISITED) && (tag == NOTCIRCLED))
                  Do_Topsort(G, i, result, index, tag);     // 递归
      if (tag == NOTCIRCLED)
          for (i=G.VerticesNum()-1; i>=0; i--)              // 逆序输出
                  cout << result[i] << "\t";
      cout<<endl;
}
```

```
void Do_Topsort(Graph& G, int v, int *result, int &index, int& tag)  {
        // 若本顶点已经VISITED过，即被DFS深入了，但还未纳入PUSHED，则形成了环
        if (G.Mark[v] == VISITED) {
                cout << "此图有环！"<< endl;        // 图有环
                tag = CIRCLED;   return;
        }
        G.Mark[v]= VISITED;              // 标记当前节点被访问

        for (Edge e= G.FirstEdge(v); G.IsEdge(e); e=G.NextEdge(e))  {
                if ((G.Mark[G.ToVertex(e)] != PUSHED) && (tag == NOTCIRCLED))

                        Do_Topsort(G, G.ToVertex(e), result, index, tag);

        }
        result[index++]=v;    // 记录每个节点结束递归的时间
        G.Mark[v] = PUSHED;   // 把该结点设置为PUSHED标记，已结束递归
}
```

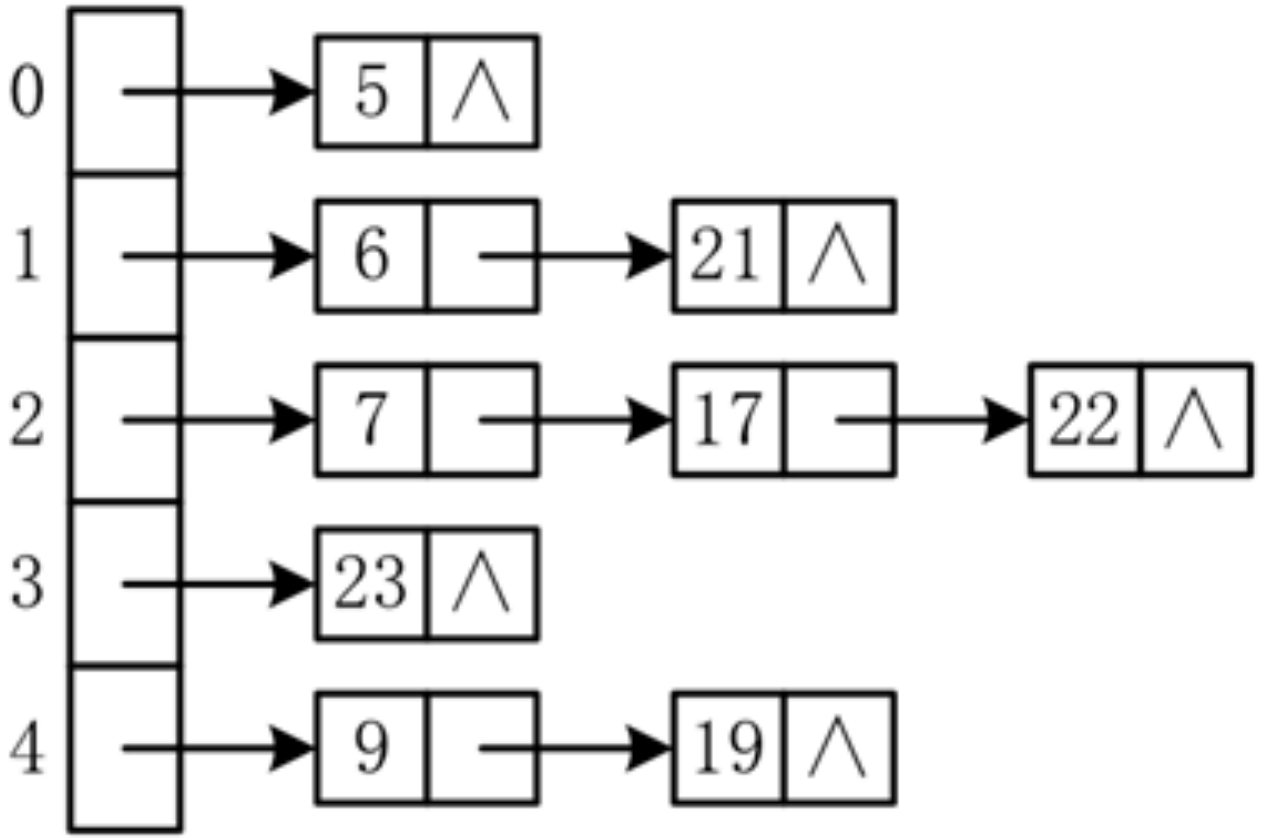
【算法结束】

两种实现算法的总结：
这两种算法分别使用链表和栈来表示结果集。
对于基于DFS的算法，加入结果集的条件是：顶点的出度为0。这个条件和Kahn算法中入度为0的顶点集合似乎有着异曲同工之妙，这两种算法的思想犹如一枚硬币的两面，看似矛盾，实则不然。一个是从入度的角度来构造结果集，另一个则是从出度的角度来构造。

实现上的一些不同之处：
Kahn算法不需要检测图为DAG，如果图为DAG，那么在出度为0的集合为空之后，图中还存在没有被移除的边，这就说明了图中存在环路。而基于DFS的算法需要首先确定图为DAG，当然也能够做出适当调整，让环路的检测和拓扑排序同时进行，毕竟环路检测也能够在DFS的基础上进行。
二者的复杂度均为O(V+E)。

检索

1. 有关键字序列{7,23,6,9,17,19,21,22,5}，Hash 函数为H(key)=key %5，采用拉链法处理冲突，试构造哈希表。



2、已知一个线性表(38，25，74，63，52，48)，采用的散列函数为 H(Key) =Key mod 7，将元素散列到表长为 7 的哈希表中存储。

（1）若采用线性探查的冲突解决策略，则在该散列表上进行等概率成功查找的平均查找长度为多少？请写出计算过程。

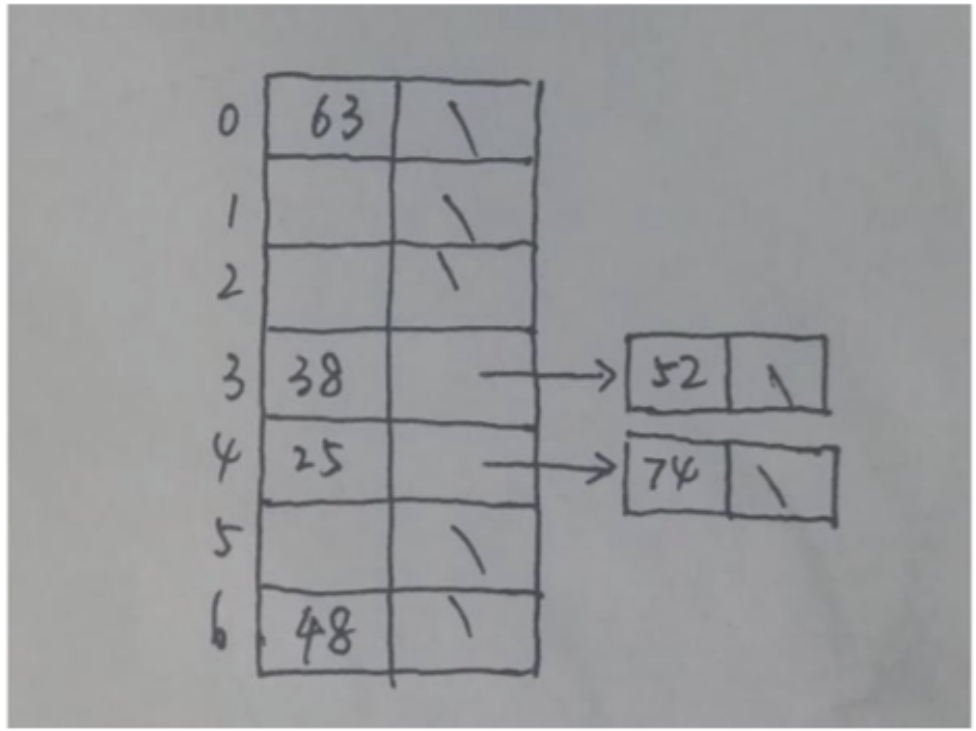
（2）若采用拉链法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为多少？请写出计算过程。

4. (1) 线性探查的散列表如下表所示。

0	1	2	3	4	5	6
63	48		38	25	74	52

ASL=1/6+1/6+2/6+1/6+4/6+3/6=2。

(2) 如下图所示，ASL=1/6+1/6+1/6+1/6+2/6+2/6=4/3



3. 写出带墓碑的插入操作的算法过程

被删除标记值称为墓碑(tombstone)
标志一个记录曾经占用这个槽,但是现在已经不再占用了

要保证被释放的墓碑位置可以再被利用
又不影响探查

```
template <class Key, class Elem, class KEComp, class EEComp>
bool hashdict<Key, Elem, KEComp, EEComp>::hashInsert(const
Elem &e)  {
    int insplace, i=0, pos=home= h(getkey(e));
    bool tomb_pos=false;
    while (!EEComp::eq(EMPTY, HT[pos])) {
        if (EEComp::eq(e, HT[pos])) return false;
            if (EEComp::eq(TOMB, HT[pos]) &&  !tomb_pos)
          {insplace=pos;  tomb_pos=true;} //第一
            pos = (home + p(getkey(e), ++ i)) % M;
     }
     if (!tomb_pos) insplace=pos; //如果没有墓碑，则insplace位于
空槽位置
          HT[insplace]=e;    return true;
}
```