2010-2011 学年 第 2 学期 数据结构 期末参考答案

一、填空题(1 ´ 20 = 20 分)

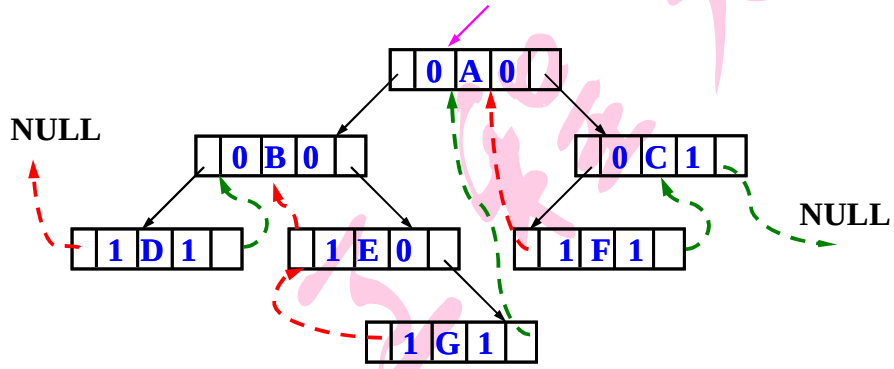
1．数据元素
2．线性、图形(网状)
3．链接、散列(哈希)
4．基本操作集
5．算法
6．O(*n*)
7．栈、LIFO、队列、FIFO
8．**0*x***1118
9．$k = \frac{i(i-1)}{2} + j - 1$
10．HFEADBGC
11．11
12．8
13．6
14．12
15．AEDCB

二、选择题(2 ´ 15 = 30 分)

AACBC　　CCDBD　　DABCB

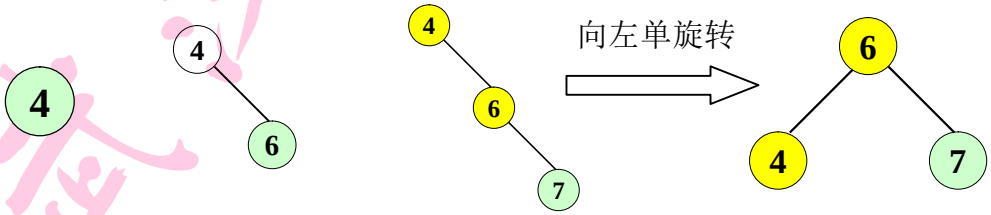
三、画图题(3 ´ 5 = 15 分)

1．中根线索二叉树

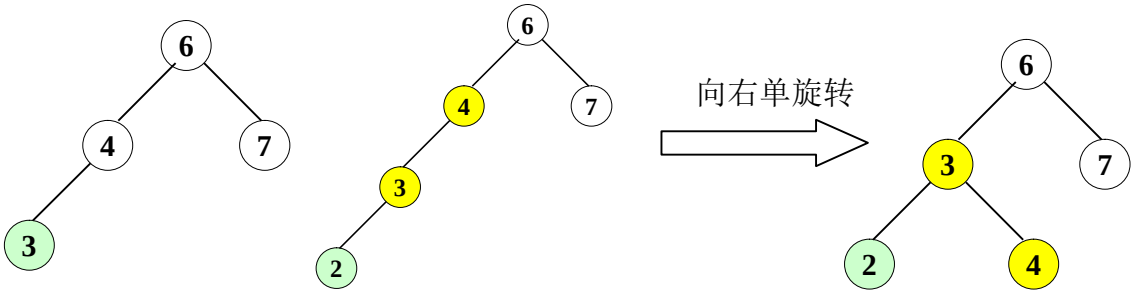


2．平衡二叉树的插入和平衡过程

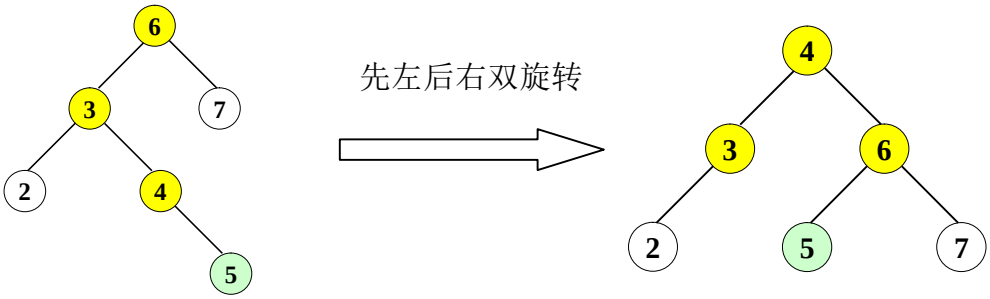
插入 4　　插入 6　　插入 7



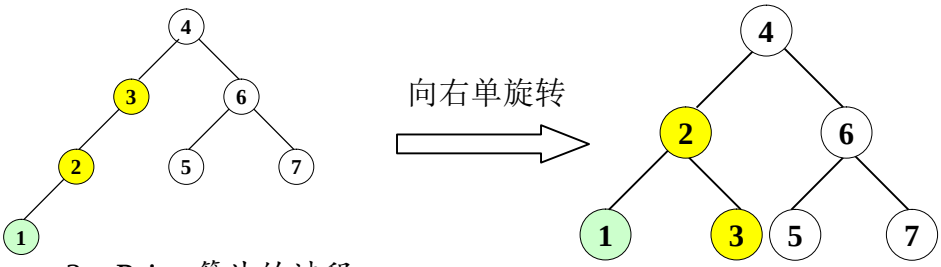
插入 3　　插入 2



插入 5

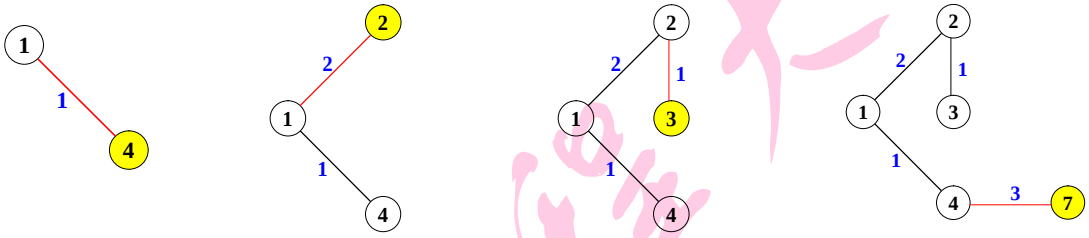


插入 1　　最终结果

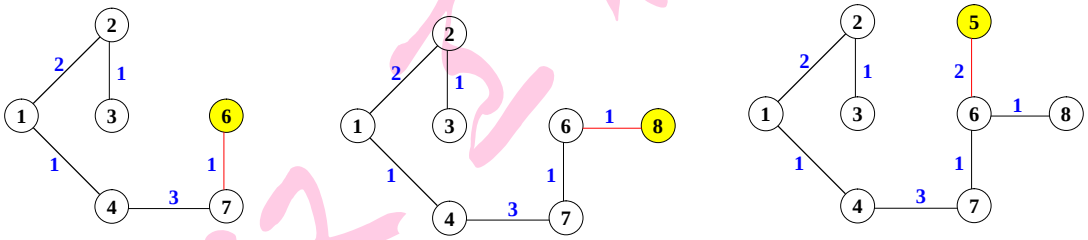


3．*Prim* 算法的过程

第 1 步　第 2 步　第 3 步　第 4 步



第 5 步　第 6 步　第 7 步



四、分析题(4´ 5 = 20 分)

1．*L* = {2, 3, 5, 6, 7, 0, 1, 8, 9, 4} 选择排序前五趟

第一趟：0, 3, 5, 6, 7, 2, 1, 8, 9, 4

第二趟：0, 1, 5, 6, 7, 2, 3, 8, 9, 4

第三趟：0, 1, 2, 6, 7, 5, 3, 8, 9, 4

第四趟：0, 1, 2, 3, 7, 5, 6, 8, 9, 4

第五趟：0, 1, 2, 3, 4, 5, 6, 8, 9, 7

2．*Dijkstra* 算法从顶点 A 出发到其余顶点的最短路径

顶点	*i* = 1	*i* = 2	*i* = 3	*i* = 4	*i* = 5	*i* = 6	*i* = 7
A	¥	¥	¥	¥	¥	¥	¥ 无
B	1 (A, B)						
C	¥	2 (A, B, C)					

D	¥	¥	6 (A, B, C, D)	5 (A, B, C, G, D)			
E	¥	¥	¥	7 (A, B, C, G, E)	6 (A, B, C, G, D, E)		
F	¥	¥	¥	9 (A, B, C, G, F)	9 (A, B, C, G, F)	7 (A, B, C, G, D, E, F)	
G	6 (A, G)	5 (A, B, G)	4 (A, B, C, G)				
v_j	B	C	G	D	E	F	
S	{A, B}	{A, B, C}	{A, B, C, G}	{A, B, C, G, D}	{A, B, C, G, D, E}	{A, B, C, G, D, E, F}	

3. 其中拓扑序列的主要约束关系

1 ⊳ 2, 4 ⊳ 7 ⊳ 5 ⊳ 8, 3 ⊳ 6 ⊳ 9

4. 最终的散列表

下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
关键字			15	2	3	5	31	18	16	4					

五、算法设计(9 + 6 = 15 分)

1. 多项式求和

算法思想：

1) 工作指针 *p*、*q* 初始化，分别指向多项式 *f* 和 *g* 的第一个结点、工作指针 *r* 指向多项式 *h* 的空链表头结点；
2) 当 *p* 不为空并且 *q* 不为空时，重复执行下列三种情形之一
   2.1 如果 *p->deg* > *q->deg*，则复制一个与当前 *p* 指向结点的副本结点接在 *r* 指向结点后成为 *h* 的新表尾、指针 *p* 后移、指针 *r* 移到新插入的表尾结点；
   2.2 如果 *p->deg* < *q->deg*，则复制一个与当前 *q* 指向结点的副本结点接在 *r* 指向结点后成为 *h* 的新表尾、指针 *q* 后移、指针 *r* 移到新插入的表尾结点；
   2.3 如果 *p->deg* == *q->deg*，则计算 *p->coef* + *q->coef*，
   如果两者之和不为 0，则在 *r* 后增加一个新结点，指数为该指数，系数为两者之和；
   指针 *p*、*q* 均后移；
3) 如果 *p* 不为空，将 *p* 所指链表所有结点依次复制并增补在 *r* 链表尾部；
4) 如果 *q* 不为空，将 *q* 所指链表所有结点依次复制并增补在 *r* 链表尾部；
5) 将多项式链表头结点的指数次数赋值给多项式结点的指数次数。

时间复杂度：

O(*m* + *n*)，*m* 和 *n* 分别为两个多项式链表的结点个数。

C 编码：

```
struct NODE
{   //   多项式结点类型声明
    double coef;            //   系数
    int deg;                //   次数
    struct NODE *next;      //   指针域
};
struct POLY
{   //   多项式类型声明
```

```
    struct NODE *head;      //  头指针
    int deg;                //  多项式次数，零多项式为-1
};
#include <stdlib.h>
void AddPoly(struct POLY *h, const struct POLY *f, const struct POLY *g)
{   //  假设 h 链表的空头结点已经初始化为空链表，结果保存在新空间
    struct NODE *p, *q, *r, *s;
    p = f->head->next;          //  p 指向 f 的第一个结点
    q = g->head->next;          //  q 指向 g 的第一个结点
    r = h->head;                //  r 指向 h 的表头结点，用于尾部插入结点
    while (p != NULL && q != NULL)
    {
        if (p->deg > q->deg)
        {   //  f 的指数大
            s = (struct NODE *)malloc(sizeof(NODE));
            s->coef = p->coef;
            s->deg = p->deg;
            s->next = r->next;      //  链表尾部插入
            r->next = s;            //  h 表尾结点指针后移
            r = s;
            p = p->next;            //  f 当前结点指针后移
        }
        else if (p->deg < q->deg)
        {   //  g 的指数大
            s = (struct NODE *)malloc(sizeof(NODE));
            s->coef = q->coef;
            s->deg = q->deg;
            s->next = r->next;
            r->next = s;
            r = s;
            q = q->next;            //  g 当前结点指针后移
        }
        else
        {   //  两者指数相同
            if (p->coef + q->coef != 0)
            {
                s = (struct NODE *)malloc(sizeof(NODE));
                s->coef = p->coef + q->coef;
                s->deg = q->deg;    //  或者 s->deg = p->deg;
                s->next = r->next;
                r->next = s;
                r = s;
            }
            p = p->next;            q = q->next;
        }
```

```
    }
    while (p != NULL)
    {   //  第一个链表没有复制完
        s = (struct NODE *)malloc(sizeof(NODE));
        s->coef = p->coef;
        s->deg = p->deg;
        s->next = r->next;          //  链表尾部插入
        r->next = s;                //  h 表尾结点指针后移
        r = s;
        p = p->next;                //  f 当前结点指针后移
    }
    while (q != NULL)
    {   //  第二个链表没有复制完
        s = (struct NODE *)malloc(sizeof(NODE));
        s->coef = q->coef;
        s->deg = q->deg;
        s->next = r->next;
        r->next = s;
        r = s;
        q = q->next;                //  g 当前结点指针后移
    }
    h->deg = h->head->next == NULL ? -1 : h->head->next->deg;   //  头结点
}
```

2．求二叉树深度

算法思想：

使用后序遍历，分别得到左右子树的高度后，再比较其大者加 1。

时间复杂度：

O(*n*)，*n* 为二叉树结点个数。

C 编码：

```
struct NODE
{   //  二叉树结点类型定义
    double data;                    //  数据域
    struct NODE *lch, *rch;         //  指针域
};
int Depth(const struct NODE *root)
{   //  空二叉树的高度为 0
    int left, right, depth = 0;
    if (root != NULL)
    {
        left = Depth(root->lch);        //  递归求左子树的深度
        right = Depth(root->rch);       //  递归求右子树的深度
        depth = 1 + (left > right ? left : right);
    }
    return depth;
}
```