第3章

蛮　力　法

3.1　概　　述

蛮力法(又称穷举搜索或完全搜索法)是一种简单的方法,常常直接基于问题的描述和所涉及的概念定义。蛮力法是一种通用的方法,几乎可以解决任何算法问题。其思想是生成问题的所有可能解决方案,然后根据问题选择最佳解决方案。

3.2　蛮力法的主要设计思想

3.2.1　使用蛮力法的几种情况

如果有足够的时间来处理所有的解决方案,蛮力法会是一种很好的方法,因为该算法通常很容易实现,并且总是能给出正确的答案。蛮力法可用于以下场景。

(1) 一些小规模的问题。当要解决的问题实例不多并且可以接受蛮力法的运算速度时,蛮力法的设计代价通常较为低廉。

(2) 问题找不到一个精确的解决方案。在许多情况下,蛮力法或其变种是唯一已知可获得精确解决方案的方法。

(3) 蛮力法可以为矩阵乘法、排序、搜索、字符串匹配等重要问题提供合理的算法。

3.2.2　蛮力法的求解步骤

蛮力法包括以下步骤。

(1) 用蛮力算法求解问题,需要事先确定问题的特殊性质,通常解是在解组合对象(如排列、组合或集合的子集)中。

(2) 系统地列出问题的所有潜在解决方案。

(3) 评估所有潜在的解决方案,取消不可行的解决方案,对于优化问题,跟踪直到找出最佳的解决方案。

(4) 返回最优解。

3.3　蛮力法示例与分析

3.3.1　选择排序

例 3.1　给定一个大小为 n 的数组，请将数组中的元素按从小到大进行排序并输出。

解题思路：选择排序算法的思想是，将数组分为已排序部分和未排序部分，每次对未排序部分进行扫描，找出最小值，将其和未排序部分的第一个元素交换位置，此时这个位置的元素已经属于已排序部分，经过 $n-1$ 次这样的操作，数组有序。

对于如图 3.1 所示的数据，可按照图 3.2～图 3.6 进行选择排序。

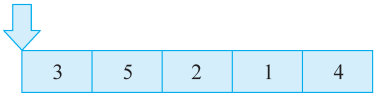

图 3.1　排序过程初始数据

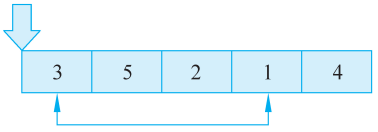

图 3.2　排序过程步骤 0

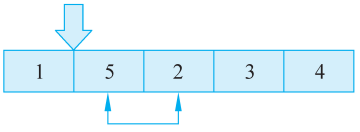

图 3.3　排序过程步骤 1

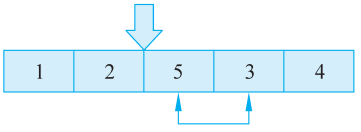

图 3.4　排序过程步骤 2

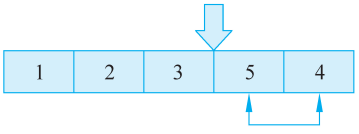

图 3.5　排序过程步骤 3

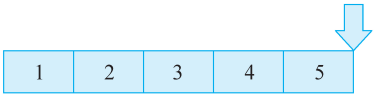

图 3.6　排序过程步骤 4

步骤 0：初始状态，所有元素都属于未排序状态。

步骤 1：将未排序部分中的最小值 1 与未排序部分的第一个元素 3 交换，排序部分元素个数加 1，未排序部分元素个数减 1，目前排序部分为{1}，未排序部分为{5,2,3,4}。

步骤 2：将未排序部分中的最小值 2 与未排序部分的第一个元素 5 交换，排序部分元素个数加 1，未排序部分元素个数减 1，目前排序部分为{1,2}，未排序部分为{5,3,4}。

步骤 3：将未排序部分中的最小值 3 与未排序部分的第一个元素 5 交换，排序部分元素个数加 1，未排序部分元素个数减 1，目前排序部分为{1,2,3}，未排序部分为{5,4}。

步骤 4：将未排序部分中的最小值 4 与未排序部分的第一个元素 5 交换，因为未排序部分只剩下两个元素，因此交换后整个数组有序。

考虑将选择排序通过代码呈现，将未排序部分起始下标 i 设为 1，重复若干次如下步骤直至 $i=n$。

(1) 扫描未排序部分，找出最小值对应下标 mnpos。

(2) 将 mnpos 对应元素与位置 i 对应元素交换。

(3) 未排序部分起始下标 i++。

选择排序要经过 $n(n-1)/2$ 次比较,因此复杂度为 $O(n^2)$。

参考代码如下。

```
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++)
        cin >> a[i];
    for(int i = 1; i <= n -1; i++)
    {
        int mn = a[i], mnpos = i;
        for(int j = i +1; j <= n; j++)
        {
            if(a[j] <mn)
            {
                mn = a[j];
                mnpos = j;
            }
        }
        if(mnpos != i)
            swap(a[i], a[mnpos]);
    }
}
```

测试结果如下。

```
请输入数组大小:
5
请输入数组元素值:
3 5 2 1 4
输出排序后结果:
1 2 3 4 5
```

3.3.2 旅行商问题

旅行商问题(Traveling Salesman Problem,TSP)描述:商品推销员要去 n 个城市推销商品,城市从 1～n 编号,任意两个城市间有一定距离,该推销员从城市 1 出发,需要经过所有城市并回到城市 1,求最短总路径长度。

解题思路:把旅行商问题看作一种排列问题,不难想出,这道题的蛮力解法即穷举所有路线。选定起点有 n 种选法,选定起点后的下一个目的地有 $n-1$ 种选择方法,再下一个有 $(n-2)$种选择方法,……总共有 $n!$种排列方法,算法复杂度达到 $O(n!)$,因此蛮力解法只能解决小规模问题。

将该题转化为加权图模型,顶点表示城市,边权表示距离,枚举所有路线并取最小值。

例 3.2 求解城市数量 $n=4$ 的 TSP 问题,其中城市直接连接关系如图 3.7 所示。

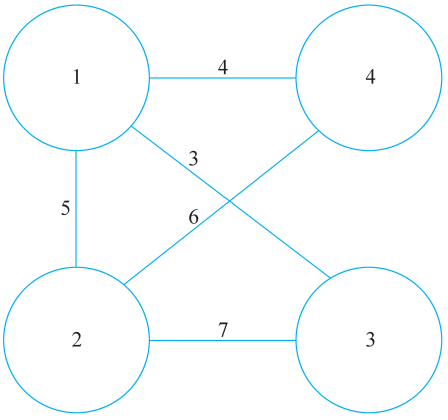

图 3.7　旅行商问题加权图模型

所有的可能路线及总距离如表 3.1 所示。

表 3.1　所有的可能路线及总距离表

路　线	总　距　离	路　线	总　距　离
1，2，3，4，1	3→4 不可行	1，3，2，4，1	20
1，2，4，3，1	4→3 不可行	1，4，2，3，1	20
1，3，4，2，1	3→4 不可行	1，4，3，2，1	4→3 不可行

利用深度优先搜索(Deep First Search，DFS)方法进行搜索，得到如图 3.8 所示的搜索树。

参考代码如下。


旅行商问题

```
void dfs(int *a, int len, int num, int u)
//数组记录历经城市的顺序,len 记录总距离,num 记录经过城市个数,
//u 表示当前在哪个城市
{
    if(num == n)
    {
        if(dis[u][1] == -1)
            return;
        len += dis[u][1];
        ans = min(ans, len);
        return;
    }
    for(int i = 2; i <= n; i++)
    {
        int flg = 0;
        for(int j = 1; j <= num; j++)
        {
```

```
            if(a[j] == i)                //排除已经经过该城市的情况
            {
                flg = 1;
                break;
            }
        }
        if(!flg && dis[u][i] != -1)   //排除无法到达该城市的情况
        {
            a[num +1] = i;
            dfs(a, len +dis[u][i], num +1, i);
        }
    }
}
```

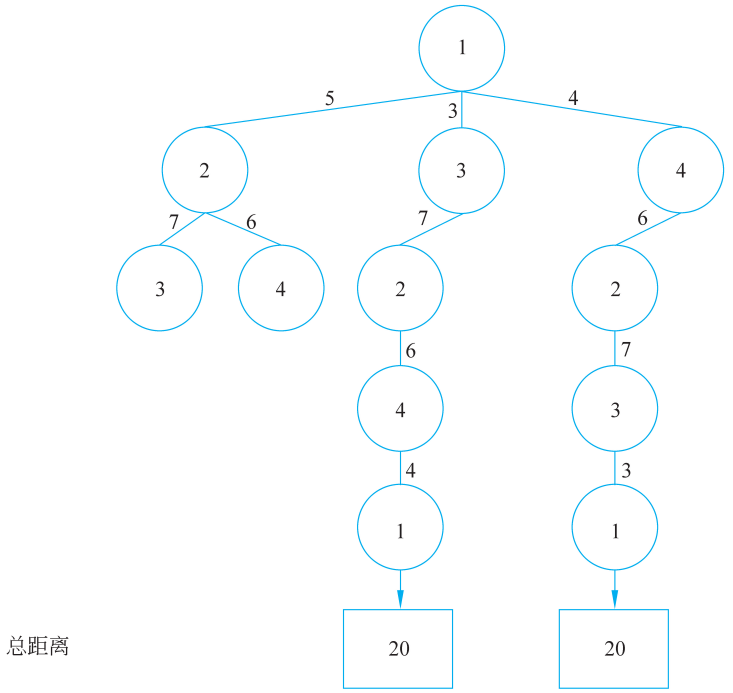

图 3.8 DFS 搜索树

最短路径长度为 20。

3.3.3 字符串匹配蛮力解决

给定一个文本串 s_1 和模式串 s_2,从文本串中找出模式串第一次出现的位置,若不存在,输出−1。

解题思路:BF 算法的原理就是将文本串的第一个字符和模式串的第一个字符比较,如果相同,则比较下一个,否则将文本串位置后移,再次尝试和模式串匹配,直至匹配成功。最多进行 nm 次比较(n、m 分别表示文本串长度和模式串长度),算法复杂度达到$O(nm)$。

例 3.3 从文本串为 s_1=aababc 中找出模式串 s_2=abc 第一次出现的位置。

文本串与模式串匹配过程如下。

（1）文本串位置＝0，模式串位置＝0，$s_1[0]=s_2[0]$＝'a'，匹配成功，模式串位置后移一位。

（2）文本串位置＝0，模式串位置＝1，$s_1[1]$＝'a'，$s_2[1]$＝'b'，匹配失败，模式串位置回到0，文本串位置后移。

（3）文本串位置＝1，模式串位置＝0，$s_1[1]=s_2[0]$＝'a'，匹配成功，模式串位置后移一位。

（4）文本串位置＝1，模式串位置＝1，$s_1[2]=s_2[1]$＝'b'，匹配成功，模式串位置后移一位。

（5）文本串位置＝1，模式串位置＝2，$s_1[3]$＝'a'，$s_2[2]$＝'c'，匹配失败，模式串位置回到0，文本串位置后移。

（6）文本串位置＝2，模式串位置＝0，$s_1[2]$＝'b'，$s_2[0]$＝'a'，匹配失败，模式串位置保持0，文本串位置后移。

（7）文本串位置＝3，模式串位置＝0，$s_1[3]=s_2[0]$＝'a'，匹配成功，模式串位置后移一位。

（8）文本串位置＝3，模式串位置＝1，$s_1[4]=s_2[1]$＝'b'，匹配成功，模式串位置后移一位。

（9）文本串位置＝3，模式串位置＝2，$s_1[5]=s_2[1]$＝'c'，匹配成功，模式串位置后移一位。

（10）模式串位置＝3，模式串与字符串匹配成功。

参考代码如下。

```
int search()
{
    int l = 0, p = 0;                  //l 表示目前匹配的文本串位置,p 表示模式串的位置
    int len = s2.size();               //匹配串的长度
    while(l+p <s1.size())
    {
        if(s1[l+p] == s2[p])           //单个字符匹配成功
        {
            p++;
            if(p == len)               //文本串和模式串匹配成功
                return l +1;
        }
        else
        {
            l++;                  //匹配失败,匹配下一个元素
            p = 0;                //p 回到模式串首
        }
    }
    return -1;
}
```

3.3.4 0-1 背包问题

有 n 种物品,第 i 种物品的质量为 W_i,价值为 V_i,有一个存放质量为 C 的背包,怎样选取物品,使得在物品总质量不超过 C 的情况下总价值最大。

解题思路:要在所有可行情况下找到最大总价值,要枚举所有可行情况,把每个物品看作一个元素,即枚举出所有可行的子集,子集生成可以通过深度优先实现。对于每个物品,选择和不选择两种状态,因此,n 个元素的子集数量达到 2^n,算法的时间复杂度为 $O(2^n)$。

例 3.4 有 4 个物品,物品的质量 $W=[1,3,5,7]$,价值 $V=[4,6,9,13]$,背包容量为 12,求背包所能存放物品的最大价值。

表 3.2 列出了问题的所有可行子集。

表 3.2 所有可行子集表

子 集	总 质 量	总 价 值
Φ	0	0
1	1	4
2	3	6
1,2	4	10
3	5	9
1,3	6	13
2,3	8	15
2,3	9	19
4	7	13
1,4	8	17
2,4	10	19
1,2,4	11	23
3,4	12	21
1,3,4	13	不可行
2,3,4	15	不可行
1,2,3,4	16	不可行


0-1 背包问题

深度优先(DFS)搜索树如图 3.9 所示。

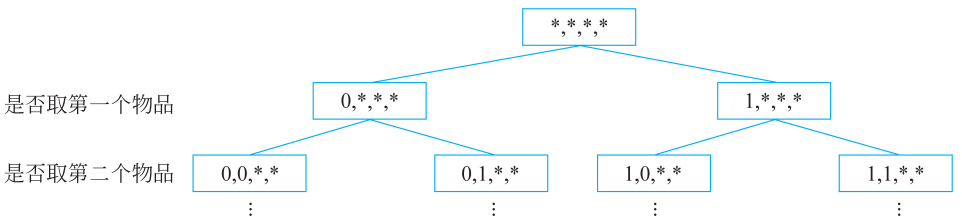

图 3.9 深度优先搜索树

参考代码如下。

```
void dfs(int num, int sumw, int sumv)
//num 表示当前是第几个物品,sumw 表示当前的总质量,sumv 表示当前价值
{
    if(num == n +1)
    {
        ans = max(ans, sumv);
        return;
    }
    if(sumw +w[num] <= c)                          //如果放得下该物品
        dfs(num +1, sumw +w[num], sumv +v[num]);   //递归构造子集
    dfs(num +1, sumw, sumv);                       //不放该物品的情况,递归构造子集
}
```

所以,当选取 1、2、4 这 3 个物品时,背包中存放物品的价值之和达到最大,为 23。

3.4　能 力 拓 展

3.4.1　连续数和

给定数字 n,试求能否将 n 分解为几个连续正整数之和,如果有多种结果,输出连续整数中最小值最大的那一组结果。如果无解,输出 -1。

解题思路:假设存在正整数 a 使得 $n=a+a+1+a+2+\cdots+a+k$。不难想到枚举 a,因为至少要分解成两个数的和,所以 $a<n/2$,但枚举 a 后 k 仍然不确定,还要依次枚举 k 判断是否为问题的解,因此,问题的复杂度为 $O(n^2)$,复杂度较高。

进一步分析可知,由于 $n=(2a+k)\times(k+1)/2$,且 a 为正整数,所以 $2a+k>k+1$,代入得 $n>(k+1)^2/2$。移项后得 $k<\sqrt{2n}-1$,所以 k 的最大值不超过 $\sqrt{2n}$。

而 a 的值也可以通过移项获得,即 $a=((n\times2)/(k+1)-k)/2$。当存在 k、a 的值为正整数时有解,并且 k 越小,a 越大,从小到大枚举 k,第一个可行解即满足题目要求的解,时间复杂度优化到了 $O(\sqrt{n})$。由此可见,就算是蛮力枚举,枚举的值不同也会对时间复杂度产生显著影响。

例 3.5　当 $n=10$ 时,输出连续数和。

该问题的求解过程如表 3.3 所示。

表 3.3　连续数和求解过程

k	a	k	a
1	9/2	3	1
2	7/3		

$n=10$ 时的连续数为 1、2、3、4。

参考代码如下。

```
int main()
{
    cin >> n;
    int flag = 0;
    for(int i = 1; i * i <= 2 * n; i++)
    {
        int a = ((n * 2) / (i +1) -i) / 2;
        if(a > 0 && (2 * a +i) * (i +1) / 2 == n)
        {
            flag = 1;          //标记找到解
            for(int j = 0; j <= i; j++)
            {
                cout <<a +j <<" ";
            }
            break;
        }
    }
    if(!flag)
        cout <<-1;
}
```

3.4.2 矩形个数

给定 $n\times n$ 的矩阵,矩阵中有 0 和 1 两个数字,现要求矩阵中只包含 0 的矩形的数量。

解题思路:枚举矩形左上角坐标 i、j,矩形右边界 k,在 i、j、k 一定的情况下能得到的满足条件的矩形个数和能向下扩展的高度有关,在 k 增加的过程中不断更新可扩展的高度。

例 3.6 给定 $n=4$,矩阵中数字如表 3.4 所示,求矩阵中只包含 0 的矩形的数量。

表 3.4 例 3.6 矩阵

0	0	1	0
1	1	0	1
0	1	0	0
0	1	0	1

该问题的蛮力搜索过程如下所示。

(1) 首先搜索第一行,如图 3.10 所示。

① $i=1$,$j=1$,$k=1$,无法再向下扩展,得到一个矩形。

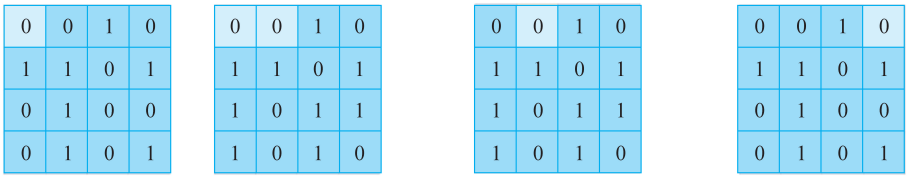

图 3.10　搜索矩阵第一行

② $i=1,j=1,k=2$，无法再向下扩展，得到一个矩形。

③ $i=1,j=2,k=2$，无法再向下扩展，得到一个矩形。

④ $i=1,j=4,k=4$，无法再向下扩展，得到一个矩形。

（2）其次搜索矩阵第二行，如图 3.11 所示。

$i=2,j=3,k=3$，可以向下扩展两层，得到 3 个矩形。

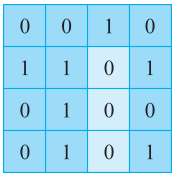

图 3.11　搜索矩阵第二行

（3）再次搜索矩阵第三行，如图 3.12 所示。

① $i=3,j=1,k=1$，可以向下扩展一层，得到两个矩形。

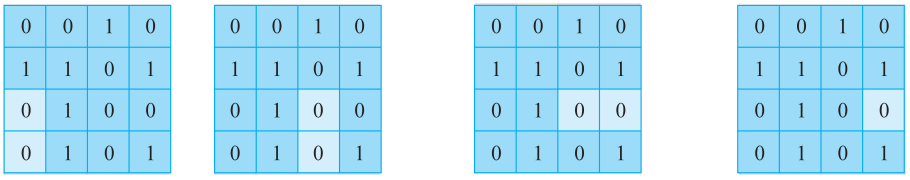

图 3.12　搜索矩阵第三行

② $i=3,j=3,k=3$，可以向下扩展一层，得到两个矩形。

③ $i=3,j=3,k=4$，无法再向下扩展，得到一个矩形。

④ $i=3,j=4,k=4$，无法再向下扩展，得到一个矩形。

（4）最后搜索矩阵第四行，如图 3.13 所示。

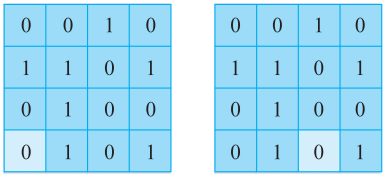

图 3.13　搜索矩阵第四行

① $i=4,j=1,k=1$，无法再向下扩展，得到一个矩形。

② $i=4,j=3,k=3$，无法再向下扩展，得到一个矩形。

因此，只要预处理出每个位置能向下扩展的最高高度 $h[i][j]$，再在枚举 k 的时候取 $\min\{h[i][j],h[i][j+1],\cdots,h[i][k]\}$，即可得到矩形的个数。

参考代码如下。

```
for(int i = n; i >= 1; i--)     //从下到上预处理每个位置可以向下扩展的最高高度
{
    for(int j = 1; j <= n; j++)
    {
        if(mp[i][j])
            h[i][j] = 0;
        else
```

```
                h[i][j] = h[i+1][j]+1;
        }
    }
    int ans = 0;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            int hei = h[i][j];                          //可扩展的高度
            for(int k = j; k <= n; k++)
            {
                hei = min(h[i][k], hei);                //可扩展的高度由最小值决定
                ans += hei;
            }
        }
    }
```

最终求得问题所示的矩阵中包含 15 个矩形。

习　　题

1. 探囊取物

小明有一个空的背包,容量为 T,他可以将两种数量无限的物品放入背包,物品 a 的体积为 A,物品 b 的体积为 B。小明可以按任意顺序将任意数量的两种物品放入包中,并且在放的过程中他可以对物品进行一次体积压缩,可以在任意时刻使用,使得放入物品的总体积从 x 变为$\left\lfloor\frac{x}{2}\right\rfloor$,小明在不超过背包容量的情况下能装的最大体积是多大。

输入描述:1 行,3 个整数 T、A、B($1\leqslant T\leqslant 5000000$,$1\leqslant A,B\leqslant T$),分别表示背包容量、物品 a 的体积、物品 b 的体积。

输出描述:1 个正整数,不超过背包容量的情况下能装的最大体积。

2. 数位乘积

给定一个 n,求 1～n 每一个数字每一位乘积的最大值。如 1～234 中最大的数位乘积是 199 的数位乘积($1\times 9\times 9=81$)。

输入描述:1 行,1 个正整数 n($1\leqslant n\leqslant 10^9$)。

输出描述:1 个正整数,表示最大数位乘积。

3. 使其相等

给定一个数组,每次可以对数组中的一个数做以下操作中的一种。

(1) 使 x 变为$\left\lfloor\frac{x}{2}\right\rfloor$。

(2) 使 x 变为 $2x$。

问最少多少次操作可以使数组中的元素都相等。

输入描述：

第一行 1 个整数 $n(1\leqslant n\leqslant 10^5)$，表示元素个数。

第二行 n 个用空格隔开的数 $a_i(1\leqslant a_i\leqslant 10^5)$，表示第 i 个元素的值。

输出描述：1 行，1 个整数，表示最少操作次数。

4. 完美图

定义一个无向图为完美图，当且仅当对于其中每一条边(v,u)有 v 和 u 互质(即 u、v 的最大公约数为 1)。当两个顶点之间没有边时不需要考虑。顶点从 1 开始标号。现给出 n 个顶点和 m 条边，是否能建立一个无重边且连通的完美图?

输入描述：1 行，2 个数 n、m，表示顶点个数和边数$(1\leqslant n,m\leqslant 10^5)$。

输出描述：如果不存在答案输出 -1。否则输出 m 行，每行输出 $v_i,u_i(1\leqslant v_i,u_i\leqslant n,v_i\neq u_i)$。

5. 统计数字

统计某个给定范围$[L,R]$的所有整数中，1～9 分别出现了几次。

例如给定范围[1,11]，数字 1 在数 1 中出现了 1 次，在数 10 中出现 1 次，在数 11 中出现 2 次，所以数字 1 在该范围内一共出现了 6 次。

输入描述：输入共 1 行，2 个正整数 L 和 $R(1\leqslant L\leqslant R\leqslant 10000)$。

输出描述：1 行，9 个数，中间用逗号隔开，第 i 个数表示数字 i 在范围内出现的次数。

6. 日期计算

小明给你一个日期，他想知道在该日期基础上经过 a 天后的日期。

输入描述：1 行，4 个整数 y、m、d、a，分别表示给定日期的年、月、日和经过的天数。

输出描述：每组数据输出 1 行，1 个结果，每行按 yyyy-mm-dd 的格式输出。

数据范围：

$1000\leqslant y\leqslant 3000$。

$1\leqslant m\leqslant 12$。

$1\leqslant d\leqslant 31$。

$1\leqslant a\leqslant 10^6$。

保证输入日期合法。

7. 01 串

现有一个由 0,01,011,0111,…组成的无限长 01 串 0010110111…，试求第 n 位上的是 0 还是 1。

输入描述：1 行 1 个整数 $n(1\leqslant n\leqslant 10^9)$。

输出描述：1 个数 0 或 1，表示第 i 位上的数为 0 还是 1。

8. 迷宫问题

先给定一个 $n \times n$ 大小的迷宫,迷宫中有 m 处障碍和一处宝藏点(x_p,y_p)。现给定起点(x_s,y_s)和终点(x_e,y_e)。

每个格子最多经过一次,有多少种从起点经过宝藏点到达终点的方案?

每次选上、下、左、右中一个方向走一格,保证起点没有障碍。

输入描述:

第一行,2 个正整数 n、m($1 \leqslant n \leqslant 5$,$1 \leqslant m \leqslant n^2$)。

第二行,起点坐标(x_s,y_s),终点坐标(x_e,y_e),宝藏点(x_p,y_p)。

接下来 m 行,每行为障碍坐标。

输出描述:1 个正整数,表示答案。

9. ABC 成比例

先给定 10 个数字 0~9,从中取出 9 个数分成 3 组,分别组成 3 个 3 位数。让 3 个数的比例为 $A:B:C$,试求出所有满足的 3 个 3 位数,若无解输出 NO。

输入描述:1 行,3 个整数 A、B、C($A<B<C$)。

输出描述:若干行,每行 3 个数字,按每行第一个数升序排列。

10. 最大乘积

给定 1 个数字,请找到一个乘积最大的连续子序列。

输入描述:

第一行 1 个数 n($1 \leqslant n \leqslant 15$),表示数组长度。

第二行 n 个数 a_i($-10 \leqslant a_i \leqslant 10$),表示各个元素。