2023 CSP-J1 试题解析

1 在 C++ 中，下面哪个关键字用于声明一个变量， 其值不能被修改?

A. unsigned

B. const

C. static

D. mutable

> B。声明常量的格式是 const 数据类型 常量名 = 常量值;，例： `const double PI = 3.14;`

2 八进制数 12345670_8 和 07654321_8 的和为

A.22222221_8

B.21111111_8

C.22111111_8

D.22222211_8

> D 。列竖式，两数相加，逢 8 进 1。
> 12345670
> 07654321
> =
> 22222211

3 阅读下述代码，请问修改 `data` 的 `value` 成员以存储 3.14 ，正确的方式是

```
union Data{
    int num;
    float value;
    char symbol;
};
union Data data;
```

A. `data.value = 3.14;`

B. `value.data = 3.14;`

C. `data -> value = 3.14;`

D. `value->data = 3.14;`

> A。考察联合体 `union`。一个 `union` 内可以定义多种不同的数据类型，一个某 `union` 类型的变量中，允许装入该 `union` 中所定义的任何一种数据，这些数据共享同一段内存，以达到节省空间的目的。
> 联合体类型的变量通过点 (.) 来取其成员。`data.value = 3.14` 的意思就是将变量 data 所表示的内存当做 float 类型，保存 3.14 这一值。 CD选项应为指针。

4 假设有一个链表的节点定义如下：

```
struct Node {
    int data;
    Node* next;
};
```

现在有一个指向链表头部的指针：`Node* head`。如果想要在链表中插入一个新节点，其成员 `data` 的值为 `42`，并使新节点成为链表的第一个节点，下面哪个操作是正确的？（）

A. `Node* newNode = new Node; newNode->data = 42; newNode->next = head; head = newNode;`

B. `Node* newNode = new Node; head->data = 42; newNode->next = head; head = newNode;`

C. `Node* newNode = new Node; newNode->data = 42; head->next = newNode;`

D. `Node* newNode = new Node; newNode->data = 42; newNode->next = hea`

> A
> 考察单链表头插法。注意 head 指向的不是头结点，而是“链表头部”，也就是第一个结点。
> 先从堆区申请结点 `Node* newNode = new Node;`，然后给新结点设值 `newNode->data = 42;`，接下来把新结点的下一个结点设为链表的第一个结点 `newNode->next = head;`，最后把指向第一个结点的指针 head 指向新结点 `head = newNode;`。

5 根节点的高度为 1，一根拥有 2023 个节点的三叉树高度至少为（）。

A. 6

B. 7

C. 8
D. 9

> C ，节点数固定，三叉树高度最小时，应该是上面每层都是满的，最下面一层可能不满。
> 考虑满三叉树每层结点数
> 第 1 层 1 个
> 第 2 层 3 个
> 第 3 层 9 个
> ...
> 第 h 层 3^{h-1} 个
> h 层满三叉树的总结点数为： $3^0+3^1+...+3^{h-1}=(3^h-1)/2$ 。
> 当 h 为 7 时，7 层满三叉树的总结点数为 $(3^7-1)/2=1093$ 。
> 当 h 为 8 时，8 层满三叉树的总结点数为 $(3^8-1)/2=3280$ 。
> 2023 个结点的三叉树结点数大于 7 层满三叉树的结点数，小于 8 层满三叉树的结点数，因此该三叉树的高度为 8 层。想让该树层数更多很容易，排成一条链，高度就是 2023。因此一个拥有 2023 个节点的三叉树高度至少为 8。

6 小明在某一天中依次有七个空闲时间段，他想要选出至少一个空闲时间段来练习唱歌，但他希望任意两个练习的时间段之间都有至少两个空闲的时间段让他休息，则小明一共有（）种选择时间段的方案。
A. 31
B. 18
C. 21
D. 33

> B
> 考察递推。
> 将问题抽象：有个 7 位二进制数，其中至少 1 位是 1，要求两个 1 之间至少有两个 0，请问这样的二进制数的个数。
> 设 `d[i]` 表示考虑前 `i` 位二进制数中，两个 1 之间至少有两个 0 的所有情况数。
> 易知 `d[1] = 2` ， `d[2] = 3` ， `d[3] = 4` 。
> 当 i > 3 时，
>
> - 如果第 i 位置放 1，则第 i-1，i-2 位置必须是 0，情况数为前 i-3 位相邻两个 1 之间至少两个 0 的的情况数，为 `d[i-3]` 。

- 如果第 i 位置放 0，前 i-1 位相邻两个 1 之间至少两个 0 的情况数为 `d[i-1]`

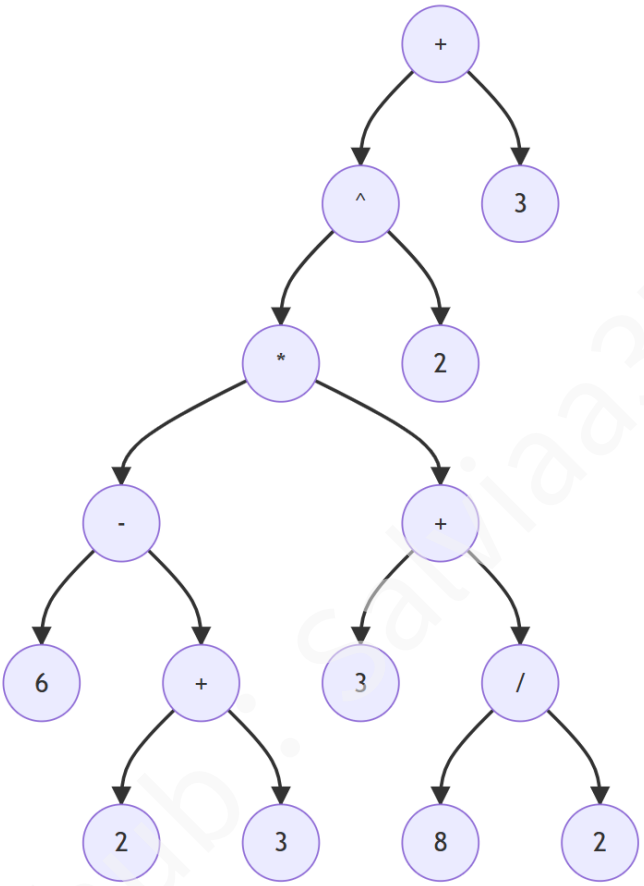
因此递推关系为 `d[i] = d[i-1]+d[i-3]`。
递推求出 `d[7]`。
`d[4] = d[3]+d[1] = 6`
`d[5] = d[4]+d[2] = 9`
`d[6] = d[5]+d[3] = 13`
`d[7] = d[6]+d[4] = 19`
`d[7]` 表示前 7 位数两个 1 之间至少有 2 个 0 的情况数，包括了每一位都为 0 的情况。而本题要求 “至少选出一段空闲时间练习唱歌”，也就是二进制数中至少有 1 位 1，应该去掉 7 位 0 的情况，因此最终结果为 `d[7]-1=18`

7 以下关于高精度运算的说法错误的是（）。
A. 高精度计算主要是用来处理大整数或需要保留多位小数的运算。
B. 大整数除以小整数的处理的步骤可以是，将被除数和除数对齐，从左到右逐位尝试将除数乘以某个数，通过减法得到新的被除数，并累加商。
C. 高精度乘法的运算时间只与参与运算的两个整数中长度较长者的位数有关。
D. 高精度加法运算的关键在于逐位相加并处理进位。

C
A. 高精度计算可以处理大整数的保存和计算，对于保留多位小数的运算，可以采用定点数的形式，在高精度的数字数组中人为规定小数点的位置（即将小数写成整数乘以基数的幂的形式，而后进行整数运算)，即可完成定点数小数高精度运算。
如 $1.2 * 7.3$，可以写成：$120 * 10^{-2} * 734 * 10^{-2} = 120 * 734 * 10^{-4}$，进行整数乘法运算后，得到 $87600 * 10^{-4}$，输出时，指定在倒数第 4 位前输出小数点。
B. 描述了除法竖式的书写过程，高精除低精运算确实就是在模拟除法竖式，B 选项描述正确。
C. 高精度乘法的运算时间与参与运算的两个整数的位数都有关，如果两个整数的位数分别为 m 和 n，那么高精乘高精的复杂度为 $O(mn)$
D. 描述正确

8 后缀表达式“`6 2 3 + - 3 8 2 / + * 2 ^ 3 +`”对应的中缀表达式是 ()。
A. `((6 - (2 + 3)) * (3 + 8 / 2)) ^ 2 + 3`
B. `6 - 2 + 3 * 3 + 8 / 2 ^ 2 + 3`

C. `(6 - (2 + 3)) * ((3 + 8 / 2) ^ 2) + 3`
D. `6 - ((2 + 3) * (3 + 8 / 2)) ^ 2 + 3`

> A
> 关于通过后缀表达式建立表达式树的方法，可以进入洛谷 P1449 后缀表达式，看里面的解法 2。
> 先根据后缀表达式建表达式树。得到：
>
> 
>
>
> 而后中序遍历表达式树，遍历子树得到的中缀表达式两边要加括号，即可得到中缀表达式。

9 数 101010_2 和 166_8 的和为 ( )
A.$(10110000)_2$
B.$(236)_8$
C.$(158)_{10}$
D.$(A0)_{16}$

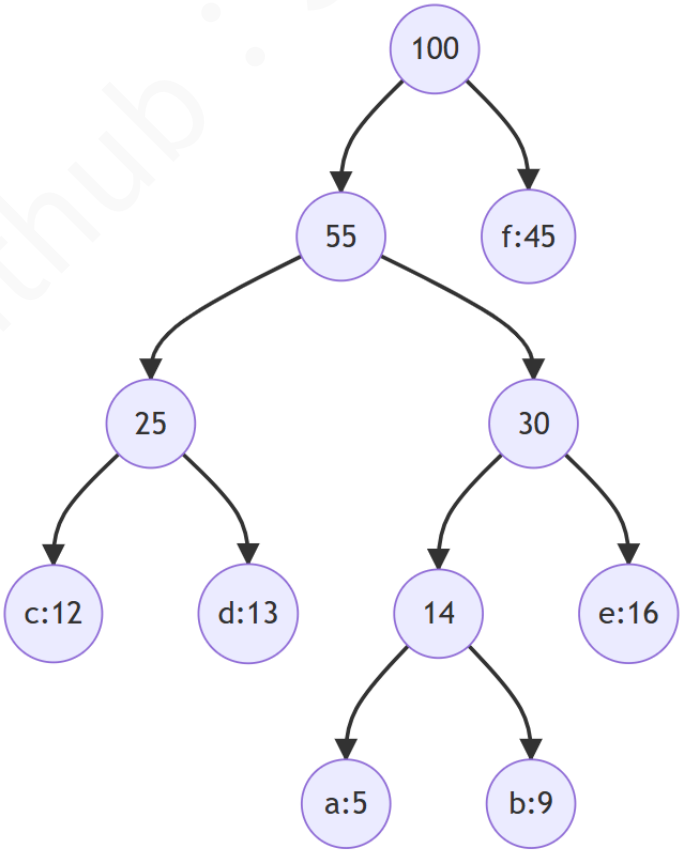
> D
> 不同进制计算最快的方式是把数字都尽量转为二进制数字进行比较。1 位 8 进制数字转为 3 位 2 进制数字。1 位 16 进制数字转为 4 位 2 进制数字。十进制转二进制用除2取余法。
> $166_8 = 1110110_2$
> $236_8 = 10011110_2$

将 158_{10} 转为 8 进制，使用除基取余法：
158/8=19...6
19/8=2...3
2/8=0...2
所以 $158_{10} = 236_8$ ，与 B 选项相同。
综上，只有 D 选项的值与题目给定的两个数字的加和相同。

10 假设有一组字符 `{a,b,c,d,e,f}`，对应的频率分别为 $5\%, 9\%, 12\%, 13\%, 16\%, 45\%$。请问以下哪个选项是字符abcdef分别对应的一组哈夫曼编码?

A. `1111,1110,101,100,110,0`
B. `1010,1001,1000,011,010,00`
C. `000,001,010,011,10,11`
D. `1010,1011,110,111,00,01`

A
构建哈夫曼树，每个结点自己是一棵树，权值是相应的频率（相对大小）。
每次循环选择权值最小的两棵树，合并为一棵树，新的树的权值是两棵子树的权值加和。



得到哈夫曼树后，将每个结点连接到左右孩子的边上分别标注 0 与 1（必须一个是 0，一个是 1，哪边是 0 哪边是 1 无所谓），从根结点到叶子结点的路

> 径经过的边上的数字合起来就是该叶子结点表示的字符的哈夫曼编码。
> 我们不能确定具体各条边上标的是 0 还是 1，但可以确定每个字母的哈夫曼编码的位数。根据哈夫曼树，a 与 b 一定是 4 位，c,d,e 一定是 3 位，f 一定是 1 位。只有 A 选项满足这种情况。

11 给定一棵二叉树，其前序遍历结果为：ABDECFG，中序遍历结果为：DEBACFG。请问这棵树的正确后序遍历结果是什么？（）

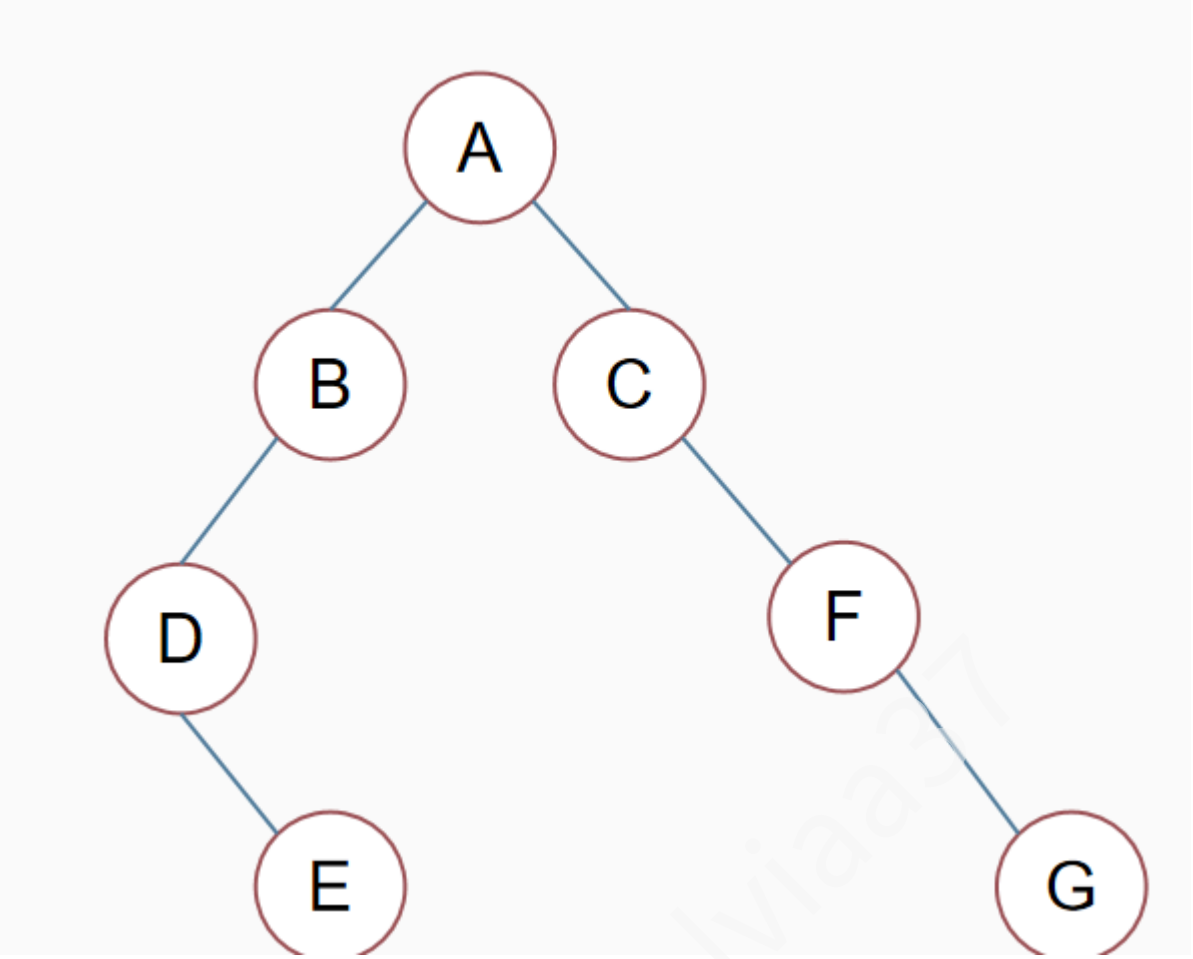
A. EDBGFCA

B. EDBGCFA

C. DEBGFCA

D. DBEGFCA

> A
> 二叉树已知前序中序求后序，方法参考
> 前序遍历根结点 A，中序遍历中找到根结点 A。
> 左子树的前序遍历序列：BDE，中序遍历序列：DEB
> 根结点 B，左子树中序：DE，左子树前序：DE。
> 根结点 D，看左子树中序，得到右孩子是 E。
> 右子树的前序遍历序列：CFG，中序遍历序列：CFG
> 根结点 C，右子树中序：FG，右子树前序：FG
> 根结点 F，右孩子 G。



对其进行后序遍历的结果是 EDBGFCA，选 A。

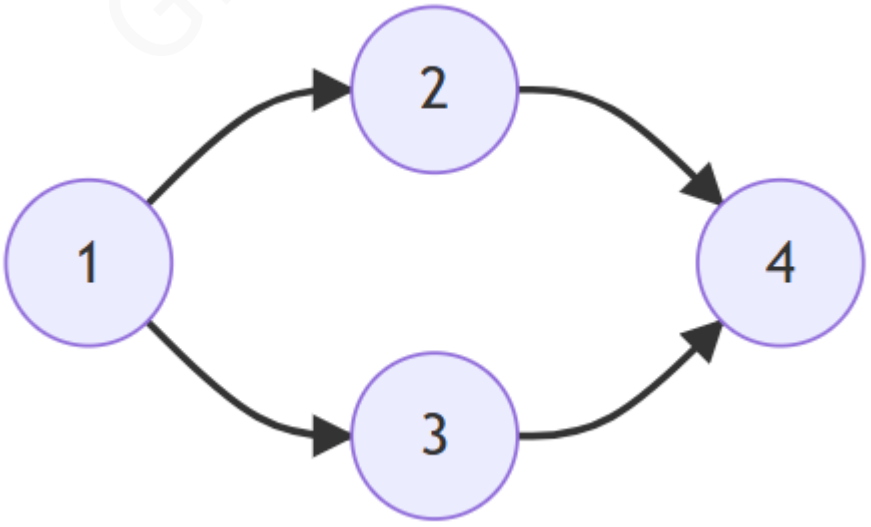
12 考虑一个有向无环图，该图包括 4 条有向边：(1,2)，(1,3)，(2,4)，和 (3,4)。以下哪个选项是这个有向无环图的一个有效的拓扑排序？（）

A. 4，2，3，1

B. 1，2，3，4

C. 1，2，4，3

D. 2，1，3，4



B。

根据求拓扑排序的 算法，每次删掉一个入度为 0 的顶点，删掉顶点的顺序

> 就是拓扑排序。
> 该图的拓扑排序可以是 1 2 3 4，或 1 3 2 4。选 B。

13 在计算机中，以下哪个选项描述的数据存储容量最小？（）
A. 字节（byte）
B. 比特（bit）
C. 字（word）
D. 千字节（kilobyte）

> B
> 字节 (byte)，简称 B。比特 (bit)，简称 b。千字节 (kilibyte)，简称 KB。
> 有换算关系：1B=8b, 1KB = 1024B。
> 字 (word)，即该计算机的字长。一个 32 位计算机的一个字(word) 就是 32 位。字一定大于字节。
> 因此，比特描述的数据存储容量最小。

14 一个班级有 10 个男生和 12 个女生。如果要选出一个 3 人的小组，并且小组中必须至少包含 1 个女生，那么有多少种可能的组合？（）
A. 1420
B. 1770
C. 1540
D. 2200

> A
> 方法 1：正向思考。3 人小组中至少包含 1 个女生，可以是 2 男 1 女，1 男 2 女，或 3 女。情况数为：$_{210112} + _{110212} + _{312} = 540 + 660 + 220 = 1420$
> 方法 2：逆向思考，补集转化。至少包含 1 个女生的情况数，为选出 3 人小组的总情况数减去不包含女生的情况数。不包含女生，即 3 人是 3 个男生。情况数为：$_{322} - _{310} = 1540 - 120 = 1420$

15 以下哪个不是操作系统？（）
A. Linux
B. Windows
C. Android
D. HTML

> D
> HTML 是超文本标记语言，用来编写网页需要用到的语言之一。Linux，

> Windows，Android 都是操作系统。

二、阅读程序

（程序输入不超过数组或字符串定义的范围；判断题正确填 √，错误填 ×；除特殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分）

1

```
#include<iostream>
#include<cmath>
using namespace std;

double f(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
int main(){
   cout.flags(ios::fixed); // 将输出设置为定点小数
   cout.precision(4); // 将输出精度设置为4

   int a,b,c;
   cin>> a >> b >> c ;
   cout<< f(a,b,c) <<endl;
   return 0;
}
```

假设输入的所有数都为不超过 1000 的正整数，完成下面的判断题和单选题：

1. （2分）当输入为 `2 2 2` 时，输出为 `1.7321` （）
2. （2分）将第7行中的 `(s-b)*(s-c)` 改为 `(s-c)*(s-b)` 不会影响程序运行的结果（）
3. （2分）程序总是输出四位小数（）
4. 当输入为 `3 4 5` 时，输出为（）
5. 当输入为 `5 12 13` 时，输出为（）

先看 `f` 函数

```
double f(double a, double b, double c) {
    double s = (a + b + c) / 2;
```

```
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

如果了解三角形面积的海伦公式，很明显可以看出 `a,b,c` 是三角形的三条边，`s` 是半周长，`f` 函数为使用海伦公式求三角形的面积。

```
int main() {
    cout.flags(ios::fixed);
    cout.precision(4);
    int a, b, c;
    cin >> a >> b >> c;
    cout << f(a, b, c) << endl;
    return 0;
}
```

主函数中前两句是用于设定浮点数输出格式，输出时固定保留 4 位小数，最后一位四舍五入，不足 4 位则补 0。
而后输入三角形三条边长，输出三角形面积。
注意：虽然本题使用了海伦公式求三角形面积，但具体在做题时，我们可以通过自己熟悉的或更方便的方法求三角形面积。

16 当输入为 `2 2 2` 时，输出为 `1.7321` （）

> ✓
> 边长为 2 的等边三角形，高为 $\sqrt{2^2-1^2}=\sqrt{3}$，面积为 $=2*\sqrt{3}/2=\sqrt{3}$，$\sqrt{3}$保留 4 位小数就是 1.7321。
> 如果忘了 $\sqrt{3}$的值了，针对这个问题，可以求一下 1.73210^2 3.0002 3 1.73205^2 2.999997 3，所以 1.73205 $\sqrt{3}$ 1.73210，四舍五入到小数点后第四位，得到$\sqrt{3}$ 1.7321。

17 将第 7 行中的 `(s-b)*(s-c)` 改为 `(s-c)*(s-b)` "不会影响程序运行的结果（）

> ✓
> 乘法交换律，两个数字相乘，交换顺序结果不变。

18 程序总是输出四位小数（）

> ×
> `cout.flags(ios: :fixed); cout.precision(4);` 运行这两句后，设置

> 了输出浮点数的格式为保留 4 位小数输出。
> 但是题目没有给定输入数据的范围，输入的三个数字可能会：两边之和小于等于第三边而无法构成三角形。如果 a+b<c，则 $=(+ +)/2$ ，$- \ 0, \ - \ 0, \ - \ 0$，会导致 $(-)(-)(-) \ 0$，在使用海伦公式 $\sqrt{(-)(-)(-)}$时无法对负数开根号，该程序如果运行，会输出 nan。

19 当输入为 "3 4 5" 时，输出为（）
A. 6.0000
B. 12.0000
C. 24.0000
D. 30.0000

> 答：A
> 3 4 5 是勾股数，边长为 3 4 5 的三角形是直角三角形，直角边为 3 4，面积为： $=3*4/2=6$

20 当输入为 "5 12 13" 时，输出为（）
A. 24.0000
B. 30.0000
C. 60.0000
D. 120.0000

> 答：B
> 5 12 13 是勾股数，边长为 5 12 13 的三角形是直角三角形，直角边为 5 12，面积为： $=5*12/2=30$

2

```cpp
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int f(string x,string y){
    int m=x.size();
    int n=y.size();
    // 二维vector用于保存状态空间
    vector<vector<int>>v(m+1,vector<int>(n+1,0));
    // 遍历x和y两个字符串
```

```
    for(int i=1;i<=m;i++){
       for(int j=1;j<=n;j++){
       // 字符相同，等于上一个状态+1
            if(x[i-1]==y[j-1]){
            //
                v[i][j]=v[i-1][j-1]+1;
            }else{
                v[i][j]=max(v[i-1][j],v[i][j-1]);
            }
        }
    }
    return v[m][n];
}
bool g(string x,string y){
    if(x.size() != y.size()){
        return false;
    }
    return f(x+x,y)==y.size();
}
int main(){
    string x,y;
    cin>>x>>y;
    cout<<g(x,y)<<endl;
    return 0;
}
```

考点

- 线性动态规划：求最长公共子序列
- STL vector

解题思路

先看 f 函数

```
int f(string x, string y) {
    int m = x.size();
    int n = y.size();
}
```

传入两个字符串 x 和 y，m 是字符串 x 的长度（字符个数），n 是字符串 y 的长度。

```
vector<vector<int>> v(m + 1, vector<int>(n + 1, 0));
```

`vector<int>(n + 1, 0)` 表示声明一个保存 int 类型变量的 vector，长度是 n+1（这个 vector 中有 n+1 个元素），元素初值为 0。
`vector<vector<int>>` 表示外层 vector 中每个元素都是 `vector<int>` 类型的对象。
`v(m+1, vector<int>(n+1, 0))` 表示外层 vector 对象名是 v，长度是 m+1，每个元素的初值都是 `vector<int>(n+1, 0)`。
外层顺序表中有 m+1 个元素，每个元素都是一个保存 int 类型的长度为 n+1 的顺序表。
实际这一段就是在使用 vector 声明一个 m+1 行 n+1 列的二维数组 v。

```
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (x[i - 1] == y[j - 1]) {
                v[i][j] = v[i - 1][j - 1] + 1;
            } else {
                v[i][j] = max(v[i - 1][j], v[i][j - 1]);
            }
        }
    }
    return v[m][n];
}
```

这一段是线性动态规划的核心部分，其功能是**求两个字符串的最长公共子序列**问题后才可以理解。
这段代码中，`v[i][j]` 是状态，表示 x 字符串的前 i 个字符和 y 字符串的前 j 个字符的最长公共子序列的长度。
最后返回 `v[m][n]`，即 x 字符串的前 m 个字符和 y 字符串的前 n 个字符的最长公共子序列的长度。
因此，`int f(string x, string y)` 求的就是 x 字符串和 y 字符串的最长公共子序列的长度。

```
bool g(string x, string y) {
    if (x.size() != y.size()) {
```

```
        return false;
    }
    return f(x + x, y) == y.size();
}
```

g 函数传入两个字符串 x，y。

- 如果两字符串长度不同，则返回 `false`。
- 如果长度相同，先求 `x+x`（x 后面再连接 x 得到的字符串）和 y 字符串的最长公共子序列的长度，看该长度是否与 y 的长度相同。也就是看 `x+x` 中是否存在一个子序列是 y 字符串。（在 x+x 中顺序选取部分字符（可以跳着选），看能否选出完整的 y 字符串）。

```
int main() {
    string x, y;
    cin >> x >> y;
    cout << g(x, y) << endl;
    return 0;
}
```

主函数，输入 x，y 字符串，输出 x+x 是否存在一个子序列是 y 字符串，如果存在输出 1，否则输出 0。

判断题

21 f 函数的返回值小于等于 `min(n,m)`。（）

> ✓
> f 函数求 x 与 y 的最长公共子序列的长度，两字符串最长公共子序列的长度不会大于两字符串中任意一个字符串，最长时也就是与两字符串中更短的字符串一样长。该判断正确。

22 f 函数的返回值等于两个输入字符串的最长公共子串的长度。（）

> ×
> f 函数的返回值是两个输出字符串的最长公共子序列的长度，而不是最长公共子串。子串是连续元素构成的子序列。子序列可以是不连续的，子串必须是连续的。
> 比如 `bcd` 是 `abcde` 的子串，也是子序列。而 `ace` 是 `abcde` 的子序列，不是子串。

23 当输入两个完全相同的字符串时，g 函数的返回值总是 true（）

> ✓
> 当 x 与 y 相同时，字符串 `x+x` 中一定存在子串 `y`（即子串 x），子串就是子序列，所以 `x+x` 与 `y` 的最长公共子序列的长度就是 y 的长度，g 函数返回真。

24 将第 19 行中的 `v[m][n]` 替换为 `v[n][m]`，那么该程序（）
A. 行为不变
B. 只会改变输出
C. 一定非正常退出
D. 可能非正常退出

> D
> 如果输入的 x 与 y 长度不等，那么不会运行到 f 函数，直接输出 0，没有非正常退出。
> 如果输入的 x 与 y 长度相等（空字符串无法输入，字符串长度最小为 1），在 g 函数中调用的是 f(x+x, y)，运行到函数内部，`int f(string x, string y)`，形参 x 的长度一定是 y 的二倍。也就是 `m == 2*n`，因此也一定有 `m > n`。
> 把 v 当做二维数组，v 的第一维的长度是 m+1，可以使用的下标范围是 `0~m`。第二维的长度是 n+1，可以使用的下标范围是 `0~n`。由于 `m>n`，当取 `v[n][m]` 时第二维数组越界，会产生运行时错误。（出现运行时错误时，程序可能会正常退出，也可能非正常退出。我们应该默认它会非正常退出。）

25 当输入为 `csp-j p-jcs` 时，输出为（）
A. 0
B. 1
C. T
D. F

> B
> 看 `csp-jcsp-j` 中是否存在子序列是 p-jcs。
> 仔细看，是存在的。`csp-jcsp-j`。
> 第 3 到第 7 字符就是 p-jcs。
> 输出 1。（输出布尔值真时，输出 1，输出布尔值假时，输出 0）

26 当输入为 `csppsc spsccp` 时，输出为（）

A. T

B. F

C. 0

D. 1

> D
> 看 `csppsccsppsc` 中是否存在子序列是 `spsccp`。
> 仔细看，是存在的。csppsccsppsc。
> 第 2，3，5，6，7，9 字符连起来就是 `spsccp`。
> 输出 1。（输出布尔值真时，输出 1，输出布尔值假时，输出 0）

3

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int solve1(int n){
    return n*n;
} // n的平方

int solve2(int n){
   int sum=0;
   for(int i=1;i <= sqrt(n);i++){
        if(n % i ==0){  // i是n的因数
            if(n / i == i){ // n能被i开平方
                sum += i * i;
            }else{
                sum += i * i + (n/i) * (n/i);
            }
        }
    }
    return sum;
} // 求n的平方和
int main(){
    int n;
    cin>>n;
    cout<<solve2(solve1(n))<<" "<<solve1((solve2(n)))<<endl;
```

```
    return 0;
}
```

假设输入的 n 是绝对值不超过 1000 的整数，完成下面的判断题和单选题。

考点

- 因数、质数

解题思路

```
int solve1(int n) {
    return n * n;
}
cpp123
```

solve1(n)，求 n^2

```
int solve2(int n) {
    int sum = 0;
    for (int i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            if (n / i == i) {
                sum += i * i;
            } else {
                sum += i * i + (n / i) * (n / i);
            }
        }
    }
    return sum;
}
cpp12345678910111213
```

i从1循环到 $\sqrt{n}$，如果i是n的因数：

- 如果n/i为i，即i为 $\sqrt{n}$，此时i是n的因数，把 i^2 加到sum中。
- 如果n/i不为i，此时i与n/i是n的因数，把 i^2 和 $(n/i)^2$ 加到sum中。

i从1循环到 $\sqrt{n}$，取循环中取到的i或n/i可以遍历n的所有因数。
因此solve2(n)求的是n的所有因数的平方和。

```cpp
int main() {
    int n;
    cin >> n;
    cout << solve2(solve1(n)) << " " << solve1((solve2(n)))
<< endl;
    return 0;
}
cpp123456
```

主函数中，第一个输出的 `solve2(solve1(n))` 是 n^2n 的所有因数的平方和。第二个输出的 `solve1((solve2(n)))` 是n的所有因数的平方和的平方。

判断题

27 如果输入的n为正整数，solve2函数的作用是计算n所有的因子的平方和（）

> ✓
> 上面已解释

28 第13~14行的作用是避免n的平方根因子i（或n/i）进入第16行而被计算两次（）

> ✓
> n的平方根因子，就是 $\sqrt{n}$ （ $\sqrt{n}$ 是整数），在计算因数的平方和时， $(\sqrt{n})^2$ 只需要计算一次，需要进入第14行，而不能进入第16行被计算两次。

29 如果输入的n为质数，solve2(n)的返回值为 n^2+1 （）

> 答：T
> 质数n只有1和n两个因数，因数的平方和为 n^2+1 。

单选题

30. 如果输入的n为质数p的平方，那么solve2(n)的返回值为（）

A. $^2+\ +1$

B. n^2+n+1

C. n^2+1

D. $^4+2^2+1$

> 答：B
> $n=^2$ ，n的所有因数为： $1,,^2$ ，因数的平方和为： $1+^2+^4$ 或写为 $1+n+n^2$

31 当输入为正整数时，第一项减去第二项的差值一定（）
A.大于 0
B.大于等于 0 且不一定大于 0
C.小于 0
D.小于等于 0 且不一定小于 0

> D
> 如果输入的是n，第一项是 n^2 所有因数的平方和。第二项是n的所有因数的平方和的平方。
> 可以用特例法：
> 如果n是1，第一项是1，第二项是1，差是0。
> 如果n是2，第一项是4的所有因数1,2,4的平方和，是21。第二项是2的所有因数1，2的平方和5的平方为25。第一项减第二项小于0。
> 只有D符合条件。

32 当输入为 `5` 时，输出为（）
A. `651 625`
B. `650 729`
C. `651 676`
D. `652 625`

> C
> 第一项是 $5^2=25$ 所有因数1,5,25的平方和： $1^2+5^2+25^2=651$
> 第二项是5的所有因数1，5的平方和26的平方： $26^2=676$

三、完善程序

（寻找被移除的元素）问题：原有长度为 n+1、公差为 1 的等差升序数列；将数列输入到程序的数组时移除了一个元素，导致长度为 n 的升序数组可能不再连续，除非被移除的是第一个或最后一个元素。需要在数组不连续时，找出被移除的元素。
试补全程序。

```
#include <iostream>
#include <vector>
using namespace std;
int find_missing(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;
```

```
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == mid + ①) {
            ②
        } else {
            ③
        }
    }
    return ④
}

int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for(int i = 0; i < n; i++)cin >> nums[i];
    int missing_number = find_missing(nums);
    if (missing_number == ⑤) {
        cout << "Sequence is consecutive" << endl;
    } else {
        cout << "Missing number is " << missing_number <<
endl;
    }
    return 0;
}
```

考点

- 二分

解题思路

先看主函数

```
int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for(int i = 0; i < n; i++)cin >> nums[i];
    int missing_number = find_missing(nums);
```

```
    if (missing_number == ⑤) {
        cout << "Sequence is consecutive" << endl;
    } else {
        cout << "Missing number is " << missing_number <<
endl;
    }
    return 0;
}
```

输入 n，输入 n 个元素保存在顺序表nums 中，下标 0~n-1。根据题目，等差数列公差为 1，一共 n+1 个数字，如果被移除的不是 `nums[0]`，那么这 n+1 个数字最小值为 `nums[0]`，最大值应该为 `nums[0]+n`，在这 n+1 个数中移除一个数字。
调用 find_missing 函数，查找丢失的数字，赋值给 missing_number。
如果丢失的数字是 (5)，那么输出 “序列是连续的”，否则输出丢失的数字。
也就是说如果丢失的数字是 n+1 个数字中的第 1 个或最后一个，find_missing 函数会返回 (5) 处应该填的值，此时整个序列就是连续的。

```
int find_missing(vector<int>& nums) {
    int left = 0, right = nums.size() - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == mid + ①) {
            ②
        } else {
            ③
        }
     }
     return ④
}
```

而后看 find_missing 函数，传入是的 nums 的引用。看这一代码结构，有 left, right, mid，自然是使用了二分查找算法。
设为 nums 的元素下标从 0~nums.size()-1，先将 left, right 指向 nums 保存的序列的两端。
只要 `left < right`，每次循环取中点位置 mid 为 `mid = left+(right-left)/2`，该写法在数学意义上等价于 `mid = (left+right)/2`。

> 之所以写成 `mid = left+(right-left)/2`，是因为这个写法可以保证即使 left+right 的值超过 int 可以表示的数值范围，仍能正确求出 `(left+right)/2` 的值

在整数域二分答案算法中，如果取中点的写法为 `mid = (left+right)/2`，那么就是在求答案变量满足某一条件的最小值。（反之，如果取中点的写法为 `mid = (left+right+1)/2`，那么就是在求答案变量满足某一条件的最大值。）
考虑答案 x 满足的条件，当 x 很小时应该不满足条件，当 x 很大时满足条件，这样才可以求出满足条件的最小值。
设整数序列起始值 `nums[0]` 为 s，去掉的数为 s+m。
答案变量初始 `l = 0, r = num.size()-1`，即答案变量初始范围设为了 `[0,n-1][0, n-1][0,n-1]`
答案变量为 x 时表示去掉的数为 `s+x`，要通过二分查找答案变量 x 满足条件的最小值，结果为 m。

	0	1	2	...	m-1	m	m+1	...
原数字序列	s	s+1	s+2	...	s+m-1	s+m	s+m+1	...
去掉 m 后的序列: nums	s	s+1	s+2	...	s+m-1	s+m+1	s+m+2	...

可以看到，在下标小于 m 的范围内，`nums[x]` 的值应该等于 `s+x`，下标大于等于 m 时，`nums[x]` 的值应该等于 `s+x-1`，即不等于 `s+x`。
要想使 m 为答案变量 x 的满足条件的最小值为 m，那么需要当 `x>=m` 时满足条件，`x<m` 时不满足条件，因而答案变量 x 应该满足的条件为 `nums[x] != s+x`。

- 如果 mid 值满足条件 `nums[mid] != s+mid`，那么应该让答案变量取更小的值，让 `right = mid`，(3) 处选 C。
- 如果 mid 值不满足条件 `nums[mid] != s+mid`，即满足 `nums[mid]==nums[0]+mid`（已知 `s` 就是 `nums[0]`，(1) 处选 B），那么应该让答案变量取更大的值，让 `left = mid+1`，(2) 处选 A。

while 循环跳出后，left 或 right 的值就是上面提到的 m，缺失的值应该是 s+m，也就是代码中的 `nums[0]+left` 或 `nums[0]+right`，(4) 处选 A 或 B。

如果去掉的数字就是第一个数字 s，或最后一个数字，整个序列是连续的序列，无论 mid 为何值，都满足 `nums[mid] == mid+nums[0]`，按照上述写法运行程序，会不断的改变 left，而 right 不变，直到 while 循环跳出，此时 left 与 right 的值都为 right 的初始值 `nums.size()-1`，返回值应该为 `nums[0]+nums.size()-1`。
在主函数中，nums 的长度 `nums.size()` 就是 n，因此当整个序列是连续序列时，find_missing 函数的返回值 missing_number 就是 `nums[0]+n-1`。
如果当答案变量 `x` 为最大值 `n-1` 时，仍然不满足条件 `nums[n-1] != nums[0]+n-1`（即满足 `nums[n-1]==nums[0]+n-1`），那么对于答案变量 x $0, n-1$，都满足 `nums[x] == nums[0]+x`，该序列是连续的。
所以只有返回的 `nums[0]+n-1` 的值与 `nums[n-1]` 的值相等时，序列才是连续的。
(5) 处选 D。

33. B
34. A
35. C
36. A 或 B
37. D

2

（编辑距离）给定两个字符串，每次操作可以选择删除(Delete)、插入(insert)替换(Replace)1个字符，求将第一个字符串转换为第二个字符串所需要的最少操作次数。

试补全动态规划算法。

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int min(int x, int y, int z) {
    return min(min(x, y), z);
}
int edit_dist_dp(string str1, string str2) {
```

```
    int m = str1.length();
    int n = str2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0)
                dp[i][j] = ①;
            else if (j == 0)
                dp[i][j] = ②;
            else if (③)
                dp[i][j] = ④;
            else
                dp[i][j] = 1 + min(dp[i][j - 1], dp[i - 1]
[j], ⑤);
        }
    }
    return dp[m][n];
}
int main() {
    string str1, str2;
    cin >> str1 >> str2;
    cout << "Mininum number of operation:" <<
edit_dist_dp(str1, str2) << endl;
    return 0;
}
```

考点

- 线性动态规划：求最长公共子序列

解题思路

```
int min(int x, int y, int z) {
    return min(min(x, y), z);
}
```

min函数求三个数的最小值

```
int main() {
    string str1, str2;
```

```
    cin >> str1 >> str2;
    cout << "Mininum number of operation:"
         << edit_dist_dp(str1, str2) << endl;
    return 0;
}
```

主函数输入两个字符串，输出两个字符串的编辑距离。

```
int edit_dist_dp(string str1, string str2) {
    int m = str1.length();
    int n = str2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0)
                dp[i][j] = ①;
            else if (j == 0)
                dp[i][j] = ②;
            else if (③)
                dp[i][j] = ④;
            else
                dp[i][j]=1+min(dp[i][j - 1],dp[i - 1][j],
⑤);
         }
    }
    return dp[m][n];
}
```

用两层 vector 设状态数组（作用相当于二维数组）dp。
`dp[i][j]` 指表示str1的前i个字符转变为str2的前j个字符的最少操作次数。
i为0时，str1的前0个字符转变为str2的前j个字符的操作方案为j次插入，操作次数为j。(1)填j，选A。
j为0时，str1的前i个字符转变为str2的前0个字符的操作方案为i次删除，操作次数为i。(2)填i，选B。
如果str1第i字符 `str1[i-1]` 与str2第j字符 `str2[j-1]` 相同，(3)填A。
那么 `dp[i][j]` 就是str1的前i-1个字符转变为str2的前j-1个字符的最少操作次数，(4)填B。
否则， `dp[i][j]` 就是最后一次进行插入、或删除、或修改时的最少操作次

数。如果最后一次将 `str1[i-1]` 修改为 `str2[j-1]` ，那么接下来要将str1的前i-1个字符转变为str2的前j-1个字符，(5)填C。

(38) ①处应该填( )

A. `j`

B. `i`

C. `m`

D. `n`

A

(39) ②处应该填( )

A. `j`

B. `i`

C. `m`

D. `n`

B

(40) ③处应该填( )

A. `str1[i - 1] == str2[j - 1]`

B. `str1[i] == str2[j]`

C. `str1[i - 1] != str2[j - 1]`

D. `str1[i] != str2[j]`

A

(41) ④处应该填( )

A. `dp[i - 1][j - 1] + 1`

B. `dp[i - 1][j - 1]`

C. `dp[i - 1][j]`

D. `dp[i][j - 1]`

B

(42) ⑤处应该填( )

A. `dp[i][j] + 1`

B. `dp[i - 1][j - 1] + 1`

C. `dp[i - 1][j - 1]`

D. `dp[i][j]`

C