2016年统考真题答案

一、选择题

1. B（同2010-15）

2. B

3. A

→ 随机访问是顺序表的特性

4. C（同2013-3）

5. A

→ 见参版 P289

6. ~~B~~ A

→ 从A[1][1]开始，$k=\begin{cases}\frac{i(i-1)}{2}+j-1, & i\geq j\\ \frac{j(j-1)}{2}+i-1, & i<j\end{cases}$ 适用于n行n列的矩阵，题干是5行6列

由于是按行存储，$LOC(5,5)=4\times6+5-1=28$，$28\times5+1000=1140$

7. A

→ 当最底层只有一个元素时结点数最少 → 前h-1层+1 $=2^{h-1}-1+1=2^{h-1}$

8. B

→ A，C，D都是石砸说法，这里考查了生成树的概念（极小连通子图），连通分量是极大连通子图。

9. C（同2013-7）

10. D

→ 考查了关键路径的概念

11. D

→ $C_n^2=\frac{n(n-1)}{2}$

12. A

→ 无向图邻接矩阵为对称矩阵，A话确。B改：非零元素个数 = 边数×2。

C改：第i行 和第i列 的非零元素个数相同，均及点V_i的度数，题干是指第i行+第i列 = V_i度数。

D改：矩阵行数 = 图中顶点数。

13. D

→ 设AVL树的高度为h，记此时结点数为N_h

→ ① $h=0$，$N_0=0$

② $h=1$，$N_1=1$

③ $h>1$，$N_h=N_{h-1}+N_{h-2}+1$

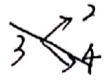
$\Rightarrow h\approx1.44\times\log_2(n+1)$

→ AVL树中每个结点查找长度与本搞是一个数量级，即$O(\log_2n)$

14.（同2015-5）

15. A

→ 考查了栈"先进后出"

b次：1→5→4→6→3→2　　c次：6→5……　　D次：  　3→5→6→4→2→1

→ 判断技巧：① X_n 出栈之后只能与其相邻的元素出栈（X_n 为最后一个元素），如c次.

② 先进后出

二. 填空题.

1. n-i

2. (front+1)%(m+1)　（分母是整个队列存储空间）

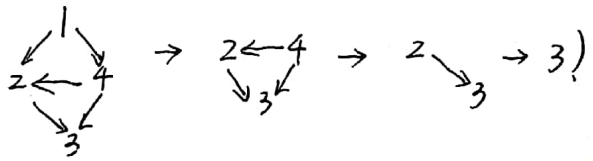
3. p→next=q→next　　free(q)

4. n+1　（见2015年第9题选择题解析）

5. 89

6. $O(\log_2 n)$　　$O(n\log_2 n)$

7. d

8. 1,4,2,3　（  ）

三. 判断题

1. f（同2010-14 → 指2010年判断题14题）

2. t（树高最小二叉排序树查找类似于二分查找，时间复杂度为$O(\log_2 n)$，最坏情况下是单支树为O

3. f

4. t

5. f（数据项是数据的最小单位，数据元素是数据的基本单位）

6. f.（普里姆算法时间复杂度为$O(n^2)$，n为顶点数，与边数无关，故可以用于稠密图

克鲁斯卡尔算法时间复杂度为排序算法决定，排序算法排序的是边的权值，因此与边数有关.

相对于prim而言不适于稠密图，适用于疏稀图）

7. f

8. t

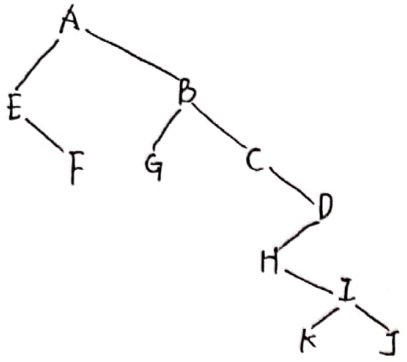
9. t

10. f

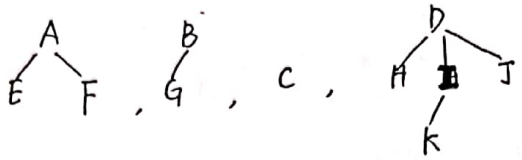
四. 简答题

1.

(1) 二叉树如下：



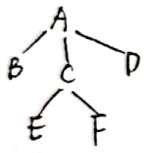
(2) 二叉树转换成森林为



→ 前序+中序 找后序或画二叉树详解见2010-3，下面介绍一下二叉树转森林与森林转二叉树.

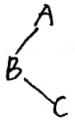
上面两个转化本质是 树与二叉树的转换.

① 树 → 二叉树 （遵循"左孩子，右兄弟"）→ 指孩子放左边，兄弟放右边.



B是A的孩子，A无兄弟

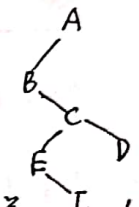
a.



C是B的兄弟，B无孩子

b.

c. E是C的孩子，D是C兄弟



d. F是E的兄弟

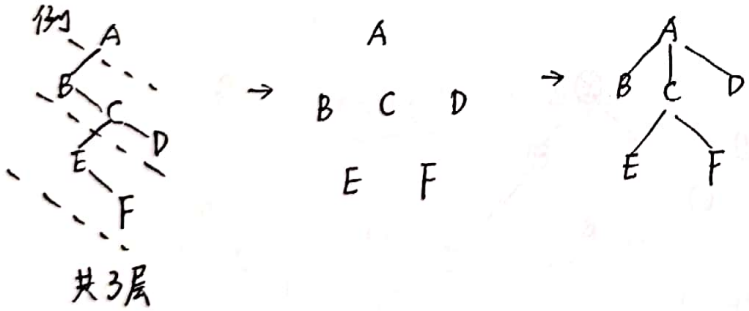
② 二叉树 → 树

a. 先把它从左上到右下分若干层

b. 找到每一层结点在其上一层的父结点

c. 将每一层的结点与其父结点相连

例



共3层

③ 二叉树 → 森林

a. 只需不断将根结点有右子树的右子树与其连接断开，直到不存在根结点为止

b. 将二叉树 → 树 （多棵二叉树）

④ 森林 → 二叉树.

a. 先将多棵树分别转化为二叉树

b. 将二叉树作为第一棵二叉树根的右子树，依次类推.（第n棵二叉树作为第n-1棵二叉树的右子树）

2. 解：由哈希函数 H(key) = key % 10 可得各元素散列地址为

key	71	23	73	14	55	89	33	43	48
Hash(key)	1	3	3	4	5	9	3	3	8

由二次探测法处理冲突得

地址	0	1	2	3	4	5	6	7	8	9	10
key		71	33	23	73	14	55	43	48	89	

各关键字比较次数

key	71	23	73	14	55	89	33	43	48
次数	1	1	2	2	2	1	3	4	1

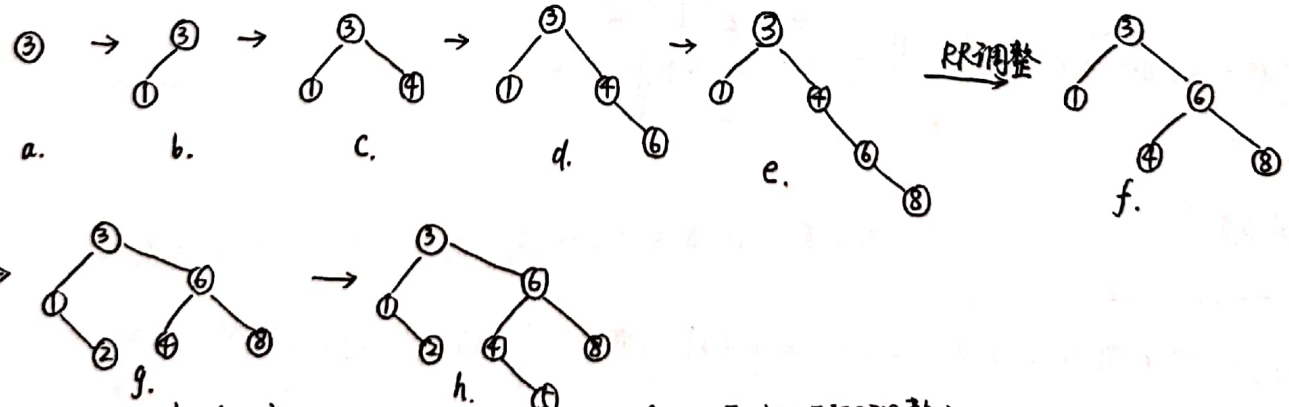
$$ASL_{成功} = \frac{1+1+2+2+2+1+3+4+1}{9} = \frac{17}{9}$$

→ 这题"易错点"在于二次探测处理冲突时表长的确定，指导版书 P257，二次探测再散列只有哈希表长 m 形如 4j+3（j为整数）的素数才能。所以 l=10 不满足条件，于是取 $l_{表}=11$.

→ 二次探测再散列 $H_i = H(key) + d_i$ ，$i=1,2,\cdots,k\ (k \le m-1)$，$d_i = 1^2, -1^2, -2^2, 2^2, \cdots, \pm k^2, k \le \frac{m}{2}$

→ 举个例子，key为33的寻址轨迹为 3→4→2，key为43的寻址轨迹为3→4→2→7.

3. 解：

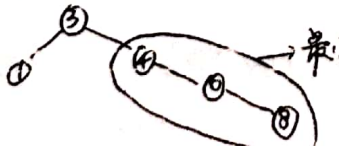


→ 考查平衡二叉树的建（每加入一个元素，就检查是否要进行平衡调整）

→ 下面着重讲平衡调整

→ 概念→ 最小不平衡子树：以距离插入结点最近，且以平衡因子绝对值大于1的结点作为根的子树

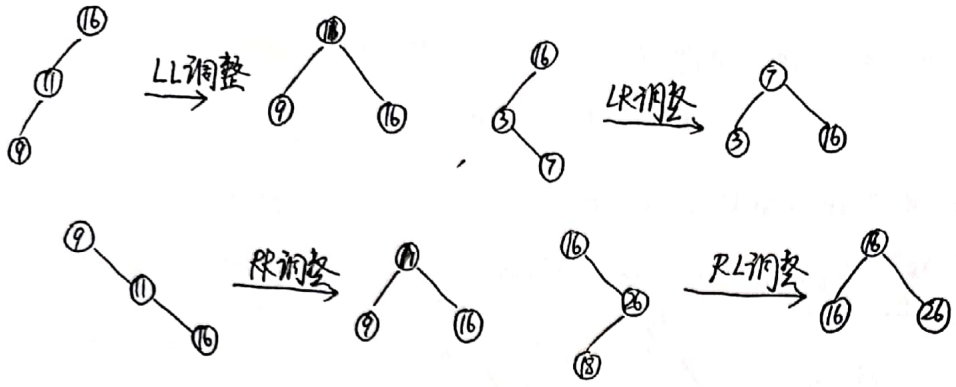
状态e：



四种调整方式（LL，RR，LR，RL）

→ LL调整：新插入结点，落在最小不平衡子树根的左孩子(L)的左子树(L)上，其他命名同理

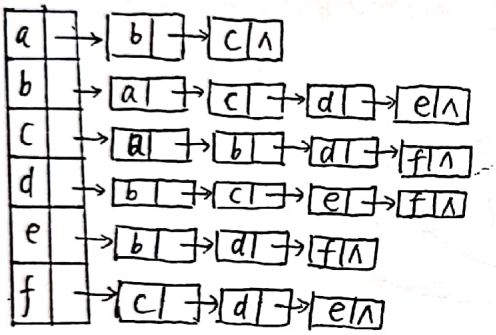
→



4.

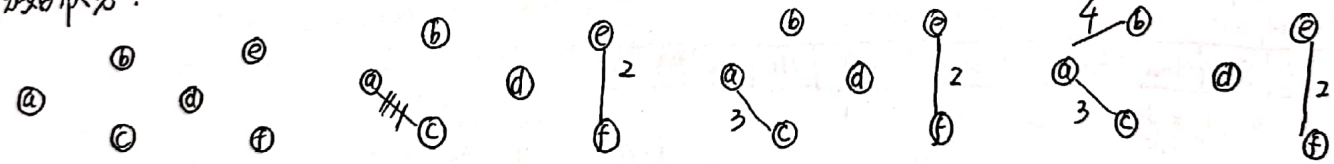
解：

(1) 如图所示：

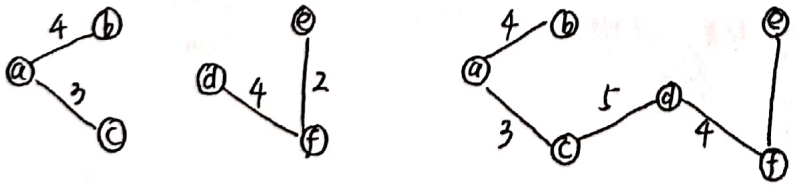


(2) 用克鲁斯卡尔算法求最小生成树

初始状态：



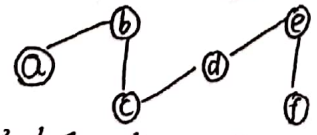
A.　　B. 引入边<e,f>　　C. 引入边<a,c>　　D. 引入边<a,b>



E. 引入边<d,f>　　F. 引入边<c,d>

(3) 从顶点a开始的深度优先搜索和深度优先生成树为

a, b, c, d, e, f



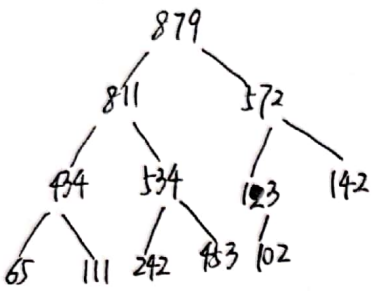
→ (2) 引入边顺序有多种　(3) 答案也很多种可能，这类题见2012-3

5.

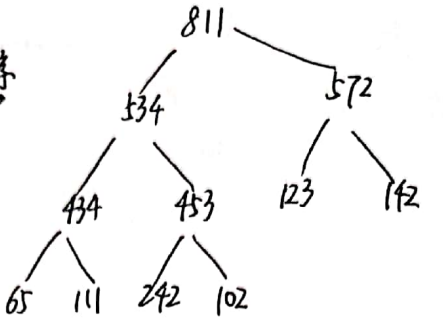
(1) 原始序列： 142 543 123 65 453 879 572 434 111 242 811 102

d=7，第一趟： 142 111 123 65 102 879 572 434 543 242 811 453

(2) 初始堆为

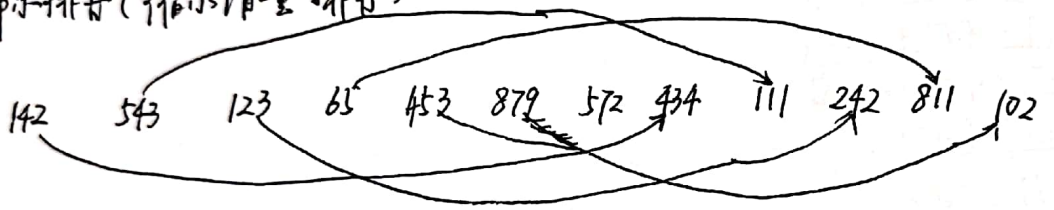


第一趟排序



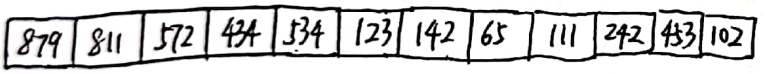
(102与879互换，879到达最终位置，不再参与排序，其他再调整

→ 希尔排序（缩小增量排序）

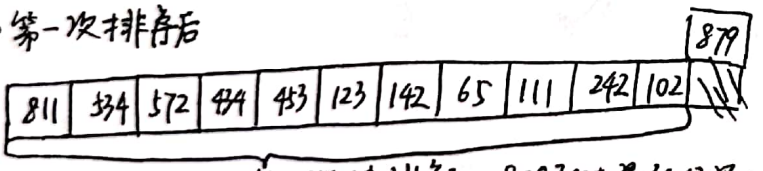


→ 堆排序详见2013-2

→ 初始堆存储结构



→ 第一次排序后



第2趟待排序列，879到达最终位置，不再参与排序

→ 还有一个考点，题干要求“升序排序”那么就要大顶堆

物道.

→ 一个易错点，四的第1小问的初始堆为堆

要满足堆定义（大顶堆/小顶堆）

五、算法填空

1.

① ~~p->next=L->next~~ p->next=L

② ~~L->next=p~~ L=p

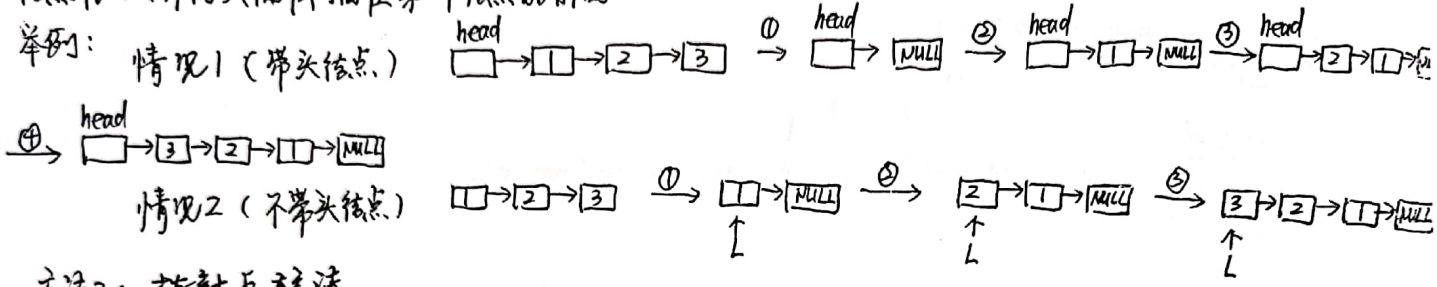
→ 这题考查了 "就地逆置"，有两种方法

方法1：将头结点摘入，然后从第一个结点开始，依次前插入到头结点的后面（头插法），直到最后一个结点为止

题干就运用了方法1，不过稍微做了变化，题干无头结点，那么就从第一个结点开始，将第2个结点及之后的结点依次采用头插法插在第一个结点的前面

举例：

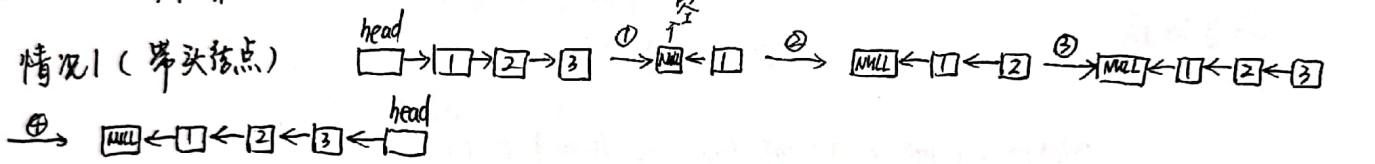
情况1（带头结点）



情况2（不带头结点）

方法2：指针反转法

情况1（带头结点）



情况2.（不带头结点）

下面都是带头结点代码示例

方法1：

```
LinkList Reverse_1(LinkList L){
    //L是带头结点的单链表，将L就地逆置
    Lnode *p,*r; //p为工作指针，r存后继指针
    p=L->next; //从第一个元素结点开始
    L->next=NULL; //将头结点首为NULL
    while(p!=NULL){ //依次将元素结点摘下
        r=p->next; //暂存p的后继
        p->next=L->next; //头插法
        L->next=p;
        p=r;
    }return L;
}
```

方法2：

```
LinkList Reverse_2(LinkList L){
    Lnode *pre,*p=L->next,*r=p->next;
    p->next=NULL; //处理第一个结点
    while(r!=NULL){ //r为空，则说明p为最后一个结
        pre=p;
        p=r;
        r=r->next
        p->next=pre; //指针反转
    }
    L->next=p; //处理最后一个结点
    return L;
}
```

→ 本代码来自<<王道>>，这两种方法在命代码填空题里的可能性比较大。

2.

③ T→lchild

④ p=p→lchild

⑤ p→rchild != T

⑥ p=p→rchild

⑦ p=p→rchild

→素材来源于岁版《数据结构》P134的算法6.5

3. ⑧ T==NULL

⑨ T→lchild

⑩ T→rchild

六、编写算法

1.

```
void quickpass(int r[], int low, int high, int ki)
{ //对r数组来一趟快速排序,枢轴为ki
  int i=low, j=high, x=ki;
  while(i<j){
      while(i<j && r[j]>x) j=j-1;
      if(i<j){
          r[i]=r[j];
          i=i+1;
       }
      while(i<j && r[i]<x) i=i+1;
      if(i<j){
          r[j]=r[i]
          j=j-1;
       }
    r[i]=x; //枢轴归位
}
```

→这是一趟快速排序,时间复杂度为$O(\log_2 n)<O(n)$满足题干.

2.

解：

```
typedef struct {
    char c;      // 字符
    int w;       // 字符权值
    char *code;  // 字符的Huffman编码
} Huffmancode[maxsize];

typedef struct {
    int weight;          // 权值
    int lchild, rchild, parent;
} HTNode, HuffmanTree[maxsize];
```

```
void SelectHTNode(HuffmanTree HT, int n, int *min1, int *min2)
{ // 查找最小和次小、位置下标
  int i;
  *min1 = *min2 = 0;
  for(i=1; i<n; i++)
  { if(HT[i].parent==0)
    { if(HT[*min1].weight >= HT[i].weight)
      { *min2 = *min1;
        *min1 = i;
      }
      else {
        if(HT[*min2].weight > HT[i].weight)
        {*min2 = i;}
      }}
  }
```

(1) 构造Huffman树代码如下：

```
Void CreateHuffmanTree(HuffmanTree HT, int length, HuffmanCode Hc)
{
    int i;
    int min1, min2; // 最小的两个权值
    HT[0].weight = MAX; // MAX是一个很大的值
    for(i=1; i<=length; i++)
    {
      HT[i].weight = HC[i].w; // 哈夫曼编码权值改成值给哈夫曼树里的权值
      HT[i].lchild = HT[i].rchild = HT[i].parent = 0 // 双亲、左右孩子全部改成值为0
    }
    for( ; i<2*length; i++)
    {  HT[i].lchild = HT[i].rchild = HT[i].parent = 0
    }
    for(i = length+1; i< 2*length; i++)
    {
        SelectHTNode(HT, i, &min1, &min2);
        HT[min1].parent = i;
        HT[min2].parent = i;
        HT[i].lchild = min1;
        HT[i].rchild = min2;
        HT[i].weight = HT[min1].weight + HT[min2].weight;
    }
}
```

(2) 生成Huffman编码：

```
void HuffmanCoding1(HuffmanTree HT, int n, Huffmancode HC)
{
    int i, start, c, f;
    char *cd;
    cd = (char*) malloc(n* sizeof(char));
    cd[n-1] = '\0';
    for(i=1; i<=n; i++)
    { start = n-1
      for(c=i; f=HT[i].parent; f!=0; c=f, f=HT[f].parent)
      {
        If(HT[f].lchild == c)
              cd[--start]='0';
        else
            cd[--start] = '1';
        HC[i].code = (char*) malloc((n-start)*sizeof(char));
        strcpy(HC[i].code, &cd[start]);
      }
      free(cd);
}
```

→ 本题最完整标准的答案请参考书版《数据结构》P197，代码比较简洁，但这里的答案是书版的具体实现，更好理解一点（含代码见 https://blog.csdn.net/Xenoverse/article/details/83415597）建议背诵后抄写书版代码!!!