# 第三章 暴力枚举

设想一下，你觉得家门口的山非常碍事，下决心发扬“愚公移山”精神，凭借一镐一担打算把山一点一点的移走。虽然精神值得褒奖，而且理论上是可行的，只要给予足够多的时间迟早能做到。但是，实际上并不可能给你那么多时间，所以使用这种办法在有生之年是不可能将山移开的（也许你可以使用更好的办法，比如使用魔法或者设法让天神感动，让他帮你移山）。然而，如果你只是把一个不到半人高的小沙堆给移走，那使用这种方法很快就可以完成了。

算法的世界高深莫测，但是很多问题的解决方法简单而粗暴——就是枚举出所有可能的情况，然后判断或者统计，从而解决问题。在很多程序设计比赛中，有许多比较简单的题目是可以通过枚举暴力解决的；而有的更有具有挑战性的题目虽然有更巧妙的解法，但依然可以使用枚举暴力完成部分任务。本章将介绍一些枚举与暴力策略，这是非常基础而且重要的，但是对初学者来说还是会有一些挑战。请务必理解本章之前的所有章节后再开始本章的学习。

## 第1节 循环枚举

循环枚举的意思是使用多重循环枚举所有的情况。和前面的模拟策略一样，循环枚举也没有固定做法，因此本章依然使用几个例题来介绍循环枚举策略。对于同一个问题，即使同样是循环枚举，也有可能分为三六九等。有的枚举做法比较优，运行速度较快；而有的枚举算法做了很多的无用功，效率很低。所以就算是这种简单粗暴的枚举策略，也要想想办法，看看能不能跑的更快一些。

**例 1**：统计方形加强版（洛谷 P2241，NOIP1997 普及组 加强）

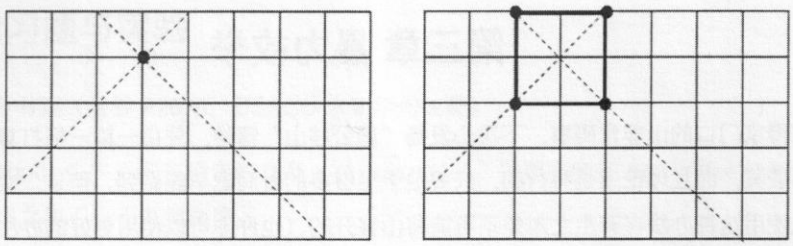
有一个 $n \times m(n,m \leq 5000)$ 的棋盘，求其方格包含多少个（四边平行于坐标轴的）正方形和长方形。

**分析**：根据题意，先来考察一下正方形和长方形的性质，并得到一个很显然结论：格点上不同行同列的两个点可以确定一个“方形”。至于具体是正方形还是长方形只需要简单看一下两点的横向距离和纵向距离是否相同。

但是这个做法是$O(n^2m^2)$的，似乎并不能胜任这道题目。那么怎么办？遇到这种情况，就需要及时更换思路：减少枚举量，或者寻找其他能枚举的要素。

**思路 1**：减少枚举量？能否只枚举方形上的一个点？不妨这样想：如果现在知道了一个点的坐标是 (x,y)，那么问题就变成了如何快速统计以它为顶点的正方形和长方形数量。

观察下图(a)，n=8,m=5 时对于圆点 (x=3,y=4) 来说，虚线上的格点都可以对它构成正方形，剩下的点都可以对它构成长方形。虚线上的格点数量是 3+4+1+1=9。可以写成用 n,m,x,y 表示公式：min(x,y)+min(y,n-x)+min(n-x,m-y)+min(m-y,x)，而剩下长方形的点用 n*m 相减正方形的点就可以得到了。




*【图：统计方形的数量】*

但是这里还有个问题。如图 (b)，这个粗线正方形被这 4 个圆点分别计算了一次，共被统计了 4 次（试试自己画图，还可以证明每个长方形也被算了四遍），解决方法很简单：只需要把计算答案除以 4 即可。代码如下：

```
#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long LL;
int main() {
    LL n, m, squ = 0, rec = 0;
    scanf("%lld%lld", &n, &m);
    for (LL x = 0; x <= n; x++)
        for (LL y = 0; y <= m; y++) {
            LL tmp = min(x, y) + min(y, n - x) + min(n - x, m - y) + min(m - y,
 x);
            squ += tmp;
            rec += n * m - tmp;
        }
    printf("%lld %lld", squ / 4, rec / 4);
}
```

但是，这个做法看起来很不优美：上面那个式子十分复杂，最后统计的答案也是重复的。有没有什么解决方案？

**思路 2：**去掉重复情况？那我能不能只枚举方形的右下角顶点？这个想法太棒了，它直接保证了每个方形只被数到了一次。对应地，构成正方形的点变成了左上角的斜线的格点；而左上方向剩下的点都能构成长方形，只需要用 x*y 减去正方形的个数。更改算法，得到这样的代码：

```
    for (LL x = 0; x <= n; x++)
        for (LL y = 0; y <= m; y++) {
            LL tmp = min(x, y);
            squ += tmp;
            rec += x * y - tmp;
        }
```

**思路 3：**枚举其他要素？那我能不能枚举方形的边长？那么这个问题就变成怎么快速统计边长 n×m 的大矩形中包含了多少边长为 a×b 的小矩形。使用类似的思路，只考虑这个小矩形右上角有多少种可能性，经过简单的画图运算后，答案就是 (n-a+1)(m-b+1)。

```
    for (LL a = 1; a <= n; a++)
        for (LL b = 1; b <= m; b++)
```

```
        if (a == b)
            squ += (n - a + 1) * (m - b + 1);
        else
            rec += (n - a + 1) * (m - b + 1);
```

到此为止，已经获得了足够简洁的思路和代码，但是这题能不能更快呢？

**思路 4**：减少枚举量？这里正方形与长方形好像有一定数量关系，我能不能只枚举正方形的边长，算出正方形的个数，最后再算出矩形的个数？可以注意到，每个矩形不是长方形就是正方形，而且矩形的数量比较好求（一个矩形由它四边所在的直线确定，那么枚举两条横线和两条竖线可知，矩形总数量是 $1/2n(n+1)\times1/2m(m+1)$，那么正方形和长方形数量总和就是 $n(n+1)m(m+1)/4$。

这让计算长方形数量带来了便利，因为正方形比长方形规则，如果可以快速计算正方形数量，也就可以算出长方形数量。而事实上，稍微观察一下上一个思路的代码就可以发现很大的优化余地：a==b，那似乎只要枚举 a 或 b 其中一个就能数正方形了！

```
for (LL a = 1; a <= min(m, n); a++)
    squ += (n - a + 1) * (m - a + 1);
rec = n * (n + 1) * m * (m + 1) / 4 - squ;
printf("%lld %lld", squ, rec);
```

我们通过更改枚举不同的要素和减少枚举量，以优秀的复杂度和越来越简洁易懂的代码来解决题目。读者在今后的训练过程中可以尝试多切换枚举方式来获得更优秀的解题效果。

**例 2**：烤鸡（洛谷 P2089）

烤鸡时需要加入 10 种配料，每种配料可放 1 到 3 克，美味程度是所有配料质量之和。给出一个美味程度 n(n<=5000)，按照字典序输出配料搭配的方案数量和所有的搭配方案。如果 n 超过 30 请输出 0。

**思路 1**：首先，美味程度最多只能达到 3×10=30，至少是 10。不在这个范围内的美味程度都是达不到的。如果考虑对每种配料都用一个 for 循环来控制它的用量，那么这题的思路就变得十分简单——似乎只要写完十层 for 循环，再加个 if 判断，就能找到所有符合条件的情况。

题意中字典序输出的意思就是前面的数越小，这个方案就越排在前面。因为的 10 层 for 蕴含了这个性质（想一想，为什么），所以字典序对解法没有影响。具体代码如下：

```
#include <cstdio>
using namespace std;
#define rep(i, a, b) for (int i = a; i <= b; i++)
int main()
{
    int n, ans = 0, cnt = 10;
    scanf("%d", &n);
    rep(a, 1, 3) rep(b, 1, 3) rep(c, 1, 3) rep(d, 1, 3) rep(e, 1, 3)
        rep(f, 1, 3) rep(g, 1, 3) rep(h, 1, 3) rep(i, 1, 3) rep(j, 1, 3)
            if (a + b + c + d + e + f + g + h + i + j == n)
                ans++;
    printf("%d\n", ans);
    rep(a, 1, 3) rep(b, 1, 3) rep(c, 1, 3) rep(d, 1, 3) rep(e, 1, 3)
        rep(f, 1, 3) rep(g, 1, 3) rep(h, 1, 3) rep(i, 1, 3) rep(j, 1, 3)
            if (a + b + c + d + e + f + g + h + i + j == n)
```

```
                printf("%d %d %d %d %d %d %d %d %d\n", a,b,c,d,e,f,g,h,i,j);
    return 0;
}
```

这里出现了宏的另外一种用法：构造语句。写下 rep(a,1,3)时，编译器会自动在代码中把你的这句话替换成 for (int a = 1; a <= 3; a++)。但是要注意的是，宏定义只会做简单的字符串替换。如果你定义了#define prod(a,b) a*b，然后你又写了 prod(a+b,c)，编译器将会把它理解成 a+b*c，并非你想的(a+b)*c。一个解决方案是定义宏时勤加括号，如#define prod(a,b) (a)*(b)，这样可以有效避免出现运算优先级的 bug。

**思路 2**：这题还能再优化吗？如果 a,b,c,d,e 加起来已经超过 n，那么显然 f,g,h,i,j 就没有继续尝试枚举的必要了。针对这个问题，对这题进行枚举剪枝——限定每个变量范围。比如 e，在满足 1≤e≤3 的同时，还能做到更好的估计：n-15-a-b-c-d≤e≤n-5-a-b-c-d。这个不等式左侧是假设后面都取 3——那么 e 至少这么大（不能再小了）；右侧是假设后面都取 1——那么 e 最多这么大（不能再大了）。

虽然之前的纯粹枚举已经可以胜任这道题，但是在这里提出一个（已经可以说是最严格的）优化思路，供读者拓宽视野。代码如下，虽然这样的暴力代码看起来不是那么的优雅。

```
#include <cstdio>
#include <algorithm>
using namespace std;
#define rep(i, a, b) for (int i = max(1, a); i <= min(3, b); i++)
int li[60000][10];
int main()
{
  int n, ans = 0, cnt = 10;
  scanf("%d", &n);
  rep(a, n-27, n-9)
   rep(b, n-24-a, n-8-a)
    rep(c, n-21-a-b, n-7-a-b)
     rep(d, n-18-a-b-c, n-6-a-b-c)
      rep(e, n-15-a-b-c-d, n-5-a-b-c-d)
       rep(f, n-12-a-b-c-d-e, n-4-a-b-c-d-e)
        rep(g, n-9-a-b-c-d-e-f, n-3-a-b-c-d-e-f)
         rep(h, n-6-a-b-c-d-e-f-g, n-2-a-b-c-d-e-f-g)
          rep(i, n-3-a-b-c-d-e-f-g-h, n-1-a-b-c-d-e-f-g-h)
           rep(j, n-a-b-c-d-e-f-g-h-i, n-a-b-c-d-e-f-g-h-i) {
             li[ans][0]=a;li[ans][1]=b;li[ans][2]=c;li[ans][3]=d;
             li[ans][4]=e;li[ans][5]=f;li[ans][6]=g;li[ans][7]=h;
             li[ans][8]=i;li[ans][9]=j;
             ans++;
           }
  printf("%d\n", ans);
  for(int i=0; i<ans; i++) {
    for(int j=0; j<10; j++)
      printf("%d ", li[i][j]);
    printf("\n");
  }
  return 0;
}
```

这里使用了一个数组 li 来记录符合要求的答案，这样就可以不需要重复枚举两次大循环了，又加快了一定的运行速度。至于这个数组要开多少，可以估计一下：10 个变量每个变量有 3 种取值，那么一共最多有 $3^{10}$ =

59049 种排列组合方式。不过实际上要不到这么多，读者也可以先运行一下不带记录答案的版本，输入从 10 到 30 都跑一遍，记录一下最多可能出现多少种结果（打擂台），然后确定数组大小。

**例 3**：三连击升级版（洛谷 P1618）

将 1，2，…，9 共 9 个数分成三组，分别组成三个三位数，且使这三个三位数的比例是 $A:B:C(A<B<C<10^9)$，试求出所有满足条件的三个三位数，若无解，输出 No!!!。

**分析**：按照套路，先来找这里可枚举的元素：三个三位数。

**思路 1**：直接枚举三个三位数再检验吗？$O(9!)$？其实还是可行的。

这里介绍一个用 STL 的简便写法。next_permutation(start,end)是 algorithm 标准库中的一个标准函数，它可以在表示 [start,end) 内存的数组中产生严格的下一个字典序排列。具体来说就是 [2,3,1] 可以变成 [3,1,2]，再下一步就是 [3,2,1]。

那么这道题只需要利用 9 的全排列，把全排列里的 9 个数位分别“划给”三个三位数就可以产生所有情况了：

```
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
int a[10];
int main() {
    long long A, B, C, x, y, z, cnt = 0;
    cin >> A >> B >> C;
    for (int i = 1; i <= 9; i++)
        a[i] = i;
    do {
        x = a[1] * 100 + a[2] * 10 + a[3];
        y = a[4] * 100 + a[5] * 10 + a[6];
        z = a[7] * 100 + a[8] * 10 + a[9];
        if (x * B == y * A && y * C == z * B)//避免浮点误差和爆 long long 的小技巧
            printf("%lld %lld %lld\n", x, y, z), cnt++;
    } while (next_permutation(a + 1, a + 10));
    if (!cnt)puts("No!!!");
    return 0;
}
```

本着尽可能降低复杂度的初衷，再换一个枚举切入点。

**思路 2**：好像可以枚举一个三位数来确定另外两个？$O(1000)$？根据题设性质大幅度降低了算法的复杂度。只要枚举第一个数，就可以根据比例确定另两个（不一定存在），最后检验得到的结果是否总共同时具有 9 个不同数位即可。

```
#include <cstdio>
#include <algorithm>
using namespace std;
int b[20];
void go(int x) { // 分解三位数到桶里
```

```
    b[x % 10] = 1;
    b[x / 10 % 10] = 1;
    b[x / 100] = 1;
}
bool check(int x, int y, int z) {
    memset(b, 0, sizeof(b));
    if (y > 999 || z > 999) return 0;
    go(x), go(y), go(z);
    for (int i = 1; i <= 9; i++)
        if (!b[i]) return 0;
    return 1;
}
int main() {
    long long A, B, C, x, y, z, cnt = 0;
    cin >> A >> B >> C;
    for (x = 123; x <= 987; x++) {
        if (x * B % A || x * C % A) continue;
        y = x * B / A, z = x * C / A;
        if (check(x, y, z))
            printf("%lld %lld %lld\n", x, y, z), cnt++;
    }
    if (!cnt) puts("No!!!");
    return 0;
}
```

**思路 3**：按照上面这么思路，好像枚举 k=x/A=y/B=z/C 中的 k 可以去掉更多的重复情况？虽然枚举的状态可能是更少了，但是复杂度没有明显降低，有兴趣的读者可以亲自尝试一下，注意要保证 A、B、C 互质才能直接枚举 k。

## 第2节 子集枚举

**例 4**：选数（洛谷 P1036，NOIP2002 普及组）

从 $n(n \le 20)$ 个整数中任选 $k$ 个整数相加，求有多少种选择情况可以使和为质数。

**分析**：本题可以认为是从一个有 n 个数字的集合中挑选出一些数字（也就是子集），然后判断该子集是否满足某个性质（其和是质数）。集合枚举的意思是从一个集合中找出它的所有子集。集合中每个元素都可以被选或不选，含有 n 个元素的集合总共有 $2^n$ 个子集（包括全集和空集）。

考虑集合 A={1,2,3,4,5} 和它的两个子集 A1={1,3,4,5}，A2={1,4,5}，A3={3}，A4={2,3}。按照某个顺序，把全集 A 中的的每个元素在每个子集中的出现状况用 0（没出现）和 1（出现了）表示出来：

| A 中元素 | 1 | 2 | 3 | 4 | 5 | 二进制 | 对应十进制 |
|---|---|---|---|---|---|---|---|
| 在 A1 中的出现情况 | 1 | 0 | 1 | 1 | 1 | 11101 | a1=29 |
| 在 A2 中的出现情况 | 1 | 0 | 0 | 1 | 1 | 11001 | a2=25 |
| 在 A3 中的出现情况 | 0 | 0 | 1 | 0 | 0 | 00100 | a3=4 |
| 在 A4 中的出现情况 | 0 | 1 | 1 | 0 | 0 | 00110 | a4=6 |

那么可以发现 A 的子集 A1 可以表示为一个“二进制数” 11101，对应十进制变量 a1=29[21]；反之，这个数字也可以表示子集 A1。注意这边的集合是大写字母，集合对应的数字是小写字母。同理，A2 可以表示为二进制数 11001，A3 可以表示成 00100，A4 可以表示成 00110。此时找到了一种让子集对应于二进制数的很直观的方法，但是这个表现法的威力不仅限于此。

本例一共有 5 个元素，表示仅包含第 $i(1 \le i \le 5)$ 个元素的集合的数字可以使用位移运算构造，写成 i<<(i-1)；而包含所有元素的全集可以表示成 a=(1<<n)-1，空集表示为 0，请读者自行验证。此外，集合之间可能存在一些联系。一些集合的常用关系有下面几种：

并集：从元素选择角度来说就是 A2、A3 包含的元素合并起来能够得到 A1。可以发现，A1 的每一位都等于对应位 A2 or A3 的结果——这不就是按位或运算吗？编程验证可得 a1=a2|a3。只需要把表示两个子集的二进制数进行或运算即可表示两个子集的并集。我们可以发现子集与二进制密不可分的关系。

交集：交集是指两个集合中同时存在的元素组成的集合，从逻辑上推导出交集含有“与”这个逻辑。类比前面的并集运算，不难猜出：当需要表示两个子集的交集时，可以把表示两个子集的二进制数与运算，即 a3=a1&a4。

包含：集合 A2 的所有元素都在 A1 中出现，说明 A1 包含 A2。易知 A1 并 A2 是 A1，同时 A1 交 A2 是 A2，也就是判断 A1 是否包含 A2 可以写成(a1|a2==a1)&&(a1&a2==a2)。

属于：属于是指某个元素在集合中，是包含的一种特殊情况——只需检查单独某项元素构成的集合是否是另一个集合的子集。一般地，可以先用左位移运算构造出那个仅含一个项的集合，然后再和原集合取交，若不为空集则为命题为真。如果要判断第 3 个元素是否属于 A1，可以写成 1<<(3-1)&a1。

补集：补集是指全集去除了某个集合后剩下元素组成的集合。按照上面的启发，大家应该能够猜出来可以用异或运算来表示一个集合对全集的补集，例如 A2 对于全集的补集就是 A3。A2 的补集可以表示为 a^a2，注意这里的^符号是指 C++ 里的异或运算。

回到例题来加深一下对上面技能的理解。首先考虑如何枚举 n 个元素组成的集合中的含有 k 个元素的子集，如果用 n 位二进制数表示子集，那么目标就是找到所有恰好只有 k 个 1 的二进制数。统计二进制中 1 的个数可以用内建函数__builtin_popcount()，它能直接返回一个数二进制下 1 的个数；当然也可以逐位分解或者逐位确认。

在找到了一个含有 k 个元素的子集后，需要把它表示的子集里的所有数提取出来：简单来说就是检查一下每个数是否属于这个集合，这个技巧刚刚提到过。剩下的事情就是把数字加起来再判断它是否为质数即可。

```
#include <iostream>
using namespace std;
int a[30];
bool check(int x) { // 判断质数
    for (int i = 2; i * i <= x; i++)
        if (x % i == 0)return 0;
    return 1;
}
```

[21] 如果还不是很了解二进制，可以阅读本书《位运算与进制转换》部分。请注意低位在右，表示元素 1 的二进制位在最右边，因此二进制数字需要左右反过来。

```
int main() {
    int n, k, ans = 0;
    cin >> n >> k;
    for (int i = 0; i < n; i++)scanf("%d", &a[i]);
    int U = 1 << n;//U-1 即为全集
    for (int S = 0; S < U; S++)//枚举所有子集[0,U)
        if (__builtin_popcount(S) == k) { //找到 k 元子集
            int sum = 0;
            for (int i = 0; i < n; i++)
                if (S & (1 << i))sum += a[i];//如果第 i 个元素在 S 中
            if (check(sum))ans++;
        }
    cout << ans;
    return 0;
}
```

虽然总是用“二进制数”来描述这些表示集合的数，但在实际操作中会把这些二进制数当做一个整体存储为单独的数字（可以认为存下了二进制数对应的十进制数），而不会把它的每一位分别存储，也不会区别对待它们与普通常量。

**例 5**：组合的输出（洛谷 P1157）

从自然数 1,2,…,$n$ 中任选 $r$ 个数作为一个组合，并输出所有的组合情况，每个组合中的数字从小到大输出。$(1 \le r \le n \le 21)$。

**分析**：乍看起来这题比前一题还要简单，但是要多考虑一步怎么对应题中要求的字典序。可以倒过来从全集枚举到 0，从高位到低位分别表示元素 1 到 n，就可以让 1 尽量出现在靠前（左）的位置，比如，可以让 10110 早于 10101 出现，符合题目的字典序要求。

```
#include <iostream>
using namespace std;
int a[30];
int main() {
    int n, r;
    cin >> n >> r;
    for (int S = (1 << n) - 1; S >= 0; S--) { //从全集枚举到 0
        int cnt = 0;
        for (int i = 0; i < n; i++)
            if (S & (1 << i))
                a[cnt++] = i; // 分离记录每一位
        if (cnt == r) {
            for (int i = r - 1; i >= 0; i--)
                printf("%3d", n - a[i]); //因为用高位表示 1，所以需要反过来输出
            puts("");
        }
    }
    return 0;
}
```

枚举子集的算法时间复杂度是 $O(2^n)$，一般情况下 1 秒钟可以枚举包含 20 到 30 个元素的集合的子集。在枚举的时候，如果对顺序有要求，就要确定枚举的方向和每一位代表什么元素。

## 第3节 排列枚举

排列枚举，顾名思义，就是要求枚举所有元素排列。例如元素 1、2、3 的所有排列有 [1,2,3] [1,3,2] [2,1,3] [2,3,1] [3,1,2] [3,2,1]，一共 6 种情况。绝大多数题目可以被之前介绍过的 next_permutation 函数，生成各个元素的不同排列，然后再进行判断或者统计即可轻松解决。

**例 6**：全排列问题（洛谷 P1706）

输出自然数 1 到 n 所有不重复的排列，即 n 的全排列。

**分析**：考察 next_permutation 的使用。从最小的排列开始（所有元素从小到大），每次更换成下一个排列，然后输出。next_permutation 是有返回值的，没有下一个排列的时候就会返回 0。

```
#include <cstdio>
#include <algorithm>
using namespace std;
int a[10], n;
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        a[i] = i;
    do {
        for (int i = 1; i <= n; i++)
            printf("%5d", a[i]);
        puts("");
    } while (next_permutation(a + 1, a + n + 1));
    return 0;
}
```

**例 7**：火星人（洛谷 P1088，NOIP2004 普及组）

对于 $n$ 的全部排列，找到一个给定排列向后的第 $m$ 个排列。

**分析**：考察 next_permutation 的熟练使用。

```
#include <cstdio>
#include <algorithm>
using namespace std;
int a[10010], n, m;
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (; m--;)
        next_permutation(a + 1, a + n + 1);
    for (int i = 1; i <= n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

枚举所有全排列的算法时间复杂度是 $O(n!)$，一般情况下 1 秒钟很难枚举超过 11 个元素的全排列。

## 第4节 课后习题与试验

**习题 1**：涂国旗（洛谷 P3392）

一个由 $N \times M$ 的小方块组成的旗帜，且满足以下条件可以认为是俄罗斯国旗。

- 从最上方若干行（不小于 1）的格子全部是白色的。
- 接下来若干行（不小于 1）的格子全部是蓝色的。
- 剩下的行（不小于 1）全部是红色的。

有一个 $N \times M$ 的小方块组成的布料，每个格子是蓝色、白色、红色之一。希望改动最少数量的格子，使其变成俄罗斯国旗。至少要修改多少个格子？

**习题 2**：First Step（洛谷 P3654）

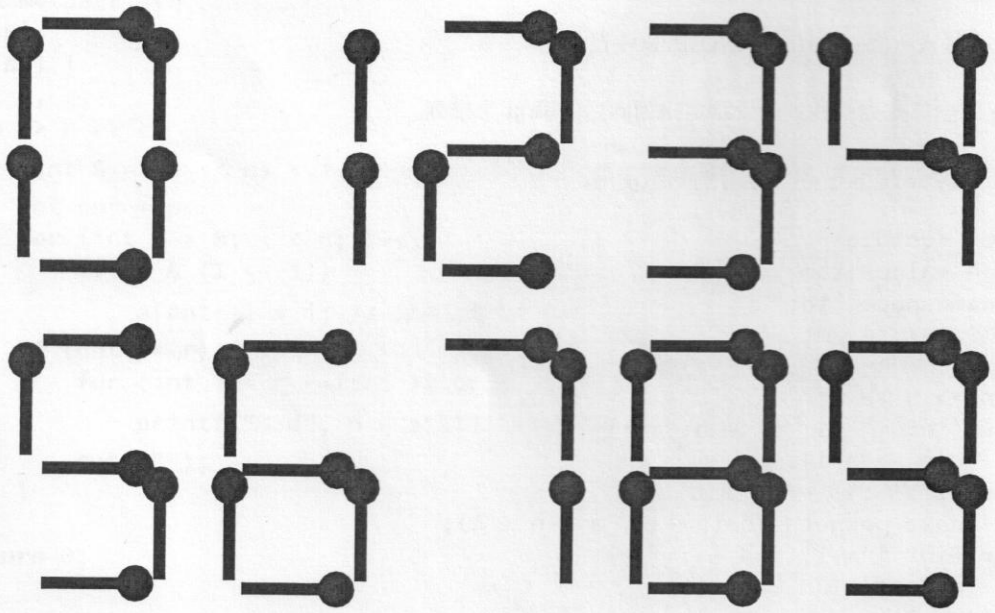
现有 $N \times M(N, M \leq 100)$ 小方格组成的篮球场，每个小方格可能是 .（空地）后者是#（障碍）。现在有一列 $1 \times K$ 的队员，每人占用一小格，排成一排站在空地上（可横可竖），求站位方案数量。

**习题 3**：回文质数（洛谷 P1217，USACO Training）

写一个程序来找出范围$[a, b](5 \leq a < b \leq 10^8)$间的所有回文质数。例如 151 既是一个质数又是一个回文数(从左到右和从右到左是看一样的)，所以 151 是回文质数。

**习题 4**：火柴棒等式（洛谷 P1149，NOIP2008 提高组）

给你 n 根火柴棍，你可以拼出多少个形如“A+B=C”的等式？等式中的 $A$、$B$、$C$ 是用火柴棍拼出的整数（若该数非零，则最高位不能是 0），请问能拼成多少种不同的等式。用火柴棍拼数字 0-9 的拼法如图所示：



*【图：火柴棒数字拼法】*

注意：

1. 加号与等号各自需要两根火柴棍
2. 如果$A \neq B$，则$A+B=C$与$B+A=C$视为不同的等式$(A,B,C>=0)$
3. $n$根火柴棍必须全部用上

**习题 5**：妖梦拼木棒（洛谷 P3799）

现有$n(n \leq 100000)$根长度不超过 5000 的木棒，从中选取 4 根组成一个等边三角形，请问有几种选法？

**习题 6**：kkksc03 考前临时抱佛脚（洛谷 P2392）

有四个科目的作业，每个科目有不超过 20 题，解决每道题都需要一定的时间。kkk 可以同时处理同一科目的两道不同的题，求他完成所有题目所需要的时间。

**习题 7**：Perket（洛谷 P2036）

制作一种美食，可能需要用到$N(1 \leq N \leq 10)$种配料，每一种配料都有一个酸度$S_i$和甜度$B_i$。我们需要选择一些配料来烹饪，至少选择一种配料，每种配料选择不超过一次。成品的总酸度是每一种配料的酸度总乘积，总甜度是配料的甜度之和。希望选取配料，以使得酸度和甜度的绝对差最小。数据保证$S_i, B_i > 0$且当加入所有配料时，酸度或者甜度都不会超过$10^9$。

**习题 8**：吃奶酪（洛谷 P1433）

房间里放着$n(n \leq 15)$块奶酪，并且知道坐标。一只小老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在(0,0)点处。（提示：有 70% 的数据范围是$n \leq 11$，因此现阶段只需拿到 70 分即可）