# 2021 天梯赛题解

赵长乐 王小川

2021 年 4 月 30 日

# 目录

# L1-3 强迫症

如果要以整数的格式读入，需要考虑有前导 0 的情况，所以考试时我采用字符串格式读入然后处理。不过这道题可以直接以数字格式读入，程序更加简洁，参考解法 2。

```
#include<iostream>

using namespace std;

int main()
{
    string str;
    cin >> str;

    if(str.size() == 6)
    cout << str.substr(0, 4) << "-" << str.substr(4)
   << endl;
    else
    {
        int y = stoi(str.substr(0, 2));
        if(y < 22) y = y + 2000;
        else y = y + 1900;
        cout << y << "-" << str.substr(2) << endl;
    }

    return 0;
}
```

侯珩乐同学的写法：

```
#include<bitsdc++.h>
using namespace std;
int main()
{
    int a;
```

```
    cin>>a;
    if(a/10000)
        printf("%04d-%02d",a/100,a%100);
    else if(a/100<22)
        printf("20%02d-%02d",a/100,a%100);
    else
        printf("19%02d-%02d",a/100,a%100);
    return 0;
}
```

## L1-6 吉老师的回归

考察点:

字符串

注意 *string* 的 *find* 函数返回值为 *string* :: *npos* 表示没有找到

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cstring>
using namespace std;
int n, m;
string str;
int main()
{
    cin>>n>>m;
    getchar();
    while (n--)
    {
        getline(cin, str);
        if (str.find("qiandao") == string::npos && str
   .find("easy") == string::npos)
```

```
        m--;
        if (m < 0)
        {
            cout<<str;
            return 0;
        }
    }
    cout<<"Wo AK le";
    return 0;
}
```

## L1-7 天梯赛的善良

考察点：

map 的使用

```
#include<cstring>
#include<iostream>
#include<map>

using namespace std;

int main()
{
    int n;
    map<int, int> m;

    cin >> n;

    while(n--)
    {
        int x;
        cin >> x;
```

```
        m[x]++;
    }

    auto minn = m.begin(), maxx = --m.end();

    cout << minn->first << " " << minn->second << endl
   ;
    cout << maxx->first << " " << maxx->second << endl
   ;

    return 0;
}
```

## L1-8 乘法口诀数列

有两个细节：[1]

- 两个一位数相乘的结果最大只有两位数
- 在计算结果数组时，用 $i$ 记录计算乘积的位置，用 $p$ 记录数组当前可以放元素的位置。类似双指针算法。

```
#include<bits/stdc++.h>

using namespace std;

const int N = 1010;

int a[N];

int main()
{
    int n;
```

[1]参考：https://blog.csdn.net/qq_45799024/article/details/116097959

```
    cin >> a[1] >> a[2] >> n;

    int p = 3;
    for(int i = 1; p <= n; ++i)
    {
        int t = a[i] * a[i + 1];
        if(t >= 10)
        {
            a[p++] = t / 10;
            a[p++] = t % 10;
        }
        else
            a[p++] = t;
    }

    for(int i = 1; i <= n; ++i)
    {
        printf(" %d" + !(i - 1), a[i]);
    }

    return 0;
}
```

## L2-1 包装机

考察点：

模拟，栈，队列

```
#include<bits/stdc++.h>

using namespace std;

const int N = 110;
```

```
int n, m, sz;
int fronts[N];
string tracks[N];
string ans;
stack<char> stk;

int main()
{
    cin >> n >> m >> sz;

    string good;
    for(int i = 1; i <= n; ++i)
    {
        cin >> tracks[i];
    }

    int x;
    while(cin >> x, x != -1)
    {
        if(x != 0)
        {
            if(fronts[x] != m)
            {
                if(stk.size() == sz)
                {
                    ans += stk.top();
                    stk.pop();
                }
                stk.push(tracks[x][fronts[x]++]);
            }
        }
        else
```

```
        {
            if(stk.size())
            {
                ans += stk.top();
                stk.pop();
            }
        }
    }

    cout << ans << endl;

    return 0;
}
```

## L2-2 病毒溯源!

考察点:

搜索

由于 $bfs$ 时，每个顶点最多入队一次所以 $bfs$ 的时间复杂度为: $O(n)$(n 是病毒数量。) 总体时间复杂度为 $O(nlogn)$ ，n 为边的数量，因为题目说明

每一种病毒都是由唯一的一种病毒突变而来，并且不存在循环变异的情况

所以边的数量为 $n-1$

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cstring>
#include<queue>
#include<stack>
```

```
using namespace std;

#define N 10004

int n; // 0-(n-1)
vector<int> edge[N];
bool into[N]; // 入度，变异链起点入度为 0
int root;
int pre[N]; // 记录顶点是否被访问过（没访问过是 -1）
   如果被访问过，记录是从哪个顶点走来的
int len[N]; // 记录变异链的长度
int maxx, loop; // 记录变异链最大长度 和 变异链的末尾

void update(int point)
{
    if (len[point] > maxx)
    {
        maxx = len[point];
        loop = point;
    }
}

void bfs(int src)
{
    queue<int> que;
    que.push(src);
    len[src] = 1;
    update(src);

    while (!que.empty())
    {
        int now = que.front();
        que.pop();
```

```
        for (auto nxt : edge[now])
        if (pre[nxt] == -1)
        {
            que.push(nxt);
            len[nxt] = len[now] + 1;
            // nxt 从 now 转移而来(题目已经说明 "每一种病毒都是由唯一的一种病毒突变而来, 并且不存在循环变异的情况")
            pre[nxt] = now;
            update(nxt);
        }
    }
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int k;
        cin>>k;
        while (k--)
        {
            int tmp;
            cin>>tmp;
            edge[i].push_back(tmp);
            into[tmp] = true;
        }
        // 按编号从小到大排序
        sort(edge[i].begin(), edge[i].end());
    }

    for (int i = 0; i < n; i++)
```

```
    {
        pre[i] = -1;
        if (!into[i])
        root = i;
    }

    bfs(root);

    cout << maxx << endl;

    stack<int> stk;
    int now = loop;
    while (now != -1)
    {
        stk.push(now);
        now = pre[now];
    }

    for (int i = 0; i < maxx; i++)
    {
        if (i)
        cout<<' ';
        cout<<stk.top();
        stk.pop();
    }

    return 0;
}
```

## L2-3 清点代码库

考察点：

map，哈希

使用 $map$ 构建 $alls$ 时间复杂度为 $O(nlogn)$；使用 $unorderd\_map$ 构建 $alls$ 时间复杂度为 $O(n)$。

无论是哪一种构建方式最后都需要排序,所以最终时间复杂度为:$O(nlogn)$

```
#include<bits/stdc++.h>

using namespace std;

const int N = 10010;

struct ANS
{
    int cnt;
    vector<int> v;

    bool operator<(ANS& a) const
    {
        if(cnt != a.cnt) return cnt > a.cnt;

        for(int i = 0; i < v.size(); ++i)
            if(a.v[i] != v[i])
                return v[i] < a.v[i];
    }
}ans[N];

int n, m;
map<vector<int>, int> alls;

int main()
{
    cin >> n >> m;

    while(n--)
```

```
    {
        vector<int> v;
        for(int i = 0; i < m; ++i)
        {
            int x;
            cin >> x;
            v.push_back(x);
        }
        alls[v]++;
    }

    int sz = 0;
    for(auto [v, cnt] : alls)
    {
        ans[sz].v = v;
        ans[sz].cnt = cnt;
        ++sz;
    }

    sort(ans, ans + sz);

    cout << sz << endl;

    for(int i = 0; i < sz; ++i)
    {
        cout << ans[i].cnt;
        for(int j = 0; j < m; ++j)
        printf(" %d", ans[i].v[j]);
        cout << endl;
    }

    return 0;
}
```

在乐子哥的指导下写出了哈希表的写法：

```
#include<bits/stdc++.h>

using namespace std;

const int N = 10010;

struct ANS
{
    int cnt;
    vector<int> v;

    bool operator<(ANS& a) const
    {
        if(cnt != a.cnt) return cnt > a.cnt;

        for(int i = 0; i < v.size(); ++i)
            if(a.v[i] != v[i])
                return v[i] < a.v[i];
    }
}ans[N];

struct MyHash
{
    size_t operator()(const vector<int>& v) const
    {
        size_t res = 0;
        for(auto e : v)
            res ^= e;
        return res;
    }
};
```

```
int n, m;
unordered_map<vector<int>, int, MyHash> alls;

int main()
{
    cin >> n >> m;

    while(n--)
    {
        vector<int> v;
        for(int i = 0; i < m; ++i)
        {
            int x;
            cin >> x;
            v.push_back(x);
        }
        alls[v]++;
    }

    int sz = 0;
    for(auto [v, cnt] : alls)
    {
        ans[sz].v = v;
        ans[sz].cnt = cnt;
        ++sz;
    }

    sort(ans, ans + sz);

    cout << sz << endl;

    for(int i = 0; i < sz; ++i)
    {
```

```
        cout << ans[i].cnt;
        for(int j = 0; j < m; ++j)
            printf(" %d", ans[i].v[j]);
        cout << endl;
    }

    return 0;
}
```

另一种哈希函数 (较慢)：

```
struct MyHash
{
    size_t operator()(const vector<int>& v) const
    {
        string s;
        for(auto e : v)
            s += to_string(e);
        return hash<string>{}(s);
    }
};
```

## L2-4 哲哲打游戏!

考察点：

模拟

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cstring>

using namespace std;
```

```cpp
#define N 100005

int n, m;
int save[102];
vector<int> edge[N];

int main()
{
    cin.tie(0);
    cin.sync_with_stdio(0);

    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        int k;
        cin>>k;
        while (k--)
        {
            int tmp;
            cin>>tmp;
            edge[i].push_back(tmp);
        }
    }

    int now = 1;
    while (m--)
    {
        int op, to;
        cin >> op >> to;
        if (!op)
            now = edge[now][to - 1];
        else if (op == 1)
```

```
        {
            cout << now << endl;
            save[to] = now;
        }
        else
            now = save[to];
    }

    cout << now;

    return 0;
}
```

## L3-1 森森旅游

考察点：

最短路

记录城市 1 到其他城市需要花费的最少现金 (1 到 $i$ 的花费记为 $from1[i]$ )，然后记录 $n$ 到其他城市需要花费的最少旅游金 ($n$ 到 $i$ 的花费记为 $fromn[i]$)。设 $a[i]$ 为在第 $i$ 个城市的汇率，则在城市 $i$ 兑换旅游金的所需携带最少现金为

$$from1[i] + \lceil \frac{fromn[i]}{a[i]} \rceil$$

(注意向上取整)

如果对于每次汇率调整，都重新遍历每个城市的汇率进行计算，时间复杂度为 $O(nq)$ 显然会超时。可以采用 $multiset$ 来优化每次汇率调整。对于每次汇率调整，删除之前汇率计算的值，加入新的汇率计算的值，最小值即为 $*multiset.begin()$。时间复杂度为 $O(qlogn)$

参考：https://www.bilibili.com/read/cv11044423/

## L3-2 还原文件

考察点：

DFS

可以直接暴搜