【模拟】-绘图机器

题目描述与示例

题目描述

绘图机器的绘图笔初始位置在原点 `(0,0)` ，机器启动后其绘图笔按下面规则绘制直线：

1) 尝试沿着横向坐标轴正向绘制直线，直到给定的终点值 `E` 。

2) 期间可通过指令在纵坐标轴方向进行偏移，并同时绘制直线，偏移后按规则 `1` 绘制直线。

指令的格式为 `X offsetY` ，表示在横坐标 `X` 沿纵坐标方向偏移， `offsetY` 为正数表示正向偏移，为负数表示负向偏移。

给定了横坐标终点值 `E` 、以及若干条绘制指令，请计算绘制的直线和横坐标轴、以及 `X=E` 的直线组成图形的面积。

输入描述

首行为两个整数 `N E` ，表示有 `N` 条指令，机器运行的横坐标终点值 `E` 。

接下来 `N` 行，每行两个整数表示一条绘制指令 `X offsetY` ，用例保证横坐标 `X` 以递增排序方式出现，且不会出现相同横坐标 `X` 。

取值范围: `0 < N <= 10000, 0 <= X <= E <= 20000, -10000 <= offsetY <= 10000` 。

输出描述

一个整数，表示计算得到的面积，用例保证，结果范围在 `0~4294967295` 内

示例

输入

```
```

```
4 10
1 1
2 1
3 1
4 -2
```

输出

```
12
```

解题思路

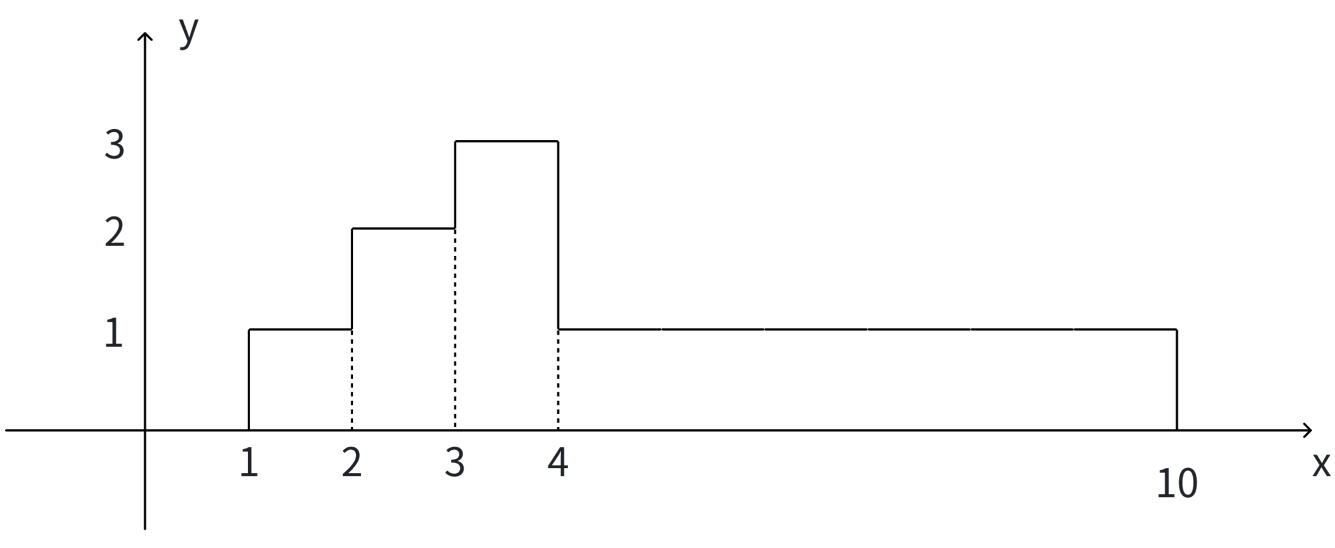
题目很难读，建议根据例子反推含义。

主要是对绘制指令 `X offsetY` 的理解。

绘制指令 `X offsetY` 表示，在 `x` 轴的 `X` 位置，向 `y` 轴偏移 `offsetY` 。

（注意单词offset是表示偏移的意思，所以绘制指令 `X offsetY` 表示一个动作，而不是一个具体的坐标）

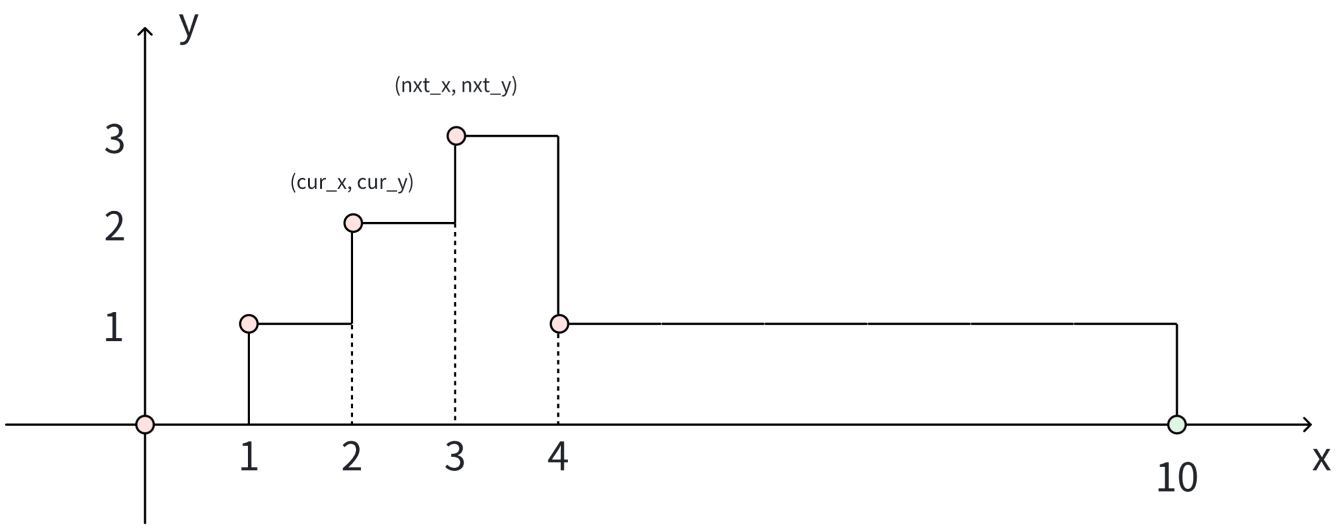
示例对应的图如下



可以看出，线段和 `x` 轴围成的面积是 `1*1 + 1*2 + 1*3 + 6*1 = 12`

理解了题意之后，问题就比较简单了。

我们初始化当前点为原点 `(cur_x, cur_y) = (0, 0)`，且初始面积 `area = 0`。



如图所示，如果我们知道图中标注出来的点的坐标，就可以计算出所有的面积。

如果已知当前点的坐标 `(cur_x, cur_y)` 和指令 `X offsetY`，那么下一个点的坐标可以很方便地通过以下式子计算得到 `(nxt_x, nxt_y) = (X, cur_y + offsetY)`，而围成的面积的增量是 `(nxt_x - cur_x) * cur_y`。

特别的，如果出现了直线位于 `x` 轴下方，那么面积需要取绝对值，即 `abs((nxt_x - cur_x) * cur_y)`。

故单次更新面积的代码为

```
X, offsetY = lst[i]
nxt_x, nxt_y = X, cur_y + offsetY
area += abs((nxt_x - cur_x) * cur_y)
cur_x, cur_y = nxt_x, nxt_y
```

代码

Python

```python
# 题目：【模拟】2024E-绘图机器
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟
# 代码看不懂的地方，请直接在群上提问


# 输入指令条数n，终点的横坐标E
n, E = map(int, input().split())

# 初始化总面积和当前点
area = 0
cur_x, cur_y = 0, 0

lst = list()
for _ in range(n):
    # 储存每一条指令的X和offsetY
    lst.append(map(int, input().split()))


# 将终点的情况加入到lst中，
# 重要的是横坐标E，对应的指令不用计算出来，后续遍历过程中也不需要用到
lst.append((E, -1))

for i in range(n+1):
    # 获得第i条指令
    X, offsetY = lst[i]
    # 计算下一个点的坐标(nxt_x, nxt_y)
    nxt_x, nxt_y = X, cur_y + offsetY
    # 更新面积
    area += abs((nxt_x - cur_x) * cur_y)
    # 更新当前点的坐标(cur_x, cur_y)
    cur_x, cur_y = nxt_x, nxt_y

print(area)
```

Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入指令条数n，终点的横坐标E
        int n = scanner.nextInt();
        long E = scanner.nextLong();

        // 初始化总面积和当前点
        long area = 0;
        long cur_x = 0, cur_y = 0;

        // 创建二维数组用于存储每一条指令的X和offsetY
        long[][] lst = new long[n + 1][2];
        for (int i = 0; i < n; i++) {
            // 储存每一条指令的X和offsetY
            lst[i][0] = scanner.nextLong();
            lst[i][1] = scanner.nextLong();
        }

        // 将终点的情况加入到lst中
        // 重要的是横坐标E，对应的指令不用计算出来，后续遍历过程中也不需要用到
        lst[n][0] = E;
        lst[n][1] = -1;

        for (int i = 0; i < n + 1; i++) {
            // 获得第i条指令
            long X = lst[i][0];
            long offsetY = lst[i][1];
            // 计算下一个点的坐标(nxt_x, nxt_y)
            long nxt_x = X;
            long nxt_y = cur_y + offsetY;
            // 更新面积
            area += Math.abs((nxt_x - cur_x) * cur_y);
            // 更新当前点的坐标(cur_x, cur_y)
            cur_x = nxt_x;
            cur_y = nxt_y;
        }

        System.out.println(area);
    }
```

```
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

int main() {
    // 输入指令条数n，终点的横坐标E
    int n;
    long long E;
    cin >> n >> E;

    // 初始化总面积和当前点
    long long area = 0;
    long long cur_x = 0, cur_y = 0;

    // 创建二维数组用于存储每一条指令的X和offsetY
    vector<vector<long long>> lst(n + 1, vector<long long>(2));
    for (int i = 0; i < n; ++i) {
        // 储存每一条指令的X和offsetY
        cin >> lst[i][0] >> lst[i][1];
    }

    // 将终点的情况加入到lst中
    // 重要的是横坐标E，对应的指令不用计算出来，后续遍历过程中也不需要用到
    lst[n][0] = E;
    lst[n][1] = -1;

    for (int i = 0; i < n + 1; ++i) {
        // 获得第i条指令
        long long X = lst[i][0];
        long long offsetY = lst[i][1];
        // 计算下一个点的坐标(nxt_x, nxt_y)
        long long nxt_x = X;
        long long nxt_y = cur_y + offsetY;
        // 更新面积
        area += abs((nxt_x - cur_x) * cur_y);
        // 更新当前点的坐标(cur_x, cur_y)
        cur_x = nxt_x;
        cur_y = nxt_y;
```

```
    }

    cout << area << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(N)`。仅需一次遍历

空间复杂度：`O(1)`。仅需若干常数变量