2012年408真题答案.

一、选择题.

1. A

→ 队列：先进先出
栈：先进后出

2. C

→ 考查栈的特性：先进后出 $\rightarrow a \rightarrow e$

→ C项分析：d $\begin{cases} c \rightarrow b \begin{cases} \rightarrow e \rightarrow a \\ \rightarrow e \rightarrow b \rightarrow a \end{cases} \\ e \rightarrow c \rightarrow b \rightarrow a \end{cases}$ ，这是C项可能所有情况，故排除C. C不满足.

3. C

→ $1+2+3+\cdots+n = \frac{n(n+1)}{2}$

4. B

→ 链表易于插入删除

5. A（同2011-9）

6. C → 不稳定排序

→ 记忆口诀：心没，快些送一班朋友玩（快：快速排序，些：希尔排序，送：选择排序，班：归并排序）

7. A（同2010-13）

8. D

A. O(1) B. O(1) B C. $O(\log_2 n)$ D. O(n)

9. (同B2011-14).

10. (同2010-4)

11. B

→ 平衡二叉树是二叉排序树，根据二叉排序树定义：Value(左) < Value(root) < Value(右)

→ 中序遍历是升序的.

12. C（同2010-12）

13. D

→ 要求循环队列未满，队满判断标准：(rear+1)% max = front ，max → 指整个队列存储空间大小.

14. B（同2010-15）

15. D

→ 深度优先与广度优先适用于有向图与无向图

二. 填空题.

1. 6 或 7

→ 这题主要争论点在于冒泡排序是①从小到大排序 ②从大到小排序

2. 线性结构与非线性结构

3. $2n-1$

4. 计算方便　散列地址分布均匀

5. 方便统一单链表的插入删除操作

6. 2^k-1　2^{k-1}

7. $2e$ （一条边代表一个出度，一个入度；针对有向图，无向图一条边对于边两端顶点都算一个度）

8. $p\to lchild == NULL$ 或 $p\to rchild == NULL$

9. 一端　先进后出

10. 021　235　346　256　558

三. 判断题.

1. t （前面讨论过. 树的先序对应二叉树的先序）

2. f （只有该顶点在邻接表与逆邻接表里的链表全为空，该顶点度为0）

3. f （首尾元素情况不一样）

4. t

5. f （同2010-6，说明一下，2010-6指问题型的该年份下的第6题）.

6. f （~~同2010~~ 前面提过，平衡树是二叉排序树）.

7. t

8. f

9. f

10. f （$\frac{n(n+1)}{2}$）

三. 简答题

1. 解：下面是快速排序的具体流程

原始序列：65 57 45 39 12 98 86 35

第一趟：35 57 45 39 12 65 86 98

第二趟：12 35 45 39 57 65 86 98

第三趟：12 35 39 45 57 65 86 98

→ 考查了快速排序的过程，快排属于交换类排序，它的原理是先选择一个枢轴(一般为第一个元素)然后将序列中小于枢轴的关键字放左边，大于枢轴的放右边，这样初步排序后形成了新的两个子序列，然后对新的子序列做同样操作直到整个序列有序。

2. 解：

① while(p->next && p->next->data<=x) p=p->next;
这句代码为了找到恰好大于或等于x的元素的位置.

② while(q->next && q->next->data<y){ s=q; q=q->next; free(s);}
这段代码块是为了删除(x,y)之间的元素.

③ p->next=q->next
这句代码是为了把链重新连接起来.

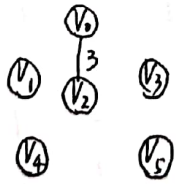
综上所述：该算法的功能是为了删除开序线性表L中值在(x,y)内的值元素.

→ 考查了单链表的删除.

3. 解：

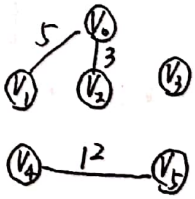
初始状态

a.

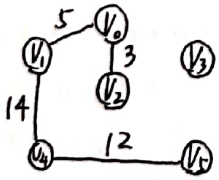


b. 引入边<V0,V2>

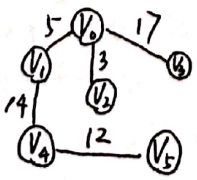
c. 引入边<V0,V1>



d. 引入边<V4,V5>



e. 引入边<V1,V4>



f. 引入边<V0,V3>

→ 这是手写克鲁斯卡尔生成最小生成树的全过程，克鲁斯卡尔原理(每次挑选本值最小的边并入生成树中，但千万注意一个"坑"，加入的边不能使其生成环，如步骤f按道理应该加入边<V2,V5>，然而加入这边后形成环了，生成树中不能有回路(环).

4.

答：

深度优先遍历得到的一种序列：$V_0, V_1, V_4, V_2, V_3, V_5, V_6$

广度优先遍历得到的一种序列：$V_0, V_1, V_2, V_3, V_4, V_5, V_6$

→题干的 a 顶点图中未标出，这里按 V_0 处理

→这里考查了深度优先与广度优先区别，所谓"深度优先"即搜索遍历完一个结点，下一个结点是遍历当前结点的子结点，而广度优先是遍历当前结点的兄弟结点（即同层次），这就是它们的本质区别

5. 解：

设置两个矩阵 A 与 path，A 用于记录当前已经求得的任意两个顶点最短路径的长度，path 用来记录当前两顶点间最短路径上要经过的中间顶点，这里我们设 -1 为无中间点情况，中间点 V_0, V_1, V_2 在 path 分别表示为 0, 1, 2

① 初始情况下：$A_{-1} = \begin{bmatrix} 0 & 8 & 12 \\ 6 & 0 & 3 \\ 5 & 1\infty & 0 \end{bmatrix}$ $path_{-1} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

② 以 V_0 为中间点，检测所有顶点对，假设当前所检测顶点对为 (i, j)，若 A[i][j] > A[i][0]+A[0][j]，则将 A[i][j] 更新为 A[i][0] + A[0][j] 的值，并将 path[i][j] 置为 0

即 $A_0 = \begin{bmatrix} 0 & 8 & 12 \\ 6 & 0 & 3 \\ 5 & 13 & 0 \end{bmatrix}$ $path_0 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$

补充一点如何从最后求得的矩阵 A 和 path 求各顶点间最短路径及长度

比如求 V_0 到 V_2 最短路径

由 $path_2$ 可知 $path_2[0][2] = 1$，说明有中间结点 V_1，再以 V_1 为起点，求 V_1 到 V_2 最短路径 $path_2[1][2] = -1$，无中间结点

即 V_0 到 V_2 最短路径为 $V_0 \to V_1 \to V_2$

长度可以从原图看，也可以看 A_2 矩阵

$A_2[0][1] + A_2[1][2] = 11$

→注意 A 与 path 矩阵是从 0 还是 1 开始，同题意而定，不过原理一样

③ 同理，以 V_1 为中间点可得

$A_1 = \begin{bmatrix} 0 & 8 & 11 \\ 6 & 0 & 3 \\ 5 & 13 & 0 \end{bmatrix}$ $path_1 = \begin{bmatrix} -1 & -1 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$

④ 以 V_2 为中间点可得

$A_2 = \begin{bmatrix} 0 & 8 & 12 \\ 6 & 0 & 3 \\ 5 & 13 & 0 \end{bmatrix}$ $path_2 = \begin{bmatrix} -1 & -1 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$

综上所述：

$V_0 \to V_1$ 最短路径为 $V_0 \to V_1$，长度为 8

$V_0 \to V_2$ 最短路径为 $V_0 \to V_1 \to V_2$，长度为 11

$V_1 \to V_0$ 最短路径为 $V_1 \to V_0$，长度为 6

$V_1 \to V_2$ 最短路径为 $V_1 \to V_2$，长度为 3

$V_2 \to V_0$ 最短路径为 $V_2 \to V_0$，长度为 5

$V_2 \to V_1$ 最短路径为 $V_2 \to V_0 \to V_1$，长度为 13

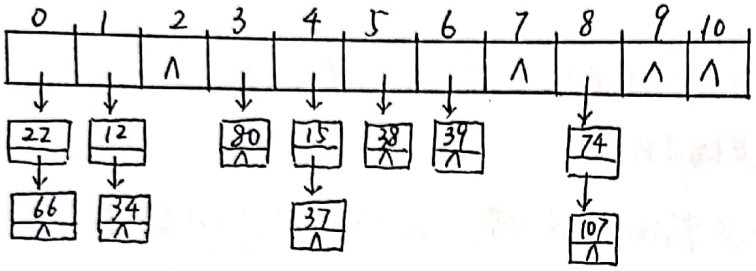
题后小结：本题做法是参考《大数》，目前个人认为是所讲的算法的最直给易懂的做法了，当然有同学觉得又臭又长，其确是有更精炼的算法（参考《王道》），但理解好，考试最重要是得分，不管"黑猫"还是"白猫"，再提一句，强大专业课答题卡是装订好接近 10 页的白纸，不用担心没位置写答案。

6. 解：

由Hash函数可求得各元素存放地址如下（Hash(key) = key mod 11）

关键字	12	15	34	37	39	22	38	66	74	80	107
Hash(key)	1	4	1	4	6	0	5	0	8	3	8

由此构造的链地址处理冲突的Hash表为



$$ASL_{成功} = \frac{1\times7+2\times4}{11} = \frac{15}{11}$$

→ 本题ASL的计算中与各指针计算比较次数不计算在内，下面给个提示，当题干无特殊说明时在以顺序表为存储结构的情况下，如果各位置作为结束标记，则与各位置比较次数也应该算在内；在以链表为存储结构情况下，与各指针比较次数不计算在内 [天勤2019版]

五. 算法填空（16分）

1.

① (QueuePre)malloc(sizeof(QNode))

② NULL

③ Q->rear->next

④ p

→ ① 为新结点分配空间 ④ 队列插入采用尾插法，置新插入结点的后继为空.

③ Q->rear->next = p ④ Q->rear = p 尾插法代码

→ 素材来自于乐服<<数据结构>> P62，只是稍微做了点改变，细心的小伙伴可以看见题干与乐服不一样参数Q少了"&"，少了引用，所以乐服的④ Q.rear=p. 而题干都是Q->rear=p，下面重本溯原！

首先从题干的结构体 typedef struct{ QueuePtr front; QueuePtr rear; }LinkQueue, *LinkQueuePtr; 说起，很多人分不清LinkQueue与*LinkQueuePtr，其实前者是结构体变量，后者为结构体指针变量，通俗来讲 LinkQueue* p 与 LinkQueuePtr p，两个p都是一个意思，再回到EnQueue()的参数 LinkQueuePtr Q，这个Q就等效于上面的p，我们知道结构体指针访问结构体内变量时用"->"，而结构体变访问时用"."，自然答案用"->".

再延伸一个问题，大家对给结点分配时空间这句" p=(QueuePtr)malloc(sizeof(QNode))"肯定有过疑惑，因为它还有另一种写法 p=(QNode*)malloc(sizeof(QNode))，大家知道malloc()作用是申请一个或一块（看sizeof前面是否有k与之相乘.即 p=(QNode*)malloc(k*sizeof(QNode))为申请一块空间，可以存k个结点）空间，用一个指针指向这个空间，上面已经讲过 QueuePtr 等价于 QNode*，这种写法自然可以，大家要擦亮眼睛，一个空2分

2.

① !StackEmpty(S) || p

② Push(S,p)

③ Pop(S,p)

④ p=p->rchild

→ 先序遍历非递归有三种写法，参考博客（https://blog.csdn.net/weixin_42130471/article/details/80319821）

① 进入循环p不为空或栈S不为空 ② 操作是为了保留访问右子树的地址

→ 先序遍历原则是 先访问根结点，再访问左右子树 → 对每棵子树这样做

六. 编写算法.

1.

解法一:

```
int  n=0 ;
Void  count ( BTNode  * p )
{    if( p!=NULL)
     {
        if( p→lchild==NULL && p→rchild==NULL)
        {
            ++n;
        }
        count( p→lchild);
        count ( p→rchild);
     }
}
```

→ 解法一很好理解，就是一个先序遍历整个树再加一些限制条件来求出树中叶子数.

解法二:

```
int   count( BTNode  *p)
{
      int  n1, n2 ;
      if ( p==NULL)
           return 0 ;
      else  if ( p→lchild==NULL && p→rchild==NULL)
           return 1 ;
      else
      {
          n1 = count( p→lchild);
          n2 = count( p→rchild);
          return  n1+n2;
      }
}
```

解法二也不难理解 → 树中叶子结点个数 = 根的左子树中叶子数 + 根的右子树的叶子数

2.

```
Status  CreateUDN (MGraph &G){
    // 采用数组(邻接矩阵)表示法,构造无向图G
    Scanf(&G.vexnum, &G.arcnum, &IncInfo); // IncInfo为0则各弧不含其他信息
    for(i=0; i<G.vexnum; ++i) Scanf(&G.vexs[i]); // 构造顶点向量
    for(i=0; i<G.vexnum; ++i)  //初始化邻接矩阵
        for(j=0; j<G.vexnum; ++j) G.arcs[i][j] = {INFINITY, NULL};
    for(k=0; k<G.arcnum; ++k){ // 构造邻接矩阵.
        Scanf(&V1, &V2, &W);
        i= LocateVex(G, V1); j = LocateVex(G, V2) // 确定V1,V2在G中位置
        G.arcs[i][j].adj = W;
        if(IncInfo) Input(*G.arcs[i][j].Info) // 若有弧相关信息,则输入.
        G.arcs[j][i] = G.arcs[i][j]   // 置<V1,V2>的对称弧<V2,V1>
    }
    return OK
}
```

→ 答案来自于罗版<<数据结构>> P162的算法7.1，注释多，不作多余说明！