1. **页码：65 • 行数：17 • 印次 1-9**

改动红色标记部分,“译码”改为“编码”：

结构体的编码过程中，bit[]存放结点的编码，start记录编码开始下标，逆向**编码**(从叶

1. **页码：127 • 行数： 3 • 印次 6-9**

图3-54替换为：



1. **页码：152 • 行数：5 • 印次 1-9**

改动红色部分：

*b*[4][2]=1，向左上角找*b*[3][1]，返回时输出*s*1[3]=“A”;

1. **页码：241• 行数：9 • 印次1-9**

改动红色标记部分：其它不变

//如果右子树是叶子

if(k>=j)

{

cout<<"e"<<k<<" is the right child of "<<"S"<<k<<endl;

}

1. **页码：242 • 行数：30 • 印次 1-9**

改动红色标记部分：其它不变

//如果右子树是叶子

if(k>=j)

{

cout<<"e"<<k<<" is the right child of "<<"S"<<k<<endl;

}

1. **页码：262 • 行数：-3 • 印次 1-9**

改动红色标记部分, *p*改为*x*：

**空间复杂度：**

回溯法的另一个重要特性就是在搜索执行的同时产生解空间。在所搜过程中的任何时刻，仅保留从开始结点到当前扩展结点的路径，从开始结点起最长的路径为*n*。程序中我们使用*bestx*[]数组记录该最长路径作为最优解，所以该算法的空间复杂度为*O*(*n*)。

1. **页码：277 • 行数：-5 • 印次 1-9**

*p*改为*x*：

**空间复杂度：**

回溯法的另一个重要特性就是在搜索执行的同时产生解空间。在所搜过程中的任何时刻，仅保留从开始结点到当前扩展结点的路径，从开始结点起最长的路径为*n*。程序中我们使用*bestx*[]数组记录该最长路径作为最优解，所以该算法的空间复杂度为*O*(*n*)。

1. **页码：347 • 行数： • 印次 1-9**

图6-16，6-17，6-18，6-19，6-20，6-21 中G结点数据8改为10：

注意：共6个图都要修改G结点



1. **页码：418,419 • 行数：-1,-9至-12 • 印次 1-9**

删除红色标记部分：其它不变

for(i=1;i<=n;i++) //找离基行(常数列/入基列正比值最小对应的行)

{

~~if(max2<kernel[i][e])~~

~~{~~

~~max2=kernel[i][e];~~

~~}~~

float temp=kernel[i][0]/kernel[i][e]; ~~//常数项在前,temp=fabs(temp);~~

if(temp>0&&temp<min) //找离基变量

{

min=temp;

k=i;

}

}

**i**~~f(max2==0)~~

~~{~~

~~cout<<"解无界"<<endl;~~

~~break;~~

~~}~~

1. **页码：421• 行数：-8 • 印次 1-9**

改动红色标记部分：其它不变

FJL[e]=JL[k];

JL[k]=temp;

for(i=0;i<=n;i++) //计算除入基列和**离**基行的所有位置的元素

1. **页码：421• 行数：-21 • 印次 1-9**

改动红色标记部分：其它不变

float temp=kernel[i][0]/kernel[i][e]; ~~//常数项在前,temp=fabs(temp);~~

if(temp>0&&temp<min) //找离基变量

1. **页码：466 • 行数：18 • 印次 1-9**

改动红色标记部分,“9”改为“10”：

请输入结点个数n和边数m：

6 10

1. **页码：69 • 行数：7，8，9 • 印次 1-9**

改动红色标记部分,“8”改为“9”：

x1=7 x2=1；//x1、x2为两个最小权值结点的序号

m1=25 m2=32；//m1、m2为两个最小权值结点的权值

HuffNode[7].parent = 9; //x1的父亲为新结点编号n+i

HuffNode[1].parent = 9; //x2的父亲为新结点编号n+i

HuffNode[9].weight =57; //新结点权值为两个最小权值之和m1+m2

HuffNode[9].lchild = 7; //新结点n+i的左孩子为x1

HuffNode[9].rchild = 1; //新结点n+i的右孩子为x2

1. **页码：3 • 行数：9 • 印次 1-9**

改动红色标记部分：

for(int i=1;i<=n;i++)

{

sum=sum+pow(-1,i); //pow(-1,i)表示-1的i次幂

}

1. **页码：106 • 行数：-4,-6 • 印次 1-9**

删除s后面的中括号，红色标记部分：

int recursionBS (int s[],int x,int low,int high)

{

    //low指向数组的第一个元素，high指向数组的最后一个元素

    if(low>high)              //递归结束条件

        return -1;

    int middle=(low+high)/2;  //计算middle值(查找范围的中间值)

    if(x==s[middle])          //*x*等于s[middle]，查找成功，算法结束

        return middle;

    else if(x<s[middle])      //*x*小于s[middle]，则从前半部分查找

             return recursionBS (s**~~[]~~**,x, low, middle-1)

           else               //*x*大于s[middle]，则从后半部分查找

             return recursionBS (s**~~[]~~**,x, middle+1, high)

}

1. **页码：214 • 行数：4 • 印次 1-9**

1改为2，红色标记部分：

• *j*=10时，***c***[**2**][10]=max{***c***[1][10]，***c***[1][5]+3}=9。