2022.4.22模拟赛题解

A.卷

题意：

给定两个均长为 n 的序列 a,b，求 $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}\lfloor\sqrt{|a_i-b_j|}\rfloor$。

其中，$1\leq n\leq 10^6, 0\leq a_i,b_i\leq 3\times 10^6$。

分析：

暴力：$O(n^2)$ 直接求，$30pts$。

示例：

```
4
1 3 3 4
2 3 3 3
```

可以发现，数列中可能有重复的数，因此如果暴力枚举每一个数必然会做很多无用功。当计算数列中 $a_i=b_j=3$ 的情况时，如果暴力枚举就得通过 6 次枚举来计算得到答案为 0。而这个枚举次数 6，它是由两个数列中 3 的个数（2×3）得到。因此我们可以通过记录两个数列中每个数的出现次数，这样在计算时数列中相同的数只需计算一次。具体见实现。

做法一：用 set 存数的同时去重，开桶记录出现次数。

做法二：用先用 map 存数，然后用 $vector$ 存 map 中每个值的 key 和 $value$。

Code：

做法一：set

```cpp
#include <bits/stdc++.h>
#define int long long
#define N 3000010
using namespace std;

set < int > a, b;

int n, ans, x[N], y[N];

inline int read(){
    int s = 0, w = 1;
    char ch = getchar();
    for (; ch < '0' || ch > '9'; w *= ch == '-' ? -1 : 1, ch = getchar());
    for (; ch >= '0' && ch <= '9'; s = s * 10 + ch - '0', ch = getchar());
    return s * w;
```

```
}

signed main(){
    n = read();
    for (int i = 1, u; i <= n; ++i) u = read(), a.insert(u), ++x[u];
    for (int i = 1, u; i <= n; ++i) u = read(), b.insert(u), ++y[u];
    for (auto i : a)
        for (auto j : b) ans += (int)sqrt(abs(i - j)) * x[i] * y[j];
    printf("%lld\n", ans);
    return 0;
}
```

做法二：$map + vector$（这个似乎跑得更快）

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

map < int, int > a, b;
vector < pair < int, int > > x, y;

int n, ans;

inline int read(){
    int s = 0, w = 1;
    char ch = getchar();
    for (; ch < '0' || ch > '9'; w *= ch == '-' ? -1 : 1, ch = getchar());
    for (; ch >= '0' && ch <= '9'; s = s * 10 + ch - '0', ch = getchar());
    return s * w;
}

signed main(){
    n = read();
    for (int i = 1; i <= n; ++i) ++a[read()];
    for (int i = 1; i <= n; ++i) ++b[read()];
    for (auto i : a) x.push_back(i);
    for (auto i : b) y.push_back(i);
    for (auto i : x)
        for (auto j : y) ans += i.second * j.second * (int)sqrt(abs(i.first - j.first));
    printf("%lld\n", ans);
    return 0;
}
```

B.造题

题意：

给定一个 n 个点 m 条边的连通图，起点编号为 0，终点有 k 个。在从起点到终点的过程中，每到达一个点（包括起点），会有任意 d 条边暂时不能经过。求在最坏情况下（即最短路最大的情况），求最短路路径长度。

分析：

如果从起点开始跑最短路，有点不好做，我们可以从终点开始跑最短路。

我们建立一个大根堆 v。对于每一个点，在跑最短路时经过这一点时，记录到达此点的路径长度。由于到达每一点会有 d 条路径不能走，最坏情况是显然是边权最小的 d 条路径不能走，最终最短路径经过此点只能是第 $d+1$ 小的路径。因此，我们存下到此点的最小的 $d+1$ 条边。在每一次经过这一点时，将其可以走到的点所连接的边权入堆，如果该点堆中元素个数大于 $d+1$，则将元素出堆直至只剩 $d+1$ 个，再选择堆顶（即第 $d+1$ 小）的边进行扩展。具体见实现。

Code：

```cpp
#include <bits/stdc++.h>
#define int long long
#define N 1000010
using namespace std;

vector < pair < int, int > > g[N];
priority_queue < int > v[N];
priority_queue < pair < int, int > > q;

int n, m, k, d, ans, dis[N], vis[N];

inline int read(){
    int s = 0, w = 1;
    char ch = getchar();
    for (; ch < '0' || ch > '9'; w *= ch == '-' ? -1 : 1, ch = getchar());
    for (; ch >= '0' && ch <= '9'; s = s * 10 + ch - '0', ch = getchar());
    return s * w;
}

void add(int u, int v, int w){g[u].push_back(make_pair(v, w));}

int dijkstra(){
    while (!q.empty()){
        int x = q.top().second;q.pop();
        if (vis[x]) continue;vis[x] = 1;
        for (auto i : g[x]){
            int y = i.first, z = i.second;
            v[y].push(dis[x] + z);
            while (v[y].size() > d + 1) v[y].pop();
            if (v[y].size() > d && dis[y] > v[y].top()) dis[y] = v[y].top(), q.push(make_pair(-dis[y], y));
        }
    }
    return vis[0] ? dis[0] : -1;
}

signed main(){
```

```
    memset(dis, 0x3f3f3f3f, sizeof(dis));
    n = read(), m = read(), k = read(), d = read();
    for (int i = 1, u, v, w; i <= m; ++i) u = read(), v = read(), w = read(), add(u, v, w), add(v, u, w);
    for (int i = 1, x; i <= k; ++i) x = read(), q.push(make_pair(0, x)), dis[x] = 0;
    printf("%lld\n", dijkstra());
    return 0;
}
```

C.平均值

题意:

给定 n 个数，为 $a_1, a_2, a_3, \ldots, a_{n-1}, a_n$。并给定三个整数 m, L, R，求 $\max\limits_{L \le r-l+1 \le R} \dfrac{Max(l,r) - Min(l,r)}{r-l+k}$。

其中，$Max(l,r)$ 表示区间 $[l,r]$ 中的最大值，$Min(l,r)$ 表示区间 $[l,r]$ 中的最小值。

分析:

考虑一段使答案最大化的区间，两端点必为最值，除非长度小于 L，被迫加入其他数。因此可以先预处理出 $r-l+1=L$ 的情况，然后使用单调队列优化。

使用二分答案，处理下界为长度等于 L 的情况，并记答案为 x，同时需要判断是否存在 $\frac{Max(l,r)-Min(l,r)}{r-l+k} \ge x$。移项，得:

$$\begin{gathered}
Max(l,r) - Min(l,r) \ge x \times (r-l+k) \\
Max(l,r) - Min(l,r) \ge rx - lx + kx \\
Max(l,r) - Min(l,r) + lx - rx \ge kx
\end{gathered}$$

由于两端点必为最值，不妨设 $a_l \le a_r$（$a_l \ge a_r$ 的情况可以通过翻转区间再做一遍），则上式变为:

$$\begin{gathered}
a_r - a_l + lx - rx \ge kx \\
a_r - rx - (a_l - lx) \ge kx
\end{gathered}$$

令 $b_i = a_i - ix$，则上式变为 $b_r - b_l \ge kx$。 可用单调队列存储。

Code:

```
#include <bits/stdc++.h>
#define int long long
#define N 50010
using namespace std;

deque < int > dq, maxn, minn, __;

int n, m, l, r, a[N];
double ans, b[N];

inline int read(){
    int s = 0, w = 1;
```

```cpp
    char ch = getchar();
    for (; ch < '0' || ch > '9'; w *= ch == '-' ? -1 : 1, ch = getchar());
    for (; ch >= '0' && ch <= '9'; s = s * 10 + ch - '0', ch = getchar());
    return s * w;
}

bool chck(double x, bool flag = 0){//滑动窗口判答案
    dq = __;
    for (int i = 1; i <= n; ++i) b[i] = a[i] - x * i;
    for (int i = l; i <= n; ++i){//枚举右端点
        while (dq.size() && b[dq.back()] > b[i - l + 1]) dq.pop_back();
        while (dq.size() && i - dq.front() + 1 > r) dq.pop_front();
        dq.push_back(i - l + 1), flag |= (b[i] - b[dq.front()]) >= x * m;//使b[左端点]最小
    }
    return flag;
}

double work(double res = 0){//滑动窗口存最值
    maxn = minn = __;
    for (int i = 1; i <= n; ++i){
        while (maxn.size() && a[maxn.back()] < a[i]) maxn.pop_back();
        while (maxn.size() && i - maxn.front() + 1 > l) maxn.pop_front();
        while (minn.size() && a[minn.back()] > a[i]) minn.pop_back();
        while (minn.size() && i - minn.front() + 1 > l) minn.pop_front();
        maxn.push_back(i), minn.push_back(i);
        res = max(res, (a[maxn.front()] - a[minn.front()]) * 1.0 / (l + m - 1));
    }
    return res;
}

double calc(double L = work(), double R = 2e9, double res = 0){//二分答案
    for (res = L; R - L > 1e-7; ){
        double mid = (L + R) / 2;
        if (chck(mid)) L = res = mid;
        else R = mid;
    }
    return res;
}

signed main(){
    for (int _ = read(); _--; ){
        n = read(), m = read(), l = read(), r = read();
        for (int i = 1; i <= n; ++i) a[i] = read();
        ans = calc(), reverse(a + 1, a + n + 1), ans = max(ans, calc());
        printf("%.4lf\n", ans);
    }
    return 0;
}
```