Алгоритм бинарного поиска

Бинарный поиск применяется для решения произвольных уравнений вида $f(x) = 0$ при условии, что известны две точки a и b такие, что $f(a) \le 0$, $f(b) \ge 0$ и функция $f(x)$ непрерывна на интервале $[a; b]$

Идея решения очень проста. Берем точку c на середине отрезка $[a; b]$

$$c = \frac{a+b}{2}$$

и находим значение $f(c)$. Если $f(c) < 0$ то продолжаем поиск на интервале $[c, b]$, иначе продолжаем поиск на интервале $[a, c]$.

За один шаг алгоритма длина интервала уменьшается в 2 раза, поэтому после k шагов длина отрезка будет равна $\frac{b-a}{2^k}$, что позволит нам найти приближенное решение уравнения с любой заранее заданной точностью. Для оценки количества итераций можно воспользоваться приближенным равенством $2^{10} \approx 10^3$. Таким образом, за 100 итераций можно получить точность $(b-a)10^{-30}$, чего достаточно для решения практически любой задачи.

Замечание 1

Если уравнение имеет несколько решений на заданном интервале, то алгоритм найдет одно из них.

Замечание 2

Если функция убывает, то есть $f(a) \ge 0$ и $f(b) \le 0$, то можно перейти к функции $f'(x) = -f(x)$ и выполнить алгоритм для функции $f'(x)$

Реализация

При реализации алгоритма на одном из языков программирования можно хранить интервал поиска в переменных `L` и `R`. Тогда основная часть алгоритма примет вид

```
L:=a;
R:=b;
for i:=1 to 100 do begin
    C:=(L+R)/2;
    V:=... {вычисляем f(C)}
    if V<0 then
        L:=C
    else
        R:=C;
end;
writeln(L);
```

Целочисленный бинарный поиск

Целочисленный бинарный поиск применяется в том случае, если известна целочисленная функция $f(i)$, определенная на всех целых числах полузакрытого интервала $[a : b)$ у которого левая граница входит в интервал, а правая — нет. Функция должна быть неубывающей, то есть для всех i должно выполняться условие $f(i) \le f(i+1)$.

Функция f может задаваться аналитически, быть результатом вычисления некоторого алгоритма или же быть просто массивом упорядоченных по возрастанию значений.

Рассмотрим задачу нахождения нижней грани числа s, которая определяется как минимальное число k для которого $f(k) \ge s$. Если все значения функции на интервале $[a; b)$ меньше s, тогда нижней гранью считается правая граница интервала — число b.

Реализация

При реализации алгоритма интервал поиска $[L; R)$ будет храниться в двух переменных. При этом будем стремиться к тому, чтобы в ходе работы всегда выполнялись два условия $f(L-1) < s$ и $f(R) \geq s$. Формально будем считать, что $f(a-1) = -\infty$ и $f(b) = +\infty$. Очевидно, что если данные условия будут выполнены и будет выполнено $L = R$, то число R и будет нижней гранью s.

Тогда основная часть алгоритма примет вид

```
L:=a;
R:=b;
while L<R do begin
    C:=(L+R) div 2;
    V:= ... {вычисляем f(C)}
    if V<s then
        L:=C+1
    else
        R:=C;
end;
```

Легко проверить, что именно такой выбор условия внутри цикла позволяет выполнить свойства $f(L-1) < s$ и $f(R) \geq s$, что и требовалось.

Тернарный поиск

Тернарный поиск применяется для нахождения минимума функции $f(x)$ в интервале $[a; b]$ при условии, что функция на этом интервале сначала убывает, проходит через минимум, потом возрастает.

В отличие от бинарного поиска, интервал $[a; b]$ делится на три равные части точками c_1 и c_2 по формулам $c_1 = \frac{2a+b}{3}$, $c_2 = \frac{a+2b}{3}$ Далее вычисляются значения функции в точках c_1 и c_2. Если $f(c_2) > f(c_1)$, то поиск продолжается в интервале $[a; c_2]$, иначе в интервале $[c_1; b]$.

Алгоритм работает корректно, так как если точки c_1 и c_2 находятся по разные стороны от минимума, то вне зависимости от выполнения условия минимум останется в интервале поиска. Если минимум находится справа от c_2 то функция убывает на отрезке от a до c_2 и $f(c_1) > f(c_2)$, и минимум останется в интервале поиска $[c_1; b]$.

Аналогично, если минимум находится слева от c_1 то функция возрастает на отрезке от $_1$ до b и $f(c_1) < f(c_2)$, и минимум останется в интервале поиска $[a; c_2]$.

За один шаг алгоритма длина интервала уменьшается в $\frac{2}{3}$ раза, а за два шага в $\frac{4}{9}$ раза. Поскольку $\frac{4}{9} < \frac{1}{2}$ алгоритм сходится примерно в два раза медленнее, чем простой бинарный поиск. Таким образом, в большинстве случаев для получения ответа будет достаточно 200 итераций.

Замечание 1

Алгоритм может работать некорректно, если функция на отрезке содержит более одного интревала непрерывного убывания или возрастания.

Замечание 2

Для нахождения максимума функции в заданном интервале можно перейти к функции $f'(x) = -f(x)$ и выполнить алгоритм для функции $f'(x)$

Реализация

При реализации алгоритма на одном из языков программирования можно хранить интервал поиска в переменных `L` и `R`. Тогда основная часть алгоритма примет вид

```
L:=a;
R:=b;
for i:=1 to 100 do begin
    C1:=(2*L+R)/3;
    C2:=(L+2*R)/3;
    V1:=... {вычисляем f(C1)}
    V2:=... {вычисляем f(C2)}
    if C1>C2 then
        L:=C1
    else
        R:=C2;
end;
writeln(L);
```