厦门大学《数据结构》期末试题·答案

考试日期：2007•1（B）　　信息学院自律督导部

一、（本题 15 分）请利用两个栈 S1 和 S2 来模拟一个队列。已知栈的三个运算定义如下：Push(Stack ST,int x):元素 x 入 ST 栈；Pop(Stack ST,int x)：ST 栈顶元素出栈，赋给变量 x；StackEmpty(Stack ST)：判 ST 栈是否为空。那么如何利用栈的运算来实现该队列的三个运算：EnQueue:插入一个元素入队列； DeQueue:删除一个元素出队列；QueueEmpty：判队列为空。

解：利用两个栈 S1、S2 模拟一个队列（如客户队列）时，当需要向队列中输入元素时，用 S1 来存放输入元素，用 push 运算实现。当需要从队列中输出元素时，到栈 S2 中去取，如果 S2 为空，则将 S1 中的元素全部送入到 S2 中，然后再从 S2 中输出元素。判断队空的条件是：S1 和 S2 同时为空。

```
Status EnQueue(DataType x)
{
    if StackFull(S1){   //S1 栈满
        if StackEmpty(S2){   // S1 栈满，S2 栈空
            while (!StackEmpty(S1)){ Pop(S1, y); Push(S2, y); //栈 S1 的内容反向搬到栈 S2
            Push(S1, x); return OK;
        }
        else //S1 栈满，S2 栈非空，则不可进行插入操作
            return ERROR;
    }
    else{      //S1 栈不满，则直接进栈
        Push(S1, x);
        return OK;
    }
}


Status DeQueue(DataType &x)

{
    if !StackEmpty(S2)   {
        Pop(S2, x);return OK;
    }
    else{
        if !StackEmpty(S1){
            while (!StackEmpty(S1)){ Pop(S1, y); Push(S2, y);} //栈 S1 的内容反向搬到栈 S2
            Pop(S2, x); return OK;
        }
        else     //栈 S1 和 S2 都为空
            return ERROR;
    }
```

}

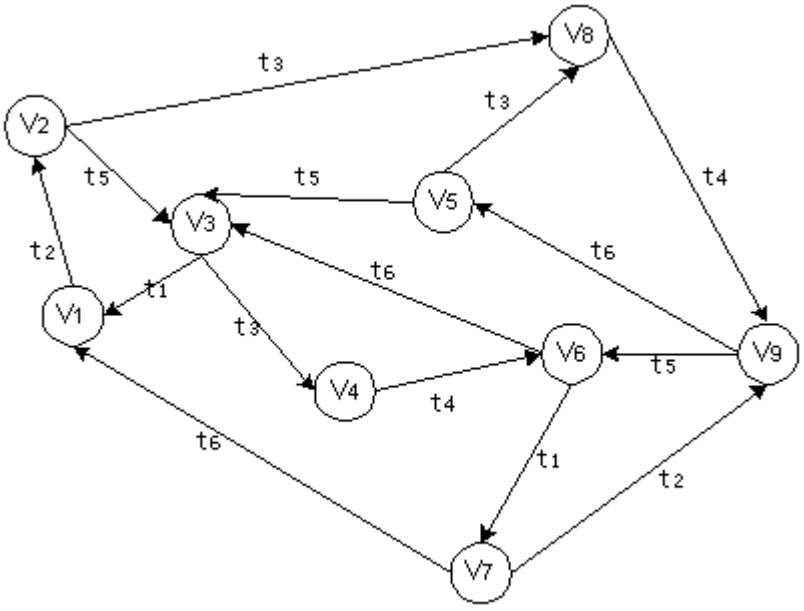
二、（本题 15 分）用孩子兄弟链表作为树的存储结构，设计算法求出树的深度。
解：算法思路：一棵树的深度可以递归定义为：若树为空，则深度为 0，否则树的深度为根结点的所有子树深度的最大值加 1。
数据结构为：

```
typedef struct CSNode{
    ElemType data;
    struct CSNode *firstchild, * nextsibling;
} CSNode, *CSTree;
```

算法如下：

```
int depth(CSNode * t)
{
    CSNode *p; int m, d;
    if (t==NULL) return 0;
    p=t→firstchild; m=0;
    while (p) {
        d=depth(p);
        if (d>m) m=d;
        p=p→nextsibling;
    }
    return m+1;
}
```

三、（本题 15 分）已知在某并发处理系统的 Petri 网基础上建立的可达图如下图所示。图中每个顶点表示系统运行中的一种状态，有向边(弧)表示事件(用 t 表示)，例如有向边(V_i，V_j)表示系统在状态 V_i 时出现该事件将引发系统由状态 V_i 到状态 V_j。



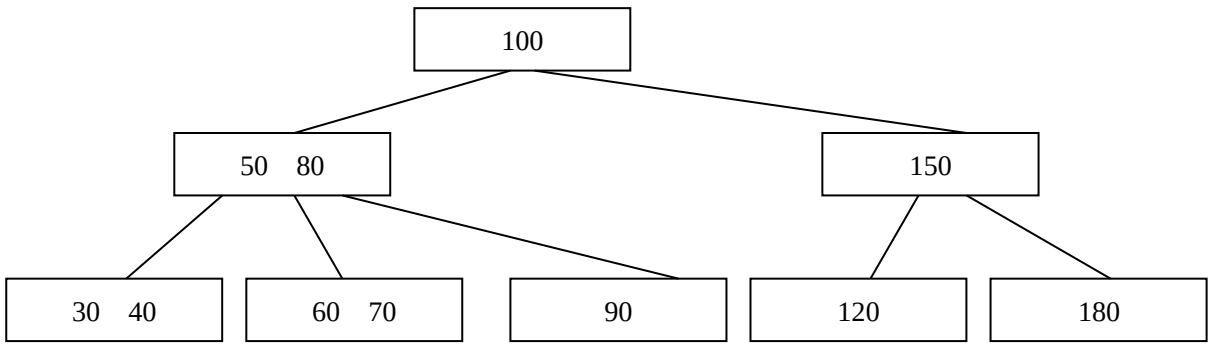
（1）请分别给出该可达图的邻接表、邻接矩阵以及邻接矩阵的三元组 3 种存储表示的形式说明和存储结果示意图，要求每种存储结构能够表达出该可达图的全部信息，并分别对这三种形式中每个部分的含义予以简要说明。
（2）若假设每个域（包括指针域）的长度为 2 个字节，请分别计算出这三种结构所占用的空间大小。

四、（本题 15 分）设计一个算法，判断无向图 G（图中有 n 个顶点）是否是一棵树。

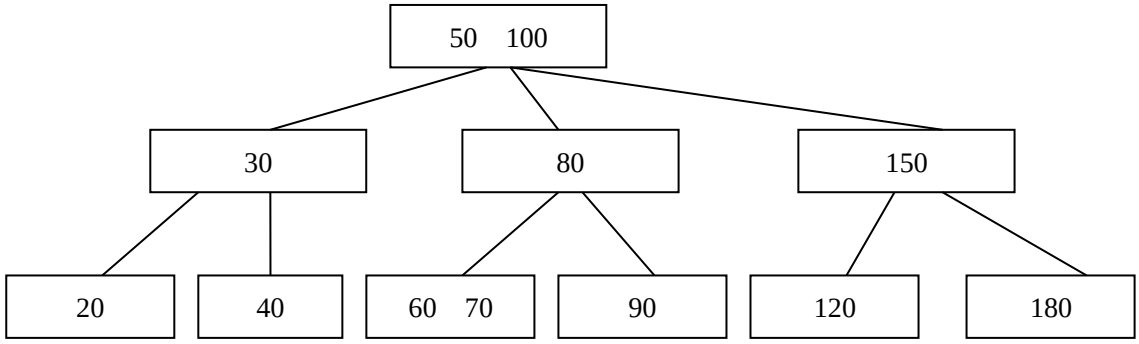
解：算法思路：从第 v 个顶点出发，对图进行深度优先搜索。若在算法结束之前，又访问了某一已访问过的顶点，则图 G 中必定存在环，G 不是一棵以 v 为根的树。若在算法结束之后，所访问的顶点数小于图的顶点个数 n，则图 G 不是连通图，G 也不是一棵以 v 为根的树。

```
Boolean visited[MAX] ;                //用于标识结点是否已被访问过
Status ( * VisitFunc) (int v);   //函数变量
void DFSTraverse( Graph G, Status ( * VisitFunc) (int v));
  { VisitFunc = Visit;
      for ( v=0; v <G.vexnum; ++v )    visited[v] = false;
      if (DFS(G, v )==FALSE) return FALSE;
      for ( v=0; v <G.vexnum; ++v )
          if (visited[v] == false) return FALSE;
      return OK;
   }
 Status DFS( Graph G,    int v );
 { Visited[v] = true;    VisitFunc(v);
      for ( w = FirstAdjVex(G, v) ; w ; w = NextAdjVex(G, v, w) )
              if (Visited[w]) return FALSE;
                   else DFS(G, w );
     return OK;
}
```

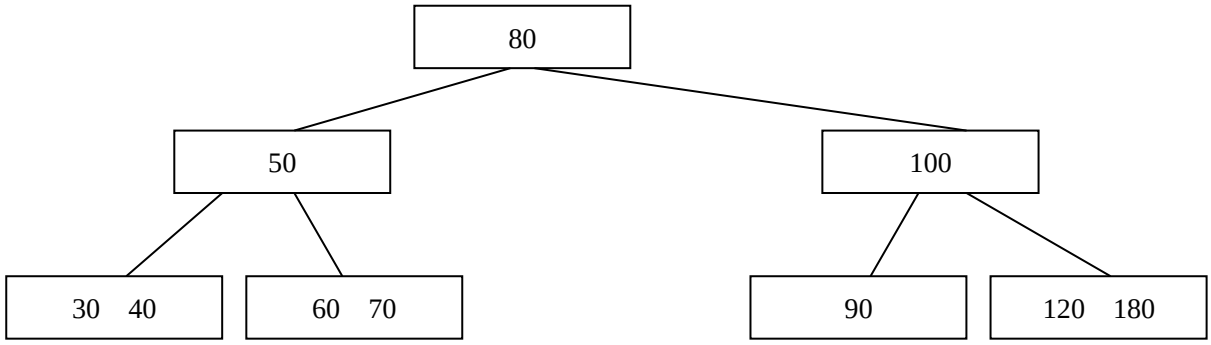
五、（本题 15 分）
（1）设有 3 阶 B－树，如下图所示，分别画出在该树插入关键字 20 和在原树删除关键字 150 得到的 B－树。



（2）包括 n 个关键字的 m 阶 B-树最大深度是多少？（要求写出推导过程）
解：（1）插入 20 后的 B－树为：



删除 150 后的 B－树为：



（2）根据 B-树的定义，第一层至少有 1 个结点；第二层至少有 2 个结点；由于除根之外的每个非终端结点至少有$\lceil m/2 \rceil$棵子树，则第三层至少有 2$\lceil m/2 \rceil$个结点；……；依次类推，第 L+1 层至少有 2($\lceil m/2 \rceil$)$^{t-1}$ 个结点。而 L+1 层的结点为叶子结点。若 m 阶 B-树种具有 n 个关键字，则叶子结点即查找不成功的结点为 n+1，由此有：

N+1>=2($\lceil m/2 \rceil$)$^{t-1}$

反之，$l \le \log_{\lceil m/2 \rceil}(\frac{N+1}{2})+1$

因此，含有 n 个关键字的 B-树，最大深度为$\log_{\lceil m/2 \rceil}(\frac{N+1}{2})+1$。

六、（本题 10 分）设关键字序列为：49，38，66，80，70，15，22。

（1）用直接插入排序法进行排序，写出每趟的结果。

（2）采用待排序列的第一个关键字作为枢轴，写出用快速排序法的一趟和二趟排序之后的状态。

解：（1）直接插入排序法

原始关键字序列为：(49)　38　66　80　70　15　22

(38　49) 66　80　70　15　22

(38　49　66) 80　70　15　22

(38　49　66　80) 70　15　22

(38　49　66　70　80) 15　22

(38　49　66　70　80) 15　22

(15　38　49　66　70　80) 22
(15　22　38　49　66　70　80)

（2）快速排序法
原始关键字序列为：49，38，66，80，70，15，22
第一趟排序后　　22　38　15　(49)　70　80　66
第二趟排序后　　15　(22)　38　66　(70)　80

七、（本题 15 分）有一种简单的排序算法，叫做计数排序。这种排序算法对一个待排序的表进行排序，并将排序结果存放到另一个新的表中。必须注意的是，表中所有待排序的关键字互不相同。计数排序算法针对表中的每个记录，扫描待排序的表一趟，统计表中有多少个记录的关键字比该记录的关键字要小。假设针对某一个记录，统计出的计算值为 c，那么这个记录在新的有序表中的合适的存放位置为 c+1。
（1）编写实现计数排序的算法；
（2）分析该算法的时间复杂性。
解：（1）假设数据结构如下：

```
#define MAXSIZE    20
typedef int KeyType；
typedef struct {
    KeyType    key;
    InfoType otherinfo;
} RedType;
typedef struct {
    RedType     r [MAXSIZE + 1 ] ; // r[0]  空或作哨兵
    int                    length;
} SqList;

void CountSort(SqList L1, SqList L2)
{//把 L1 计数排序后，结果放在 L2
     int i, j, n, count;
     n=L1.length;
     L2.length=L1.length;
     for (i=1; i<=n; i++){
          count=0;
          for (j=1; j<=n; j++)
               if (L1.r[j]<L1.r[i]) count++;
          L2.r[count+1]=L1.r[i];
     }
}
```

（2）基本操作是关键字比较操作和记录移动操作。其中关键字比较操作为 $O(n^2)$，记录移动操作为 $O(n)$。因此，总的时间复杂性为 $O(n^2)$。