数据结构-单链表

一、AcWing 826. 单链表

【题目描述】

实现一个单链表，链表初始为空，支持三种操作：

- 向链表头插入一个数；
- 删除第k个插入的数后面的数；
- 在第k个插入的数后插入一个数。

现在要对该链表进行M次操作，进行完所有操作后，从头到尾输出整个链表。

注意：题目中第k个插入的数并不是指当前链表的第k个数。例如操作过程中一共插入了n个数，则按照插入的时间顺序，这n个数依次为：第1个插入的数，第2个插入的数，...，第n个插入的数。

【输入格式】

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令，操作命令可能为以下几种：

- `H x`，表示向链表头插入一个数x。
- `D k`，表示删除第k个插入的数后面的数（当k为0时，表示删除头结点）。
- `I k x`，表示在第k个插入的数后面插入一个数x（此操作中k均大于0）。

【输出格式】

共一行，将整个链表从头到尾输出。

【数据范围】

$1 \leq M \leq 100000$

所有操作保证合法。

【输入样例】

```
10
H 9
I 1 1
D 1
D 0
H 6
I 3 6
I 4 5
I 4 5
I 3 4
D 6
```

【输出样例】

```
6 4 6 5
```

【代码】

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

const int N = 100010;
int head, idx, e[N], ne[N];

//将元素x插入单链表头部
void add_to_head(int x)
{
    e[idx] = x, ne[idx] = head, head = idx++;
}

//在下标为k的元素后插入元素x
void add(int k, int x)
{
    e[idx] = x, ne[idx] = ne[k], ne[k] = idx++;
}

//删除下标为k的元素的后一个数
void del(int k)
{
    ne[k] = ne[ne[k]];
}

int main()
```

```
{
    int m;
    cin >> m;
    head = -1, idx = 0;//初始化单链表
    while (m--)
    {
        char op;
        int k, x;
        cin >> op;
        if (op == 'H') { cin >> x; add_to_head(x); }
        else if (op == 'D')
        {
            cin >> k;
            //特判删除头结点的情况
            if (k == 0) head = ne[head];
            else del(k - 1);
        }
        else { cin >> k >> x; add(k - 1, x); }
    }
    for (int i = head; ~i; i = ne[i]) cout << e[i] << ' ';
    return 0;
}
```

数据结构-双链表

一、AcWing 827. 双链表

【题目描述】

实现一个双链表，双链表初始为空，支持5种操作：

- 在最左侧插入一个数；
- 在最右侧插入一个数；
- 将第k个插入的数删除；
- 在第k个插入的数左侧插入一个数；
- 在第k个插入的数右侧插入一个数。

现在要对该链表进行M次操作，进行完所有操作后，从左到右输出整个链表。

注意：题目中第k个插入的数并不是指当前链表的第k个数。例如操作过程中一共插入了n个数，则按照插入的时间顺序，这n个数依次为：第1个插入的数，第2个插入的数,...,第n个插入的数。

【输入格式】

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令，操作命令可能为以下几种：

- `L x`，表示在链表的最左端插入数x。
- `R x`，表示在链表的最右端插入数x。
- `D k`，表示将第k个插入的数删除。
- `IL k x`，表示在第k个插入的数左侧插入一个数。
- `IR k x`，表示在第k个插入的数右侧插入一个数。

【输出格式】

共一行，将整个链表从左到右输出。

【数据范围】

$1 \leq M \leq 100000$

所有操作保证合法。

【输入样例】

```
10
R 7
D 1
L 3
IL 2 10
D 3
IL 2 7
L 8
R 9
IL 4 7
IR 2 2
```

【输出样例】

```
8 7 7 3 2 9
```

【代码】

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

```

```cpp
const int N = 100010;
int l[N], r[N], e[N], idx;

//初始化双链表
void init()
{
    //0表示左端点，1表示右端点
    r[0] = 1, l[1] = 0;
    idx = 2;
}

//在下标为k的节点右边插入元素x
void add(int k, int x)
{
    e[idx] = x;
    r[idx] = r[k];
    l[idx] = k;
    l[r[k]] = idx;//最后两句顺序不能颠倒
    r[k] = idx++;
}

//删除下标为k的点
void del(int k)
{
    r[l[k]] = r[k];
    l[r[k]] = l[k];
}

int main()
{
    int m;
    cin >> m;
    init();
    while (m--)
    {
        string op;
        int x, k;
        cin >> op;
        if (op == "L") { cin >> x; add(0, x); }
        else if (op == "R") { cin >> x; add(l[1], x); }
        else if (op == "D") { cin >> k; del(k + 1); }
        else if (op == "IL") { cin >> k >> x; add(l[k + 1], x); }

        else { cin >> k >> x; add(k + 1, x); }
```

```
    }
    for (int i = r[0]; i != 1; i = r[i]) cout << e[i] << ' ';
    return 0;
}
```

数据结构-栈

一、AcWing 828. 模拟栈

【题目描述】

实现一个栈，栈初始为空，支持四种操作：

- `push x`：向栈顶插入一个数x；
- `pop`：从栈顶弹出一个数；
- `empty`：判断栈是否为空；
- `query`：查询栈顶元素。

现在要对栈进行M个操作，其中的每个操作**3**和操作**4**都要输出相应的结果。

【输入格式】

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令，操作命令为`push x`，`pop`，`empty`，`query`中的一种。

【输出格式】

对于每个`empty`和`query`操作都要输出一个查询结果，每个结果占一行。

其中，`empty`操作的查询结果为`YES`或`NO`，`query`操作的查询结果为一个整数，表示栈顶元素的值。

【数据范围】

$1 \leq M \leq 100000$

$1 \leq x \leq 10^9$

所有操作保证合法。

【输入样例】

```
10
push 5
query
push 6
pop
query
pop
empty
push 4
query
empty
```

【输出样例】

```
5
5
YES
4
NO
```

【代码】

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

const int N = 100010;
int m;
int stk[N], tt;

int main()
{
    cin >> m;
    while (m--)
    {
        string op;
        int x;
        cin >> op;
        if (op == "push")
        {
            cin >> x;
            stk[++tt] = x;
        }
```

```
        else if (op == "pop") tt--;
        else if (op == "query") cout << stk[tt] << endl;
        else cout << (tt > 0 ? "NO" : "YES") << endl;
    }
    return 0;
}
```

二、AcWing 3302. 表达式求值

【题目描述】

给定一个表达式，其中运算符仅包含`+ - * /`（加 减 乘 整除），可能包含括号，请你求出表达式的最终值。

注意：

- 数据保证给定的表达式合法。
- 题目保证符号`-`只作为减号出现，不会作为负号出现，例如，`-1+2`，`(2+2)*(-(1+1)+2)`之类表达式均不会出现。
- 题目保证表达式中所有数字均为正整数。
- 题目保证表达式在中间计算过程以及结果中，均不超过$2^{31}-1$。
- 题目中的整除是指向0取整，也就是说对于大于0的结果向下取整，例如$5/3=1$，对于小于0 的结果向上取整，例如$5/(1-4)=-1$。
- C++和Java中的整除默认是向零取整；Python中的整除`//`默认向下取整，因此Python的`eval()`函数中的整除也是向下取整，在本题中不能直接使用。

【输入格式】

共一行，为给定表达式。

【输出格式】

共一行，为表达式的结果。

【数据范围】

表达式的长度不超过10^5。

【输入样例】

```
(2+2)*(1+1)
```

【输出样例】

```
8
```

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <stack>
#include <unordered_map>
using namespace std;

string str;
stack<int> num;
stack<char> op;
unordered_map<char, int> pr{ {'+', 1}, {'-', 1}, {'*', 2}, {'/', 2} };

void eval()
{
    int b = num.top(); num.pop();
    int a = num.top(); num.pop();
    char c = op.top(); op.pop();
    if (c == '+') num.push(a + b);
    else if (c == '-') num.push(a - b);
    else if (c == '*') num.push(a * b);
    else num.push(a / b);
}

int main()
{
    cin >> str;
    for (int i = 0; i < str.size(); i++)
    {
        char c = str[i];
        if (isdigit(c))
        {
            int x = 0, j = i;
            while (j < str.size() && isdigit(str[j]))
                x = x * 10 + str[j++] - '0';
            i = j - 1;
            num.push(x);
        }
        else if (c == '(') op.push(c);
        else if (c == ')')

        {
```

```
            while (op.top() != '(') eval();
            op.pop();
        }
        else
        {
            while (op.size() && pr[op.top()] >= pr[c]) eval();
            op.push(c);
        }
    }
    while (op.size()) eval();
    cout << num.top() << endl;
    return 0;
}
```

数据结构-队列

一、AcWing 829. 模拟队列

【题目描述】

实现一个队列，队列初始为空，支持四种操作：

- `push x`：向队尾插入一个数x；
- `pop`：从队头弹出一个数；
- `empty`：判断队列是否为空；
- `query`：查询队头元素。

现在要对队列进行M个操作，其中的每个操作**3**和操作**4**都要输出相应的结果。

【输入格式】

第一行包含整数M，表示操作次数。

接下来M行，每行包含一个操作命令，操作命令为`push x`，`pop`，`empty`，`query`中的一种。

【输出格式】

对于每个`empty`和`query`操作都要输出一个查询结果，每个结果占一行。

其中，`empty`操作的查询结果为`YES`或`NO`，`query`操作的查询结果为一个整数，表示队头元素的值。

【数据范围】

$1 \le M \le 100000$

$1 \le x \le 10^9$

所有操作保证合法。

【输入样例】

```
10
push 6
empty
query
pop
empty
push 3
push 4
pop
query
push 6
```

【输出样例】

```
NO
6
YES
4
```

【代码】

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

const int N = 100010;
int q[N], hh, tt = -1;
int m;

int main()
{
    cin >> m;
    while (m--)
    {
        string op;

        int x;
```

```
        cin >> op;
        if (op == "push") { cin >> x; q[++tt] = x; }
        else if (op == "pop") hh++;
        else if (op == "query") cout << q[hh] << endl;
        else cout << (hh > tt ? "YES" : "NO") << endl;
    }
    return 0;
}
```

数据结构-单调栈

一、AcWing 830. 单调栈

【题目描述】

给定一个长度为N的整数数列，输出每个数左边第一个比它小的数，如果不存在则输出-1。

【输入格式】

第一行包含整数N，表示数列长度。

第二行包含N个整数，表示整数数列。

【输出格式】

共一行，包含N个整数，其中第i个数表示第i个数的左边第一个比它小的数，如果不存在则输出-1。

【数据范围】

$1 \leq N \leq 10^5$

$1 \leq$ 数列中元素 $\leq 10^9$

【输入样例】

```
5
3 4 2 7 5
```

【输出样例】

```
-1 3 -1 2 2
```

【分析】

用单调递增栈，当该元素可以入栈的时候，栈顶元素就是它左侧第一个比它小的元素。

【代码】

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

stack<int> stk;
int n;

int main()
{
    cin >> n;
    while (n--)
    {
        int x;
        cin >> x;
        while (stk.size() && stk.top() >= x) stk.pop();
        if (stk.size()) cout << stk.top() << ' ';
        else cout << -1 << ' ';
        stk.push(x);
    }
    return 0;
}
```

数据结构-单调队列

一、AcWing 154. 滑动窗口

【题目描述】

给定一个大小为$n(n \leq 10^6)$的数组。

有一个大小为k的滑动窗口，它从数组的最左边移动到最右边。

你只能在窗口中看到k个数字。

每次滑动窗口向右移动一个位置。

例如：数组为$[1,3,-1,-3,5,3,6,7]$，$k=3$。

窗口位置	最小值	最大值
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

【输入格式】

输入一共有两行，第一行有两个正整数n,k。第二行n个整数，表示序列a。

【输出格式】

输出共两行，第一行为每次窗口滑动的最小值，第二行为每次窗口滑动的最大值。

【输入样例】

```
8 3
1 3 -1 -3 5 3 6 7
```

【输出样例】

```
-1 -3 -3 -3 3 3
3 3 5 5 6 7
```

【分析】

以最小值为例：构造一个单调递增队列Q，Q中存数组a中元素的下标。

第一次：$Q=\{1\}$

第二次：$Q=\{1,2\}$

第三次：$Q=\{3\}$（因为队列单调递增，-1最小，所以挤掉1和3，找最小值的时候也不会轮到它们）

第四次：$Q=\{4\}$（1已经在外面了。$-3<-1$果断挤掉）

第五次：$Q=\{4,5\}$（$5>-3$可以作为候选人）

第六次：$Q=\{4,6\}$（果断挤掉，有比5合适的候选人）

第七次：$Q = \{6, 7\}$（4不在范围内，出列）

第八次：$Q = \{6, 7, 8\}$

每次都输出队列中的第一个元素，即为当前窗口中的最小值。求最大值同理，构造单调递减队列即可

【手写队列代码】

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

const int N = 1000010;
int a[N], Q[N], hh, tt;
int n, k;

int main()
{
    cin >> n >> k;
    for (int i = 0; i < n; i++) cin >> a[i];
    //构造单增队列输出最小值
    hh = 0, tt = -1;
    for (int i = 0; i < n; i++)
    {
        if (hh <= tt && i - Q[hh] == k) hh++;//队首元素超出窗口范围时,移除队首元素
        while (hh <= tt && a[i] < a[Q[tt]]) tt--;//若待入队元素小于队尾元素,则移除队尾元素,构造单增队列
        Q[++tt] = i;
        if (i >= k - 1) cout << a[Q[hh]] << ' ';
    }
    cout << endl;
    //构造单减队列输出最大值
    hh = 0, tt = -1;
    for (int i = 0; i < n; i++)
    {
        if (hh <= tt && i - Q[hh] == k) hh++;
        while (hh <= tt && a[i] > a[Q[tt]]) tt--;
        Q[++tt] = i;
        if (i >= k - 1) cout << a[Q[hh]] << ' ';
    }
    return 0;
}
```

【deque实现代码】

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

const int N = 1000010;
int n, k;
int a[N];
deque<int> Q;

int main()
{
    cin >> n >> k;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++)
    {
        if (Q.size() && i - Q.front() == k) Q.pop_front();
        while (Q.size() && a[i] < a[Q.back()]) Q.pop_back();
        Q.push_back(i);
        if (i >= k - 1) cout << a[Q.front()] << ' ';
    }
    cout << endl;
    Q.clear();
    for (int i = 0; i < n; i++)
    {
        if (Q.size() && i - Q.front() == k) Q.pop_front();
        while (Q.size() && a[i] > a[Q.back()]) Q.pop_back();
        Q.push_back(i);
        if (i >= k - 1) cout << a[Q.front()] << ' ';
    }
    return 0;
}
```

数据结构-KMP

一、AcWing 831. KMP字符串

【题目描述】

给定一个模式串S，以及一个模板串P，所有字符串中只包含大小写英文字母以及阿拉伯数字。

模板串P在模式串S中多次作为子串出现。

求出模板串P在模式串S中所有出现的位置的起始下标。

【输入格式】

第一行输入整数N，表示字符串P的长度。

第二行输入字符串P。

第三行输入整数M，表示字符串S的长度。

第四行输入字符串S。

【输出格式】

共一行，输出所有出现位置的起始下标（下标从0开始计数），整数之间用空格隔开。

【数据范围】

$1 \leq N \leq 10^5$

$1 \leq M \leq 10^6$

【输入样例】

```
3
aba
5
ababa
```

【输出样例】

```
0 2
```

【代码】

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

const int N = 100010, M = 1000010;
int n, m;
int ne[N];
char p[N], s[M];//s为模式串，p为模板串

```

```cpp
int main()
{
    cin >> n >> p + 1 >> m >> s + 1;
    //求next数组
    for (int i = 2, j = 0; i <= n; i++)
    {
        while (j && p[i] != p[j + 1]) j = ne[j];
        if (p[i] == p[j + 1]) j++;
        ne[i] = j;
    }
    //KMP匹配过程
    for (int i = 1, j = 0; i <= m; i++)
    {
        while (j && s[i] != p[j + 1]) j = ne[j];
        if (s[i] == p[j + 1]) j++;
        if (j == n)
        {
            cout << i - n << ' ';
            j = ne[j];
        }
    }
    return 0;
}
```

数据结构-Trie

一、AcWing 835. Trie字符串统计

【题目描述】

维护一个字符串集合，支持两种操作：

- `I x`向集合中插入一个字符串`x`；
- `Q x`询问一个字符串在集合中出现了多少次。

共有N个操作，输入的字符串总长度不超过10^5，字符串仅包含小写英文字母。

【输入格式】

第一行包含整数N，表示操作数。

接下来N行，每行包含一个操作指令，指令为`I x`或`Q x`中的一种。

【输出格式】

对于每个询问指令`Q x`，都要输出一个整数作为结果，表示`x`在集合中出现的次数。

每个结果占一行。

【数据范围】

$1 \leq N \leq 2 \times 10^4$

【输入样例】

```
5
I abc
Q abc
Q ab
I ab
Q ab
```

【输出样例】

```
1
0
1
```

【代码】

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

const int N = 100010;
int son[N][26], cnt[N], idx;
char str[N];
int n;

void insert(char str[])
{
    int p = 0;
    for (int i = 0; str[i]; i++)
    {
        //将a~z映射到0~25
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }

    cnt[p]++;
```

```
}

int query(char str[])
{
    int p = 0;
    for (int i = 0; str[i]; i++)
    {
        int u = str[i] - 'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}

int main()
{
    cin >> n;
    while (n--)
    {
        char op;
        cin >> op >> str;
        if (op == 'I') insert(str);
        else cout << query(str) << endl;
    }
    return 0;
}
```

二、AcWing 143. 最大异或对

【题目描述】

在给定的N个整数$A_1, A_2, \ldots, A_N$中选出两个进行xor（异或）运算，得到的结果最大是多少？

【输入格式】

第一行输入一个整数N。

第二行输入N个整数$A_1 \sim A_N$。

【输出格式】

输出一个整数表示答案。

【数据范围】

$1 \leq N \leq 10^5$

$0 \leq A_i < 2^{31}$

【输入样例】

```
3
1 2 3
```

【输出样例】

```
3
```

【分析】

这道题目很难想到是字典树，如果不是放在字典树单元的话。

其实来说，一个整数，是可以转化成为一个32位的二进制数，而也就可以变成长度为32位的二进制字符串。

既然如此话，那么我们可以这么做：遍历数组A，对于每一个元素A_i，每一次检索的时候，我们都往与当前A_i这一位相反的位置走，也就是让xor值最大，如果说没有路可以走的话，那么就走相同的路。这样可以迅速找到A中所有元素与A_i异或的最大值，对于每一个A_i的最大异或值取max即为最终答案。

【代码】

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

const int N = 100010, M = N * 31;
int son[M][2], idx;
int n, a[N];

void insert(int x)
{
    int p = 0;
    //从x的二进制数的最高位开始做
    for (int i = 30; ~i; i--)
    {
        if (!son[p][x >> i & 1])
            son[p][x >> i & 1] = ++idx;
        p = son[p][x >> i & 1];
```

```
    }
}

int query(int x)
{
    int p = 0, res = 0;
    for (int i = 30; ~i; i--)
    {
        //优先选择与x的第i位异或值为1的节点
        if (son[p][!(x >> i & 1)])
        {
            res += 1 << i;
            p = son[p][!(x >> i & 1)];
        }
        else p = son[p][x >> i & 1];
    }
    return res;
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
        insert(a[i]);
    }
    int res = 0;
    for (int i = 0; i < n; i++) res = max(res, query(a[i]));
    cout << res << endl;
    return 0;
}
```

数据结构-并查集

一、AcWing 836. 合并集合

【题目描述】

一共有n个数，编号是$1 \sim n$，最开始每个数各自在一个集合中。

现在要进行m个操作，操作共有两种：

- `M a b`，将编号为a和b的两个数所在的集合合并，如果两个数已经在同一个集合中，则忽略这个操作；
- `Q a b`，询问编号为a和b的两个数是否在同一个集合中。

【输入格式】

第一行输入整数n和m。

接下来m行，每行包含一个操作指令，指令为`M a b`或`Q a b`中的一种。

【输出格式】

对于每个询问指令`Q a b`，都要输出一个结果，如果a和b在同一集合内，则输出`Yes`，否则输出`No`。

每个结果占一行。

【数据范围】

$1 \leq n, m \leq 10^5$

【输入样例】

```
4 5
M 1 2
M 3 4
Q 1 2
Q 1 3
Q 3 4
```

【输出样例】

```
Yes
No
Yes
```

【代码】

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

const int N = 100010;
int pre[N];
int n, m;


int find(int k)
```

```
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i;//初始化并查集
    while (m--)
    {
        char op;
        int x, y;
        cin >> op >> x >> y;
        if (op == 'M') pre[find(x)] = find(y);
        else cout << (find(x) == find(y) ? "Yes" : "No") << endl;
    }
    return 0;
}
```

二、AcWing 837. 连通块中点的数量

【题目描述】

给定一个包含n个点（编号为$1 \sim n$）的无向图，初始时图中没有边。

现在要进行m个操作，操作共有三种：

- `C a b`，在点a和点b之间连一条边，a和b可能相等；
- `Q1 a b`，询问点a和点b是否在同一个连通块中，a和b可能相等；
- `Q2 a`，询问点a所在连通块中点的数量。

【输入格式】

第一行输入整数n和m。

接下来m行，每行包含一个操作指令，指令为`C a b`，`Q1 a b`或`Q2 a`中的一种。

【输出格式】

对于每个询问指令`Q1 a b`，如果a和b在同一个连通块中，则输出`Yes`，否则输出`No`。

对于每个询问指令`Q2 a`，输出一个整数表示点a所在连通块中点的数量。

每个结果占一行。

【数据范围】

$1 \leq n, m \leq 10^5$

【输入样例】

```
5 5
C 1 2
Q1 1 2
Q2 1
C 2 5
Q2 5
```

【输出样例】

```
Yes
2
3
```

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
using namespace std;

const int N = 100010;
int pre[N], cnt[N];
int n, m;

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i, cnt[i] = 1;
    while (m--)
    {
        string op;

        int a, b;
```

```
        cin >> op;
        if (op == "C")
        {
            cin >> a >> b;
            int px = find(a), py = find(b);
            if (px != py) cnt[py] += cnt[px], pre[px] = py;
        }
        else if (op == "Q1")
        {
            cin >> a >> b;
            find(a) == find(b) ? puts("Yes") : puts("No");
        }
        else
        {
            cin >> a;
            cout << cnt[find(a)] << endl;
        }
    }
    return 0;
}
```

三、AcWing 240. 食物链

【题目描述】

动物王国中有三类动物A,B,C，这三类动物的食物链构成了有趣的环形。

A吃B，B吃C，C吃A。

现有N个动物，以$1 \sim N$编号。

每个动物都是A,B,C中的一种，但是我们并不知道它到底是哪一种。

有人用两种说法对这N个动物所构成的食物链关系进行描述：

- 第一种说法是`1 X Y`，表示X和Y是同类；
- 第二种说法是`2 X Y`，表示X吃Y。

此人对N个动物，用上述两种说法，一句接一句地说出K句话，这K句话有的是真的，有的是假的。

当一句话满足下列三条之一时，这句话就是假话，否则就是真话。

- 当前的话与前面的某些真的话冲突，就是假话；
- 当前的话中X或Y比N大，就是假话；
- 当前的话表示X吃X，就是假话。

你的任务是根据给定的N和K句话，输出假话的总数。

【输入格式】

第一行是两个整数N和K，以一个空格分隔。

以下K行每行是三个正整数`D X Y`，两数之间用一个空格隔开，其中D表示说法的种类。

若$D=1$，则表示X和Y是同类。

若$D=2$，则表示X吃Y。

【输出格式】

只有一个整数，表示假话的数目。

【数据范围】

$1 \leq N \leq 50000$

$0 \leq K \leq 100000$

【输入样例】

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

【输出样例】

```
3
```

【维护集合中每个结点至根结点的距离写法代码】

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

const int N = 50010;
int pre[N], dis[N];
int n, k;

int find(int x)
```

```
{
    if (pre[x] != x)
    {
        //dis[pre[x]]不一定表示pre[x]到根节点的距离，find一遍后会更新，就表示的是到根节点的距离了
        int temp = find(pre[x]);
        dis[x] += dis[pre[x]];
        pre[x] = temp;
    }
    return pre[x];
}

int main()
{
    cin >> n >> k;
    for (int i = 1; i <= n; i++) pre[i] = i;
    int res = 0;
    while (k--)
    {
        //如果x到根的距离%3比y到根的距离%3多1则x吃y
        //如果x到根的距离%3比y到根的距离%3多2则y吃x
        int t, x, y;
        cin >> t >> x >> y;
        if (x > n || y > n) res++;//x或y不合法则为假
        else if (t == 1)//如果x与y为同类
        {
            int px = find(x), py = find(y);
            //如果x与y在同一个集合但dis[x]%3!=dis[y]%3则说明x与y是同类为假
            //由于余数正负号的问题，写为(dis[x]-dis[y])%3!=0
            if (px == py && (dis[x] - dis[y]) % 3) res++;
            else if (px != py)
            {
                pre[px] = py;
                dis[px] = dis[y] - dis[x];//(dis[x]+dis[px])%3==dis[y]%3
            }
        }
        else//如果x吃y
        {
            int px = find(x), py = find(y);
            //如果x与y在同一个集合但dis[x]%3-1!=dis[y]%3则说明x吃y为假
            if (px == py && (dis[x] - dis[y] - 1) % 3) res++;
            else if (px != py)

            {
```

```
                pre[px] = py;
                dis[px] = dis[y] - dis[x] +
1;//(dis[x]+dis[px]-1)%3==dis[y]%3
            }
        }
    }
    cout << res << endl;
    return 0;
}
```

【创建拓展域写法代码】

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

const int N = 150010;
int pre[N];
int n, k;

int find(int x)
{
    if (pre[x] == x) return x;
    return pre[x] = find(pre[x]);
}

void unite(int x, int y)
{
    pre[find(x)] = find(y);
}

int main()
{
    cin >> n >> k;
    //因为需要拓展为3个域，所以初始化3n个节点
    //x为同类域，x+n为捕食域，x+n+n为天敌域
    for (int i = 1; i <= 3 * n; i++) pre[i] = i;
    int res = 0;
    while (k--)
    {
        int t, x, y;
        cin >> t >> x >> y;
        if (x > n || y > n) res++;//x或y不合法则为假

        else if (t == 1)//如果x与y为同类
```

```
        {
            //如果x在y的捕食域和天敌域中则为假
            if (find(x) == find(y + n)) res++;
            else if (find(x) == find(y + n + n)) res++;
            else
            {
                unite(x, y);//x的同类域加入y
                unite(x + n, y + n);//x的捕食域加入y的捕食域
                unite(x + n + n, y + n + n);//x的天敌域加入y的天敌域
            }
        }
        else//如果x吃y
        {
            //如果x与y为同类或x在y的捕食域中则为假
            if (find(x) == find(y)) res++;
            else if (find(x) == find(y + n)) res++;
            else
            {
                unite(x, y + n + n);//x的同类域加入y的天敌域
                unite(x + n, y);//x的捕食域加入y
                unite(x + n + n, y + n);//x的天敌域加入y的捕食域
            }
        }
    }
    cout << res << endl;
    return 0;
}
```

数据结构-堆

一、AcWing 838. 堆排序

【题目描述】

输入一个长度为n的整数数列，从小到大输出前m小的数。

【输入格式】

第一行包含整数n和m。

第二行包含n个整数，表示整数数列。

【输出格式】

共一行，包含m个整数，表示整数数列中前m小的数。

【数据范围】

$1 \leq m \leq n \leq 10^5$

$1 \leq$ 数列中元素 $\leq 10^9$

【输入样例】

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

【输出样例】

```
1 2 3
```

【代码】

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

const int N = 100010;
int h[N], cnt;
int n, m;

void down(int u)
{
    int t = u;
    if (u * 2 <= cnt && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= cnt && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u != t)
    {
        swap(h[u], h[t]);
        down(t);
    }
}

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

    for (int i = 1; i <= n; i++) cin >> h[i];
```

```
    //从非叶结点开始down
    for (int i = n / 2; i; i--) down(i);
    while (m--)
    {
        cout << h[1] << ' ';
        h[1] = h[cnt--];
        down(1);
    }
    return 0;
}
```

二、AcWing 839. 模拟堆

【题目描述】

维护一个集合，初始时集合为空，支持如下几种操作：

- `I x`，插入一个数x；
- `PM`，输出当前集合中的最小值；
- `DM`，删除当前集合中的最小值（数据保证此时的最小值唯一）；
- `D k`，删除第k个插入的数；
- `C k x`，修改第k个插入的数，将其变为x。

现在要进行N次操作，对于所有第2个操作，输出当前集合的最小值。

【输入格式】

第一行包含整数N。

接下来N行，每行包含一个操作指令，操作指令为`I x`，`PM`，`DM`，`D k`或`C k x`中的一种。

【输出格式】

对于每个输出指令`PM`，输出一个结果，表示当前集合中的最小值。

每个结果占一行。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq x \leq 10^9$

数据保证合法。

【输入样例】

```
8
I -10
PM
I -10
D 1
C 2 8
I 6
PM
DM
```

【输出样例】

```
-10
6
```

【代码】

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

const int N = 100010;
int h[N], ph[N], hp[N], cnt;
int n, m, k, x;
string op;

void heap_swap(int a, int b)
{
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}

void down(int u)
{
    int t = u;
    if (u * 2 <= cnt && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= cnt && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u != t)
    {
        heap_swap(u, t);
        down(t);
```

```
    }
}

void up(int u)
{
    while (u / 2 && h[u / 2] > h[u])
    {
        heap_swap(u, u / 2);
        u /= 2;
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n;
    while (n--)
    {
        cin >> op;
        if (op == "I")
        {
            cin >> x;
            ph[++m] = ++cnt;
            hp[cnt] = m;
            h[cnt] = x;
            up(cnt);
        }
        else if (op == "PM") cout << h[1] << endl;
        else if (op == "DM")
        {
            heap_swap(1, cnt);
            cnt--;
            down(1);
        }
        else if (op == "D")
        {
            cin >> k;
            k = ph[k];
            heap_swap(k, cnt);
            cnt--;
            down(k), up(k);
        }

        else
```

```
        {
            cin >> k >> x;
            k = ph[k];
            h[k] = x;
            down(k), up(k);
        }
    }
    return 0;
}
```

数据结构-哈希表

一、AcWing 840. 模拟散列表

【题目描述】

维护一个集合，支持如下几种操作：

- `I x`，插入一个数x；
- `Q x`，询问数x是否在集合中出现过。

现在要进行N次操作，对于每个询问操作输出对应的结果。

【输入格式】

第一行包含整数N，表示操作数量。

接下来N行，每行包含一个操作指令，操作指令为`I x`，`Q x`中的一种。

【输出格式】

对于每个询问指令`Q x`，输出一个询问结果，如果x在集合中出现过，则输出`Yes`，否则输出`No`。

每个结果占一行。

【数据范围】

$1 \leq N \leq 10^5$

$-10^9 \leq x \leq 10^9$

【输入样例】

```
5
I 1
I 2
I 3
Q 2
Q 5
```

【输出样例】

```
Yes
No
```

【拉链法代码】

```cpp
#include <iostream>
#include <cstring>
#include <string>
using namespace std;

const int N = 100003;
int e[N], ne[N], h[N], idx;
int n, x;
string op;

void insert(int x)
{
    int k = (x % N + N) % N;
    e[idx] = x, ne[idx] = h[k], h[k] = idx++;
}

bool find(int x)
{
    int k = (x % N + N) % N;
    for (int i = h[k]; ~i; i = ne[i])
        if (e[i] == x) return true;
    return false;
}

int main()
{
    cin >> n;
    memset(h, -1, sizeof h);
    while (n--)
```

```
    {
        cin >> op >> x;
        if (op == "I") insert(x);
        else if (find(x)) cout << "Yes" << endl;
        else cout << "No" << endl;
    }
    return 0;
}
```

【开放寻址法代码】

```
#include <iostream>
#include <cstring>
#include <string>
using namespace std;

const int N = 200003;
const int INF = 0x3f3f3f3f;
int n, x, h[N];
string op;

//若存在x则返回x所在的位置，否则返回x应该插入的位置
int find(int x)
{
    int k = (x % N + N) % N;
    while (h[k] != x && h[k] != INF)
        if (k++ > N) k = 0;
    return k;
}

int main()
{
    cin >> n;
    memset(h, 0x3f, sizeof h);
    while (n--)
    {
        cin >> op >> x;
        int idx = find(x);
        if (op == "I") h[idx] = x;
        else if (h[idx] != INF) cout << "Yes" << endl;
        else cout << "No" << endl;
    }

    return 0;
```

```
}
```

二、AcWing 841. 字符串哈希

【题目描述】

给定一个长度为n的字符串，再给定m个询问，每个询问包含四个整数l_1,r_1,l_2,r_2，请你判断$[l_1,r_1]$和$[l_2,r_2]$这两个区间所包含的字符串子串是否完全相同。

字符串中**只包含大小写英文字母和数字**。

【输入格式】

第一行包含整数n和m，表示字符串长度和询问次数。

第二行包含一个长度为n的字符串，字符串中只包含大小写英文字母和数字。

接下来m行，每行包含四个整数l_1,r_1,l_2,r_2，表示一次询问所涉及的两个区间。

注意，字符串的位置从1开始编号。

【输出格式】

对于每个询问输出一个结果，如果两个字符串子串完全相同则输出`Yes`，否则输出`No`。

每个结果占一行。

【数据范围】

$1 \leq n,m \leq 10^5$

【输入样例】

```
8 3
aabbaabb
1 3 5 7
1 3 6 8
1 2 1 2
```

【输出样例】

```
Yes
No
Yes
```

【代码】

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

typedef unsigned long long ULL;
const int N = 100010, P = 131;
char str[N];
int n, m;
ULL h[N], p[N];//p[i]表示P^i

ULL getHash(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m >> str + 1;
    p[0] = 1;//P^0=1
    for (int i = 1; i <= n; i++)
    {
        h[i] = h[i - 1] * P + str[i];
        p[i] = p[i - 1] * P;
    }
    while (m--)
    {
        int l1, r1, l2, r2;
        cin >> l1 >> r1 >> l2 >> r2;
        if (getHash(l1, r1) == getHash(l2, r2)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```