

1. 整数划分
2. 棋盘覆盖
3. 合并排序
4. 集合划分
5. 整数(素数)因子分解
6. 最长公共子序列
7. 0-1 背包问题
8. 算法实现题 3-5 编辑距离问题
- 9 . Dijkstra 算法
10. 编程实现采用破圈法求加权图的最小生成树。
- 11 推箱子问题
12. 符号三角形
13. N 皇后

## 1. 整数划分 -p13

<https://blog.csdn.net/XZ2585458279/article/details/89444888>

整数划分，是指把一个正整数  $n$  表示成系列正整数之和：

例如正整数 6 有如下 11 种不同的划分，所有  $p(6)=11$

1	6
2	5+1
3	4+2, 4+1+1
4	3+3, 3+2+1, 3+1+1+1
5	2+2+2, 2+2+1+1, 2+1+1+1+1
6	1+1+1+1+1+1

```
1. #include<iostream>
2. using namespace std;
3.
4. int split(int n,int m)
5. {
6.     if(n == 1 || m == 1) return 1;
7.     else if(n < m) return split(n,n);
8.     else if(n == m) return split(n,n-1) + 1;
9.     else return split(n,m-1)+split(n-m,m);
10. }
11.
12. int main()
13. {
14.     int n;
15.     printf("输入划分数:");
16.     scanf("%d",&n);
17.     printf("整数划分为:%d\n",split(n,n));
18.     return 0;
19. }
```

## 2. 棋盘覆盖 -p21

[https://blog.csdn.net/acm\\_JL/article/details/50938164](https://blog.csdn.net/acm_JL/article/details/50938164)

在棋盘覆盖问题中，要用图示的 4 种不同形态的 L 型骨牌覆盖一个给定的特殊棋盘上除特殊方格以外的所有方格，且任何 2 个 L 型骨牌不得重叠覆盖。易知，在任何一个  $2^k \times 2^k$  的棋盘覆盖中，用到的 L 型骨牌个数恰为  $(4^k - 1) / 3$ 。

课堂作业：把每次放 L 型骨牌时的 3 个坐标输出

```
1. #include <iostream>
2.
3. int tile = 1; // 骨牌序号
4. int board[128][128]; // 二维数组模拟棋盘
5.
6. // (tr,tc)表示棋盘的左上角坐标(即确定棋盘位置), (dr,dc)表示特殊方块的位置, size=2^k 确定棋盘大小
7. void chessBoard(int tr, int tc, int dr, int dc, int size)
8. {
```

```

9.      // 递归出口
10.     if (size == 1)
11.         return;
12.     int s = size / 2; //分割棋盘
13.     int t = tile++;    //t 记录本层骨牌序号
14.     // 判断特殊方格在不在左上棋盘
15.     if (dr < tr + s && dc < tc + s)
16.     {
17.         chessBoard(tr, tc, dr, dc, s); //特殊方格在左上棋盘的递归处理方法
18.     }
19.     else
20.     {
21.         board[tr + s - 1][tc + s - 1] = t;          // 用 t 号的 L 型骨牌覆盖右下角
22.         chessBoard(tr, tc, tr + s - 1, tc + s - 1, s); // 递归覆盖其余方格
23.     }
24.     // 判断特殊方格在不在右上棋盘
25.     if (dr < tr + s && dc >= tc + s)
26.     {
27.         chessBoard(tr, tc + s, dr, dc, s);
28.     }
29.     else
30.     {
31.         board[tr + s - 1][tc + s] = t;
32.         chessBoard(tr, tc + s, tr + s - 1, tc + s, s);
33.     }
34.     // 判断特殊方格在不在左下棋盘
35.     if (dr >= tr + s && dc < tc + s)
36.     {
37.         chessBoard(tr + s, tc, dr, dc, s);
38.     }
39.     else
40.     {
41.         board[tr + s][tc + s - 1] = t;
42.         chessBoard(tr + s, tc, tr + s, tc + s - 1, s);
43.     }
44.
45.     // 判断特殊方格在不在右下棋盘
46.     if (dr >= tr + s && dc >= tc + s)
47.     {
48.         chessBoard(tr + s, tc + s, dr, dc, s);
49.     }
50.     else
51.     {
52.         board[tr + s][tc + s] = t;
53.         chessBoard(tr + s, tc + s, tr + s, tc + s, s);
54.     }
55. }
56.
57. int main()
58. {
59.     int boardSize = 8;          // 棋盘边长
60.     chessBoard(0, 0, 3, 3, boardSize); // (0, 0)为顶点, 大小为 boardSize 的棋盘; 特殊方块位于
    (3, 3)
61.     // 打印棋盘
62.     int i, j;
63.     printf("\n\n\n");
64.     for (i = 0; i < boardSize; i++)
65.     {
66.         for (j = 0; j < boardSize; j++)
67.         {
68.             printf("%d\t", board[i][j]);
69.         }
70.         printf("\n\n\n");
71.     }
72.     return 0;
73. }

```

### 3. 合并排序

给定有序表 A[1:n]，修改合并排序算法，求出该有序表的逆序对数。

P23 页在 void Merge()函数中进行修改。首先定义一个全局变量 count 用来记录逆序对的个数。然后在 else d[k++]=c[j++]中间加入 count += m-i+1;

```
1. package hello;
2. import org.omg.CORBA.PUBLIC_MEMBER;
3.
4. public class M2 {
5.     public static int n=0;
6.     public static void merge(int a[],int first,int mid,int last)
7.     {
8.         int temp[] = new int[last-first+1]; //临时数组，用于临时存放比较后的数字
9.         int i=first,j=mid+1,k=0;
10.
11.         while(i<=mid&&j<=last) //遍历比较左右两个部分
12.         {
13.             if(a[i]<=a[j])
14.                 temp[k++] = a[i++]; //左半部分元素小于右半部分的元素，将左边该元素存入临时数组
15.
16.             else
17.             {
18.                 temp[k++] = a[j++];
19.                 n=n+(mid-i+1); //统计左半边能和右半边该元素构成的逆序对数
20.             }
21.         }
22.         while(i<=mid)
23.             temp[k++] = a[i++];
24.         while(j<=last)
25.             temp[k++] = a[j++];
26.         for(i=0; i < k; i++)
27.             a[first + i] = temp[i]; //从临时数组取出放回原数组
28.     }
29.
30.     public static void mergesort(int a[],int first,int last)
31.     {
32.         if(first < last)
33.         {
34.             int mid = (first+last)/2;
35.             mergesort(a,first,mid); //递归排序左半部分
36.             mergesort(a,mid+1,last); //递归排序右半部分
37.             merge(a,first,mid,last); //将处理后的两个部分合并
38.         }
39.     }
40.
41.     public static void main(String[] args) {
42.         // TODO Auto-generated method stub
43.         int a[] = {4,3,2,1};
44.         System.out.println("序列:");
45.         for(i=0;i<4;i++)
46.             System.out.println(a[i]);
47.         mergesort(a,0,3);
48.         System.out.println("逆序对数: ");
49.         System.out.println(n-1);
50.     }
51. }
```

### 4. 集合划分

P45 算法实现题 2-10 集合划分问题

```
1. #include <stdio.h>
```

```

2. #include <stdlib.h>
3. int jihe(int n,int m)
4. {
5.     if(n==m||m==1)
6.         return 1;
7.     else
8.     {
9.         return jihe(n-1,m-1)+m*jihe(n-1,m);
10.    }
11.
12. }
13. int main()
14. {
15.     int n,m;
16.     scanf("%d%d",&n,&m);
17.     int t;
18.     t=jihe(n,m);
19.     printf("%d",t);
20.
21.     return 0;
22. }

```

## 5.整数(素数)因子分解

大于 1 的正整数  $n$  可以分解为  $n=x_1*x_2*.....*x_m$ ，例如，当  $n=12$  时，共有下面 3 种不同的分解式：

$$12=3*2*2$$

$$12=2*3*2$$

$$12=2*2*3$$

```

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <math.h>
4.
5. bool is_prime(int n) {
6.     int flag = 0;
7.     int i;
8.     for (i = 2; i <= sqrt(n); i++) {
9.         if (n % i == 0) {
10.            flag = 1;
11.            break;
12.        }
13.    }
14.    if (flag == 1) {
15.        return false;
16.    } else {
17.        return true;
18.    }
19.
20. }
21.
22. int solve(int n) {
23.     int i;
24.     int count = 0;
25.     if (n == 1 || is_prime(n)) {
26.         return 1;
27.     } else {
28.         for (i = 2; i <= n; i++) {
29.             if (n % i == 0 & is_prime(i)) {
30.                 count += solve(n / i);
31.             }
32.         }

```

```

33. }
34. return count;
35. }
36.
37.
38. int main() {
39. int n = 12;
40. int c;
41. c = solve(100);
42. printf("%d", c);
43. }

```

## 6.最长公共子序列 -p54

X=ABCADCBA

Y=CBACBD

求 X 与 Y 的最长公共子序列

```

1. void fun(string x,string y)
2. {
3.     int m = x.size();
4.     int n = y.size();
5.     vector<vector<int>> c(m+1, vector<int>(n+1, 0));
6.     vector<vector<int>> b(m+1, vector<int>(n+1, 0));
7.     for (int i = 1; i <= m;i++)
8.     {
9.         for (int j = 1; j <= n;j++)
10.        {
11.            if(x[i-1]==y[j-1])
12.            {
13.                c[i][j] = c[i - 1][j - 1] + 1;
14.                b[i][j] = 1;
15.            }
16.            else if(c[i-1][j]>=c[i][j-1])
17.            {
18.                c[i][j] = c[i - 1][j];
19.                b[i][j] = 2;
20.            }
21.            else
22.            {
23.                c[i][j] = c[i][j - 1];
24.                b[i][j] = 3;
25.            }
26.        }
27.    }
28. }
29. int main()
30. {

```

```

31.     string x="ABCADCBA",y="CBACBD";
32.     fun(x, y);
33.     return 0;
34. }

```

```

0000000
0001111
0011122
0111222
0112222
0112223
0112333
0122344
0123344
$$$$$$$
0000000
0221333
0212213
0122122
0221222
0222221
0122132
0212213
0221222
ABCADCBA

```

X= ABCADCBA Y= CBACBD

b

	C	B	A	C	B	D		C	B	A	C	B	D
	0	0	0	0	0	0		0	0	0	0	0	0
A	0	0	0	1	1	1		0	2	2	1	3	3
B	0	0	1	1	1	2		0	2	1	2	2	1
C	0	1	1	1	2	2		0	1	2	2	1	2
A	0	1	1	2	2	2		0	2	2	1	2	2
D	0	1	1	2	2	3		0	2	2	2	2	1
C	0	1	1	2	3	3		0	1	2	2	1	3
B	0	1	2	2	3	4		0	2	1	2	2	1
A	0	1	2	3	3	4		0	2	2	1	2	2

B A C B

结果：BACB

Ps: 上面的矩阵是 c, 下面的是 b

### 7.0-1 背包 -p74

W={4,3,6,8,2,10,3}

V={8,3,6,2,10,3,6}

C=16

```
1. #include <iostream>
2. #include <cstring>
3. using namespace std;
4.
5. const int N=150;
6.
7. int v[N]={0,12,8,9,5};
8. int w[N]={0,15,10,12,8};
9. int x[N];
10. int m[N][N];
11. int c=30;
12. int n=4;
13. void traceback()
14. {
15.     for(int i=n;i>1;i--)
16.     {
17.         if(m[i][c]==m[i-1][c])
18.             x[i]=0;
19.         else
20.         {
21.             x[i]=1;
22.             c-=w[i];
23.         }
24.     }
25.     x[1]=(m[1][c]>0)?1:0;
26. }
27.
28. int main()
29. {
30.
31.
32.     memset(m,0,sizeof(m));
33.     for(int i=1;i<=n;i++)
34.     {
35.         for(int j=1;j<=c;j++)
36.         {
37.             if(j>=w[i])
38.                 m[i][j]=max(m[i-1][j],m[i-1][j-w[i]]+v[i]);
39.
40.             else
41.                 m[i][j]=m[i-1][j];
42.         }
43.     }/*
44.     for(int i=1;i<=6;i++)
45.     {
46.         for(int j=1;j<=c;j++)
47.         {
48.             cout<<m[i][j]<<' ';
49.         }
50.         cout<<endl;
51.     }
52. */
53.     traceback();
```



```

54.     for(int i=1;i<=n;i++)
55.         cout<<x[i];
56.     return 0;
57. }

```

$w_i = (4, 3, 6, 8, 2, 10, 3)$   
 $v_i = (18, 3, 6, 2, 10, 3, 6)$   
 $C = 16$

$m(i, j)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	10	10	10	16	18	18	19	24	24	24	27	27	27	30	30
2	0	0	10	10	10	16	16	16	19	19	19	22	22	22	25	25	25
3	0	0	10	10	10	16	16	16	16	16	16	22	22	22	18	18	19
4	0	0	10	10	10	16	16	16	16	16	16	16	16	16	16	19	19
5	0	0	10	10	10	16	16	16	16	16	16	16	16	16	9	9	9
6	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	0	0	0	6	6	6	6	6	6	6	6	6	6	6	6	6	6

$max = 30$   
 $x = (1, 0, 1, 0, 1, 0, 1)$

### 8.算法实现题 3-5 编辑距离问题

★问题描述:设 A 和 B 是 2 个字符串。要用最少的字符操作将字符串 A 转换为字符串 B。这里所说的字符操作包括:

- (1)删除一个字符;
- (2)插入一个字符;
- (3)将一个字符改为另一个字符。

将字符串 A 变换为字符串 B 所用的最少字符操作数称为字符串 A 到 B 的编辑距离, 记为  $d(A, B)$ 。试设计一个有效算法,对任给的 2 个字符串 A 和 B,计算出它们的编辑距离  $d(A, B)$ 。

★算法设计:对于给定的字符串 A 和字符串 B,计算其编辑距离  $d(A, B)$ 。

<https://zhuanlan.zhihu.com/p/80682302>

```

1. int minDistance(String s1, String s2) {
2.     int m = s1.length(), n = s2.length();
3.     int[][] dp = new int[m + 1][n + 1];
4.     // base case
5.     for (int i = 1; i <= m; i++)
6.         dp[i][0] = i;
7.     for (int j = 1; j <= n; j++)
8.         dp[0][j] = j;

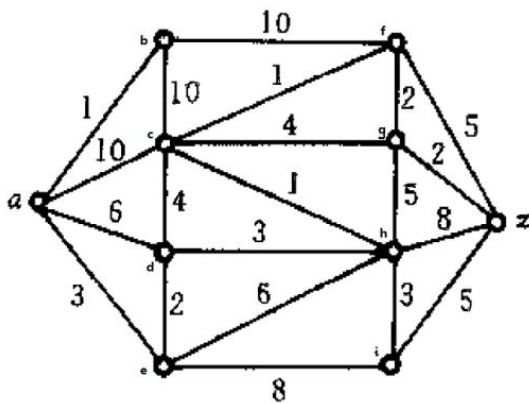
```

```

9. // 自底向上求解
10. for (int i = 1; i <= m; i++)
11.     for (int j = 1; j <= n; j++)
12.         if (s1.charAt(i-1) == s2.charAt(j-1))
13.             dp[i][j] = dp[i - 1][j - 1];
14.         else
15.             dp[i][j] = min(
16.                 dp[i - 1][j] + 1,
17.                 dp[i][j - 1] + 1,
18.                 dp[i-1][j-1] + 1
19.             );
20. // 储存着整个 s1 和 s2 的最小编辑距离
21. return dp[m][n];
22. }
23.
24. int min(int a, int b, int c) {
25.     return Math.min(a, Math.min(b, c));
26. }

```

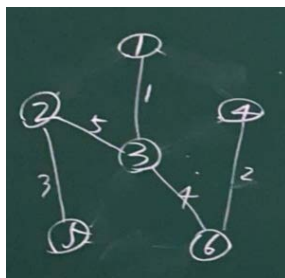
9. 根据所示的无向加权图，请应用 Dijkstra 算法求出从 a 到 z 的最短路径及其长度。要求列表显示每一步的迭代计算过程。



迭代	S	u	dist[b]	dist[c]	dist[d]	dist[e]	dist[f]	dist[h]	dist[g]	dist[i]	dist[z]
初始	{a}	-	1	10	6	3	maxint	maxint	maxint	maxint	maxint
1	{a,b}	b	1	10	6	3	11	maxint	maxint	maxint	maxint
2	{a,b,e}	e	1	10	5	3	11	9	maxint	11	maxint
3	{a,b,e,d}	d	1	9	5	3	11	8	maxint	11	maxint
4	{a,b,e,d,h}	h	1	9	5	3	11	8	12	11	16
5	{a,b,e,d,h,c}	c	1	9	5	3	10	8	12	11	16
6	{a,b,e,d,h,c,f}	f	1	9	5	3	10	8	12	11	16
7	{a,b,e,d,h,c,f,i}	i	1	9	5	3	10	8	12	11	16
8	{a,b,e,d,h,c,f,i,g}	g	1	9	5	3	10	8	12	11	14
9	{a,b,e,d,h,c,f,i,z}	z	1	9	5	3	10	8	12	11	14

## 10.编程实现采用破圈法求加权图的最小生成树。-P108

破圈法：从原图中找回路，每找到一个回路时，删除回路上权值最大的边，反复循环直到无回路。



```

1. # include<stdio.h>
2. #define n 5
3. int a[n][n];
4. int flag,am,p,q;
5.
6. INPUT()
7. {
8.     int i,j;
9.     printf("输入图的带权连接矩阵: \n");
10.    for(i=0;i<n;i++)
11.    {
12.        for(j=0;j<n;j++)
13.        {
14.            scanf("%d",&a[i][j]);
15.        }
16.    }
17. }
18.
19. OUTPUT(int a[n][n])
20. {
21.     int i,j;
22.     for(i=0;i<n;i++)
23.     {
24.         for(j=0;j<n;j++)
25.         {
26.             scanf("%d",&a[i][j]);
27.         }
28.     }
29. }

```

```

30.
31. MAX(int a1[n][n],int am1, int p1, int q1)
32. {
33.     int i,j,ptm,qtm;
34.     int max;
35.     max=0;
36.     for(i=0;i<n;i++)
37.     {
38.         if((a1[i][j]>max)&&(a1[i][j]<=am1)&&((i!=p1)|| (j!=q1)))
39.         {
40.             max=a1[i][j];
41.             ptm=i;
42.             qtm=j;
43.         }
44.     }
45.     am=max;
46.     printf("max=%5d\t",am);
47.     p=ptm;
48.     q=qtm;
49.     a[p][q]=0;
50.     a[q][p]=0;
51. }
52.
53. WSHALL(int array[n][n])
54. {
55.     int i,j,k,m=0;
56.     int r[n][n],B[n][n];
57.     for(i=0;i<n;i++)
58.     {
59.         for(j=0;j<n;j++)
60.         {
61.             r[i][j]=0;
62.             B[i][j]=array[i][j];
63.             if(array[i][j]>=1)
64.             {
65.                 B[i][j]=1;
66.             }
67.             else
68.             {
69.                 B[i][j]=0;
70.             }
71.         }
72.     }
73.     for(j=0;j<n;j++)
74.     {
75.         for(i=0;i<n;i++)
76.         {
77.             if(B[i][j]>=1)
78.             {
79.                 for(k=0;k<n;k++)
80.                 {
81.                     if(B[k][j]>=1)
82.                         B[k][j]=B[j][k]=1;
83.                 }
84.             }
85.         }
86.     }
87.     for(i=0;i<n;i++)
88.     {
89.         for(j=0;j<n;j++)
90.             if(!B[i][j])
91.             {
92.                 return 0;
93.             }
94.     }
95.     return 1;

```

```

96. }
97.
98. int main()
99. {
100.     int i,j,sm,wt=0;
101.     am=10000,p=-1,q=-1,sm=0;
102.     INPUT();
103.     for(i=0;i<n;i++)
104.     {
105.         for(j=0;j<n;j++)
106.         {
107.             if(a[i][j]>0)
108.                 sm=sm+1;
109.         }
110.     }
111.     printf("\nsm=%d\n",sm);
112.     printf("输出图的带权图邻接矩阵: \n");
113.     OUTPUT(a);
114.     printf("\n");
115.     while(sm>n-1)
116.     {
117.         MAX(a,am,p,q);
118.         flag=WSHALL(a);
119.         if(flag==1)
120.         {
121.             sm=sm-1;
122.         }
123.         else
124.         {
125.             a[p][q]=am;
126.             a[q][p]=am;
127.         }
128.     }
129.     for(i=0;i<n;i++)
130.         for(j=i;j<n;j++)
131.             wt=wt+a[i][j];
132.     printf("\n 输出最小生成树的带权邻接矩阵: \n");
133.     OUTPUT(a);
134.     printf("最下生成树的树权是: %d\n",wt);
135. }

```

## 11 推箱子问题

★问题描述:码头仓库是划分为  $n \times m$  个格子的矩形阵列。有公共边的格子是相邻格子。当前仓库中有的格子是空闲的,有的格子则已经堆放了沉重的货物。由于堆放的货物很重,单凭仓库管理员的力量是无法移动的。……………

```

1.  #include <iostream>
2.  #include <algorithm>
3.  #include <cstring>
4.  #include <cstdio>
5.  #include <cmath>
6.  #include <stack>
7.  #include <map>
8.  #include <queue>
9.  #include <climits>
10.
11. using namespace std;
12. const int MAX = 110;
13. char my_map[MAX][MAX];
14. int n, m, my_book[MAX][MAX]= {0}, Mx, My, Px, Py, Kx, Ky, mov[4][2] = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
15.
16. struct node

```

```

17. {
18.     int x, y, cnt;
19. };
20.
21. bool judge(node q)
22. {
23.     if (my_book[q.x][q.y]) return false;
24.     if (my_map[q.x][q.y] == 'S') return false;
25.     if (q.x < 0 || q.x >= n || q.y < 0 || q.y >= m) return false;
26.     return true;
27. }
28.
29. int bfs1()
30. {
31.     node q1, q2;
32.     q1.x = Mx, q1.y = My, q1.cnt = 0;
33.     my_book[Mx][My] = 1;
34.     queue <node> Q;
35.     Q.push(q1);
36.     while(!Q.empty())
37.     {
38.         q1 = Q.front();
39.         if(my_map[q1.x][q1.y] == 'P') return q1.cnt - 1;
40.         for(int i = 0; i < 4; ++ i)
41.         {
42.             q2 = q1;
43.             q2.x = q1.x + mov[i][0];
44.             q2.y = q1.y + mov[i][1];
45.             q2.cnt = q1.cnt + 1;
46.             if (judge(q2))
47.             {
48.                 Q.push(q2);
49.                 my_book[q2.x][q2.y] = 1;
50.             }
51.         }
52.         Q.pop();
53.     }
54.     return 0;
55. }
56.
57. int bfs2()
58. {
59.     memset(my_book, 0, sizeof(my_book));
60.     node q1, q2;
61.     q1.x = Px, q1.y = Py, q1.cnt = 0;
62.     my_book[Mx][My] = 1;
63.     queue <node> Q;
64.     Q.push(q1);
65.     while(!Q.empty())
66.     {
67.         q1 = Q.front();
68.         if(my_map[q1.x][q1.y] == 'K') return q1.cnt - 1;
69.         for(int i = 0; i < 4; ++ i)
70.         {
71.             q2 = q1;
72.             q2.x = q1.x + mov[i][0];
73.             q2.y = q1.y + mov[i][1];
74.             q2.cnt = q1.cnt + 1;
75.             if (judge(q2))
76.             {
77.                 Q.push(q2);
78.                 my_book[q2.x][q2.y] = 1;
79.             }
80.         }
81.         Q.pop();
82.     }

```

```

83.     return 0;
84. }
85.
86. int main()
87. {
88.     cin >> n >> m;
89.     for(int i = 0; i < n; ++ i)
90.     {
91.         cin >> my_map[i];
92.         for(int j = 0; j < m; ++ j)
93.         {
94.             if (my_map[i][j] == 'M')
95.             {
96.                 Mx = i, My = j;
97.             }
98.             if (my_map[i][j] == 'P')
99.             {
100.                 Px = i, Py = j;
101.             }
102.             if (my_map[i][j] == 'K')
103.             {
104.                 Kx = i, Ky = j;
105.             }
106.         }
107.     }
108.     int t1 = bfs1();
109.     if(t1 == 0)
110.     {
111.         cout << "No Solution!" << endl;
112.         return 0;
113.     }
114.     int t2 = bfs2();
115.     if (t2 == 0)
116.     {
117.         cout << "No Solution!" << endl;
118.     }
119.     else
120.     {
121.         cout << t1 + t2 << endl;
122.     }
123. }

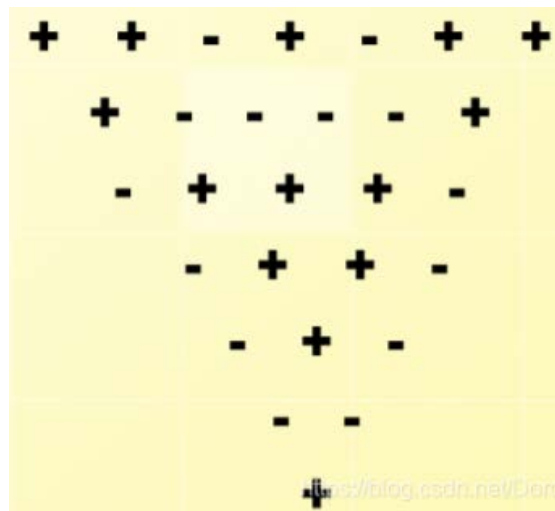
```

## 12.符号三角形 -p133

[https://blog.csdn.net/Doro\\_/article/details/106000668](https://blog.csdn.net/Doro_/article/details/106000668)

给定第一行的符号（只有+，-）数目 n，每行比上一行数目少一（形成一个倒三角），2 个相同符号下面为“+”号，2 个不同符号下面为“-”号，要求有多少种情况使得两种符号数目相同

第一行为 7 的符号三角形之一：



```
1. #include<stdio.h>
2. #define N 100
3. int n;
4. int cnt=0,sum=0;
5. int half;
6. int p[N][N]={0};
7. void backtrack(int t){
8.     int j,i;
9.     if(cnt>half||t*(t-1)/2-cnt>half) //符号大于一半时退出
10.        return;
11.     if(t>n){//到达叶结点
12.         sum++;
13.         return ;
14.     }
15.     for(i=0;i<2;i++){
16.         p[1][t]=i;
17.         cnt+=i;
18.         for(j=2;j<=t;j++){//j 层
19.             p[j][t-j+1]=p[j-1][t-j+1]^p[j-1][t-j+2];
20.             cnt+=p[j][t-j+1];
21.         }
22.         backtrack(t+1);
23.         for(j=2;j<=t;j++){
24.             cnt-=p[j][t-j+1];
25.         }
26.         cnt-=i;
27.     }
28. }
29. int main()
30. {
```



```
31.     printf("请输入第一行符号个数: ");
32.     scanf("%d",&n);
33.     half=n*(n+1)/2;//总的符号数
34.     if(half%2==1) printf("结果为 0 个\n");
35.     else {
36.         half/=2;//最开始忘记除 2 ， 除以 2 才是真正的一半；
37.         backtrack(1);
38.         printf("结果为%d 个\n",sum);
39.     }
40.     return 0;
41. }
```

### 13. N 皇后 -p135