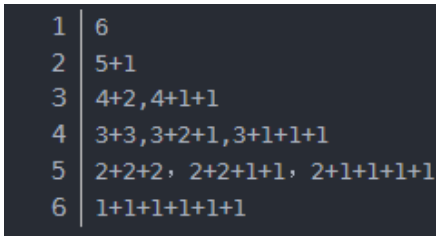
1. 整数划分
2. 棋盘覆盖
3. 合并排序
4. 集合划分
5.整数(素数)因子分解
6.最长公共子序列
7.0-1 背包问题
8. 算法实现题 3-5 编辑距离问题
9．Dijkstra 算法
10.编程实现采用破圈法求加权图的最小生成树。
11 推箱子问题
12.符号三角形
13.N 皇后

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




```cpp
#include<iostream>
using namespace std;

int split(int n,int m)
{
    if(n == 1 || m == 1) return 1;
    else if(n < m) return split(n,n);
    else if(n == m) return split(n,n-1) + 1;
    else return split(n,m-1)+split(n-m,m);
}

int main()
{
    int n;
    printf("输入划分数:");
    scanf("%d",&n);
    printf("整数划分为:%d\n",split(n,n));
    return 0;
}
```

2. 棋盘覆盖 -p21

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

```cpp
#include <iostream>

int tile = 1;        // 骨牌序号
int board[128][128]; // 二维数组模拟棋盘

// (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
void chessBoard(int tr, int tc, int dr, int dc, int size)
{
```

```
    // 递归出口
    if (size == 1)
        return;
    int s = size / 2; //分割棋盘
    int t = tile++;   //t 记录本层骨牌序号
    // 判断特殊方格在不在左上棋盘
    if (dr < tr + s && dc < tc + s)
    {
        chessBoard(tr, tc, dr, dc, s); //特殊方格在左上棋盘的递归处理方法
    }
    else
    {
        board[tr + s - 1][tc + s - 1] = t;             // 用 t 号的 L 型骨牌覆盖右下角
        chessBoard(tr, tc, tr + s - 1, tc + s - 1, s); // 递归覆盖其余方格
    }
    // 判断特殊方格在不在右上棋盘
    if (dr < tr + s && dc >= tc + s)
    {
        chessBoard(tr, tc + s, dr, dc, s);
    }
    else
    {
        board[tr + s - 1][tc + s] = t;
        chessBoard(tr, tc + s, tr + s - 1, tc + s, s);
    }
    // 判断特殊方格在不在左下棋盘
    if (dr >= tr + s && dc < tc + s)
    {
        chessBoard(tr + s, tc, dr, dc, s);
    }
    else
    {
        board[tr + s][tc + s - 1] = t;
        chessBoard(tr + s, tc, tr + s, tc + s - 1, s);
    }

    // 判断特殊方格在不在右下棋盘
    if (dr >= tr + s && dc >= tc + s)
    {
        chessBoard(tr + s, tc + s, dr, dc, s);
    }
    else
    {
        board[tr + s][tc + s] = t;
        chessBoard(tr + s, tc + s, tr + s, tc + s, s);
    }
}

int main()
{
    int boardSize = 8;                 // 棋盘边长
    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)
    // 打印棋盘
    int i, j;
    printf("\n\n\n");
    for (i = 0; i < boardSize; i++)
    {
        for (j = 0; j < boardSize; j++)
        {
            printf("%d\t", board[i][j]);
        }
        printf("\n\n\n");
    }
    return 0;
}
```

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.     {
19.     temp[k++] = a[j++];
20.      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
21.     }
22.     }
23.     while(i<=mid)
24.     temp[k++]=a[i++];
25.     while(j<=last)
26.     temp[k++]=a[j++];
27.     for(i=0; i < k;i++)
28.     a[first + i] = temp[i];//从临时数组取出放回原数组
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.     public static void main(String[] args) {
41.         // TODO Auto-generated method stub
42.         int a[] = {4,3,2,1},i;
43.         System.out.println("序列:");
44.         for(i=0;i<4;i++)
45.             System.out.println(a[i]);
46.         mergesort(a,0,3);
47.         System.out.println("逆序对数: ");
48.         System.out.println(n-1);
49.     }
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>
```

```c
#include <stdlib.h>
int jihe(int n,int m)
{
    if(n==m||m==1)
        return 1;
    else
    {
        return jihe(n-1,m-1)+m*jihe(n-1,m);
    }

}
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    int t;
    t=jihe(n,m);
    printf("%d",t);

    return 0;
}
```

5.整数(素数)因子分解

大于 1 的正整数 n 可以分解为 n=x1*x2*.....*xm，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

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

bool is_prime(int n) {
int flag = 0;
int i;
for (i = 2; i <= int(sqrt(n)); i++) {
if (n % i == 0) {
flag = 1;
break;
}
}
if (flag == 1) {
return false;
} else {
return true;
}

}

int solve(int n) {
int i;
int count = 0;
if (n == 1 || is_prime(n)) {
return 1;
} else {
for (i = 2; i <= n; i++) {
if (n % i == 0 & is_prime(i)) {
count += solve(n / i);
}
}
```

```
}
return count;
}


int main() {
int n = 12;
int c;
c = solve(100);
printf("%d", c);
}
```

6.最长公共子序列 -p54

X=ABCADCBA

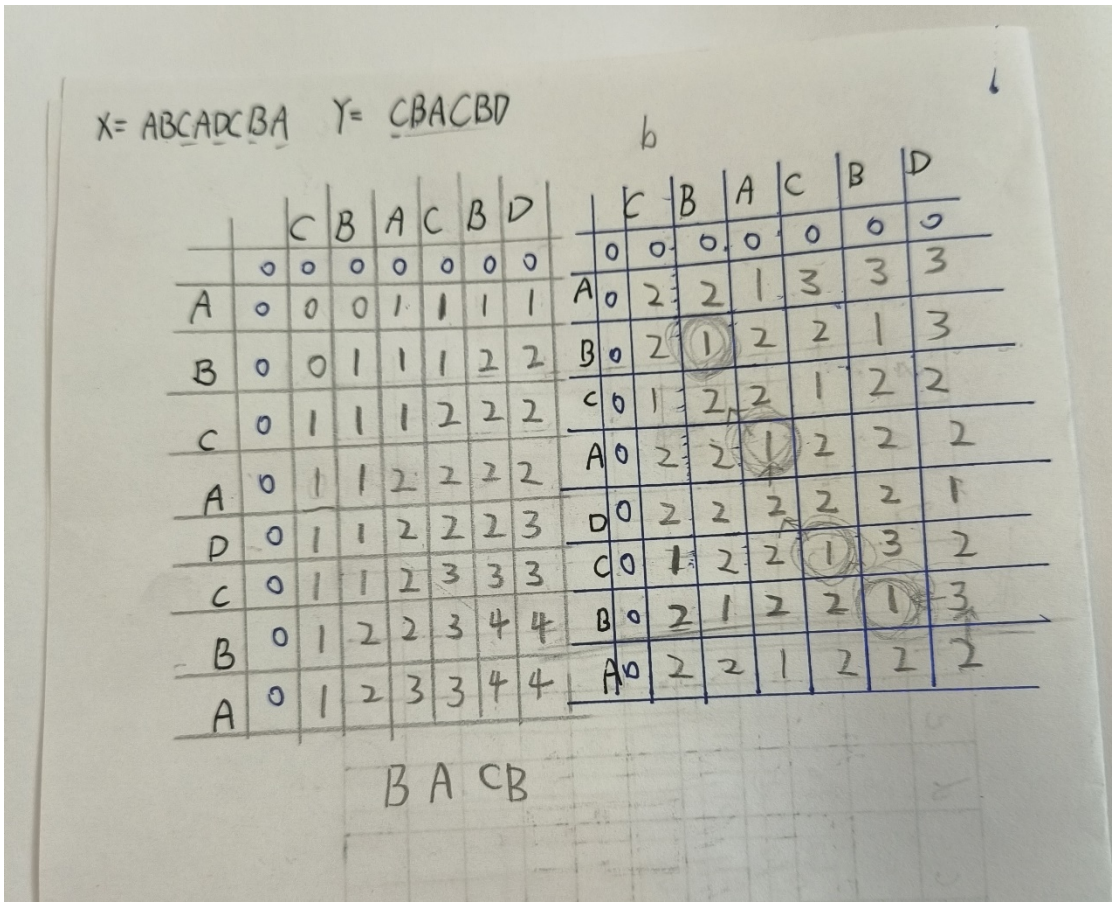
Y=CBACBD

求 X 与 Y 的最长公共子序列

```
void fun(string x,string y)
{
    int m = x.size();
    int n = y.size();
    vector<vector<int>> c(m+1, vector<int>(n+1, 0));
    vector<vector<int>> b(m+1, vector<int>(n+1, 0));
    for (int i = 1; i <= m;i++)
    {
        for (int j = 1; j <= n;j++)
        {
            if(x[i-1]==y[j-1])
            {
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            }
            else if(c[i-1][j]>=c[i][j-1])
            {
                c[i][j] = c[i - 1][j];
                b[i][j] = 2;
            }
            else
            {
                c[i][j] = c[i][j - 1];
                b[i][j] = 3;
            }
        }
    }
}
int main()
{
```

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

```
0 0 0 0 0 0 0
0 0 0 1 1 1 1
0 0 1 1 1 2 2
0 1 1 1 2 2 2
0 1 1 2 2 2 2
0 1 1 2 2 2 3
0 1 1 2 3 3 3
0 1 2 2 3 4 4
0 1 2 3 3 4 4
$$$$$$$$$
0 0 0 0 0 0 0
0 2 2 1 3 3 3
0 2 1 2 2 1 3
0 1 2 2 1 2 2
0 2 2 1 2 2 2
0 2 2 2 2 2 1
0 1 2 2 1 3 2
0 2 1 2 2 1 3
0 2 2 1 2 2 2
ABCADCBA
```



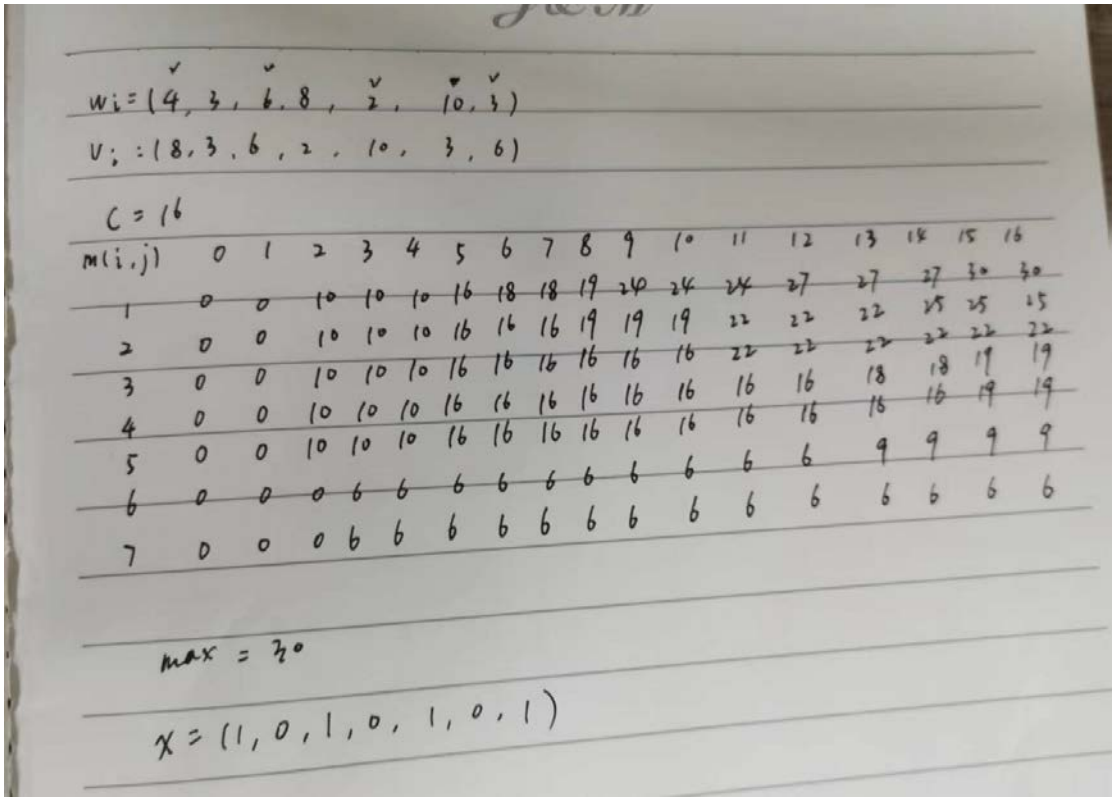
结果：BACB
Ps：上面的矩阵是 c，下面的是 b

7.0-1 背包 -p74

W={4,3,6,8,2,10,3}
V={8,3,6,2,10,3,6}
C=16

```
#include <iostream>
#include <cstring>
using namespace std;

const int N=150;

int v[N]={0,12,8,9,5};
int w[N]={0,15,10,12,8};
int x[N];
int m[N][N];
int c=30;
int n=4;
void traceback()
{
    for(int i=n;i>1;i--)
    {
        if(m[i][c]==m[i-1][c])
            x[i]=0;
        else
        {
            x[i]=1;
            c-=w[i];
        }
    }
    x[1]=(m[1][c]>0)?1:0;
}

int main()
{


    memset(m,0,sizeof(m));
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=c;j++)
        {
            if(j>=w[i])
                m[i][j]=max(m[i-1][j],m[i-1][j-w[i]]+v[i]);

            else
                m[i][j]=m[i-1][j];
        }
    }/*
    for(int i=1;i<=6;i++)
    {
        for(int j=1;j<=c;j++)
        {
            cout<<m[i][j]<<' ';
        }
        cout<<endl;
    }
*/
    traceback();
```

```
    for(int i=1;i<=n;i++)
        cout<<x[i];
    return 0;
}
```



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)插人一个字符;

(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)。

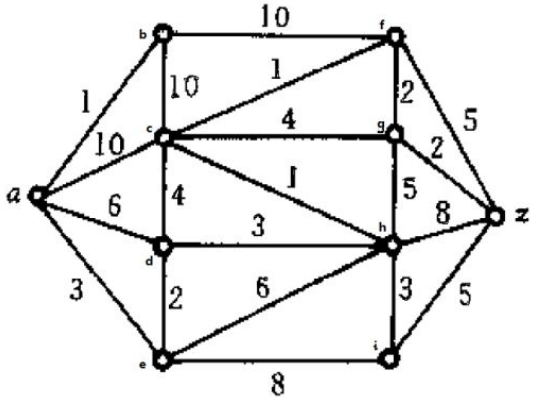
★算法设计:对于给定的字符串 A 和字符串 B.计算其编辑距离 d(A,B)。

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

```
int minDistance(String s1, String s2) {
    int m = s1.length(), n = s2.length();
    int[][] dp = new int[m + 1][n + 1];
    // base case
    for (int i = 1; i <= m; i++)
        dp[i][0] = i;
    for (int j = 1; j <= n; j++)
        dp[0][j] = j;
```

```
    // 自底向上求解
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (s1.charAt(i-1) == s2.charAt(j-1))
                dp[i][j] = dp[i - 1][j - 1];
            else
                dp[i][j] = min(
                    dp[i - 1][j] + 1,
                    dp[i][j - 1] + 1,
                    dp[i-1][j-1] + 1
                );
    // 储存着整个 s1 和 s2 的最小编辑距离
    return dp[m][n];
}

int min(int a, int b, int c) {
    return Math.min(a, Math.min(b, c));
}
```

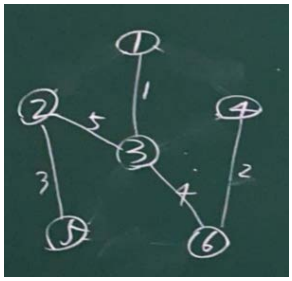
9. 根据所示的无向加权图，请应用 Dijkstra 算法求出从 a 到 z 的最短路径及其长度。要求列表显示每一步的迭代计算过程。



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。




```c
# include<stdio.h>
#define n 5
int a[n][n];
int flag,am,p,q;

INPUT()
{
    int i,j;
    printf("输入图的带权连接矩阵：\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}

OUTPUT(int a[n][n])
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
}
```

```
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;
```

```
}

int main()
{
    int i,j,sm,wt=0;
    am=10000,p=-1,q=-1,sm=0;
    INPUT();
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(a[i][j]>0)
                sm=sm+1;
        }
    }
    printf("\nsm=%d\n",sm);
    printf("输出图的带权图邻接矩阵：\n");
    OUTPUT(a);
    printf("\n");
    while(sm>n-1)
    {
        MAX(a,am,p,q);
        flag=WSHALL(a);
        if(flag==1)
        {
            sm=sm-1;
        }
        else
        {
            a[p][q]=am;
            a[q][p]=am;
        }
    }
    for(i=0;i<n;i++)
        for(j=i;j<n;j++)
            wt=wt+a[i][j];
    printf("\n 输出最小生成树的带权邻接矩阵：\n");
    OUTPUT(a);
    printf("最下生成树的树权是：%d\n",wt);
}
```

11 推箱子问题

★问题描述:码头仓库是划分为 nXm 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。……

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <stack>
#include <map>
#include <queue>
#include <climits>

using namespace std;
const int MAX = 110;
char my_map[MAX][MAX];
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}};

struct node
```

```
{
    int x, y, cnt;
};

bool judge(node q)
{
    if (my_book[q.x][q.y]) return false;
    if (my_map[q.x][q.y] == 'S') return false;
    if (q.x < 0 || q.x >= n || q.y < 0 || q.y >= m) return false;
    return true;
}

int bfs1()
{
    node q1, q2;
    q1.x = Mx, q1.y = My, q1.cnt = 0;
    my_book[Mx][My] = 1;
    queue <node> Q;
    Q.push(q1);
    while(!Q.empty())
    {
        q1 = Q.front();
        if(my_map[q1.x][q1.y] == 'P') return q1.cnt - 1;
        for(int i = 0; i < 4; ++ i)
        {
            q2 = q1;
            q2.x = q1.x + mov[i][0];
            q2.y = q1.y + mov[i][1];
            q2.cnt = q1.cnt + 1;
            if (judge(q2))
            {
                Q.push(q2);
                my_book[q2.x][q2.y] = 1;
            }
        }
        Q.pop();
    }
    return 0;
}

int bfs2()
{
    memset(my_book, 0, sizeof(my_book));
    node q1, q2;
    q1.x = Px, q1.y = Py, q1.cnt = 0;
    my_book[Mx][My] = 1;
    queue <node> Q;
    Q.push(q1);
    while(!Q.empty())
    {
        q1 = Q.front();
        if(my_map[q1.x][q1.y] == 'K') return q1.cnt - 1;
        for(int i = 0; i < 4; ++ i)
        {
            q2 = q1;
            q2.x = q1.x + mov[i][0];
            q2.y = q1.y + mov[i][1];
            q2.cnt = q1.cnt + 1;
            if (judge(q2))
            {
                Q.push(q2);
                my_book[q2.x][q2.y] = 1;
            }
        }
        Q.pop();
    }
```

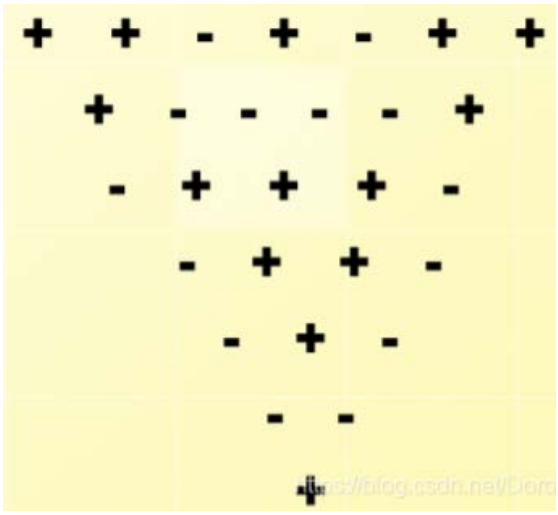
```
    return 0;
}

int main()
{
    cin >> n >>m;
    for(int i = 0; i < n; ++ i)
    {
        cin >>my_map[i];
        for(int j = 0; j < m; ++ j)
        {
            if (my_map[i][j] == 'M')
            {
                Mx = i, My = j;
            }
            if (my_map[i][j] == 'P')
            {
                Px = i, Py = j;
            }
            if (my_map[i][j] == 'K')
            {
                Kx = i, Ky = j;
            }
        }
    }
    int t1 = bfs1();
    if(t1 == 0)
    {
        cout <<"No Solution!" <<endl;
        return 0;
    }
    int t2 = bfs2();
    if (t2 == 0)
    {
        cout <<"No Solution!" <<endl;
    }
    else
    {
        cout <<t1 + t2 <<endl;
    }
}
```

12.符号三角形 -p133

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：



```
#include<stdio.h>
#define N 100
int n;
int cnt=0,sum=0;
int half;
int p[N][N]={0};
void backtrack(int t){
    int j,i;
    if(cnt>half||t*(t-1)/2-cnt>half) //符号大于一半时退出
        return;
    if(t>n){//到达叶结点
        sum++;
        return ;
    }
    for(i=0;i<2;i++){
        p[1][t]=i;
        cnt+=i;
        for(j=2;j<=t;j++){//j 层
            p[j][t-j+1]=p[j-1][t-j+1]^p[j-1][t-j+2];
            cnt+=p[j][t-j+1];
        }
        backtrack(t+1);
        for(j=2;j<=t;j++){
            cnt-=p[j][t-j+1];
        }
        cnt-=i;
    }
}
int main()
{
```

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

13. N 皇后 -p135