Universiteit Utrecht

Team Code Reference

Jens Heuseveldt, Ludo Pulles & Peter Ypma

NWERC

November 20, 2016

1 Voorbereiding

- Goed slapen
- Genoeg flesjes water mee
- Toetsenbord inclusief kabel
- Iets te eten mee (bananen, ligakoeken en dat soort dingen.
- Pennen, potloden, passer, geodriehoek en markeerstiften mee.

2 Begin van contest

Peter en Ludo nemen **op hun gemak** alle opgaven samen door. Daarbij noteren ze het volgende:

- een overzichtslijst van alle opgaven, waarop aangegeven staat van welk type de opgaven zijn en wie de opgave verder gaat uitwerken.
- Bij opgaven waar dat van toepassing is: noteren welk deel van de TCR daar mogelijk handig voor nodig is. Dit blaadje wordt direct aan Jens doorgegeven.

Jens typt vervolgens de volgende dingen op de computer:

2.1 VIM

```
set nu sw=4 ts=4 noexpandtab autoindent hlsearch
syntax on
colorscheme slate
```

2.2 Template

```
#include<bits/stdc++.h>

#define x first
#define y second

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<int> vi;

const int INF = 2147483647; // (1 << 30) - 1 + (1 << 30)
const ll LLINF = 9223372036854775807LL; // (1LL << 62) - 1 + (1LL << 62)
const double pi = acos(-1.0);

// lambda-expression: [] (args) -> retType { body }

const bool LOG = false;
void Log() { if(LOG) cerr << "\n\n"; }
template<class T, class... S>
void Log(T t, S... s) {
    if(LOG) cerr << t << "\t", Log(s...);
}

```

```cpp
template<class T1, class T2>
ostream& operator<<(ostream& out, const pair<T1,T2> &p) {
    return out << '(' << p.first << ",␣" << p.second << ')';
}

template<typename T1, typename T2>
ostream& operator<<(ostream &out, pair<T1, T2> p) {
    return out << "(" << p.first << ",␣" << p.second << ")";
}

template<class T>
using min_queue = priority_queue<T, vector<T>, greater<T>>;

// Order Statistics Tree (if this is supported by the judge software)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<class TIn, class TOut> // key, value types. TOut can be null_type
using order_tree = tree<TIn, TOut, less<TIn>,
    rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order(int r) (0-based)
// order_of_key(TIn v)
// use key pair<Tin,int> {value, counter} for multiset/multimap

int main() {
    ios_base::sync_with_stdio(false); // faster IO
    cin.tie(NULL);                    // faster IO
    cerr << boolalpha;                // (print true/false)
    (cout << fixed).precision(10);    // set floating point precision
    // TODO: code
    return 0;
}
```

2.3 Git

```
git config --global user.name "Git␣Diff␣Solution"
git config --global user.email winnaars@NWERC.com
git init
git add *
git commit -m "Initialisatie"
```

Overig

Wanneer de VIM-instellingen en de template klaar is, maakt Jens achtereenvolgens:

- De testcase bestanden: A1.in, A2.in, etcetera.
- Het typen van de delen van de TCR die waarschijnlijk nodig zijn bij de opgaven (hij krijgt deze van Ludo en Peter die deze lijst maken tijdens het doorlopen van de opgaven).

In de tussentijd werkt Ludo op papier één à twee niet zeer gemakkelijke opgaven uit die hij als hij er klaar voor is (of als Jens klaar is, gaat typen). Vanaf dat moment focused Jens zich op de opgaven. waarvan Peter/Ludo dachten dat Jens die goed op kan lossen en de opgaven die heel snel opgelost zijn door andere teams. Peter is ondertussen bezig met de wiskundigere opgaven.

3 Geometry

Tricky testcases

- Verticale lijnen
- Evenwijdige lijnen
- Niet convex
- Wat gebeurt er als het convex omhulsul uit alle punten bestaat?
- Let op afrondingen (bij floating points)

3.1 Punten (2D) en rotaties

```cpp
typedef double NUM; // double of long long

struct pt { NUM x, y; // Maak punt door pt var = pt(x,y)
    pt() { x = y = 0; }
    pt(NUM _x, NUM _y) : x(_x), y(_y) {}
// Definieer operaties: scalar (*), inproduct (*), 2D-uitproduct (^)
    pt operator*(NUM scalar) const {return pt(scalar * x, scalar * y);}
    NUM operator*(const pt &rhs) const {return x * rhs.x + y * rhs.y;}
    NUM operator^(const pt &rhs) const {return x * rhs.y - y * rhs.y;}
    pt operator+(const pt &rhs) const{return pt(x + rhs.x, y + rhs.y);}
    pt operator-(const pt &rhs) const{return pt(x - rhs.x, y - rhs.y);}
    NUM lensq() const { return x*x + y*y; }     // Lengte van lijn in
        kwadraat.
    NUM len() const { return sqrt(lensq()); }
// Als Num = double moeten onderstaande operatoren met fabs(x-rhs.x) < EPS.
    bool operator==(const pt &rhs) const{return x == rhs.x && y == rhs.y;}
    bool operator!=(const pt &rhs) const{return x != rhs.x || y != rhs.y;}
    bool operator<(const pt &rhs) const{ // Handig voor sorteren.
        if (x != rhs.x) return x < rhs.x; return y < rhs.y;}
};
NUM sqDist(const pt &a, const pt &b) {return (b-a).lensq();}
NUM Dist(const pt &a, const pt &b) {return (b-a).len();}

pt rotate(pt p, double theta) {  // draait punt tegen de klok in rond (0,0).
double rad = (theta) * acos(-1.0) / 180.0;  // mits theta in graden.
return pt(p.x * cos(rad) - p.y * sin(rad),
            p.x * sin(rad) + p.y * cos(rad)); }
// Als je punt b rond punt a wilt draaien: rotate( b - a, theta) + a

struct poolpt { double r, phi;  // Poolcoordinaten!!
    poolpt() {r = phi = 0;}
    poolpt(double _r, double _phi) : r(_r), phi(_phi) {}
};
poolpt car_to_pool(pt p) { return poolpt(p.len(), atan2(p.y,p.x)); }
pt pool_to_car(poolpt p) { return pt(p.r * cos(p.phi), p.r * sin(p.phi));}
```

3.2 Lijnen

```cpp
#include <math.h>
#include "./points.cpp"
double EPS = 0.00000001;
// We definieren een lijn met ax+by+c=0. Daarbij kiezen we b = 1 als b != 0.
struct line { double a, b, c; };
// Het initialiseren van de lijn kan met de volgende functie:
line pttoline(pt p1, pt p2)
{   line l;
    if (fabs(p1.x - p2.x) < EPS) { l.a = 1.0; l.b = 0.0; l.c = -p1.x; }
    else { l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x); l.b = 1.0; l.c = -(
        double)(l.a * p1.x) - p1.y;}
    return l;
}   // Parallelcheck:
bool zijnParallel(line l1, line l2) {
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }
// Snijpunten + check (let op: return ook false als lijnen hetzelfde zijn!)
bool snijpunten(line l1, line l2, pt &p) { // in &p komt het snijpunt.
  if (zijnParallel(l1, l2)) return false;
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else                  p.y = -(l2.a * p.x + l2.c); // Line 1 is verticaal!
  return true; }

// Voor verplaatsen van lijnstukken en afstandberekeningen zijn vectoren
    handig.
struct vec { double x, y;
    vec(double _x, double _y) : x(_x), y(_y) {}
    NUM operator*(const vec &rhs) const {return x * rhs.x + y * rhs.y;}
    NUM operator^(const vec &rhs) const {return x * rhs.y - y * rhs.x;}
    NUM lensq() const { return x*x + y*y;}
};
vec verschilvector(pt a, pt b) { return vec(b.x - a.x, b.y - a.y); }
vec schalen(vec v, double s) { return vec(v.x * s, v.y * s); }
pt verplaatsen(pt p, vec v) { return pt(p.x + v.x , p.y + v.y); }

// Afstand tussen een punt en een lijn bepalen.
double afstandTotLijn(pt p, pt a, pt b, pt &c) {
  vec ap = verschilvector(a, p), ab = verschilvector(a, b);
  double u = (ap * ab) / ab.lensq();
  c = verplaatsen(a, schalen(ab, u));
  return sqrt((p-c).len()); }
 // In het geval dat we de afstand tot lijnsegment ab willen: als u < 0 is
     de afstand |ap| en als u > 1 dan is de afstand |bp|.

double hoek(pt a, pt o, pt b) {  // geeft hoek aob in radialen.
  vec oa = verschilvector(o, a), ob = verschilvector(o, b);
  return acos(oa * ob / sqrt(oa.lensq() * ob.lensq())); }
// Onderstaande test of lopen van p -> q -> r een hoek naar links is (
    counterclockwise). Als < EPS is het een check of p,q,r collinear zijn.
 bool ccw(pt p, pt q, pt r) {
  return (verschilvector(p, q) ^ verschilvector(p, r)) > 0; }
```

3.3 Veelhoeken

```cpp
#include "./lines.cpp"
// We slaan veelhoeken op als een vector van punten in counterclockwise
    volgorde. Hierbij zetten we het eerste punt ook op het eind.
// VB: vector<pt> P: P.push_back(pt(1, 1)); P.push_back(pt(3, 3)); P.
    push_back(pt(1, 5)); P.push_back(P[0]) is een driehoek.
#include <vector>
using namespace std;
double omtrek(const vector<pt> &P) {
  double resultaat = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++)
    resultaat += Dist(P[i], P[i+1]);
  return resultaat; }
// De oppervlakte is de helft van de determinant van aanliggende punten.
double oppervlakte(const vector<pt> &P) {
  double resultaat = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    resultaat += P[i]^P[i+1];
  }     return fabs(resultaat) / 2.0; }
// Een veelhoek is convex DESDA alle hoeken linksaf zijn (wanneer je ze
    tegen de klok in doorloopt).
bool isConvex(const vector<pt> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;   // punten en lijnen (sz=2/3) zijn niet convex
  bool isLinks = ccw(P[0], P[1], P[2]);
  for (int i = 1; i < sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLinks)
      return false;
  return true; }
// Dit checkt of een punt binnen een veelhoek ligt door te checken of de som
     van de hoeken (linksaf positief en rechtsaf negatief) gelijk is aan 360
    graden.
// Let op: Dit werkt ook voor niet-convexe veelhoeken, maar detecteert geen
    punten op de rand (voeg dus op lijnstukcheck toe als dit mogelijk is).
bool inVeelvlak(pt punt, const vector<pt> &P) {
  if ((int)P.size() == 0) return false;
  double som = 0;    // assume the first vertex is equal to the last vertex
  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(punt, P[i], P[i+1]))
        som += hoek(P[i], punt, P[i+1]);                    // left turn/ccw
    else som -= hoek(P[i], punt, P[i+1]); }                 // right turn/cw
  return fabs(fabs(som) - 2*acos(-1.0)) < EPS; }
```

3.4 Convex Hull

```cpp
// Dit is het Graham's scan algoritme.
// Input: verzameling punten
pt pivot;
bool sorteerHoeken(pt a, pt b) {
  if (collinear(pivot, a, b))
    return dist(pivot, a) < dist(pivot, b);
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }

vector<pt> CH(vector<pt> P) {   // De punten van P worden hergesorteerd.
```

```
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P;                             // special case, the CH is P itself
  }
  // first, find P0 = point with lowest Y and if tie: rightmost X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;
  pt temp = P[0]; P[0] = P[P0]; P[P0] = temp;
  // second, sort points by angle w.r.t. pivot P0
  pivot = P[0];                       // use this global variable as reference
  sort(++P.begin(), P.end(), sorteerHoeken);  // we do not sort P[0]
  // third, the ccw tests
  vector<pt> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]);   // initial S
  i = 2;                                              // then, we check the rest
  while (i < n) {           // note: N must be >= 3 for this method to work
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);  // left turn, accept
    else S.pop_back(); }   // or pop the top of S until we have a left turn
  return S; }                                          // return the result
```

3.5 Cirkels

```
// Een koorde in een cirkel die middelpuntshoek alpha heeft,
// heeft lengte 2 * r * sin(alpha/2).
#include "./lines.cpp"
// 0 (in cirkel, 1 op rand, 2 erbuiten)
int binnenCirkel(pt p, pt c, NUM r) {
  NUM dx = p.x - c.x, dy = p.y - c.y;
  NUM Euc = dx * dx + dy * dy, rSq = r * r;
  return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } // Als double gebruik <EPS

bool vindMiddelpunt(pt p1, pt p2, double r, pt &c) {
  double d2 = sqDist(p1, p2);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false; // Geen oplossing als afstand > 2R.
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * sqrt(det);
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * sqrt(det);
  return true; } // ander middelpunt krijg je door p2 en p1 om te draaien.
// Wanneer je drie punten hebt, kun je de cirkel door deze punten vinden
    door eerst de straal te berekenen met Dist(p1,p2) * Dist(p2,p3) * Dist(p3
    ,p1) / (4 * oppDriehoek(p1,p2,p3)) en dan het juiste middelpunt te nemen.
```