SCP-J 2023 讲评

disangan233
中国人民大学
上海洛谷网络科技有限公司

洛谷
www.luogu.com.cn

选择题

1. 网址 www.luogu.com.cn 当中，顶级域名是（ B ）

最后一个 cn 是顶级域名

选择题

2. 以下物品可以携带进 CSP 第二轮测试考场的是（ C ）

A. 可以发出巨大响声的发光机械键盘
B. 写有 kkk 签名的《深入浅出程序设计竞赛 基础篇》的封皮
C. 印有 dzd 照片（已设法获得授权）的文化衫 T 恤
D. 具有拍照功能的游标卡尺

选择题

3. 将元素 a,b,c,d,e,f 依次入栈，则以下选项中出栈序列不可能是（D）

不可能是 d,c,f,b,e,a

选择题

4.「流程结构」是编程中用于控制程序执行流程的一种方式，它包括顺序结构、分支结构和循环结构。在一些诗歌作品中，也有对「流程结构」的体现。下列诗歌片段中体现循环结构的是？（B）

语文题

B. 只要我还能够行走，只要我还能够张望，只要我还能够呼吸，就一直走向前方

「只要」：`for/while`

选择题

5. 已知两个二进制整数 a, b:

$$a = 1010001010_{(2)}$$
$$b = 1110100110_{(2)}$$

则表达式 (a&b)^(a|b) 的值为（ B ）

(a&b)^(a|b)=a^b

所以答案为 0100101100

选择题

6. 一个有 10 个结点的有向图，要使得每一个满足 $1 \leq i,j \leq 10, i \neq j$ 的点对 (i,j) 都存在一条从 i 到达 j 的路径，至少需要连（ B ）条有向边。

如果是无向图，那么是一棵树

有向图连成一个环即可，需要 10 条有向边

选择题

7. 观察下列代码：

```
int a[] = {5, 4, 3, 2, 1};
auto p = a + 3;
auto q = &p;
(*q) ++;
auto k = *p;
```

其中 k 的类型以及 k 的值分别为？（A）

(*q)++ 其实就是 (*(&p))++ 也就是 p++

所以 k=*(a+4)=a[4]=1

选择题

8. 中缀表达式 a + (b + c) - d * e / f 改写为后缀表达式为（B）

a b c + + d e * f / -

选择题

9. 一张大小为 6114×8192 的 24 位彩色图片，采用 .bmp 格式存储，占用的空间大小约为？（A）

$$6114 \times 8192 \times 24 \div 8 = 150{,}257{,}664\ \text{B} \approx 143.3\ \text{MiB}$$

选择题

10. 以下程序片段的时间复杂度为？（A）

```
int cnt = 0;
 for(int i = 1;i <= n;i ++){
     for(int j = 1;j <= n;j += i){
         for(int k = 1;k <= n;k += j){
             ++ cnt;
         }
     }
 }
```

$$\sum_{i=1}^{n}\sum_{j=1}^{n/i}\frac{n}{ij}=n\log^2 n$$

但是只是这样是不对的

因为 j=1 的情况没有被算进去

j=1 时实际上是 n^2 的，所以总复杂度是 $O(n^2+n\log^2 n)=O(n^2)$

运行一下就会发现 $cnt=n^2+n\log^2 n$

选择题

11. 依次抛出四个六面骰子，按照抛出顺序将骰子上的数值记为 a,b,c,d。则 $a<b,b>c,c<d$ 的概率为（A）

考虑枚举 b,c，方案数为

$$\sum_{i=2}^{6}(i-1)\sum_{j<i}(6-j)=1\times5+2\times9+3\times12+4\times14+5\times15$$

$$=190$$

所以概率为

$$\frac{190}{6^4}=\frac{95}{648}$$

选择题

12. 十进制小数 0.3，转写成八进制为（ B ）

$$0.3 \times 8 = 2.4, 0.4 \times 8 = 3.2, 0.2 \times 8 = 1.6, 0.6 \times 8 = 4.8, 0.8 \times 8 = 6.4$$

所以转写成八进制为 $0.231463146\cdots$

选择题

13. 观察如下代码片段：

```
union U{
    bool flag1, flag2, flag3, flag4, flag5;
    signed   short a;
    unsigned short b;
    enum E{
        CardA = 0, CardB = 1,
        CardC = 2, CardD = 114514
    } e;
} u;
```

其中，sizeof(u) 的值为（A）

union 取的是 max

bool 占 1 字节，short 占 2 字节，enum 视为 int 占 4 字节

所以 sizeof(u) 的值为 4

选择题

14. 已知某种可用来维护序列的数据结构，支持 $O(\log n)$ 向某个位置后面插入元素（即，你可以指定一个数 k，将该元素插入到第 k 个元素后面）、$O(n)$ 查询某个元素的排名，$O(n \log n)$ 遍历整个序列，那么用上述三种操作实现插入排序的时间复杂度最坏为（ A ）

由于可以指定位置插入，所以只有输出时才需要一次遍历，时间复杂度为

$$O(n \log n + n^2 + n \log n) = O(n^2)$$

选择题

15. 今年是 CCF（中国计算机学会）第（ C ）次举办 CSP-J/S（计算机非专业级别的软件能力认证）

第一次 CSP 在 2019 年举办，所以 2023 年是第 5 次

没有 NOIP2019

阅读程序 1

容易发现代码是在把 s 中的子串 a 替换成 b，忽略大小写

```
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 判断题

1. 保持 s,b 的字母大小写不变，将 a 里的小写英文字母改写成大写、将大写英文字母改写成小写，输出结果不变。（T）

p,q 忽略大小写，所以依然可以判断子串，输出结果不变

```
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 判断题

2. 每次调用 s.size() 的复杂度是 $O(1)$ 的，但若是把 s.size() 替换成 s.length() 则调用的复杂度将会变成 $O(l)$，其中 l 是 s 当前的长度。这是因为 s.length() 作为 strlen() 函数的 string 版本，会每次重新计算 s 最后一个元素的位置。（F）

s.size() 和 s.length() 都是 $O(1)$ 的

说一大堆话也不代表是对的

```
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 判断题

3. 当输入的字符串 s, a, b 均由小写字母 a 或 b 组成，记 s, a, b 的长度分别为 n, m, k，则程序的时间复杂度最坏为 $O(nm + nk)$（T）

要么判断完 n 次 $O(m)$ 是否含有子串 a，要么加上 n 次 $O(k)$ 的 b

```cpp
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 判断题

4. 当输入的字符串 s, a, b 均由小写字母 a 组成，记 s, a, b 的长度分别为 n, m, k 且有 $n > m > k$，那么上述程序的总复杂度为 $O(n)$。（T）

均由小写字母 a 组成，所以除了最后不够的情况下，每次都会 flag=true

总复杂度为 $O(n)$

```cpp
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 单选题

5. 针对下列输入数据，程序的输出为？（B）

```
National Olympiad in Informatics A154
A
C
```

把所有的 a,A 换成 C，所以输出为

NCtionCl OlympiCd in InformCtics C154

```
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 1 单选题

6. 针对下列输入数据，程序的输出为？（C）

```
abaabaaabaaaabaaaaababababab
aa
ab
```

把所有的 aa 换成 ab，aaa 的情况算遍历顺序靠左的变为 aba，答案为 ababbababababbababababababab

```
#include <iostream>
#include <cstring>
using namespace std;
string s, t;
string a, b;
int main(){
    getline(cin, s);
    getline(cin, a);
    getline(cin, b);
    for(int i = 0;i < s.size();i ++){
        bool flag = true;
        if(i + a.size() <= s.size()){
            for(int j = 0;j < a.size();j ++){
                char p = s[i + j], q = a[j];
                if('a' <= p && p <= 'z') p = p - 'a' + 'A';
                if('a' <= q && q <= 'z') q = q - 'a' + 'A';
                if(p != q)
                    flag = false;
            }
        } else
            flag = false;
        if(flag == true){
            t += b;
            i += a.size() - 1;
        } else
            t += s[i];
    }
    cout << t << endl;
    return 0;
}
```

阅读程序 2

已经告诉了是一种排序算法

可以发现，每次会二分值域中点 mid，将 $> mid$ 的数放在左边，$\leq mid$ 的数放在右边，继承下标后再次递归排序

所以是从大到小排序

```cpp
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 判断题

1. 上述代码实现了一种排序算法，可以将 a 数组按照从小到大的顺序排序。（F）

是从大到小的顺序

```
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 判断题

2. 如果在程序开始之前向 b 数组里写入数据（保证不会发生数组越界），则上述代码的输出不会发生变化。（T）

调用的 b 会先在比较的时候赋值，也只会再次访问相同范围的下标，所以是否有初值不影响

```
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 判断题

3. 若 $n = m$，存在某种数据构造方式，使得上述代码运行的时间复杂度为 $O(n^2)$，这是因为算法本身是对快速排序的改进，但是这种改进不能避免由于对数组的划分不够均等而在极端数据下导致复杂度发生退化。（F）

每一次递归，$[l, r]$ 的长度会减半，所以最坏条件下都是 $O(n \log m)$ 的。

```
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 判断题

4. 如果将①处的 l>=r 条件删除（同时删除 || 使得程序能正常编译运行，下同），程序的时间复杂度不会发生变化；而将 p>q 条件删除，程序在某些数据下的运行效率将会明显降低。（F）

因为是在二分值域，所以数组下标不会有影响，但是值域的 return 条件有

```
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 选择题

5. 不认为 n, m 同阶，即可能出现 n 远大于 m 或 m 远大于 n 的情况。则该程序的最坏时间复杂度为？（D）

每一次递归，$[l, r]$ 的长度会减半，所以最坏条件下复杂度为 $O(n \log m)$

```cpp
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 2 选择题

6. 若输入数据为 10 10 10 4 5 2 2 3 1 5 8 3，那么程序执行到②位置时，b 数组内的值为？（C）

此时 b 数组内跟 a 数组一样，已经是完全排好序后的结果了

```
#include<iostream>
using namespace std;
int a[100005], b[100005], n, m;
void very_quick_sort(int l, int r, int p, int q){
    if(l >= r || p > q){    // ①
        return;
    }
    int mid = (l + r) / 2;
    int p0 = p - 1;
    int q0 = q + 1;
    for(int i = p;i <= q;i ++){
        if(a[i]  > mid) b[++ p0] = a[i];
            else        b[-- q0] = a[i];
    }
    for(int i = p;i <= q;i ++)
        a[i] = b[i];
    very_quick_sort(mid + 1, r, p, p0);
    very_quick_sort(l, mid, q0, q);
}
int main(){
    cin >> n >> m;
    for(int i = 1;i <= n;i ++)
        cin >> a[i];
    very_quick_sort(1, m, 1, n);
        // ②
    for(int i = 1;i <= n;i ++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

阅读程序 3

```
void dfs(int u, int fa){
    d[u] = d[fa] + 1;
    f[u] = fa;

    for(auto &v : e[u]) if(v != fa){
        dfs(v, u);
    }
}
bool cmp(int a, int b){
    return d[a] < d[b];
}
int main(){
    cin >> n >> m >> k;
    for(int i = 2;i <= n;i ++){
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1, 0);                          // ①
    for(int i = 1;i <= m;i ++){
        int x, w;
        cin >> x >> w;
        w1[x] = (w1[x] + w) % mod;
        w2[x] = (w2[x] + w) % mod;
    }
    for(int i = 1;i <= n;i ++)
        id[i] = i;
    sort(id + 1, id + 1 + n, cmp);
    for(int i = 1;i <= k;i ++){
        for(int j = n;j >= 1;j --){
            int x = id[j];
            for(auto &y : e[x]) if(y != f[x]){
                w1[y] = (w1[y] + w1[x]) % mod;
            }
            w1[x] = 0;
        }
        for(int x = 1;x <= n;x ++)
            w1[x] = (w1[x] - w0[x] + mod) % mod,
            w0[x] = 0;
        for(int j = 1;j <= n;j ++){ // ②
            int x = id[j];
            if(f[x]){
                w1[f[x]] = (w1[f[x]] + w2[x]) % mod;
                w2[f[x]] = (w2[f[x]] + w2[x]) % mod;
                w0[x] = (w0[x] + w2[x]) % mod;
                w2[x] = 0;
            }
        }
    }
    for(int i = 1;i <= n;i ++)
        cout << w1[i] << " ";
    return 0;
}
```

阅读程序 3

在完整阅读完程序之后，可以发现：

实质上就是对所有距离恰好为 k 的点加上 x，一共 m 组操作

代码中 w1[x] 对所有子节点更新，

w2[x] 对父节点更新，而 w0[x] 的作用是可以处理实际距离为 k，防止走回来

```
    for(int i = 1;i <= k;i ++){
        for(int j = n;j >= 1;j --){
            int x = id[j];
            for(auto &y : e[x]) if(y != f[x]){
                w1[y] = (w1[y] + w1[x]) % mod;
            }
            w1[x] = 0;
        }
        for(int x = 1;x <= n;x ++)
            w1[x] = (w1[x] - w0[x] + mod) % mod,
            w0[x] = 0;
        for(int j = 1;j <= n;j ++){ // ②
            int x = id[j];
            if(f[x]){
                w1[f[x]] = (w1[f[x]] + w2[x]) % mod;
                w2[f[x]] = (w2[f[x]] + w2[x]) % mod;
                w0[x] = (w0[x] + w2[x]) % mod;
                w2[x] = 0;
            }
        }
    }
    for(int i = 1;i <= n;i ++)
        cout << w1[i] << " ";
    return 0;
}
```

阅读程序 3 判断题

1. 如果更改 ① 处 `dfs(1,0)` 为 `dfs(n,0)`，则输出结果可能有变化。（F）

不会有变化

因为有 w0[x] 的存在，根节点并没有影响

依然计算出深度然后排序

```
int main(){
    cin >> n >> m >> k;
    for(int i = 2;i <= n;i ++){
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    dfs(1, 0);                          // ①
    for(int i = 1;i <= m;i ++){
        int x, w;
        cin >> x >> w;
        w1[x] = (w1[x] + w) % mod;
        w2[x] = (w2[x] + w) % mod;
    }
```

阅读程序 3 判断题

2. 如果 $k = n$，那么输出结果均为 0。（T）

实质上就是对所有距离恰好为 k 的点加上 x，一共 m 组操作

不存在恰好为 n 的点，所以所有点的权值都为 0

```
for(int i = 1;i <= k;i ++){
    for(int j = n;j >= 1;j --){
        int x = id[j];
        for(auto &y : e[x]) if(y != f[x]){
            w1[y] = (w1[y] + w1[x]) % mod;
        }
        w1[x] = 0;
    }
    for(int x = 1;x <= n;x ++)
        w1[x] = (w1[x] - w0[x] + mod) % mod,
        w0[x] = 0;
    for(int j = 1;j <= n;j ++){ // ②
        int x = id[j];
        if(f[x]){
            w1[f[x]] = (w1[f[x]] + w2[x]) % mod;
            w2[f[x]] = (w2[f[x]] + w2[x]) % mod;
            w0[x] = (w0[x] + w2[x]) % mod;
            w2[x] = 0;
        }
    }
}
```

阅读程序 3 判断题

3. 如果更改 ② 处 `for(int j=1;j<=n;j++)` 为 `for(int j=n;j>=1;j--)`，那么对于任意合法的输入数据，更改前后程序的输出均相同。（F）

排序好的顺序是按照深度从小到大更新父节点，这样才能保证只更新一次

如果按照深度从大到小，那么一个点将对该点到根节点链上的所有点都更新

```
bool cmp(int a, int b){
    return d[a] < d[b];
}
```

```
for(int j = 1;j <= n;j ++){ // ②
    int x = id[j];
    if(f[x]){
        w1[f[x]] = (w1[f[x]] + w2[x]) % mod;
        w2[f[x]] = (w2[f[x]] + w2[x]) % mod;
        w0[x] = (w0[x] + w2[x]) % mod;
        w2[x] = 0;
    }
}
```

阅读程序 3 单选题

4. 该程序的时间复杂度为？（B）

复杂度瓶颈在于循环枚举 k 次

每一次都是对所有点 $O(n)$ 更新

而初始处理所有修改是 $O(m)$ 的

于是总复杂度为 $O(nk+m)$

```
for(int i = 1;i <= k;i ++){
    for(int j = n;j >= 1;j --){
        int x = id[j];
        for(auto &y : e[x]) if(y != f[x]){
            w1[y] = (w1[y] + w1[x]) % mod;
        }
        w1[x] = 0;
    }
    for(int x = 1;x <= n;x ++)
        w1[x] = (w1[x] - w0[x] + mod) % mod,
        w0[x] = 0;
    for(int j = 1;j <= n;j ++){ // ②
        int x = id[j];
        if(f[x]){
            w1[f[x]] = (w1[f[x]] + w2[x]) % mod;
            w2[f[x]] = (w2[f[x]] + w2[x]) % mod;
            w0[x] = (w0[x] + w2[x]) % mod;
            w2[x] = 0;
        }
    }
}
```

阅读程序 3 单选题

5. 如果输入数据为：

```
5 2 1
1 2
2 3
3 4
3 5
1 5
3 2
```

则程序的输出应该是：（A）

有 $m=2, k=1$，1 距离为 1 的点是 2，3 距离为 1 的点是 2 4 5

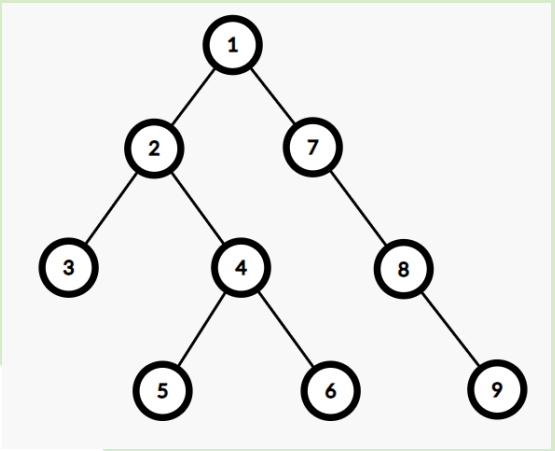
所以输出为 0 7 0 2 2

阅读程序 3 单选题

6. 如果输入数据为：

```
9 9 2
1 2
1 7
2 3
2 4
7 8
4 5
4 6
8 9
```

```
1 1
2 10
3 100
4 1000
5 10000
6 100000
7 1000000
8 10000000
9 100000000
```



那么输出数据应该为：（A）

画图算一下，每个点距离为 2 的点是哪几个就好

完善程序 1

给定序列 a_n，求有多少对 (i,j) 满足 $a_i < a_j$，$n \le 10^6$，$a_i \le 10^9$。

对于任意的 $a_i \neq a_j$，我们发现 (i,j) 或 (j,i) 能对答案产生 1 的贡献，因此我们只需要用总的对数减去 $a_i = a_j$ 的 (i,j) 数量就能得到答案

用二分来完成，先对 a_n 排序，并计算 a_n 的前缀和 s_n

然后对于每一个值，二分相同元素组成的最大区间，然后计算对答案的贡献

完善程序 1

1. ① 处应当填入（A）

$n \le 10^6$，所以数组大小也要开到 1e6+7

A. 1e6+7

B. 1000000

C. 1e6

D. 1000000000000

```
#include<bits/stdc++.h>
using i64 = long long;
const int Maxn = ①;
int n, a[Maxn]; i64 s[Maxn], ans;
bool check(int l, int r) {
    if (s[r] - s[l - 1] == ②) return true;
    return false;
}
```

完善程序 1

2. ② 处应当填入（C）

s 代表前缀和，所以检查区间 $[l,r]$ 内元素是否相等应该是 $s_r - s_{l-1} = (r-l+1)a_l$

注意 long long

A. (r - l + 1) * a[l]

B. s[r] - s[l - 1]

C. 1ll * (r - l + 1) * a[l]

D. 0

```
#include<bits/stdc++.h>
using i64 = long long;
const int Maxn = ①;
int n, a[Maxn]; i64 s[Maxn], ans;
bool check(int l, int r) {
    if (s[r] - s[l - 1] == ②) return true;
    return false;
}
```

完善程序 1

3. ③ 处应当填入（ D ）

ans 的初值应该是所有的对数

所以是 $\frac{n(n-1)}{2}$，注意 long long

A. n * (n + 1) / 2

B. 1ll * n * (n + 1) / 2

C. n * (n - 1) / 2

D. 1ll * n * (n - 1) / 2

```
for (int i = 1; i <= n; i ++) s[i] = s[i - 1] + a[i];
ans = ③;
for (int i = 1; i <= n;) {
    int l = i, r = n, pos = n;
    while (④) {
        int mid = (l + r) >> 1;
        if(check(l, mid)) l = mid + 1, pos = mid;
        else r = mid - 1;
    } ans -= ⑤;
    i = pos + 1;
} std :: cout << ans;
```

完善程序 1

4. ④ 处应当填入（ B ）

l=mid+1,r=mid-1

很常规的二分条件

A. l < r

B. l <= r

C. l <= r + 1

D. l > r

```
for (int i = 1; i <= n; i ++) s[i] = s[i - 1] + a[i];
ans = ③;
for (int i = 1; i <= n;) {
    int l = i, r = n, pos = n;
    while (④) {
        int mid = (l + r) >> 1;
        if(check(l, mid)) l = mid + 1, pos = mid;
        else r = mid - 1;
    } ans -= ⑤;
    i = pos + 1;
} std :: cout << ans;
```

完善程序 1

5. ⑤ 处应当填入（ D ）

二分出来的区间是 $[i, pos]$，内部的对数是 $\frac{(pos-i+1)(pos-i)}{2}$

A. (pos - l + 1) * (pos - l) / 2

B. 1ll * (pos - l + 1) * (pos - l) / 2

C. (pos - i + 1) * (pos - i) / 2

D. 1ll * (pos - i + 1) * (pos - i) / 2

```
for (int i = 1; i <= n; i ++) s[i] = s[i - 1] + a[i];
ans = ③;
for (int i = 1; i <= n;) {
    int l = i, r = n, pos = n;
    while (④) {
        int mid = (l + r) >> 1;
        if(check(l, mid)) l = mid + 1, pos = mid;
        else r = mid - 1;
    } ans -= ⑤;
    i = pos + 1;
} std :: cout << ans;
```

完善程序 2

给定序列 a_n，求其所有子区间异或和的和，其中 $n \le 10^5$，$0 \le a_i \le 10^9$。

思路：我们对每一位独立计算，对右端点扫描线，并用异或前缀和辅助统计。

所有子区间异或和的和，等价于异或前缀和两两异或和的和

拆位，扫一遍用桶记录 0 和 1 的个数，与当前数当前位不同的个数乘上位的大小，就是当前数贡献的答案

完善程序 2

1. ① 处应当填入（ B ）

维护异或前缀和，所以是异或上前一个 a[i]

A. a[i-1]^=a[i]

B. a[i]^=a[i-1]

C. a[i-1]+=a[i]

D. a[i]+=a[i-1]

```
int n, a[N], cnt[2]; i64 ans;
int main () {
    cin >> n;
    for (int i = 1; i <= n; i ++) cin >> a[i], ①;
    for (int bit = ②; ③; bit --) {
        cnt[0] = cnt[1] = 0;
        for (int i = 0; i <= n; i ++) {
            cnt[④] ++;
            ans += ⑤;
        }
    } cout << ans;
}
```

完善程序 2

2. ② 处应当填入（B）

数据范围是 $10^9 < 2^{30}$

所以一共有 30 位，开到 bit=29

A. n-1

B. 29

C. n

D. n+1

```
int n, a[N], cnt[2]; i64 ans;
int main () {
    cin >> n;
    for (int i = 1; i <= n; i ++) cin >> a[i], ①;
    for (int bit = ②; ③; bit --) {
        cnt[0] = cnt[1] = 0;
        for (int i = 0; i <= n; i ++) {
            cnt[④] ++;
            ans += ⑤;
        }
    } cout << ans;
}
```

完善程序 2

3. ③ 处应当填入（ D ）

从 29 取到 0，共 30 位

bit>=0 等价于 ~bit

A. bit

B. bit>=n

C. bit-1

D. ~bit

```
int n, a[N], cnt[2]; i64 ans;
int main () {
    cin >> n;
    for (int i = 1; i <= n; i ++) cin >> a[i], ①;
    for (int bit = ②; ③; bit --) {
        cnt[0] = cnt[1] = 0;
        for (int i = 0; i <= n; i ++) {
            cnt[④] ++;
            ans += ⑤;
        }
    } cout << ans;
}
```

完善程序 2

4. ④ 处应当填入（A）

统计当前位应该是 A

选项 B,C 返回的不是 0/1

顺序从高到低或者从低到高无所谓

A. (a[i] >> bit) & 1

B. a[i] & (1 << bit)

C. a[i] & (1ll << bit)

D. (a[i] >> bit) ^ 1

```
int n, a[N], cnt[2]; i64 ans;
int main () {
    cin >> n;
    for (int i = 1; i <= n; i ++) cin >> a[i], ①;
    for (int bit = ②; ③; bit --) {
        cnt[0] = cnt[1] = 0;
        for (int i = 0; i <= n; i ++) {
            cnt[④] ++;
            ans += ⑤;
        }
    } cout << ans;
}
```

完善程序 2

5. ⑤ 处应当填入（B）

```
for (int bit = ②; ③; bit --) {
    cnt[0] = cnt[1] = 0;
    for (int i = 0; i <= n; i ++) {
        cnt[④] ++;
        ans += ⑤;
    }
} cout << ans;
```

当前位的大小是 1 << bit

计算的桶应该是 (a[i] >> bit) & 1 ^ 1

记得 long long

A. cnt[(a[i] >> bit) & 1] << i)

B. 1ll * cnt[(a[i] >> bit) & 1 ^ 1] << bit

C. 1ll * cnt[(a[i] >> bit) & 1] << bit

D. cnt[(a[i] >> bit) & 1 ^ 1] << i