枚举、模拟与排序

一、AcWing 1210. 连号区间数

【题目描述】

小明这些天一直在思考这样一个奇怪而有趣的问题：

在$1 \sim N$的某个排列中有多少个连号区间呢？

这里所说的连号区间的定义是：

如果区间$[L, R]$里的所有元素（即此排列的第L个到第R个元素）递增排序后能得到一个长度为$R - L + 1$的“连续”数列，则称这个区间连号区间。

当N很小的时候，小明可以很快地算出答案，但是当N变大的时候，问题就不是那么简单了，现在小明需要你的帮助。

【输入格式】

第一行是一个正整数N，表示排列的规模。

第二行是N个不同的数字P_i，表示这N个数字的某一排列。

【输出格式】

输出一个整数，表示不同连号区间的数目。

【数据范围】

$1 \leq N \leq 10000$

$1 \leq P_i \leq N$

【输入样例1】

```
4
3 2 4 1
```

【输出样例1】

```
7
```

【输入样例2】

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

【输出样例2】

```
9
```

【样例解释】

第一个用例中，有7个连号区间分别是：$[1,1],[1,2],[1,3],[1,4],[2,2],[3,3],[4,4]$

第二个用例中，有9个连号区间分别是：$[1,1],[1,2],[1,3],[1,4],[1,5],[2,2],[3,3],[4,4],[5,5]$

【分析】

由于给定的序列是$1\sim N$的某种排列，即$1\sim N$中的每个数有且仅出现一次。因此如果某个区间$[L,R]$是连续区间，那么区间中的最大值$maxv$减去区间中的最小值$minv$一定等于$R-L$。

根据该性质，我们可以枚举区间的左端点i，对于每个i我们再枚举区间的右端点j，在从小到大枚举j的时候我们可以看成每次区间中新加入一个数$a[j]$，因此可以动态的维护区间$[i,j]$中的最大值与最小值。该做法的时间复杂度为$O(n^2)$，且常数很小，可以通过本题。

【代码】

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

const int N = 10010;
int a[N];
int n, res;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++)
    {
        int minv = 1e9, maxv = -1e9;

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

```
        {
            minv = min(minv, a[j]);
            maxv = max(maxv, a[j]);
            if (maxv - minv == j - i) res++;
        }
    }
    cout << res << endl;
    return 0;
}
```

二、AcWing 1236. 递增三元组

【题目描述】

给定三个整数数组：

$A = [A_1, A_2, \ldots, A_N]$

$B = [B_1, B_2, \ldots, B_N]$

$C = [C_1, C_2, \ldots, C_N]$

请你统计有多少个三元组(i, j, k)满足：

- $1 \le i, j, k \le N$
- $A_i < B_j < C_k$

【输入格式】

第一行包含一个整数N。

第二行包含N个整数$A_1, A_2, \ldots, A_N$。

第三行包含N个整数$B_1, B_2, \ldots, B_N$。

第四行包含N个整数$C_1, C_2, \ldots, C_N$。

【输出格式】

一个整数表示答案。

【数据范围】

$1 \le N \le 10^5$

$0 \le A_i, B_i, C_i \le 10^5$

【输入样例】

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

【输出样例】

```
27
```

【分析】

最暴力的做法就是三重循环枚举i,j,k，时间复杂度为$O(n^3)$，只能过部分样例。

本题的数据量允许我们只能枚举一个序列，如果枚举A，则B,C中选出的数的相对大小关系不好确定，枚举C同理。而如果我们枚举B，那么我们只需要求出A中小于B_i的数的数量$cnt1$，以及C中大于B_i的数的数量$cnt2$，那么将$cnt1*cnt2$累加到结果中即可。

求出$cnt1,cnt2$的方式有很多，可以排序+二分或者排序+双指针。本题以双指针为例，首先将三个数组排好序，设置指针$idx1=idx2=0$分别指向A和C（三个数组下标均从0开始），然后我们从小到大枚举B_i。对于每个B_i，我们先更新$idx1,idx2$的位置，$idx1$指向的是数组A中第一个大于等于B_i的位置，$idx2$指向的是数组C中第一个大于B_i的位置，由于B_i是单调递增的，因此两个指针也一定是单调递增的。则$idx1$即为$cnt1$，$n-idx2$即为$cnt2$，因此答案累加上$idx1*(n-idx2)$即可。

【代码】

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

typedef long long LL;
const int N = 100010;
int a[N], b[N], c[N];
int n;

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++) cin >> b[i];
    for (int i = 0; i < n; i++) cin >> c[i];
```

```
    sort(a, a + n); sort(b, b + n); sort(c, c + n);
    int idx1 = 0, idx2 = 0;
    LL res = 0;
    for (int i = 0; i < n; i++)
    {
        while (idx1 < n && a[idx1] < b[i]) idx1++;
        while (idx2 < n && c[idx2] <= b[i]) idx2++;
        res += (LL)idx1 * (n - idx2);
    }
    cout << res << endl;
    return 0;
}
```

三、AcWing 1245. 特别数的和

【题目描述】

小明对数位中含有$2, 0, 1, 9$的数字很感兴趣（不包括前导0），在$1 \sim 40$中这样的数包括$1, 2, 9, 10, \ldots, 32, 39, 40$，共$28$个，他们的和是$574$。

请问，在$1 \sim n$中，所有这样的数的和是多少？

【输入格式】

共一行，包含一个整数n。

【输出格式】

共一行，包含一个整数，表示满足条件的数的和。

【数据范围】

$1 \leq n \leq 10000$

【输入样例】

```
40
```

【输出样例】

```
574
```

【分析】

直接暴力即可~

【代码】

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

int n, res;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x = i;
        while (x)
        {
            int t = x % 10;
            x /= 10;
            if (t == 2 || t == 0 || t == 1 || t == 9) { res += i; break; }
        }
    }
    cout << res << endl;
    return 0;
}
```

四、AcWing 1204. 错误票据

【题目描述】

某涉密单位下发了某种票据，并要在年终全部收回。

每张票据有唯一的ID号。

全年所有票据的ID号是连续的，但ID的开始数码是随机选定的。

因为工作人员疏忽，在录入ID号的时候发生了一处错误，造成了某个ID断号，另外一个ID重号。

你的任务是通过编程，找出断号的ID和重号的ID。

假设断号不可能发生在最大和最小号。

【输入格式】

第一行包含整数N，表示后面共有N行数据。

接下来N行，每行包含空格分开的若干个（不大于100个）正整数（不大于100000），每个整数代表一个ID号。

【输出格式】

要求程序输出1行，含两个整数m,n，用空格分隔。

其中，m表示断号ID，n表示重号ID。

【数据范围】

$1 \leq N \leq 100$

【输入样例】

```
2
5 6 8 11 9
10 12 9
```

【输出样例】

```
7 9
```

【分析】

将读入的所有数据排好序，然后遍历一遍找出重号与断号即可。注意本题的读入由于每一行的长度不定，因此可以使用`stringstream`读入或者直接忽略行数直接读到空为止。以下是`stringstream`的用法：

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

const int N = 100010;
int a[N];
int n, cnt;

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

```
    string line;
    getline(cin, line);
    while (n--)
    {
        getline(cin, line);
        stringstream ssin(line);
        while (ssin >> a[cnt]) cnt++;
    }
}
```

【代码】

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

const int N = 10010;
int a[N];
int n, cnt;

int main()
{
    cin >> n;
    while (cin >> a[cnt]) cnt++;
    sort(a, a + cnt);
    int res1, res2;
    for (int i = 0; i < cnt - 1; i++)
        if (a[i] == a[i + 1]) res2 = a[i];
        else if (a[i + 1] - a[i] != 1) res1 = a[i] + 1;
    cout << res1 << ' ' << res2 << endl;
    return 0;
}
```

五、AcWing 466. 回文日期

【题目描述】

在日常生活中，通过年、月、日这三个要素可以表示出一个唯一确定的日期。

牛牛习惯用8位数字表示一个日期，其中，前4位代表年份，接下来2位代表月份，最后2位代表日期。

显然：一个日期只有一种表示方法，而两个不同的日期的表示方法不会相同。

牛牛认为，一个日期是回文的，当且仅当表示这个日期的8位数字是回文的。

现在，牛牛想知道：在他指定的两个日期之间（包含这两个日期本身），有多少个真实存在的日期是回文的。

一个8位数字是回文的，当且仅当对于所有的$i(1 \leq i \leq 8)$从左向右数的第i个数字和第$9-i$个数字（即从右向左数的第i个数字）是相同的。

例如：

- 对于2016年11月19日，用8位数字20161119表示，它不是回文的。
- 对于2010年1月2日，用8位数字20100102表示，它是回文的。
- 对于2010年10月2日，用8位数字20101002表示，它不是回文的。

每一年中都有12个月份：

其中，$1,3,5,7,8,10,12$月每个月有31天；$4,6,9,11$月每个月有30天；而对于2月，闰年时有29天，平年时有28天。 一个年份是闰年当且仅当它满足下列两种情况其中的一种：

1．这个年份是4的整数倍，但不是100的整数倍；
2．这个年份是400的整数倍。

【输入格式】

输入包括两行，每行包括一个8位数字。

第一行表示牛牛指定的起始日期$date1$，第二行表示牛牛指定的终止日期$date2$。保证$date1$和$date2$都是真实存在的日期，且年份部分一定为4位数字，且首位数字不为0。

保证$date1$一定不晚于$date2$。

【输出格式】

输出共一行，包含一个整数，表示在$date1$和$date2$之间，有多少个日期是回文的。

【输入样例】

```
20110101
20111231
```

【输出样例】

```
1
```

【分析】

第一种思路是枚举每个日期，判断是不是回文，另一种思路是枚举回文，也就是当前四位或后四位确定时整个回文串就确定了，再判断这个回文串构成的日期合不合法。

【枚举日期代码】

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

int date1, date2;
int y, m, d, gy, gm, gd;
int res;

bool check()
{
    int dy[4] = { 0 }, dm[2] = { 0 }, dd[2] = { 0 };
    int ty = y, tm = m, td = d;
    for (int i = 3; i >= 0; i--) dy[i] = ty % 10, ty /= 10;
    for (int i = 1; i >= 0; i--) dm[i] = tm % 10, tm /= 10;
    for (int i = 1; i >= 0; i--) dd[i] = td % 10, td /= 10;
    if (dy[0] != dd[1] || dy[1] != dd[0] || dy[2] != dm[1] || dy[3] != dm[0]) return false;
    return true;
}

int main()
{
    cin >> date1 >> date2;
    y = date1 / 10000, m = date1 % 10000 / 100, d = date1 % 100;
    gy = date2 / 10000, gm = date2 % 10000 / 100, gd = date2 % 100;
    while (true)
    {
        if (y == gy && m == gm && d == gd)
        {
            if (check()) res++;
            break;
        }
        if (check()) res++;
        d++;
        if (d > 28)

        {
```

```
            if (m == 2 && d > 28)
            {
                if (((y % 4 == 0 && y % 100 != 0) || y % 400 == 0) && d > 29) m++, d %= 29;
                if (!((y % 4 == 0 && y % 100 != 0) || y % 400 == 0) && d > 28) m++, d %= 28;
            }
            else if ((m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12) && d > 31) m++, d %= 31;
            else if ((m == 4 || m == 6 || m == 9 || m == 11) && d > 30) m++, d %= 30;
            if (m > 12) y++, m %= 12;
        }
    }
    cout << res << endl;
    return 0;
}
```

【枚举回文串代码】

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

int date1, date2, res;
int days[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };//每个月的天数

bool check(int date)
{
    int year = date / 10000, month = date % 10000 / 100, day = date % 100;
    if (month < 1 || month > 12 || day < 1) return false;
    if (month != 2)
    {
        if (day > days[month]) return false;
        return true;
    }
    int leap = year % 100 && year % 4 == 0 || year % 400 == 0;
    if (day > 28 + leap) return false;//平年时leap为0,闰年时为1
    return true;

}
```

```
22
23 int main()
24 {
25     cin >> date1 >> date2;
26     for (int i = 1000; i < 10000; i++)//枚举年份
27     {
28         int date = i, x = i;//date表示将年份扩充成回文日期的结果
29         for (int j = 0; j < 4; j++) date = date * 10 + x % 10, x /= 10;
30         if (date >= date1 && date <= date2 && check(date)) res++;
31     }
32     cout << res << endl;
33 }
```

六、AcWing 787. 归并排序

【题目描述】

给定你一个长度为n的整数数列。

请你使用归并排序对这个数列按照从小到大进行排序。

并将排好序的数列按顺序输出。

【输入格式】

输入共两行，第一行包含整数n。

第二行包含n个整数（所有整数均在$1 \sim 10^9$范围内），表示整个数列。

【输出格式】

输出共一行，包含n个整数，表示排好序的数列。

【数据范围】

$1 \leq n \leq 100000$

【输入样例】

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

【输出样例】

```
1 2 3 4 5
```

【代码】

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

const int N = 100010;
int a[N], tmp[N];
int n;

void merge_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(a, l, mid);
    merge_sort(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (a[i] <= a[j]) tmp[k++] = a[i++];
        else tmp[k++] = a[j++];
    while (i <= mid) tmp[k++] = a[i++];
    while (j <= r) tmp[k++] = a[j++];
    for (int i = l, j = 0; i <= r; i++, j++) a[i] = tmp[j];
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) cin >> a[i];
    merge_sort(a, 0, n - 1);
    for (int i = 0; i < n; i++) cout << a[i] << ' ';
    return 0;
}
```

七、AcWing 1219. 移动距离

【题目描述】

X星球居民小区的楼房全是一样的，并且按矩阵样式排列。

其楼房的编号为$1, 2, 3 \ldots$

当排满一行时，从下一行相邻的楼往反方向排号。

比如：当小区排号宽度为6时，开始情形如下：

```
1  2  3  4  5  6
12 11 10 9  8  7
13 14 15 ...
```

我们的问题是：已知了两个楼号m和n，需要求出它们之间的最短移动距离（不能斜线方向移动）。

【输入格式】

输入共一行，包含三个整数w,m,n，w为排号宽度，m,n为待计算的楼号。

【输出格式】

输出一个整数，表示m,n两楼间最短移动距离。

【数据范围】

$1 \leq w,m,n \leq 10000$

【输入样例】

```
6 8 2
```

【输出样例】

```
4
```

【分析】

首先我们先将所有的标号减一，变为如下的形式：

```
0  1  2  3  4  5
11 10 9  8  7  6
12 13 14 15 ...
```

可以观察到标号为n时其二维坐标的行号为n/w，如果不存在反向排号，则其列号为$n\%w$。如果n所在的行为奇数行，那么其列号应该反向排列，即列号为$w-1-n\%w$。那么我们就可以在$O(1)$的时间求出任何一个标号所在的行号与列号。

【代码】

```
#include <iostream>

#include <cstring>
```

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

int main()
{
    int w, n, m;
    cin >> w >> n >> m;
    n--, m--;
    int x1 = n / w, x2 = m / w, y1 = n % w, y2 = m % w;
    if (x1 % 2) y1 = w - 1 - y1;
    if (x2 % 2) y2 = w - 1 - y2;
    cout << abs(x1 - x2) + abs(y1 - y2) << endl;
    return 0;
}
```

八、AcWing 1229. 日期问题

【题目描述】

小明正在整理一批历史文献。这些历史文献中出现了很多日期。

小明知道这些日期都在1960年1月1日至2059年12月31日。

令小明头疼的是，这些日期采用的格式非常不统一，有采用年/月/日的，有采用月/日/年的，还有采用日/月/年的。

更加麻烦的是，年份也都省略了前两位，使得文献上的一个日期，存在很多可能的日期与其对应。

比如02/03/04，可能是2002年03月04日、2004年02月03日或2004年03月02日。

给出一个文献上的日期，你能帮助小明判断有哪些可能的日期对其对应吗？

【输入格式】

一个日期，格式是`AA/BB/CC`。

即每个`/`隔开的部分由两个$0\sim9$之间的数字（不一定相同）组成。

【输出格式】

输出若干个不相同的日期，每个日期一行，格式是`YYYY-MM-DD`。

多个日期按从早到晚排列。

【数据范围】

$0\le A,B,C\le 9$

【输入样例】

```
02/03/04
```

【输出样例】

```
2002-03-04
2004-02-03
2004-03-02
```

【分析】

枚举1960年1月1日至2059年12月31日的所有日期，对于每个日期，判断其是否合法且是否有一种表示形式与所给形式相同，如果两者都满足那么直接输出当前日期。由于是从小到大枚举日期，因此输出的结果一定是从早到晚排列的。

【代码】

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

int a, b, c;
int days[13] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

bool check(int year, int month, int day)
{
    year %= 100;
    if (month < 1 || month > 12 || day < 1) return false;
    if (month != 2)
    {
        if (day > days[month]) return false;
    }
    else
    {
        int leap = year % 100 && year % 4 == 0 || year % 400 == 0;
        if (day > 28 + leap) return false;
    }
    if (year == a && month == b && day == c) return true;//年/月/日
    if (month == a && day == b && year == c) return true;//月/日/年
    if (day == a && month == b && year == c) return true;//日/月/年
```

```
    return false;
}

int main()
{
    scanf("%d/%d/%d", &a, &b, &c);
    for (int i = 19600101; i <= 20591231; i++)
    {
        int year = i / 10000, month = i % 10000 / 100, day = i % 100;
        if (check(year, month, day)) printf("%d-%02d-%02d\n", year,
month, day);
    }
    return 0;
}
```

九、AcWing 1231. 航班时间

【题目描述】

小h前往美国参加了蓝桥杯国际赛。

小h的女朋友发现小h上午十点出发，上午十二点到达美国，于是感叹到“现在飞机飞得真快，两小时就能到美国了”。

小h对超音速飞行感到十分恐惧。

仔细观察后发现飞机的起降时间都是当地时间。

由于北京和美国东部有12小时时差，故飞机总共需要14小时的飞行时间。

不久后小h的女朋友去中东交换。

小h并不知道中东与北京的时差。

但是小h得到了女朋友来回航班的起降时间。

小h想知道女朋友的航班飞行时间是多少。

对于一个可能跨时区的航班，给定来回程的起降时间。

假设飞机来回飞行时间相同，求飞机的飞行时间。

【输入格式】

一个输入包含多组数据。

输入第一行为一个正整数T，表示输入数据组数。

每组数据包含两行，第一行为去程的起降时间，第二行为回程的起降时间。

起降时间的格式如下：

- `h1:m1:s1 h2:m2:s2`
- `h1:m1:s1 h3:m3:s3 (+1)`
- `h1:m1:s1 h4:m4:s4 (+2)`

第一种格式表示该航班在当地时间$h1$时$m1$分$s1$秒起飞，在当地时间当日$h2$时$m2$分$s2$秒降落。

第二种格式表示该航班在当地时间$h1$时$m1$分$s1$秒起飞，在当地时间次日$h2$时$m2$分$s2$秒降落。

第三种格式表示该航班在当地时间$h1$时$m1$分$s1$秒起飞，在当地时间第三日$h2$时$m2$分$s2$秒降落。

【输出格式】

对于每一组数据输出一行一个时间`hh:mm:ss`，表示飞行时间为hh小时mm分ss秒。

注意，当时间为一位数时，要补齐前导零，如三小时四分五秒应写为`03:04:05`。

【数据范围】

保证输入时间合法（$0 \le h \le 23, 0 \le m, s \le 59$），飞行时间不超过$24$小时。

【输入样例】

```
3
17:48:19 21:57:24
11:05:18 15:14:23
17:21:07 00:31:46 (+1)
23:02:41 16:13:20 (+1)
10:19:19 20:41:24
22:19:04 16:41:09 (+1)
```

【输出样例】

```
04:09:05
12:10:39
14:22:05
```

【分析】

假设飞机飞行时间为$time$，从a飞到b的时差为$+d$，即总共花费的时间为$time+d$，从b飞回a的时差为$-d$，即总共花费的时间为$time-d$。那么往返两次的花费时间相加为$time+d+time-d=2*time$，因此时差在此处并没有影响，两次的往返时间之差的和除以2即为飞行时间。

本题的难点在于字符串的处理，由于输入的每一行字符串格式可能不一样，因此如果末尾没有`(+1)`之类的标注我们默认将其改为`(+0)`，这样就可以将字符串统一格式。将每个串中的数字抽取出来可以使用`sscanf`，具体使用方式见代码。

【代码】

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

int n;
string line;

int get_second(int h, int m, int s)
{
    return h * 3600 + m * 60 + s;
}

int get_time()
{
    getline(cin, line);
    if (line.back() != ')') line += " (+0)";
    int h1, m1, s1, h2, m2, s2, day;
    sscanf(line.c_str(), "%d:%d:%d %d:%d:%d (+%d)", &h1, &m1, &s1, &h2, &m2, &s2, &day);
    return get_second(h2, m2, s2) - get_second(h1, m1, s1) + day * 86400;
}

int main()
{
    scanf("%d", &n);
    getline(cin, line);
    while (n--)
    {

        int time = (get_time() + get_time()) / 2;
```

```
        int h = time / 3600, m = time % 3600 / 60, s = time % 60;
        printf("%02d:%02d:%02d\n", h, m, s);
    }
    return 0;
}
```

十、AcWing 1241. 外卖店优先级

【题目描述】

“饱了么”外卖系统中维护着N家外卖店，编号$1 \sim N$。

每家外卖店都有一个优先级，初始时（0时刻）优先级都为0。

每经过1个时间单位，如果外卖店没有订单，则优先级会减少1，最低减到0；而如果外卖店有订单，则优先级不减反加，每有一单优先级加2。

如果某家外卖店某时刻优先级大于5，则会被系统加入优先缓存中；如果优先级小于等于3，则会被清除出优先缓存。

给定T时刻以内的M条订单信息，请你计算T时刻时有多少外卖店在优先缓存中。

【输入格式】

第一行包含3个整数N, M, T。

以下M行每行包含两个整数t和id，表示t时刻编号id的外卖店收到一个订单。

【输出格式】

输出一个整数代表答案。

【数据范围】

$1 \le N, M, T \le 10^5$

$1 \le t \le T$

$1 \le id \le N$

【输入样例】

```
2 6 6
1 1
5 2
3 1
6 2
2 1
6 2
```

【输出样例】

```
1
```

【样例解释】

6时刻时，1号店优先级降到3，被移除出优先缓存；2号店优先级升到6，加入优先缓存。

所以是有1家店（2号）在优先缓存中。

【分析】

---

1. 首先对输入的m个订单信息排序（时间t为第一优先级，订单id为第二优先级）。
2. 遍历订单信息（记得此时订单大体是按照时间顺序排的）。
3. 假设当前订单为第i个，循环判断后面有没有相同的订单，即t和id都相等（有的话则这些订单一定连续）。
4. 当到第j个时订单不相同，此时相同订单的数量为$cnt = j - i$，下一次循环从j处开始遍历。
5. 记录此时的时刻t和店铺id，计算id的优先权，有两部分：

（1）上一个拿到订单的时间$last[id]$和t之间，中间没订单所以要-1，没订单的数量是$t - last[i] - 1$（比如第3和第6时刻都有订单，没有订单的时候就是$4, 5$），然后计算优先权，如果为负值更新为0。如果小于等于3，则更新优先缓存$st[id] = false$；

（2）此时，t时刻拿到订单，并且拿到的数量为cnt，优先级要加上$2 * cnt$，然后计算优先权，如果大于5，则更新优先缓存$st[id] = true$。

6. 解释上面那个，因为此时这几个相同的订单都计算过了不需要再计算了，所以下一次循环要从j开始。
7. 循环最后，店铺id上一次拿到订单的时间$last[id]$更新为t。
8. 如果最后一个订单时刻为T，则没问题。如果不是T，那么最后一个拿到订单时刻到T时刻的这部分减法需要手动计算，即优先级需要减去T时刻与该店最后一个订单时刻$last[id]$的差值。换而言之，如果上一个拿到订单的时间$last[id]$小于T，则优先权减去$T - last[id]$。注意这里不减1，因为T时刻也没订单。如果小于等于3，则更新优先缓存$st[id] = false$。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
const int N = 100010;
int score[N], last[N];//score[i]表示第i个店铺当前的优先级,last[i]表示第i个店铺上一次有订单的时刻
PII order[N];//表示所有订单
bool st[N];//st[i]表示第i个店铺是否处于优先缓存中
int n, m, T;

int main()
{
    scanf("%d%d%d", &n, &m, &T);
    for (int i = 0; i < m; i++) scanf("%d%d", &order[i].X, &order[i].Y);
    sort(order, order + m);//将订单按照时间从小到大排序
    for (int i = 0; i < m;)
    {
        int j = i;
        while (j < m && order[i] == order[j]) j++;//找出一批相同的订单一起处理
        int t = order[i].X, id = order[i].Y, cnt = j - i;
        i = j;
        score[id] -= t - last[id] - 1;//t时刻有订单因此需要减一
        if (score[id] < 0) score[id] = 0;
        if (score[id] <= 3) st[id] = false;
        score[id] += cnt * 2;//将t时刻的订单算上
        if (score[id] > 5) st[id] = true;
        last[id] = t;//更新last
    }
    for (int i = 1; i <= n; i++)
        if (last[i] < T)//在最后一段时间内没收到订单
        {
            score[i] -= T - last[i];//T时刻没有订单因此不需要减一
            if (score[i] <= 3) st[i] = false;

        }
```

```
    int res = 0;
    for (int i = 1; i <= n; i++) res += st[i];
    printf("%d\n", res);
    return 0;
}
```

十一、AcWing 788. 逆序对的数量

【题目描述】

给定一个长度为n的整数数列，请你计算数列中的逆序对的数量。

逆序对的定义如下：对于数列的第i个和第j个元素，如果满足$i < j$且$a[i] > a[j]$，则其为一个逆序对；否则不是。

【输入格式】

第一行包含整数n，表示数列的长度。

第二行包含n个整数，表示整个数列。

【输出格式】

输出一个整数，表示逆序对的个数。

【数据范围】

$1 \leq n \leq 100000$

数列中的元素的取值范围$[1, 10^9]$。

【输入样例】

```
6
2 3 4 5 6 1
```

【输出样例】

```
5
```

【分析】

在归并排序的合并过程中，两个数组内的元素都为有序状态，因此若左半部分数组中的第一个元素$a[i]$大于右半部分数组的第一个元素$a[j]$，说明$a[i]$以及其后面的所有元素都是$a[j]$的逆序对（因为左半部分数组内部是有序的，$a[i]$之后的元素一定大于等于$a[i]$），故可以利用此性质在归并排序的过程中求解逆序对的数量。

【代码】

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

typedef long long LL;
const int N = 100010;
int n, a[N], tmp[N];
LL res;

void merge_sort(int a[], int l, int r)
{
    if (l >= r) return;
    int mid = l + r >> 1;
    merge_sort(a, l, mid);
    merge_sort(a, mid + 1, r);
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (a[i] <= a[j]) tmp[k++] = a[i++];
        else
        {
            res += mid - i + 1;
            tmp[k++] = a[j++];
        }
    while (i <= mid) tmp[k++] = a[i++];
    while (j <= r) tmp[k++] = a[j++];
    for (int i = l, j = 0; i <= r; i++, j++) a[i] = tmp[j];
}

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &a[i]);
    merge_sort(a, 0, n - 1);
    printf("%lld", res);
    return 0;
}
```