哈尔滨工业大学

2008 年春季学期数据结构与算法 试卷 A 参考答案

一、填空题（每空 2 分，共 28 分）

1. (1+2+3+4+5+6+7)+5=33
2. 广义表第一个元素为表头，其余元素组成的表为表尾，如果只有一个元素，则表尾为空即()。A=((a,b,c),(d,e,f))，tail(A)=((d,e,f))，tail(tail(A))=()，head(tail(tail(A)))=()
3. O(1)　O(n)
4. 23 12 3 * 4 / 34 5 * 7 / + + 108 9 / +
5. 3
6. 99　+∞
7. 如果完全二叉树结点的编号从 1 开始，则当 $\lceil \log_2 i \rceil$ 等于 $\lfloor \log_2 j \rfloor$ 时，两者在同一层。
8. 4　WPL=7*4+19*2+2*5+6*4+32*2+5*3+21*2+10*4=261
9. 9　n
10. 2　4　7
11. (15，20，50，40)
12. 0　回路　环路
13. 1 + 2 + … + k-1 = k(k-1)/2
14. 设 n 为总结点个数，n_0 为叶子结点(即度为 0 的结点个数)，则有：

$n=n_0+n_1+n_2+\cdots+n_m$ (1)

又有（分支总数）：$n-1=n_1*1+n_2*2+n_3*3+\cdots+n_m*m$ (2)

（因为一个结点对应一个分支）

式(2)-(1)得：

$1=n_0-n_2-2n_3-\cdots-(m-1)n_m$

则有：$n_0=1+n_2+2n_3+\cdots+(m-1)n_m$

二、简答题（共 32 分）

1. 线索二叉树查找前驱和后继

（1）中序线索二叉树：若结点的 ltag=1，lchild 指向其前驱；否则，该结点的前驱是以该结点为根的左子树上按中序遍历的最后一个结点。若 rtag=1，rchild 指向其后继；否则，该结点的后继是以该结点为根的右子树上按中序遍历的第一个结点。

求后继的算法如下：

```
1    bithptr*INORDERNEXT(bithptr*p)
2    {
3    if(p->rtag==1)
4    return(p->rchild);
5    else
6    {
7    q=p->rchild;/*找右子树最先访问的结点*/
8    while(q->ltag==0)
9    q=q->lchild;
```

```
10    return(q);
11    }
12    }
```

求前驱的算法如下：

```
1     bithptr*INORDERNEXT(bithptr*p)
2     {
3     if(p->ltag==1)
4     return(p->lchild);
5     else
6     {
7     q=p->lchild;/*找左子树最后访问的结点*/
8     while(q->rtag==0)
9     q=q->rchild;
10    return(q);
11    }
12    }
```

2. Kruskal 算法是从空图出发，由生成森林到生成树。它是精确算法，即每次都能求得最优解，但对于规模较大的最小生成树问题，求解速度较慢，算法的总的计算量为 $m\log_2 m+n+m+n(n-1)\sim n^2\log_2 n$。

Prim 算法同样是从空图出发，将点进行二分化，从而逐步加边得到最小生成树。它是近似求解算法，虽然对于大多数最小生成树问题都能求得最优解，但相当一部分求得的是近似最优解，具体应用时不一定很方便。但是它可以看作是很多种最小树算法的概括，在理论上有一定的意义。算法的总计算量约为 n^2。

破圈法是从图 G 出发，逐步去边破圈得到最小生成树。它最适合在图上工作，当图较大时，可以几个人同时在各个子图上工作，因此破圈法在实用上是很方便的。算法总的计算量为 n^3。

3. 如果在待排序序列的后面的若干排序码比前面的排序码小，则在冒泡排序的过程中，排序码可能向与最终它应移向的位置相反的方向移动。如下面示例所示：

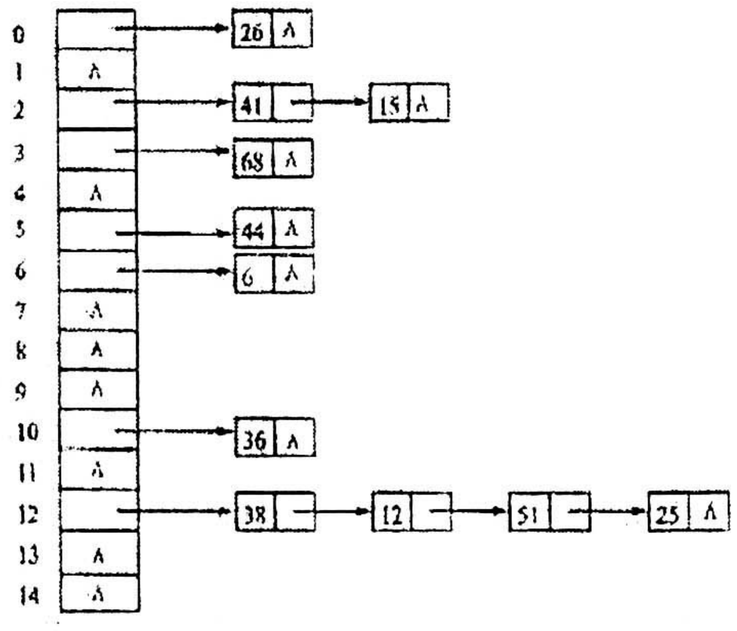
57	40	38	11	13	34	48	75	6	19	9	7	如 9 向相反方向移动
6	57	40	38	11	13	34	48	75	7	19	9	如 19 向相反方向移动
6	7	57	40	38	11	13	34	48	75	9	19	如 9 向最终方向移动
6	7	9	57	40	38	11	13	34	48	75	19	如 13 向相反方向移动
6	7	9	11	57	40	38	13	19	34	48	75	如 13 向最终方向移动
6	7	9	11	13	57	40	38	19	34	48	75	如 34 向相反方向移动
6	7	9	11	13	19	57	40	38	34	48	75	
6	7	9	11	13	19	34	57	40	38	48	75	

4. A B F J　　E D H　　C K G

5. 由α=0.75，得表长 m=11/0.75≈15。

(1)在一般情况下，H(K)=K MOD P 中，P 取质数或者不包含小于 20 的质因数的合数，因此选择 P=13，散列函数 H(K)=K MOD 13。

(2)散列表：



(3)等概率情况下查找成功的平均查找长度：ASL=(1×7+2×2+3×1+4×1)/11=18/11。

(4) 等概率情况下查找不成功的平均查找长度：ASL=(1×5+2×1+4×1)/13=11/13。

三、算法设计题（每小题 10 分，共 20 分）

1. 采用链式存储实现队列的初始化、入队、出队操作

```
#include<stdio.h>
#include<stdlib.h>
#define OK 1
#define OVERFLOW 2
#define ERROR -1
typedef int QElemType;
typedef int Status ;
typedef struct QNode{
    QElemType data;
    struct QNode *next;
}QNode,*QueuePtr;
typedef struct {
    QueuePtr front;
    QueuePtr rear;
}LinkQueue;
Status InitQueue(LinkQueue &Q){
    Q.front=(QueuePtr)malloc(sizeof(QNode));
    Q.rear=(QueuePtr)malloc(sizeof(QNode));
    if(!Q.front)exit(OVERFLOW);
    Q.front=Q.rear;
    return OK;
}
Status Chushihua(LinkQueue &Q){
```

```
QueuePtr p;
int e;
printf("请输入元素，以-1 结束\n");
while(scanf("%d",&e),e!=-1){
    p=(QueuePtr)malloc(sizeof(QNode));
    p->data=e;
    p->next=NULL;
    Q.rear->next=p;
    Q.rear=p;
}
p=Q.front->next;
while(p!=NULL)
{
    printf("%d ",p->data);
    p=p->next;
}
printf("\n");
    return OK;
}
void tishi()
{
    printf("所有操作如下：\n");
    printf(" (1) 采用链式存储实现队列的初始化操作。\n");
    printf(" (2) 采用链式存储实现队列的入队操作。\n");
    printf(" (3) 采用链式存储实现队列的出队操作。\n");
    printf(" (-1) 退出\n");
    printf("请选择：");
}
Status DeQueue(LinkQueue &Q){
    QueuePtr p;
    if(Q.front==Q.rear)
    {
    printf("队列为空!!!!!!!!\n");
    return ERROR;
}
```

```
    p=Q.front->next;
    printf("%d\n",p->data);
    Q.front->next=p->next;
    if(Q.rear==p)Q.rear=Q.front;
    free(p);
    return OK;
}
Status EnQueue(LinkQueue &Q){
    QueuePtr p;
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p)exit(OVERFLOW);
    printf("请输入要入队的元素：");
    scanf("%d",&p->data);
    Q.rear->next=p;
    p->next=NULL;
    Q.rear=p;
    return OK;
}
int main()
{
    LinkQueue q;
    InitQueue(q);
    int m;
    do {
        tishi();
        scanf("%d",&m);
        switch(m){
        case 1:
            Chushihua(q);
            break;
        case 2:
            EnQueue(q);
            break;
        case 3:
            DeQueue(q);
```

```
            break;
        }
    }while(m!=-1);
    return 0;
}
```

2.

```
typedef struct{ Node * p; int rvisited; }SNode //Node 是二叉树的结点结构，
rvisited==1代表p所指向的结点的右结点已被访问过。
lastOrderTraverse(BiTree bt){
    //首先，从根节点开始，往左下方走，一直走到头，将路径上的每一个结点入栈。
    p = bt;
    while (bt){
        push(bt, 0); //push到栈中两个信息，一是结点指针，一是其右结点是否被访问过
        bt = bt.lchild;
    }
    while (!Stack.empty()){ //只要栈非空
        sn = Stack.getTop(); // sn是栈顶结点
        if (!sn.p.rchild || sn.rvisited){
            p = pop();
            visit(p);
        }
        else //若它的右孩子存在且rvisited为0，说明以前还没有动过它的右孩子，于是就去
处理一下其右孩子。
        {
            //此时我们要从其右孩子结点开始一直往左下方走，直至走到尽头，将这条路径上
的所有结点都入栈。
            sn.rvisited = 1;
//往左下方走到尽头，将路径上所有元素入栈
            p = sn.p.rchild;
            while (p != 0){
                push(p, 0);
                p = p.lchild;
            }
        }
    }
}
```