删数游戏

题目描述

键盘输入一个高精度的正整数 N（不超过 250 位），去掉其中任意 k 个数字后剩下的数字按原左右次序将组成一个新的非负整数。编程对给定的 N 和 k，寻找一种方案使得剩下的数字组成的新数最小。

输入格式

输入两行正整数。

第一行输入一个高精度的正整数 n。

第二行输入一个正整数 k，表示需要删除的数字个数。

输出格式

输出一个整数，最后剩下的最小数。

样例输入

```
175438
4
```

样例输出

```
13
```

Solution

对于序列$a_1, a_2, \cdots, a_n$而言，假设我们删除的位置位于i，则序列会变为
$a_1, a_2, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n$

和原序列相比，前i位分别为：

$$a_1, a_2, \cdots, a_{i-1}, a_i$$
$$a_1, a_2, \cdots, a_{i-1}, a_{i+1}$$

可以看出，前i位只有a_i变成了a_{i+1}，如果有$a_i > a_{i+1}$，整个序列的**高位**就会变小，如果$a_i < a_{i+1}$，则整个序列的**高位**就会变大

因为存在一种删除方法，即删除a_n，这样就可以保证每次删除后序列的**高位**都**不变**，因此，我们只会删除$a_i > a_{i+1}$的情况

如果存在多个点$i, j, k, \cdots$，都有$a_i > a_{i+1}, a_j > a_{j+1}, a_k > a_{k+1}, \cdots$，那么我们该选择哪一个进行删除呢（假设$i < j < k < \cdots$）

我们以三个点为例，写下删除这三个点后的前k位：

$$a_1, a_2, \cdots, a_{i-1}, a_{i+1}, \cdots, a_j, a_{j+1}, \cdots, a_k, a_{k+1}$$
$$a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_{j-1}, a_{j+1}, \cdots, a_k, a_{k+1}$$
$$a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_{j-1}, a_j, \cdots, a_{k-1}, a_{k+1}$$

由于在数的**长度相等**的情况下，高位越小，数越小，又有$a_i > a_{i+1}$，因此删除第i个数最优

接下来我们将贪心的结论推广到多次删除

先以$k=2$为例，我们假设存在位置$x,y(x<y)$，使得删除a_x,a_y后比删除a_i,a_j后的数**更小**

先假设$x<i$，我们写下删除后的前i位

$$arr_x=a_1,a_2,\cdots,a_{x-1},a_{x+1},\cdots a_i,a_{i+1}$$
$$arr_i=a_1,a_2,\cdots,a_{x-1},a_x,\cdots a_{i-1},a_{i+1}$$

由于i是第一位$a_i>a_{i+1}$，因此有$a_x\le a_{x+1},a_{x+1}\le a_{x+2},\cdots$，也就是说，$arr_i$的前$i-2$位不会大于$arr_x$，而对于第$i-1$位而言，$a_i<a_{i-1}$，因此我们得到$arr_x>arr_i$，故$i\le x$，而$i<x$的情景我们也证明了删除第$i$位会更小，因此得到$x==i$

由于$x==i$，此时问题从$k=2$转换成了$k=1$，以此类推即可

Code

```
void solve(){
    string str; cin >> str;
    int k; cin >> k;

    while (k--) {
        int fd = str.size() - 1;
        for (int i = 1; i < str.size(); i++) {
            if (str[i] < str[i - 1]) { fd = i - 1; break;}
        }
        str.erase(fd, 1);
    }

    while (str.size() && str[0] == '0') str.erase(str.begin());
    cout << (str.size() ? str : "0") << endl;
}
```

混合牛奶

题目描述

由于乳制品产业利润很低，所以降低原材料（牛奶）价格就变得十分重要。帮助 Marry 乳业找到最优的牛奶采购方案。

Marry 乳业从一些奶农手中采购牛奶，并且每一位奶农为乳制品加工企业提供的价格可能相同。此外，就像每头奶牛每天只能挤出固定数量的奶，每位奶农每天能提供的牛奶数量是一定的。每天 Marry 乳业可以从奶农手中采购到小于或者等于奶农最大产量的整数数量的牛奶。

给出 Marry 乳业每天对牛奶的需求量，还有每位奶农提供的牛奶单价和产量。计算采购足够数量的牛奶所需的最小花费。

注：每天所有奶农的总产量大于 Marry 乳业的需求量。

输入格式

第一行二个整数 n,m，表示需要牛奶的总量，和提供牛奶的农民个数。

接下来 m 行，每行两个整数 p_i,a_i，表示第 i 个农民牛奶的单价，和农民 i 一天最多能卖出的牛奶量。

其中，$0\le n,a_i\le 2\times 10^6$，$0\le m\le 5000$，$0\le p_i\le 1000$

输出格式

单独的一行包含单独的一个整数，表示 Marry 的牛奶制造公司拿到所需的牛奶所要的最小费用。

样例输入

```
100 5
5 20
9 40
3 10
8 80
6 30
```

样例输出

```
630
```

Solution

显然，我们只需要根据牛奶**单价**从小到大排序，贪心地选择单价最低的牛奶进行购买，直到满足需求即可

Code

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<pair<int, int>> a(m);
    for (auto& p : a) cin >> p.first >> p.second;

    sort(a.begin(), a.end());

    int ans = 0;
    for (auto p : a) {
        auto pri = p.first, t = p.second;
        int b = min(n, t);
        ans += b * pri;
        n -= b;
    }

    cout << ans << endl;
}
```

纪念品分组

题目描述

元旦快到了，校学生会让乐乐负责新年晚会的纪念品发放工作。为使得参加晚会的同学所获得 的纪念品价值相对均衡，他要把购来的纪念品根据价格进行分组，但每组最多只能包括两件纪念品， 并且每组纪念品的价格之和不能超过一个给定的整数。为了保证在尽量短的时间内发完所有纪念品，乐乐希望分组的数目最少。

你的任务是写一个程序，找出所有分组方案中分组数最少的一种，输出最少的分组数目。

输入格式

共 $n+2$ 行：

第一行包括一个整数 $w(80 \leq w \leq 200)$，为每组纪念品价格之和的上限。

第二行为一个整数 $n(1 \leq n \leq 3 \times 10^4)$，表示购来的纪念品的总件数。

第 $3 \sim n+2$ 行每行包含一个正整数 $P_i(5 \leq P_i \leq w)$ 表示所对应纪念品的价格。

输出格式

一个整数，即最少的分组数目。

样例输入

```
100
9
90
20
20
30
50
60
70
80
90
```

样例输出

```
6
```

Solution

假设我们一开始每个纪念品都是一组，那么我们要做的，相当于是每次**合并**两个组

首先将纪念品按照价格排序得到数组$p_1, p_2, \cdots p_n$，对于每个纪念品，我们讨论可以和它合并的纪念品下标区间$[L_i, R_i]$，很明显，随着i的增大，P_i也单调不减，那么在L_i不变的同时，R_i单调不增

因此，我们i越小的数，合并时需要优先考虑下标越大的数

我们使用一个排好序的双向队列模拟取数的过程，每次取出队列中最大的数，看他能否和队列中最小的数合并，如果可以，则合并，如果不行，则它单独一组

Code

```cpp
void solve() {
    int w, n; cin >> w >> n;
    deque<int> a(n);
    for (auto& v: a) cin >> v;

    sort(all(a));

    int ans = 0;
```

```
    while (!a.empty()) {
        int tot = a.back(); a.pop_back();
        if (a.size() && a.front() + tot <= w) a.pop_front();
        ans++;
    }

    cout << ans << endl;
}
```

分组

题目描述

小可可的学校信息组总共有 n 个队员，每个人都有一个实力值 a_i。现在，一年一度的编程大赛就要到了，小可可的学校获得了若干个参赛名额，教练决定把学校信息组的 n 个队员分成若干个小组去参加这场比赛。

但是每个队员都不会愿意与实力跟自己过于悬殊的队员组队，于是要求分成的每个小组的队员实力值连续，同时，一个队不需要两个实力相同的选手。举个例子：$[1,2,3,4,5]$ 是合法的分组方案，因为实力值连续；$[1,2,3,5]$ 不是合法的分组方案，因为实力值不连续；$[0,1,1,2]$ 同样不是合法的分组方案，因为出现了两个实力值为 1 的选手。

如果有小组内人数太少，就会因为时间不够而无法获得高分，于是小可可想让你给出一个合法的分组方案，满足所有人都恰好分到一个小组，使得人数最少的组人数最多，输出人数最少的组人数的最大值。

注意：实力值可能是负数，分组的数量没有限制。

输入格式

输入有两行：

第一行一个正整数 $n(1 \leq n \leq 10^5)$，表示队员数量

第二行有 n 个整数，第 i 个整数 $a_i(|a_i| \leq 10^9)$ 表示第 i 个队员的实力。

输出格式

输出一行，包括一个正整数，表示人数最少的组的人数最大值。

样例输入

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

样例输出

```
3
```

Solution

先将队员的实力从小到大排序，并且统计每种实力的队员分别有几个

随后我们从小到大遍历所有**实力**，同时维护一个**可重**集合$begin$，集合内元素表示为能够满足实力能够在区间$[begin_i, last]$（$last$是上一个实力）内连续

现在考虑新下一个实力k（假设有m个人有实力k）

如果$k \neq last + 1$，此时没有任何一个 $begin_i$能满足集合连续，因此集合清零，并更新答案，同时将m个k塞进集合内

如果$k = last + 1$，此时我们至多能满足至多m个集合内元素，如果m比集合的大小$size$大，我们就可以在满足该集合所有元素的同时，还往集合内塞入$m - size$个k，否则我们就要从集合内删除$size - m$个元素并更新答案

被删除元素的选取，我们会优先选择**区间起点靠前**的元素，这样能够保证较小的区间能够优先满足，即能最大化最小区间

Code

```
void solve() {
    int n; cin >> n;
    map<int, int> mp;
    for (int i = 0; i < n; i++) {
        int u; cin >> u;
        mp[u]++;
    }

    priority_queue<int, vector<int>, greater<int>> pq;
    int lst = (*mp.begin()).first - 1, ans = n;

    for (auto [f, s]: mp) {
        if (f != lst + 1) {
            while (!pq.empty()) {
                auto v = pq.top(); pq.pop();
                cmin(ans, lst - v + 1);
            }
        }

        while (pq.size() < s) pq.push(f);
        while (pq.size() > s) {
            auto v = pq.top(); pq.pop();
            cmin(ans, f - v);
        }
        lst = f;
    }

    while (!pq.empty()) {
        auto v = pq.top(); pq.pop();
        cmin(ans, lst - v + 1);
    }

    cout << ans << endl;
}
```

旅行家的预算

题目描述

一个旅行家想驾驶汽车以最少的费用从一个城市到另一个城市（假设出发时油箱是空的）。给定两个城市之间的距离 D_1、汽车油箱的容量 C（以升为单位）、每升汽油能行驶的距离 D_2、出发点每升汽油价格P和沿途油站数 N（N 可以为零），油站 i 离出发点的距离 D_i、每升汽油价格 P_i（$i=1,2,\ldots,N$）。计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出 `No Solution`。

输入格式

第一行，D_1，C，D_2，P，N。

接下来有 N 行。

第 $i+1$ 行，三个数字，油站 i 的编号，离出发点的距离 D_i 和每升汽油价格 P_i。

其中，$N \leq 6$，其余数字 ≤ 500。

输出格式

所需最小费用，计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出 `No solution`。

样例输入

```
275.6 11.9 27.4 2.8 2
1 102.0 2.9
2 220.0 2.2
```

样例输出

```
26.95
```

Solution

我们先讨论无解的情况，显然，将加油站按照距出发点的距离排序后，若相邻两个加油站（包括起点和终点）的距离超过最大行驶里程$C \times D_2$，则无解

除此以外，最优解到达目的地时邮箱一定是**空的**，否则我们如果不加这些油，就能得到一个更小的花费

我们考虑从某个加油站i出发的情况，此时我们会面临一个抉择：

1. 加适当的油，等到某个价格更低的加油站再进行补充
2. 加满油

在情况1下，我们要求**最大行驶里程内**存在一个价格更低的加油站，如果不存在这样的加油站，我们就需要按照情况2，先加满油， 再在**最大行驶里程内**寻找汽油单价**最低**的加油站，并在那个加油站再次出发

按照如上策略，我们使用的每份油，单价都是局部最低的，这种情况能够拓展到全局，故贪心算法正确性得证

该题代码细节较为繁琐，同学们可以参考下述代码实现后再独立完成

Code

```
void solve(){
    int n;
    db d1, c, d2, p;
    cin >> d1 >> c >> d2 >> p >> n;
    // pos, price
    vector<pair<db, db>> a(n + 2);
    a[0].second = p; a[n + 1].first = d1;
    for (int i = 1; i <= n; i++) {
        int idx; cin >> idx;
        cin >> a[i].first >> a[i].second;
    }

    sort(a.begin(), a.end());
    db cost = 0, rest = 0, max_dis = c * d2;
    int pos = 0;

    // 存在两个加油站距离过远
    for (int i = 1; i <= n + 1; i++)
        if (a[i].first - a[i - 1].first > max_dis) {
            cout << "No solution" << endl; return;
        }

    while (1) {
        if (pos == n + 1) {cout << cost << endl; return;}

        int nxt = pos, type = 1;
        // 找到最大路程之内第一家价格小于当前位置的加油站
        for (int i = pos + 1; i <= n + 1; i++) {
            if (a[i].first - a[pos].first > max_dis) break;
            if (a[i].second < a[pos].second) {nxt = i; break;}
        }

        // 最大路程之内不存在价格小于当前位置的加油站，找到其中价格最低的加油站
        if (nxt == pos) {
            nxt = pos + 1; type = 2;
            for (int i = pos + 1; i <= n + 1; i++) {
                if (a[i].first - a[pos].first > max_dis) break;
                if (a[i].second < a[nxt].second) nxt = i;
            }
        }

        if (type == 1) {
            db buy = max(0.0, (a[nxt].first - a[pos].first) / d2 - rest);
            cost += buy * a[pos].second; rest = 0;
        }
        else {
            db buy = c - rest;
            cost += buy * a[pos].second;
            rest = c - (a[nxt].first - a[pos].first) / d2;
        }

        pos = nxt;
    }
```

```
    cout << cost << endl;
}
```