УДК 004

Поиск с использованием золотого сечения на языке программирования C++

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен алгоритм поиска с применением золотого сечения. Объяснена работа алгоритма и описана его реализация на языке программирования C++, а также вычислена эффективность работы алгоритма и сравнена с алгоритмом троичного поиска.
Ключевые слова: поиск с помощью золотого сечения, троичный поиск, сортировка, алгоритмизация, алгоритмы поиска, C++

Golden Section search in the C++ programming language

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article deals with the search algorithm using the Golden Ratio. The work of the algorithm is explained and its implementation in the C++ programming language is described, and the efficiency of the algorithm is calculated and compared with the ternary search algorithm.
Keywords: golden section search, ternary search, sorting, algorithmization, search algorithms, C++

1. Введение

1.1 Актуальность

Алгоритмы поиска остаются актуальными уже долгое время. Связано это со скоростью развития вычислительных средств. В настоящее время многие сферы деятельности человека так или иначе подразумевают работу с большими наборами данных. В перечень действий с данными входит поиск. Часто необходимо за короткий срок найти ту или иную запись в наборе и для этого существуют алгоритмы поиска.

Алгоритмы поиска также развиваются вслед за технологиями и создание новых или улучшение уже существующих являются не менее важной задачей в настоящее время.

1.2 Обзор исследований

Я.Е. Ромм и Д.А. Чабанюк описали реализацию параллельного построения двоичного дерева на основе сортировки [1]. А.М. Глухов представил сравнение линейного, бинарного и интерполирующих алгоритмов поиска [2]. О.Б. Попова и Н.В. Богацкий описали способы реализации двоичного дерева поиска и длинной арифметики на языке программирования C# [3]. Т.С. Бадасян и С.К. Авагян рассмотрели красно-чёрное дерево поиска, а также его свойства, реализацию с использованием программирования с последующей балансировкой [3].

1.3 Цель исследования

Цель – реализовать алгоритм поиска с использованием золотого сечения для заранее заданной математической функции.

2. Материалы и методы

Для реализации поставленной задачи будет использован язык программирования C++.

3. Результаты и обсуждения

Самые популярные алгоритмы поиска: линейный и бинарный. Популярны они за счёт своей простоты, но при этом имеют недостатки. Так, алгоритм линейного поиска выполняется слишком долго и не подходит для работы с большими данными, так как предполагается, что алгоритм будет сравнивать все элементы массива с ключом (значение, которое нужно найти в массиве или любом другом наборе данных). И не смотря на проблему времени выполнения, алгоритм может подойти для решения небольших задач.

Решить проблему скорости выполнения алгоритма должен бинарный поиск. И данный алгоритм намного быстрее линейного (O($log_2 n$) против O(n), соответственно), но проблема алгоритма заключается в требованиях к входным данным: при базовой реализации алгоритм может работать только с отсортированными данными, что сокращает количество случаев, когда он может пригодиться.

Линейный алгоритм может использоваться в сортировке, но из-за своей скорости выполнения не применяется. Бинарный также может быть использован, но в сортировке нет необходимости, если входные данные уже отсортированы.

Помимо линейного и бинарного алгоритмов, существуют и другие методы поиска. Одним из таких является троичный поиск. Троичный поиск – алгоритм нахождения минимума или максимума на графике функции. Часто применяется в математическом анализе как раз для поиска экстремумов, в информатике не получил широкое распространение. Суть алгоритма состоит в следующем:

1. Выбираются границы поиска (это может быть, как вся функция (но в таком случае алгоритм может долго работать из-за), так и её отрезок) и

вычисляются положения точек A и B по определённым математическим формулам;

2. Вычисляются значения функции в полученных точках и сравниваются;

3. Если f(a) больше f(b), то выбирается средняя и правая части функции. Границы поиска изменяются (точка A становится левой границей, а правая граница при этом не изменяется) и шаги 1 и 2 повторяются;

4. Если f(a) меньше f(b), то выбирается средняя и левая части функции. Границы поиска изменяются (точка B становится правой границей, а левая граница при этом не изменяется) и шаги 1 и 2 повторяются;

5. Шаги 1-4 выполняются до тех пор, пока не будет достигнута ранее заданная мера точности ε, то есть пока не будет выполнено условие “r – l <= ε”, где r и l – границы поиска.

Алгоритм не сложен в реализации и хорошо работает при рекурсивном подходе. Главная проблема данного алгоритма: скорость выполнения. Троичный поиск демонстрирует хорошее время работы, но из-за необходимости на каждом шаге вычислять значения в точках A и B время увеличивается в 2 раза тоже: $2log_{\frac{3}{2}}(\frac{r-l}{e})$. При больших данных алгоритм не уступает бинарному поиску. Также троичный поиск не требует работы с уже отсортированными данными, но при этом алгоритм не гарантирует нахождения минимального значения среди всех элементов массива. На граничных случаях (отсортированный согласно условию и согласно обратному условию) алгоритм работает быстрее линейного и двоичного поиска за счёт того, что исходный массив делится на три части и значительно сужает область поиска. Описать алгоритмически троичный поиск для массива целых чисел можно следующим образом:

```
if (r-l > e) {
vector<int> new_arr;
if (arr[a] > arr[b]) {
     for (int i=a;i<r+1;i++) new_arr.push_back(arr[i]);
          TernarySearch(new_arr,0,new_arr.size(),e);
}
else {
     for (int i=l;i<b+1;i++)  new_arr.push_back(arr[i]);
     TernarySearch(new_arr,0,new_arr.size(),e);
}
}
else {
int min=arr[0];
for (int i=0;i<arr.size();i++) {
     if (arr[i] < min) min = arr[i];
}
cout << min << endl;
}
```

Для решения проблемы со скоростью выполнения поиска ввели золотое сечение.

Золотое сечение – математическое определение, характеризующее отношение частей и целого, при котором отношения частей между собой и каждой части к целому равны. Золотое сечение встречается повсеместно: в науке, быту, природе. Золотое сечение предполагает деление набора в определённом отношении. В процентном значении золотое сечение – это деление набора или отрезка в отношении 62% и 38%.

Использование золотого сечения позволяет улучшить время работы алгоритма троичного поиска за счёт того, что исчезает необходимость вычисления значения в двух точках (кроме первой итерации, когда набор данных надо разделить на 3 части).

Алгоритм поиска состоит из следующих шагов:

1. Сначала определяются границы поиска l и r, а после вычисляются x1 и x2 по следующим формулам: $x1 = l + \frac{r-l}{\varphi+1}$ и $x2 = r - \frac{r-l}{\varphi+1}$, где φ это обратное золотое число. Золотое число является основной характеристикой при делении золотым сечением. Золотое число Ф вычисляется по следующей формуле: $\frac{1+\sqrt{5}}{2}$ и является числом с неопределённым количеством цифр в остатке. Принято брать первые 5 цифр после запятой и в таком случае $\Phi = 1.61803$. В формулах для x1 и x2 также необходимо использовать золотое число, которое также равно сумме обратного числа и единицы, потому что обратное золотое число $\varphi = 0.61803$.
2. После нахождения x1 и x2 вычисляются значения функции в этих точках, и они сравниваются. Если f(x1) > f(x2), то левая граница сдвигается к x1, а после x2 вычисляется по той ж формуле, что описана в первом шаге. Если же f(x1) < f(x2), то правая граница сдвигается к x2, а x1 вычисляется по формуле, которая указана в первом шаге.
3. После изменения границы, проверяется выполнение условия достижения значения меры точности ε. Данная величина характеризует точность поиска и чем она меньше, тем ближе алгоритм будет к минимальному значению (если $\varepsilon = 1$, то поиск остановится, когда будет найдено конкретное значение). Достижение меры точности проверяется с помощью следующего неравенства: $r - l < \varepsilon$. Если данное неравенство выполняется, то алгоритм выбирает значение и прекращает свою работу. Если нет, то шаги повторяются вновь.

По завершении работы алгоритма выбирается точка x и значение в ней. Точку выбирать необходимо по следующей формуле: $x = \frac{l+r}{2}$. Значение в этой точке и будет искомым минимумом.

Реализовать такой алгоритм в итеративной форме, без применения рекурсии. Так как алгоритм применим для математических функций, то можно заранее задать функцию, в которой будет происходить поиск, например $y = \frac{14*\tan(x)}{15*x}$:

```
double f(double n) {
     return (14*tan(n))/(15*n);
}
```

Вычисление x1 и x2 может быть описано следующим образом:

```
double x1 = r - (r - l) / 1.61803;
double x2 = l + (r - l) / 1.61803;
```

Перед началом вычисления новых границ, нужно проверить – выполняется ли неравенство $r - l < \varepsilon$. В случае невыполнения нам достаточно будет вычислить X по заданной формуле и вывести значение функции в этой точке. Для удобства стоит реализовать вычисления в цикле while, внутри которого выполняется проверка значений функции в точках x1 и x2:

1. Если f(x1) > f(x2), то l = x1, x1 = x2, x2 = r – (x1 – l);
2. Если f(x1) < f(x2), то r = x2, x2 = x1, x1 = l + (r – x2).

```
while (r - l > e) {
     if (f(x1) > f(x2)) {
          l = x1;
          x1 = x2;
          x2 = r - (x1 - l);
     }
     else {
          r = x2;
          x2 = x1;
          x1 = l + (r - x2);
     }
}
```

После остаётся вычислить X и вывести ответ пользователю:

```
double x = (x1 + x2) / 2;
```

Алгоритм можно считать завершённым и начинать тестировать. Для функции $y = \frac{14*\tan(x)}{15*x}$ выбраны следующие входные параметры:

1. l = -1
2. r = 1
3. ε = 0.001

При заданном ε минимальное значение будет 0.933 в точке -0.00603821 (см. рис. 1).

№	l	r	r – l	x1	x2	f(x1)	f(x2)
1	-1	1	2	-0.236	0.236	0.951	0.951
2	-1	0.236	1.236	-0.528	-0.236	1.031	0.951
3	-0.528	0.236	0.764	-0.236	0.056	0.951	0.934
4	-0.236	0.236	0.472	-0.056	0.056	0.934	0.934
5	-0.236	0.056	0.292	-0.125	-0.055	0.938	0.934
6	-0.125	0.056	0.180	-0.055	-0.013	0.934	0.933
7	-0.056	0.056	0.111	-0.013	0.013	0.933	0.933
8	-0.056	0.013	0.069	-0.029	-0.013	0.934	0.933
9	-0.029	0.013	0.043	-0.013	-0.003	0.933	0.933
10	-0.013	0.013	0.026	-0.003	0.003	0.933	0.933
11	-0.013	0.003	0.016	-0.007	-0.003	0.933	0.933
12	-0.007	0.003	0.010	-0.003	-0.002	0.933	0.933
13	-0.003	0.003	0.006	0.002	0.002	0.933	0.933
14	-0.003	0.002	0.005	0.0004	-0.002	0.933	0.933
15	-0.003	0.002	0.0012	-0.005	0.0004	0.933	0.933
16	-0.005	-0.002	0.003	0.0004	-0.007	0.933	0.933
17	-0.0002	0.0007	0,0009	0,0001	0,0004	0,933	0,933

Рисунок 1. Результат работы алгоритма при конкретных входных значениях

Теперь можно оценить эффективность работы алгоритма. Как ранее было указано: время работы троичного поиска составляет $2log_{\frac{3}{2}}(\frac{r-l}{\varepsilon})$, что является хорошим результатом, но алгоритм может показать себя намного слабее золотого сечения при больших значениях. Время работы алгоритма поиска с помощью золотого сечения составляет $log_{Ф}\frac{r-l}{\varepsilon}$ и связано это с несколькими факторами: алгоритм не вычисляет значения в двух точках на каждой итерации, вместо этого он сдвигает один край области поиска и вычисляет новое значение за итерацию. Область поиска уменьшается в Ф раз на каждой итерации, то есть в 1.61803 раз (что больше троичного поиска, но меньше бинарного). Таким образом, поиск с помощью золотого сечения требует меньше приблизительно в 2.37 раз вычислений, чем в троичном поиске.

Алгоритм, как и троичный поиск, не устойчив при работе с целочисленными массивами и в своей стандартной реализации не подходит для сортировки (можно использовать в качестве гибрида с другими сортировками, но нахождение минимума среди элементов массива не гарантируется).

В статье был рассмотрен алгоритм поиска с использованием золотого сечения, а также представлена его реализация. Также была оценена эффективность работы алгоритма и сравнена с эффективностью троичного поиска.

Библиографический список

1. Ромм Я.Е., Чабанюк Д.А. Параллельное построение двоичного дерева на основе сортировки // Вестник Донского государственного технического университета. 2018. Т. 18. № 4. С. 449-454.
2. Глухов А.М. Сравнительный анализ трудоемкости алгоритмов поиска в программировании // Академия педагогических идей новация. Серия: студенческий научный вестник. 2018. № 5. С. 232-242.
3. Попова О.Б., Богацкий Н.В. Способы организации двоичного дерева поиска и длинной арифметики на языке программирования C#. Петрозаводск: Международный центр научного партнерства «Новая Наука» (ИП Ивановская И.И.), 2020. С.116-128.
4. Бадасян Т.С., Авагян С.К. Красно-чёрное дерево: балансирование и сложность // Наука, техника и образование. 2020. № 3 (67). С. 26-31.
5. Поиск с помощью золотого сечения – Викиконспекты. URL: https://neerc.ifmo.ru/wiki/index.php?title=Поиск_с_помощью_золотого_сечения. - Дата обращения: 25.06.2022.
6. Метод золотого сечения. Симметричные методы. URL: http://www.machinelearning.ru/wiki/index.php?title=Метод_золотого_сечения._Симметричные_методы. - Дата обращения: 25.06.2022.