第3章 穷举法

3.1 本章知识结构

本章主要讨论穷举法的概念、算法执行中列举元素的方法、基本优化数据结构和穷举法经典应用示例，其知识结构如图 3.1 所示。

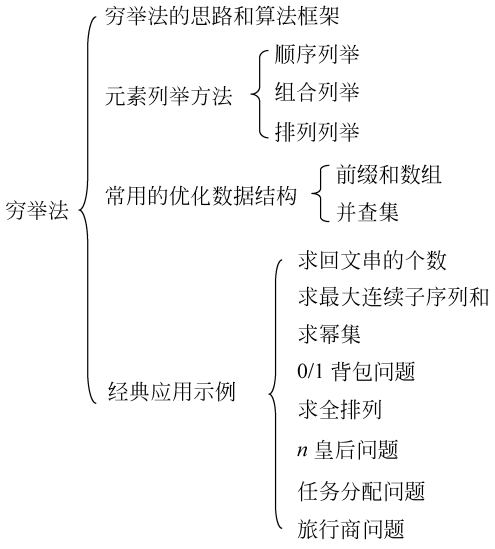

图 3.1 本章知识结构图

3.2 《教程》中的练习题及其参考答案

1. 简述穷举法的基本思想。

答：穷举法的基本思想是根据问题的部分条件确定解的大致范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。若某个情况验证符合题目的全部条件，则为本问题的一个解；若全部情况验证后都不符合题目的全部条件，则本问题无解。

2. 简述穷举法中有哪几种列举方法。

答：穷举法中常用的列举方法如下。

(1) 顺序列举：指问题的解范围内的各种情况很容易与自然数对应甚至就是自然数，可以按自然数的变化顺序去列举。

(2) 组合列举：指问题的解表现为一些元素的组合，可以通过组合列举方式枚举所有的组合情况，通常情况下组合列举是无序的。

(3) 排列列举：指问题的解表现为一组元素的排列，可以通过排列列举方式枚举所有的排列情况，针对不同的问题有些排列列举是无序的，有些是有序的。

3. 举一个例子说明前缀和数组的应用。

答：例如有 m 个询问，某个询问要求 a 到 b 的素数的个数(a、b 均为大于 1 的正整数，

$a \leqslant b$，并且 b 的最大值不超过 10 000)。定义一个数组 $c[10000]$，采用素数筛选法求出 c，其中 $c[i]=1$ 表示整数 i 是素数，$c[i]=0$ 表示整数 i 不是素数。再定义一个前缀和数组 psum，其中 psum[i]表示 2～i 中素数的个数。对应的递推关系如下：

psum[1]=0

psum[2]=c[2]=1

psum[i]=c[2]+c[3]+…+c[$i-1$]+c[i]=psum[$i-1$]+c[i]　　$i>2$

psum[j]=c[2]+c[3]+…+c[$i-1$]+c[i]+c[$i+1$]+…+c[j]

=psum[$i-1$]+c[$i+1$]+…+c[j]　　$j \geqslant i$

因此有 psum[j]−psum[i]=c[$i+1$]+…+c[j]或者 psum[j]−psum[$i-1$]=c[i]+…+c[j]，所以 a 到 b 的素数的个数为 psum[b]−psum[$a-1$]。当一次性求出 psum 数组后，m 个询问花费的时间为 $O(m)$。

4. 举一个例子说明并查集的应用。

答：例如给定一个用邻接矩阵 $\boldsymbol{A}$ 表示的无向图(顶点的编号为 0～$n-1$)，求其中连通分量的个数。采用并查集求解，首先调用 Init(n)初始化并查集，然后遍历 $\boldsymbol{A}$，当 $A[i][j]=1$ 时调用 Union(i,j)合并顶点 i 和 j。最后求其中子集树的个数即为图的连通分量的个数。

5. 考虑下面的算法，用于求数组 a 中相差最小的两个元素的差。请对这个算法做尽可能多的改进。

```
int mindif(int a[],int n) {
    int dmin=INF;
    for (int i=0;i<=n-2;i++) {
        for (int j=i+1;j<=n-1;j++){
            int temp=abs(a[i]-a[j]);
            if (temp<dmin) dmin=temp;
        }
    }
    return dmin;
}
```

答：该算法的时间复杂度为 $O(n^2)$，采用的是最基本的穷举法。改进方法是先对 a 中的元素递增排序，然后依次比较相邻元素的差，求出最小差，对应的优化算法如下：

```
int mindif1(int a[],int n) {
    sort(a,a+n);                                    //递增排序
    int dmin=a[1]-a[0];
    for (int i=2;i<n;i++){
        int tmp=a[i]-a[i-1];
        if (tmp<dmin) dmin=tmp;
    }
    return dmin;
}
```

优化算法的时间主要花费在排序上，算法的时间复杂度为 $O(n\log_2 n)$。

6. 为什么采用穷举法求解 n 皇后问题时列举方式是排列列举？

答：n 皇后问题是在 $n\times n$ 的棋盘中放置互相不冲突的 n 个皇后，假设行、列号是 0～$n-1$，每一行只能放置一个皇后，剩下的问题是确定每一个皇后的列号，全部 n 个皇后的列号恰好是 0～$n-1$ 的一个排列，所以采用穷举法求解时列举方式是排列列举。

7. 给定一个整数数组 $a=(a_0,a_1,\cdots,a_{n-1})$，若 $i<j$ 且 $a_i>a_j$，称$<a_i,a_j>$为一个逆

序对。例如数组(3,1,4,5,2)的逆序对有<3,1>、<3,2>、<4,2>、<5,2>。设计一个算法采用穷举法求 a 中逆序对的个数,即逆序数。

解:采用两重循环直接判断是否为逆序对,算法的时间复杂度为 $O(n^2)$。对应的穷举法算法如下:

```
int revcnt(int a[],int n) {
    int ans = 0;
    for(int i = 0;i < n - 1;i++) {
        for(int j = i + 1;j < n;j++) {
            if(a[i]> a[j]) ans++;
        }
    }
    return ans;
}
```

8. 求最长回文子串(LeetCode5★★)。给定一个字符串 s,设计一个算法求 s 中的最长回文子串,如果有多个最长回文子串,求出其中的任意一个。例如,s="babad",答案为"bab"或者"aba"。要求设计如下成员函数:

```
string longestPalindrome(string s) { }
```

解:利用《教程》中3.3节求回文子串问题的优化算法求解,用 ans 存放 s 中的最长回文子串(初始为空串)。修改其中的 cnt 算法求 ans,对于子串 $s[l..r]$,若 $s[l]=s[r]$,说明 $s[l..r]$为回文,执行 l——和 r++继续向两边扩展。当循环结束后此时的 $s[l+1..r-1]$就是本次中心点扩展得到的最长回文子串,其长度为 $r-l-1$,若该长度大于 ans.size(),则置 ans=s.substr($l+1$,$r-l-1$)。最后返回 ans 即可。对应的算法如下:

```
class Solution {
    int n;
    string ans;
public:
    string longestPalindrome(string s) {
        ans = "";
        n = s.size();
        for(int c = 0;c < n;c++)                    //考虑每个字符位置为回文中心点
            cnt(s,c,c);
        for(int c = 0;c < n - 1;c++)                //考虑每两个字符的中间位置为回文中心点
            cnt(s,c,c + 1);
        return ans;
    }
    void cnt(string &s,int l,int r) {               //求最长回文子串
        while (l >= 0 && r < n && s[l] == s[r]) {
            l -- ; r++;
        }
        if(r - l - 1 > ans.size())
            ans = s.substr(l + 1,r - l - 1);
    }
};
```

上述程序提交后通过,执行用时为20ms,内存消耗为10.8MB。

9. 求解涂棋盘问题。小易有一块 $n\times n$ 的棋盘,棋盘的每一个格子都为黑色或者白色,小易现在要用他喜欢的红色去涂画棋盘。小易会找出棋盘的某一列中拥有相同颜色的最大区域去涂画,帮助小易计算他会涂画多少个棋盘格。

输入格式：输入数据包括 $n+1$ 行，第一行为一个整数 $n(1\leqslant n\leqslant 50)$，即棋盘的大小，接下来的 n 行每行一个字符串，表示第 i 行棋盘的颜色，'W'表示白色，'B'表示黑色。

输出格式：输出小易会涂画的区域大小。

输入样例：

```
3
BWW
BBB
BWB
```

输出样例：

```
3
```

解：采用穷举法求解，统计每一列相同颜色的相邻棋盘格的个数 countj，在 countj 中求最大值。对应的程序如下：

```
#include<iostream>
#include<algorithm>
using namespace std;
#define MAXN 51
int n;
char board[MAXN][MAXN];
int getmaxarea(){                                    //求解算法
    int maxarea=0;
    for(int j=0;j<n;j++){
        int countj=1;
        for(int i=1;i<n;i++){                        //统计第 j 列中相同颜色的相邻棋盘格的个数
            if(board[i][j]==board[i-1][j]) countj++;
            else countj=1;
        }
        maxarea=max(maxarea,countj);
    }
    return maxarea;
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        scanf("%s",board[i]);
    printf("%d\n",getmaxarea());
    return 0;
}
```

10. 电话号码的字母组合(LeetCode17★★)。给定一个仅包含数字 2～9 的长度为$n(0\leqslant n\leqslant 4)$的字符串 digits，设计一个算法求所有能表示的字母组合，答案可以按任意顺序返回。给出数字到字母的映射如图 3.2 所示(与电话的按键相同)，注意 1 不对应任何字母。例如，digits="23"，答案是{"ad","ae","af","bd","be","bf","cd","ce","cf"}。要求设计如下成员函数：

图 3.2 用电话按键表示的数字到字母的映射

```
vector<string> letterCombinations(string digits){ }
```

解：用 unordered_map<char,string>类型的容器 hmap 表

示数字到字母的映射关系，用 vector < string >容器 ps 存放 digits 的所有能表示的字母组合。这里以 digits＝"23"为例进行说明，用 i 遍历 digits，其求解过程如下：

(1) i＝0，digits[0]＝'2'，将 hmap['2']映射的字符串中的每个字符作为一个字符串添加到 ps 中。这里 hmap['2']＝"abc"，则 ps＝{"a","b","c"}。

(2) i＝1，digits[1]＝'3'，置 ps1＝ps，清空 ps。提取 mapstr＝hmap['3']，在 ps1 中每个字符串的末尾添加 mapstr 的每个字符，将结果存放在 ps 中。这里 mapstr＝hmap['3']＝"def"，ps1＝{"a","b","c"}，组合起来得到 ps＝{"ad","ae","af","bd","be","bf","cd","ce","cf"}。

从中可以看出，求解过程就是枚举 digits 的每一个字符 digits[i]，对应的值域为 hmap[digits[i]]，全部组合起来得到结果。对应的代码如下：

```
class Solution {
public:
    vector < string > letterCombinations(string digits) {
        int n = digits.size();
        if(n == 0) return {};
        unordered_map < char,string > hmap = {{'2',"abc"},{'3',"def"},{'4',"ghi"},
                {'5',"jkl"},{'6',"mno"},{'7',"pqrs"},{'8',"tuv"},{'9',"wxyz"}}; //映射表
        vector < string > ps;
        for (int i = 0;i < hmap[digits[0]].size();i++)
            ps.push_back(string(1,hmap[digits[0]][i]));
        for (int i = 1;i < n;i++) {
            vector < string > ps1 = ps;
            ps.clear();
            for (int j = 0;j < ps1.size();j++) {
                string mapstr = hmap[digits[i]];
                for (int k = 0;k < mapstr.size();k++) {
                    string e = ps1[j];
                    e.push_back(mapstr[k]);
                    ps.push_back(e);
                }
            }
        }
        return ps;
    }
};
```

11. 求子数组的和为 k 的个数(LintCode838★)。给定一个整数数组 nums 和一个整数 k，设计一个算法求该数组中连续子数组的和为 k 的总个数。例如，nums＝{2,1,－1,1,2}，k＝3，和为 3 的子数组为(2,1)、(2,－1,1,2)、(2,1,－1,1)和(1,2)，答案为 4。要求设计如下成员函数：

```
int subarraySumEqualsK(vector < int > &nums, int K) { }
```

解：用整数变量 ans 存放答案(初始为 0)。采用直接穷举法，通过两重循环穷举所有的连续子序列，对应的代码如下：

```
class Solution {
public:
    int subarraySumEqualsK(vector < int > &nums, int K) {
        int n = nums.size();
        int ans = 0;
```

```
        for (int i = 0;i < n;i++) {                    //用两重循环穷举所有的连续子序列
            for (int j = i;j < n;j++) {
                int cursum = 0;
                for (int k = i;k < = j;k++)
                    cursum += nums[k];
                if(cursum == K) ans++;
            }
        }
        return ans;
    }
};
```

上述程序会超时(time limit exceeded)。改进的方法是采用前缀和数组 psum 提高时间性能,psum[i]表示 nums[0..i]的元素和,另外定义一个 unordered_map < int,int >类型的哈希表 cntmap 实现前缀和数组的计数,置 cntmap[0]=1。在求出 psum 数组后,用 i 遍历 psum,若当前 psum[i]−k 出现在 cntmap 中,说明找到了和为 k 的连续子数组,其个数为 cntmap[psum[i]−k],将其累计到 ans 中,同时将 cntmap[psum[i]]增 1。最后返回 ans。对应的代码如下:

```
class Solution {
public:
    int subarraySumEqualsK(vector < int > &nums, int K) {
        unordered_map < int,int > cntmap;
        int n = nums.size();
        int psum[n];
        psum[0] = nums[0];
        for (int i = 1;i < n;i++)
            psum[i] = psum[i - 1] + nums[i];
        int ans = 0;
        cntmap[0] = 1;
        for (int i = 0;i < n;i++) {
            ans += cntmap[psum[i] - K];
            cntmap[psum[i]]++;
        }
        return ans;
    }
};
```

12. 给定 n 个城市(城市编号为从 1 到 n),城市和无向道路成本之间的关系为三元组(A,B,C),表示在城市 A 和城市 B 之间有一条路,成本是 C,所有的三元组用 tuple 表示。现在需要从城市 1 开始找到旅行所有城市的最小成本。每个城市只能通过一次,可以假设能够到达所有的城市。例如,n=3,tuple={{1,2,1},{2,3,2},{1,3,3}},答案为 3,对应的最短路径是 1→2→3。

解:本问题与旅行商问题类似,不同之处是不必回到起点。先将边数组 tuple 转换为邻接矩阵 $\boldsymbol{A}$,为了简单,将城市编号由 1～n 改为 0～n−1,这样起点变成了顶点 0,并且不必考虑路径中最后一个顶点到顶点 0 的道路。然后采用《教程》第 3 章中 3.10 节求 TSP 问题的穷举法求出最短路径长度 ans 并返回。对应的穷举法算法如下:

```
const int INF = 0x3f3f3f3f;                            //表示∞
int mincost(int n,vector < vector < int >> &tuple) {
    if(n == 1) return 0;
    vector < vector < int >> A(n,vector < int >(n,INF));   //邻接矩阵
```

```
    for(int i = 0;i < tuple.size();i++) {
        int a = tuple[i][0] - 1;
        int b = tuple[i][1] - 1;
        int w = tuple[i][2];
        A[a][b] = A[b][a] = w;
    }
    vector < int > path;
    for(int i = 1;i < n;i++)                    //初始化 path = {1,2,…,n - 1}
        path.push_back(i);
    int ans = INF;
    do {
        int curlen = 0,u = 0,j = 0;
        while(j < path.size()) {
            int v = path[j];
            curlen += A[u][v];                  //对应一条边< u,v >
            u = v;
            j++;
        }
        ans = min(ans,curlen);
    } while(next_permutation(path.begin(),path.end()));
    return ans;
}
```

13. n 皇后问题Ⅱ(LintCode34★★)。给定一个整数 n($n \leqslant 10$),设计一个算法求 n 皇后问题的解的数量。例如,$n=4$ 时答案为 2,也就是说 4 皇后问题共有两个解。要求设计如下成员函数:

```
int totalNQueens(int n) { }
```

解:利用求 n 皇后问题的全部解的思路,仅改为当找到一个解时不是输出解而是累计解的个数 ans,最后返回 ans 即可。对应的代码如下:

```
class Solution {
public:
    int totalNQueens(int n) {                   //求解算法
        int q[12];
        for(int i = 0;i < n;i++)q[i] = i;       //初始化 q 为 0～n - 1
        int ans = 0;
        do {
            if(isaqueen(n,q))ans++;             //当 q 是一个解时 ans 增 1
        } while(next_permutation(q,q + n));     //取 q 的下一个排列
        return ans;
    }
    bool isaqueen(int n,int q[]) {              //判断 q 是否为 n 皇后问题的一个解
        for(int i = 1;i < n;i++) {
            if(!valid(i,q)) return false;
        }
        return true;
    }
    bool valid(int i,int q[]) {                 //测试(i,q[i])位置是否与前面的皇后不冲突
        if (i == 0) return true;
        int k = 0;
        while (k < i) {                         //k = 0～i - 1 是已放置了皇后的行
            if ((q[k] == q[i]) || (abs(q[k] - q[i]) == abs(k - i)))
                return false;                   //(i,q[i])与皇后 k 有冲突
            k++;
        }
```

```
            return true;
        }
    };
```

上述程序提交后通过，执行用时为 365ms，内存消耗为 5.42MB。

3.3 补充练习题及其参考答案

扫一扫 在线资源

3.3.1 单项选择题及其参考答案

1. 列举所有可能的情况，逐个判断哪些符合问题所要求的条件，从而得到问题的解，这是_____的思路。

A. 解析法　　B. 顺序查找算法　　C. 递归法　　D. 穷举法

2. 穷举法的适用范围是_____。

A. 一切问题　　B. 解的个数极多的问题

C. 解的个数有限且可一一列举的问题　　D. 不适合设计算法的问题

3. 以下关于穷举法的描述中正确的是_____。

A. 穷举法是一种适用于解决小规模问题的方法

B. 可作为产生其他有效算法的基础

C. 可作为其他有效算法的衡量标准

D. 以上都正确

4. 以下关于穷举算法的描述中正确的是_____。

A. 穷举算法的时间复杂度一般都比较高，在求解问题时不可取

B. 穷举算法的时间复杂度与枚举对象的数目有关，减少枚举对象的数目是提高穷举算法效率的重要手段

C. 穷举算法只能用循环实现

D. 穷举算法不能用递归实现

5. 有 100 块砖，100 个人搬，一个男人搬 4 块，一个女人搬 3 块，两个小孩抬一块，要求一次全搬完，问需要男、女、小孩各多少人。该搬砖问题适合采用_____求解。

A. 穷举法　　B. 解析法　　C. 递归法　　D. 排序法

6. 如果一个 4 位数恰好等于它的各位数字的四次方之和，则这个数被称为“玫瑰花数”。例如 1634 就是一个玫瑰花数，即 $1634=1^4+6^4+3^4+4^4$。如果要求出所有的玫瑰花数，下列算法中合适的是_____。

A. 查找法　　B. 解析法　　C. 穷举法　　D. 排序法

7. 一张单据上有一个 5 位数的号码 67??8，其中百位和十位上的数字看不清楚了，但知道该数能够被 78 整除，也能被 67 整除，设计一个算法求出该号码。下列算法中合适的是_____。

A. 穷举法　　B. 解析法　　C. 排序法　　D. 查找法

8. 以下适合采用穷举法求解的是_____。

A. 求两个实数之间所有实数的个数

B. 对一个边数少于 10 的带权有向图求出其中一个顶点到另外一个顶点的所有路径

C. 列出所有的汉字

D. 给定一个三角形的边长求其面积

9. 在采用穷举法求解以下问题时属于组合列举的是_____。

A. n 皇后问题

B. 查找某个班最高分的学生的姓名

C. 在 n 个学生分数中选择总分数为 k 的 m 个分数

D. 在一个递增有序的数组中查找一个元素

10. 在采用穷举法求解以下问题时属于排列列举的是_____。

A. n 皇后问题

B. 查找某个班最高分的学生的姓名

C. 在 n 个学生分数中选择总分数为 k 的 m 个分数

D. 在一个递增有序的数组中查找一个元素

扫一扫

在线资源

3.3.2 问答题及其参考答案

1. 某城市的电话号码共 8 位，采用穷举法找到其中某个电话号码最多需要比较多少次？

2. 有人说穷举算法的时间性能差，没有任何意义，你如何理解？

3. 有人说穷举算法只能采用迭代实现，不能采用递归实现，你如何理解？

4. 求 1～1000 中所有能够被 5 和 7 整除的奇数，给出采用穷举法求解时尽可能优化的枚举值域和约束条件。

5. 鸡兔同笼问题是一个笼子里有鸡和兔若干只，数一数，共有 a 个头、b 条腿。假设鸡和兔各有 x 和 y 只，给出采用穷举法求解时尽可能优化的枚举值域和约束条件。

6. 为什么采用穷举法求解 0/1 背包问题时列举方式是组合列举？

7. 假设有一个含 n 个整数的数组 a 以及一个整数 t，现在在 a 中找出若干和等于 t 的元素，问采用穷举法求解时列举方式是什么？

3.3.3 算法设计题及其参考答案

1. 有 1～4 共 4 个数字，设计一个算法求能组成多少个互不相同而且无重复数字的三位数。

解：用 abc 表示互不相同而且无重复数字的三位数，其中 a、b 和 c 的取值范围均为 1～4，用 ans 存放这样的三位数的个数(初始为 0)。显然满足要求的条件是 a!＝b && a!＝c && b!＝c。对应的穷举算法如下：

```
int count() {
  int ans = 0;
    for(int a = 1;a < 5;a++) {
        for(int b = 1;b < 5;b++) {
            for(int c = 1;c < 5;c++) {
```

```
                if(a!= b && a!= c && b!= c) ans++;
            }
        }
    }
    return ans;
}
```

2. 所谓“水仙花数”是指一个三位数，其各位数字的立方和等于该数本身。例如 153 是一个水仙花数，因为 $153=1^3+5^3+3^3$。设计一个算法求所有水仙花数。

解：用 abc 表示一个三位数，a 的取值范围是 1～9，b 和 c 的取值范围是 0～9，求所有满足 $a\times100+b\times10+c=a\times a\times a+b\times b\times b+c\times c\times c$ 的水仙花数。对应的穷举算法如下：

```
void flower() {
    for(int a = 1;a < 10;a++) {
        for(int b = 0;b < 10;b++) {
            for(int c = 0;c < 10;c++) {
                int s = a * 100 + b * 10 + c;
                if(a * a * a + b * b * b + c * c * c == s)
                    printf(" % d\n",s);
            }
        }
    }
}
```

3. 有一个三位数，个位数字比百位数字大，而百位数字又比十位数字大，并且各位数字之和等于各位数字相乘之积，求此三位数。

解：用 abc 表示该三位数，依题意 $c>a$、$a>b$，即 $c>a>b$，b 的取值范围是 0～9，同时满足 $a+b+c=a\times b\times c$。对应的穷举算法如下：

```
void threed() {
    for(int b = 0;b < 10;b++) {
        for(int a = b + 1;a < 10;a++) {
            for(int c = a + 1;c < 10;c++) {
                if(a + b + c == a * b * c)
                    printf("abc = % d % d % d\n",a,b,c);
            }
        }
    }
}
```

4. 有 $n(n\geqslant4)$个正整数，存放在数组 a 中，设计一个算法从中选出 3 个正整数组成周长最长的三角形，输出该三角形的周长，若无法组成三角形则输出 0。

解：采用直接穷举思路，用 i、j、k 三重循环，让 $i<j<k$ 以避免正整数被重复选中。设选中的 3 个正整数 $a[i]$、$a[j]$、$a[k]$之和为 curlen，其中最大正整数为 ma，能组成三角形的条件是两边之和大于第三边，即 ma<curlen-ma。对应的穷举算法如下：

```
int maxlen(int a[],int n) {
    int ans = 0;
    for (int i = 0;i < n;i++) {
        for (int j = i + 1;j < n;j++) {
            for (int k = j + 1;k < n;k++) {
                int curlen = a[i] + a[j] + a[k];
                int ma = max3(a[i],a[j],a[k]);
```

```
                if (ma < curlen - ma)                    //a[i]、a[j]、a[k]能组成一个三角形
                    ans = max(ans,curlen);               //求最长的周长
            }
        }
    }
    return ans;
}
```

5. 有一堆硬币，面值只有 1 分、2 分和 5 分 3 种，其中有 57 枚面值不是 5 分，有 77 枚面值不是 2 分，有 72 枚面值不是 1 分，设计一个算法求 1 分、2 分和 5 分的硬币各有多少，输出全部答案。

解：设 1 分、2 分、5 分硬币的个数分别是 x、y 和 z，则 $x+y=57$，$x+z=77$，$y+z=72$，可以推出 $x\leqslant 57$，$y\leqslant 57$，$z\leqslant 72$。对应的穷举算法如下：

```
void coins() {
    for (int x = 0;x < = 57;x++) {
        for (int y = 0;y < = 57;y++) {
            for (int z = 0;z < = 72;z++) {
                if (x + y == 57 && x + z == 77 && y + z == 72)
                    printf("x = %d,y = %d,z = %d\n",x,y,z);
            }
        }
    }
}
```

6. 对于给定的整数 $n(n>1)$，设计一个穷举算法求 $1!+2!+\cdots+n!$，并改进该算法提高时间性能。

解：直接采用穷举算法如下。

```
long fact(int n) {                                   //求 n!
    long fn = 1;
    for (int i = 2;i < = n;i++) fn = fn * i;
    return fn;
}
long fsum1(int n) {                                  //求 1! + 2! + … + n!
    long ans = 0;
    for (int i = 1;i < = n;i++)
        ans += fact(i);
    return ans;
}
```

实际上，fact(n)=fact($n-1$)×n，fact(1)=1，在求 fact(n)时可以利用 fact($n-1$)的结果。改进后的算法如下：

```
long fsum2(int n) {                                  //求 1! + 2! + … + n!
    long ans = 0;
    long fn = 1;
    for (int i = 1;i < = n;i++) {
        fn = fn * i;
        ans += fn;
    }
    return ans;
}
```

7. 某年级的同学集体去公园划船，如果每只船坐 10 个人，那么多出两个座位；如果每只船多坐两个人，那么可少租一只船。设计一个算法用穷举法求该年级的最多人数。

解：设该年级的人数为 x，租船数为 y。因为每只船坐 10 个人多出两个座位，则 $x=10\times y-2$；因为每只船多坐两个人(即 12 人)时可少租一只船(没有说座位是否恰好全部占满)，有 $x+z=12\times(y-1)$，z 表示此时空出的座位，显然 $z<12$。让 y 从 1 到 100(实际上 y 取更大范围的结果是相同的)、z 从 0 到 11 枚举，求出最大的 x 即可。对应的穷举算法如下：

```
int maxstuds() {
    int x;
    for (int y = 1;y <= 100;y++) {
        for (int z = 0;z < 12;z++) {
            if (10 * y - 2 == 12 * (y - 1) - z) x = 10 * y - 2;
        }
    }
    return x;
}
```

8. 求解完数问题。如果一个大于 1 的正整数的所有因子之和等于它本身，则称这个数是完数，例如 6、28 都是完数，因为 $6=1+2+3$，$28=1+2+4+7+14$。设计一个算法求两个正整数之间完数的个数。

输入格式：输入数据包含多行，第一行是一个正整数 n，表示测试实例的个数，然后是 n 个测试实例，每个实例占一行，由两个正整数 num1 和 num2 组成($1<$num1，num2$<10\,000$)。

输出格式：对于每组测试数据，输出 num1 和 num2 之间(包括 num1 和 num2)存在的完数的个数。

输入样例：

```
2
2 5
5 7
```

输出样例：

```
0
1
```

解：对于输入的 n，循环 n 次，每次输入 num1 和 num2，调用 count(num1，num2)求这两个数之间完数的个数，该算法枚举 num1 和 num2 之间的数(包括这两个数)，判断是不是完数，如果是完数，则累计到 ans 中，最后返回 ans。对应的程序如下：

```
#include <iostream>
#include <algorithm>
using namespace std;
int n;
int count(int num1,int num2) {                    //求 num1 和 num2 之间完数的个数
    int ans = 0;
    for(int j = num1;j <= num2;j++) {             //执行 num2 - num1 + 1 次循环
        int sum = 0;
        for(int k = 1;k < j;k++) {
            if(j % k == 0) sum += k;              //累计 j 的因子
        }
        if (sum == j) ans++;                      //如果是完数,累计个数
    }
```

```
    return ans;
}
int main(){
    int num1,num2;
    scanf("%d",&n);
    for(int i=0;i<n;i++) {                          //执行 n 次循环
        scanf("%d%d",&num1,&num2);                  //输入两个整数
        printf("%d\n",count(num1,num2));
    }
    return 0;
}
```

9. 三角形最大面积(LintCode1005★)。给定二维平面上的 n(3≤n≤50)个点 points(−50≤points[i][j]≤50),设计一个算法求由其中 3 个点形成的三角形的最大面积。例如,points={{0,0},{0,1},{1,0},{0,2},{2,0}},答案是 2。要求设计如下成员函数:

```
double largestTriangleArea(vector<vector<int>> &points) {}
```

解:枚举 points 中的任意 3 个点 x、y、z,求出其面积 s,通过比较求最大面积 ans(当 x、y 和 z 中存在相同点时面积为 0,所以不必判重),最后返回 ans 即可。对应的代码如下:

```
class Solution {
public:
    double largestTriangleArea(vector<vector<int>> &points) {
        int n=points.size();
        double ans=0;
        for(auto x:points) {
            for(auto y:points) {
                for(auto z:points) {
                    double s=0.5*(x[0]*y[1]+y[0]*z[1]+z[0]*x[1]-x[0]*z[1]-y
[0]*x[1]-z[0]*y[1]);
                    ans=max(ans,s);
                }
            }
        }
        return ans;
    }
};
```

10. 图是否为树(LintCode178★★)。给出 n 个顶点,编号为 0～n−1,并且给出一个无向边的列表(给出每条边的两个顶点),设计一个算法判断这个无向图是否为一棵树。假设列表中没有重复的边。例如,n=5,edges={{0,1},{0,2},{0,3},{1,4}},答案是 true。要求设计如下成员函数:

```
bool validTree(int n, vector<vector<int>> &edges) { }
```

解:采用并查集求解,遍历 edges 的每一条边(a,b),求出对应子集树的编号 ra 和 rb,若 ra=rb,说明之前 a 和 b 已经连通,再加上边(a,b)一定出现回路,则该图不是树,返回 false,否则合并 ra 和 rb。当 edges 遍历完毕,说明图中没有回路,再求出子集树的棵数 cnt,若只有一棵子集树,说明该图是树,返回 true,否则说明该图不是树,返回 false。对应的算法如下:

```
class Solution {
public:
    int n;
```

```
    vector < int > parent;                    //并查集存储结构
    vector < int > rnk;                       //存储结点的秩(近似于高度)
    bool validTree(int n,vector < vector < int >> &edges) {
        this -> n = n;
        int m = edges.size();
        parent.resize(n);
        rnk.resize(n);
        Init();
        for(int i = 0;i < m;i++) {
            int a = edges[i][0];
            int b = edges[i][1];
            int ra = Find(a);
            int rb = Find(b);
            if(ra == rb) return false;
            else Union(ra,rb);
        }
        int cnt = 0;
        for(int i = 0;i < n;i++) {
            if(parent[i] == i) cnt++;
        }
        return cnt == 1;
    }
    void Init() {                             //并查集的初始化
        for (int i = 0;i < n;i++) {
            parent[i] = i;
            rnk[i] = 0;
        }
    }
    int Find(int x) {                         //递归算法:在并查集中查找 x 结点的根结点
        if (x!= parent[x])
            parent[x] = Find(parent[x]);      //路径压缩
        return parent[x];
    }
    void Union(int rx,int ry) {               //两个根合并
        if (rx == ry) return;                 //x 和 y 属于同一棵树的情况
        if (rnk[rx]< rnk[ry])
            parent[rx] = ry;                  //rx 结点作为 ry 的孩子
        else {
            if (rnk[rx] == rnk[ry])           //秩相同,合并后 rx 的秩增 1
                rnk[rx]++;
            parent[ry] = rx;                  //ry 结点作为 rx 的孩子
        }
    }
};
```

11. 求解好多鱼问题。牛牛有一个鱼缸,鱼缸里面已经有 n 条鱼,每条鱼的大小为 fishSize[i](1≤i≤n,均为正整数),牛牛现在想把新捕捉的鱼放入鱼缸。鱼缸内存在着大鱼吃小鱼的定律,经过观察,牛牛发现若一条鱼 A 为另外一条鱼 B 的大小的 2～10 倍(含 2 倍和 10 倍),则鱼 A 会吃掉鱼 B。牛牛需要保证放进去的鱼是安全的,不会被其他鱼吃掉,并且放进去的鱼也不能吃掉其他鱼。

鱼缸里面存在的鱼已经相处了很久,不必考虑它们互相捕食。现在知道新放入鱼的大小范围[minSize,maxSize](考虑鱼的大小都是整数表示),牛牛想知道有多少种大小的鱼可以放入这个鱼缸。

输入格式：输入数据包括3行，第一行为新放入鱼的大小范围[minSize，maxSize](1≤minSize，maxSize≤1000)，以空格分隔，第二行为鱼缸里面已经有的鱼的数量 n(1≤n≤50)，第三行为已经有的鱼的大小 fishSize[i](1≤fishSize[i]≤1000)，以空格分隔。

输出格式：输出有多少种大小的鱼可以放入鱼缸。考虑鱼的大小都是整数表示。

输入样例：

```
1 12
1
1
```

输出样例：

```
3
```

解：直接采用穷举法求解。设有 ans 种大小的鱼可以放入鱼缸(初始为0)。i 从 minSize 到 maxSize 循环枚举，如果 i 满足题目要求，ans＋＋。最后输出 ans。对应的程序如下：

```
#include<iostream>
using namespace std;
#define MAX 51
int fishSize[MAX];
int n;
int minSize,maxSize;
int ans=0;
void solve(){                                   //求解算法
    bool flag;
    for(int i=minSize;i<=maxSize;i++){
        flag=true;
        for(int j=0;j<n;j++){
            if((i>=fishSize[j]*2 && i<=fishSize[j]*10)||(fishSize[j]>=i*2 &&
fishSize[j]<=i*10)){
                flag=false;                     //不能放入
                break;
            }
        }
        if(flag) ans++;                         //能够放入
    }
}
int main(){
    scanf("%d%d",&minSize,&maxSize);
    scanf("%d",&n);
    for(int i=0; i<n; ++i)
        scanf("%d",&fishSize[i]);
    solve();
    printf("%d\n",ans);
    return 0;
}
```

12. 最大子数组之和为 k(LintCode911★★)。给一个数组 nums 和目标值 k，设计一个算法找到数组中最长的子数组，使其中的元素和为 k，如果没有则返回0。例如，nums＝{1，－1，5，－2，3}，k＝3，其中子数组(1，－1，5，－2)的和为3，且长度最大，答案为4。要求设计如下成员函数：

```
int maxSubArrayLen(vector<int>&nums, int k){ }
```

解：用整数变量 ans 存放答案(初始为 0)。为了提高时间性能，采用前缀和数组 psum 和 unordered_map<int,int>类型的哈希容器 hmap，psum[i]表示 nums[0..$i-1$]的元素和，即前 i 个元素的和，hmap 存放 psum[i]+k 在 hmap 中第一次出现的序号 i，即最小序号。

用 i 遍历 psum，若在 hmap 中找到 psum[i]，说明存在一个元素和为 k 的子数组，其长度为 i-hmap[psum[i]](该子数组是 nums 的前 i 个元素除去前 hmap[psum[i]]个元素后剩下的元素)，通过比较将最大长度存放到 ans 中，若 psum[i]+k 在 hmap 中没有出现，置 hmap[psum[i]+k]=i。最后返回 ans。对应的代码如下：

```
class Solution {
public:
    int maxSubArrayLen(vector<int> &nums, int k) {
        unordered_map<int,int> hmap;
        int n = nums.size();
        int psum[n + 1];
        psum[0] = 0;
        hmap[k] = 0;
        int ans = 0;
        for(int i = 1;i <= n;i++){
            psum[i] = psum[i - 1] + nums[i - 1];
            if(hmap.find(psum[i])!= hmap.end()){
                ans = max(ans,i - hmap[psum[i]]);
            }
            if(hmap.find(psum[i] + k) == hmap.end()) {
                hmap[psum[i] + k] = i;
            }
        }
        return ans;
    }
};
```

上述程序提交后通过，执行用时为 61ms，内存消耗为 5.59MB。实际上可以将前缀和数组 psum 改为单个变量，以节省空间，对应的代码如下：

```
class Solution {
public:
    int maxSubArrayLen(vector<int> &nums, int k) {
        unordered_map<int,int> hmap;
        int n = nums.size();
        int psum = 0;
        hmap[k] = 0;
        int ans = 0;
        for(int i = 1;i <= n;i++){
            psum += nums[i - 1];
            if(hmap.find(psum)!= hmap.end()) {
                ans = max(ans,i - hmap[psum]);
            }
            if(hmap.find(psum + k) == hmap.end()) {
                hmap[psum + k] = i;
            }
        }
        return ans;
    }
};
```

上述程序提交后通过，执行用时为 61ms，内存消耗为 3.61MB。

13. 集合的合并(LintCode1396★★)。有一个由 n($n\leqslant1000$)个集合组成的 list，如果两个集合有相同的元素，将它们合并，设计一个算法返回最后还剩下几个集合。例如，list＝{{1,2,3},{3,9,7},{4,5,10}}，答案是 2，合并后剩下的两个集合是{1,2,3,9,7}和{4,5,10}。要求设计如下成员函数：

```
int setUnion(vector < vector < int >> &sets) { }
```

解：采用并查集进行优化。n 个集合的编号为 0～$n-1$，用 unordered_map < int，vector < int >>类型的哈希映射 hmap 记录每个元素所属的集合编号列表，然后遍历每个元素的所有集合编号列表，将列表中的所有集合编号进行合并，最后求出子集树的个数 ans 并返回。

例如，list＝{{1,2,3},{3,9,7},{4,5,10}}，3 个集合的编号为 0～2，遍历后求出每个元素的集合编号列表如下：

```
1:{0}
2:{0}
3:{0,1}
4:{2}
5:{2}
7:{1}
9:{1}
10:{2}
```

其中只有集合列表{0,1}的长度大于 1，将 0 和 1 合并。也就是说，3 个集合对应 U＝{0,1,2}，通过上述操作得到关系 R＝{(0,1)}，处理 R 后得到等价类{(0,1),(2)}，所以答案为 2。对应的代码如下：

```
class Solution {
    int n;
    vector < int > parent;                    //并查集存储结构
    vector < int > rnk;                       //存储结点的秩(近似于高度)
public:
    int setUnion(vector < vector < int >> &sets) {
        n = sets.size();
        parent.resize(n);
        rnk.resize(n);
        unordered_map < int,vector < int >> hmap;
        Init();
        for(int i = 0;i < n;i++) {
            for(int j = 0;j < sets[i].size();j++)
                hmap[sets[i][j]].push_back(i);
        }
        for(auto it = hmap.begin();it!= hmap.end();it++) {
            vector < int > tmp = it -> second;
            if(tmp.size()> 1) {
                for(int i = 0;i < tmp.size() - 1;i++)
                    Union(tmp[i],tmp[i + 1]);
            }
        }
        int ans = 0;
        for(int i = 0;i < n;i++) {
            if(parent[i] == i) ans++;
```

```
        }
        return ans;
    }
    void Init() {                          //并查集的初始化
        for (int i = 0;i < n;i++) {
            parent[i] = i;
            rnk[i] = 0;
        }
    }
    int Find(int x) {                      //递归算法:在并查集中查找 x 结点的根结点
        if (x!= parent[x])
            parent[x] = Find(parent[x]);   //路径压缩
        return parent[x];
    }
    void Union(int x,int y) {              //并查集中 x 和 y 的两个集合的合并
        int rx = Find(x);
        int ry = Find(y);
        if (rx == ry) return;              //x 和 y 属于同一棵树的情况
        if (rnk[rx]< rnk[ry])
            parent[rx] = ry;               //rx 结点作为 ry 的孩子
        else {
            if (rnk[rx] == rnk[ry])        //秩相同,合并后 rx 的秩增 1
                rnk[rx]++;
            parent[ry] = rx;               //ry 结点作为 rx 的孩子
        }
    }
};
```

上述程序提交后通过,执行用时为 163ms,内存消耗为 5.46MB。

14. 容器储存的最大水量(LeetCode11★★)。给定一个长度为 n($2 \leqslant n \leqslant 100\,000$) 的整数数组 height,表示有 n 条垂线,第 i 条线的两个端点是(i,0)和(i,height[i])。设计一个算法找出其中的两条线使得它们与 X 轴共同构成的容器可以容纳最多的水,返回容器可以储存的最大水量。例如,height=$\{1,8,6,2,5,4,8,3,7\}$,答案是 49,对应的容器是由 height[1]和 height[8]之间的垂线构成的。要求设计如下成员函数:

```
int maxArea(vector < int > & height) { }
```

解:采用双指针方法,i 和 j 分别指向 height 的两个端点,height[i]和 height[j]两条垂线构成的容器的容水量 $S_{i,j}=\min(\text{height}[i],\text{height}[j])\times(j-i)$,若 height[$i$]<height[$j$],显然有 $S_{i,j-1} \leqslant S_{i,j}$ 成立,所以执行 $i++$,否则有 $S_{i+1,j} \leqslant S_{i,j}$ 成立,所以执行 $j--$。以此类推,直到 $i=j$,通过比较每次求出的容水量 S 得到最大值 ans,最后返回 ans 即可。对应的代码如下:

```
class Solution {
public:
    int maxArea(vector < int > & height) {
        int n = height.size();
        int i = 0,j = n - 1;
        int ans = 0;
        while(i < j) {
            int curs = min(height[i],height[j]) * (j - i);
            if(height[i]< height[j]) i++;
            else j-- ;
            ans = max(ans,curs);
```

```
        }
        return ans;
    }
};
```

上述程序提交后通过，执行用时为 80ms，内存消耗为 57.6MB。

15. 最长山脉子数组的长度(LeetCode845★★)。把一个长度为 n 的符合下列属性的数组 arr 称为山脉数组：

(1) $n \geqslant 3$。

(2) 存在下标 $i(0<i<n-1)$，满足 arr[0]<arr[1]<…<arr[$i-1$]<arr[i]并且 arr[i]>arr[$i+1$]>…> arr[$n-1$]。

给出一个整数数组 arr，设计一个算法求最长山脉子数组的长度，如果不存在山脉子数组则返回 0。例如，arr={2,1,4,7,3,2,5}，答案是 5，最长山脉子数组是{1,4,7,3,2}。要求设计如下成员函数：

```
int longestMountain(vector<int>& arr) { }
```

解法 1：采用双指针 i、j 枚举山脉子数组。置 $i=0$，置 $j=i+1$，若 arr[i]<arr[$i+1$]，先找左侧山脉，接着找右侧山脉，得到一个山脉子数组 arr[$i..j$]，其长度为 $j-i+1$。再置 $i=j$ 继续。在所有的山脉子数组中进行比较求最大长度 ans，最后返回 ans 即可。对应的代码如下：

```
class Solution {
public:
    int longestMountain(vector<int>& arr) {
        int n = arr.size();
        int ans = 0, i = 0;
        while(i + 2 < n) {
            int j = i + 1;
            if(arr[i]< arr[i + 1]) {
                while (j + 1 < n && arr[j]< arr[j + 1]) j++;
                if (j < n - 1 && arr[j]> arr[j + 1]) {
                    while (j + 1 < n && arr[j]> arr[j + 1]) j++;
                    ans = max(ans, j - i + 1);
                }
                else j++;
            }
            i = j;
        }
        return ans;
    }
};
```

上述程序提交后通过，执行用时为 24ms，内存消耗为 18.1MB。

解法 2：采用类似前缀和数组的思路，设置 left 和 right 两个一维数组，left[i]表示 arr[i]左侧山脉的长度，right[i]表示 arr[i]右侧山脉的长度。枚举 arr[i]，其作为山峰的山脉子数组长度为 left[i]+right[i]+1，通过比较求最大长度 ans，最后返回 ans 即可。对应的代码如下：

```
class Solution {
public:
    int longestMountain(vector<int>& arr) {
```

```
        int n = arr.size();
        if (n == 0) return 0;
        vector < int > left(n);
        for (int i = 1;i < n;i++) {
            if (arr[i - 1]< arr[i]) left[i] = left[i - 1] + 1;
            else left[i] = 0;
        }
        vector < int > right(n);
        for (int j = n - 2;j > = 0;j -- ) {
            if(arr[j + 1]< arr[j]) right[j] = right[j + 1] + 1;
            else right[j] = 0;
        }
        int ans = 0;
        for (int i = 0;i < n;i++) {
            if (left[i]> 0 && right[i]> 0)
                ans = max(ans,left[i] + right[i] + 1);
        }
        return ans;
    }
};
```

上述程序提交后通过，执行用时为 12ms，内存消耗为 19.2MB。