# 疑难问题

## 等式全排序

输入：小明手上牌的数字， 桌上牌的个数，桌上牌的数字，目标数字

输出：小明手上牌的数字与桌上牌的数字的四则运算组合（结果为目标数字）

如：小明手上牌 为 5 ，桌上数字有 13 6 1 3 9 4 2 2，目标数字为13

所得组合为：

5 / 1 / 3 * 9 – 4 + 2
5 / 1 * 3 – 9 * 4 + 2 / 2
5 / 1 * 3 + 9 + 4 – 2 / 2
5 / 1 – 3 * 9 / 4 + 2 * 2
5 / 1 – 3 * 9 + 4 / 2 + 2
5 / 1 – 3 + 9 + 4 – 2

```cpp
#include <vector>
#include <cstdio>
#include <iostream>
#define N 4
using namespace std;
int n[100];
vector<int> num_v, renum;
vector<char> char_v, rechar;
int max_len = 0;
double re;
char oper[N] = { '/', '*', '-', '+'};
void seq(int m,int k, int len, double sum) {//k为利用的数字的位数
    if (sum == re ) {//如果当前计算结果sum=re要求的结果并且第一次满足，则跳出递归，保存结果
        renum = num_v;          //数字结果存入renum
        rechar = char_v;        //字符结果存入rechar
        cout << m <<" ";
        for (int i = 0; i < rechar.size() ; ++i) {
            if( i < renum.size()){
                cout << rechar[i] << " ";
            }
            cout << renum[i] << " ";
        }
        cout << endl;
        return;
    }
    if (k >= len) return;       //k大于总长度时，退出递归，表示找不到正确的结果
    num_v.push_back(n[k]);      //将第k位数字加入num_v
    for (int i = 0; i < N; i++) {//对字符进行选择，先往符号数组加入，再递归调用
        char op = oper[i];
        if (op == '+') {
            char_v.push_back('+');
            seq(m,k + 1, len, sum + n[k]);
        }
        else if (op == '*') {
            char_v.push_back('*');
            seq(m,k + 1, len, sum * n[k]);

```

```
        }
        else if (op == '-') {
            char_v.push_back('-');
            seq(m,k + 1, len, sum - n[k]);
        }
        else if (op == '/') {
            char_v.push_back('/');
            seq(m,k + 1, len, sum / n[k]);
        }
        char_v.pop_back();//都试过却不可以得出正确结果则把符号弹出
    }
    num_v.pop_back();
}
int main() {
    int len;
    double m;//m 小明手上牌的数字， re 运算结果的要求 ，len 桌上牌的个数
    scanf("%lf %lf %d", &m, &re, &len);
    for (int i = 0; i < len; i++)// 桌上牌的数字
        scanf("%d", &n[i]);
    int k = 0;
    double sum = m;//sum为当前的计算结果
    seq(m, k, len, sum);
    return 0;
}
/*
 本程序利用回溯法
 输入5 13 6 1 3 9 4 2 2
 输出
 5 / 1 / 3 * 9 - 4 + 2
 5 / 1 * 3 - 9 * 4 + 2 / 2
 5 / 1 * 3 + 9 + 4 - 2 / 2
 5 / 1 - 3 * 9 / 4 + 2 * 2
 5 / 1 - 3 * 9 + 4 / 2 + 2
 5 / 1 - 3 + 9 + 4 - 2
 等等全部从5（小明的牌） 开始 尽可能到后面的结果
 拓展一下：
    ① 将包括所有能成立的等式，数字只要是 牌的数字（即 5 13 6 1 3 9 4 2 2里面任意选择多个
    可以相连的数字 也可以不相连） 组成所有满足结果的等式应该怎么写
    ② 我预想用一个结构体来处理写出 所有满足结果的等式中最长最短的所有等式 ，思想就是：
 struct myEqu{
    string equ;//存储等式
    int k;     //数字的个数
    bool operator <(const myEqu &a)const{
        return k < a.k;
 }
    bool operator >(const myEqu &a)const{
        return k > a.k;
 }
 }myEqu[100]
    先将所有等式存储在myEqu中，然后分别sort()排序一下，这样就排好最长和最短等式了，最后输出即可
 */

```

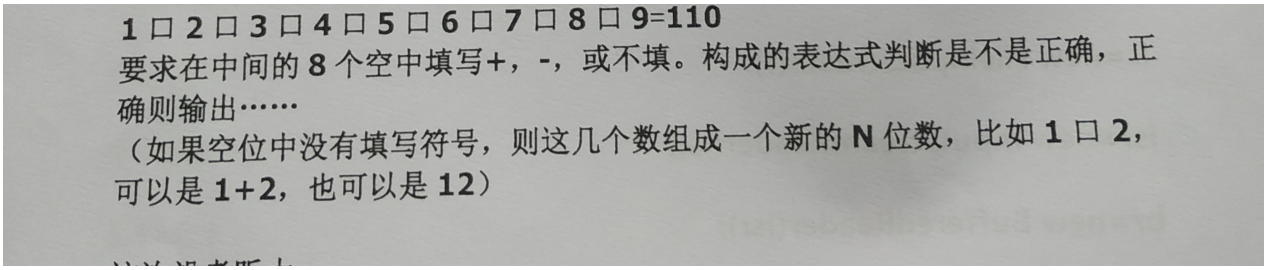
```
#include <iostream>
#include <vector>
#include <cstdio>
#include <iostream>
#define N 4
using namespace std;
int n[100];
vector<int> num_v, renum;
```

```cpp
vector<char> char_v, rechar;
int max_len = 0;
double re;
char oper[N] = { '/', '*', '-', '+' };
void cal(int k,int m,int sum,int len);
void seq(int m, int k, int len, double sum) {
    //k为利用的数字的位数
    if (sum == re) {//如果当前计算结果sum=re要求的结果，则跳出递归，保存结果
        renum = num_v;          //数字结果存入renum
        rechar = char_v;        //字符结果存入rechar
        cout << m << " ";
        for (int i = 0; i < rechar.size(); ++i) {//输出结果
            if (i < renum.size()) {
                cout << rechar[i] << " ";
            }
            cout << renum[i] << " ";
        }
        cout << endl;
        if (k >= len) return;//如果k大于所给数字的总长度，则跳出
       // cal(k, m, sum, len);//如果还有数字没有使用到，则计算等于re 的剩下数字构成的表达式
        return;
    }
    if (k >= len) return; //k大于总长度时，退出递归，表示找不到正确的结果
    seq(m, k + 1, len, sum);//当还有数字需要利用，并且结果不满足条件的时候，继续递归
    cal(k,m,sum,len);
}
void cal(int k,int m,int sum,int len){
    //第k位数字开始，首元素是m，当前算式结果是sum，当前数字数组总长度为len
    num_v.push_back(n[k]);      //将第k位数字加入num_v
    for (int i = 0; i < N; i++) {//对字符进行选择，先往符号数组加入，再递归调用
        char op = oper[i];
        if (op == '+') {
            char_v.push_back('+');
            seq(m, k + 1, len, sum + n[k]);
        }
        else if (op == '*') {
            char_v.push_back('*');
            seq(m, k + 1, len, sum * n[k]);
        }
        else if (op == '-') {
            char_v.push_back('-');
            seq(m, k + 1, len, sum - n[k]);
        }
        else if (op == '/') {
            char_v.push_back('/');
            seq(m, k + 1, len, sum / n[k]);
        }
        char_v.pop_back();//都试过却不可以得出正确结果则把符号弹出
    }
    num_v.pop_back();
}
int main() {
    int len;
    double m;//m 小明手上牌的数字，  re 运算结果的要求  ，len  桌上牌的个数
    scanf("%lf %lf %d", &m, &re, &len);
    for (int i = 0; i < len; i++)// 桌上牌的数字
        scanf("%d", &n[i]);
    int k = 0;//k为从桌上的第k张牌开始
    double sum = m;//sum为当前的计算结果
    seq(m, k, len, sum);
    return 0;
}

```

```cpp
#include <iostream>
#include <vector>
#include <cstdio>
#include <iostream>
#define N 4
using namespace std;
int n[100];
vector<int> num_v, renum;
vector<char> char_v, rechar;
int max_len = 0;
double re;
char oper[N] = { '/', '*', '-', '+' };
void seq(int m, int k, int len, double sum) {
    //k为利用的数字的位数
    if (sum == re) {//如果当前计算结果sum=re要求的结果并且第一次满足，则跳出递归，保存结果
        renum = num_v;          //数字结果存入renum
        rechar = char_v;        //字符结果存入rechar
        cout << m << " ";
        for (int i = 0; i < rechar.size(); ++i) {
            if (i < renum.size()) {
                cout << rechar[i] << " ";
            }
            cout << renum[i] << " ";
        }
        cout << endl;
        if (k >= len) return;
        num_v.push_back(n[k]);      //将第k位数字加入num_v
        for (int i = 0; i < N; i++) {//对字符进行选择，先往符号数组加入，再递归调用
            char op = oper[i];
            if (op == '+') {
                char_v.push_back('+');
                seq(m, k + 1, len, sum + n[k]);
            }
            else if (op == '*') {
                char_v.push_back('*');
                seq(m, k + 1, len, sum * n[k]);

            }
            else if (op == '-') {
                char_v.push_back('-');
                seq(m, k + 1, len, sum - n[k]);
            }
            else if (op == '/') {
                char_v.push_back('/');
                seq(m, k + 1, len, sum / n[k]);
            }
            char_v.pop_back();//都试过却不可以得出正确结果则把符号弹出
        }
        num_v.pop_back();
        return;
    }
    if (k >= len) return;       //k大于总长度时，退出递归，表示找不到正确的结果

    seq(m, k + 1, len, sum);

    num_v.push_back(n[k]);      //将第k位数字加入num_v
    for (int i = 0; i < N; i++) {//对字符进行选择，先往符号数组加入，再递归调用
        char op = oper[i];
        if (op == '+') {
            char_v.push_back('+');
```

```
            seq(m, k + 1, len, sum + n[k]);
        }
        else if (op == '*') {
            char_v.push_back('*');
            seq(m, k + 1, len, sum * n[k]);

        }
        else if (op == '-') {
            char_v.push_back('-');
            seq(m, k + 1, len, sum - n[k]);
        }
        else if (op == '/') {
            char_v.push_back('/');
            seq(m, k + 1, len, sum / n[k]);
        }
        char_v.pop_back();//都试过却不可以得出正确结果则把符号弹出
    }
    num_v.pop_back();
}
int main() {
    int len;
    double m;//m 小明手上牌的数字， re 运算结果的要求 ，len 桌上牌的个数
    scanf("%lf %lf %d", &m, &re, &len);
    for (int i = 0; i < len; i++)// 桌上牌的数字
        scanf("%d", &n[i]);
    int k = 0;
    double sum = m;//sum为当前的计算结果
    seq(m, k, len, sum);
    return 0;
}

```

## 1 2 3 4 5 6 7 8 9 = 110




```
#include <iostream>
#include <vector>
#include <cstdio>
#include <iostream>
using namespace std;
const int N = 3;
int a[9] = { 1,2,3,4,5,6,7,8,9 };
//将数字间的空格看做符号，扫描到空格则合并，扫描到符号则计算，共有3^8种数字组合形式
char c[N] = { '+','-',' ' };
int ans = 110;
vector<char> equs;
int joint(int old,int right){//拼接old与right
    int newnum = old*10 + right;
    return newnum;
}
```

```cpp
//枚举所有的符号
void cal(int k, int len) {//k为当前扫描的数字的位数，len为数组中间的可填充符号个数
    if (k >= len) {
        int i = 0, j = 0;
        int left = 0;
        int right;
        char op = '+';
        while (i < 9) {
            while (i < equs.size() && equs[i] == ' ')i++;
            right = a[j];
            while (j < i) {
                right = joint(right, a[j + 1]);
                j++;
            }//如果遇到‘ ’则把两边的数字拼接起来，一个一个地拼接
            if (op == '+'){
                left += right;
            }else {
                left -= right;
            }
            op = equs[i++];
            j = i;
        }
        if (left == ans) {
            cout<<1;
            for (int i = 0; i < equs.size(); i++) {
                if (equs[i]!=' ') {
                    cout << " " << equs[i] << " ";
                }
                cout << a[i+1];
            }
            cout << " = " << ans << endl;
        }
        return;
    }
    for (int i = 0; i < N; i++) {
        //先对各种符号进行枚举，全部压入equs里，对字符进行选择，先往符号数组加入，再递归调用
        equs.push_back(c[i]);
        cal(k + 1, len);
        equs.pop_back();//都试过却不可以得出正确结果则把符号弹出
    }
}
int main() {
    cal(0, 8);
    return 0;
}
```

## 删除不合法字符

现在有一串字符串由【（）】组成，请判断其是否括号匹配（。如果合法就输出合法，如果不合法就删除不合法的字符，使其成为合法字符串。

### C++从string中删除所有的某个特定字符

**std::remove**

C++中要从string中删除所有某个特定字符， 除了自己写一个函数外，还可用如下代码：

```cpp
str.erase(std::remove(str.begin(), str.end(), 'a'), str.end());
```

其中, remove来自, 它的签名是

```
template <class ForwardIterator, class T>
  ForwardIterator remove (ForwardIterator first, ForwardIterator last, const T& val);
```

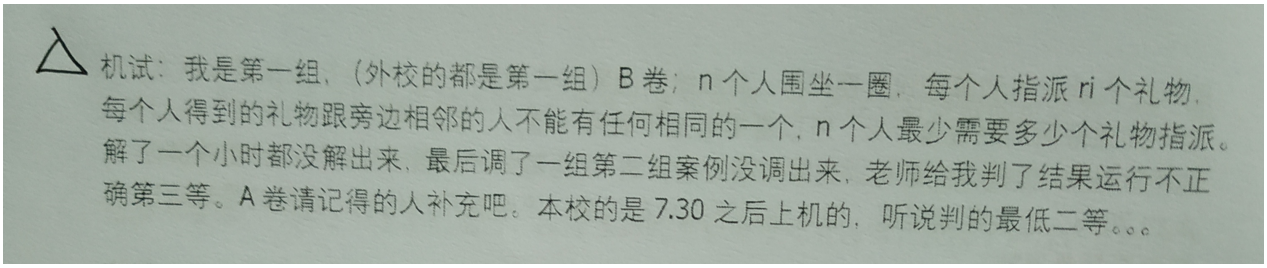
作用：为在容器中, 删除[first, last)之间的所有值等于val的值。删除方法： 将某个等于val的值用下一个不等于val的值覆盖。返回值:：一个迭代器 (记作newEnd), 该迭代器指向最后一个未被删除元素的下一个元素，即相当于容器新的end。

所以, 运行完remove后， 容器的[first, newEnd)内的元素即为所有未被删除的元素， [newEnd, end)之间的元素已经没用了。这样， 再运行str.erase(newEnd, str.end())即可清空[newEnd, end)之间的元素。

```
#include <iostream>
#include <cstring>
#include <string>
#include <stack>
#include <cstdio>
#include <algorithm>//remove函数来自algorithm头文件
using namespace std;
const int MAXN = 200;
int a[MAXN];//存储有错误的位置
struct Equ{
    int x;//坐标
    char c;//字符
    Equ(int x,char c):x(x),c(c){};
    Equ(){}

};
Equ equals[MAXN];
void cal(Equ equal[],int n){
    stack<Equ> sta;
    for (int i = 0; i < n; ++i) {
        if (equal[i].c == '(') {//左括号入栈
            sta.push(equal[i]);
        }
        if (equal[i].c== '[') {
            sta.push(equal[i]);
        }
        if (equal[i].c== ')') {
            if (sta.empty()) {//如果一开始就是不匹配的，比如'))'就直接跳过
                 a[i] = 1;
                continue;
            }else{
            char op = sta.top().c;
            if (op == '(') {
                sta.pop();
                continue;
            }else{
                a[i] = 1;
            }
        }
    }
        if (equal[i].c== ']') {
            if (!sta.empty()) {
                char op = sta.top().c;
                if (op == '[') {
                    sta.pop();
                    continue;
                }else{
                    a[i] = 1;
                }
```

```
            }else{
                a[i] = 1;
                continue;
            }

        }
    }
    while (!sta.empty()) {
        int u = sta.top().x;
        a[u] = 1;
        sta.pop();
    }
}
int main(){
    string s;
    cout << "输入待匹配的等式" << endl;
    cin >> s;
    int n = s.size();
    memset(a, 0, sizeof(a));
    memset(equals, 0, sizeof(equals));
    for (int i = 0; i < n; ++i) {
        equals[i] = Equ(i, s[i]);
    }
    cal(equals, n);
    for (int i = 0; i < n; ++i) {
        if (a[i] == 1) {
            s[i] = ' ';
        }
    }
    cout << s << endl;
    s.erase(remove(s.begin(), s.end(), ' '), s.end());//删除‘ ’符号
    cout << s;
    return 0;
}
//))))((()()()()([][]

```

## 分发礼物




思路：

- 先把第一个小朋友礼物分好，有 r1 个不同的礼物
- 从第2个小朋友开始，分配礼物，需要看他与前一个小朋友的礼物不能够重合，优先从已经出现的但是前一个小朋友没有的礼物集合diff中进行选择，如果把集合选择完，礼物还是不够，那就分配全新的未出现的礼物，直到所需的个数满足要求
- 最后一个小朋友，不仅diff集合要和前一个小朋友比较，也要和第一个小朋友比较，与他们都不相同的礼物才能进入集合diff，随后继续利用diff集合分配礼物

```
#include <cstdio>
```

```cpp
#include <iostream>
#include <algorithm>
#include <set>
#include <vector>
using namespace std;
const int MAXN = 200;
int a[MAXN];
set<int> gift[MAXN]; //每个小朋友分到的礼物的编号
int main() {
    int n;
    printf("有几个小朋友需要礼物? ");
    scanf("%d", &n); // n至少为2
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]); // 每个小朋友需要的礼物数量
    }
    int has_dist = 0; //已经分配出去的礼物个数
    set<int> am; // 已经分配出去的礼物编号，每个礼物都不同，用set集合保证编号各不相同
    for (int i = 0; i < a[0]; i++) {//第一个小朋友获得礼物必须各不相同
        am.insert(i + 1);
        gift[0].insert(i + 1);
    }
    has_dist = a[0];
    vector<int> diff; // 存储所有分配出去礼物的编号集合与前一个小朋友分到礼物的编号集合的差集
    for (int i = 1; i < n; i++) {
        int temp = a[i];//第i+1个小朋友得到的礼物数量
        set<int> pre = gift[i - 1];//前一个小朋友pre的礼物组成集合
        vector<int> diff(1000);//存储与前一个小朋友不同的礼物并且其他小朋友已经有的礼物的集合
//默认初始化 diff的语句， 作用是 diff[0] = ... = diff[999] = 0;
//auto的原理就是根据后面的值，来自己推测前面的类型是什么。
//set_diference就是取集合差集用前面参数的集合减去后面参数的集合把结果存在最后一个vector里面
//用于求两个集合的差集，结果集合中包含所有属于第一个集合但不属于第二个集合的元素
        //iter为已经分配出去的编号集合减去前一个小朋友得到礼物集合的集合
        auto iter = set_difference(am.begin(), am.end(), pre.begin(), pre.end(), diff.begin());
        diff.resize(iter - diff.begin());
//是把diff这个集合修整为刚刚好的大小,恰好保存与前一个小朋友不同但是其他小朋友已经有的礼物种类的集合
//1、resize(n)调整容器的长度大小，使其能容纳n个元素。如果n小于容器的当前的size，则删除多出来的元素。
//否则，添加采用值初始化的元素。
//2、resize(n，t)多一个参数t，将所有新添加的元素初始化为t。
        if (i == n - 1) {//分配到了最后一个小朋友，返回其他小朋友有但是第一个小朋友没有的礼物的集合
            auto iter = set_difference(diff.begin(), diff.end(),
                                       gift[0].begin(), gift[0].end(),
                                       diff.begin());
            diff.resize(iter - diff.begin());
        }
        for (int j = 0; j < diff.size() && temp > 0; j++) {
            //当j小于可能分配集合的大小，并且还需分配礼物的时候
            //向当前小朋友礼物集合中加入可能分配的礼物
            gift[i].insert(diff[j]);
            temp--;
        }
        while (temp--) {//当可能分配的礼物已经被分完的时候，添加全新的未出现的礼物
            has_dist++;//记录已经出现的礼物种类
            am.insert(has_dist);//往当前可分配礼物集合当中加入新礼物的编号
            gift[i].insert(has_dist);//往第i+1位小朋友中加入全新的礼物
        }
    }
    printf("%d", has_dist);
    return 0;
}
```