勘误表

1、第 14 页上的第二段代码：

```
void delLink(LinkList *list ,LinkList q){
    LinkList r;
    if(q==list){                                /*  改为 if(q==*list)*/
        *list=q->next;
        free(q);
    }
    else{
        for(r=*list;r->next!=q;r=r->next);   /*遍历链表,找到 q 的前驱结点的指针*/
        if(r->next!=NULL){
            r->next=q->next;
            free(q);
            }
    }
}
```

中的第三行 if(q==list) 改为 if(q==*list)。

2、第 16 页的源程序中，函数 delLink 作同上相应的修改。

3、P26，入队列操作的代码第 4 行应该是：

```
    if( p == NULL) exit(0);       /*创建元素结点失败*/
```

而不应该是

```
    if( !q->front) exit(0);       /*创建头结点失败*/
```

4、第 33 页，1.62 节上面一行，“这 3 棵树为根结点 A 的子树(TubTree)”将 TubTree 改为 SubTree。

5、第 47 页，1.77 节上面倒数第 5 行，“其中用粗体字标注……”，印刷时没有印上粗体部分，步骤 2，7，12，13，14 为粗体字部分。

6、第 85 页，代码上面倒数第 3 行“循环执行上述操作，直到 w[i]<=c，表明…”改为 w[i]>c。

7、第 85 页，代码中第一个注释：/*动态开辟一个临时数组，存放 w[]的下标，如果 t[i],t[j],i<j，则 w[i]≤w[j]*/ 改为/*动态开辟一个临时数组，存放 w[]的下标，如果 t[i],t[j],i<j，则 w[t[i]]≤w[t[j]]*/

8、第 86 页，最上面的一段话中第 3 行：“即 w[1]的质量小于集装箱 w[t[0]]即 w[0]的质量”，改为“即 w[1]的质量小于集装箱 w[t[2]]，即 w[0]的质量”。

9、第 119 页，寻找矩阵鞍点的算法描述，改为下面的描述更为清楚。

按行寻找矩阵鞍点的算法描述：

```
对于一个 m*n 的矩阵
i←0;
Repeat:
    找出第 i 行中最大的元素 A[i][t];
    If(本行中有与元素 A[i][t]的值相等的元素)
            Then  本行中没有鞍点，执行 i ← i+1，并跳出本次 Repeat 循环;
    Else  将元素 A[i][t]与第 t 列中的每个元素逐一进行比较，
        If(存在小于或等于 A[i][t]的元素)
            Then 说明 A[i][t]不是该矩阵的鞍点，执行 i ← i+1，跳出本次 Repeat 循环;
    Else   A[i][t]是鞍点，返回该元素在矩阵中的位置（i,t），程序结束。
until   i>=m
返回 0，该矩阵中无鞍点。
```

10、第 161 页，寻找 3000 以内亲密数的算法描述，改为

```
将 1，2，…，3000 各元素存放在 x[1…3000]中;
for(i=1;i<=3000;i++)
    if(x[i]没有找到其亲密数，即 x[i]仍在集合 B 中){
        for(j=i+1;j<=3000;j++)
        if(x[j]为 x[i]的亲密数)
            输出亲密数(x[i],x[j])，并记录 x[j]已经找到其亲密数;
        }
```

也就是将书中算法第 4 行的 n 改为 3000。下面的那个算法描述也是同样改法。

11、第 163 页，中间算法描述改为：

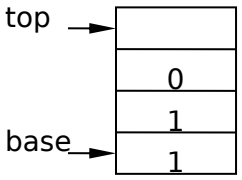
```
输入一个数（1~999999 的整数）N;
过程 A：翻译千位数
    a=N/1000;
```

```
    if(a != 0) 则 a 一定在 1~999 之间，调用子过程 B 按规则翻译 a，并打印 thousand;
    a=N%1000;
    if(a != 0) 则 a 一定在 1~999 之间，调用子过程 B 按规则翻译 a。
过程 B：翻译百位数
    b=a/100;
    if(b != 0) 调用子过程 C 按规则翻译 b，并打印 hundred;
    b=a%100;
    f(b != 0) 调用子过程 C 按规则翻译 b。
    else 程序结束
过程 C：翻译十位数和个位数
    if(b<=19) 直接翻译 c;
    else c=b/10，按规则翻译十位数 c，再按规则翻译个位数 b%10。
```

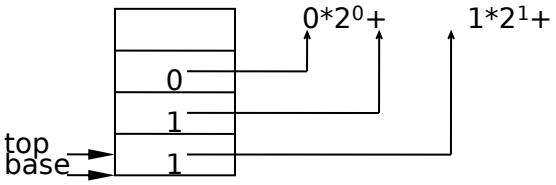
也就是将第 4 行的 10~999 改为 1~999。

12、第 165 页，图 5-33 印刷不清楚，见下图。

（1）将二进制数 110 从高位到低位依次入栈



（2）再从栈顶取数分别与 $2^0,2^1\cdots2^n$ 相乘，并累加求和



13、第 241 页，本页的算法中变量 flag 写的有问题，应该改为如下：

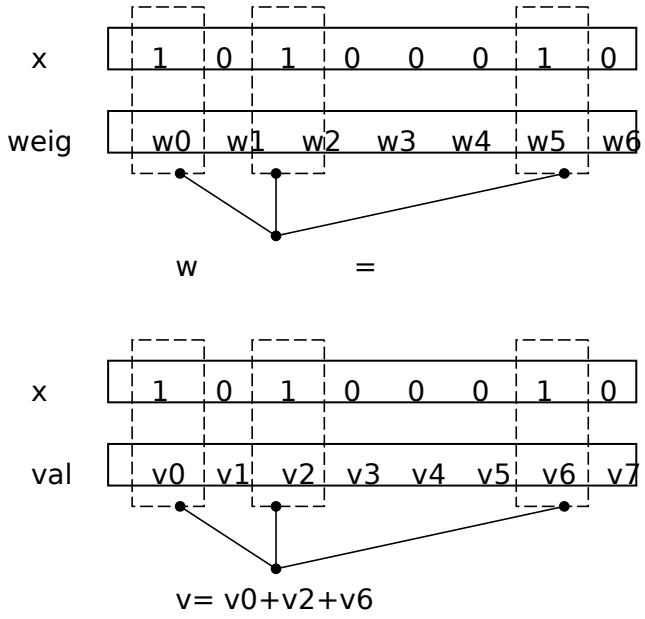
```
int JusticCompleteBiTree(BiTree T,int level ,int n,int *flag){
    if(!T){
        return 1;
        }
    if(level == n)
    {
       if(T->lchild == NULL && T->rchild != NULL) return 0;

       if(*flag == 0){/*同层的前面的结点无空指针*/
             if(T->rchild == NULL) * flag = 1; /*出现空指针*/
             }
             else if(*flag == 1){   /*同层的前面的结点有空指针*/
                  if(T->lchild!=NULL || T->rchild!=NULL)   return 0;
             }
```

```
        }
         if(level != n &&level !=n+1)
         {
            if(T->lchild == NULL || T->rchild == NULL) return 0;
         }
        if(!JusticCompleteBiTree(T->lchild,level+1,n,flag)) return 0;
        if(!JusticCompleteBiTree(T->rchild,level+1,n,flag)) return 0;
       return 1;
    }
```

也就是有几个 flag 前面要加上星号“*”，因为参数 flag 是纪录第 k-1 层结点右孩子是否为 NULL 的标志，因此要在递归中保留该值。后面的程序 7-10 中，函数 JusticCompleteBiTree 作相应的修改。

14、书中第 301 页的图，印刷不清楚，应为



15、第 303 页，图 9-16 和图 9-17 印刷不清楚，应为

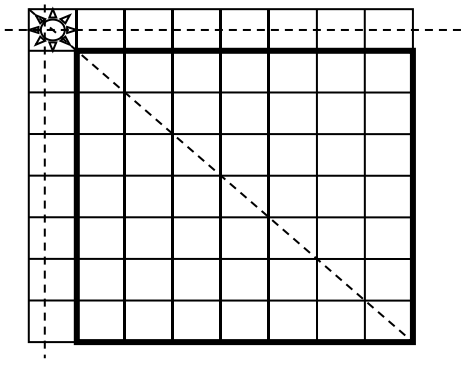


图 9-16 八皇后问题的递归结构示意（1）

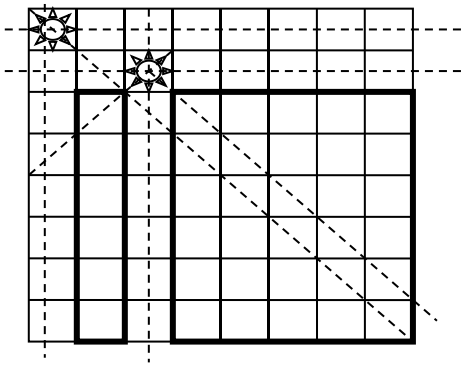


图 9-17 八皇后问题的递归结构示意（2）

16、第 337 页，中间算法描述，函数 getMax 中，代码第 9 行改为

if(str[i-1] == '1'&& i!=0)　/*如果是字符串的 1-0 转换点*/

第 19 行改为

if(str[i-1] == '0'&& i!=0)　/*如果是字符串的 0-1 转换点*/

即多加一个条件 i!=0，这样程序更加严谨。

17、第 363 页，例 10-44 的分析中，第二行“下面给出每一步操作(Push,Pop)后堆栈、入栈队列和出栈队列的状态，如图 10-40 所示。”改为如图 10-43 所示。

18、P375 图 10-56 错了，与图 10-53 印成一样的了，应该是

```
ABC  D  E F
C is at level 3 in the BITree
```

图 10-56 程序 10-34 的运行结果