27. 分治

27.1 CDQ分治

27.1.1 三维偏序

题意

给定编号为$1 \sim n$的$n\ \ (1 \leq n \leq 1e5)$个元素,其中第i个元素有a_i, b_i, c_i三种属性.设$f(i)$满足如下四个条件:①$a_j \leq a_i$;②$b_j \leq b_i$;③$c_j \leq c_i$;④$j \neq i$的j的数量.对每个$d \in [0, n)$,求满足$f(i) = d$的i的数量.

输入第一行包含两个整数n, m,表示元素数量和最大属性值.接下来n行每行输入三个整数$a_i, b_i, c_i\ \ (1 \leq a_i, b_i, c_i \leq m \leq 2e5)$,表示第$i$个元素的三个属性值.

输出n行,每行输出一个整数,其中第$(d+1)$行的整数表示满足$f(i) = d$的i的数量.

思路

考虑一维版本,即对每个i,求满足$a_j \leq a_i\ \ (j \neq i)$的$j$的数量,只需将所有元素升序排列,设下标从1开始,则$\leq a_i$的元素都在a_i的左边,共$(i-1)$个元素.

考虑二维版本,以第一维为第一关键字、第二维为第二关键字升序排序.从前往后扫描,则满足$a_j \leq a_i\ \ (j \neq i)$的$a_j$都在$a_i$的左边,只需在其中找出满足$b_j \leq b_i$的$j$的数量,这可用BIT实现:先将$b_i$离散化为$1 \sim n$,求有多少个$b_j \leq b_i\ \ (j \neq i)$即求$b_i$的前缀和,求完之后再将第$i$个元素插入BIT,即$b_i++$.也可用分治实现:类似于归并排序求逆序对.将所有的(i, j)对分为三类:①i, j都在序列中点的左边,只需递归左边;②i, j都在序列中点的右边,只需递归右边;③i, j在序列中点的左右,因用双关键字排序,则只能是j在左,i在右,问题转化为从左边取一个j,从右边取一个i,求满足$b_j \leq b_i\ \ (j \neq i)$的$j$的数量.因归并排序过程中可顺带将左右两边按$b_j$升序排列,则可用双指针求解.对每个指针$i$,将指针$j$右移到第一个$> b_i$的位置,则满足$b_j \leq b_i\ \ (j \neq i)$的$j$的数量即$1 \sim (j-1)$.

考虑三维版本,先按三关键字升序排序.归并排序过程中可顺带将左右两边按第二维升序排序,同上用双指针求出满足$b_j \leq b_i\ \ (j \neq i)$的$j$的数量,即$1 \sim (j-1)$,再在其中用BIT求出满足$c_j \leq c_i\ \ (j \neq i)$的$j$的数量.每次归并将区间平分,共$O(\log n)$层.第一层的时间复杂度即BIT的时间复杂度$O(n \log n)$,第二层的时间复杂度$O\left(2 \cdot \dfrac{n}{2} \log \dfrac{n}{2}\right) \leq O(n \log n)$,第三层时间复杂度$O\left(4 \cdot \dfrac{n}{4} \log \dfrac{n}{4}\right) \leq O(n \log n), \cdots$,总时间复杂度$O(n \log^2 n)$,

因可能出现三维都相等的元素,需先去重,用一个数组记录这样的元素出现的次数.对元素相异的序列用上述方法求出数量后,对同类的元素更新答案.设有k个元素与第i个元素相同,则其答案加上$(k-1)$.

代码

```
const int MAXN = 1e5 + 5, MAXM = 2e5 + 10;
int n, m;  // 元素个数、最大属性值
struct Data {
  int a, b, c;
  int cnt;  // 三维都相等的元素的个数
  int res;  // 与该元素配对的元素的个数

  bool operator<(const Data& t)const {
    if (a != t.a) return a < t.a;
    else if (b != t.b) return b < t.b;
    else return c < t.c;
  }

  bool operator==(const Data& t)const { return a == t.a && b == t.b && c == t.c; }
```

```
}ques[MAXN],tmp[MAXN];  // 原数组、归并排序的临时数组
int BIT[MAXM];  // 树状数组
int ans[MAXN];

void add(int x, int v) {  // 树状数组插入元素
  for (int i = x; i < MAXM; i += lowbit(i)) BIT[i] += v;
}

int query(int x) {  // 树状数组求前缀和
  int res = 0;
  for (int i = x; i; i -= lowbit(i)) res += BIT[i];
  return res;
}

void merge_sort(int l, int r) {
  if (l >= r) return;

  int mid = l + r >> 1;
  merge_sort(l, mid), merge_sort(mid + 1, r);

  int i = l, j = mid + 1, k = 0;  // 此处的i,j与思路中的i,j相反
  while (i <= mid && j <= r) {
    if (ques[i].b <= ques[j].b) {
      add(ques[i].c, ques[i].cnt);  // 插入树状数组
      tmp[k++] = ques[i++];  // 存入临时数组
    }
    else {  // j已移动到第一个满足b_j>b_i的下标
      ques[j].res += query(ques[j].c);  // 答案加上前缀和
      tmp[k++] = ques[j++];  // 存入临时数组
    }
  }

  // 循环完剩下的部分
  while (i <= mid) add(ques[i].c, ques[i].cnt), tmp[k++] = ques[i++];
  while (j <= r) ques[j].res += query(ques[j].c), tmp[k++] = ques[j++];

  for (int i = l; i <= mid; i++) add(ques[i].c, -ques[i].cnt);  // 清空BIT

  for (int i = l, j = 0; j < k; i++, j++) ques[i] = tmp[j];  // 拷回原数组
}

int main() {
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    int a, b, c; cin >> a >> b >> c;
    ques[i] = { a,b,c,1 };  // 出现次数为1
  }
  sort(ques, ques + n);  // 按三关键字排序

  // 去重,并统计有重复的元素的个数
  int k = 1;
  for (int i = 1; i <= n; i++) {
    if (ques[i] == ques[k - 1]) ques[k - 1].cnt++;
    else ques[k++] = ques[i];
  }

  merge_sort(0, k - 1);  // 注意不是n-1

```

```
  for (int i = 0; i < k; i++) ans[ques[i].res + ques[i].cnt - 1] += ques[i].cnt;

  for (int i = 0; i < n; i++) cout << ans[i] << endl;
}
```

27.1.2 老C的任务

题意

二维平面上有$n\ \ (1\le n\le 1e5)$个基站$(x_i,y_i)\ \ (-2^{31}\le x_i,y_i<2^{31})$,每个基站有一个功率$p_i\ \ (-2^{31}\le p_i<2^{31})$,任意两基站坐标不同,且$x_i,y_i,p_i\in\mathbb{Z}$.现有$m\ \ (1\le m\le 1e5)$个询问,每个询问输入四个整数$x_{1j},y_{1j},x_{2j},y_{2j}$,表示询问以$(x_{1j},y_{1j})$、$(x_{2j},y_{2j})\ \ (-2^{31}\le x_{1j},y_{1j},x_{2j},y_{2j}<2^{31},x_{1j}\le x_{2j},y_{1j}\le y_{2j})$为对角线的矩形区域中(含边界)所有基站的功率之和.

思路

用二维前缀和,转化为对每个点,求其左下方的基站的功率之和.注意到对查询点(x_i,y_i),其左下方的点(x_j,y_j)满足$x_j\le x_i,y_j\le y_i$.因要区分查询点和基站,不妨每个点添加一个维度z,其中查询点$z_i=1$,基站点$z_j=1$.问题转化为:对每个i,求使得$x_j\le x_i,y_j\le y_i,z_j<z_i$的所有基站$j$的功率之和,用CDQ分治实现.因第三维只有两种取值,则无需用BIT维护,只需对每个$z_i=1$,求$z_j=0$的前缀和,用一个变量sum维护即可.

因$z_j<z_i$,则$z_j=z_i$的点都不满足,故无需判相等.

基站最多有$1e5$个点,每个询问输入四个点,故点的数组要开$5e5$.功率和可能爆int,要开ll.

代码

```
const int MAXN = 5e5 + 5;
int n, m;  // 基站数、询问数
struct Data {
  int x, y;  // 坐标
  int z;  // 查询点为1,基站点为0
  int p;  // 功率
  int id;  // 询问的编号
  int sgn;  // 符号,表示在二维前缀和中该点是加还是减
  ll sum;  // 基站点的功率前缀和

  bool operator<(const Data& t)const {
    if (x != t.x) return x < t.x;
    else if (y != t.y) return y < t.y;
    else return z < t.z;
  }
}ques[MAXN], tmp[MAXN];  // 基站点+查询点、归并排序的临时数组
ll ans[MAXN];

void merge_sort(int l, int r) {
  if (l >= r)return;

  int mid = l + r >> 1;
  merge_sort(l, mid), merge_sort(mid + 1, r);

  int i = l, j = mid + 1, k = 0;
  ll sum = 0;  // 基站点的功率前缀和
```

```
  while (i <= mid && j <= r) {
    if (ques[i].y <= ques[j].y) {
      sum += !ques[i].z * ques[i].p;  // 基站点才需要加功率
      tmp[k++] = ques[i++];  // 存入临时数组
    }
    else {  // j指向第一个满足ques[i].y>ques[j].y的下标
      ques[j].sum += sum;  // 查询点的功率为0,无需特判
      tmp[k++] = ques[j++];  // 存入临时数组
    }
  }

  // 循环完剩下的部分
  while (i <= mid) sum += !ques[i].z * ques[i].p, tmp[k++] = ques[i++];
  while (j <= r) ques[j].sum += sum, tmp[k++] = ques[j++];

  for (int i = l, j = 0; j < k; i++, j++) ques[i] = tmp[j];  // 拷回原数组
}

int main() {
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    int x, y, p; cin >> x >> y >> p;
    ques[i] = { x,y,0,p };  // 基站点
  }

  int k = n;  // 查询点下标从n开始(数组下标从0开始)
  for (int i = 1; i <= m; i++) {
    int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
    ques[k++] = { x2,y2,1,0,i,1 };
    ques[k++] = { x1 - 1,y2,1,0,i,-1 };
    ques[k++] = { x2,y1 - 1,1,0,i,-1 };
    ques[k++] = { x1 - 1,y1 - 1,1,0,i,1 };
  }

  sort(ques, ques + k);  // 三关键字排序

  merge_sort(0, k - 1);

  for (int i = 0; i < k; i++)
    if (ques[i].z) ans[ques[i].id] += ques[i].sum * ques[i].sgn;  // 查询点更新答案

  for (int i = 1; i <= m; i++) cout << ans[i] << endl;
}
```

27.1.3 动态逆序对

题意

给定$1\sim n\ \ (1\leq n\leq 1e5)$的一个排列,按某种顺序依次删除$m\ \ (1\leq m\leq 5e4)$个元素,在每次删除元素前统计整个序列中逆序对的对数.

输入第一行包含两个整数n,m,表示初始元素的个数和删除元素的个数.接下来n行包含一个$1\sim n$间的正整数,表示初始排列.接下来m行每行包含一个正整数,表示每次删除的元素.

输出m行,表示删除每个元素前序列的逆序对的对数.

思路

考虑如何构造三维数对.第一维p_i表示元素i的下标,第二维a_i表示元素i的值,第三维t_i表示元素i被删除的时间.因有些数不会被删除,则时间戳可倒序分配,即第一个被删除的数的时间戳为n,第二个被删除的数的时间戳为$(n-1),\cdots$,第m个被删除的数的时间戳为$(n-m+1)$,剩下的$1\sim(n-m)$随机分配,这样方便在BIT中求前缀和.显然本题中任意两元素的任意维都不相等,故无需处理相等的情况.

按照元素被删除的时间统计答案.对每个时间t,在其前面的数都已被删去.对每个时间戳t_i,要求与在t_i时删除的元素构成逆序对的的数的个数(只考虑t_i时还存在的元素).为保证统计答案时不重复,可规定每对逆序对的答案累加到先被删除的(时间戳大的)数上.

对时间戳求一个$s[]$数组,其中$s[i]$表示第i个删除的元素及其还未被删除的元素构成的序列中逆序对的对数,则对每个时间戳t_i,只需求$s[]$数组$1\sim t_i$的前缀和即为答案.

因每对逆序对(x,y)满足$p_x<p_y$且$a_x>a_y$,而删除的元素可能是逆序对的first或second,则对每个j,需分别求满足①$t_i<t_j,p_i<p_j,a_i>a_j$的i的个数;②$t_i<t_j,p_j<p_i,a_j>a_i$的i的个数,两者的答案都累加到第j个元素上.对归并排序的左右区间,①的情况,因$p_i<p_j$,则i在左,j在右.在所有满足$a_i>a_j$的i中求满足$t_i<t_j$的数的数量,则双指针从右往左走;②的情况,j在左,i在右,双指针从左往右走.

注意到输入时已按下标排序,则也无需再进行三关键字排序,也无需存p_i.

代码

```
const int MAXN = 1e5 + 5;
int n, m;  // 元素个数、删除个数
struct Data {
  int a, t;  // 元素值、时间戳
  int res;  // 与该元素配对的元素的个数
}ques[MAXN], tmp[MAXN];  // 原数组、归并排序的临时数组
int BIT[MAXN];  // 树状数组
int pos[MAXN];  // 记录要删除的数的值对应的下标
ll ans[MAXN];

void add(int x, int v) {  // 元素插入树状数组
  for (int i = x; i < MAXN; i += lowbit(i)) BIT[i] += v;
}

int query(int x) {  // 树状数组求前缀和
  int res = 0;
  for (int i = x; i; i -= lowbit(i)) res += BIT[i];
  return res;
}

void merge_sort(int l, int r) {
  if (l >= r) return;

  int mid = l + r >> 1;
  merge_sort(l, mid), merge_sort(mid + 1, r);

  // 情况一,双指针从右往左走
  int i = mid, j = r;
  while (i >= l && j > mid) {
    if (ques[i].a > ques[j].a) add(ques[i].t, 1), i--;
    else ques[j].res += query(ques[j].t - 1), j--;
  }
  while (j > mid) ques[j].res += query(ques[j].t - 1), j--;  // 循环完剩下的
```

```cpp
  for (int k = i + 1; k <= mid; k++) add(ques[k].t, -1);  // 清空BIT

  // 情况二,双指针从左往右走
  j = l, i = mid + 1;
  while (j <= mid && i <= r) {
    if (ques[i].a < ques[j].a) add(ques[i].t, 1), i++;
    else ques[j].res += query(ques[j].t - 1), j++;
  }
  while (j <= mid) ques[j].res += query(ques[j].t - 1), j++;  /// 循环完剩下的
  for (int k = mid + 1; k < i; k++) add(ques[k].t, -1);  // 清空BIT

  // 归并排序(可用sort代替)
  i = l, j = mid + 1;
  int k = 0;
  while (i <= mid && j <= r) {
    if (ques[i].a <= ques[j].a) tmp[k++] = ques[i++];
    else tmp[k++] = ques[j++];
  }
  while (i <= mid) tmp[k++] = ques[i++];
  while (j <= r) tmp[k++] = ques[j++];
  for (i = l, j = 0; j < k; i++, j++) ques[i] = tmp[j];
}

int main() {
  cin >> n >> m;
  for (int i = 0; i < n; i++) {
    cin >> ques[i].a;
    pos[ques[i].a] = i;  // 记录元素值对应的下标
  }

  for (int i = 0, tstamp = n; i < m; i++) {  // 倒序分配时间戳
    int a; cin >> a;
    ques[pos[a]].t = tstamp--;  // 分配时间戳
    pos[a] = -1;  // 记录被删除
  }

  for (int i = 1, tstamp = n - m; i <= n; i++)  // 没被删除的元素分配时间戳,注意i从1开始枚举
    if (~pos[i]) ques[pos[i]].t = tstamp--;

  merge_sort(0, n - 1);

  for (int i = 0; i < n; i++) ans[ques[i].t] = ques[i].res;  // 记录答案

  for (int i = 2; i <= n; i++) ans[i] += ans[i - 1];  // 求前缀和

  for (int i = 0, j = n; i < m; i++, j--) cout << ans[j] << endl;  // 输出前j个答案
}
```

27.1.4 Petya and Array

原题指路:https://codeforces.com/contest/1042/problem/D

题意 (2 s)

给定一个长度为$n\ \ (1\leq n\leq 2\mathrm{e}5)$的序列$a_1,\cdots,a_n\ \ (|a_i|\leq 1\mathrm{e}9)$和一个整数$t\ \ (|t|\leq 2\mathrm{e}14)$,问有多少对$(l,r)\ \ (1\leq l\leq r\leq n)\ s.t.\ a_l+\cdots+a_r<t$.

思路

考察$a[]$的前缀和$pre[]$.分治,递归处理左右两半区间,分别将两区间内的$pre[]$非降序排列后,用双指针求总区间对答案的贡献.

代码

```
const int MAXN = 2e5 + 5;
int n;
ll t;
ll pre[MAXN];
ll ans;

void cdq(int l, int r) {  // [l, r]
  if (l == r) return;

  int mid = l + r >> 1;
  cdq(l, mid), cdq(mid + 1, r);

  int i = l, j = mid;
  for (int k = mid + 1; k <= r; k++) {
    while (i <= mid && pre[k] >= pre[i] + t) i++;
    ans += j - i + 1;
  }

  sort(pre + l, pre + r + 1);
}

void solve() {
  cin >> n >> t;
  for (int i = 1; i <= n; i++) {
    int a; cin >> a;
    pre[i] = pre[i - 1] + a;
  }

  cdq(0, n);
  cout << ans;
}

int main() {
  solve();
}
```

27.2 分治

27.2.1 平面最近点对

原题指路: https://www.luogu.com.cn/problem/P1257

题意

给定平面上$n\ \ (2\leq n\leq 1\mathrm{e}4)$个整点的坐标$\ \ (0\leq x,y\leq 1\mathrm{e}9)$, 求平面最近点对间的距离, 四舍五入保留4位小数.

思路

$O(n^2)$暴力枚举每对点, 更新最短距离.

代码

```
typedef pair<int, int> pii;
#define x first
#define y second

void solve() {
  int n; cin >> n;
  vector<pii> points(n);
  for (auto& [x, y] : points) cin >> x >> y;

  double ans = INF;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      ans = min(ans,
                hypot(points[i].x - points[j].x, points[i].y - points[j].y));
    }
  }
  cout << fixed << setprecision(4) << ans << endl;
}

int main() {
  solve();
}
```

27.2.2 平面最近点对(加强版)

原题指路: https://www.luogu.com.cn/problem/P1429

题意

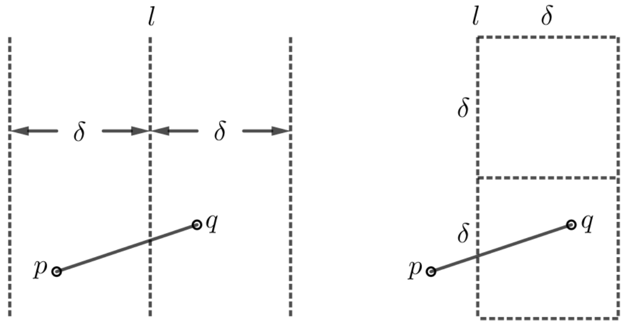
给定平面上$n\ \ (2\leq n\leq 2\mathrm{e}5)$个整点的坐标$\ \ (0\leq x,y\leq 1\mathrm{e}9)$, 求平面最近点对间的距离, 四舍五入保留4位小数.

思路I

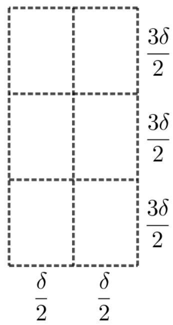
以x坐标为第一关键字、y坐标为第二关键字, 将平面上的点非降序排列后, 按下标分为左右两个集合. 考虑分治, 递归求出左右两个集合的平面最近点对间的距离dis_l和dis_r后, 考虑合并. 最优解的两点都在左边的集合或都在右边集合的情况是平凡的, 下面讨论最优解的两点一个在左边的集合、一个在右边的集合的情况.

设$dis=\min\{dis_l,dis_r\}$, 点按上述方法排序后中间点的x坐标为mid, 考察区域$[mid-dis,mid+dis]$, 则若存在距离$<dis$的两点, 则必在该区域中, 进而枚举该区域的左半边区域$[mid-dis,mid]$中的点, 再枚举右半边区域$[mid,mid+dis]$中的点即可求出平面最近点对. 可能的最坏情况: 区域$[mid-dis,mid+dis]$包含所有点, 时间复杂度$O(n^2)$.

Preparata和Shamos分析了最优解的两点在区域$[mid-\delta, mid+\delta]$中的情况, 下面证明: 若(p,q)是平面最近点对, 且$p \in [mid-\delta, mid], q \in [mid, mid+\delta]$, 则$q$在如下图所示的$\delta \times (2\delta)$的矩形中, 且该矩形至多包含$[mid, mid+\delta]$中的6个点, 且每个点对间的距离$\geq \delta$.



[证] 如下图, 将$\delta \times (2\delta)$的长方形划分为6个$\left(\dfrac{\delta}{2}\right) \times \left(\dfrac{3\delta}{2}\right)$的小矩形.



若该长方形包含$[mid, mid+\delta]$中的至少7个点, 由抽屉原理: $\exists$一个矩形至少包含$[mid, mid+\delta]$中的2个点.

注意到一个小矩形中两点间最短距离为对角线的长度, 即$\sqrt{\left(\dfrac{\delta}{2}\right)^2+\left(\dfrac{3\delta}{2}\right)^2}=\dfrac{5\delta}{6}<\delta$, 矛盾.

故合并的时间复杂度为$O(n)$. 设总时间复杂度为$T(n)$, 则$T(n)=\begin{cases}1, n \leq 2 \\ 2T\left(\dfrac{n}{2}\right)+O(n), n>2\end{cases}$. 由主定理: $T(n)=O(n^{\log_2 n}\log^{0+1} n)=O(n\log n)$.

代码I

```cpp
typedef pair<int, int> pii;
#define x first
#define y second

vector<pii> points;

double getDistance(int i, int j) {  // dis(points[i], points[j])
    return hypot(points[i].x - points[j].x, points[i].y - points[j].y);
}

double divide(int l, int r) {
    if (l == r) return INF;  // 只包含一个点
    else if (l + 1 == r) return getDistance(l, r);  // 只包含2个点

    int mid = l + r >> 1;
    double dis1 = divide(l, mid), dis2 = divide(mid + 1, r);

    // 合并
    double dis = min(dis1, dis2);
    vector<int> tmp;  // 在区域[points[mid].x - dis, points[mid].x + dis]中的点
```

```
    for (int i = l; i <= r; i++) {
        if (fabs(points[i].x - points[mid].x) <= dis)
            tmp.push_back(i);
    }
    sort(all(tmp), [&](const int& i, const int& j) {
        return points[i].y < points[j].y;
    });

    int siz = tmp.size();  // siz <= 6
    for (int i = 0; i < siz; i++) {
        for (int j = i + 1; j < siz && points[tmp[j]].y - points[tmp[i]].y < dis; j++)
            dis = min(dis, getDistance(tmp[j], tmp[i]));
    }
    return dis;
}

void solve() {
    int n; cin >> n;
    points.resize(n);
    for (auto& [x, y] : points) cin >> x >> y;
    sort(all(points), [&](const pii& A, const pii& B) {
        return A.x != B.x ? A.x < B.x : A.y < B.y;
    });

    cout << fixed << setprecision(4) << divide(0, n - 1) << endl;
}

int main() {
    solve();
}
```

思路II

类似于统计序列的思想, 对每个点, 将它和它左边的所有元素的贡献加入答案中. 具体地, 将所有点以x坐标为第一关键字、以y坐标为第二关键字非降序排列后, 将点逐个加入集合中. 具体地, 维护一个以y坐标为第一关键字、以x坐标为第二关键字的multiset和当前的最优解ans.

对每个点i, 做如下操作:

①因集合以y为第一关键字, 则集合中满足$x_i - x_j \geq dis$的点j显然不是最优解, 删除即可.

②对集合中满足$|x_i - x_j| < dis$的点j, 暴力更新答案.

③将点i加入集合中.

因每个点至多被插入和删除一次, 则插入和删除点的时间复杂度为$O(n\log n)$. 更新答案的部分类似于**思路I**中分治合并的过程, 可以证明集合中只有≤ 6个点j满足$|x_i - x_j| < dis$, 故更新答案的时间复杂度$O(n)$. 总时间复杂度$O(n\log n)$.

代码II

```
typedef pair<int, int> pii;
#define x first
#define y second

vector<pii> points;

```

```cpp
struct cmp {
    bool operator()(const pii& A, const pii& B)const {
        return A.y < B.y;
    }
};

double getDistance(const pii& A, const pii& B) {  // dis(A, B)
    return hypot(A.x - B.x, A.y - B.y);
}

void solve() {
    int n; cin >> n;
    points.resize(n);
    for (auto& [x, y] : points) cin >> x >> y;
    sort(all(points), [&](const pii& A, const pii& B) {
        return A.x != B.x ? A.x < B.x : A.y < B.y;
    });

    double ans = INF;
    multiset<pii, cmp> s;
    for (int i = 0, j = 0; i < n; i++) {
        // 删除集合中x_i - x_j >= ans的点j
        while (j < i && points[i].x - points[j].x >= ans)
            s.erase(s.find(points[j++]));

        // 暴力更新集合中满足|x_i - x_j| < ans的点j的答案
        for (auto it = s.lower_bound(pii(points[i].x, points[i].y - ans));
            it != s.end() && it->y - points[i].y < ans; it++) {
                ans = min(ans, getDistance(*it, points[i]));
            }

        s.insert(points[i]);
    }
    cout << fixed << setprecision(4) << ans << endl;
}

int main() {
    solve();
}
```

思路III

算法过程:

(1)循环下面过程直至删完所有点:

①随机选一个点, 求它到其他所有点的最短距离d.

②将所有点划分到$l=\left\lfloor\dfrac{d}{3}\right\rfloor$的网格中, 如$\left(\left\lfloor\dfrac{x}{l}\right\rfloor,\left\lfloor\dfrac{y}{l}\right\rfloor\right)$.

③删除九宫格内的孤立点, 此时所有点间的最短距离$\geq\dfrac{2\sqrt{2}}{3}d$, 其中$\dfrac{2\sqrt{2}}{3}<1$.

(2)取最后一个d, 将所有点划分到$\left(\left\lfloor\dfrac{x}{d}\right\rfloor,\left\lfloor\dfrac{y}{d}\right\rfloor\right)$的网格中, 暴力求九宫格中的答案.

时间复杂度:

(1)第一部分，每次期望删除至少一半的点, 因为有$\geq \frac{1}{2}$的概率取到一个最近点距离$<$中位数的数，故第一部分的时间复杂度为$O(n)$. 第二部分的复杂度类似于分治的分析, 可以证明对每个点, 只需考察常数个点, 故第二部分的时间复杂度为$O(n)$. 故总时间复杂度为$O(n)$.

(2)严谨证明见论文"A Simple Randomized Sieve Algorithm for the Closet-Pair Problem": https://www.cs.umd.edu/~samir/grant/cp.pdf

(3)下面的实现中用哈希来记录各点, 时间复杂度$O((9n+6n)\log n)=O(n\log n)$, 常数较大.

Hytidel 03-20 13:23:04	Accepted 100	P1429 平面最近点对（加强版）	期望线性	2.09s / 16.72MB / 2.70KB C++17
Hytidel 03-17 07:38:03	Accepted 100	P1429 平面最近点对（加强版）	multiset	627ms / 1.92MB / 1.45KB C++17
Hytidel 03-16 22:56:08	Accepted 100	P1429 平面最近点对（加强版）	分治	1.31s / 1.93MB / 1.67KB C++17

代码III

```cpp
typedef pair<int, int> pii;
#define x first
#define y second

mt19937 rnd(time(0));
const ll INFF = 0x3f3f3f3f3f3f3f3f;
const double eps = 1e-8;
const int d[] = { 0, -1, 1 };  // 偏移量

vector<pii> points;
unordered_map<ll, int> cnt;  // 哈希值相同的点数
unordered_map<ll, vector<pii>> mp;  // 哈希值对应的点

double getDistance(const pii& A, const pii& B) {
    return ((ll)A.x - B.x) * (A.x - B.x) + ((ll)A.y - B.y) * (A.y - B.y);
}

ll getHash(ll x, ll y) {
    return x << 30 ^ y;
}

bool check(ll x, ll y) {
    if (cnt[getHash(x, y)] > 1) return true;
    for (int i = 0; i < 3; i++) {  // 枚举x坐标的偏移量
        for (int j = 0; j < 3; j++)  // 枚举y坐标的偏移量
            if ((i || j) && cnt.count(getHash(x + d[i], y + d[j]))) return true;
    }
    return false;
}

double cal(pii u, ll x, ll y) {
    double res = INFF;
    for (int i = 0; i < 2; i++) {  // 枚举x坐标的偏移量, 注意i只能取0, 1
        for (int j = 0; j < 3; j++) {  // 枚举y坐标的偏移量
            ll ha = getHash(x + d[i], y + d[j]);
            for (auto v : mp[ha])
                if (u != v) res = min(res, getDistance(u, v));
        }
    }
    return res;
```

```cpp
}

void solve() {
    int n; cin >> n;
    points.resize(n);
    for (auto& [x, y] : points) cin >> x >> y;
    shuffle(all(points), rnd);

    vector<pii> a = points;
    double d;  // 最短距离的平方
    while (a.size() > 1) {
        d = INFF;
        for (int i = 1; i < a.size(); i++)
            d = min(d, getDistance(a[i], a[0]));
        if (d < eps) break;

        double l = sqrt(d) / 3;
        for (auto [x, y] : a)  // 将a[]中的点加入哈希表
            cnt[getHash(x / l, y / l)]++;

        vector<pii> tmp;  // 网格中的非孤立点
        for (auto [x, y] : a)
            if (check(x / l, y / l)) tmp.push_back({ x, y });

        a = tmp;
        cnt.clear();
    }

    double dd = sqrt(d), ans = d;
    if (ans > eps) {
        for (auto [x, y] : points)
            mp[getHash(x / dd, y / dd)].push_back({ x, y });

        for (auto [x, y] : points)
            ans = min(ans, cal({ x, y }, x / dd, y / dd));
    }
    cout << fixed << setprecision(4) << sqrt(ans) << endl;
}

int main() {
    solve();
}
```

27.2.3 平面最近点对(二次加强版)

原题指路: https://www.luogu.com.cn/problem/P7883

题意 $(350\ \mathrm{ms})$

给定平面上n $(2\le n\le 4\mathrm{e}5)$个整点的坐标 $(0\le |x|,|y|\le 1\mathrm{e}7)$, 求平面最近点对间的距离, 输出距离的平方.

思路

将所有点绕原点随机旋转同一角度后按$x \cdot y$非降序排列. 可以证明: 随机旋转后, 最优解的两点在数组中相距不远. 具体地, 只需取每个点之前的50个点更新答案.

代码

```
mt19937 rnd(time(0));
const int r = rnd();
const double sinr = sin(r), cosr = cos(r);

struct Point  {
    int x, y;
    double xx, yy;

    void rotate() {
        xx = x * sinr - y * cosr;
        yy = x * sinr + y * cosr;
    }
};
vector<Point> points;

ll getDistance(int i, int j) {
    return ((ll)points[i].x - points[j].x) * (points[i].x - points[j].x)
        + ((ll)points[i].y - points[j].y) * (points[i].y - points[j].y);
}

void solve() {
    int n; cin >> n;
    points.resize(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> points[i].x >> points[i].y;
        points[i].rotate();
    }
    sort(points.begin() + 1, points.end(), [&](const Point& A, const Point& B) {
        return A.xx * A.yy < B.xx * B.yy;
    });

    ll ans = 0x3f3f3f3f3f3f3f3f;
    for (int i = 1; i <= n; i++) {
        for (int j = max(i - 50, 1); j <= i - 1; j++)
            ans = min(ans, getDistance(i, j));
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

27.2.4 Tricky Function

原题指路: https://codeforces.com/problemset/problem/429/D

题意 (2 s)

给定一个长度为$n\ \ (2\leq n\leq 1e5)$的序列$a=[a_1,\cdots,a_n]\ \ (|a_i|\leq 1e4)$. 定义函数$f(i,j)=(i-j)^2+g(i,j)^2\ \ (1\leq i,j\leq n)$, 其中$g(i,j)$的计算方式如下:

```
int g(int i, int j) {
    int sum = 0;
    for (int k = min(i, j) + 1; k <= max(i, j); k++)
        sum += a[k];
    return sum;
}
```

求$\min\limits_{i\neq j} f(i,j)$.

思路

设$a[]$的前缀和数组为$pre[]$, 则$g(i,j)=pre_j-pre_i$, 进而$f(i,j)=(i-j)^2+(pre_i-pre_j)^2$.

注意到上式的RHS形如平面上两点$(i,pre_i),(j,pre_j)$间距离的平方, 问题转化为求平面最近点对间距离的平方. 因$n\leq 1e5$, 可用分治或随机化求解, 下面的实现以分治为例.

代码

```
typedef pair<int, int> pii;
#define x first
#define y second

vector<pii> points;

ll getDistance(int i, int j) {  // dis(points[i], points[j])
    return ((ll)points[i].x - points[j].x) * (points[i].x - points[j].x)
        + ((ll)points[i].y - points[j].y) * (points[i].y - points[j].y);
}

ll divide(int l, int r) {
    if (l == r) return INF;  // 只包含一个点
    else if (l + 1 == r) return getDistance(l, r);  // 只包含2个点

    int mid = l + r >> 1;
    ll dis1 = divide(l, mid), dis2 = divide(mid + 1, r);

    // 合并
    ll dis = min(dis1, dis2);
    vector<int> tmp;  // 在区域[points[mid].x - dis, points[mid].x + dis]中的点
    for (int i = l; i <= r; i++) {
        if (((ll)points[i].x - points[mid].x) * (points[i].x - points[mid].x) <= dis)
            tmp.push_back(i);
    }
    sort(all(tmp), [&](const int& i, const int& j) {
        return points[i].y < points[j].y;
    });

```

```
    int siz = tmp.size();  // tmp <= 6
    for (int i = 0; i < siz; i++) {
        for (int j = i + 1; j < siz &&
            ((ll)points[tmp[j]].y - points[tmp[i]].y) * (points[tmp[j]].y -
points[tmp[i]].y) < dis; j++) {
                dis = min(dis, getDistance(tmp[j], tmp[i]));
            }
    }
    return dis;
}

void solve() {
    int n; cin >> n;
    vector<int> pre(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> pre[i];
        pre[i] += pre[i - 1];
    }

    for (int i = 1; i <= n; i++)
        points.push_back({ i, pre[i] });

    sort(all(points), [&](const pii& A, const pii& B) {
        return A.x != B.x ? A.x < B.x : A.y < B.y;
    });
    cout << divide(0, n - 1) << endl;
}

int main() {
    solve();
}
```

27.2.5 Merge Sort

原题指路: https://codeforces.com/problemset/problem/873/D

题意 (2 s)

调用归并函数的递归函数 $mergeSort(l,r)$ 会对序列 $a[]$ 的区间 $[l,r)$ 作归并排序, 其过程如下:

① 若 $a[]$ 的区间 $[l,r)$ 已是非降序, 则返回.

② 设 $mid=\left\lfloor\dfrac{l+r}{2}\right\rfloor$.

③ 递归调用 $mergeSort(l,mid)$ 和 $mergeSort(mid,r)$.

④ 合并区间 $[l,mid)$ 和 区间 $[mid,r)$ 的有序序列.

给定两个整数 $n,k\ \ (1\leq n\leq 1\mathrm{e}5, 1\leq k\leq 2\mathrm{e}5)$, 构造任一 $1\sim n$ 的排列 $a=[a_0,\cdots,a_{n-1}]$, 使得对其调用 $mergeSort(0,n)$ 时, 恰调用 $mergeSort()$ 函数 k 次. 若无解, 输出 -1 .

思路

注意到每次调用 $mergeSort()$ 函数会产生 0 个或 2 个分支, 加上初始调用的一次, 总调用次数为奇数, 进而 k 为偶数时无解.

k 为奇数时, 考虑从升序排列的 $1 \sim n$ 构造答案 $ans = [ans_0, \cdots, ans_{n-1}]$. 先减去初始调用的一次, 并调用构造答案的分治函数 $divide(0, n)$, 其过程如下:

① 递归终止条件: 无剩余调用次数或当前区间只有一个元素.

② 若有剩余调用次数 m , 则 $m \geq 2$. 考虑递归到左右两边构造.

为保证每次分支对 m 的减量严格为 2 , 对当前区间 $[l, r)$, 设 $mid = \left\lfloor \dfrac{l + r}{2} \right\rfloor$,

交换 $ans[mid - 1]$ 与 $ans[mid]$, 此时区间 $[l, r)$ 无序, 但区间 $[l, mid)$ 和区间 $[mid, r)$ 都有序.

注意到 $divide(0, n)$ 与 $mergeSort()$ 可视为互逆操作, 则它们的调用次数相等, 进而递归结束后, 检查是否有剩余的递归次数, 若有则无解.

代码

```cpp
void solve() {
    int n, m; cin >> n >> m;  // 序列长度、调用次数

    if (m % 2 == 0) {
        cout << -1 << endl;
        return;
    }

    vector<int> ans(n);
    iota(all(ans), 1);

    function<void(int, int)> divide = [&](int l, int r) {
        if (!m) return;
        if (l + 1 >= r) return;  // 单个元素

        int mid = l + r >> 1;
        swap(ans[mid - 1], ans[mid]);
        m -= 2;
        divide(l, mid), divide(mid, r);
    };

    m--;  // 初始调用
    divide(0, n);

    if (m) cout << -1 << endl;  // 还剩余调用次数, 则无解
    else {
        for (int i = 0; i < n; i++)
            cout << ans[i] << " \n"[i == n - 1];
    }
}

int main() {
    solve();
}
```