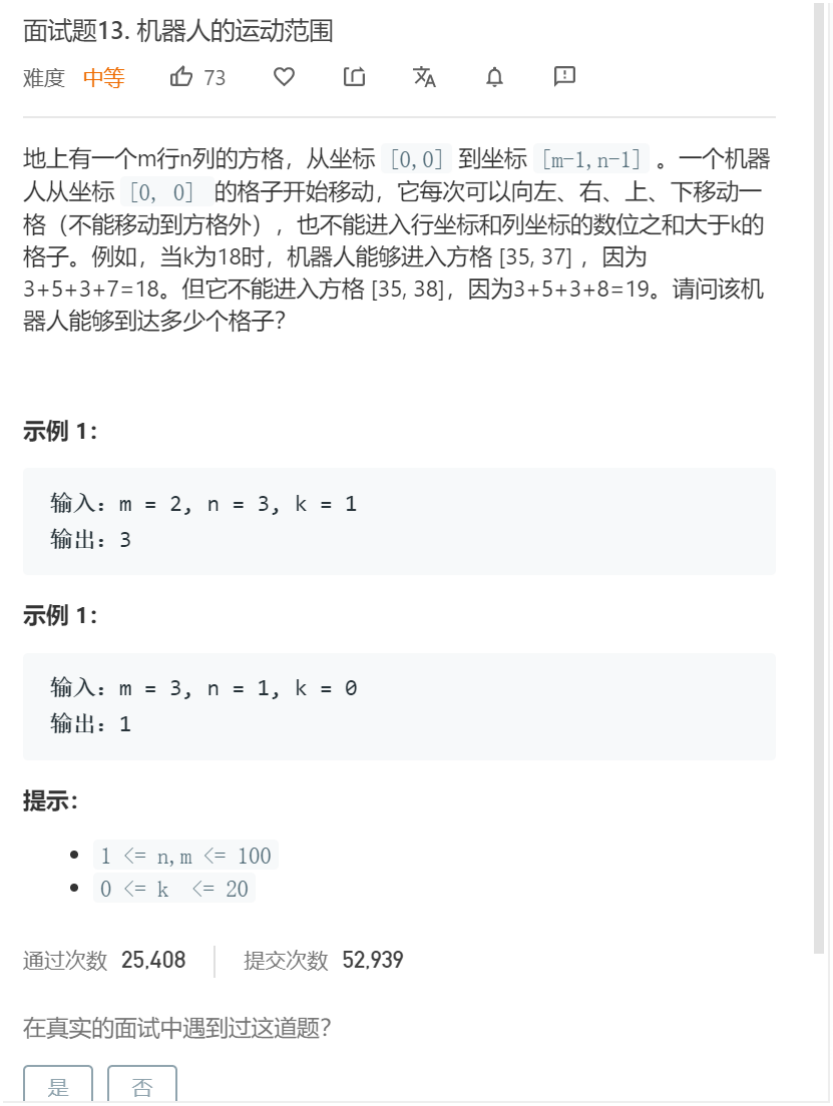
问题描述：



面试题13. 机器人的运动范围

难度 中等 73

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

示例 1：

```
输入：m = 2, n = 3, k = 1
输出：3
```

示例 1：

```
输入：m = 3, n = 1, k = 0
输出：1
```

提示：

- 1 <= n,m <= 100
- 0 <= k <= 20

通过次数 25,408 | 提交次数 52,939

在真实的面试中遇到过这道题？

是 否

解决：

本来简单的想直接暴力遍历，但是不可以。因为机器人走路是从一个点到一个点中间是连通的。例如k设置为8，（0，8）可以，（0，9）不可以，（0，10）又可以了。但是正确答案不该有（0，10）这个点。机器人的行走路径如果画出来是一个等腰三角形，其拐点在于数位和出现突变的地方。例如（0，10）（0，20）。简化想，机器人从（0，0）出发只需要搜索向下方向与向右方向即可。

正确方法是采用BFS与DFS，DFS带回溯，BFS用队列。

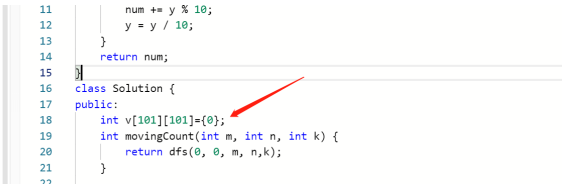
代码实现：

```
//DFS
#include<iostream>
#include<stdio.h>
#include<vector>
using namespace std;
```

```cpp
// 计算数位之和
int get(int x,int y)
{
    int num = 0;
    while (x != 0)
    {
        num += x % 10;
        x = x / 10;
    }
    while (y != 0)
    {
        num += y % 10;
        y = y / 10;
    }
    return num;
}
// 深度优先遍历
class Solution {
public:
    int movingCount(int m, int n, int k) {
        // 建立M*N矩阵，vector套vector,后面参数表示 m 个vector容器,每个容器大小为n
        vector<vector<int>>v(m,vector<int>(n));

        return dfs(0, 0, m, n,k,v);

    }

private:
    int dfs(int x, int y, int m, int n,int k, vector<vector<int>>&v) {
        if (x >= m || y >= n || v[x][y]==1 || get(x, y) > k)
            return 0;
        v[x][y] = 1;

        return 1+dfs(x + 1, y, m, n, k, v) + dfs(x, y + 1, m, n, k, v);
    }
};

int main()
{

    Solution s;
    printf("%d",s.movingCount(3, 2, 17));


    return 0;
}
```

在此基础之上，可以达到双100%

但是可以继续优化，例如使用全局变量可以节省内存消耗



:

BFS实现：自己实现了个双100%还是蛮开心的。

```
#include<iostream>
#include<stdio.h>
#include<vector>
#include<queue>
using namespace std;

// 计算数位之和
int get(int x,int y)
{
    int num = 0;
    while (x != 0)
    {
        num += x % 10;
        x = x / 10;
    }
    while (y != 0)
    {
        num += y % 10;
        y = y / 10;
    }
    return num;
}
// 广度优先遍历
class Solution {
public:
    int movingCount(int m, int n, int k) {
        // 建立M*N矩阵，vector套vector,后面参数表示 m 个vector容器,每个容器大小为n
        vector<vector<int>>v(m,vector<int>(n));

        return bfs(0, 0, m, n,k,v);

    }
typedef pair <int, int> Dot;            // 定义pair<int,int>类型名为Dot
private:
    int bfs(int x, int y, int m, int n,int k, vector<vector<int>>&v) {
        //判断当前结点是否有效，有效则入队
        if (x >= m || y >= n || v[x][y]==1 || get(x, y) > k||k<0)
            return 0;
        v[x][y] = 1;
        queue<pair<int, int>>Q;      // pair也是一种模板类型，类似容器，分为first与
second访问,使用时必须提供好类型名，二者不必相同
        Q.push(make_pair(x, y));     // make_pair 使成pair函数
        int sum = 1;
        while (!Q.empty())
```

```
        {
            Dot a = Q.front();          //取队首元素
            Q.pop();                    //pop队首元素,该元素已经过判定，下判定右方向与下方向元素
            Dot a_right,a_down;         //得到右方向元素和下方向元素
            a_right.first = a.first;
            a_right.second = a.second+1;

            a_down.first = a.first + 1;
            a_down.second = a.second;

            if (a_right.first >= m || a_right.second >= n || v[a_right.first][a_right.second] || get(a_right.first, a_right.second) > k)
                ;
            else
            {
                v[a_right.first][a_right.second] = 1;
                Q.push(a_right);
                sum++;
            }
            if (a_down.first >= m || a_down.second >= n || v[a_down.first][a_down.second] || get(a_down.first, a_down.second) > k)
                ;
            else
            {
                Q.push(a_down);
                v[a_down.first][a_down.second] = 1;
                sum++;
            }
        }
        return sum;

    }
};



int main()
{

    Solution s;
    printf("%d",s.movingCount(3, 2, 17));


    return 0;
}
```

但是BFS使用了更多的内存，应该是因为使用容器太多了。