Week5、6 总结：

1.大数据用 long long，否则会爆 int

2.关同步，不然有可能超时 https://www.cnblogs.com/cytus/p/7763569.html

3.kruskal 结合并查集很方便

4.图的存储方式里面邻接矩阵存起来方便但具体算法实现起来可能稍复杂，邻接链表虽然结构本身比较复杂，但是可自定义的东西很多。

5.双端队列 deque 的用法

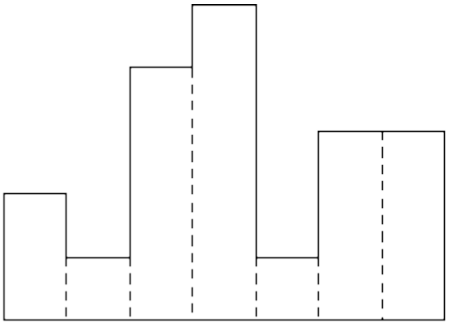
6.很多问题换个角度就很好理解了，比如平衡字符串那道题，利用最后空闲字符数判断是否可以替换达到平衡；

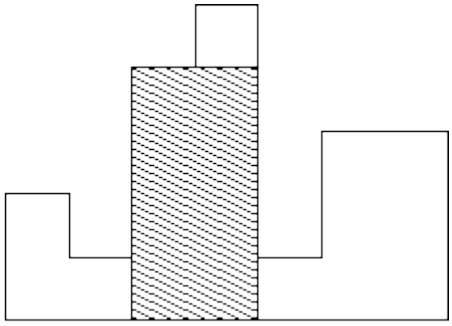
7.尺取法要理解

Week5

A - 最大矩形 HDU - 1506

给一个直方图，求直方图中的最大矩形的面积。例如，下面这个图片中直方图的高度从左到右分别是 2, 1, 4, 5, 1, 3, 3，他们的宽都是 1，其中最大的矩形是阴影部分。





Input

输入包含多组数据。每组数据用一个整数 n 来表示直方图中小矩形的个数，你可以假定 1 <= n <= 100000. 然后接下来 n 个整数 h1, ..., hn，满足 0 <= hi <= 1000000000. 这些数字表示直方图中从左到右每个小矩形的高度，每个小矩形的宽度为 1。 测试数据以 0 结尾。

Output

对于每组测试数据输出一行一个整数表示答案。

Sample Input

7 2 1 4 5 1 3 3

4 1000 1000 1000 1000

0

Sample Output

8

4000

思路：

定义矩形结构体，对读入的每个小矩形（高度），记录此高度和对应的左右端点。左端点 l 为往左第一个小于此高度的点，右端点 r 为往右第一个小于此高度的点。这个过程可以用单调栈来实现。

如左端点，每当要加入的小矩形高度小于栈顶元素的高度，则取出栈顶，将其对应的矩形的左端点设为当前要加入的小矩形的坐标位置。需要将里面所有的矩形左端点设为-1。右端点也是相似的过程。

最后得到左右端点 l 和 r，r-l-1 即为此高度矩形最大面积，最后遍历得到的所有面积找出最大的就是答案；

还要注意，面积注意用 long long 型；

代码：

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

```
struct rectangle {
    int index;
    long long int height;  //数很大，用 long long
};

int n;
long long int ans;
int _left[100001], _right[100001];  //左端点与右端点

rectangle p[100001],cur;
stack<rectangle> s;

int main() {
    while (cin >> n) {
        if (n == 0) break;
        for (int i = 0; i < n; i++) {
            p[i].index = i;
            cin >> p[i].height;
        }
        ans = 0;

        //左端点
        for (int i = n - 1; i >= 0; i--) {  //往左数第一个小于 h[i]的点
            while (s.size() > 0 && s.top().height > p[i].height) {
                cur = s.top();
                s.pop();
                _left[cur.index] = i;
            }
            s.push(p[i]);
        }
        while (!s.empty()) {
            cur = s.top();
            s.pop();
            _left[cur.index] = -1;
        }
        //右端点
        for (int i = 0; i < n; i++) {  //往右数第一个小于 h[i]的点
            while (s.size() > 0 && s.top().height > p[i].height) {
                cur = s.top();
                s.pop();
                _right[cur.index] = i;
            }
            s.push(p[i]);
        }
```

```
        while (!s.empty()) {
            cur = s.top();
            s.pop();
            _right[cur.index] = n;
        }
        //长度为_right-_left-1
        for (int i = 0; i < n; i++) {
            long long int ss = (_right[i] - _left[i] - 1) * p[i].height;
            if (ss > ans) ans = ss;
        }
        cout << ans << endl;
    }
}
```

B - TT's Magic Cat Gym - 272542A

Thanks to everyone's help last week, TT finally got a cute cat. But what TT didn't expect is that this is a magic cat.

One day, the magic cat decided to investigate TT's ability by giving a problem to him. That is select nn cities from the world map, and a[i]a[i] represents the asset value owned by the ii-th city.

Then the magic cat will perform several operations. Each turn is to choose the city in the interval [l,r][l,r] and increase their asset value by cc. And finally, it is required to give the asset value of each city after qq operations.

Could you help TT find the answer?

Input

The first line contains two integers n,qn,q ($1 \leqslant n,q \leqslant 2 \cdot 105$)($1 \leqslant n,q \leqslant 2 \cdot 105$) — the number of cities and operations.

The second line contains elements of the sequence aa: integer numbers a1,a2,...,ana1,a2,...,an ($-106 \leqslant ai \leqslant 106$)($-106 \leqslant ai \leqslant 106$).

Then qq lines follow, each line represents an operation. The ii-th line contains three integers l,rl,r and cc ($1 \leqslant l \leqslant r \leqslant n$, $-105 \leqslant c \leqslant 105$)($1 \leqslant l \leqslant r \leqslant n$, $-105 \leqslant c \leqslant 105$) for the ii-th operation.

Output

Print nn integers a1,a2,⋯,ana1,a2,⋯,an one per line, and aiai should be equal to the final asset value of the ii-th city.

Examples

Input

4 2

-3 6 8 4
4 4 -2
3 3 1
Output
-3 6 9 2
Input
2 1
5 -2
1 2 4
Output
9 2
Input
1 2
0
1 1 -8
1 1 -6
Output
-14

思路：

题意大概就是同时让数组里面某区间的所有数同时+c，第一时间用暴力硬改，超时，然后老老实实用差分数组。

核心为 b[0]=a[0]; b[i]=a[i]-a[i-1]; //(i>0)，从中可以看出 city[i]数组=差分数组 a[i]前 i 项和。让 city 数组中某个区间的元素全部+c，只需要让 a[l]+c，再让 a[r+1]-c 就行。

最后从 city[1]开始向后每个元素+a[i]，得到最终修改后的结果。

代码：

```
#include <iostream>
using namespace std;
long long t;
int n, q;
long long city[1000010], a[1000010];
int main() {
    while (cin >> n >> q) {
        for (int i = 0; i < n; i++) {
            cin >> city[i];
        }
        a[0] = city[0];
        for (int i = 1; i < n; i++) {
            a[i] = city[i] - city[i - 1];
        }
        int l, r, c;
        while (q--) {
            cin >> l >> r >> c;
            a[l - 1] += c;
            a[r] -= c;
```

```
        }
        cout << a[0] << " ";
        t = a[0];
        for (int i = 1; i < n; i++) {
            t = t + a[i];
            cout << t << " ";
        }
        cout << endl;
    }
}
```

C - 平衡字符串 Gym - 270737B

一个长度为 n 的字符串 s，其中仅包含 'Q', 'W', 'E', 'R' 四种字符。
如果四种字符在字符串中出现次数均为 n/4，则其为一个平衡字符串。
现可以将 s 中连续的一段子串替换成相同长度的只包含那四个字符的任意字符串，使其变为一个平衡字符串，问替换子串的最小长度?
如果 s 已经平衡则输出 0。

Input

一行字符表示给定的字符串 s

Output

一个整数表示答案

Examples

Input
QWER
Output
0
Input
QQWE
Output
1
Input
QQQW
Output
2
Input
QQQQ
Output
3

Note

1<=n<=10^5

n 是 4 的倍数
字符串中仅包含字符 'Q', 'W', 'E' 和 'R'.

思路：

题意大概是替换字符串中的某连续部分为 Q、W、E、R，使得字符串中每种字母的数量相同，求这个最小长度。

所求结果为一个最小连续区间，可以使用尺取法：若当前区间可以通过替换达到平衡，则 l++(l==r 时 l++,r++)；否则 r++。

然后就是判断的部分。思路为通过替换使得 4 类字符数量一致，再判断剩余是否是 4 的倍数。首先用 sumQ、sumW、sumE、sumR 记录字符串里 QWER 的个数，用 sumQQ、sumWW、sumEE、sumRR 记录[l,r]内的 QWER 个数。记 max 是 sumQQ~sumRR 中最大的，则(max-sumQQ)+(ma-sumWW)+(max-sumEE)+(max-sumRR)是[l,r]区间应给区间外的字符串补充的字符个数。当前区间[l,r]字符总数=r-l+1，减去要补充的字符数，还剩下一些空闲位置。若空闲位置数量>=0 且为 4 的倍数，就可以通过替换达到平衡。

代码：

```
#include <iostream>
#include <string>
using namespace std;
string s;
int sumQ, sumW, sumE, sumR;
int ans = 100000;
int main() {
    while (cin >> s) {
        sumQ = 0, sumW = 0, sumE = 0, sumR = 0;
        int len = s.size();
        for (int i = 0; i < len; i++) {
            if (s[i] == 'Q')
                sumQ++;
            else if (s[i] == 'W')
                sumW++;
            else if (s[i] == 'E')
                sumE++;
            else if (s[i] == 'R')
                sumR++;
        }

        if (sumQ == sumW && sumW == sumE && sumE == sumR)  //当前已经平衡,结束
        {
            cout << "0" << endl;
            continue;
        }
        int left = 0, right = 0;
        while (right <= len - 1) {
            //计算 l,r 之间的 QWER 个数
            int sumQQ = 0, sumWW = 0, sumEE = 0, sumRR = 0;
```

```
            for (int i = left; i <= right; i++) {
                if (s[i] == 'Q')
                    sumQQ++;
                else if (s[i] == 'W')
                    sumWW++;
                else if (s[i] == 'E')
                    sumEE++;
                else if (s[i] == 'R')
                    sumRR++;
            }
            sumQQ = sumQ - sumQQ;
            sumWW = sumW - sumWW;
            sumEE = sumE - sumEE;
            sumRR = sumR - sumRR;
            int max = 0;
            if (sumQQ > max) max = sumQQ;
            if (sumWW > max) max = sumWW;
            if (sumEE > max) max = sumEE;
            if (sumRR > max) max = sumRR;
            int total = right - left + 1;
            int free = total - ((max - sumQQ) + (max - sumWW) + (max - sumEE) +
                                (max - sumRR));
            //满足要求的
            if (free >= 0 && free % 4 == 0) {
                if (total < ans) ans = total;
                if (left == right) {
                    left++;
                    right++;
                } else
                    left++;
            } else
                right++;
        }
        cout << ans << endl;
        s.clear();
    }
}
```

D - 滑动窗口（C++和 G++都交一下） POJ - 2823

ZJM 有一个长度为 n 的数列和一个大小为 k 的窗口，窗口可以在数列上来回移动．现在 ZJM 想知道在窗口从左往右滑的时候，每次窗口内数的最大值和最小值分别是多少．例如：
数列是 [1 3 -1 -3 5 3 6 7]，其中 *k* 等于 3.

Window position	Minimum value	Maximum value
[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

Input

输入有两行。第一行两个整数 n 和 k 分别表示数列的长度和滑动窗口的大小，1<=k<=n<=1000000。第二行有 n 个整数表示 ZJM 的数列。

Output

输出有两行。第一行输出滑动窗口在从左到右的每个位置时，滑动窗口中的最小值。第二行是最大值。

Sample Input

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

Sample Output

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

思路：

题意就是求数组特定长度区间内的最大数和最小数。

首先暴力不可取，会超时。然后想到可以采取单调队列，使用 deque 队列可以同时操作队首队尾。单调队列，即单调递减或单调递增的队列。 队列中的元素在原来的列表中的位置是由前往后的(随着循环顺序入队)，队列中元素的大小是单调递增或递减的。以单调增队列为例，若队列为空或是新的元素大于队列尾元素，则新元素入队；否则如果此元素入队列，则不满队列内元素单调的性质，因此我们从队列尾开始将所有大于这个新元素的元素出队，然后将它入队，由于单调队列的性质，此时这个队列头则为这个局部区间内最小的元素，队尾为最大元素。由于单调队列是窗口，有大小限制，因此窗口滑动时队首元素也要出队，同时也不会不影响队列内部单调性。而这样依次滑动得到的每个队列的队首元素则为这个区间内的最小的元素，队尾元素为最大元素。

记得关同步：std::ios::sync_with_stdio(false);或者只用 scanf\printf。

代码：

```
#include <stdio.h>
#include <iostream>
#include <queue>
```

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

deque<long long> q;
long long n, k;

int main() {
    std::ios::sync_with_stdio(false);
    while (~scanf("%lld %lld", &n, &k)) {  //忘了加"~"
        vector<int> a(n);
        for (long long i = 0; i < n; i++) scanf("%d", &a[i]);
        for (long long i = 0; i < k - 1; i++)  //先放进去前 k-1 个
        {
            while (!q.empty() && a[q.back()] > a[i]) q.pop_back();
            q.push_back(i);
        }
        for (long long i = k - 1; i < n; i++) {
            while (!q.empty() && a[q.back()] > a[i]) q.pop_back();
            q.push_back(i);
            if (i - q.front() + 1 > k) q.pop_front();
            printf("%d ", a[q.front()]);
        }
        q.clear();
        cout << endl;
        for (long long i = 0; i < k - 1; i++)  //先放进去前 k-1 个
        {
            while (!q.empty() && a[q.back()] < a[i]) q.pop_back();
            q.push_back(i);
        }
        for (long long i = k - 1; i < n; i++) {
            while (!q.empty() && a[q.back()] < a[i]) q.pop_back();
            q.push_back(i);
            if (i - q.front() + 1 > k) q.pop_front();
            printf("%d ", a[q.front()]);
        }
        cout << endl;
    }
}
```

CSP-201512-3-画图

问题描述

用 ASCII 字符来画图是一件有趣的事情，并形成了一门被称为 ASCII Art 的艺术。例如，下图是用 ASCII 字符画出来的 CSPRO 字样。

```
..____.____..____..____...___..
./.___/.___||..._.\|..._.\./._.\.
|.|...\___.\|.|_).|.|_).|.|.|.|
|.|___.___).|..__/|.._.<|.|_|.|
.\____|____/|_|...|_|.\_\\___/.
```

本题要求编程实现一个用 ASCII 字符来画图的程序，支持以下两种操作：

画线：给出两个端点的坐标，画一条连接这两个端点的线段。简便起见题目保证要画的每条线段都是水平或者竖直的。水平线段用字符 - 来画，竖直线段用字符 | 来画。如果一条水平线段和一条竖直线段在某个位置相交，则相交位置用字符 + 代替。

填充：给出填充的起始位置坐标和需要填充的字符，从起始位置开始，用该字符填充相邻位置，直到遇到画布边缘或已经画好的线段。注意这里的相邻位置只需要考虑上下左右 4 个方向，如下图所示，字符 @ 只和 4 个字符 * 相邻。

```
.*.
*@*
.*.
```

输入格式

第 1 行有三个整数 m, n 和 q。m 和 n 分别表示画布的宽度和高度，以字符为单位。q 表示画图操作的个数。

第 2 行至第 q + 1 行，每行是以下两种形式之一：

0 x_1 y_1 x_2 y_2：表示画线段的操作，(x_1, y_1)和(x_2, y_2)分别是线段的两端，满足要么 $x_1 = x_2$ 且 $y_1 \neq y_2$，要么 $y_1 = y_2$ 且 $x_1 \neq x_2$。

1 x y c：表示填充操作，(x, y)是起始位置，保证不会落在任何已有的线段上；c 为填充字符，是大小写字母。

画布的左下角是坐标为 (0, 0) 的位置，向右为 x 坐标增大的方向，向上为 y 坐标增大的方向。这 q 个操作按照数据给出的顺序依次执行。画布最初时所有位置都是字符 .（小数点）。

输出格式

输出有 n 行，每行 m 个字符，表示依次执行这 q 个操作后得到的画图结果。

样例输入

```
4 2 3
1 0 0 B
0 1 0 2 0
1 0 0 A
```

样例输出

```
AAAA
A--A
```

样例输入

```
16 13 9
0 3 1 12 1
0 12 1 12 3
0 12 3 6 3
0 6 3 6 9
0 6 9 12 9
```

```
0 12 9 12 11
0 12 11 3 11
0 3 11 3 1
1 4 2 C
```

样例输出

```
.................
...+--------+...
...|CCCCCCCC|...
...|CC+-----+...
...|CC|.........
...|CC|.........
...|CC|.........
...|CC|.........
...|CC|.........
...|CC+-----+...
...|CCCCCCCC|...
...+--------+...
.................
```

评测用例规模与约定

所有的评测用例满足：$2 \leqslant m, n \leqslant 100$，$0 \leqslant q \leqslant 100$，$0 \leqslant x < m$（$x$ 表示输入数据中所有位置的 x 坐标），$0 \leqslant y < n$（y 表示输入数据中所有位置的 y 坐标）。

思路：

开一个 map 二维数组，一开始根据输入 m,n 初始化所有范围内的点为“.”。

画线段时根据 x1 和 x2 是否相等判断是竖线还是横线，用循环操作分别相应划线，标记未标记的点（visited0），注意如果已标记位置如果与当前进行的操作（横线\竖线）不同，则要画“+”。

填充时，用 bfs 配合标记数组 visited1 来完成。

最后输出地图内容。

代码：

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

int m, n, q;
char map[102][102];
bool visited0[102][102];
bool visited1[102][102];
int dx[4] = {0, 0, 1, -1};
int dy[4] = {-1, 1, 0, 0};
struct point {
```

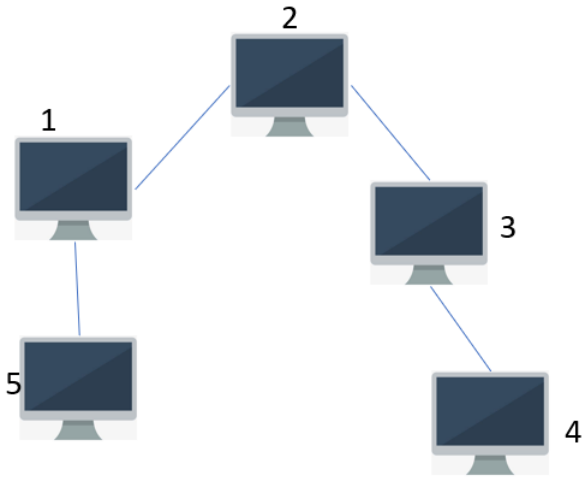
```
    int x_;
    int y_;
};
queue<point> que;
void bfs(int x, int y, char c) {
    memset(visited1, 0, sizeof(visited1));
    //起始位置
    point p;
    p.x_ = x;
    p.y_ = y;
    visited1[x][y] = true;
    map[x][y] = c;
    que.push(p);
    while (!que.empty()) {
        point cur = que.front();
        que.pop();
        for (int i = 0; i < 4; i++) {
            point _new;
            int _x = cur.x_ + dx[i];
            int _y = cur.y_ + dy[i];
            if (visited1[_x][_y] == false && visited0[_x][_y] == false &&
                _x >= 0 && _x < m && _y >= 0 && _y < n) {
                visited1[_x][_y] = true;
                _new.x_ = _x;
                _new.y_ = _y;
                map[_x][_y] = c;
                que.push(_new);
            }
        }
    }
}

int main() {
    while (cin >> m >> n >> q) {
        int op, x1, y1, x2, y2, x, y;
        char c;
        memset(visited0, 0, sizeof(visited0));
        //初始化画图区域
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++) map[i][j] = '.';
        for (int i = 0; i < q; i++) {
            cin >> op;
            if (op == 0) {  //画线段
                cin >> x1 >> y1 >> x2 >> y2;
```

```
                if (x1 == x2) {  //画竖线
                    if (y1 > y2) {
                        int t = y1;
                        y1 = y2;
                        y2 = t;
                    }
                    for (int k = y1; k <= y2; k++) {
                        if (visited0[x1][k] == false) {  //没有画过线段
                            visited0[x1][k] = true;
                            map[x1][k] = '|';
                        } else if (map[x1][k] == '-')
                            map[x1][k] = '+';
                    }
                } else {
                    if (x1 > x2) {
                        int t = x1;
                        x1 = x2;
                        x2 = t;
                    }
                    for (int k = x1; k <= x2; k++) {
                        if (visited0[k][y1] == false) {  //没有画过线段
                            visited0[k][y1] = true;
                            map[k][y1] = '-';
                        } else if (map[k][y1] == '|')
                            map[k][y1] = '+';
                    }
                }
            } else {                //填充
                cin >> x >> y >> c;  //起始位置是 x,y
                bfs(x, y, c);
            }
        }
        for (int j = n - 1; j >= 0; j--) {  //输出
            for (int i = 0; i < m; i++) {
                cout << map[i][j];
            }
            cout << endl;
        }
    }
    // system("pause");
    return 0;
}
```

Week6

A - 氪金带东 HDU - 2196

实验室里原先有一台电脑(编号为 1)，最近氪金带师咕咕东又为实验室购置了 N-1 台电脑，编号为 2 到 N。每台电脑都用网线连接到一台先前安装的电脑上。但是咕咕东担心网速太慢，他希望知道第 i 台电脑到其他电脑的最大网线长度，但是可怜的咕咕东在不久前刚刚遭受了宇宙射线的降智打击，请你帮帮他。



提示：样例输入对应这个图，从这个图中你可以看出，距离 1 号电脑最远的电脑是 4 号电脑，他们之间的距离是 3。 4 号电脑与 5 号电脑都是距离 2 号电脑最远的点，故其答案是 2。5 号电脑距离 3 号电脑最远，故对于 3 号电脑来说它的答案是 3。同样的我们可以计算出 4 号电脑和 5 号电脑的答案是 4.

Input

输入文件包含多组测试数据。对于每组测试数据，第一行一个整数 N (N<=10000)，接下来有 N-1 行，每一行两个数，对于第 i 行的两个数，它们表示与 i 号电脑连接的电脑编号以及它们之间网线的长度。网线的总长度不会超过 10^9，每个数之间用一个空格隔开。

Output

对于每组测试数据输出 N 行，第 i 行表示 i 号电脑的答案 (1<=i<=N).

Sample Input

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

Sample Output

```
3
2
3
4
```

4

思路：

题意就是求每个点和离它最远的点之间的距离。

可将题目中的电脑想象成点，线想象成带权值的边（权值为距离），整个图就变成了一棵树，这样问题就变成加权无向图中任一点到最远点距离，假设树的最长路的两个叶子节点为 v1,v2,从任意一点 u 出发走到的最远点一定是 v2 或者 v1。由此经过两次遍历就能找到最长路径。节点 x 的最长路径要么是到 v1 的路径，要么就是到 v2 的路径。

由于不知道往哪边走才是最长的路径，所以需要分别从 v1, v2 点再次遍历来确定哪一边是最长路线，所以共需要三次遍历（第一次 dfs 走到离指定初始点最远的一个点，也就是两个最长路径端点 v1v2 之一，然后再从端点进行一遍 dfs 到另一个端点，最后一次再 dfs 回来），之后可以得到每个点离他最远点的距离（存在 length 数组内，由 dfs 函数中 max 函数位置进行更正最大长度）。

代码：

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

struct node {
    int nextNode;
    int weight;
};
int length[10001];
int maxLen, s;
vector<node> p[10001];
void DFS(int u, int fa, int len) {
    if (maxLen <= len) {
        s = u;
        maxLen = len;
    }
    int Len = p[u].size();
    for (int i = 0; i < Len; i++) {
        int v = p[u][i].nextNode;
        if (v == fa) continue;
        DFS(v, u, len + p[u][i].weight);
        length[v] = max(length[v], len + p[u][i].weight);
    }
}
int main() {
    int n;
    while (cin >> n) {
        for (int i = 0; i <= n; i++) {
```

```
            p[i].clear();
        }
        for (int i = 2; i <= n; i++) {
            int x, y;
            cin >> x >> y;
            node temp;
            temp.nextNode = x;
            temp.weight = y;
            p[i].push_back(temp);
            temp.nextNode = i;
            p[x].push_back(temp);
        }
        memset(length, 0, sizeof(length));
        maxLen = 0;
        s = 0;
        DFS(1, -1, 0);
        DFS(s, -1, 0);
        DFS(s, -1, 0);
        for (int i = 1; i <= n; i++) {
            cout << length[i] << endl;
        }
    }
    return 0;
}
```

B - 戴好口罩！ POJ - 1611

新型冠状病毒肺炎（Corona Virus Disease 2019，COVID-19），简称“新冠肺炎”，是指 2019 新型冠状病毒感染导致的肺炎。
如果一个感染者走入一个群体，那么这个群体需要被隔离！
小 A 同学被确诊为新冠感染，并且没有戴口罩！！！！！
危！！！
时间紧迫！！！！
需要尽快找到所有和小 A 同学直接或者间接接触过的同学，将他们隔离，防止更大范围的扩散。
众所周知，学生的交际可能是分小团体的，一位学生可能同时参与多个小团体内。
请你编写程序解决！戴口罩！！

Input

多组数据，对于每组测试数据：
第一行为两个整数 n 和 m（n = m = 0 表示输入结束，不需要处理），n 是学生的数量，m 是学生群体的数量。0 < n <= 3e4 ， 0 <= m <= 5e2
学生编号为 0~n-1
小 A 编号为 0

随后，m 行，每行有一个整数 num 即小团体人员数量。随后有 num 个整数代表这个小团体的学生。

Output

输出要隔离的人数，每组数据的答案输出占一行

Sample Input

100 4
2 1 2
5 10 13 11 12 14
2 0 1
2 99 2
200 2
1 5
5 1 2 3 4 5
1 0
0 0

Sample Output

4
1
1

思路：

很明显这是一道考并查集的题。

首先将每个学生作为一个独立的分组，即 father[i]=i。接着将每一个团体内的学生所在分组合并，同时更新分组内学生个数。 最后得到需要隔离的群体的人数。

代码：

```
#include <iostream>
using namespace std;
int father[30005], _rank[30005];
int find(int x) {
    if (father[x] == x) return x;
    return father[x] = find(father[x]);
}
void combine(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (_rank[x] > _rank[y]) swap(y, x);
    father[x] = y;
    _rank[x] = (_rank[y] += _rank[x]);
}
int n, m;
int main() {
    while (cin >> n >> m) {
        if (n == 0 && m == 0) {
            break;
```

```
        }

        for (int i = 0; i < n; i++) _rank[i] = 1;
        for (int i = 0; i < n; i++) father[i] = i;
        for (int i = 0; i < m; i++) {
            int num;
            cin >> num;
            int stu1, stu2;
            cin >> stu1;
            for (int j = 0; j < num - 1; j++) {
                cin >> stu2;
                combine(stu1, stu2);
                stu1 = stu2;
            }
        }
        cout << _rank[find(0)] << endl;
    }
    return 0;
}
```

C - 掌握魔法の东东 I Gym - 270437H

东东在老家农村无聊，想种田。农田有 n 块，编号从 1~n。种田要灌氵
众所周知东东是一个魔法师，他可以消耗一定的 MP 在一块田上施展魔法，使得黄河之水天上来。他也可以消耗一定的 MP 在两块田的渠上建立传送门，使得这块田引用那块有水的田的水。 (1<=n<=3e2)
黄河之水天上来的消耗是 Wi，i 是农田编号 (1<=Wi<=1e5)
建立传送门的消耗是 Pij，i、j 是农田编号 (1<= Pij <=1e5, Pij = Pji, Pii =0)
东东为所有的田灌氵的最小消耗

Input

第 1 行：一个数 n
第 2 行到第 n+1 行：数 wi
第 n+2 行到第 2n+1 行：矩阵即 pij 矩阵

Output

东东最小消耗的 MP 值

Example

Input

```
4
5
4
4
3
0 2 2 2
2 0 3 3
```

2 3 0 4
2 3 4 0
Output
9

思路：

最小生成树+并查集

定义边结构体，包含两个点（农田）和一个权重（引水花费），输入的时候把黄河水这个点的边（定义为 0 号农田到 i 号农田）和农田之间的边存储到边集中，之后按照花费进行排序再 kruskal 找到最小生成树；

Kruskal 部分利用并查集判断当前边的两个点是否都已经引水完成来添加新边，当加入所有点时输出总花费。

代码：

```
#include <algorithm>
#include <iostream>
using namespace std;
struct edge {
    int u, v, w;
    bool operator<(edge &e) { return w < e.w; }
};
int n, a, total;
int p[400];  //并查集
edge e[100001];
int find(int x) {
    if (p[x] == x) return x;
    return p[x] = find(p[x]);
}
bool combine(int x, int y);
void kruskal(int n);
int main() {
    while (cin >> n) {
        for (int i = 1; i <= n; i++) {
            cin >> a;
            total++;
            e[total].u = 0;
            e[total].v = i;
            e[total].w = a;
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cin >> a;
                if (i != j) {
                    total++;
                    e[total].u = i;
                    e[total].v = j;
```

```
                    e[total].w = a;
                }
            }
        }
        sort(e + 1, e + 1 + total);
        kruskal(n);
    }
    return 0;
}
bool combine(int x, int y) {
    int a = find(x), b = find(y);
    if (a == b) return false;
    p[a] = b;
    return true;
}
void kruskal(int n) {
    for (int i = 0; i <= n; i++) p[i] = i;
    int num = 0, sum = 0;  // e[index]
    for (int i = 1; i <= total; i++) {
        if (combine(e[i].u, e[i].v))  //不在同一个集合中
        {
            num++;
            sum += e[i].w;
        }
        if (num == n) {
            cout << sum << endl;
            return;
        }
    }
}
```

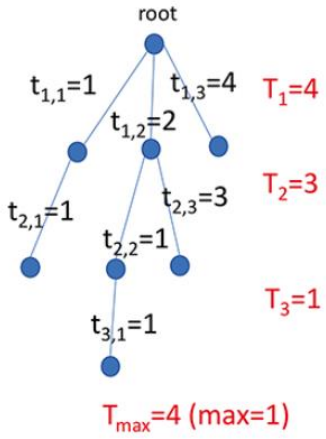
D - 数据中心 Gym - 270437I

【知识背景】

在一个集中式网络中，存在一个根节点，需要长时间接收其余所有节点传输给它的反馈数据。

【题目描述】

存在一个 n 节点的网络图，编号从 1 到 n。该网络的传输是全双工的，所以是无向图。如果两节点 v_i, u_i 相连，表明 v_i, u_i 之间可以互相收发数据，边权是传输数据所需时间 t_i。**现在每个节点需要选择一条路径将数据发送到 $root$ 号节点。**希望求出一个最优的树结构传输图，使得完成这个任务所需要的时间最少。$root$ 节点只能接收数据，其余任何一个节点可以将数据传输给另外的一个节点，但是不能将数据传输给多个节点。所有节点可以接收多个不同节点的数据。



一个树结构传输图的传输时间为 T_{max}，其中 $T_{max} = \max(T_h)$，h 为接收点在树中的深度，$T_h = \max(t_{h,j})$，$t_{h,j}$ 表示 j 条不同的边，这 j 条边接收点的深度都为 h。

【输入格式】

从标准输入读入数据。

输入的第 1 行包含一个正整数 n，保证 $n \le 5 \times 10^4$。

输入的第 2 行包含一个正整数 m，保证 $m \le 10^5$。

输入的第 3 行包含一个正整数 $root$，保证 $root \le 5 \times 10^4$。

【输出格式】

输出到标准输出。

输出仅有一行，包含一个正整数 ans，表示最优的树结构流水线所耗时 T_{max}。

Example

Input

4

5

1

1 2 3

1 3 4

1 4 5

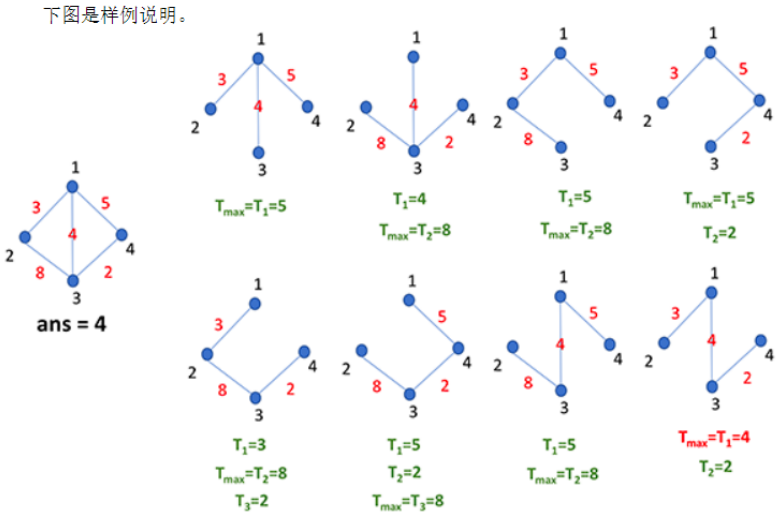
2 3 8

3 4 2

Output

4

Note



【子任务】

子任务 1（20 分）：$n \le 10^2$，$m \le 4,500$。

子任务 2（30 分）：$n \le 10^3$，$m \le 3 \times 10^4$。

子任务 3（20 分）：$n \le 10^4$，$m \le 5 \times 10^4$。

子任务 4（30 分）：$n \le 5 \times 10^4$，$m \le 10^5$。

所有子任务：v_i, u_i, t_i，保证 $v_i \le 5 \times 10^4$，$u_i \le 5 \times 10^4$，$t_i \le 10^6$，$u_i \ne v_i$

思路：

最小生成树

题意就是找到一个根为 root 的最小生成树

方法与 C 题几乎相同，直接套用 C 题套路进行 kruskal 找最小生成树算出花费

代码：

```
#include <algorithm>
#include <iostream>
using namespace std;
int u, v, w;
int root, total, n, m;
struct edge {
    int u, v, w;
    bool operator<(edge &e) { return w < e.w; }
};
edge e[200001];
int p[100001];
int find(int x) {
    if (p[x] == x) return x;
    p[x] = find(p[x]);
    return p[x];
```

```
}
bool combine(int x, int y) {
    int a = find(x), b = find(y);
    if (a == b) return false;
    p[a] = b;
    return true;
}
void kruskal() {
    sort(e + 1, e + 1 + m);
    for (int i = 1; i <= n; i++) p[i] = i;
    int ans = 0, num = 0;
    for (int i = 1; i <= m; i++) {
        if (combine(e[i].u, e[i].v)) {
            num++;
            ans = e[i].w;
        }
        if (num == n - 1) {
            break;
        }
    }
    cout << ans << endl;
}

int main() {
    while (cin >> n >> m >> root) {
        for (int i = 1; i <= m; i++) {
            cin >> u >> v >> w;
            total++;
            e[total].u = u;
            e[total].v = v;
            e[total].w = w;
        }
        kruskal();
    }

    return 0;
}
```

Week6_限时模拟 A - 掌握魔法の东东 II Gym - 270437J

从瑞神家打牌回来后，东东痛定思痛，决定苦练牌技，终成赌神!
东东有 $A \times B$ 张扑克牌。每张扑克牌有一个大小(整数，记为 a，范围区间是 0 到 $A - 1$）和一个花色（整数，记为 b，范围区间是 0 到 $B - 1$。

扑克牌是互异的，也就是独一无二的，也就是说没有两张牌大小和花色都相同。

“一手牌”的意思是你手里有 5 张不同的牌，这 5 张牌没有谁在前谁在后的顺序之分，它们可以形成一个牌型。我们定义了 9 种牌型，如下是 9 种牌型的规则，我们用“低序号优先”来匹配牌型，即这“一手牌”从上到下满足的第一个牌型规则就是它的“牌型编号”（一个整数，属于 1 到 9）：

1. 同花顺：同时满足规则 2 和规则 3.
2. 顺子 ：5 张牌的大小形如 $x, x+1, x+2, x+3, x+4$
3. 同花 ：5 张牌都是相同花色的.
4. 炸弹 ：5 张牌其中有 4 张牌的大小相等.
5. 三带二 ：5 张牌其中有 3 张牌的大小相等，且另外 2 张牌的大小也相等.
6. 两对：5 张牌其中有 2 张牌的大小相等，且另外 3 张牌中 2 张牌的大小相等.
7. 三条：5 张牌其中有 3 张牌的大小相等.
8. 一对：5 张牌其中有 2 张牌的大小相等.
9. 要不起：这手牌不满足上述的牌型中任意一个.

现在，东东从 $A \times B$ 张扑克牌中拿走了 2 张牌！分别是 (a_1, b_1) 和 (a_2, b_2). （其中 a 表示大小，b 表示花色）

现在要从剩下的扑克牌中再随机拿出 3 张！组成一手牌!!

其实东东除了会打代码，他业余还是一个魔法师，现在他要预言他的未来的可能性，即他将拿到的“一手牌”的可能性，我们用一个“牌型编号（一个整数，属于 1 到 9）”来表示这手牌的牌型，那么他的未来有 9 种可能，但每种可能的方案数不一样。

现在，东东的阿戈摩托之眼没了，你需要帮他算一算 9 种牌型中，每种牌型的方案数。

Input

第 1 行包含了整数 A 和 B ($5 \leq A \leq 25, 1 \leq B \leq 4$).

第 2 行包含了整数 a_1, b_1, a_2, b_2 ($0 \leq a_1, a_2 \leq A - 1, 0 \leq b_1, b_2 \leq B - 1, (a_1, b_1) \neq (a_2, b_2)$).

Output

输出一行，这行有 9 个整数，每个整数代表了 9 种牌型的方案数（按牌型编号从小到大的顺序）

Examples

Input

5 2

1 0 3 1

Output

0 8 0 0 0 12 0 36 0

Input

25 4

0 0 24 3

Output

0 0 0 2 18 1656 644 36432 113344

思路：

题意就是已知两张牌，然后从所有牌里面再抽出三张与这两张不同的牌，然后判断牌型，最后统计每种牌型的数量;

定义 poke 类，包含花色和大小，并重载“==”和“<”，方便排序。

开始，直接比较每张牌花色和大小（很暴力），判断牌型，但是最后几个牌型不能判断，而且由于我用全部牌中枚举跳过已有牌的方法出了点问题，连前几个牌型都没有判断对。

于是有了辅助方法，将抽到的五张牌排序，从第二个牌面开始，前减后，统计相减为 0 的个数（用来判断之前的方法不能区分的牌型），同时在读入已知两张牌后，将它们从牌堆删除，然后在剩余牌里面枚举。

判断牌型部分，开始几种简单的牌型，直接逐个比较花色和牌面大小判断前五种牌型（同花顺，同花，顺子，炸弹，三代二）。然后，先判断第七种（三条）：在统计 0 的个数时记录了第一个产生 0 的位置，从这里开始向后遍历看是不是有三张同样大小的排，最后根据统计 0 的个数判断第五，七，八种牌型（前几种牌型用到这个会产生错误判断）。

代码：

```
#include "algorithm"
#include "iostream"
using namespace std;
struct poke {
    int size;
    int color;
    bool operator<(const poke& S) const {
        return size < S.size || size == S.size && color < S.color;
    }
    bool operator==(const poke& S) const {
        return size == S.size && color == S.color;
    }
};
int A, B, x[2], y[2];
poke a[5];
int main() {
    while (cin >> A >> B) {
        int t2 = 0;
        int type[9] = {0};
        poke set[A * B];

        for (int i = 0; i < 2; i++) {
            cin >>x[i]>> y[i];
        }
        //Í¨¹ýÍ§¹ýÍ¨¹ý
        int cnt = 0;
        for (int i = 0; i < A; i++) {
```

```
    for (int j = 0; j < B; j++) {
        if (i == x[0] && j == y[0] || i == x[1] && j == y[1]) {
            set[cnt].size = 999;
            set[cnt].color = 999;
            continue;
        }
        set[cnt].size = i;
        set[cnt].color = j;
        cnt++;
    }
}
sort(set, set + cnt);
/*for (int i = 0; i < cnt; i++) {
    cout << set[i].size << " " << set[i].color << endl;
}*/

int c = 0;
for (int i = 0; i < (A * B - 4); i++) {
    for (int j = i + 1; j < (A * B - 3); j++) {
        for (int k = j + 1; k < A * B - 2; k++) {
            a[0].size = x[0];
            a[0].color = y[0];
            a[1].size = x[1];
            a[1].color = y[1];
            a[2].size = set[i].size;
            a[2].color = set[i].color;
            a[3].size = set[j].size;
            a[3].color = set[j].color;
            a[4].size = set[k].size;
            a[4].color = set[k].color;
            sort(a, a + 5);
            /*for (int tt = 0; tt < 5; tt++) {
                cout << a[tt].size << " " << a[tt].color << endl;
            }
            cout << "=====" << endl;*/
            int c1 = 0;
            int temp;
            int t[4];
            for (int o = 0; o < 4; o++) {
                int temp = a[o + 1].size - a[o].size;
                if (temp == 0) {
                    t[c1] = o;
                    c1++;
                }
```

```
}

if ((a[0].size + 1 == a[1].size) &&
    (a[1].size + 1 == a[2].size) &&
    (a[2].size + 1 == a[3].size) &&
    (a[3].size + 1 == a[4].size) &&
    (a[0].color == a[1].color) &&
    (a[1].color == a[2].color) &&
    (a[2].color == a[3].color) &&
    (a[3].color == a[4].color)) {
    type[0]++;
} else if ((a[0].size + 1 == a[1].size) &&
           (a[1].size + 1 == a[2].size) &&
           (a[2].size + 1 == a[3].size) &&
           (a[3].size + 1 == a[4].size)) {
    type[1]++;
} else if ((a[0].color == a[1].color) &&
           (a[1].color == a[2].color) &&
           (a[2].color == a[3].color) &&
           (a[3].color == a[4].color)) {
    type[2]++;
} else if (((a[0].size == a[1].size) &&
            (a[1].size == a[2].size) &&
            (a[2].size == a[3].size)) ||
           ((a[1].size == a[2].size) &&
            (a[2].size == a[3].size) &&
            (a[3].size == a[4].size))) {
    type[3]++;
} else if (((a[0].size == a[1].size) &&
            (a[1].size == a[2].size) &&
            (a[3].size == a[4].size)) ||
           ((a[0].size == a[1].size) &&
            (a[2].size == a[3].size) &&
            (a[3].size == a[4].size))) {
    type[4]++;
} else if (c1 == 2 && a[t[0]].size == a[t[0] + 1].size &&
           a[t[0] + 1].size == a[t[0] + 2].size) {
    type[6]++;
} else if (c1 == 2) {
    type[5]++;
} else if (c1 == 1) {
    type[7]++;
} else {
    type[8]++;
```

```
                    }
                    c++;
                }
            }
        }
        for (int i = 0; i < 9; i++) {
            cout << type[i] << " ";
        }
        //cout << c << endl;
    }
}
```