《算法设计与分析-最近点对》实验报告

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/10/15　　得　分：

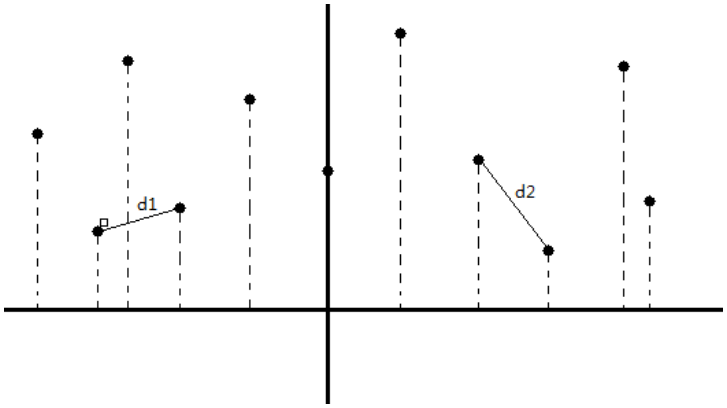
一、实验内容：

算法设计与分析课程作业——最近点对解题报告，题目选择洛谷 P1429。

二、所用算法的基本思想及复杂度分析：

1. 基本思想

点数非常多暴力求解复杂度为$O(n^2)$时间不足。采用分治法可以有效减少无效点对的查询，将大问题转换为各自的小问题求解后再合并。可以选择从处于问题中间的点画一条竖直线将平面划分为左右两个平面，从而问题拆解为以下两个步骤进行求解：



1. 求出端点同时处于左或右两个平面的最近点对
2. 求出端点分别处于左右两平面的最近点对

上两个步骤的取最优解当前问题即获得解决，划分的左右两个平面进一步划分并重复上述方法即实现了分治。下面考虑具体操作与优化：

1. 以何为标准选取中间位置的点？我们只考虑一维即x坐标，将所有点进行升序排序，选择处于序列中间位置的点即可
2. 第二步如何哪些点需要考虑？假设第一步求出来的最优解是tmp，那么第二步只需要看看是否存在距离比tmp还要小的点对。所以只需要将距离竖直线左右两侧x坐标小于tmp的点进行考虑。

3. 第二步选好点如何优化暴力？如果点非常密集第二步完全$O(n^2)$暴力求解效率也会降低。根据第二个问题进一步考虑，这些点对的y坐标同样距离不能大于tmp，所以我们可以将找到的点集进一步再按照y坐标升序排序，此时二重循环计算点对关系如果第二重循环中出现大于tmp则后面的点也不需要考虑。

2. 复杂度分析

复杂度不考虑$sqrt$函数和STL容器与函数的效率问题，则主要集中于分治法的实现复杂度，由于本身为$O(logn)$内部还进行了一次y坐标排序所以总的时间复杂度为$O(nlog^2\mathrm{n})$。

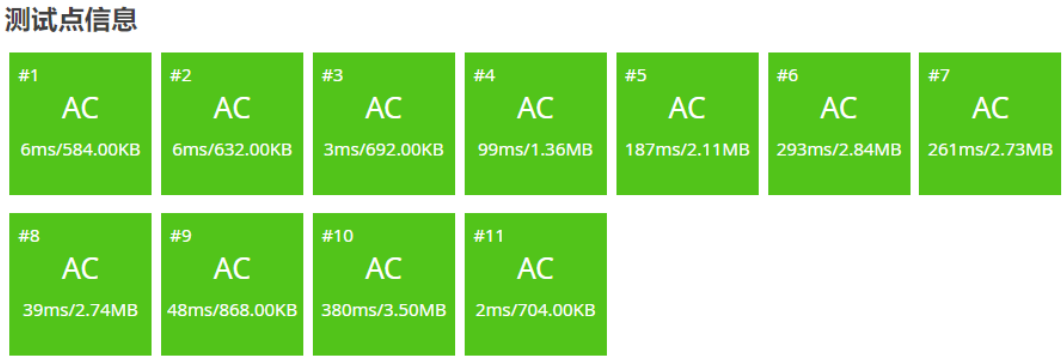
三、源程序及注释：

```
#include <bits/stdc++.h>
#pragma GCC optimize(2)
#pragma G++ optimize(2)
#define endl "\n"
#define fi first
#define se second
#define pb push_back
#define all(x) x.begin(), x.end()
#define rep(i, x, y) for (auto i = (x); i != (y + 1); ++i)
#define dep(i, x, y) for (auto i = (x); i != (y - 1); --i)
#ifdef LOCAL
#define de(...) cout << '[' << #__VA_ARGS__ << "] = " << __VA_ARGS__ <<
endl;
#else
#define de(...)
#endif
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int maxn = 2e5 + 5;

struct node {
    double x, y;
} p[maxn];

bool cmp1(node a, node b) {
    if (a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}
```

```cpp
bool cmp2(int a, int b) {
    return p[a].y < p[b].y;
}

double cal(node a, node b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

double dfs(int l, int r) {
    if (l == r) return INT_MAX;
    if (l + 1 == r) return cal(p[l], p[r]);
    int mid = l + r >> 1;
    double ans = min(dfs(l, mid), dfs(mid + 1, r));
    vector<int> vec;
    for (int i = l; i <= r; ++i)
        if (fabs(p[mid].x - p[i].x) < ans) vec.push_back(i);
    sort(all(vec), cmp2);
    for (int i = 0; i < vec.size(); ++i)
        for (int j = i + 1; j < vec.size() && p[vec[j]].y - p[vec[i]].y <
ans; ++j)
            ans = min(ans, cal(p[vec[i]], p[vec[j]]));
    return ans;
}

void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i)
        cin >> p[i].x >> p[i].y;
    sort(p + 1, p + 1 + n, cmp1);
    cout << fixed << setprecision(4) << dfs(1, n) << endl;
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
#ifdef LOCAL
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    clock_t start, end;
    start = clock();
#endif
    solve();
#ifdef LOCAL
```

```
    end = clock();
    cout << endl
         << "Runtime: " << (double)(end - start) / CLOCKS_PER_SEC <<
"s\n";
#endif
    return 0;
}
```

四、运行输出结果：



五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

问题一：关于复杂度进一步考虑，由于分治法内部的排序导致复杂度还是很高，可不可以优化掉。

解决措施：理论上可以，因为排序问题本身也可以采用分治法，所以可以在求解最近点对问题的同时进行y坐标排序。

获得教训：无。