Алгоритм "Spigot" для нахождения числа π

Общее описание. Преимущества. Недостатки.

Алгоритм основан на представлении чисел в смешанной системе счисления. Некоторые иррациональные числа становятся периодическими в таких системах. Подробное описание этого приводится ниже. Основным преимуществом алгоритма является то, что он использует только целочисленную арифметику для получения каждого знака. Вероятно, поэтому авторы алгоритма пназвали его именно так (spigot англ. "кран"). Каждое число вытекает, подобно каплям из "крана". Также к преимуществам можно отнести еще размер программы. Основным недостатком является то, что диапазон работы алгоритма ограничен памятью, но для данной задачи этот недостаток не проявляется. Итак, чтобы понять принцип работы алгоритма, разберём более простой пример.

Система счисления в которой знаки числа e периодические.

Простейшим примером смешанной системы счисления является дата. T.e d дней, h часов, m минут, s секунд будут представлены как $d \cdot 24 \cdot 60 \cdot 60 + h \cdot 60 \cdot 60 + m \cdot 60 + s$ секунд.
Теперь ближе к делу. Как хорошо известно, число можно представить в виде многочлена. Например,

$$\sqrt{2} = 1.41421356... = 1 + \frac{1}{10}(4 + \frac{1}{10}(1 + \frac{1}{10}(4 + \frac{1}{10}(2 + \frac{1}{10}(1 + ...)))))$$

T.e. сейчас мы имеем представление числа в системе $(\frac{1}{10}, \frac{1}{10}, \frac{1}{10}, ...)$. Теперь если мы сменим это представление на представление в системе $\mathbf{b} = (\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, ...)$. Тогда имеем представление корня из 2.

$$a_0 + \frac{1}{2}(a_1 + \frac{1}{3}(a_2 + \frac{1}{4}(a_3 + \frac{1}{5}(a_4 + \frac{1}{6}(a_5 + ...)))))$$

Где a_i неотрицательные целые числа. Если $0 <= a_i <= i$ для $i >= 1$, то представление называется регулярным. В такой системе число имеет вид $(a_0, a_1, a_2, ...)_\mathbf{b}$. Каждому числу сопоставляется такое представление. При этом имеет место важное замечание. Мы исключаем представления, которые заканчиваются максимальными знаками. T.e, например, $\frac{1}{2} = (0, 1, 0, 0, ...)_\mathbf{b} = (0, 0, 2, 3, 4, 5, 6, ...)_\mathbf{b}$. Первое представление мы исключаем.

Лемма 1.(без доказательства)
(a) Если $i >= 1$, $(0, 0, 0, ..., 0, a_i, a_{i+1}, ...)$, $b < \frac{1}{i!}$, в частности, $(0, a_1, a_2, a_3, a_4, ...)_b < 1$.
(b) Представления чисел использующие систему b единственные.
(c) Целая часть $(a_0, a_1, a_2, a_3, a_4, ...)_b$ a_0, а дробная $(0; a_1, a_2, a_3, a_4, ...)$

В этой системе, некоторые иррациональные числа становятся периодическими. Например, число $e = (2; 1, 1, 1, ...)_b$; (это просто иная запись ряда $\sum \frac{1}{i!}$)

$$1 + \frac{1}{1}(1 + \frac{1}{2}(1 + \frac{1}{3}(1 + \frac{1}{4}(1 + \frac{1}{5}(1 + ...))))).$$

Десятичные знаки вещественного числа x, могут быть получены взятием целой части от x, умножения его дробной части на 10, взятия целой части числа, умножения дробной части и т.д. Если $x = (a_0; a_1, a_2, ..., a_n)_\mathbf{b}$, тогда $10x = (10a_0; 10a_1, 10a_2, ..., 10a_n)$. Для того, чтобы привести полученное число к регулярному виду, будем уменьшать i-е число по модулю i. Начиная эти сокращения справа, перетаскиваем полученные частные влево и так до 0-ого элемента (последнее частное и будет являться искомым знаком). Таким образом получим регулярное представление числа $10x$ в этой системе. и так далее, опять домножим текущее представление на 10, затем полученное снова домножим на 10... Демонстрация для первых трех чисел приведена в таблице ниже.

"Spigot"для знаков числа π. Для числа π можно провести подобные рассуждения (с некоторыми отличиями). Отправной точкой здесь будет являться следующий ряд:

$$\pi = \sum_{i=0}^{\infty} \frac{(i!)^2 2^{i+1}}{(2i+1)!}$$

Который может быть получен из формулы Валлиса для π. Если этот ряд немного преобразовать, то можно получить, что:

$$\frac{\pi}{2} = \sum_{i=0}^{\infty} \frac{i!}{(2i+1)!!} = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + ...$$

Base 10		1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11
2.718281826...	**2**	1	1	1	1	1	1	1	1	1	1
7.18281826...		10	10	10	10	10	10	10	10	10	10
carries	**7**	±3	±3	±2	±1	±1	±1	±1	±1	±0	--
		14	13	12	11	11	11	11	11	10	10
0.18281826...		0	1	0	1	5	4	3	2	0	10
1.8281826...		0	10	0	10	50	40	30	20	0	100
carries	**1**	±3	±0	±3	±9	±6	±4	±2	±0	±9	--
		3	10	3	19	56	44	32	20	9	100
0.8281826...		1	1	3	4	2	2	0	2	9	1
8.281826...		10	10	30	40	20	20	0	20	90	10
carries	**8**	±6	±9	±8	±3	±2	±0	±3	±9	±0	--
		16	19	38	43	22	20	3	29	90	10
0.281826...		0	1	2	3	4	6	3	2	0	10
2.81826...		0	10	20	30	40	60	30	20	0	100
carries	**2**	±5	±6	±7	±8	±9	±4	±2	±0	±9	--
		5	16	27	38	49	64	32	20	9	100
0.81826...		1	1	3	3	1	1	0	2	9	1

Рис. 1: Здесь жирным выделены знаки числа e. Сокращения выполняются по модулю i. Первый знак результат аппроксимации.

Что также можно представить в виде:

$$\frac{\pi}{2} = 1 + \frac{1}{3}(1 + \frac{2}{5}(1 + \frac{3}{7}(1 + \frac{4}{9}(1 + ...))))$$

Таким образом получаем новую смешанную систему счисления $\mathbf{c} = (\frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{4}{9}, ...)$, в которой π представляется в виде $(2, 2, 2, 2, ...)_{\mathbf{c}}$. Для того, чтобы представление было регулярным, необходимо, чтобы знак на i-ом месте лежал в отрезке $[0, 2i]$. К сожалению, это еще не все.

Лемма 2. (без доказательства)
Представление $(0, 2, 4, 6, 8, ...)_{\mathbf{c}} = 2$. Следовательно регулярные представления вида $(0, a, b, c, ...)_c$ лежат в пределах 0 и 2.

Лемма 2 означает, что представления в $\mathbf{c}$ не единственны. К примеру, $(0; 2, 4, 6, 8, ...)_{\mathbf{c}} = 2 - \frac{2}{3} = \frac{4}{3}$, в то время как $(0; 0, 2, 3, 4, ...)_{\mathbf{c}} = \frac{2}{3} = (0; 2, 0, 0, 0, ...)_{\mathbf{c}}$.
Целая часть $(a_0; a_1, a_2, ...)_{\mathbf{c}}$ может быть a_0 или $a_0 + 1$, в зависимости от того $(0; a_1, a_2, ...)_{\mathbf{c}}$ принадлежит $[0, 1)$ или $[1, 2)$. Речь идет о тех ситуациях, когда знак числа π получается равным 10, в таком случае на i-ом месте, конечно, будет стоять 0, но необходимо и увеличить на единицу предшествующие знаки, а если предшествующий знак был равен 9, то и на его место вписать 0. Продолжать это действие до того, пока не встретится отличный от 9 знак. Теперь можно смело написать алгоритм.

Алгоритм. π-spigot.

- *Инициализация:* Массив остатков remainders $= (2, 2, 2, ..., 2)$ (размер массива $[10n/3]$, можно доказать, что этого достаточно)

- Повторить n раз:
  Умножить на 10: умножить каждое число в remainders на 10.
  Привести remainders к регулярному виду: на i-ом месте вычисляем частное (quotient) при делении

(remainders[i]+ перенос с предыдущего разряда) на $2i-1$ (см. **c**), вычисляем остаток от деления этой суммы. Оставляем остаток в remainders[i]. Для вычисления переноса умножаем частное (quotient) на $(i-1)$ (числитель соответствующей дроби в **c**).

- Далее если частное получилось равным 10, то в res[i] пишем 0, а знак до него увеличиваем на 1, если этот знак равен 9, то тогда пишем и на это место 0, продолжаем пока не встретим знак отличный от 9.

Нахождение первых знаков числа π проиллюстрировано в таблице ниже.

	Digits of π		1/3	2/5	3/7	4/9	5/11	6/13	7/15	8/17	9/19	10/21	11/23	12/25
Initialize		2	2	2	2	2	2	2	2	2	2	2	2	2
× 10		20	20	20	20	20	20	20	20	20	20	20	20	20
Carry	3	+10	+12	+12	+12	+10	+12	+7	+8	+9	+0	+0	+0	=
		30	32	32	32	30	32	27	28	29	20	20	20	20
Remainders		0	2	2	4	3	10	1	13	12	1	20	20	20
× 10		0	20	20	40	30	100	10	130	120	10	200	200	200
Carry	1	+13	+20	+33	+40	+65	+48	+98	+88	+72	+150	+132	+96	=
		13	40	53	80	95	148	108	218	192	160	332	296	200
Remainders		3	1	3	3	5	5	4	8	5	8	17	20	0
× 10		30	10	30	30	50	50	40	80	50	80	170	200	0
Carry	4	+11	+24	+30	+40	+40	+42	+63	+64	+90	+120	+88	+0	=
		41	34	60	70	90	92	103	144	140	200	258	200	0
Remainders		1	1	0	0	0	4	12	9	4	10	6	16	0
× 10		10	10	0	0	0	40	120	90	40	100	60	160	0
Carry	1	+4	+2	+9	+24	+55	+84	+63	+48	+72	+60	+66	+0	=
		14	12	9	24	55	124	183	138	112	160	126	160	0

Рис. 2: Нахождение первых четырёх знаков

Возможные улучшения.

Алгоритм можно ускорить благодаря, выводу более, чем по одному знаку за итерацию, однако, тогда необходимо также увеличивать еще и размер выделяемой памяти. Более того, если взять основание, например, 100000, то можно получить удивительно короткую программу (что-то около 15-ти строчек), если быть уверенным, что пределы вычислений в которых будет работать программа не содержит комбинации знаков 00000.