卷子

2018年6月13日 16:50

1. 若线性表采用顺序存储结构存放，在长度为m的线性表中第j（1≤j≤m）个数据元素之前插入一个新的数据元素需要移动（C）个数据元素。

A. j　B. m-j　C. m-j+1　D. m-j-1

2. 高度为h（h≥1）的完全二叉树至少有（C）个结点。

A. 2^h-1　B. 2^{h-1}　C. $2^{h-1}+1$　D. 2^h

3. 下面（B）方法可判断出一个有向图是否有回路。

A. 迪杰斯特拉算法　B. 拓扑排序　C. 弗洛伊德算法　D. 普里姆算法

4. 对n个数据元素进行　　排序，最坏情况下进行（　　）次数据元素间的比较，（　　）次数据元素移动。

5. 设有一个栈，元素的进栈次序为a,b,c,d,e,f，下列（B）为可能的出栈序列。

A. cdaefde　B. cbafed　C. cabfed　D. abfdec

6. 已知关键字序列为64、35、96、59、88、31、81、43，以位于最左位置的数据元素为基准进行快速排序，第一趟快速排序的结果为（A）。　43 35 31 59　64　88 81 96

A. 43 35 31 59 64 88 81 96　B. 59 43 35 31 64 88 81 96

C. 31 35 43 59 64 81 88 96　D. 43 31 35 59 64 88 96 81

7. 以下排序方法中，稳定的排序方法（D）

A. 直接插入、希尔　B. 堆排序、归并

C. 基数、堆排序　D. 直接插入、归并

8. 设A是一个40阶的矩阵，$A=(a_{ij})_{40\times40}$，采用压缩存储将某个三角部分以行为主序存储在一维数组F[]中，a_{11}在F[0]，则a_{ij}在F[k]，$1\le j\le i\le 40$，k=（D）

A. j(j-1)/2+i-1　B. j(j+1)/2+i　C. i(i+1)/2+j　D. i(i-1)/2+j-1

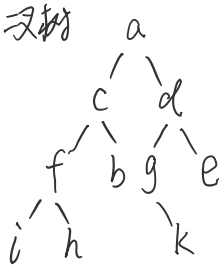
a_{ij}　$\frac{(i-1)(i+1)}{2}+j-1$

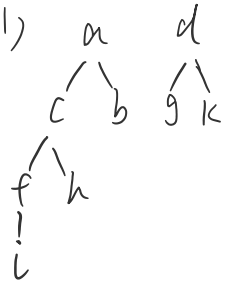
一森林的先序遍历序列为 a c f i h b d g k e，中序为 i f h c b a g k d e

小）画出该森林　（2）给出该森林第一棵树的双亲表示

data parent

1) 画出该森林 (2) 给出该森林第一棵树的双亲表示

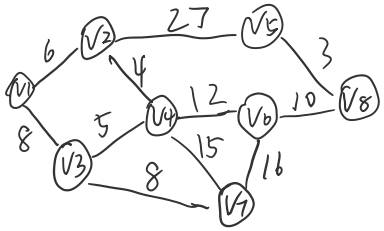


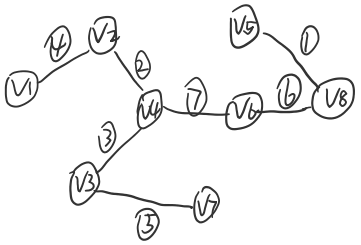
1) 

2)

	data	parent
0	a	-1
1	c	0
2	b	0
3	f	1
4	h	1
5	i	3

无向图如下图所示，采用克鲁斯卡尔算法求最小生成树，
请画出得到的最小生成树，并给出加边顺序



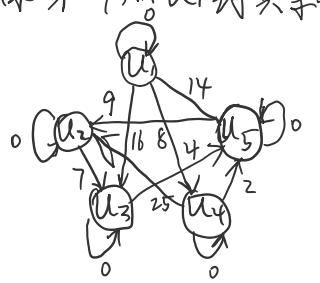


有5个顶点{u_1, u_2, u_3, u_4, u_5}的带权有向图的邻接矩阵

1) 根据此邻接矩阵画出相应的带权有向图

2) 利用迪杰斯特拉算法求第一个点 u_1 到其余各点的最短路径，并给出计算过程

$$\begin{bmatrix} 0 & \infty & 16 & 8 & 14 \\ \infty & 0 & 7 & \infty & \infty \\ \infty & \infty & 0 & \infty & 4 \\ \infty & 25 & \infty & 0 & 2 \\ \infty & 9 & \infty & \infty & 0 \end{bmatrix}$$



u_2：(u_1, u_4, u_5, u_2)

u_3：(u_1, u_3)

u_4：(u_1, u_4)

u_5：(u_1, u_4, u_5)

终点	i=1	i=2	i=3	i=4
u_2	∞	33 (u_1, u_4, u_2)	19 (u_1, u_4, u_5, u_2)	19 (u_1, u_4, u_5, u_2)
u_3	16 (u_1, u_3)	16 (u_1, u_3)	16 (u_1, u_3)	
u_4	8 (u_1, u_4)			
u_5	14 (u_1, u_5)	10 (u_1, u_4, u_5)		
u_j	u_4	u_5	u_3	u_2
S	{u_1, u_4}	{u_1, u_4, u_5}	{u_1, u_3, u_4, u_5}	{u_1, u_2, u_3, u_4, u_5}

关键字序列为{57, 81, 43, 55, 30, 63, 65}，表长m=11，哈希函数为H(key)= key mod 11

1) 采用二次探测再散列解决冲突，请画出相应的哈希表

(1) 采用二次探测再散列解决冲突，请画出相应的哈希表

(2) 查找63和65分别需要和哪些数据元素比较

0	1	2	3	4	5	6	7	8	9	10
55		57	65	81				30	63	43

63：30, 63

65：43, 55, 63, 65

按以下次序输入关键字的值建立2-3树（3阶B-树）

{85, 51, 69, 39, 71, 93, 104, 25}

(1) 请画出所建的2-3树T_1 (2) 画出在T_1上删除71后的2-3树T_2

(3) 画出在T_2删除25后的2-3树T_3

(1)

(2)

(3)

设待排序的数据元素：77, 64, 98, 47, 85, 53, 102, 88, 60 按照递增序排序

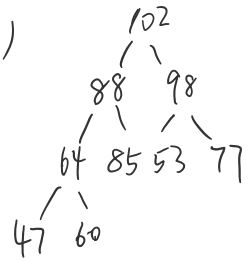
(1) 给出冒泡排序的第一趟、第二趟排序结果

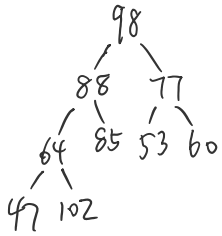
(2) 画出初建的大顶堆，并给出堆排序的第一趟、第二趟排序结果

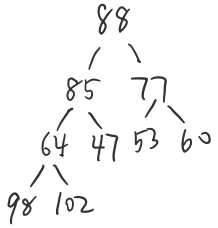
(1) 64 77 47 85 53 98 88 60 [102]

64 47 74 53 85 88 60 [98 102]

(2)

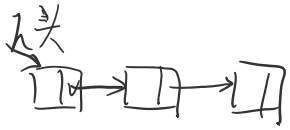






用类C语言设计算法，在头指针为h的带表头结点的有序（递减）单链表中插入一个新的数据元素a，要求插入后链表仍有序。

```
void Insert(Linklist h, Elemtype a)
{
    Linklist p = h;
    Linklist s = (Linklist)malloc(sizeof(LNode));
    s->data = a;
    while(p->next && p->next->data > a)
```



头结点保证算法一致

if (!p->next) 插a入p后 ✓

```
    s->data = a;
    while ( p->next && p->next->data > a)
        p = p->next;
    s->next = p->next;
    p->next = s;
    return;
}
```

while (p->next && p->next->data < a)

找a的p后

设二叉树以二叉链表形式存放，用类C语言设计非递归算法输出根结点为r的二叉树中层次值最大的一层含有的结点数。

```
int levelOrder(TreeNode *root) {
        if (root == nullptr) return 0;
        queue<TreeNode*> q1, q2;//用来切换的两个队列
        q1.push(root);
        TreeNode *curr = nullptr;
        int count = 0;
        while (!q1.empty() || !q2.empty()) {
            if (!q1.empty()) ++count;
            while (!q1.empty()) {
                curr = q1.front();//打印
                q1.pop();
                if (curr->left != nullptr) {
                    q2.push(curr->left);
                }
                if (curr->right != nullptr) {
                    q2.push(curr->right);
                }
                /*
                if (curr->left == nullptr && curr->right == nullptr) {//遇到叶子结点就停止遍历，返回层数（最小深度）
                    return count;
                }*/
            }
            if (!q2.empty()) ++count;
            while (!q2.empty()) {
                curr = q2.front();//打印
                q2.pop();
                if (curr->left != nullptr) {
                    q1.push(curr->left);
                }
                if (curr->right != nullptr) {
                    q1.push(curr->right);
                }
               /*
                if (curr->left == nullptr && curr->right == nullptr) { //返回最小深度
                    return count;
                }
                */
            }
        }
        return count; //返回最大深度
}
```

全是叶子结点，则输出个数