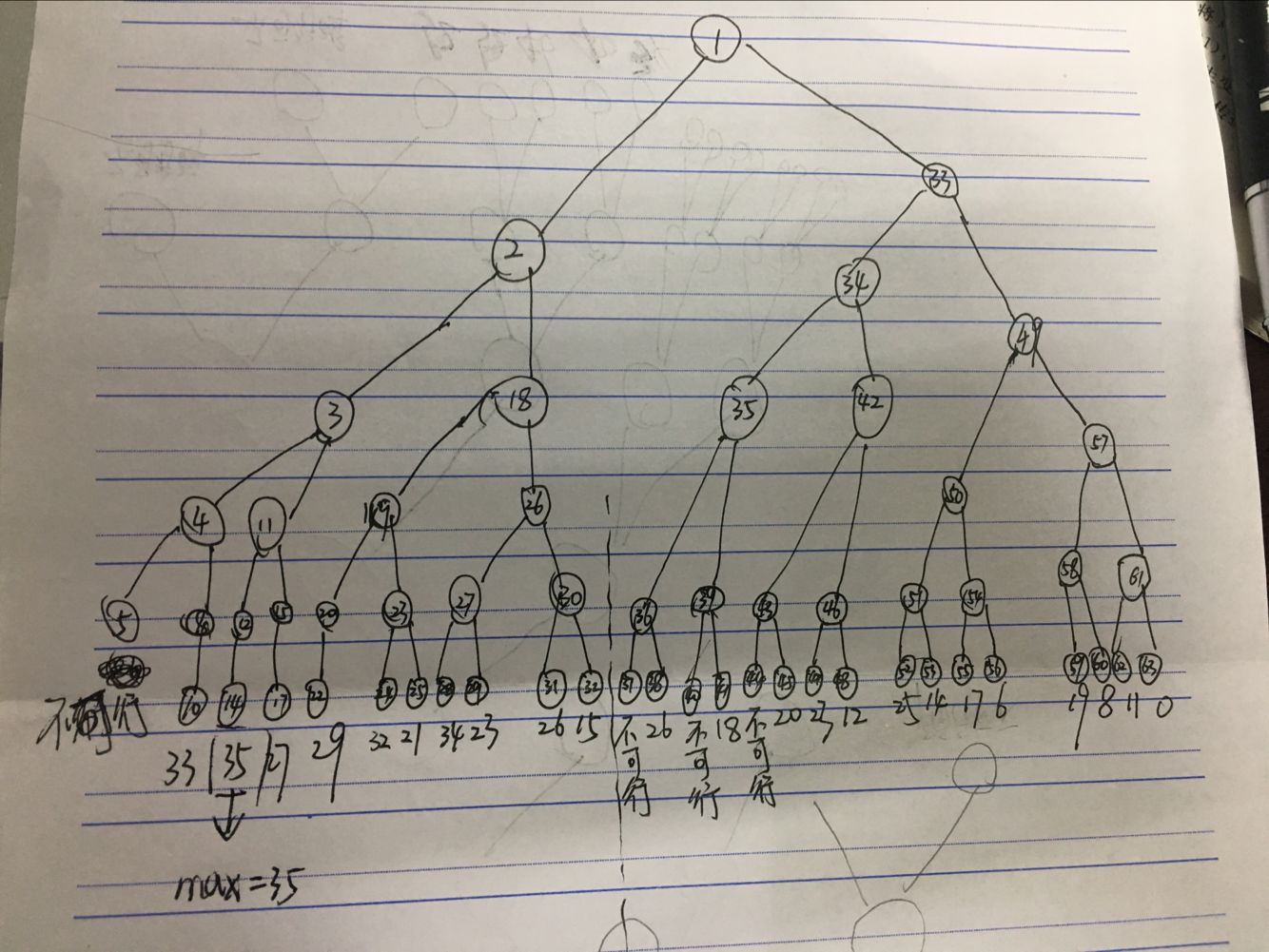
**第八章**

**8.2**

**按照要求，去掉重为10，价值15的物品，使用回溯法求解背包问题搜索空间如下图：**

****

**8.4**

**问题描述：回溯法解决n皇后问题**

回溯法解题的一般步骤：

1、针对所给问题，确定问题的解空间

2、利用适于搜索的方法组织解空间

3、利用深度优先搜索解空间

4、在搜索过程中用剪枝函数避免无效搜索

则本题：

1. 首先找出解空间：给棋盘的行和列都编上1到N的号码，皇后也给编上1到N的号码。由于一个皇后应在不同的行上，可以假定第i个皇后将放在第i行上的某列。因此N皇后问题的解空间可以用一个N元组（X1，X2，.....Xn）来表示，其中Xi是放置皇后i所在的列号。这意味着所有的解都是N元组（1，2，3，.......，N）的置换。解空间大小为N！。
2. 约束条件：只用判断不在同一列和不在同一斜线的约束。因为Xi表示皇后所在的列号，所以第k个皇后和第i个皇后同列的判断条件是X（k）=X（i）。所以不同列的判段条件是X（k）！=X（i），1<k<i 。又因为同一斜线的特征是要么行号和列号之和不变（右高左低）要么是行号和列号只差相等（左高右低），所以第k个皇后和第i个皇后在同斜线的判断条件是 i+X（i）=  k+X（k） 或 i-X(i) =k-X(k),两式合并得 |X(i)-X(k)|=|i-k|

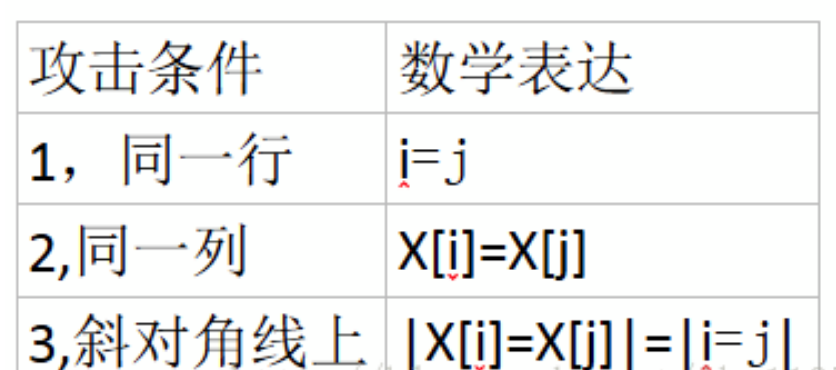


图 1 约束条件描述

具体实现上可以采用递归和非递归两种:

非递归：

初始化a[n],i;

i = 1;

while (i > 0(有路可走) and (未达到目标)){

if (i > n){

搜索到一个解，输出；

}

else{

a[i]第一个可能的值；

while (a[i]在不满足约束条件且在搜索空间内)

{

a[i]下一个可能的值；

}

if (a[i]在搜索空间内){

i = i + 1; //到下一个节点

}

else{

//回溯

i = i - 1;

}

}

}

递归:  
t表示递归深度，n用来控制递归深度。当t>n时，算法以搜索至叶结点,output(x)输出,for循环中的f(n,t)和g(n,t)分别表示在当前扩展结点处未搜索过的子树的起始编号和终止编号,h(i)表示在当前扩展结点处x[t]的第i个可选值,constraint(t)和bound(t)是当前扩展结点处的约束函数和限界函数。调用一次backtrack(1)即可完成整个回溯搜索过程。

void backtrack(int t){  
if(t>n)  
output(x);  
else{  
for(i=f(n,t); i<=g(n,t); i++){  
x[t]=h(i);  
if(constraint(t)&&bound(t))  
backtrack(t+1);  
 }  
 }  
}

完整代码见附录

**8.5（编程题）**

**问题描述：给定一个正整数集合X={x1,x2,…, xn}和一个正整数y，设计回溯算法，求集合X的一个子集Y，使得Y中元素之和等于y**

对于给定集合x[N]={2,1,3,4,2}和正整数12为输入例子

1．x={x1,x2,x3……xn }，sum=0，y={ }为解向量，初始化为全0;

2．k=0;

3．while (k>=0)

3.1 y[k]=y[k]-1;

3.2 如果((y[k]==1||y[k]==0)&&k<N)

3.2.1 sum=sum+(y[k]?x[k]:0);

3.2.2 如果(sum==y){break;},找到解，到步骤4

3.2.3 否则

3.2.3.1 如果(sum<n){k++;},搜索下一个

3.2.3.2 否则 sum=sum-(y[k]?x[k]:0);

3.3 否则 回溯

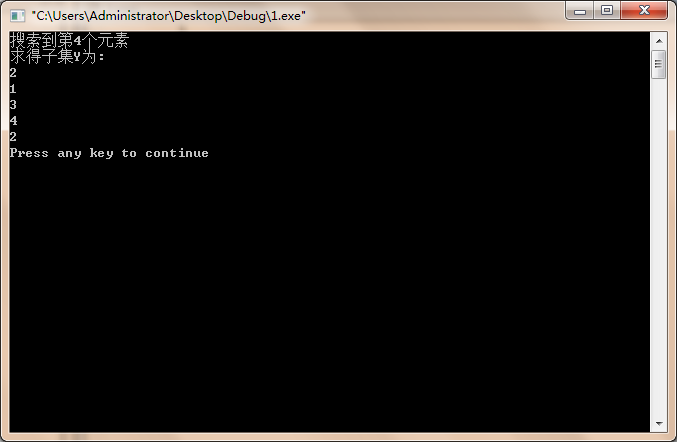
3.3.1 sum=sum-(y[k]?x[k]:0);

y[k]=2;

k--;

sum=sum-(y[k]?x[k]:0);

4.输出k

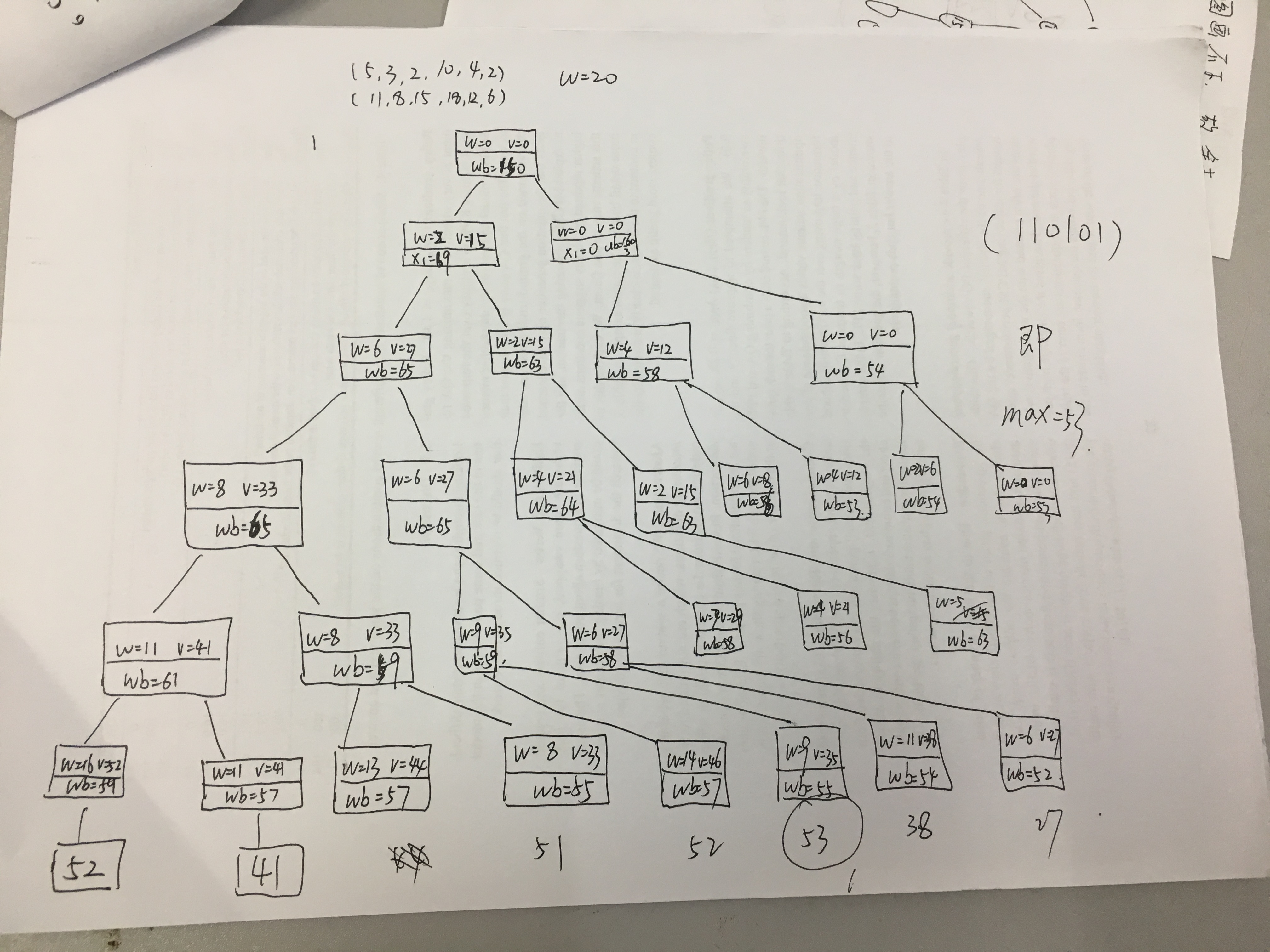
**输出如下**

**思路分析：解分量的和小于y为剪枝函数。当搜索到结点，并且解分量的和等于y时，找到问题的解。**

**完整代码见附录**

**第九章 9.3**

**按照要求，使用分支限界法求解背包问题搜索空间如下图**



附：

8.4代码

#include<stdio.h>

#include<math.h>

using namespace std;

#define N 60

int sum = 0;

int x[N];

bool isPlaceAble(int queenAtRow)  {

    int i;

    for(i=1;i<queenAtRow;i++)

    {

        if(abs(queenAtRow-i)==abs(x[queenAtRow]-x[i]) || x[queenAtRow] == x[i])

        {

            return false;

        }

    }

    return true;

}

int queen(int queenAtRow,int n)  {

    if( queenAtRow>n && n>0)

    {

          sum++;

    }

    else

    {

      for(int i=1;i<=n;i++)  {

          x[queenAtRow] = i;

          if(isPlaceAble(queenAtRow))

           {

               queen(queenAtRow+1, n);

          }

      }

    }

    return sum;

}

int NQueen(int n)  {

    if( n==0 )

    {

        return 0;

    }

    int methodNum = queen(1, n);

return methodNum;

}

int main()  {

    int  sum;

    sum = NQueen(5);

    cout<<sum<<endl;

    return 0;

}

8.5代码

#include <iostream.h>

const int N=5;

int f(int x[],int y[],int n) {

for(int i=0;i<N;i++)

y[i]=2;

int k=0;

int sum=0;

while(k>=0) {

y[k]=y[k]-1;

if((y[k]==1||y[k]==0)&&k<N){

sum=sum+(y[k]?x[k]:0);

if(sum==n){break;}

else{

if(sum<n){k++;}

else{

sum=sum-(y[k]?x[k]:0);

}

}

}

else{

y[k]=2;

k--;

sum=sum-(y[k]?x[k]:0);

}

}

return k;

}

void main() {

int x[N]={2,1,3,4,2};

int y[N];

int n=12;

int k=f(x,y,n);

cout<<"搜索到第"<<k<<"个元素"<<endl;

for(int i=0;i<N;i++)

cout<<(y[i]==1?x[i]:0)<<endl;

}