My Courses

平均气温

Description

A weather station collects temperature data from observation stations all over the country every day, and provides statistical inquiry services to remote users through the Internet. One of the most common types of queries is to calculate the average temperature based on observations from all observatories in the user-specified rectangular area. As more observatories continue to build, the size of the raw data itself has ballooned. In addition, although it can be assumed that the data collected every day is relatively fixed, as the user population expands, the frequency of queries increases. In view of the fact that the efficiency of the traditional brute force algorithm can no longer meet the practical requirements, the weather station has to ask you to help, improve the efficiency of the query by improving the data structure and algorithm.

With a set of function interfaces provided by the weather station, the server can access all the collected data and report the results of the query.

Interface description

```
int GetNumOfStation(void);
```

This function must be called first, which returns the number n of observatories.

```
void GetStationInfo(int no, int *x, int *y, int *temp);
```

This function returns information of no-th observatories: its geographic coordinates (*x, *y) and its measured temperature value *temp. The measurement accuracy of each observation station is based on 0.01 ° C, for example, 12.34 ° C is expressed as an integer of 1234.

```
int GetQuery(int *x1, int *y1, int *x2, int *y2);
```

This function receives the next query request. If it returns 1, it means this is a valid query. The four sides of the matrix area are parallel to the x or y axis, respectively. (*x1, *y1) and (*x2, *y2) are the coordinates of their southwest and northeast corners, respectively. An observatory that passes through a rectangular boundary is also considered to fall within it. If it returns 0, it means there are no more queries and your program can exit.

```
void Response(int temp);
```

For the current query, you can truncate the result after calculating the corresponding average temperature (for example, the output of 12.345 °C is 1234, the output of -12.345 °C is -1234), and then sent to the interface.

Pay Attention: When a query is received by GetQuery(), if the result of the query is not reported by the Response() function, an error will be output because the result of the previous query cannot be reported. That is, GetQuery() and Response() must be called alternately, n times each.

Test description

For your debugging and testing, the temperature.h and temperature_lib.c files are included with the problem. The former stipulates the above interface, the latter is an implementation of this set of interfaces - the implementation on OJ is different, but the interface is completely consistent. You can compile them with your code when you debug, but you don't have to submit them when testing. Even if you submit them, OJ will ignore them automatically.

download interface file (attachment/7c8e/7c8eb67a41099c1ad3ac803a75e25b76df2c0bfd.zip)

Input

When you debug offline, the three input interfaces implemented by temperature_lib.c will read data from file temperature.in in current directory, so you can set different input data by changing the file temperature.in in the following format

```
The first line has two integers "n, m" where n is the number of observatories and m is the nu
mber of queries.

The following n lines describe each observatory, each line contains three integers "x, y, t"
where (x, y) is the coordinates of the observatory and t is the measured temperature value of
the station.

The next m lines describe each query operation, each line contains four integer "x1,y1,x2,y2"
where (x1,y1) represents the southwest corner and (x2,y2) represents the northeast corner.
```

Output

When you debug offline, the Response() interface implemented by temperature_lib.c will write all output results to file temperature.out after the program runs.

Output file has n lines, each line contains one integer, indicating the average temperature obtained per query.

If the query area doesn't contain any observatories, please output 0.

Example

Input

```
4 2
0 0 1000
1 1 1300
2 2 1600
3 3 1100
0 0 1 1
0 0 10 10
```

Output

```
1150
1250
```

Restrictions

0<=n<=50000

0<=m<=500000

The coordinates of the observatory is in [-2^31, 2^31), and the coordinates of query area satisfy x1<=x2 and y1<=y2.

Time: 10 sec

Memory: 256 MB

Hints

Please use 64-bit integer for temperature calculations to prevent overflow.

kd-tree

range tree

题目描述

某气象台每天都要从遍布于各地的观察站采集气温数据，并通过互联网为远程用户提供统计查询服务。其中最常见的一类查询是，根据用户指定矩形区域内所有观察站的观测值计算出平均气温。随着更多观察站的不断建立，原始数据本身的规模急剧膨胀。另外，尽管可以假设每天采集的数据相对固定，但随着用户群体的扩大，查询的频率也日益激增。鉴于传统蛮力算法的效率已无法满足实用要求，气象台只好请你帮忙，通过改进数据结构和算法，提高查询的效率。

借助气象台提供的一组函数接口，服务器端可访问已采集到的所有数据，并报告查询结果。

接口说明

```
int GetNumOfStation(void);
```

该函数必须首先调用，返回现有观察站的总数n。

```
void GetStationInfo(int no, int *x, int *y, int *temp);
```

获得第no个（0 ≤ no < n）观察站的信息：其地理坐标(*x,*y)及其所测温度值*temp。各观测站的测量精度统一以0.01℃为基准单位，比如12.34℃表示为整数1234。

```
int GetQuery(int *x1, int *y1, int *x2, int *y2);
```

接收下一查询请求。返回值1对应于一次有效的查询。矩阵区域的四边分别与x或y轴平行，(*x1,*y1)和(*x2,*y2)分别为其西南角和东北角的坐标。恰好被矩形边界穿过的观察站，也视作落在其中。若返回0，则表示没有更多的查询，你的程序可以退出。

```
void Response(int temp);
```

针对当前的查询，在计算出对应的平均气温后，你可通过这一接口报告所得数值(截断取整，比如12.345℃输出为1234，-12.345℃输出为-1234)。

特别注意：每调用GetQuery()接收一次查询后，若未能通过Response()函数报告该次查询的结果就再次调用GetQuery()接收下一查询，则将因为前次查询的结果无法报告而注定输出错误。也就是说，GetQuery()和Response()必须交替调用，各n次。

测试说明

为便于你调试和测试，随题还附带有temperature.h和temperature_lib.c文件。前者约定了上述接口，后者是这组接口的一种实现——OJ上的实现与之不同，但接口完全一致。调试时可将它们与你的代码一同编译，但在线测试时不必提交；即便提交，OJ也会自动忽略它们。

下载接口文件 (attachment/7c8e/7c8eb67a41099c1ad3ac803a75e25b76df2c0bfd.zip)

输入

脱机调试时，temperature_lib.c所实现的三个输入接口，实际上是从当前目录下的temperature.in文件读入数据，因此通过按如下格式更改该文件，即可设定不同的输入数据：

```
第一行为两个整数：观察站总数n，所需查询的总次数m
以下n行分别描述各观察站：位置坐标为整数(x, y)，该站所测得温度值为整数t
再以下m行分别对应于各次查询操作，整数(x1, y1)和(x2, y2)分别表示其西南角和东北角
```

输出

脱机调试时，temperature_lib.c所实现的Response()接口会在程序运行后，将所有的输出结果写入temperature.out文件。

文件共m行，各含1个整数，表示每次查询所得平均温度。

若查询区域不含任何观测站，则输出0。

输入样例

```
4 2
0 0 1000
1 1 1300
2 2 1600
3 3 1100
0 0 1 1
0 0 10 10
```

输出样例

```
1150
1250
```

限制

0 ≤n ≤ 50,000

0 ≤ m ≤ 500,000

观测站坐标取值范围是[-2^31, 2^31)

查询区域的坐标 x1 ≤ x2 且 y1 ≤ y2

时间限制：10秒

内存限制：256 MB

提示

温度计算请使用64位整数，以保证累加不致溢出

kd-tree

range tree
