Лекция 1.

- точки, struct, вектора, operator-.
- расстояние между точками (функция dist, _hypot).
- проверка на треугольность: если max длины <= суммы двух других длин.
- медиана треугольника: они пересекаются в одной точке, делятся в отношении 2 к 1, это центр масс (в обоих пониманиях), формула длины: $m_C = \sqrt{\frac{2a^2 + 2b^2 - c^2}{4}}$ (вывод – достроение до параллелограмма); строить можно вектор до середины.
- биссектриса треугольника: пересекаются в одной точке (центре вписанной окружности), делит противоположную сторону в отношении длин прилежащих сторон, строить можно как полусумму двух нормированных векторов.
- серединные перпендикуляры пересекаются в одной точке – центре описанной окружности.
- вписанная окружность: 2S / P, находить – как пересечение биссектрис.
- описанная окружность: abc / 4S, находить – как пересечение серединных перпендикуляров.
- теорема косинусов: a^2 = b^2 + c^2 – 2 b c cos alpha. Отсюда можно восстанавливать угол.
- теорема синусов: a / sin alpha = 2 R.
- скалярное произведение: определение через координаты и через длины, применение для восстановления угла, значение знака.
- псевдоскалярное (косое) произведение: |a| |b| sin alpha, тогда можно находить площадь треугольника, можно находить ориентированный угол и проверять направленность тройки, проверять вектора на коллинеарность, x1 y2 – x2 y1.
- векторное произведение: это вектор, перп. им обоим, имеющий длину модуля их косого произведения, сам вектор можно искать через определитель (в 3D).
- угол между векторами: через теорему косинусов, или через арктангенсы (atan, atan2).
- площадь треугольника: через векторное произведение или через формулу Герона (точность).
- теорема Пика: S = I + B/2 – 1. Док-во: в одну сторону: из многоугольника плюс треугольник можно пересчитать и получить для нового многоугольника (основываемся на том, что любой многоугольник можно триангулировать). Для треугольника формулу доказываем: сначала для прямоугольника, потом для прямоугольного треугольника, потом для произвольного (достраивая его до прямоугольного).

Лекция 2.

- задача на теорему Пика: число целочисленных точек внутри многоугольника (координаты большие).
- формула Эйлера, планарный граф ($f+n-m=2=1+k$, док-во: удаляем ребро пока не дойдём до дерева; док-во линейности числа рёбер ($3f<=2m$)); задача о нахождении числа областей, которые получаются в результате пересечения заданных прямых.
- прямые: задание алгебраически (A,B,C, но независимы только A или B и C; можно нормализовать; (A,B) – вектор нормали, т.е. можно =(cos,sin)), и параметрически ($a+vt$).
- построение нормальной прямой.
- расстояние от точки до прямой, расположение точки относительно прямой.
- проверка двух прямых на пересечение (параллельность), пересечение прямых (в алгебраическом способе – решение системы линейных уравнений, в параметрическом – решение трёх линейных уравнений).
- полуплоскости, работа с ними, проверка точки на принадлежность пересечению полуплоскостей, отсечение многоугольника полуплоскостью.
- расстояние от точки до отрезка (тернарка или взять скалярное произведение и, если надо, расстояние до прямой).
- проверка двух отрезков на пересечение: если не лежат на одной прямой, то пересечь две прямые или посмотреть на знаки векторных произведений; иначе посмотреть, что пересекаются bounding-boxes.
- расстояние между отрезками (проверка на пересечение плюс минимум из расстояний).
- принадлежность точки прямой, лучу, отрезку.
- площадь многоугольника (методом трапеций, методом треугольников) – значение знака (орентирование в нужную сторону).
- проверка точки на принадлежность треугольнику (по сумме площадей или направлению поворотов; особые случаи – когда на границе, когда треугольник вырожден).
- проверка на принадлежность точки произвольному многоугольнику (методом случайного луча, методом горизонтального луча).
- проверка на принадлежность точки выпуклому (звездному) многоугольнику (бинарный поиск по сектору, и потом проверка на принадлежность треугольнику).
- проверка на выпуклость.
- окружности: проверка точки на принадлежность; проверка отрезка/прямой на пересечение/принадлежность; точка пересечения окружности и прямой, точка пересечения двух окружностей.

Лекция 3.

- восстановление треугольника по трём сторонам (в итоге линейное уравнение); по двум сторонам и медиане (в итоге линейное уравнение); по высоте, биссектрисе, медиане (бинпоиск по углу при вершине).
- формулы поворота; поворот на 90 градусов; задача: достроить многоугольник до правильного, зная координаты двух вершин и их число.
- триангуляция; центры масс в случае равномерного распределения массы; тетраэдроизация.
- построение планарного графа по заданным отрезкам.
- треугольник максимальной площади.
- вертикальная декомпозиция: площадь объединения треугольников.

Лекция 4.

- две ближайшие точки в 2D (можно в полосе брать от каждой точки вниз 7 соседей (т.к. по 4 от каждой половинки, что легко доказать разбиением на 4 квадрата), или 6 соседей из правой половинки (т.к. в прямоугольнике delta * 2delta может быть только 6 таких точек).

```
sort (a, a+n, &cmp_x);
get (0, n-1);
void get (int l, int r) {
	if (l == r)  return;

	int m = (l + r) >> 1;
	int midx = a[m].x;
	get (l, m),  get (m+1, r);
	inplace_merge (a+l, a+m+1, a+r+1, &cmp_y);

	static pt t[MAXN];
	int tsz = 0;
	for (int i=l; i<=r; ++i)
		if (sqr (a[i].x - midx) < mindist) {
			for (int j=tsz-1, k=0; j>=0 && k<7; --j, ++k)
				upd_ans (a[i], t[j]);
			t[tsz++] = a[i];
		}
}
```

- две ближайшие точки в пространствах большей размерности.

простой алгоритм даёт N log^(d-1) N (Т.к. U(n,d) = 2U(n/2,d) + U(n,d-1)), — когда разбиваем медианной плоскостью по любой координате на два множества, рекурсивно пускаем от каждого, а потом выделяем полосу, проецируем на плоскость, и от неё пускаем другую функцию (которая тоже находит медианную плоскость, разделяет ей на две, и проецирует в полосе).

```
void project_onto_plane (vector<pt>::iterator begin, vector<pt>::iterator end, vector<pt> & res,
	int idx, int cnt, double plane_coo, double delta)
{
	while (begin != end) {
		if (abs (begin->a[idx] - plane_coo) <= delta + EPS) {
			res.push_back (*begin);
			swap (res.back().a[idx], res.back().a[cnt-1]);
		}
		begin++;
	}
}

double solve_medium_layer (vector<pt>::iterator begin, vector<pt>::iterator end, int cnt, double delta) {
	int n = int (end - begin);
	if (n == 1 || cnt == 0)
		return 1E20;

	int sel = cnt-1;
	sort (begin, end, cmp (sel));
	vector<pt>::iterator mid = begin + n / 2;
	double result = min (solve_medium_layer (begin, mid, cnt, delta),
		solve_medium_layer (mid, end, cnt, delta));

	if (cnt > 1) {
		vector<pt> tmp;
		project_onto_plane (begin, end, tmp, sel, cnt, mid->a[sel], delta);
		result = min (result, solve_medium_layer (tmp.begin(), tmp.end(), cnt-1, delta));
	} else {
		for (int i=0; i<n; ++i)
			for (int j=i+1; j<n && abs ((begin+i)->a[0] - (begin+j)->a[0]) <= delta + EPS; ++j)
				result = min (result, (begin+i)->dist (*(begin+j)));
	}
```

```
	return result;
}

double solve_medium (vector<pt>::iterator begin, vector<pt>::iterator end) {
	int n = int (end - begin);
	if (n <= 1)
		return 1E20;

	int sel = N-1;
	sort (begin, end, cmp (sel));
	vector<pt>::iterator mid = begin + n / 2;
	double result = min (solve_medium (begin, mid), solve_medium (mid, end));

	vector<pt> tmp;
	project_onto_plane (begin, end, tmp, sel, N, mid->a[sel], result);
	result = min (result, solve_medium_layer (tmp.begin(), tmp.end(), N-1, result));

	return result;
}
```

- пара пересекающихся отрезков за N log N

Док-во основано на том, что для самой левой нижней точки пересечения найдётся такое событие (и оно в ней или слева от неё), в котором пересекающиеся отрезки соседние.

```
bool cmp (event a, event b) {
	return a.p.x < b.p.x || a.p.x == b.p.x && (a.type > b.type || a.type == b.type && a.p.y < b.p.y);
}

int vect (pt a, pt b, pt c) {
	return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

bool operator< (seg p, seg q) {
	if (p.a.x < q.a.x) {
		int s = vect (p.a, p.b, q.a);
		if (s > 0)  return true;
		if (s == 0 && p.a.y < q.a.y)  return true;
		return false;
	}
	if (q.a.x < p.a.x) {
		int s = vect (q.a, q.b, p.a);
		if (s < 0)  return true;
		if (s == 0 && p.a.y < q.a.y)  return true;
		return false;
	}
	return p.a.y < q.a.y;
}

bool intersect_1 (int a, int b, int c, int d) {
	return max (a, b) >= min (c, d) && max (c, d) >= min (a, b);
}

bool intersect (seg p, seg q) {
	pt a = p.a,  b = p.b,
		c = q.a,  d = q.b;
	int s11 = vect (a, b, c);
	int s12 = vect (a, b, d);
	int s21 = vect (c, d, a);
	int s22 = vect (c, d, b);
	if (s11 == 0 && s12 == 0 && s21 == 0 && s22 == 0)
		return intersect_1 (a.x, b.x, c.x, d.x)
			&& intersect_1 (a.y, b.y, c.y, d.y);
	else
		return (s11 * s12 <= 0) && (s21 * s22 <= 0);
}
```

```
pair<int,int> solve (vector<seg> a) {
	int n = (int) a.size();

	vector<event> b;
	for (int i=0; i<n; ++i) {
		a[i].id = i;
		if (a[i].b < a[i].a)
			swap (a[i].a, a[i].b);
		b.push_back (event (a[i].a, i, +1));
		b.push_back (event (a[i].b, i, -1));
	}
	sort (b.begin(), b.end(), cmp);

	set<seg> q;
	for (int i=0; i<n*2; ++i) {
		int id = b[i].id;
		if (b[i].type == +1) {
			set<seg>::iterator it = q.lower_bound (a[id]);
			if (it != q.end() && intersect (*it, a[id]))
				return make_pair (it->id, a[id].id);
			if (it != q.begin() && intersect (*--it, a[id]))
				return make_pair (it->id, a[id].id);
			q.insert (a[id]);
		}
		else {
			set<seg>::iterator it = q.lower_bound (a[id]),
				next = it,  prev = it;
			if (it != q.begin() && it != --q.end()) {
				++next,  --prev;
				if (intersect (*next, *prev))
					return make_pair (next->id, prev->id);
			}
			q.erase (it);
		}
	}
	return make_pair (-1, -1);
}
```