2026年8月 SACCC 五级题解

苏州市人工智能学会 · 青少年人工智能核心算法素养考核（SACCC）

级别：五级 期次：2026年8月

题目清单

顺序	题目	主要考点
1	阶乘计算	高精度乘法、阶乘
2	训练编号数论统计	最大公因数、素数判断
3	课程排期压峰值	二分答案、贪心检查
4	任务树巡检	建树、BFS、DFS 前序遍历

第 1 题 阶乘计算

主要考点： 高精度乘法、阶乘

题意概括

计算 n! = 1 × 2 × … × n 的完整十进制结果。由于 n 最大为 1000，结果远超普通整数类型的表示范围。

核心思路

用十进制字符串保存高精度整数，从 1 开始依次乘以 2 到 n。字符串乘法先把每一对数位的乘积累加到对应位置，再从低位向高位统一处理进位，最后跳过前导零并转换回字符串。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

string multiplyString(const string &a, const string &b) {
    if (a == "0" || b == "0") return "0";

    vector<int> result(a.size() + b.size(), 0);
    for (int i = (int)a.size() - 1; i >= 0; --i) {
        for (int j = (int)b.size() - 1; j >= 0; --j) {
            int pos = i + j + 1;
            // 先把所有数位乘积累加到对应位置，稍后统一进位。
            result[pos] += (a[i] - '0') * (b[j] - '0');
        }
    }

    // 从低位向高位处理十进制进位。
    for (int i = (int)result.size() - 1; i > 0; --i) {
        result[i - 1] += result[i] / 10;
        result[i] %= 10;
    }

    string ans;
    int start = 0;
    while (start + 1 < (int)result.size() && result[start] == 0) ++start;
    for (int i = start; i < (int)result.size(); ++i) {
        ans.push_back(char('0' + result[i]));
    }
    return ans;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

    string ans = "1";
    for (int i = 2; i <= n; ++i) {
        ans = multiplyString(ans, to_string(i));
    }

    cout << ans << '\n';
    return 0;
}
```

第 2 题 训练编号数论统计

主要考点： 最大公因数、素数判断

题意概括

给定若干个正整数，需要求出所有数的整体最大公因数，同时统计其中有多少个数本身是素数。

核心思路

整体最大公因数可以从 g=0 开始，依次令 g=gcd(g,x)。判断素数时，只需尝试从 2 到 sqrt(x) 的可能因子；如果存在整除因子就不是素数，扫描结束仍未找到因子则是素数。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

bool isPrime(int x) {
    if (x < 2) return false;
    // 若 x 是合数，至少有一个因子不超过 sqrt(x)。
    for (int d = 2; 1LL * d * d <= x; ++d)
        if (x % d == 0) return false;
    return true;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    int g = 0, primeCount = 0;
    for (int i = 0; i < n; ++i) {
        int x;
        cin >> x;
        // gcd(0, x) = x，因此 g 可以从 0 开始累计。
        g = std::gcd(g, x);
        if (isPrime(x)) ++primeCount;
    }
    cout << g << ' ' << primeCount << '\n';
    return 0;
}
```

第 3 题 课程排期压峰值

主要考点： 二分答案、贪心检查

题意概括

把保持原顺序的连续课程任务分配到不超过 k 天，每个任务不能拆分。目标是让每天工作量的最大值尽可能小，并输出这个最小值。

核心思路

答案具有单调性：若限制 limit 可以完成，那么任何更大的限制也可以完成。固定 limit 后，从左到右贪心地把尽可能多的连续任务放在当天，放不下时再开启新的一天，这样使用的天数最少。于是可以在“最大单个任务量”到“全部任务量之和”之间二分最小可行限制。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

bool canFinish(const vector<int> &a, int k, long long limit) {
    int days = 1;
    long long cur = 0;
    for (int x : a) {
        // 当前任务放不下时必须换到下一天。
        if (cur + x > limit) {
            ++days;
            cur = x;
        } else {
            cur += x;
        }
    }
    return days <= k;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    long long left = 0, right = 0;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        left = max(left, (long long)a[i]);
        right += a[i];
    }
    // 可行性关于 limit 单调，二分寻找最小可行值。
    while (left < right) {
        long long mid = (left + right) / 2;
        if (canFinish(a, k, mid)) right = mid;
        else left = mid + 1;
    }
    cout << left << '\n';
    return 0;
}
```

第 4 题 任务树巡检

主要考点： 建树、BFS、DFS 前序遍历

题意概括

根据 2 号到 n 号节点各自的父节点建立一棵以 1 为根的树，并分别输出从根开始的广度优先遍历顺序和深度优先前序遍历顺序。

核心思路

用孩子表保存树：读到节点 i 的父节点 p 时，把 i 加入 p 的孩子序列。节点按编号递增读入，因此每个孩子序列天然按编号递增。广度优先遍历使用队列逐层访问；深度优先前序遍历先记录当前节点，再按顺序递归访问每个孩子。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> childNodes;
vector<int> dfsOrder;

void dfs(int u) {
    // 前序遍历先访问当前节点，再依次访问孩子。
    dfsOrder.push_back(u);
    for (int v : childNodes[u]) dfs(v);
}

void printOrder(const vector<int> &order) {
    for (int i = 0; i < (int)order.size(); ++i) {
        if (i) cout << ' ';
        cout << order[i];
    }
    cout << '\n';
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    childNodes.assign(n + 1, {});
    for (int i = 2; i <= n; ++i) {
        int p;
        cin >> p;
        // i 递增读入，所以同一父节点的孩子已经按编号升序保存。
        childNodes[p].push_back(i);
    }
    vector<int> bfsOrder;
    queue<int> q;
    q.push(1);
    while (!q.empty()) {
        // 队列保证节点按层次从近到远访问。
        int u = q.front();
        q.pop();
        bfsOrder.push_back(u);
        for (int v : childNodes[u]) q.push(v);
    }
    dfs(1);
    printOrder(bfsOrder);
    printOrder(dfsOrder);
    return 0;
}
```