

А. Тренировка памяти (35 баллов)

Денис тренирует память. Он запоминает большие числа. Однажды, вспоминая 9-разрядное число 359 878 466, он ошибся и случайно поменял местами две его цифры – 5 и 4. В результате он получил число 349 878 566. Денис с удивлением обнаружил, что в результате такой перестановки у него получилось самое маленькое число, которое только можно было получить из исходного числа, поменяв местами две его цифры.

Требуется выяснить, какие две цифры нужно поменять местами в заданном N-разрядном числе, чтобы из него получилось как можно меньшее N-разрядное число (если это в принципе возможно).

Формат входных данных

В первой строке входного файла *input.txt* записано натуральное число N – количество цифр в числе ($2 \leq N \leq 10^3$). Во второй строке – последовательность из N цифр исходного числа, разделённых пробелом.

Формат выходных данных

В единственной строке выходного файла *output.txt* должны быть записаны в возрастающем порядке два натуральных числа – это позиции тех цифр исходного числа, которые нужно поменять местами, чтобы получить как можно меньшее N-разрядное число (позиции в N-разрядном числе нумеруются слева направо, начиная с 1). Если же в исходном числе нельзя переставить никакие две цифры так, чтобы оно стало меньше, оставаясь N-разрядным числом, то выдать 0.

Пример

input.txt	output.txt
9 3 5 9 8 7 8 4 6 6	2 7

Тренировка памяти (35 баллов)

Пробегаем по числу слева направо и ищем цифру (пусть это будет цифра A), правее которой имеются цифры меньшие, чем A .

- 1) Если такой цифры A нет, то ответ 0. Задача решена.
- 2) Если такая цифра A есть и она *не является* первой цифрой заданного числа, то пусть $V_1, V_2, \dots, V_k$ – это цифры, которые меньше и правее, чем A . Цифры $V_1, V_2, \dots, V_k$ могут повторяться. Среди них находим минимальную. Если она повторяется несколько раз, то берём её самое правое вхождение. Это и есть искомая цифра, которую нужно поменять местами с цифрой A . Задача решена.
- 3) Если цифра A является первой цифрой заданного числа, а $V_1, V_2, \dots, V_k$ – это цифры, которые меньше, чем A , то среди них ищем минимальную, *строго больше нуля*. Если она повторяется несколько раз, то берём её самое правое вхождение. Это и есть искомая цифра, которую нужно поменять местами с цифрой A . Задача решена.
- 4) Описанный алгоритм имеет сложность $O(N^2)$, где N – это количество цифр в заданном числе. Поскольку $N \leq 10^3$, то такое решение проходит по времени.

Pascal

```
var n,i,j,x,y,z: int64;  
    a: array [1..1000] of byte;  
  
begin  
  
    read(n);  
  
    for i:=1 to n do read(a[i]);  
  
    x:=0;  
    for i:=n-1 downto 2 do  
        begin z:=9;  
            for j:=i+1 to n do  
                if (a[i]>a[j]) and (a[j]<=z) then  
                    begin x:=i; y:=j; z:=a[j]; end;  
            end;  
  
            z:=9;  
            for j:=2 to n do  
                if (a[1]>a[j]) and (a[j]<=z) and (a[j]>0) then  
                    begin x:=1; y:=j; z:=a[j]; end;  
  
            if x=0 then write(0) else write(x,' ',y);  
  
end.
```

C++

```
signed main() {  
    // #ifdef _DEBUG  
    freopen("input.txt", "r", stdin); freopen("output.txt", "w", stdout);  
    // #endif  
    ios_base::sync_with_stdio(0); cin.tie(0);  
    int n;  
    cin >> n;  
    vector<int> v(n);  
    for (auto& i : v) {  
        cin >> i;  
    }  
    int pos_min = -1;  
    for (int j = n - 1; j > 0; --j) {  
        if (v[j] != 0 && v[j] < v[0]) {  
            if (pos_min == -1 || v[j] < v[pos_min]) {  
                pos_min = j;  
            }  
        }  
    }  
    if (pos_min != -1) {  
        cout << 1 << ' ' << pos_min + 1 << endl;  
        return 0;  
    }  
    int l = -1, r = -1;  
    pos_min = n - 1;  
    for (int j = n - 2; j > 0; --j) {  
        if (v[j] < v[pos_min])  
            pos_min = j;  
        if (v[j] > v[pos_min]) {  
            l = j;  
            r = pos_min;  
        }  
    }  
    if (l == -1)  
        cout << 0 << endl;  
    else  
        cout << l + 1 << ' ' << r + 1 << endl;  
    return 0;  
}
```

Python

```
n = int(input())
a = list(map(int, input().split()))

mi, mj = -1, -1
for i in range(n):
    mj = -1
    for j in range(i + 1, n):
        if a[i] > a[j] and (i > 0 or a[j] != 0) and (mj == -1 or a[mj] >= a[j]):
            mi, mj = i, j
    if mj != -1:
        break
if mi == -1:
    print(0)
else:
    print(mi + 1, mj + 1)
```

В. Мелочёвая проблема (30 баллов)

Денис изобрёл свою собственную денежную систему. В ней нет купюр, а есть только монеты разного достоинства. Он вообразил себя кассиром соседнего супермаркета, а своего друга Мишку – покупателем, которому он должен сдать сдачу в размере S рублей. Он заглянул в свою кассу, пересчитал там всю мелочь и задумался: каким минимальным количеством он сможет сдать Мишке сдачу?

Формат входных данных

В первой строке входного файла *input.txt* записаны через пробел два натуральных числа S и N – размер сдачи в рублях и общее количество монет в кассе ($1 \leq S \leq 10^{10}$, $1 \leq N \leq 20$). Во второй строке входного файла – последовательность из N натуральных чисел, разделённых пробелом, не превосходящих 10^{10} . Эти числа означают номиналы имеющихся в кассе монет и могут повторяться.

Формат выходных данных

В единственной строке выходного файла *output.txt* должно быть записано минимальное количество монет из входного файла, составляющих сумму S . Если же из имеющихся монет набрать сумму S не получится, то выдать 0.

Примеры

input.txt	output.txt
15 5 3 8 7 4 7	2
9 5 3 8 7 4 7	0

Мелочёвая проблема (30 баллов)

- 1) Задача решается перебором всех возможных комбинаций монет, имеющихся в кассе, и проверкой для каждой комбинации – даёт ли она требуемую сумму S .
- 2) Если имеется N монет, то таких комбинаций будет ровно $2^N - 1$, так как их можно «закодировать» N -разрядными векторами из 0 и 1. Если k -я монета присутствует в данной комбинации, то в кодирующем её векторе на k -й позиции будет стоять 1.
- 3) Чтобы сгенерировать все возможные N -разрядные вектора из 0 и 1, можно перевести в двоичную систему все числа от 1 до $2^N - 1$.
- 4) Описанный алгоритм имеет сложность $O(N \cdot 2^N)$, где N – это количество монет в кассе. При ограничении $N \leq 20$ такое решение проходит по времени.

Pascal

```
var i,j,k,n,s,x,y,z,t,b: int64;  
    a: array [1..20] of int64;  
  
begin  
  
    readln(s,n);  
  
    z:=1;  
    for i:=1 to n do  
        begin read(a[i]); z:=z*2 end;  
  
    k:=n;  
    b:=0;  
    for j:=1 to z-1 do  
        begin t:=j;  
            y:=0;  
            x:=0;  
            for i:=1 to n do  
                begin if (t mod 2) = 1 then x:=x+a[i];  
                    y:=y + (t mod 2);  
                    t:=t div 2;  
                end;  
                if x=s then  
                    begin if k>y then k:=y; b:=1; end;  
                end;  
  
            if b=1 then write(k) else write(0);  
        end.
```

C++

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

const int INF = (int)1e9;

int main() {
    int n;
    int64_t need;
    cin >> need >> n;
    vector<int64_t> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    int res = INF;
    for (int m = 1; m < (1 << n); m++) {
        int cnt = 0;
        int64_t s = 0;
        for (int i = 0; i < n; i++) {
            if ((m >> i) & 1) {
                s += a[i];
                cnt += 1;
            }
        }
        if (s == need) {
            res = min(res, cnt);
        }
    }
    cout << (res == INF ? 0 : res) << endl;
}
```

Python

```
s, n = map(int, input().split())
a = list(map(int, input().split()))
ans = 100
for mask in range(2 ** n):
    cnt = 0
    sum = 0
    i = 0
    while mask:
        if mask & 1:
            cnt += 1
            sum += a[i]
        i += 1
        mask >>= 1
    if sum == s:
        ans = min(ans, cnt)
if ans > n:
    ans = 0
print(ans)
```

С. Литературный отрывок (35 баллов)

К следующему уроку литературы Денис должен был выполнить разбор литературного произведения. Это оказалось непростой задачей. Сначала Денис посчитал количество букв в нём. А потом нашёл его самый короткий отрывок, содержащий все буквы, которые встречаются в этом литературном произведении.

Требуется в заданной символьной строке найти самую короткую подстроку, содержащую все буквы, встречающиеся в заданной строке.

Формат входных данных

В первой строке входного файла *input.txt* записано натуральное число L – длина символьной строки ($1 \leq L \leq 10^4$). Во второй строке входного файла записана сама символьная строка, состоящая из L заглавных букв латинского алфавита.

Формат выходных данных

В единственной строке выходного файла *output.txt* должно быть записано натуральное число – минимальная длина подстроки заданной строки, содержащей все буквы, которые встречаются в заданной строке.

Пример

input.txt	output.txt
10 LITERATURE	8

Литературный отрывок (35 баллов)

- 1) Первый раз пробегаем по заданной строке слева направо и подсчитываем количество различных букв в ней. Это удобно делать с помощью массива X , у которого индексы элементов не числовые, а символьные и меняются в пределах от «A» до «Z».
- 2) Далее используем два указателя. Первый указатель меняется во внешнем цикле в пределах от 1 до L (длина строки) и указывает на начало текущей подстроки. Второй указатель меняется во внутреннем цикле и указывает на конец текущей подстроки.
- 3) Для текущей подстроки проверяем, все ли буквы исходной строки в ней присутствуют. Эту проверку можно делать с помощью массива X из п.1), обнуляя его всякий раз, когда меняется указатель начала подстроки.
- 4) Запоминаем длину текущей подстроки, если она является «подходящей», т.е. содержит все буквы исходной строки и короче, чем предыдущая «подходящая» подстрока.
- 5) Описанный алгоритм имеет сложность $O(L^2)$. При ограничении $L \leq 10^4$ такое решение проходит по времени.

Pascal

```
var n,L,x,i,j,k,t: int64;
    c: char;
    s: array [1..10000] of char;
    a: array ['A'..'Z'] of int64;

begin

    readln(n);

    for c:='A' to 'Z' do a[c]:=0;
    t:=0;
    for i:=1 to n do
        begin read (s[i]);
            if a[s[i]]=0 then t:=t+1;
            a[s[i]]:=1; end;

    L:=n;
    for i:=1 to n do
        begin x:=0;
            for c:='A' to 'Z' do a[c]:=0;
            for j:=i to n do
                begin if a[s[j]]=0 then x:=x+1;
                    a[s[j]]:=1;
                    if (x=t) and (j-i+1<L) then L:=j-i+1;
                end;
            end;

        write(L);

    end.
```

C++

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

    int letters_cnt = (int)unordered_set<char>(s.begin(), s.end()).size();

    int res = (int)s.size();
    for (int i = 0; i < n; i++) {
        unordered_set<char> letters;
        for (int j = i; j < n; j++) {
            letters.insert(s[j]);
            if ((int)letters.size() == letters_cnt) {
                res = min(res, j - i + 1);
                break;
            }
        }
    }
    cout << res << endl;
}
```

Python

```
n = int(input())
s = input()

assert len(s) == n
assert s.isalpha()
assert s.isupper()
assert s.isascii()

total = len(set(s))
curr = 0
cnt = {chr(i): 0 for i in range(ord("A"), ord("Z") + 1)}
i = 0
j = 0
ans = n
while j < n:
    while j < n and curr < total:
        if not cnt[s[j]]:
            curr += 1
        cnt[s[j]] += 1
        j += 1
    while cnt[s[i]] > 1:
        cnt[s[i]] -= 1
        i += 1
    if curr == total:
        ans = min(ans, j - i)
    cnt[s[i]] -= 1
    curr -= 1
    i += 1
print(ans)
```

D. Туристическая тропа (40 баллов)

Мишка любит прогуляться в лесу по туристической тропе. В начале тропы имеется отметка «0-й километр». Начальную и конечную точку для своей прогулки Мишка всегда выбирает произвольно. Например, в прошлый раз он прошагал от отметки «2-й километр» до отметки «5-й километр». После всех прогулочных дней Мишка решил выяснить, какова суммарная длина тех участков тропы, на которые его нога ещё ни разу не ступала.

Формат входных данных

В первой строке входного файла *input.txt* записаны через пробел два натуральных числа L и N – длина туристической тропы в километрах и количество прогулочных дней ($1 \leq L \leq 10^6$, $1 \leq N \leq 10^4$). В каждой из последующих N строк записаны через пробел *в возрастающем порядке* по два целых неотрицательных числа – километровые отметки начальной и конечной точек очередной прогулки. В начале тропы имеется отметка «0-й километр», в конце тропы – « L -й километр».

Формат выходных данных

В единственной строке выходного файла *output.txt* должно быть записано целое неотрицательное число – суммарная длина тех участков тропы, по которым Мишка ещё ни разу не прогуливался.

Пример

input.txt	output.txt
10 3 1 5 7 9 2 5	4

Пояснение к примеру: не пройденными оказались участки от 0-го до 1-го километра, от 5-го до 7-го километра и от 9-го до 10-го километра. Их суммарная длина составляет 4 километра.

Туристическая тропа (40 баллов)

- 1) Отсортируем все по возрастанию все километровые отметки, которые встречаются во входном файле. Получим массив X из $2 \cdot N$ элементов, среди которых могут быть повторяющиеся элементы.
- 2) В процессе сортировки для каждого элемента запоминаем, является он «началом» или «концом» прогулки.
- 3) Устанавливаем начальное значение счётчика $K = 0$.
- 4) Пробегаем по элементам массива X и увеличиваем счётчик K на 1, если очередной элемент оказался «началом» прогулки, либо уменьшаем на 1, если очередной элемент – это «конец» прогулки.

- 5) Если текущий элемент оказался «концом» прогулки, а счётчик K обнулится, то это означает, что интервал от текущего до следующего элемента массива X является ни разу не пройденным участком тропинки. Его длину добавляем к суммарной накопленной длине ни разу не пройденных участков.
- 6) Описанный алгоритм имеет сложность $O(N^2)$ или $O(N \log N)$, где N – это количество прогулочных дней в заданном числе. При $N \leq 10^4$ такое решение проходит по времени.

Pascal

```
var n,L,i,j,k,p,q,s: int64;
    a: array [0..20000,1..2] of int64;

begin
  read(L);
  read(n);
  for i:=1 to n do
    begin read(a[i,1],a[n+i,1]);
           a[i,2]:=0;
           a[n+i,2]:=1;
    end;

  for i:=1 to 2*n-1 do
    for j:=i+1 to 2*n do
      if a[i,1]>a[j,1] then
        begin k:=a[i,1]; a[i,1]:=a[j,1]; a[j,1]:=k;
              k:=a[i,2]; a[i,2]:=a[j,2]; a[j,2]:=k;
        end;

  k:=1;
  p:=0;
  q:=0;
  s:=0;
  a[0,1]:=0;
  a[0,2]:=1;
  for i:=0 to 2*n-1 do
    begin if a[i,2]=0 then k:=k+1 else k:=k-1;
          if (k=0) and (a[i,2]=1) then s:=s+a[i+1,1]-a[i,1];
    end;

  write(s+L-a[2*n,1]);

end.
```

C++

```
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    int l, n;
    cin >> l >> n;
    vector<pair<int, int>> a(n);
    for(i, n)
        cin >> a[i].first >> a[i].second;
    sort(all(a));
    int mx = 0;
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        ans += max(a[i].first - mx, 0);
        mx = max(a[i].second, mx);
    }
    ans += l - mx;
    cout << ans;
    return 0;
}
```

Python

```
l, n = map(int, input().split())
events = []
for i in range(n):
    a, b = map(int, input().split())
    events.append((a, 1))
    events.append((b, -1))
events.sort()
opened = 0
ans = 0
last = 0
for event in events:
    if opened == 0:
        ans += event[0] - last
        opened += event[1]
        last = event[0]
print(ans + 1 - last)
```

Задача «Дачный бассейн»



Е. Дачный бассейн (60 баллов)

Денис планирует разместить на дачном участке бассейн прямоугольной формы. Размеры бассейна ограничивают растущие на участке деревья и границы самого участка, которые, естественно, нельзя передвинуть.

Будем считать, что дачный участок имеет прямоугольную форму, две стороны которого лежат на координатных осях. Пусть нам известны размеры дачного участка и координаты всех деревьев на участке. Требуется найти размеры прямоугольного бассейна максимальной площади со сторонами, параллельными осям координат. Ни одно дерево не должно оказаться внутри бассейна, но может быть на его границе.

Формат входных данных

В первой строке входного файла *input.txt* записаны через пробел два натуральных числа A и B ($1 \leq A, B \leq 10^4$) – это длины сторон прямоугольника (дачного участка), лежащих соответственно на координатных осях OX и OY . Во второй строке входного файла записано натуральное число N – количество деревьев на участке ($1 \leq N \leq 25$). В каждой из последующих N строк записаны через пробел по два целых неотрицательных числа – координаты точек, обозначающих деревья на участке. Все точки попарно различны и располагаются внутри или на границе дачного участка.

Формат выходных данных

В единственной строке выходного файла *output.txt* должны быть записаны два натуральных числа в *неубывающем* порядке – длины сторон прямоугольного бассейна максимальной площади, который можно разместить на дачном участке, так чтобы он не выходил за границы участка, и ни одно дерево не оказалось внутри бассейна. Если существует несколько прямоугольников с максимальной площадью, то в качестве ответа нужно выбрать тот из них, который имеет минимальный периметр.

Формат выходных данных

В единственной строке выходного файла *output.txt* должны быть записаны два натуральных числа в *неубывающем* порядке – длины сторон прямоугольного бассейна максимальной площади, который можно разместить на дачном участке, так чтобы он не выходил за границы участка, и ни одно дерево не оказалось внутри бассейна. Если существует несколько прямоугольников с максимальной площадью, то в качестве ответа нужно выбрать тот из них, который имеет минимальный периметр.

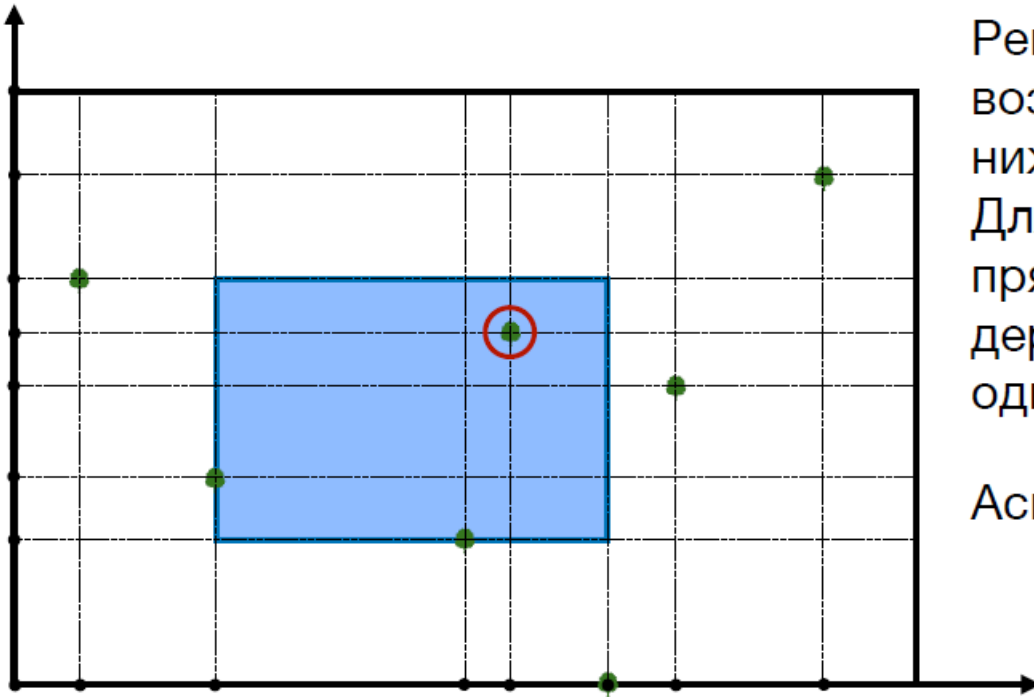
Примеры

input.txt	output.txt
10 7 1 3 5	5 10
10 7 2 3 5 6 3	5 6

Пояснение ко второму примеру: дачный участок – это прямоугольник 10x7. Его вершины имеют координаты (0,0), (10,0), (0,7) и (10,7). Два дерева расположены в точках с координатами (3,5) и (6,3). Максимальную площадь 30 имеют два прямоугольника: 6x5 и 10x3. Вершины первого из них имеют координаты (0,0), (6,0), (0,5) и (6,5). Его периметр $2 \cdot (5 + 6) = 22$. Вершины второго прямоугольника имеют координаты (0,0), (10,0), (0,3) и (10,3). Его периметр $2 \cdot (10 + 3) = 26$. Согласно условию задачи, из этих двух прямоугольников следует выбрать тот, который имеет меньший периметр, и перечислить длины его сторон в неубывающем порядке.

Дачный бассейн

Идея 1. Абсциссы (ординаты) вершин самого большого бассейна всегда совпадают с абсциссами (ординатами) деревьев или границ участка.

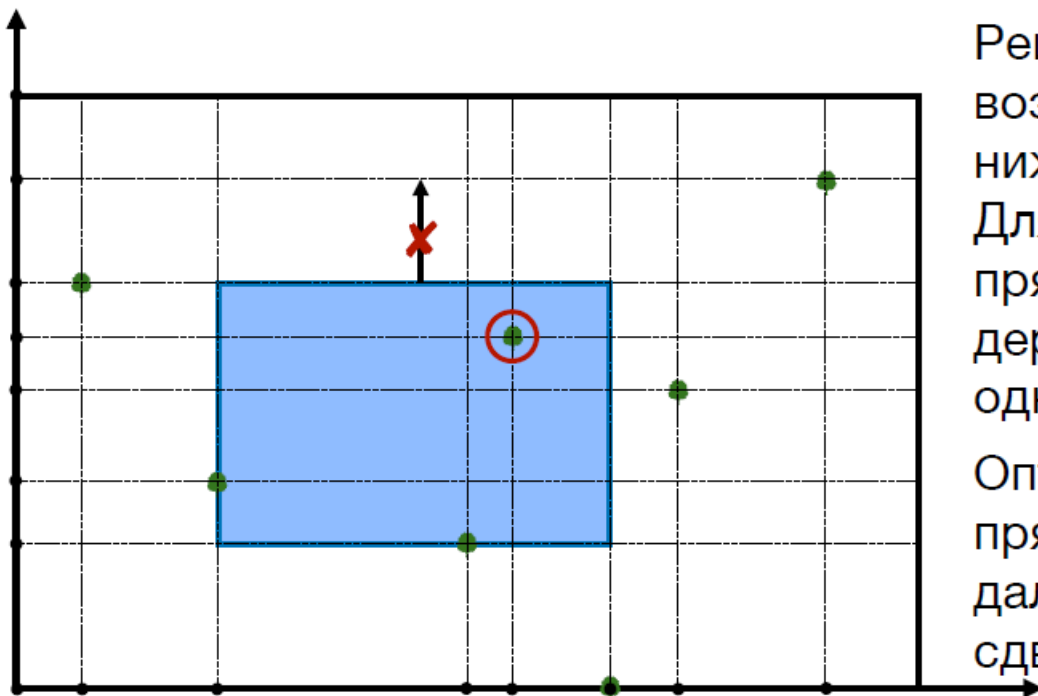


Решение: переберем все возможные левые, правые, нижние и верхние границы. Для каждого такого прямоугольника переберем все деревья и проверим, нет ли хоть одного дерева внутри.

Асимптотика: $O(n^5)$.

Дачный бассейн

Идея 1. Абсциссы (ординаты) вершин самого большого бассейна всегда совпадают с абсциссами (ординатами) деревьев или границ участка.

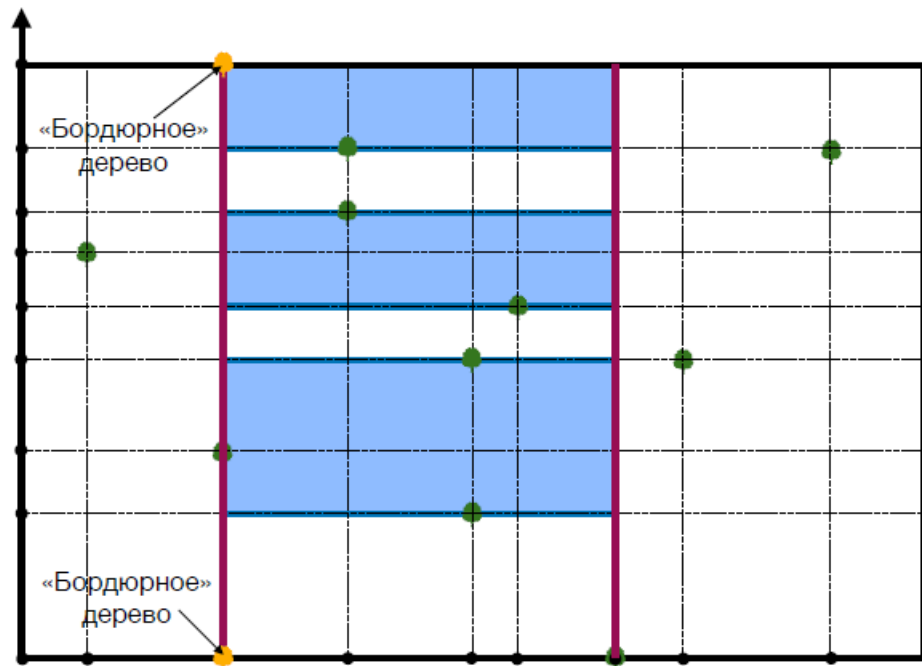


Решение: переберем все возможные левые, правые, нижние и верхние границы. Для каждого такого прямоугольника переберем все деревья и проверим, нет ли хоть одного дерева внутри.

Оптимизация: если в каком-то прямоугольнике нашли дерево, дальше верхнюю границу не сдвигаем, выходим из цикла.

Дачный бассейн

Идея 2. Если левая и правая граница фиксированы, то все подходящие бассейны находятся между «соседними по вертикали» деревьями из этой полосы.

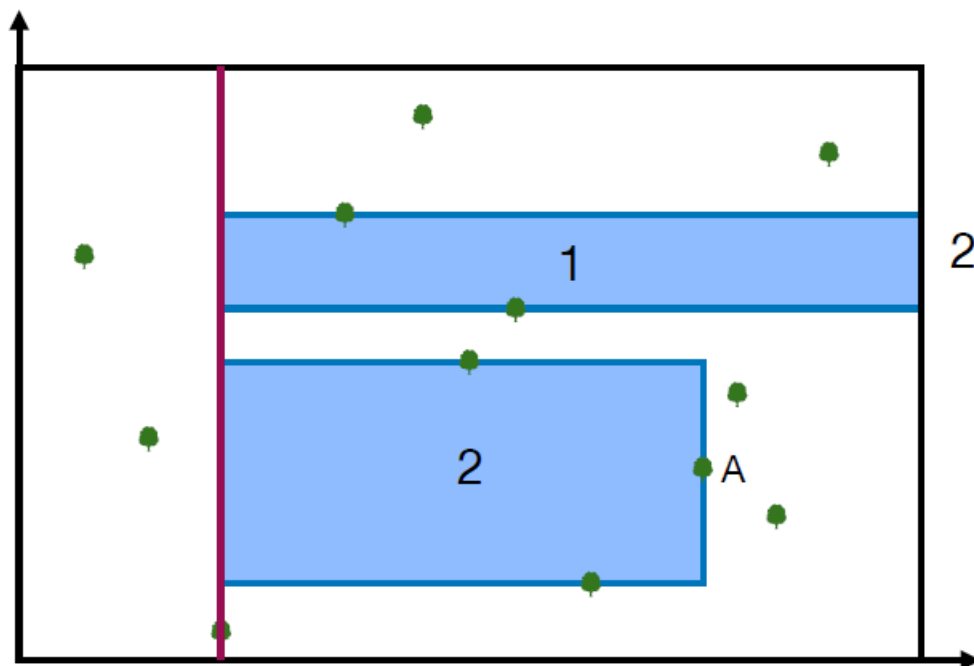


Сортируем возможные абсциссы слева направо. Перебираем левые границы. Перебираем правые границы, поддерживая упорядоченный снизу вверх список деревьев, попадающих в полосу. При сдвиге правой границы в список добавляется новое дерево (или несколько). Перебираем бассейны, лежащие между соседними деревьями.

Асимптотика: $O(n^3)$.

Дачный бассейн

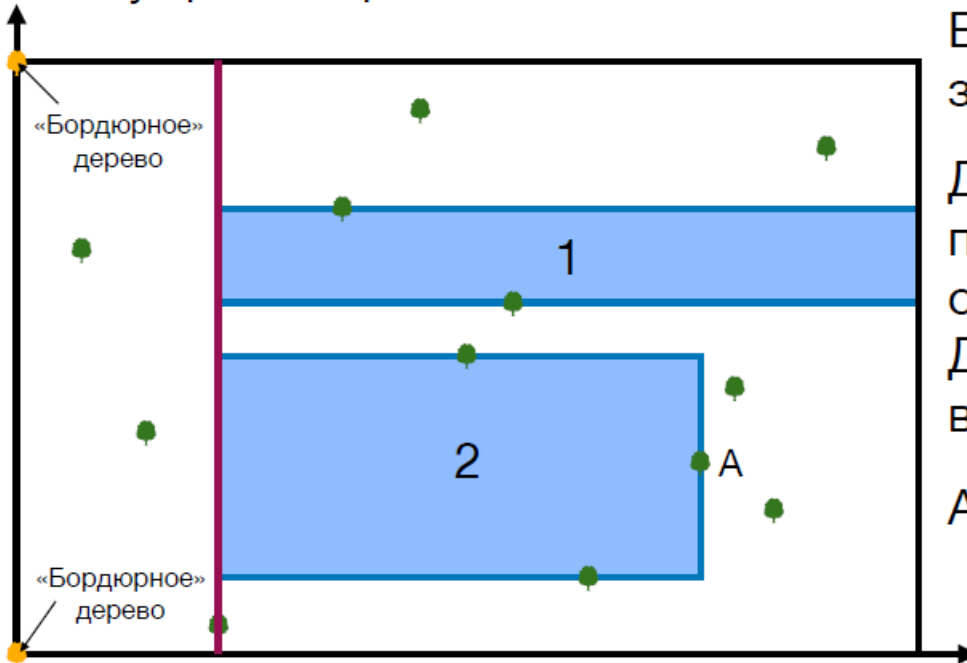
Идея 3. Если левая граница фиксирована, то подходящие бассейны могут быть двух видов:



1. Правая граница совпадает с границей участка. Бассейн расположен между «соседними по вертикали» деревьями из правой части участка.
2. Правая граница упирается в дерево А. Верхняя граница - ближайшее сверху дерево, лежащее левее А (из деревьев в правой части участка). Нижняя граница - ближайшее снизу дерево, лежащее левее А.

Дачный бассейн

Поддерживаем упорядоченный снизу вверх список деревьев, лежащих в правой части участка. При сдвиге левой границы удаляем из него деревья с текущей абсциссой.



Все бассейны вида 1 перебираем за $O(n)$.

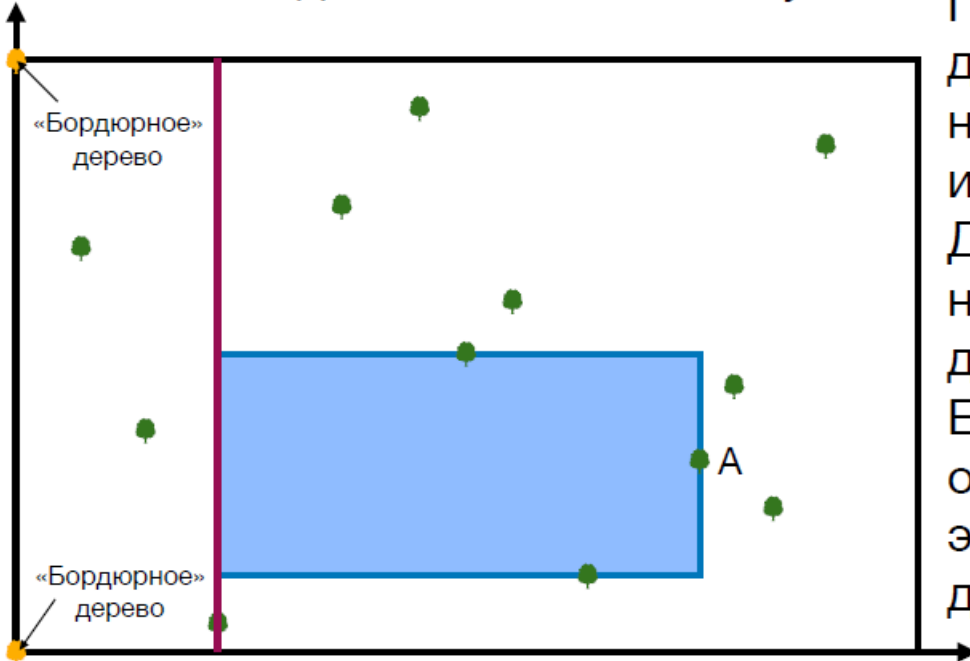
Для перебора бассейнов вида 2 перебираем деревья, в которые они упрутся правой границей. Для каждого из них находим верхнюю и нижнюю границу.

Асимптотика: $O(n^3)$.

Дачный бассейн

Идея 4. Ближайшие сверху деревья, которые находятся левее дерева A, можно найти сразу для всех деревьев из списка за $O(n)$.

Аналогично для ближайших снизу.

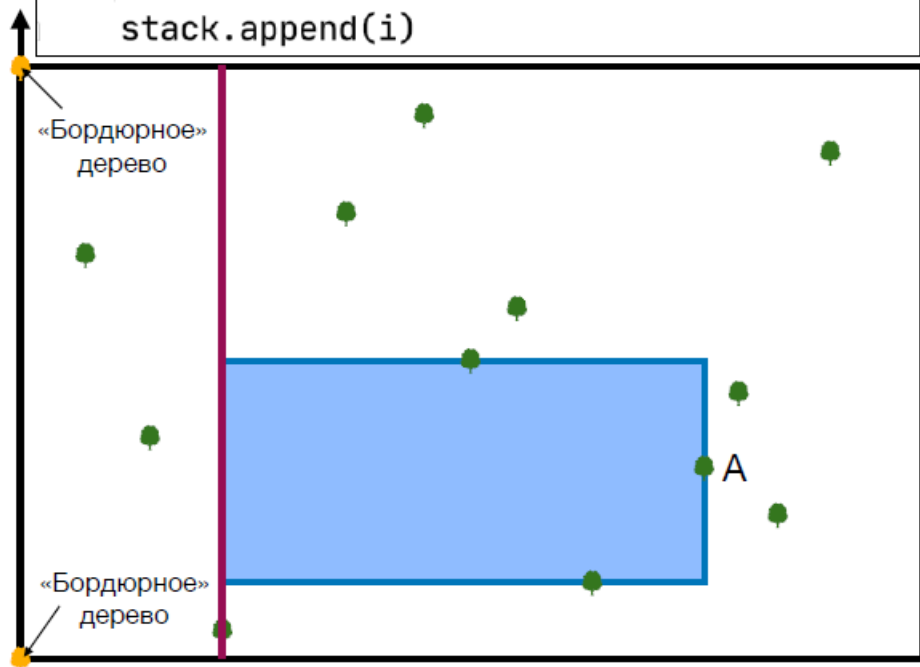


Поддерживаем стек индексов деревьев, для которых еще не найден ответ. Изначально в стеке индекс одного бордюрного дерева. Для очередного дерева проверяем, не является ли оно ответом для деревьев, лежащих сверху списка. Если является, записываем для них ответы и удаляем из стека. После этого добавляем индекс текущего дерева в стек.

```

stack = [0]
topmin = [0] * len(trees)
for i in range(1, len(trees)):
    while trees[i][0] < trees[stack[-1]][0]:
        topmin[stack[-1]] = i
        stack.pop()
    stack.append(i)

```



Дачный бассейн

Поддерживаем стек индексов деревьев, для которых еще не найден ответ. Изначально в стеке индекс одного бордюрного дерева. Для очередного дерева проверяем, не является ли оно ответом для деревьев, лежащих сверху списка. Если является, записываем для них ответы и удаляем из стека. После этого добавляем индекс текущего дерева в стек.

После такого подсчета перебираем все бассейны вида 2 за $O(n)$. Итоговая асимптотика: $O(n^2)$.

C++

```
struct Pnt {
    int x, y;

    explicit Pnt(int x=0, int y=0) : x(x), y(y) { }
    void read() {
        cin >> x >> y;
    }
    bool operator<(Pnt const& p) const {
        return make_pair(y, x) < make_pair(p.y, p.x);
    }
    bool operator==(Pnt const& p) const {
        return make_pair(y, x) == make_pair(p.y, p.x);
    }
};

int main() {
    int w, h, n;
    cin >> w >> h >> n;
    vector<Pnt> pnts(n);
    for (int i = 0; i < n; i++) {
        pnts[i].read();
    }
    if (count(pnts.begin(), pnts.end(), Pnt(0, 0)) == 0) {
        pnts.push_back(Pnt(0, 0)); n++;
    }
    if (count(pnts.begin(), pnts.end(), Pnt(w, h)) == 0) {
        pnts.push_back(Pnt(w, h)); n++;
    }
    sort(pnts.begin(), pnts.end());
}
```

```

int res_a = 0, res_b = 0;
for (int i = 0, j = 0; i < n; i = j) {
    int y_first = pnts[i].y;
    while (j < n && pnts[j].y == y_first) {
        j++;
    }
    set<int> pnts_set = {0, w};
    map<int, int> dist; dist[w] = 1;

    for (int k = j; k < n; k++) {
        map<int, int>::const_iterator it_last = dist.end();
        int max_dist = (--it_last)->first;
        int y = pnts[k].y;
        int area = (y - y_first) * max_dist, perimeter = (y - y_first) + max_dist;
        // cerr << "! " << max_dist << " " << (y - y_first) << endl;
        if (make_pair(res_a * res_b, -res_a - res_b) < make_pair(area, -perimeter)) {
            res_a = max_dist;
            res_b = y - y_first;
        }

        int x = pnts[k].x;
        set<int>::const_iterator it = pnts_set.lower_bound(x);
        if (*it != x) {
            set<int>::const_iterator it2 = it; --it;
            int d = *it2 - *it;
            dist[d]--;
            if (dist[d] == 0) {
                dist.erase(d);
            }
            dist[x - *it]++;
            dist[*it2 - x]++;
            pnts_set.insert(x);
        }
    }
}
if (res_a > res_b) {
    swap(res_a, res_b);
}
cout << res_a << " " << res_b << endl;
}

```

Python

```
def update_ans(a, b):
    global ans
    if a * b > ans[0] * ans[1]:
        ans = (a, b)
    elif a * b == ans[0] * ans[1] and 2 * (a + b) < 2 * (ans[0] + ans[1]):
        ans = (a, b)

w, h = map(int, input().split())
n = int(input())
trees = [(0, 0)] + sorted(tuple(map(int, input().split())) for _ in range(n))
rest = [(0, 0)] + sorted(trees, key=lambda x: (x[1], x[0])) + [(0, h)]
ans = (0, 0)
for tree in trees:
    rest.remove(tree)
    for i in range(1, len(rest)):
        update_ans(rest[i][1] - rest[i - 1][1], w - tree[0])
    stack = [0]
    rightmin = [0] * len(rest)
    for i in range(1, len(rest)):
        while rest[i][0] < rest[stack[-1]][0]:
            rightmin[stack[-1]] = i
            stack.pop()
        stack.append(i)
    stack = [len(rest) - 1]
    leftmin = [0] * len(rest)
    for i in range(len(rest) - 2, -1, -1):
        while rest[i][0] < rest[stack[-1]][0]:
            leftmin[stack[-1]] = i
            stack.pop()
        stack.append(i)
    for i in range(1, len(rest) - 1):
        update_ans(rest[rightmin[i]][1] - rest[leftmin[i]][1], rest[i][0] - tree[0])
print(*sorted(ans))
```

Спасибо за внимание!

