第二周直播课（上午）
作业讲解

洛谷网校
基础衔接提高计划
2025-02
Maxmilite

课前提示

- 上课的时候专心听讲解，**不要跟着老师抄代码**，下课后独立完成。
- 不使用 AI 做题，AI 会做不等于自己会。
- 不抄袭题解（含对照题解抄一遍），抄对不等于会做。

  看完题解后，关闭题解独立练习。

  练习中途遇到问题，应当分析题目及自己的思路，而非回忆题解或再次参考题解。

图的存储与出边的排序

题目链接 https://www.luogu.com.cn/problem/B3613

给定一个 n 个点 m 条边的有向图 G，结点编号从 1 至 n。

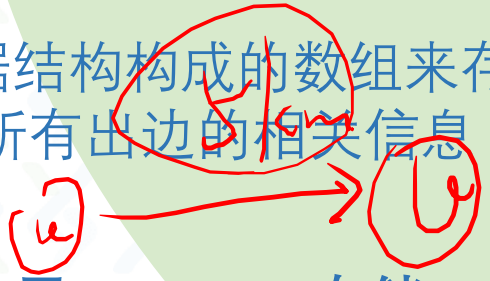
对于 $u = 1, 2, 3, \ldots, n$，依次完成如下要求：

对于 u 的所有出边（即从 u 出发的边），按照从小到大的顺序输出出边所指向的节点编号。

依次完成的含义是，先按顺序输出 $u = 1$ 的出边所指向的点的编号，再按顺序输出 $u = 2$ 的出边所指向的点的编号……最后按顺序输出 $u = n$ 的出边所指向的点的编号。

邻接表

使用一个支持动态添加元素的数据结构构成的数组来存边，例如 vector。存储内容为与一个点的所有出边的相关信息（终点、边权等）。



仅介绍并**强烈推荐在一般情况下使用 vector 存储**。

```cpp
struct Edge { int to, val; Edge(int a, int b) { to = a, val = b; } };
/* to 表示终点，val 表示边权 */
vector<Edge> e2[105];
for (int i = 1; i <= m; ++i) {
    int u, v; cin >> u >> v;
    e2[u].push_back(Edge(v, 0));
    e2[v].push_back(Edge(u, 0));
}
for (int i = 1; i <= m; ++i) {
    int u, v, w; cin >> u >> v >> w;
    e2[u].push_back(Edge(v, w));
}
```

图的存储与出边的排序

思路

使用邻接表存图后，对每个 vector 单独排序即可。

形如 `sort(g[i].begin(), g[i].end);`

图的存储与出边的排序

代码实现

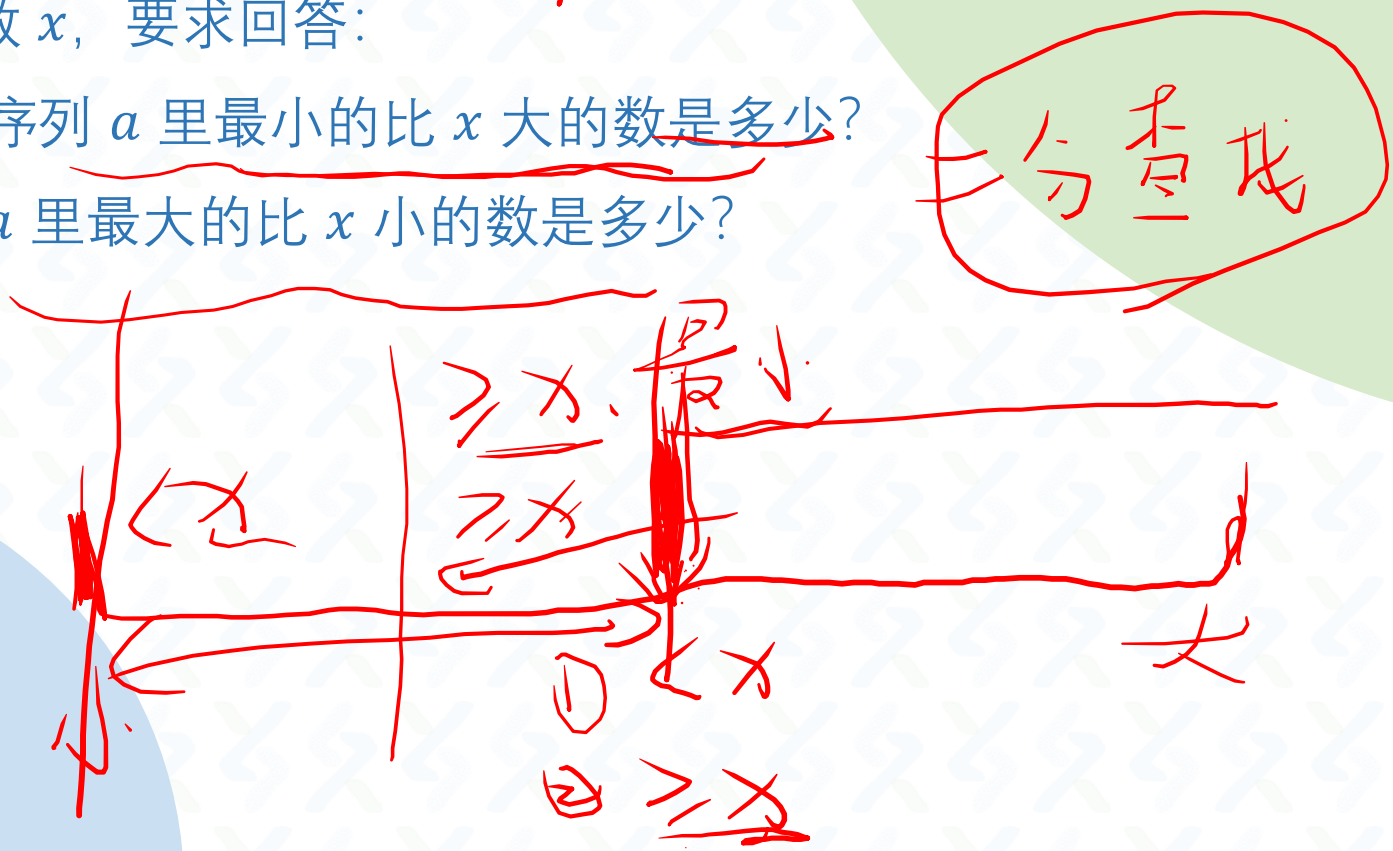
```
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        g[i].clear();
    }
    for (int i = 1; i <= m; ++i) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    for (int i = 1; i <= n; ++i) {
        sort(g[i].begin(), g[i].end());
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < g[i].size(); ++j) {
            cout << g[i][j] << " ";
        }
        cout << '\n';
    }
}
```

eqal_range

题目链接 https://www.luogu.com.cn/problem/B4362

给定数列 $a = [a_1, a_2, \ldots, a_n]$，有 q 次询问，每次询问给出一个整数 x，要求回答：

- 序列 a 里最小的比 x 大的数是多少？
- a 里最大的比 x 小的数是多少？



eqal_range

思路

二分模板题，将整个序列由小到大排序后二分即可。

规定 $l=1, r=n$，每次二分 $\text{mid}=\frac{l+r}{2}$。二分判定规则：

- 序列 a 里最小的比 x 大的数：

如果 $\text{mid}<x$ 或 $\text{mid}=x$，不是我们要找的答案，向右找；

如果 $\text{mid}>x$，是答案，记录，但是为让答案更小，向左找。

- 序列 a 里最大的比 x 小的数：

如果 $\text{mid}>x$ 或 $\text{mid}=x$，不是我们要找的答案，向左找；

如果 $\text{mid}<x$，是答案，记录，但是为让答案更大，向右找。

eqal_range

代码实现

```
int find1(int x) {
    int l = 1, r = n, ans = -1;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (a[mid] > x) {
            ans = a[mid]; r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
    return ans;
}
```

```
int find2(int x) {
    int l = 1, r = n, ans = -1;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (a[mid] < x) {
            ans = a[mid]; l = mid + 1;
        } else {
            r = mid - 1;
        }
    }
    return ans;
}
```

eqal_range

代码实现

```
void solve() {
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    sort(a + 1, a + n + 1);
    while (q--) {
        int x;
        cin >> x;
        cout << find1(x) << " " << find2(x) << '\n';
    }
}
```

【第二周】孤独摇滚

题目链接 https://www.luogu.com.cn/problem/T635881

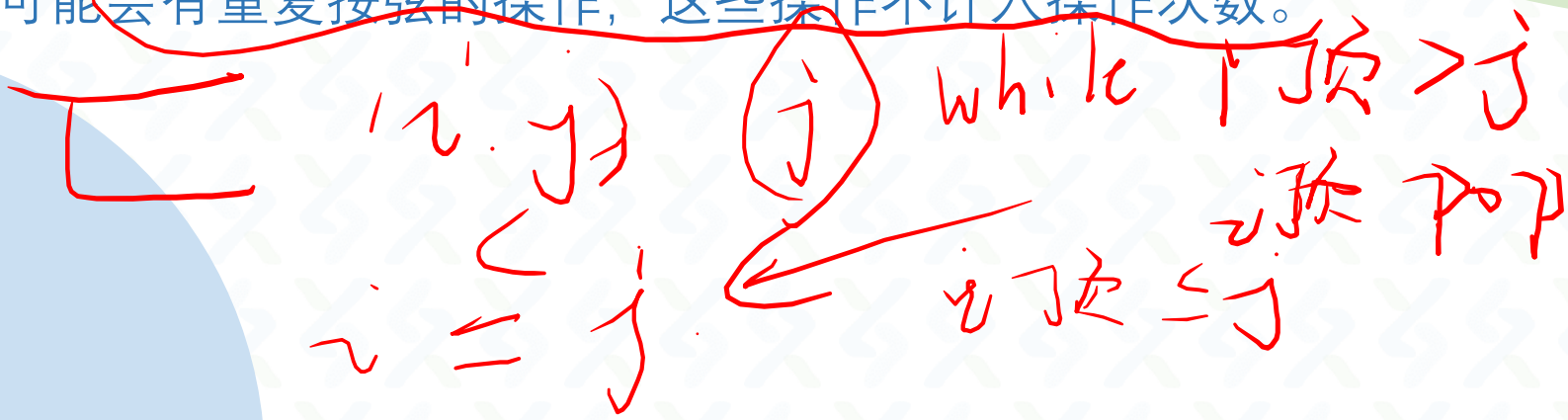
六根弦的吉他，每根弦被分为 P 段。

有 n 次按弦，每次按弦的位置在 (i,j)，依次代表弦号和段号。

如果在某一次按弦时，第 i 根弦上有 $>j$ 的段在之前被按住，那么需要把这些段松开。

按弦和松弦都需要一次操作次数，求总的操作次数。

可能会有重复按弦的操作，这些操作不计入操作次数。



【第二周】孤独摇滚

解析

注意到每次按弦时，需要将 j 位置及之后的，之前已经按下的弦松开。

如果我们将某一根线上的按弦信息从小到大排序，那么不难发现所有的操作都可以基于栈进行。

我们使用 6 个栈存储按弦信息。每次按弦时，将对应位置的栈进行检查，当栈顶元素大于当前按弦段数时，进行弹出，直至栈顶元素小于当前段数。

由于需要去重，使用一个 `vis[][]` 数组进行标记。`vis[i][j]` 可以代表第 `i` 根弦的第 `j` 段是否被按下。

【第二周】孤独摇滚

代码实现

```
stack<int> s[7];
int vis[300005][7];
int n, p;

cin >> n >> p;
int ans = 0;
for (int i = 1; i <= 6; ++i)
    s[i].push(0);
for (int i = 1; i <= n; ++i) {
    int x, y;
    cin >> x >> y;
    while (s[x].top() > y) {
        vis[s[x].top()][x] = 0;
        s[x].pop(), ++ans;
    }
    if (s[x].top() < y)
        s[x].push(y), ++ans, vis[y][x] = 1;
}
cout << ans << endl;
```

【第二周】小粉兔家族

题目链接 https://www.luogu.com.cn/problem/T635882

小粉兔家族里共有 n 只兔子，第 i 只兔子的名字是 s_i。

族长按顺序共点了 m 个名字 $t_1, t_2, \ldots t_m$。对于每个 t_i，

- 如果 t_i 第一次被点到且 t_i 真的是某只兔子的名字，输出 OK；
- 如果 t_i 不是任何一只兔子的名字，输出 WRONG；
- 如果 t_i 在之前被点到过（且确实是某只兔子的名字），输出 REPEAT。

map<string, int> m
每个字符串对应一个数

【第二周】小粉兔家族

解析

可以使用 map 来标记姓名。

建立 map<string, int> mp,

- 用 mp[s] = 1 表示有此兔，未点名；
- 用 mp[s] = 2 表示有此兔，已点名；
- 用 mp[s] = 0（默认值）表示没有此兔，或用 mp.count(s) == 0；

【第二周】小粉兔家族

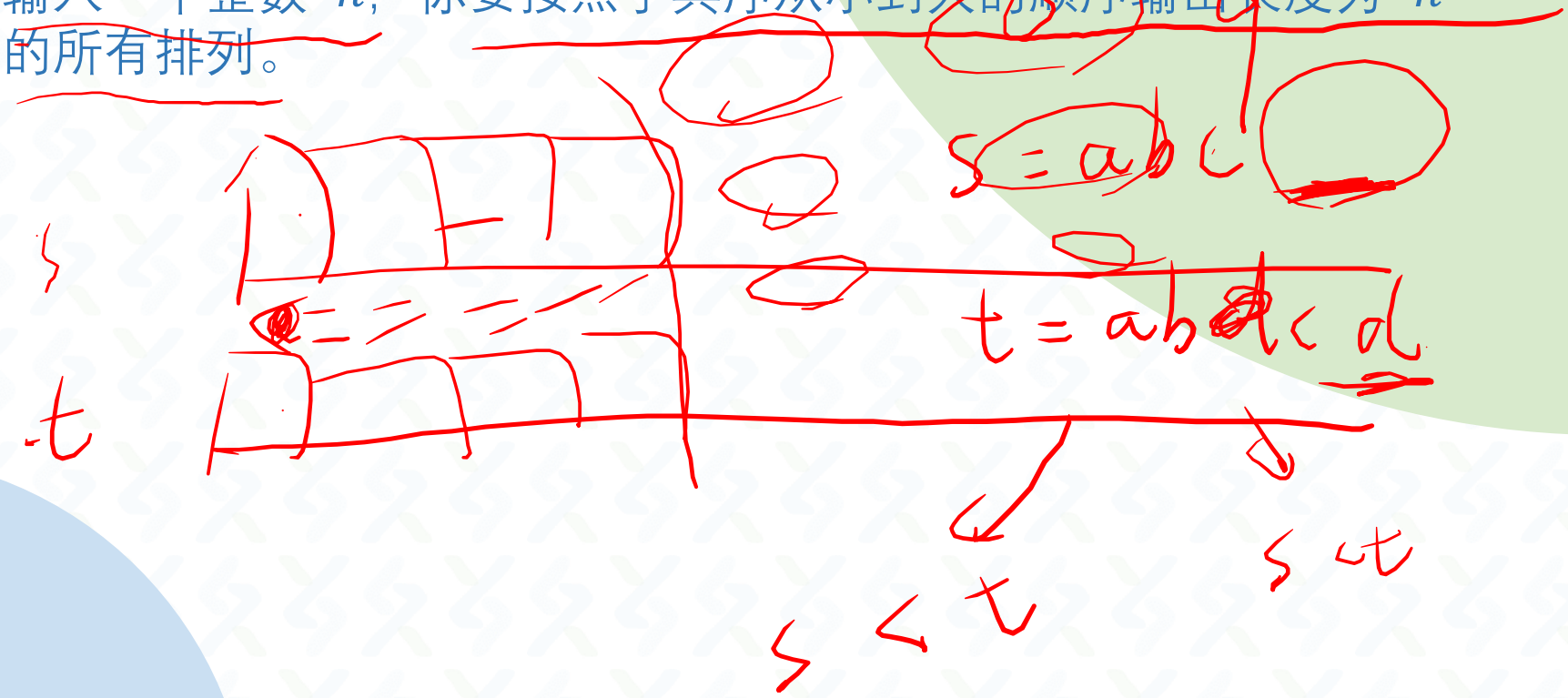
代码实现

```cpp
map<string, int> mp;
cin >> n;
for (int i = 1; i <= n; ++i) {
    string s; cin >> s; mp[s] = 1;
}
cin >> m;
for (int i = 1; i <= m; ++i) {
    string s; cin >> s;
    if (!mp[s]) {
        cout << "WRONG" << endl;
    } else if (mp[s] == 1) {
        ++mp[s];
        cout << "OK" << endl;
    } else {
        cout << "REPEAT" << endl;
    }
}
```

【第二周】全排列问题

题目链接 https://www.luogu.com.cn/problem/T635883

输入一个整数 n，你要按照字典序从小到大的顺序输出长度为 n 的所有排列。



【第二周】全排列问题

解析

next_permutation 练习题。

将 1 ~ n 依次放到一个数组 a[1...n] 中，之后用：

```
do {
    for (int i = 1; i <= n; i++)
        cout << "    " << a[i];
    cout << "\n";
} while (next_permutation(a + 1, a + n + 1));
```

输出即可。

【第二周】小粉兔字典

题目链接 https://www.luogu.com.cn/problem/T635884

有一个字典，字典里有 n 个单词。

需要按字典序从小到大的顺序输出以字符串 s 为前缀的字符串。

【第二周】小粉兔字典

解析

使用 XXX.substr() 可以将某个字符串的某个子串提取出来。

对于要输出的字符串 a[1..n]，首先全部排序。

其次对每个字符串 a[i]，用 .substr 将开头长度为 s.length() 的串提取出来，如果

```
a[i].substr(0, s.length()) == s
```

则可以输出。

整个过程时间复杂度是 $O(nm\log n)$ 的（m 是字符串的最大长度）。

【第二周】小粉兔字典

代码实现

10:00

```
cin >> n;
for (int i = 1; i <= n; i++) {
    cin >> a[i];
}
sort(a + 1, a + n + 1);
cin >> s;
for (int i = 1; i <= n; i++) {
    if (a[i].substr(0, s.length()) == s) {
        cout << a[i] << '\n';
    }
}
```

【第二周】扫地机器人

题目链接 https://www.luogu.com.cn/problem/T635885

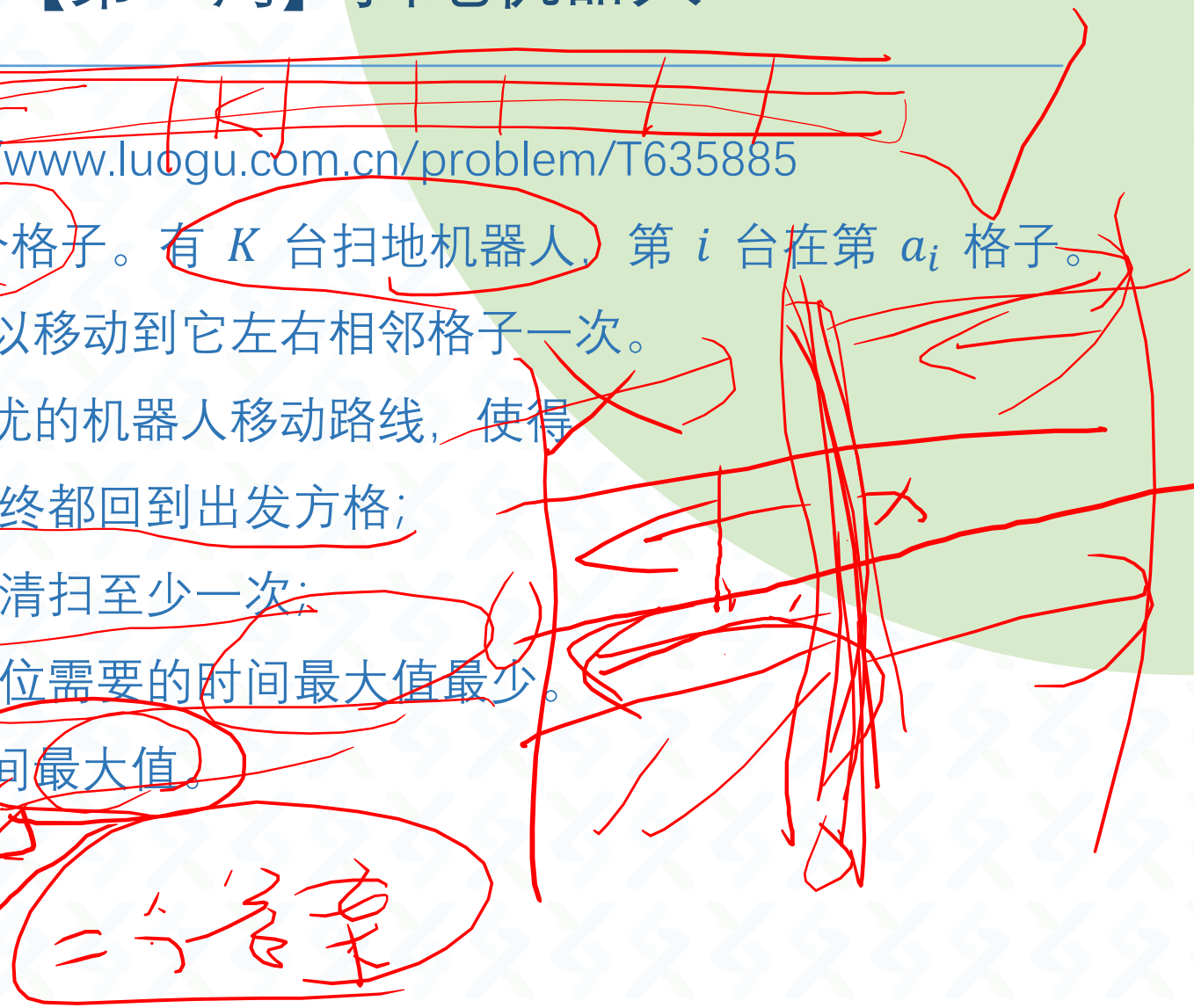
一片区域有 N 个格子。有 K 台扫地机器人，第 i 台在第 a_i 格子。

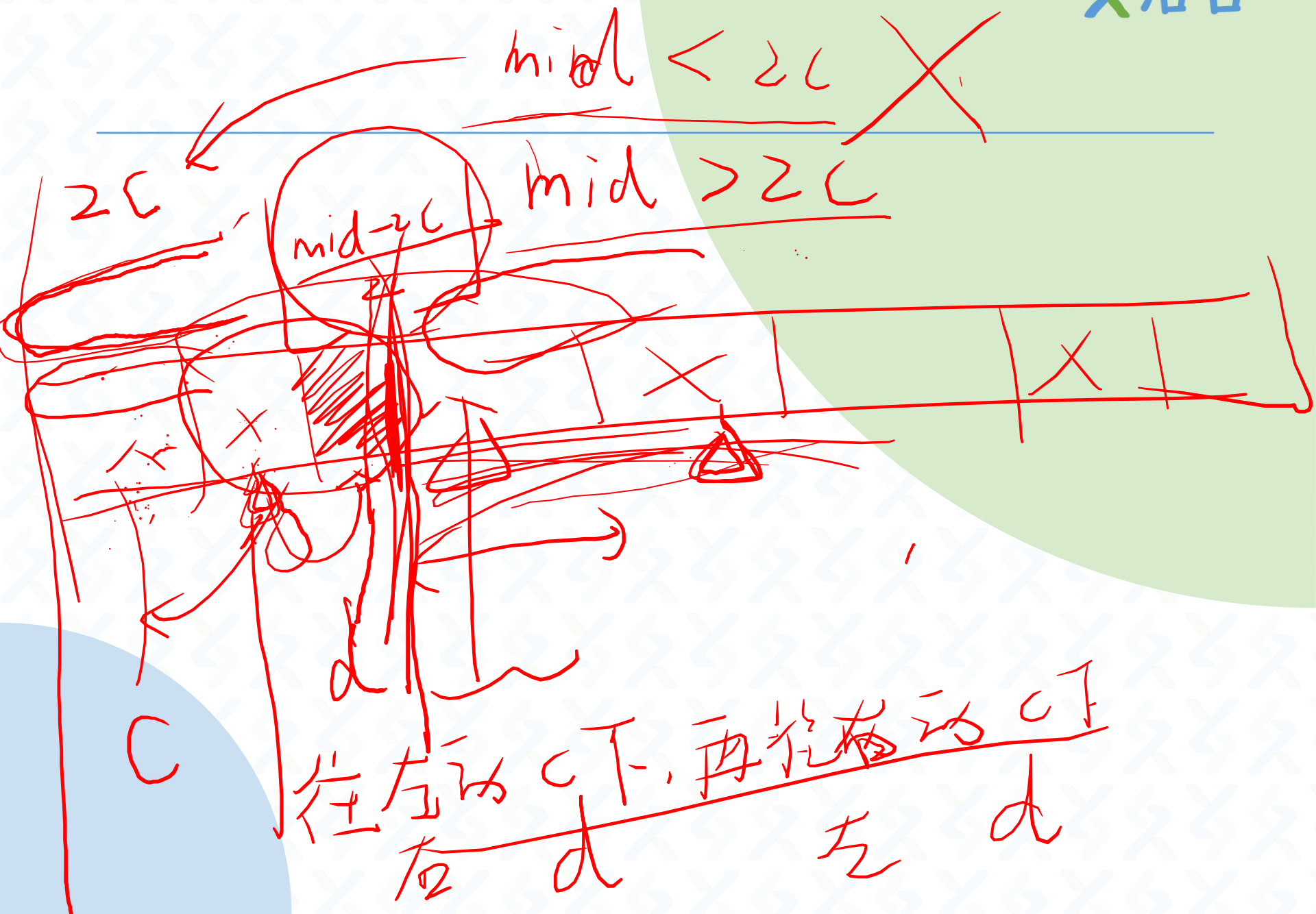
机器人每分钟可以移动到它左右相邻格子一次。

要求找到一种最优的机器人移动路线，使得

- 所有机器人最终都回到出发方格；
- 每个方格都被清扫至少一次；
- 所有机器人归位需要的时间最大值最少。

求这个最少的时间最大值。




mid < 2C
2C
mid > 2C
mid-2C
C
在左边C下，再往右边C下
左 d 左 d

【第二周】扫地机器人

思路

考虑二分答案。

二分清扫时间 t，考虑时间 t 内每个机器人的路线分配。

从最左边的机器人开始，注意到它左边的区域由它清理是最优的。如果有剩余时间，它应该在保证能回来的前提下尽可能向右移动。

对于第二个机器人也是同样的，如果它左边（即与第一个机器人间隔的区域）有未被清理的方格，应该首先清理这些方格。如果有剩余时间，也应该在保证能回来的前提下尽可能向右移动。以此类推。

【第二周】扫地机器人

代码实现

```
int check(lint x) {
    lint cur = 1;
    for (int i = 1; i <= k; ++i) {
        lint delta = x;
        if (cur < a[i]) {
            lint t = (a[i] - cur) * 2;
            if (t > x)
                return 0;
            delta -= t;
        }
        cur = a[i] + (delta / 2) + 1;
    }
    return cur >= n + 1;
}
```

```
cin >> n >> k;
for (int i = 1; i <= k; ++i)
    cin >> a[i];
sort(a + 1, a + k + 1);
lint l = 0, r = (n * 2), ans = 0;
while (l <= r) {
    lint mid = (l + r) >> 1;
    if (check(mid)) {
        ans = mid;
        r = mid - 1;
    } else
        l = mid + 1;
}
cout << ans << endl;
```

出边排序 2

题目链接 https://www.luogu.com.cn/problem/B3613

给定一个 n 个点 m 条边的有向图 G，结点编号从 1 至 n。

每个点有一权值 $w_1, \cdots, w_n$。

对于 $u = 1, 2, 3, \ldots, n$，依次完成如下要求：

对于 u 的所有出边（即从 u 出发的边），按照权值从小到大的顺序输出出边所指向的节点编号。如果两个点的权值相同，先输出编号较小的。

依次完成的含义是，先按顺序输出 $u = 1$ 的出边所指向的点的编号，再按顺序输出 $u = 2$ 的出边所指向的点的编号……最后按顺序输出 $u = n$ 的出边所指向的点的编号。

出边排序 2

思路

同第一题，使用邻接表存图后，对每个 vector 单独排序即可。

只是这里排序规则不同，首先按权值排序，其次按编号排序。因此排序时需要自定义排序函数。

形如 sort(g[i].begin(), g[i].end, cmp);

```
bool cmp(int a, int b){
    if(w[a] != w[b])
        return w[a]<w[b];
    else return a<b;
}
```

出边排序 2

代码实现

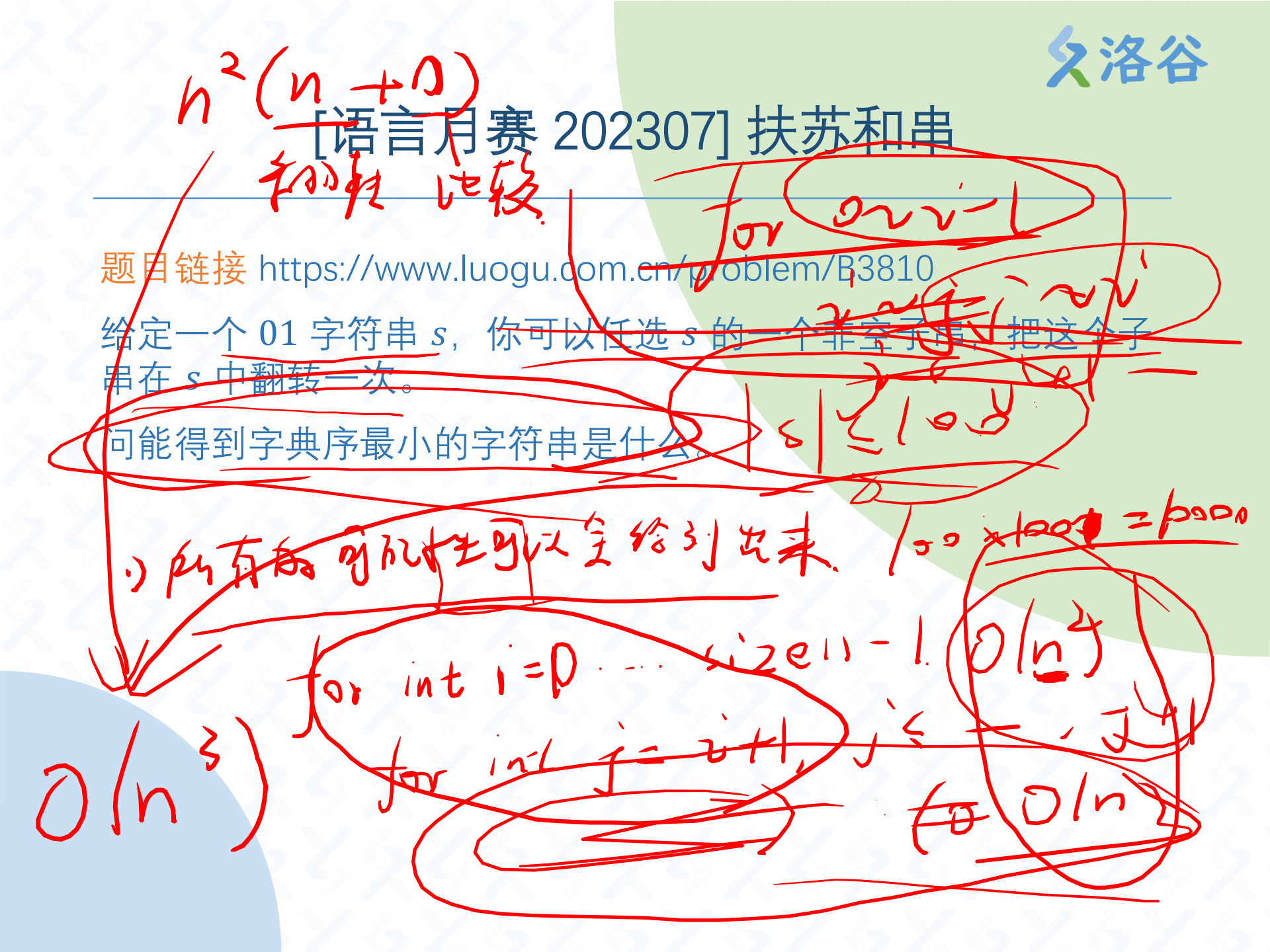
```cpp
bool cmp(int a, int b) {
    if (w[a] != w[b]) { return w[a] < w[b]; }
    return a < b;
}
void solve() {
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) { cin >> w[i]; }
    for (int i = 1; i <= n; i++) { g[i].clear(); }
    for (int i = 1; i <= m; ++i) {
        int u, v;  cin >> u >> v;  g[u].push_back(v);
    }
    for (int i = 1; i <= n; ++i) {
        sort(g[i].begin(), g[i].end(), cmp);
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < g[i].size(); ++j) {
            cout << g[i][j] << " ";
        }
        cout << '\n';
    }
}
```

[语言月赛 202307] 扶苏和串

题目链接 https://www.luogu.com.cn/problem/B3810

给定一个 01 字符串 s，你可以任选 s 的一个非空子串，把这个子串在 s 中翻转一次。

问能得到字典序最小的字符串是什么。



[语言月赛 202307] 扶苏和串

思路

可以使用 substr() 函数和 reverse 函数来解决。

枚举每个可能的翻转子串，假设要反转 s[i] ~ s[j]；

可以用：

- left = s.substr(0, i);
- mid = s.substr(i, j - i + 1);
- right = s.substr(j + 1);（如果截到最后，可省略长度）

之后 reverse (mid.begin(), mid.end())，用

left + mid + right

即可拼出新字符串。再使用擂台法比较即可。

[语言月赛 202307] 扶苏和串

代码实现

```cpp
string s; cin >> s; string v = s;
int n = s.length();
for (int i = 0; i + 1 < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
        string left = s.substr(0, i);
        string mid = s.substr(i, j - i + 1);
        string right = s.substr(j + 1);
        reverse(mid.begin(), mid.end());
        string tmp = left + mid + right;
        v = min(v, tmp);
    }
}
cout << v << endl;
```

单向链表

题目链接 https://www.luogu.com.cn/problem/B3631

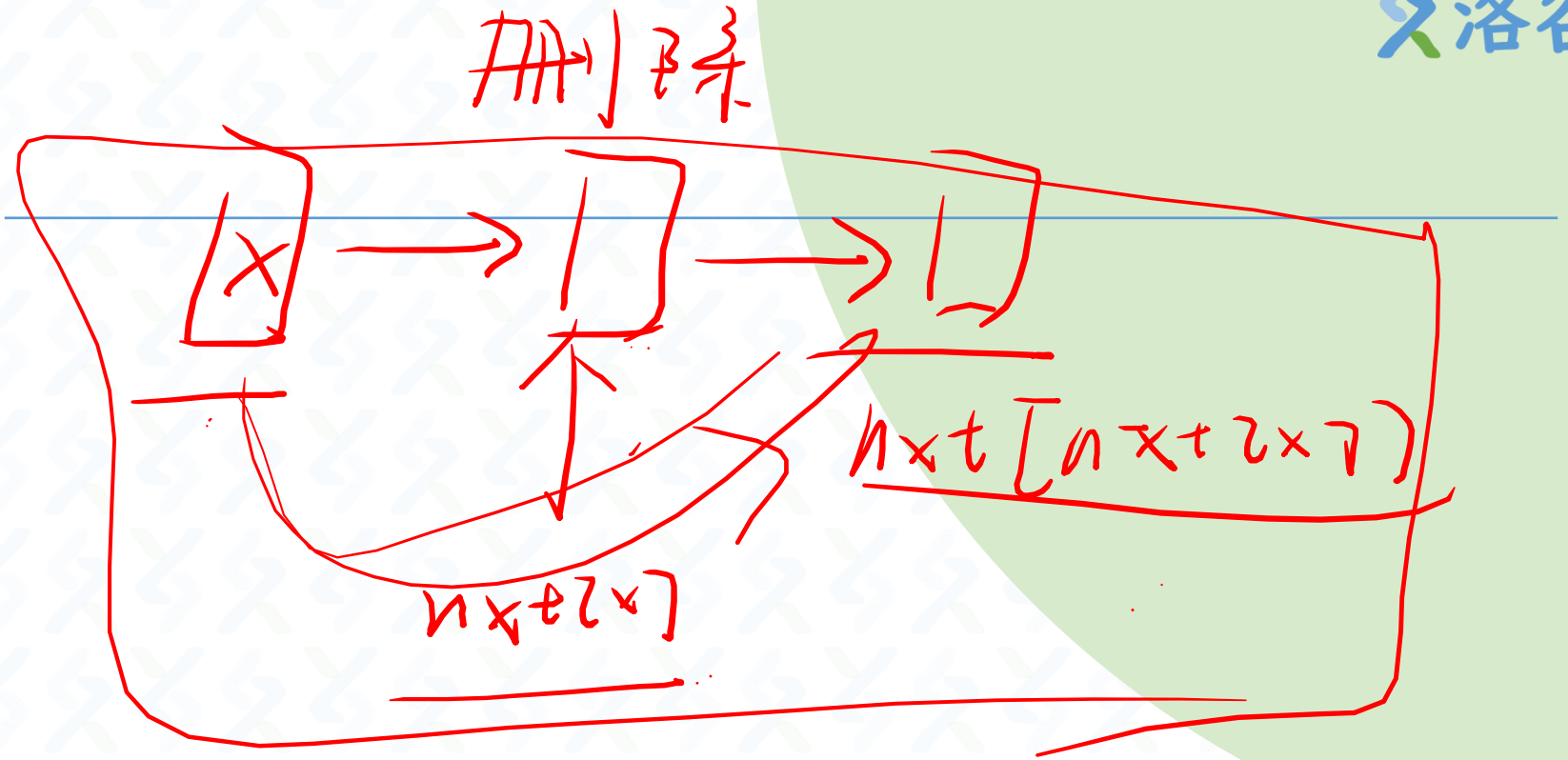
实现一个数据结构，维护一张表（最初只有一个元素 1）。需要支持下面的操作，其中 x 和 y 都是 1 到 10^6 范围内的正整数，且保证任何时间表中所有数字均不相同，操作数量不多于 10^5：

$1\ x\ y$：将元素 y 插入到 x 后面；

$2\ x$：询问 x 后面的元素是什么。如果 x 是最后一个元素，则输出 0；

$3\ x$：从表中删除元素 x 后面的那个元素，不改变其他元素的先后顺序。

nxt[x]
x → 后面
是吗



tmp1 = x tmp2 = nxt[x] tmp3 = nxt[nxt[x]]

nxt[tmp1] = tmp3

nxt[tmp2] = 0

单向链表

思路

直接采用一个 int 类型的 nxt 数组就可以。nxt[i] 表示数据为 i 的结点的 NEXT。

操作 1：nxt[y] = nxt[x]; nxt[x] = y;

操作 2：直接输出 nxt[x];

操作 3：nxt[x] = nxt[nxt[x]];

单向链表

代码实现

```
cin >> q;
for (int o = 1; o <= q; o++) {
    int op;
    cin >> op;
    if (op == 1) {
        int x, y; cin >> x >> y;
        nxt[y] = nxt[x];
        nxt[x] = y;
    } else if (op == 2) {
        int x; cin >> x;
        cout << nxt[x] << endl;
    } else if (op == 3) {
        int x;  cin >> x;
        nxt[x] = nxt[nxt[x]];
    }
}
```

小清新数据结构题

题目链接 https://www.luogu.com.cn/problem/B3631

给定 n 条数据，第 i 条数据有 s_i 个数，依次记为 $a_{i,1}, a_{i,2}, \cdots, a_{i,s_i}$。

现在有 q 次询问，每次询问第 x 条数据的第 y 个数，即 $a_{x,y}$ 是多少。

要输出所有询问的答案的按位异或和。

按位异或指的是 C++ 中的「^」运算符。

$n \leq 3 \times 10^6$

int [3e6+10]

小清新数据结构题

思路

注意到不知道每条数据的具体大小，只知道总和。且二维数组会内存超限。暗示可能需要动态数组。

使用 vector 来解决即可。

对每条数据开一个 vector a[i]，之后将读入的数据全部 push_back 到其中即可。

由于 $0 \le a_i < 2^{32}$，要使用 unsigned int 来存储数据。

小清新数据结构题

代码实现

```
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) {
        int s; cin >> s;
        for (int j = 1; j <= s; ++j) {
            unsigned x; cin >> x;
            a[i].push_back(x);
        }
    }
    vector<unsigned> ans;
    while (q--) {
        int x, y; cin >> x >> y; y--;
        ans.push_back(a[x][y]);
    }
    cout << getXorSum(ans) << endl;
}
```

「Wdoi-4」兔已着陆

题目链接 https://www.luogu.com.cn/problem/P7870

使用了一种类似于「栈」的结构。在一天的开始，这个栈为空栈。现在有 n 次操作，分为两种：

- `1 l r`：将价格为 $l, l+1, \cdots, r-1, r$ 的团子依次入栈。也就是在栈顶依次加入 $l, l+1, l+2, \cdots r-1, r$。
- `2 k`：有一位客人想要购买 k 个团子。依次从栈顶取出 k 个团子并售出。保证 k 不大于当前栈内的团子个数。

对于每个操作 2，输出这些团子的总价格。

「Wdoi-4」兔已着陆

思路

注意到如果真的模拟逐个元素出入栈会很慢。

可以将所有区间存入栈。每次购买时，从中取出若干区间，凑出 k 个团子。区间的总价格可以使用求和公式来解决：

$$\text{平均单价}\ \frac{l+r}{2} \times \text{数量}\ (r-l+1)$$

如果最后一个区间没用完就凑够了 k 个团子，则将剩余区间重新塞入栈即可。

「Wdoi-4」兔已着陆

```
int n; cin >> n;
while (n--) {
    int op; cin >> op;
    if (op == 1) {
        int l, r; cin >> l >> r;
        s.push({l, r});
    } else {
        lint k; cin >> k; lint ans = 0;
        while (k) {
            node top = s.top(); s.pop();
            lint tot_val = calc(top.l, top.r);
            lint tot_len = top.r - top.l + 1;
            if (tot_len <= k) {
                ans += tot_val; k -= tot_len;
            } else {
                top.r = top.l + (tot_len - k) - 1;
                ans += tot_val - calc(top.l, top.r);
                s.push(top); k = 0;
            }
        }
        cout << ans << '\n';
    }
}
```

```
struct node {
    int l, r;
};
stack<node> s;

lint calc(int l, int r)
{
    return (lint) (r -
l + 1) * (r + l) / 2;
}
```

后缀表达式

题目链接 https://www.luogu.com.cn/problem/P1449

所谓后缀表达式是指这样的一个表达式：式中不再引用括号，运算符号放在两个运算对象之后，所有计算按运算符号出现的顺序，严格地由左而右新进行（不用考虑运算符的优先级）。

输入一行一个字符串 s，表示后缀表达式。

输出一个整数，表示表达式的值。

逆波兰表达式

概念复习

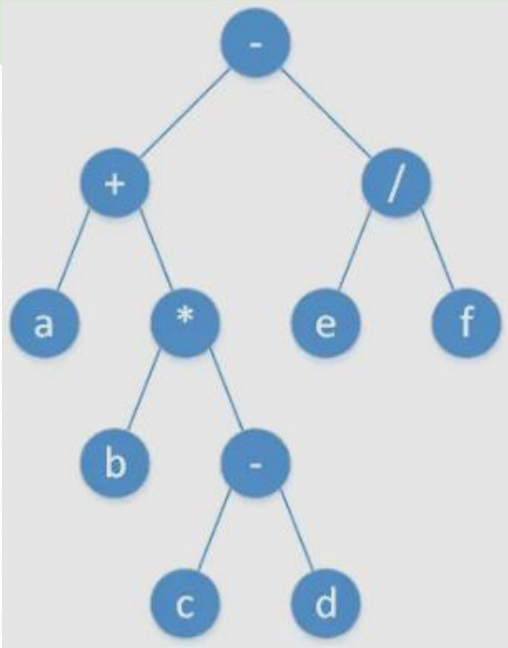
表达式树上进行树的遍历可以得到不同类型的表达式。

算术表达式分为三种，分别是前缀表达式、中缀表达式、后缀表达式。

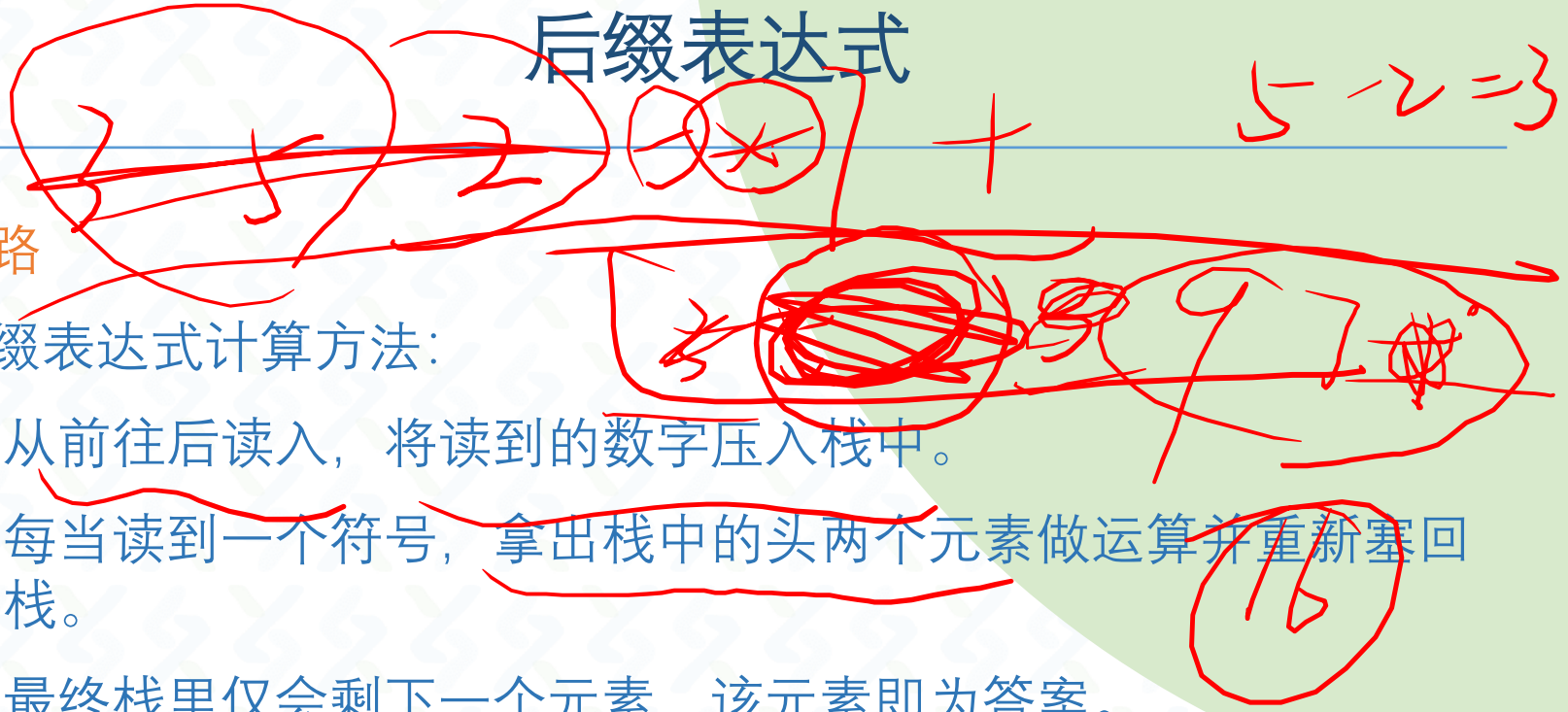
中缀表达式是日常生活中最常用的表达式；

后缀表达式是计算机容易理解的表达式。

前后缀表达式转中缀表达式：栈



后缀表达式



思路

后缀表达式计算方法：

- 从前往后读入，将读到的数字压入栈中。
- 每当读到一个符号，拿出栈中的头两个元素做运算并重新塞回栈。
- 最终栈里仅会剩下一个元素，该元素即为答案。

处理字符串时，对每个字符：

- 如果其是数字，则用一个 `int` 实时转换，读取到 `.` 后压入栈中；
- 否则，按后缀表达式符号方法计算。

后缀表达式

代码实现

```
string s; cin >> s;
stack<int> st; int num = 0; bool has_num = false;
for (size_t i = 0; i < s.size(); ++i) {
    char c = s[i];
    if (isdigit(c)) {num = num * 10 + (c - '0'); has_num = true; }
    else if (c == '.') {
        if (has_num) { st.push(num); num = 0; has_num = false; }
    } else if (c == '+' || c == '-' || c == '*' || c == '/') {
        int b = st.top(); st.pop();
        int a = st.top(); st.pop();
        if (c == '+') st.push(a + b);
        else if (c == '-') st.push(a - b);
        else if (c == '*') st.push(a * b);
        else if (c == '/') st.push(a / b);
    } else if (c == '@') {
        break;
    }
}
cout << st.top() << endl;
```

小小的堉轮兵团

题目链接 https://www.luogu.com.cn/problem/P7505

给定 n 个士兵及其初始在数轴上的位置。数轴长度为 $[-k,k]$，超出这个长度的士兵会消失。

有三种操作：全体士兵正向移动 x 个单位长度；全体士兵负向移动 x 个单位长度；统计目前在队列中的士兵个数。

对于每次统计，输出当前的士兵个数。

士兵正向移动 x 单位长度

⇒ 数轴整体向左偏移 x

小小的堆轮兵团

解析

本题定位为 deque 双端队列练习题。

考虑到对士兵逐个移动，时间复杂度为 $O(nm)$，无法接受。因此考虑相对性，直接移动坐标轴的长度范围。

在移动后，考虑如何处理消失的士兵。

逐个枚举的复杂度过高，无法接受。考虑是否有一个结构，可以直接查询最靠近边界的士兵是哪一位。

考虑双端队列的特性。将所有士兵按初始时的位置由小到大排序，置入一个双端队列。每次移动长度范围后，判断队列的头和尾是否出界，如果出界则不断弹出。

查询直接使用 `size()` 成员函数即可。

小小的堆轮兵团

代码实现

```
int n, m, k;
int a[300005];
lint l, r;

cin >> n >> m >> k;
for (int i = 1; i <= n; ++i) {
    cin >> a[i];
}
l = -k, r = k;
sort(a + 1, a + n + 1);
deque<int> q;
for (int i = 1; i <= n; ++i) {
    q.push_back(a[i]);
}
```

```
while (m--) {
    int op, x;
    cin >> op;
    if (op == 3) {
        cout << q.size() << endl;
        continue;
    }
    cin >> x;
    if (op == 1) {
        l -= x, r -= x;
    } else {
        l += x, r += x;
    }
    while (q.size() && q.back() > r)
        q.pop_back();
    while (q.size() && q.front() < l)
        q.pop_front();
}
```

[CSP-J 2021] 小熊的果篮

题目链接 https://www.luogu.com.cn/problem/P7912

给定 n 个水果，每个水果分类可以分为 $0,1$。n 个水果从左向右排列可以排成一个 $01010100010010101011\ldots$ 串。

一段连续的 0 或者 1 称为一个“块”。需要进行若干次操作，每次从每个“块”里取出最头上的元素。

每将元素取出后，某些块可能被取完。此时，块之间的相邻关系会有更新。如果更新后，有两个相邻块的元素相同，则将这两个块合并。

求所有操作取出的元素各是什么。

[CSP-J 2021] 小熊的果篮

思路

问题的重点在“块”，分块内和块间。

1．块内

块内需要维护的是这个块里还有几个元素，分别是什么，建议使用队列维护。

2．块间

块间使用类似链表的方式维护，对于每一个块，维护它前面和后面的块。

维护好这些以后，对每一次操作，按照题意进行处理。分删除和合并两步。可以使用上面维护的内容来实现。

[CSP-J 2021] 小熊的果篮

思路

1．删除：弹出队列中的头部元素并输出，块内元素数 -1。

2．合并：当一个块中没有元素了，与链表类似做以下合并。

1）让该块前面的块的“后面的块”指向该块的“后面的块”。

2）让该块后面的“前面的块”指向该块的“前面的块”

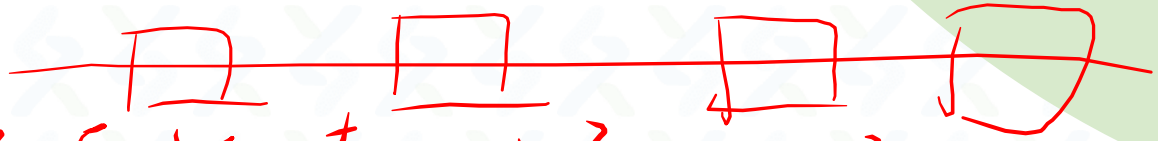
3）由于一次删除可能导致很多块中没有元素，因此这里要进行判断，判断该块“前面的块”和“后面的块”里的元素是否一致，一致的话则进行合并，合并的步骤是：让“前面的块”的“后面的块”指向“后面的块”的“后面的块”，然后把“后面的块”队列中的元素推入“前面的块”中，并顺带更新块内元素个数。

块 队列未存 链表

洛谷

1) 遍历链表，里面的每个队列弹出元素

2) 哪些队列空了，移出去



从链表里把这些队列删

3) 遍历链表，对每个队列为

① 和 nxt[x] 元素种类是否相同

相同，则将nxt[x]的元素全部拉到x里面

nxt[x]从链表里删掉

[CSP-J 2021] 小熊的果篮

代码实现

```
int n;
int block_size[200005]; // 块内元素个数
queue<int> num[200005]; // 题解中所指的队列
int cnt; // 块个数
int last_element[200005]; // 块中的元素是什么
int next_block[200005], prev_block[200005]; // 块的“前面的块、后面的块”

last_element[0] = -1; // 让起点块的元素总是与后面不同
for (int i = 1; i <= n; ++i) {
    int x; cin >> x; // 填充入块操作，判断是否与最后一个块内元素相同，不相同则开个新块
    if (x != last_element[cnt]) {
        ++cnt; next_block[cnt - 1] = cnt;
        prev_block[cnt] = cnt - 1; last_element[cnt] = x;
    }
    ++block_size[cnt]; num[cnt].push(i);
}
```

[CSP-J 2021] 小熊的果篮

代码实现

```
while (cnt) { // 当还有块的时候
    // 删除
    int p = next_block[0]; // 起点出发
    while (p) {            // 当 p 还指向一个块的时候
        --block_size[p];   // 删除操作
        cout << num[p].front() << ' ';
        num[p].pop();
        p = next_block[p];
    }
    cout << endl;

...
```

[CSP-J 2021] 小熊的果篮

```
p = next_block[0]; // 重置，再次从起点出发
while (p) {
    if (!block_size[p]) { // 该块为空
        int prev_t = prev_block[p], t = p; // t 当前块，prev_t 该块“前面的块”
        while (!block_size[t] && t) // 找到下一个有元素的块
            t = next_block[t], --cnt;
        if (t == 0) { // 找不到
            next_block[prev_t] = t; p = t;
        } else if (last_element[prev_t] != last_element[t]) { // 找到，元素不同
            next_block[prev_t] = t; prev_block[t] = prev_t; p = t;
        } else { // 元素相同
            block_size[prev_t] += block_size[t]; --cnt; // 块大小，块的个数 - 1
            next_block[prev_t] = next_block[t];  // 类似链表操作
            prev_block[next_block[t]] = prev_t;
            while (!num[t].empty()) { // 队列里的元素进行转移
                int v = num[t].front(); num[t].pop(); num[prev_t].push(v);
            }
            p = next_block[prev_t];
        }
    } else
        p = next_block[p];
    }
}
```

洛谷

12:05