1. 山峰轮廓

给定一个尺寸为m x n（m和n均不超过100）的网格状矩形地图，每个网格（i, j）的边长为1，并且有一个高度h，h为0代表平原，h > 0代表山峰。请你设计C/C++程序，计算山峰轮廓的周长总和，即与平原邻接部分的周长（不包含山峰之间的重叠部分；假设地图之外均为平原）。

输入：
4 4
0 1 0 0
5 8 3 0
0 9 0 0
0 0 4 1
预期输出：
18

穷举；每个山峰轮廓为4的基础上，后续相邻山峰减2

解法一：穷举；每个山峰轮廓为4的基础上，后续相邻山峰减2

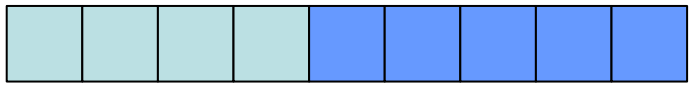
```
int Outline(int b[][100], int m, int n)
{
 int outlen = 0;
 for (int i = 0; i < m; ++i)
 {
    for (int j = 0; j < n; ++j)
    {
        if (b[i][j] > 0)
        {
            outlen += 4;
            if(i>0 && b[i - 1][j] > 0) outlen -= 2;
            if(j>0 && b[i][j - 1] > 0) outlen -= 2;
         }
    }
 }
 ……
```

```
int b[100][100] = {0};
……
for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
      cin >> b[i][j];


cout << Outline(b, m, n);
```

解法一优化：穷举；每个山峰轮廓为4的基础上，后续相邻山峰减2

当前行j列之前的方格值 上一行j列之后的方格值



a[j]

```
int main() {
    int m, n, temp, a[N]={0};
     int outlen = 0;
    cin >> m >> n;
     for (int i = 1; i <= m; ++i) {
          for (int j = 1; j <= n; ++j) {
               cin >> temp;
               if (temp > 0) outlen += 4;
               if (temp * a[j] > 0) outlen -= 2;
               if (temp * a[j-1] > 0) outlen -= 2;
               a[j] = temp;
          }
     }
     cout << outlen << endl;
     return 0;
}
```

解法二：穷举；每一边是平原就加1

给定一个尺寸为m x n（m和n均不超过100）的网格状矩形地图，每个网格（i, j）的边长为1，并且有一个高度h，h为0代表平原，h > 0代表山峰。请你设计C/C++程序，计算山峰轮廓的周长总和，即与平原邻接部分的周长（不包含山峰之间的重叠部分；假设地图之外均为平原）。

输入：
4 4
0 1 0 0
5 8 3 0
0 9 0 0
0 0 4 1
预期输出：
18

穷举；每一边是平原就加1

解法二：穷举；每一边是平原就加1

```
int Outline2(int b[][102], int m, int n)
{
 int outlen = 0;
 for (int i = 1; i <= m; ++i)
 {
    for (int j = 1; j <= n; ++j)
    {
        if (b[i][j] > 0)
        {
            outlen += (b[i - 1][j] == 0);
            outlen += (b[i + 1][j] == 0);
            outlen += (b[i][j - 1] == 0);
            outlen += (b[i][j + 1] == 0);
        }
    }
 }……
```

```
int b[102][102] = {0};
……
for (int i = 1; i <= m; ++i)
    for (int j = 1; j <= n; ++j)
      cin >> b[i][j];

cout << Outline2(b, m, n);
```

2. 递归三角形

请你设计C/C++程序，根据输入的整数n (1<=n<=7)，按照样例所示规律 用星号输出对应的递归三角形图案。

1

```
*
```

2

```
* *
*
```

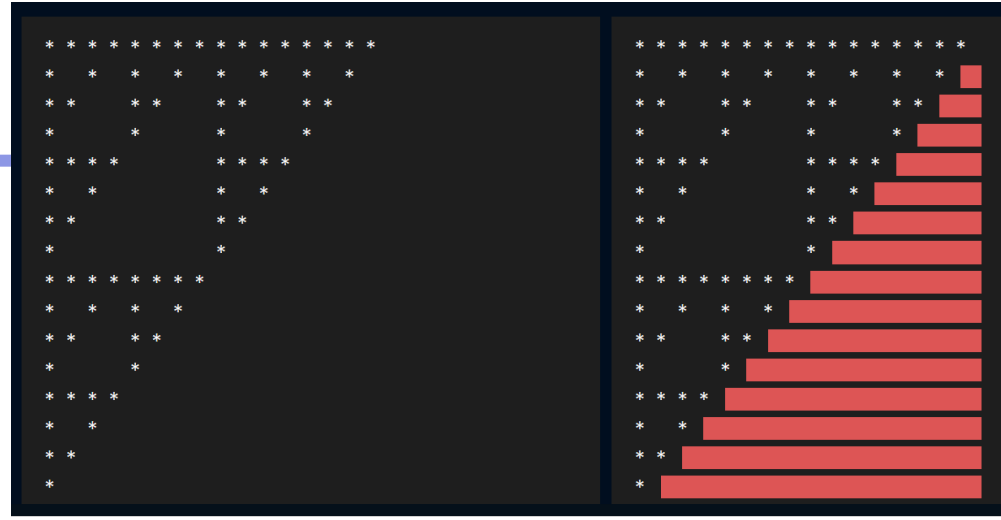
3

```
* * * *
*   *
* *
*
```

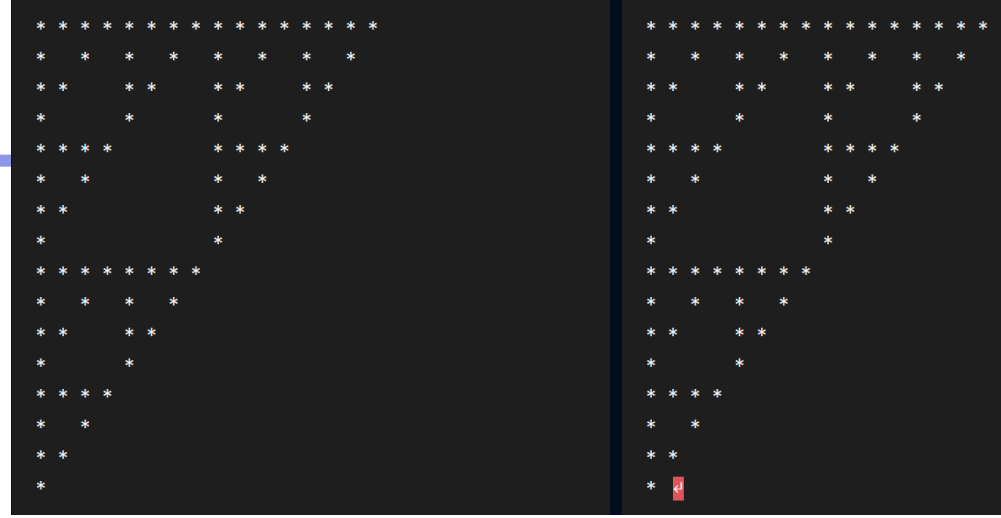
4

```
* * * * * * * *
*   *   *   *
* *     * *
*       *
* * * *
*   *
* *
*
```

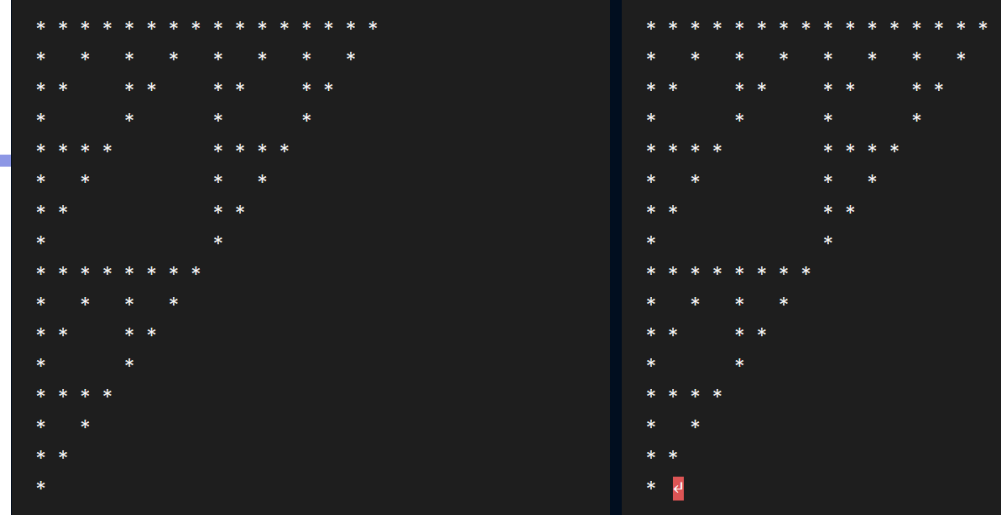
```
* * * * * * * * * * * * * * * *
*   *   *   *   *   *   *   *
* *     * *     * *     * *
*       *       *       *
* * * *         * * * *
*   *           *   *
* *             * *
*               *
* * * * * * * *
*   *   *   *
* *     * *
*       *
* * * *
*   *
* *
*
```



```
int main(){
    int n, a[100][100] = { 0 };
    scanf("%d", &n);
    int row = 0, column = 0;
    Draw(a, n, row, column);
    for (int i = 0; i < pow(2, n - 1); ++i){
        for (int j = 0; j < pow(2, n - 1); ++j){
            if (a[i][j] == 1)
                printf("* ");
            else
                printf("  ");
        }
        printf("\n");
    }
    return 0;
}
```



```
int main(){
    int n, a[100][100] = { 0 };
    scanf("%d", &n);
    int row = 0, column = 0;
    Draw(a, n, row, column);
    for (int i = 0; i < pow(2, n - 1); ++i){
        for (int j = 0; j < pow(2, n - 1); ++j){
            if (a[i][j] == 1)
                printf("* ");
            else{
                if (j >= pow(2, n - 1) - i)
                    break;
                printf("  ");
            }
        }
        printf("\n");
    }
    return 0;
}
```



```
int main(){
    int n, a[100][100] = { 0 };
    scanf("%d", &n);
    int row = 0, column = 0;
    Draw(a, n, row, column);
    int w = pow(2, n - 1);
    for (int i = 0; i < w; ++i) {
        for (int j = 0; j < w; ++j){
            if (a[i][j] == 1)
                printf("* ");
            else{
                if (j >= w - i)
                    break;
                printf("  ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

2 << (n - 2)

```
void Draw(int a[][100], int n, int row, int column)
{
    if (n == 1)
    {
        a[row][column] = 1;
    }
    else if (n == 2)
    {
        a[row][column] = 1;
        a[row][column + 1] = 1;
        a[row + 1][column] = 1;
    }
    else
    {
        Draw(a, n - 1, row, column);
        Draw(a, n - 1, row + pow(2, n - 2), column);//下部
        Draw(a, n - 1, row, column + pow(2, n - 2));//右部
    }//不是输出，只是给数组元素置值，所以下部、右部顺序不要紧
}
```

2 << (n - 3)

或

```
int main(){
    int n;
    scanf("%d", &n);
    char a[100][100];
    for (int i = 0; i < 100; ++i)
        for (int j = 0; j < 100; ++j)
            a[i][j] = ' ';
    int w = 1;
    for (int i = 1; i <= n-1; ++i)
       w *= 2;
    Draw(a, n, 0, 0);
    for (int i = 0; i < w; ++i){
        for (int j = 0; j < w - i; ++j)
            printf("%c ", a[i][j]);
        printf("\n");
    }
    return 0;
}
```

```
void Draw(char a[][100], int n, int row, int column)
{
    if (n == 1)
    {
        a[row][column] = '*';
        return;
    }

    int w = 1;
    for (int i = 1; i <= n-2; ++i)
       w *= 2;

    Draw(a, n - 1,row, column);
    Draw(a, n - 1, row + w, column);
    Draw(a, n - 1, row, column + w);
}
```

3. 组合数

组合数（Combination）是组合数学中的一个概念，表示从n个不同的元素中选取k个元素的组合数目。组合数的计算公式为
C(n, k) = n!/k!(n-k)!

请编写一个C++函数int combination(int n, int k)，使用递归的方式计算组合数C(n, k)，并在main函数里面读取n和k，然后调用上述函数进行计算。

```
int f(int n, int k) {
    if (n == k || k == 0) return 1;
    return f(n - 1, k - 1) + f(n - 1, k);
}
int main() {
    int n, k;
    cin >> n >> k;
    cout << f(n, k) << endl;
}
```

4. 爬楼梯

小蓝鲸正在爬楼梯，需要爬 n 阶才能到达楼顶。每次可以爬 1 或 2 个台阶。请问小蓝鲸有多少种不同的方法可以爬到楼顶呢?

输入格式:

一行，1个正整数n (1 <= n <= 45)

输出格式:

一行，1个正整数，表示到达楼顶的方法数

解法一：递归

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

int walk(int n){
    if(n == 1) return 1;
    else if(n == 2) return 2;
    else return walk(n - 1) + walk(n - 2);
}

int main(){
    int n;
    cin >> n;
    cout << walk(n) << endl;
    return 0;
}
```

解法一优化：递归+查表

```
int walk(int n) {
    if (rest[n] != 0)
        return rest[n];
    if (n == 1) {
        rest[n] = 1;
        return 1;
    }
    if (n == 2) {
        rest[n] = 2;
        return 2;
    }
    rest[n - 1] = walk(n - 1);
    rest[n - 2] = walk(n - 2);
    return rest[n - 1] + rest[n - 2];
}
```

解法二：动态规划

```cpp
#include<iostream>
using namespace std;
int dp[50];

int main() {
    int n;
    cin >> n;
    dp[1] = 1;
    dp[2] = 2;
    for (int i = 3; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    cout << dp[n] << endl;
    return 0;
}
```

解法三：二维递归（超时）

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

int combination(int n, int k){
    if (k == 0) return 1;
    if (k == 1 && n == 1) return 1;
    if (n < k) return 0;
        return combination(n - 1, k - 1) + combination(n - 1, k);
}
```

解法三：二维递归（超时）

```
int main(){
    int n;
    cin >> n;
    int sum = 0;
    for (int i = 0; i <= n; i += 2){
        int k = n - i, a = 0;
        a = combination(i / 2 + k, k);
        sum += a;
    }
    cout << sum << endl;
    return 0;
}
```

解法三：二维递归优化

```
int memo[100][100];
void initMemo() {
    memset(memo, -1, sizeof(memo));
}
int combination(int n, int k) {
    if (memo[n][k] != -1)
        return memo[n][k];
    if (k == 0) memo[n][k] = 1;
    else if (k == 1 && n == 1) memo[n][k] = 1;
    else if (n < k) memo[n][k] = 0;
    else memo[n][k] = combination(n-1, k-1) + combination(n-1, k);
    return memo[n][k];
}
```

5. 朋友圈

朋友的朋友也是朋友，朋友的朋友的朋友…也是朋友，自己和自己也是朋友。你和你所有的直接朋友及间接朋友构成一个朋友圈。给定nxn的矩阵f，如果f[i][j]=1，则第i+1个人和第j+1个人是直接朋友，如果f[i][j]=0，则他们不是直接朋友。请你设计C/C++程序，推断间接朋友关系，并输出f对应的朋友圈数量。

解法1：搜索

朋友关系是对称的 | 无任何朋友关系的1个人也构成一个朋友圈

```
#define N 5

int CofFrnds(int x[][N]);
int main()
{
    int f[N][N] = {0};

    for(int i = 0; i < N; ++i)
      for(int j = 0; j < N; ++j)
         scanf("%d", &f[i][j]);

    cout << CofFrnds(f) << endl;
    return 0;
}
```

	T	T	T		
	1	0	0	1	1	1
	0	1	0	1	1	0
	0	0	1	1	0	1

```
int CofFrnds(int x[][N])
{
      bool r[N];
      for(int i = 0; i < N; ++i)
            r[i] = true;
      //每个人自成一个朋友圈
      ……
```

解法1：搜索

朋友关系是对称的 | 无任何朋友关系的1个人也构成一个朋友圈

```
#define N 5

int CofFrnds(int x[][N]);
int main()
{
    int f[N][N] = {0};

    for(int i = 0; i < N; ++i)
      for(int j = 0; j < N; ++j)
         scanf("%d", &f[i][j]);

    cout << CofFrnds(f) << endl;
    return 0;
}
```

```
      T     T     T
      1     0     0     1     1     1
      0     1     0     1     1     0
      0     0     1     1     0     1
```

```
int CofFrnds(int x[][N])
{
      bool r[N];
      for(int i = 0; i < N; ++i)
            r[i] = true;
      //每个人自成一个朋友圈
      ……
```

解法1：搜索

朋友关系是对称的	无任何朋友关系的1个人也构成一个朋友圈

```
#define N 5

int CofFrnds(int x[][N]);
int main()
{
    int f[N][N] = {0};

    for(int i = 0; i < N; ++i)
      for(int j = 0; j < N; ++j)
         scanf("%d", &f[i][j]);

    cout << CofFrnds(f) << endl;
    return 0;
}
```

T	T	T	F	F	T
1	0	0	1	1	1
0	1	0	1	1	0
0	0	1	1	0	1

```
int CofFrnds(int x[][N])
{
      bool r[N];
      for(int i = 0; i < N; ++i)
            r[i] = true;
      //每个人自成一个朋友圈
      ……
```

解法1：搜索

```
    for(int i = 0; i < N-1; ++i)
        for(int j = i+1; j < N; ++j)                    //比如j为2
            if(x[i][j])                                 //0和2是朋友
                for(int k = j-1; k > 0; --k)
                    if(x[i][k])                         //0和1也是朋友
                        x[k][j] = 1;                    //则1和2也是朋友

    for(int i = 0; i < N-1; ++i)
        for(int j = i+1; j < N; ++j)
            if(r[i] && x[i][j])
                r[i] = false;           //合并朋友圈(做减法)

    int cnt = 0;
    for(int i=0; i < N; ++i)
        if(r[i])
            ++cnt;
    return cnt;
}
```

解法1优化：深度优先搜索

```
#define MAXN 102
int adj[MAXN][MAXN];
int visit[MAXN] = {0};

void dfs(int node, int n) {
    for (int i = 0; i < n; ++i) {
        if (adj[node][i] && !visit[i]) {
            visit[i] = 1;
            dfs(i, n);
        }
    }
}
```

解法1优化：深度优先搜索

```
int main() {
    int n; cin >> n;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> adj[i][j];
        }
    }
    int res = 0;
    for (int i = 0; i < n; ++i) {
        if (!visit[i]) {
            dfs(i, n);
            visit[i] = 1;
            ++res;
        }
    }
    cout << res << endl;
    return 0;
}
```

要搜索完整

解法2：并查集（数组实现）

```
#define MAXN 102
//sets[i]表示第i个朋友圈的代表
int sets[MAXN]={0};
//朋友圈个数
int cnt;

int find_set(int x){
    return sets[x];
}
```

```
void union_set(int x, int y){
    if(x == y)
        return;
    x = find_set(x); //找到所在集合
    y = find_set(y);
    if(x == y)
        return;
    //以较小元素代表集合，进行集合的合并
    int small = min(x, y);
    int large = max(x, y);
    for(int i = 0; i < MAXN; i++){
        if(sets[i] == large)
            sets[i] = small;
    }
    --cnt; //合并两个集合，则数量减一
}
```

解法2：并查集（数组实现）

```
int main() {
    int n, temp;
    cin >> n;

    for (int i = 1; i <= n; ++i)
        sets[i] = i; //每个人都是一个朋友圈（集合）
    cnt = n; //朋友圈（集合）个数初始化为n

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) {
            cin >> temp;
            if (temp == 1 && j < i) union_set(i, j);
        }
    }

    cout << cnt << endl;
    return 0;
}
```