离散化

离散化的使用场景

例题：https://www.acwing.com/problem/content/description/804/

考虑这样一个问题：假定有一个无限长的数轴，数轴上每个坐标上的数都是0。现在，我们首先进行n次操作，每次操作将某一位置x上的数加c。接下来，进行m次询问，每个询问包含两个整数l和r，你需要求出在区间$[l, r]$之间的所有数的和。

数据范围方面：

$-10^9 \leq x \leq 10^9$ $1 \leq n, m \leq 10^5$ $-10^9 \leq l \leq r \leq 10^9$ $-10000 \leq c \leq 10000$

如果x的数据范围仅仅是非负整数，并且范围比较小的话，那我们可以直接开一个数组`a`，然后每次`a[x] += c`，求区间和的话可以用前缀和数组来快速求出。

但是这道题的困难之处就在于x的范围非常大，并且还包含负数，如果强行开数组，会导致空间非常浪费。但我们同时注意到，整个用到的坐标的数量实际上最多是$10^5 + 2 \times 10^5 = 3 \times 10^5$，这个长度的数组空间方面是可以接受的。这里我们就需要“离散化”来解决了。

离散化的思想就是，先将需要用到的数都映射到范围较小的数，一般就直接映射到$1, 2, 3, \ldots$。具体做法是：

1. 先将所有要用到的数存入一个vector
2. 将vector排序，然后去重
3. 开一个哈希表存储从离散化之前到之后的映射关系。这个映射关系也可以通过二分来求，或者通过C++内置函数`lower_bound`或者`upper_bound`。当然从离散化之后到之前的映射关系已经可以通过上面的vector来得到了

代码如下：

```
// xs存了所有需要用到的离散化之前的数，先排序
sort(xs.begin(), xs.end());
// 去重
xs.erase(unique(xs.begin(), xs.end()), xs.end());
// 开一个unordered_map，存储一下离散化的映射关系。一般映射到1,2,...
mp.reserve(xs.size());
for (int i = 0; i < xs.size(); i++) mp.emplace(xs[i], i + 1);
```

后面应答询问的时候，可以先将询问的左右边界求成离散化之后的值，然后根据新值来应答询问即可。代码如下：

```cpp
#include <algorithm>
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;
using PII = pair<int, int>;

const int N = 3e5 + 10;
int n, m;
int a[N], s[N];
vector<int> xs;
vector<PII> adds, qs;
unordered_map<int, int> mp;

int main() {
  scanf("%d%d", &n, &m);
  xs.reserve(n + 2 * m);
  for (int i = 1; i <= n; i++) {
    int x, c;
    scanf("%d%d", &x, &c);
    adds.emplace_back(x, c);
    xs.push_back(x);
  }

  for (int i = 1; i <= m; i++) {
    int l, r;
    scanf("%d%d", &l, &r);
    qs.emplace_back(l, r);
    xs.push_back(l), xs.push_back(r);
  }

  sort(xs.begin(), xs.end());
  xs.erase(unique(xs.begin(), xs.end()), xs.end());
  mp.reserve(xs.size());
  for (int i = 0; i < xs.size(); i++) mp.emplace(xs[i], i + 1);

  for (auto [x, c] : adds) a[mp[x]] += c;
  for (int i = 1; i <= xs.size(); i++) s[i] = s[i - 1] + a[i];

  for (auto [l, r] : qs) printf("%d\n", s[mp[r]] - s[mp[l] - 1]);
}
```

作业

上面例题