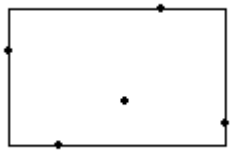
A Квадрат

Решение

Для начала найдем прямоугольник минимальной площади, покрывающий все точки.



Это не сложно – его левая сторона – минимум из всех X-ов точек. Правая – максимум их X-ов. Тоже самое и для верхней и нижней сторон. Это так называемый ограничивающий прямоугольник (bounding box)

Ну далее не сложно понять, что оптимальный квадрат всегда можно сдвинуть так, чтобы он касался двух сторон ограничивающего прямоугольника. Это значит, что одной из его сторон будет максимальная сторона прямоугольника.

Итого O(N)

```
int main()
{
	init();

	double  oo=1e20;
	double xl,yl,xr,yr;
	xl=yl=oo;
	xr=yr=-oo;
	int n;
	scanf("%d\n",&n);

	for (int i=0;i<n;i++)
	{
		double x,y;
		scanf("%lf %lf",&x,&y);
		xl=min(xl,x);
		yl=min(yl,y);
		xr=max(xr,x);
		yr=max(yr,y);
	}

	double l=max(xr-xl,yr-yl);

	printf("%.2lf\n",l*l);

	return 0;
}
```

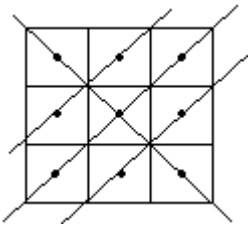
B Покрыть точки

Ответ: $max(x_i + y_i)$.

C Точки

Решение

Оптимальное решение всегда может быть таким



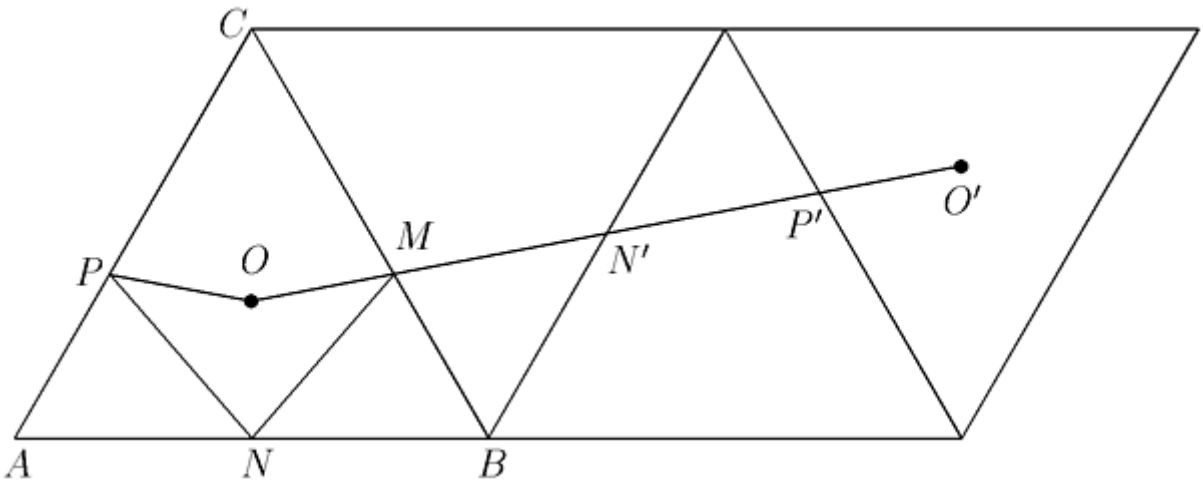
Ответ – количество чисел на границе пополам. Так как за 1 шаг мы не можем вычеркнуть более 2 чисел на границе.

Если одна из размерностей равна 1 то ответ – большая сторона

```
long long n,m;

int main()
{

	cin >> n >> m;

	if (n==1 || m==1) cout << n+m-1; else
	cout << n+m-2;

	return 0;
}
```

D Остров

Так как нужно пройти циклический путь причем обязательно посетить точку O то с нее можно и начинать обход. Пусть путь – ломаная OMNPO. Она соответствует пути луча, отразившегося от сторон и вернувшегося обратно. Если мы 3 раза перекатим треугольник через стороны (осевая симметрия - в первом случае относительно стороны BC),то получим отрезок, равный по длине нашей ломаной.



По теореме Пифагора находим ответ: a * $\sqrt{7/3}$.

Если просто умножать по семплу на 1.53 то ответ получался не совеем точным для больших n))

E Многоугольники

Решение

Мы можем построить многоугольник, тогда, когда максимальная сторона больше суммы всех остальных. На каждом шаге нам нужно быстро находить максимум в множестве чисел. Сумму достаточно поддерживать одной переменной. Эта задача решается использование очереди с приоритетом или структуры multi_set (в отличие от set поддерживает дубликаты)

Как проверить на треугольник. Переберем каждую сторону – пусть она будет большей. Нужно найти 2, такие, что их сумма больше этой стороны. Очевидно, что эти 2 стороны должны быть максимальными из меньших самой длинной стороны.

Есть 2 варианта решения:

1. Очевидный, но сложнее в реализации.

Нужно на каждом шаге поддерживать количество разностей, больших нуля. То есть для каждой тройки сторон a,b,c , где a<=b<=c нам нужно знать c-(a+b)

2. Заметить что, если у нас в множестве более 30 сторон , то треугольник всегда может быть построен. Почему: худший случай: 1 1 2 3 5 8 13 .. Каждый следующий равен сумме двух предыдущих – очень быстро растет (ограничение на длину стороны 10^6)

А такую задачу легко решить, пройдясь итератором по этим элементам.

Утверждается, что обращение происходит за O(1) в среднем.

Сложность **O(N * log(N)).**

```
multiset <int> m;
long long sum;

char calc()
{
	if (sz(m)<3) return 'n';
	int mas[40];
	multiset <int>::iterator it = m.end();
	it--;
	int pol = 0, tr = 0;
	int mx = *it;
	if (sum-mx>mx) pol = 1;
	int cur=0;
	for (it = m.begin();it!=m.end();it++)
	{
		mas[cur] = *it;
		cur++;
		if (cur>=40) break;
	}
	for (int i=2;i<cur;i++)
		if (mas[i]<mas[i-1]+mas[i-2]) tr=1;
	if (!tr && !pol) return 'n';
	if (tr && pol) return 'b';
	if (pol) return  'p';
	return 't';
}

int main()
{

	int n;
	scanf("%d\n",&n);

	m.clear();
	sum=0;
	while (n--)
	{
		int a;
```

```
            scanf("%d",&a);
            if (a>0)
            {
                  sum+=a;
                  m.insert(a);
            } else
            {
                  a=-a;
                  sum-=a;
                  m.erase(m.find(a));
            }
            printf("%c\n",calc());
      }

      return 0;
}
```