普通莫队算法

形式

假设 $n=m$，那么对于序列上的区间询问问题，如果从 $[l,r]$ 的答案能够 $O(1)$ 扩展到 $[l-1,r],[l+1,r],[l,r+1],[l,r-1]$（即与 $[l,r]$ 相邻的区间）的答案，那么可以在 $O(n\sqrt{n})$ 的复杂度内求出所有询问的答案。

实现

离线后排序，顺序处理每个询问，暴力从上一个区间的答案转移到下一个区间答案（一步步移动即可）。

排序方法

对于区间 $[l,r]$，以 l 所在块的编号为第一关键字，r 为第二关键字从小到大排序。

模板

```
inline void move(int pos, int sign) {
  // update nowAns
}

void solve() {
  BLOCK_SIZE = int(ceil(pow(n, 0.5)));
  sort(querys, querys + m);
  for (int i = 0; i < m; ++i) {
    const query &q = querys[i];
    while (l > q.l) move(--l, 1);
    while (r < q.r) move(r++, 1);
    while (l < q.l) move(l++, -1);
    while (r > q.r) move(--r, -1);
    ans[q.id] = nowAns;
  }
}
```

例题 & 代码

> 例题 「国家集训队」小 Z 的袜子
>
> 题目大意：
>
> 有一个长度为 n 的序列 {c_i}，现在给出 m 个询问，每次给出两个数 l,r，从编号在 l 到 r 之间的数中随机选出两个不同的数，求两个数相等的概率。

思路：莫队算法模板题。

对于区间 $[l,r]$以 l 所在块的编号为第一关键字r 为第二关键字从小到大排序。

然后从序列的第一个询问开始计算答案，第一个询问通过直接暴力算出，复杂度为 $O(n)$后面的询问在前一个询问的基础上得到答案。

具体做法：

对于区间 $[i,i]$由于区间只有一个元素，我们很容易就能知道答案。然后一步一步从当前区间（已知答案）向下一个区间靠近。

我们设 $col[i]$ 表示当前颜色 i 出现了多少次ans 表示当前共有多少种可行的配对方案（有多少种可以选到一双颜色相同的袜子），然后每次移动的时候更新答案——设当前颜色为 k如果是增长区间就是 ans 加上 $C^2_{col[k]+1}-C^2_{col[k]}$如果是缩短就是 ans 减去 $C^2_{col[k]}-C^2_{col[k]-1}$

而这个询问的答案就是 $\displaystyle\frac{ans}{C^2_{r-l+1}}$

这里有个优化$C^2_a=\displaystyle\frac{a(a-1)}{2}$

所以 $C^2_{a+1}-C^2_a=\displaystyle\frac{(a+1)a}{2}-\displaystyle\frac{a(a-1)}{2}=a$

所以 $C^2_{col[k]+1}-C^2_{col[k]}=col[k]$

算法总复杂度$O(n\sqrt{n})$

下面的代码中 deno 表示答案的分母(denominator)nume 表示分子(numerator)sqn 表示块的大小$\sqrt n$arr 是输入的数组，node 是存储询问的结构体，tab 是询问序列（排序后的），col 同上所述。

注意：由于 ++l 和 --r 的存在，下面代码中的移动区间的 4 个 for 循环的位置很关键，不能改变它们之间的位置关系。

参考代码：

```
#include <algorithm>
#include <cmath>
#include <cstdio>
using namespace std;
const int N = 50005;
int n, m, maxn;
int c[N];
long long sum;
int cnt[N];
long long ans1[N], ans2[N];
struct query {
  int l, r, id;
  bool operator<(const query &x) const {
    if (l / maxn != x.l / maxn) return l < x.l;
    return (l / maxn) & 1 ? r < x.r : r > x.r;
  }
} a[N];
```

```
void add(int i) {
  sum += cnt[i];
  cnt[i]++;
}
void del(int i) {
  cnt[i]--;
  sum -= cnt[i];
}
long long gcd(long long a, long long b) { return b ? gcd(b, a % b) : a; }
int main() {
  scanf("%d%d", &n, &m);
  maxn = sqrt(n);
  for (int i = 1; i <= n; i++) scanf("%d", &c[i]);
  for (int i = 0; i < m; i++) scanf("%d%d", &a[i].l, &a[i].r), a[i].id = i;
  sort(a, a + m);
  for (int i = 0, l = 1, r = 0; i < m; i++) {
    if (a[i].l == a[i].r) {
      ans1[a[i].id] = 0, ans2[a[i].id] = 1;
      continue;
    }
    while (l < a[i].l) del(c[l++]);
    while (l > a[i].l) add(c[--l]);
    while (r < a[i].r) add(c[++r]);
    while (r > a[i].r) del(c[r--]);
    ans1[a[i].id] = sum;
    ans2[a[i].id] = (long long)(r - l + 1) * (r - l) / 2;
  }
  for (int i = 0; i < m; i++) {
    if (ans1[i] != 0) {
      long long g = gcd(ans1[i], ans2[i]);
      ans1[i] /= g, ans2[i] /= g;
    } else
      ans2[i] = 1;
    printf("%lld/%lld\n", ans1[i], ans2[i]);
  }
  return 0;
}
```

普通莫队的优化

我们看一下下面这组数据

```
// 设块的大小为 2 (假设)
1 1
2 100
3 1
4 100
```

手动模拟一下可以发现□r 指针的移动次数大概为 300 次，我们处理完第一个块之后□$l=2,r=100$□此时只

需移动两次 l 指针就可以得到第四个询问的答案，但是我们却将 r 指针移动到 1 来获取第三个询问的答案，再移动到 100 获取第四个询问的答案，这样多了九十几次的指针移动。我们怎么优化这个地方呢？这里我们就要用到奇偶化排序。

什么是奇偶化排序？奇偶化排序即对于属于奇数块的询问□r 按从小到大排序，对于属于偶数块的排序□r 从大到小排序，这样我们的 r 指针在处理完这个奇数块的问题后，将在返回的途中处理偶数块的问题，再向 n 移动处理下一个奇数块的问题，优化了 r 指针的移动次数，一般情况下，这种优化能让程序快 30% 左右。

排序代码：

压行

```
//这里有个小细节等下会讲
int unit;  //块的大小
struct node {
  int l, r, id;
  bool operator<(const node &x) const {
    return l / unit == x.l / unit
               ? (r == x.r ? 0 : ((l / unit) & 1) ^ (r < x.r))
               : l < x.l;
  }
};
```

不压行

```
struct node {
  int l, r, id;
  bool operator<(const node &x) const {
    if (l / unit != x.l / unit) return l < x.l;
    if ((l / unit) & 1)
      return r <
             x.r;  //注意这里和下面一行不能写小于（大于）等于，否则会出错（详见下面的小细节）
    return r > x.r;
  }
};
```

> 小细节：如果使用 sort 比较两个函数，不能出现 $a<b$ 和 $b<a$ 同时为真的情况，否则会运行错误。

对于压行版，如果没有 `r == x.r` 的特判，当 l 属于同一奇数块且 r 相等时，会出现上面小细节中的问题（自己手动模拟一下），对于压行版，如果写成小于（大于）等于，则也会出现同样的问题。

参考链接

[OI Wiki](https://oi-wiki.org/misc/mo-algo/)