Code Jam 2009 - World Finals

Analysis: Min Perimeter

This problem is similar to the classical problem of finding the closest pair in a set of points. Algorithms that solve the closest-pair problem can be adapted to solve this one.

The number of points can be pretty large so we need an efficient algorithm. We can solve the problem in *O(n log n)* time using divide and conquer. We will split the set of points by a vertical line into two sets of equal size. There are now three cases for the minimum-perimeter triangle: its three points can either be entirely in the left set, entirely in the right set, or it can use points from each half.

We find the minimum perimeters for the left and right sets using recursion. Let the smallest of those perimeters be *p*. We can use *p* to make finding the minimum perimeter of the third case efficient, by only considering triangles that could have an area less than *p*.

To find the minimum perimeter for the third case (if it is less than *p*) we select the points that are within a distance of *p/2* from the vertical separation line. Then we iterate through those points from top to bottom, and maintain a list of all the points in a box of size *p x p/2*, with the bottom edge of the box at the most recently considered point. As we add each point to the box, we compute the perimeter of all triangles using that point and each pair of points in the box. (We could exclude triangles entirely to the left or right of the dividing line, since those have already been considered.)

We can prove that the number of points in the box is at most 16, so we only need to consider at most a small constant number of triangles for each point.

Splitting the current set of points by a vertical line requires the points to be sorted by *x* and going through the points vertically requires having the points sorted by *y*. If we do the *y* sort at each step that gives us an $O(n \log^2 n)$ algorithm, but we can keep the set of points twice, one array would have the points sorted by *x* and one would have the points sorted by *y*, and this way we have an *O(n log n)* algorithm.

The time limits were a bit tight and input limits were large because some $O(n^2)$ algorithms work really well on random cases. This is why during the contest some solutions that had the right idea but used a *p x p* box size or sorted by *y* at each step didn't manage to solve the large test cases fast enough.

You can read Tomek Czajka's source to get the details of a good implementation:

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <vector>
using namespace std;
#define REP(i,n) for(int i=0;i<(n);++i)
template<class T> inline int size(const T&c) { return c.size();}

const int BILLION = 1000000000;
const double INF = 1e20;
typedef long long LL;
```

```
struct Point {
  int x,y;
  Point() {}
  Point(int x,int y):x(x),y(y) {}
};

inline Point middle(const Point &a, const Point &b) {
  return Point((a.x+b.x)/2, (a.y+b.y)/2);
}

struct CmpX {
  inline bool operator()(const Point &a, const Point &b) {
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
  }
} cmpx;

struct CmpY {
  inline bool operator()(const Point &a, const Point &b) {
    if(a.y != b.y) return a.y < b.y;
    return a.x < b.x;
  }
} cmpy;

inline LL sqr(int x) { return LL(x) * LL(x); }

inline double dist(const Point &a, const Point &b) {
  return sqrt(double(sqr(a.x-b.x) + sqr(a.y-b.y)));
}

inline double perimeter(const Point &a,
                        const Point &b,
                        const Point &c) {
  return dist(a,b) + dist(b,c) + dist(c,a);
}

double calc(int n, const Point points[],
            const vector<Point> &pointsByY) {
  if(n<3) return INF;
  int left = n/2;
  int right = n-left;
  Point split = middle(points[left-1], points[left]);
  vector<Point> pointsByYLeft, pointsByYRight;
  pointsByYLeft.reserve(left);
  pointsByYRight.reserve(right);
  REP(i,n) {
    if(cmpx(pointsByY[i], split))
      pointsByYLeft.push_back(pointsByY[i]);
    else
      pointsByYRight.push_back(pointsByY[i]);
  }
  double res = INF;
  res = min(res, calc(left, points, pointsByYLeft));
  res = min(res, calc(right, points+left, pointsByYRight));
  static vector<Point> closeToTheLine;
  int margin = (res > INF/2) ? 2*BILLION : int(res/2);
  closeToTheLine.clear();
  closeToTheLine.reserve(n);
  int start = 0;
  for(int i=0;i<n;++i) {
    Point p = pointsByY[i];
    if(abs(p.x - split.x) > margin) continue;
    while(start < size(closeToTheLine) &&
          p.y - closeToTheLine[start].y > margin) ++start;
    for(int i=start;i<size(closeToTheLine);++i) {
      for(int j=i+1;j<size(closeToTheLine);++j) {
        res = min(res, perimeter(p, closeToTheLine[i],
                                    closeToTheLine[j]));
      }
    }
    closeToTheLine.push_back(p);
  }
  return res;
}

double calc(vector<Point> &points) {
  sort(points.begin(), points.end(), cmpx);
  vector<Point> pointsByY = points;
  sort(pointsByY.begin(), pointsByY.end(), cmpy);
  return calc(size(points), &points[0], pointsByY);
}

int main() {
  assert(0==system("cat > Input.java"));
  fprintf(stderr, "Compiling generator\n");
  assert(0==system("javac Input.java"));
  fprintf(stderr, "Running generator\n");
  assert(0==system("java -Xmx512M Input > input.tmp"));
  fprintf(stderr, "Solving\n");
  FILE *f = fopen("input.tmp", "r");
  int ntc; fscanf(f, "%d", &ntc);
  REP(tc,ntc) {
    int n; fscanf(f, "%d", &n);
    vector<Point> points;
    points.reserve(n);
    REP(i,n) {
      int x,y; fscanf(f, "%d%d", &x, &y);
      points.push_back(Point(2*x-BILLION,2*y-BILLION));
    }
    double res = calc(points);
    printf("Case #%d: %.15e\n", tc+1, res/2);
  }
  fclose(f);
}
```