Вещественный бинарный поиск.

При помощи бинарного поиска можно подбирать не только целочисленные, но и вещественные ответы. Правда в алгоритм нужно будет внести некоторые изменения.

Попробуем этот подход на следующей, несколько искусственной задаче.

- Дано вещественное число N>1.
- Требуется определить $\sqrt[3]{N}$ без использования функцией вычисления степени из математической библиотеки.

```
 l:=1;r:=n;
while(???) do begin
    x:=(l+r)/2.0;
    if(x*x*x<=n) then l:=x
else r:=x;
  end;
  writeln(l);
```

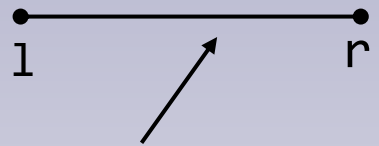
```
l:=1;r:=n;
 for var i:=1 to 64 do begin
    x:=(l+r)/2.0;
    if(x*x*x<=n) then l:=x
else r:=x;
  end;
  writeln(l);
end.
```

Output Window

```
100
4.64158883361278
4.64158883361278
```

Подход номер 1

```
eps:=1e-9;
l:=1;r:=n;
while((r-l)>eps) do begin
```



Между ними нет ни одного вещественного числа, которое может быть записано в double, поэтому при вычислении середины мы попадаем либо в `l`, либо в `r` и ЗАЦИКЛИВАЕМСЯ.

Вместо while - for !!!

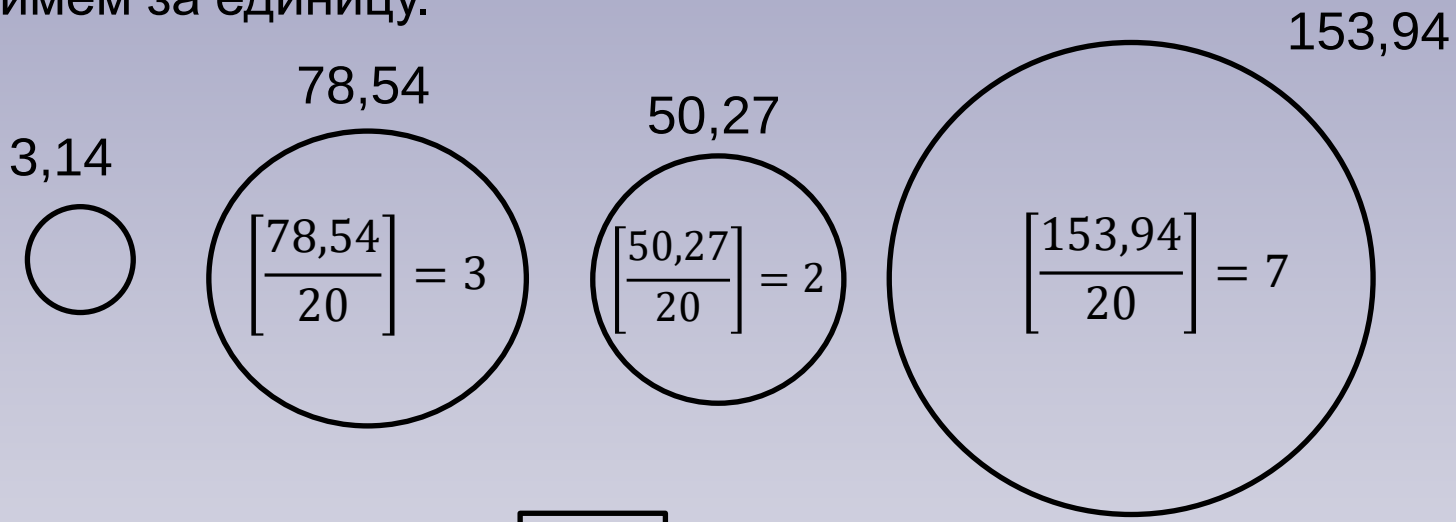
У героя задачи день рождения!!!

◆ Даны N круглых тортов одинаковой высоты с заданными радиусами.

◆ Нужно получить M кусков равного объема (необязательно одной формы). Каждый кусок должен быть получен из одного торта.

◆ Требуется определить максимально возможный объем куска.

Пусть у нас есть четыре торта с радиусами 1, 5, 4, 7, высоту примем за единицу.



Кандидат на ответ | 40 |

0, 1, 0, $\left[\frac{153,94}{40}\right] = 3$. Всего 5 кусков – нам не подходит.

Кандидат на ответ | 20 |

0+3+2+7=12. Нам подходит.

```
const n=4;//количество тортов
type arr=array[1..n] of real;
function can(var a:arr;
n,g:integer; x: real):
boolean;//массив с объемами,
количество тортов, число
гостей, проверяемый объем
var t: integer;
begin
  t:=0;
  for var i:=1 to n do
t:=t+floor(a[i]/x);
if (t>=g) then result:=true
else result:=false;
end;
```

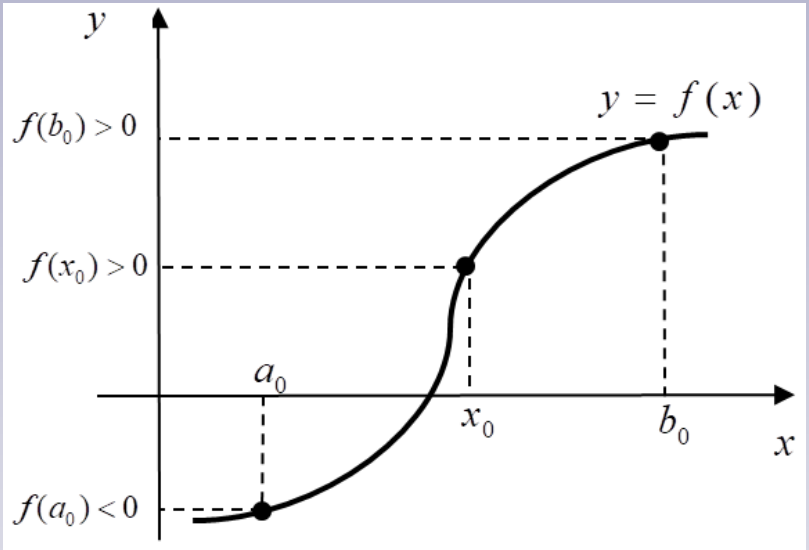
```
var a:arr = (1,5,4,7);
    m,l,r,h: real;
    g:integer;//количество гостей
begin
h:=1; g:=8;
for var i:=1 to n do
a[i]:=Pi*a[i]*a[i]*h;
l:=0;r:=1000000000;
for var i:=1 to 64 do begin
  m:= (l+r)/2;
  if(can(a,n,g,m)) then l:=m else
r:=m;
  end;
  writeln(l);
end.
```

Output Window

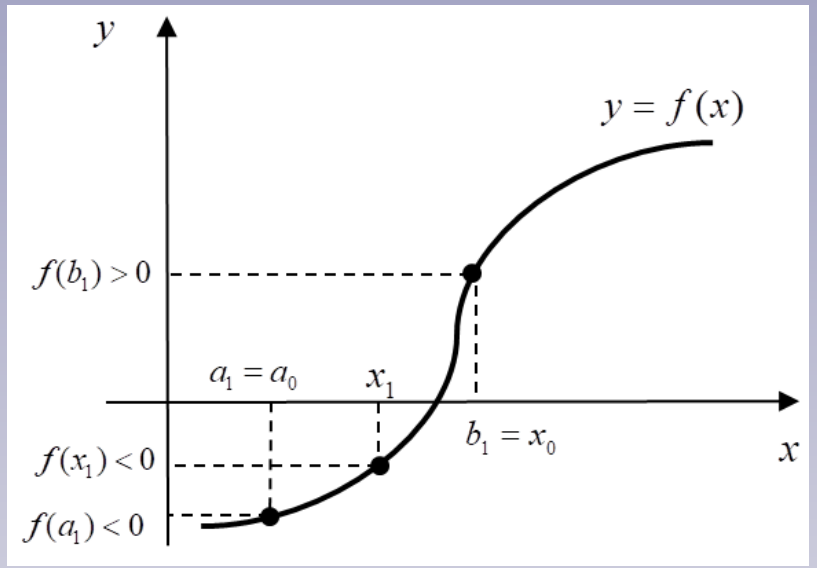
30.7876080051666

Решение нелинейных уравнений.

Интервал изоляции – интервал, внутри которого функция монотонна и принимает разные знаки на его концах.



Начальный интервал изоляции $[a_0, b_0]$ делим пополам, т.е. находим середину отрезка: $x_0 = \frac{a_0+b_0}{2}$.



В качестве нового интервала выбираем ту половину отрезка, на концах которого функция имеет разные знаки.

Другую половину, на которой функция знак не меняет, отбрасываем.

Новый интервал вновь делим пополам, получаем очередное приближение к корню $x_0 = \frac{a_1+b_1}{2}$ и т.д.

$x^2 + \sqrt{x} = 15{,}6$ на отрезке $[0; 5]$.

$f(x) = x^2 + \sqrt{x} - 15{,}6;$ $f(0) < 0,$ $f(5) > 0.$

```
function fun(t:real):real;
begin
  result:=t*t+sqrt(t)-15.6;
end;
var x, eps, a, b: real ;
begin
  a:=0;b:=5;
  for var i:=1 to 64 do begin
  x:=(a+b)/2.0;
  if(fun(a)*fun(x)<0) then b:=x   else a:=x; end;
  writeln ((a+b)/2.0);
end.
```

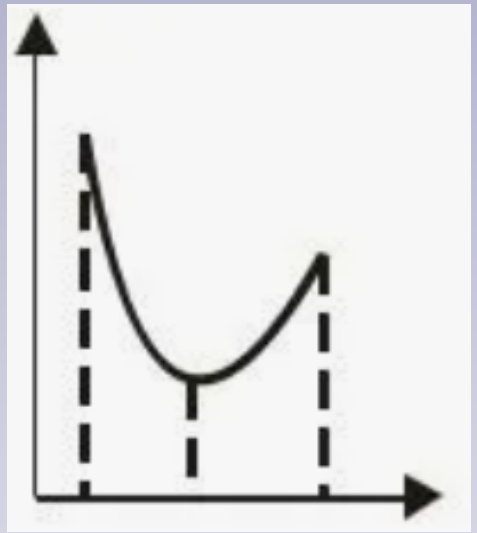
Output Window

```
3.69823216882969
```

Данный метод будет работать и для немонотонной функции со значениями разных знаков на конце некоторого заданного отрезка, но в этом случае будет найден какой-то корень из этого отрезка, не обязательно единственный.

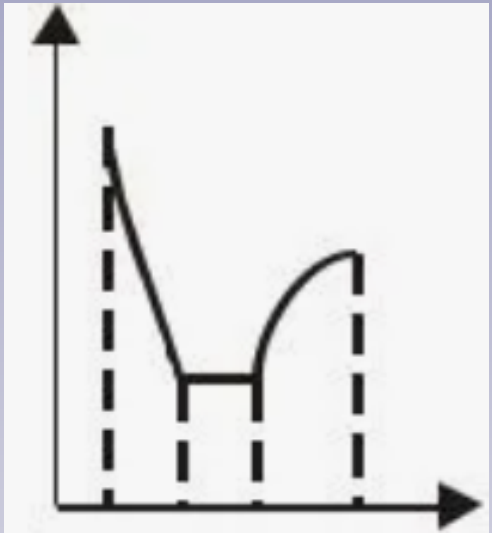
Тернарный поиск

Представим, что у нас есть функция со следующим графиком



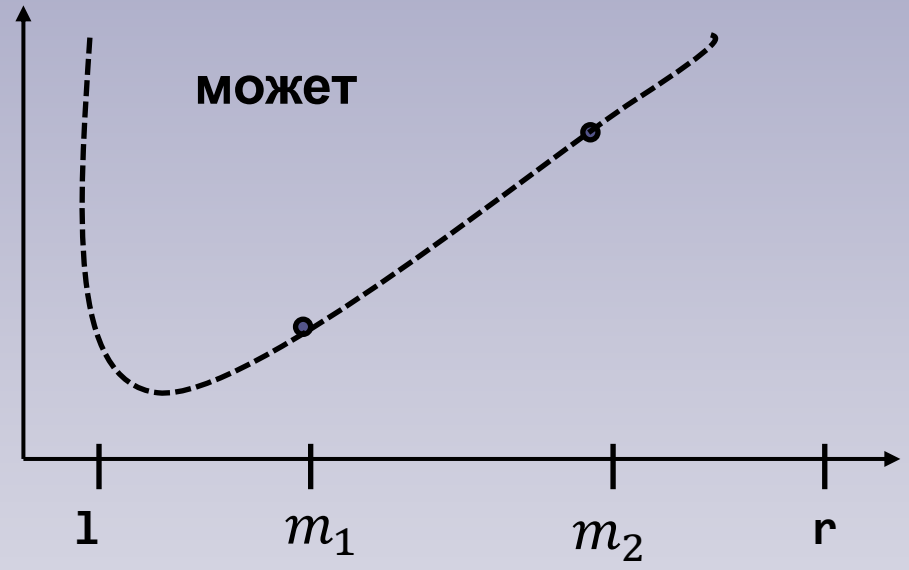
Значения функции с ростом x сначала строго убывают, затем в какой-то точке достигают минимума и далее строго возрастают.

Наша задача – определить минимум этой функции



Вообще говоря, этот минимум на графике не обязан быть одной точкой, это также может быть непрерывный отрезок, на котором значения функции совпадают и минимальны. Главное, чтобы такой отрезок был только один.

Поставим указатели `l` и `r` на границы рассматриваемой области



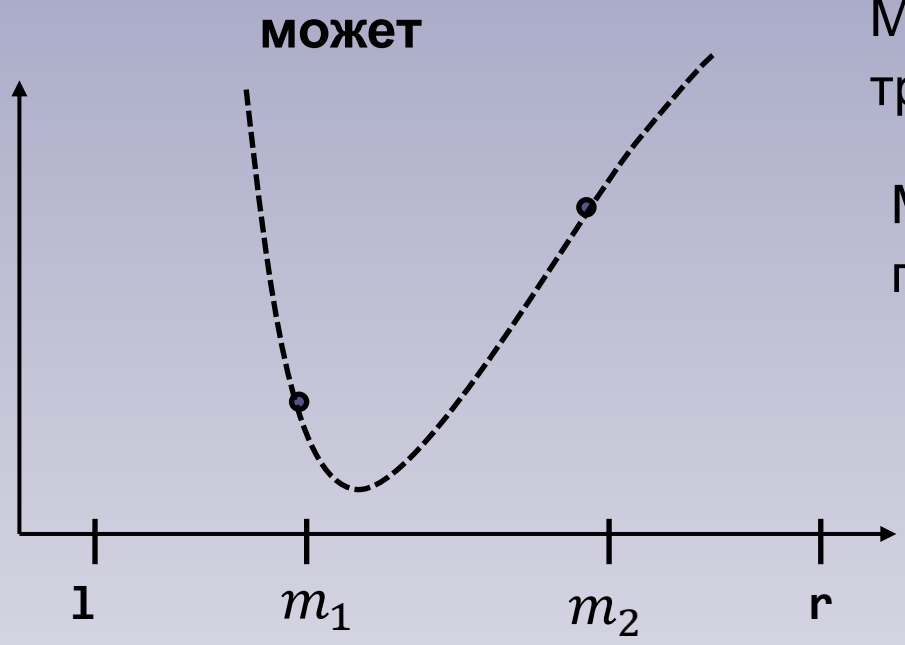
Поделим рассматриваемую область на **три** части

Вычислим значения функции в m_1 и m_2

Предположим, что $f(m_1)$ оказалось меньше, чем $f(m_2)$.

В какой из получившихся третей поиска может оказаться минимум?

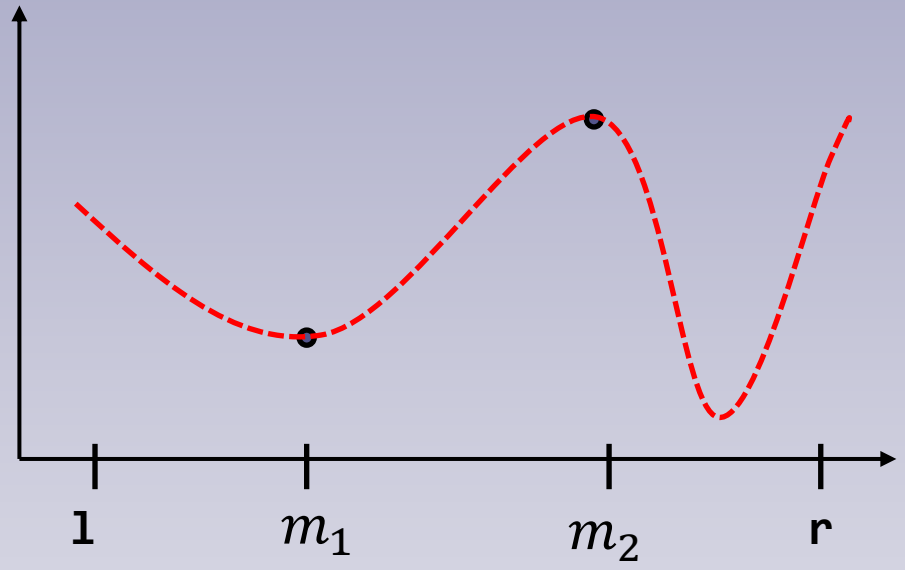
Может ли он оказаться в первой трети, между **`l` и m_1?**



Может ли он оказаться во второй трети, между m_1 и m_2**?**

Может ли минимум оказаться в последней трети, между m_2 и r **?**

Может ли минимум оказаться в последней трети, между m_2 и r **?**



В таком случае левая ветвь графика не будет строго монотонной, так как если от минимума функция поднималась до точки с абсциссой m_2, то дальше ей придется опускаться до точки с абсциссой m_1.

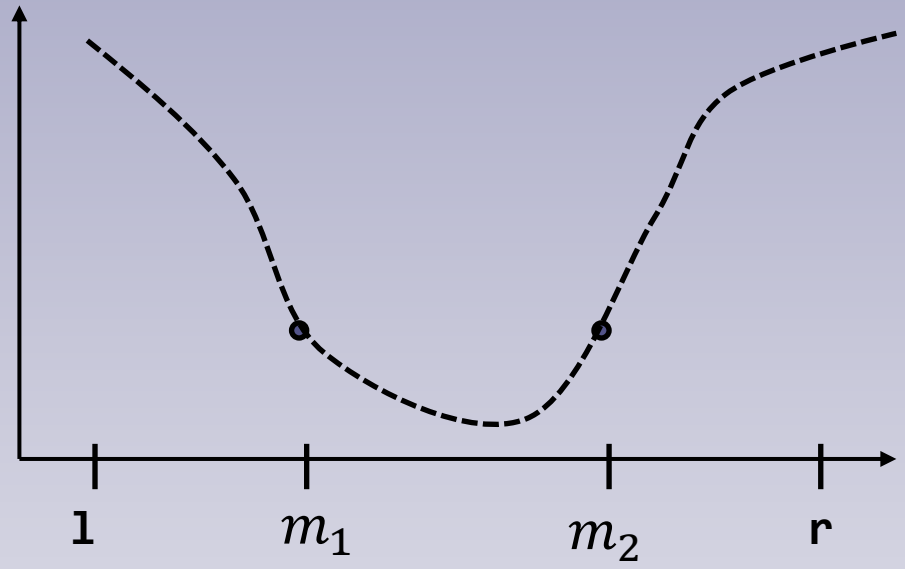
Ответ: не может

Следовательно, мы можем не продолжать наш поиск в последней трети.

Следовательно, мы можем сдвинуть указатель r в позицию m_2

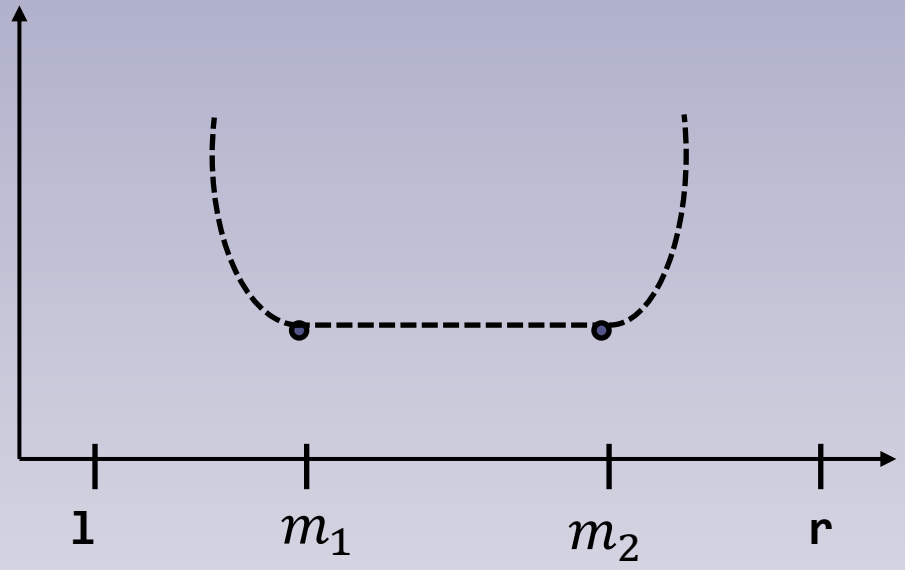
Проведя аналогичный анализ в случае, если $f(m_1)$ оказалось больше чем $f(m_2)$, можно прийти к выводу, что в этом случае можно передвинуть указатель `l` в позицию m_1 .

Что будет если $f(m_1) = f(m_2)$?



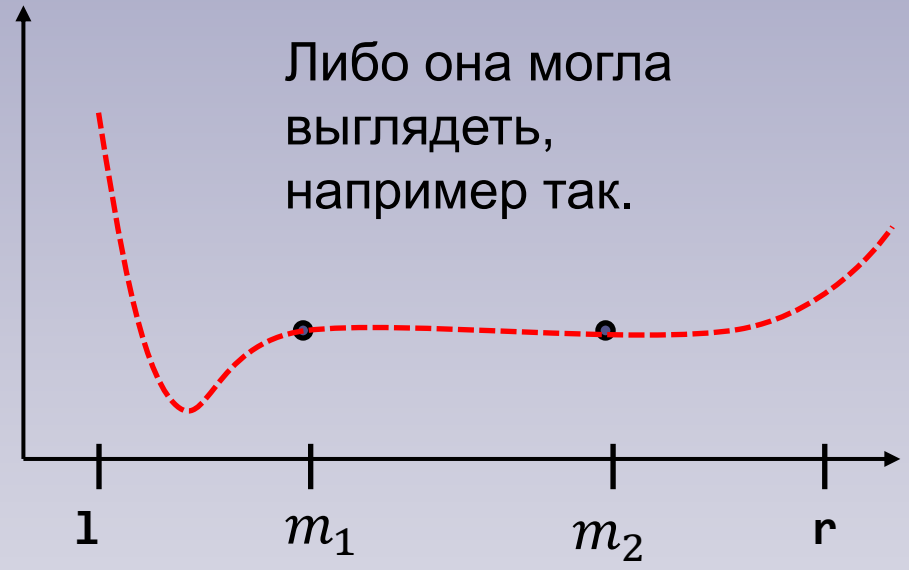
Если эти две точки принадлежали разным ветвям графика, значит минимум должен быть где-то между ними, следовательно, минимум может оказаться в центральной трети

Что будет если $f(m_1) = f(m_2)$?



Может случится так, что обе эти точки как раз принадлежат отрезку минимума, в этом случае нам также безопасно смещаться в центральную треть.

Могла ли возникнуть ситуация, когда минимум находится либо в правой, либо в левой трети?



Либо она могла выглядеть, например так.

Если это было бы так, то либо наша функция имела бы больше одного локального минимума, как на показанном выше красном графике.

Такая ситуация невозможна

Здесь правая ветвь графика не является строго монотонной, а мы договаривались, что на одной и той же ветви графика значения должны либо строго убывать, либо строго возрастать.

Итак, если $f(m_1) = f(m_2)$, то это значит, что поиск можно продолжать только в центральной трети.

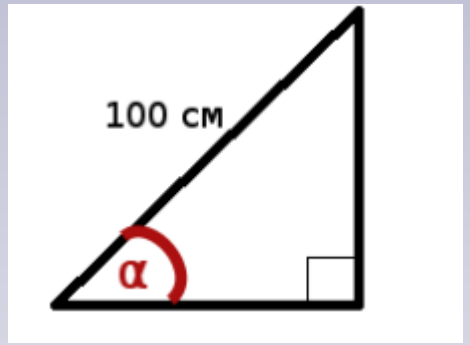
В реальности такая ситуация встречается очень редко, поэтому обработка данной ситуации не выносится в отдельный случай, а просто считают что при таком исходе безопасно перемещать как `l` в m_1, так и `r` в m_2.

$$m_1 = l + \frac{r - l}{3}$$

$$m_2 = r - \frac{r - l}{3}$$

```
function f(a,c:real):real;
begin
  a:=a*Pi/180.0;
  result:=0.5*c*c*sin(a)*cos(a);
end;
begin
  var l,r,m1,m2,c:real;
  c:=100;
  l:=0; r:=90;
  for var i:=1 to 100 do begin
    m1:=l+(r-l)/3.0;
    m2:=r-(r-l)/3.0;
    if(f(m1,c)<f(m2,c)) then l:=m1 else r:=m2;
  end;
  writeln(l);
end.
```

Имеется прямоугольный треугольник, у которого гипотенуза равна 100 сантиметрам. Нужно вычислить при каком угле α площадь треугольника будет максимальна.



Для двумерного случая делают так. Сначала мы пишем решение для фиксированной первой переменной (которая передаётся как параметр); при одной фиксированной переменной задача становится одномерной, и к ней можно просто применить описанный выше алгоритм. Теперь переходим к решению всей задачи: теперь мы тернарным поиском подбираем значение первой переменной и вызываем каждый раз решение с фиксированной переменной. Итого получается - один тернарный поиск, в который вложен другой.

Пусть есть `f(x,y)` - мат. функция 2х переменных.
`searchV(valX,…)` - функция которая ищет значение максимума функции f(x,y) по y, подставляя вместо x значение `valX`.
ее код аналогичен коду выше, только вместо `if (f (m1) < f (m2))` подставляется
`if (f (valX,m1) < f (valX,m2)`
`search(…){` - функция возвращает значение максимума функции `f(x,y)`
ее код тоже аналогичен коду выше, только вместо `if (f (m1) < f (m2))` подставляется
`if ( searchV(m1,…) < searchV(m2,…) ).`

Футбольные ворота

Соня, в отличие от многих студентов мат-меха, спортивна не только в программировании. В один прекрасный день она отправилась поиграть в футбол с друзьями. К сожалению, нигде поблизости не оказалось специально оборудованного футбольного поля, только высокая берёза одиноко красовалась в глубине двора.

Покопавшись дома в кладовке, Соня нашла две палки и решила соорудить футбольные ворота из палок и берёзы. Конечно, берёза будет использована как одна из боковых стоек ворот. Осталось сделать из двух палок вторую стойку и перекладину.

Соня, конечно, хочет забить как можно больше голов. Поэтому она решила сделать ворота максимальной площади. Стандартные футбольные ворота имеют прямоугольную форму, но Соня — человек креативный, и она считает, что ворота могут иметь форму произвольного четырёхугольника.

Можно считать, что берёза является отрезком прямой и растёт строго перпендикулярно земле.

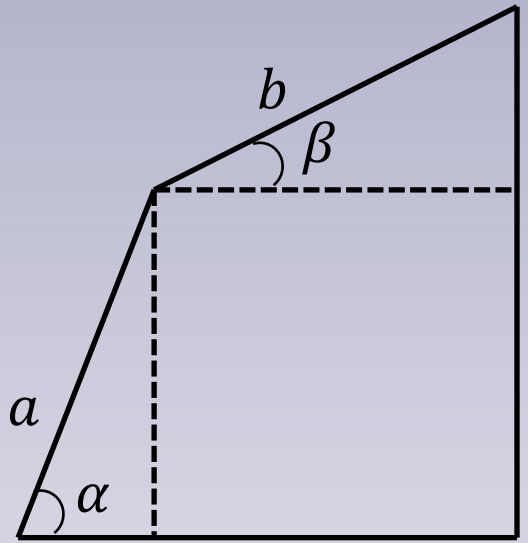
Исходные данные

В единственной строке записаны целые числа a, b — длины палок ($1 \le a, b \le 10\,000$). Известно, что суммарная длина палок строго меньше высоты берёзы.

Результат

Выведите максимальную площадь ворот, которые можно соорудить из палок и берёзы. Ответ следует вывести с точностью не менее шести знаков после десятичной точки.

Найти максимальную площадь ворот можно, подобрав углы для палок α и β.



Сперва мы находим углы α_1 и α_2, которые разбивают область поиска на три равные части. Далее для угла α_1 с помощью вложенного тернарного поиска ищем угол β_1, при котором площадь фигуры максимальна, аналогично для угла α_2 находим подходящий угол β_2

```
begin
  var l,r,m1,m2,a,b:real;
  a:=2;
  b:=2;
  l:=0; r:=90;
  for var i:=1 to 100 do begin
//по alpha
    m1:=l+(r-l)/3.0;
    m2:=r-(r-l)/3.0;
    if(f(m1,a,b)<f(m2,a,b))
then l:=m1 else r:=m2;
  end;
  writeln(f((l+r)/2,a,b):3:9);
end.
```

```
function f(alpha,a,b:real):real;
var l1,r1,m11,m22:real;
begin
  l1:=0; r1:=90;
  for var i:=1 to 100 do begin
    m11:=l1+(r1-l1)/3.0;
    m22:=r1-(r1-l1)/3.0;

if(square(alpha,m11,a,b)<square(alpha,m22,a,b))
then l1:=m11 else r1:=m22;
  end;
  result:=square(alpha,(l1+r1)/2.0,a,b);
end;
```

```
function square(alpha,beta,a,b:real):real;
begin
 alpha:=alpha*Pi/180.0;
 beta:=beta*Pi/180.0;
result:=0.5*a*a*sin(alpha)*cos(alpha)+
0.5*b*b*sin(beta)*cos(beta)+a*b*sin(alpha)*cos(beta);
end;
```

Output Window

```
4.828427125
```