[2016ACM多校] HDU5809 KD树 并查集

# 题意

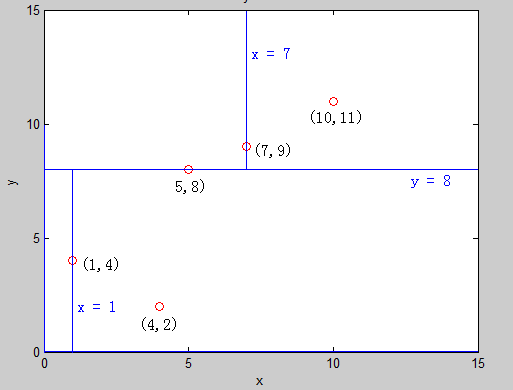
二维平面有N(<=10^5)个蚁穴，每个蚁穴的蚂蚁会不断的爬向最近的蚁穴(距离相同则按x,y坐标选最小的)，Q(<=10^5)个查询，问两只蚂蚁是否会相遇。

# 思路

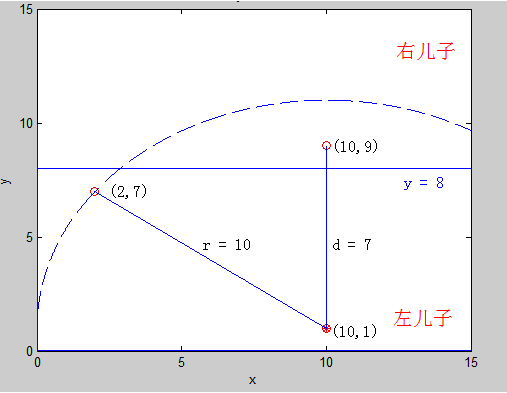
其实行走过程就像不断寻找、逼近最近点对，最终再最近点对循环，那么落入同一个最近点对的蚂蚁就会相遇。即把每个点和离他最近的点连边，不必考虑方向，同一个连通分量中的蚂蚁一定会相遇。建立图之后用并查集维护一下联通关系即可。

建图的难点就在于如何寻找最近点，有一个很好的数据结构KD树可以再O(NlogN)的时间建立，并完成每次平均O(logN)，最坏O(sqrt(N))的查询，而且还可以查第K大。

## KD树——K-Dimensional Tree



KD树首先通过不断划分空间的方法，规划一条logN级别的搜索路径来探测最近点。也就是说，对当前N个点，在某个维度选取中位点，然后将点集在这个维度以中位点坐标分成两部分，往目标点所在的那部分去查询。然后递归进入相应的那一半区间重复此操作。这样就可以在这条渐进路上得到一个最小值。



但显然有这样一种情况，跨过分界线的点更优。对于这种情况有个必要条件就是目标点到分解线的距离比已知的最小值要小。所以KD树是在回溯的时候在以目标点为圆心，当前最小值为半径做圆，如果和分界线相交那么在递归另外一边。也正是这个例外的存在会导致KD树处理询问会退化到O(根号N)。

怎样让跨界点的情况尽可能少呢？我们要在划分的时候选择尽可能稀疏的维度，来避免在靠近根的地方产生额外分支。所以划分是选择哪一维时，可以用取当前点集方差最大的维度。

有人写了一个比较通用完善的KD树500多行，ACM中对于低维度空间，只要用深度去摸来随机决定哪一位划分就很不错了。

# AC代码 C++

#include <stdio.h>

#include <algorithm>

using namespace std;

#define MAXD 2 //空间维度

#define MAXN 100005

#define EPS 0.5 //判断重点的阈值

struct KD\_P //空间点结构

{

int x[MAXD]; //坐标

int id; //初始编号

bool operator < (const KD\_P& b)

{

int i;

for (i = 0; i < MAXD; i++)

if (x[i] != b.x[i])

return x[i] < b.x[i];

return false;

}

};

KD\_P point[MAXN]; //原始点

KD\_P node[MAXN]; //KD树划分用排序节点

//int split[MAXN]; //这一点的划分参考维度,一般选择方差最大的一维,或奇偶划分

int split\_now; //当前排序的维度

bool KD\_cmp(const KD\_P a, const KD\_P b)

{

return a.x[split\_now] < b.x[split\_now];

}

void KD\_build(int l, int r, int d)

{

if (l >= r)

return;

int mid = l + r >> 1, dem, i;

//double ave, var, maxvar = -1.0;

//for (dem = 0; dem < MAXD; dem++) //求方差最大的维度

//{

// ave = var = 0.0;

// for (i = l; i <= r; i++)

// {

// ave += node[i].x[dem];

// var += node[i].x[dem] \* node[i].x[dem];

// }

// var -= ave \* ave / (r - l + 1);

// if (maxvar < 0 || var > maxvar)

// {

// maxvar = var;

// split\_now = dem;

// }

//}

//split[mid] = split\_now;

split\_now = d & 1;

nth\_element(node + l, node + mid, node + r + 1, KD\_cmp); //粗略排序找中位数,放好第mid大的点

KD\_build(l, mid - 1, d + 1);

KD\_build(mid + 1, r, d + 1);

}

KD\_P dest; //要询问最近点的点,本题中是KD中的点,所以要用eps判断是否是同一个点

long long ans; //当前查询的最近距离

int ansid; //最近点编号

long long get\_dis(int\* a, int\* b) //得到距离的平方

{

int i;

long long dis = 0;

for (i = 0; i < MAXD; i++)

dis += (long long)(a[i] - b[i]) \* (long long)(a[i] - b[i]);

return dis;

}

void KD\_query(int l, int r, int d)

{

if (l > r)

return;

int mid = l + r >> 1;

long long dis = get\_dis(node[mid].x, dest.x); //询问点到当前根的距离方

if (dis && (dis < ans || dis == ans && node[mid] < point[ansid]))

{

ans = dis;

ansid = node[mid].id;

}

split\_now = d & 1;

long long radius = (long long)(dest.x[split\_now] - node[mid].x[split\_now])\*(dest.x[split\_now] - node[mid].x[split\_now]); //询问点到分裂面的距离方

if (dest.x[split\_now] < node[mid].x[split\_now])

{

KD\_query(l, mid - 1, d + 1);

if (radius <= ans)

KD\_query(mid + 1, r, d + 1);

}

else

{

KD\_query(mid + 1, r, d + 1);

if (radius <= ans)

KD\_query(l, mid - 1, d + 1);

}

}

int pa[MAXN]; //并查集

int findpa(int n)

{

return pa[n] != n ? (pa[n] = findpa(pa[n])) : n;

}

int main()

{

int T, t, n, q, i, u, v;

scanf("%d", &T);

for (t = 1; t <= T; t++)

{

printf("Case #%d:\n", t);

scanf("%d%d", &n, &q);

for (i = 1; i <= n; i++)

{

scanf("%d%d", point[i].x, point[i].x + 1);

point[i].id = i;

node[i] = point[i];

}

KD\_build(1, n, 0);

for (i = 1; i <= n; i++)

pa[i] = i;

for (i = 1; i <= n; i++)

{

dest = point[i];

ans = (long long)1 << 62;

KD\_query(1, n, 0);

u = findpa(i);

v = findpa(ansid);

pa[u] = v;

}

while (q--)

{

scanf("%d%d", &u, &v);

puts(findpa(u) != findpa(v) ? "NO" : "YES");

}

}

return 0;

}

# 参考

http://blog.csdn.net/zhjchengfeng5/article/details/7855241