Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Иркутский государственный университет»
(ФГБОУ ВО «ИГУ»)

СБОРНИК СТАТЕЙ

Материалы международной конференции

Иркутск, 2022

Печатается по разрешению редакционно-издательского совета
Иркутского государственного педагогического университета

УДК 512.7+121-564.7
ББК 23.42+25.21-23.12

Сборник статей. Материалы международной конференции. — Иркутск, издательство ГОУ ВПО «Иркутский государственный педагогический университет», 2004.— 28с.

Содержание

Теория чисел

Бухштаб А. А. *

Аннотация. Способы разложения числа e в цепную дробь путём приведения числа к уровнению пределов.

Ключевые слова: Теория чисел, дробь, цепная дробь.

Разложение числа e в цепную дробь

В качестве примера рассмотрим разложение в цепную дробь числа e.

Теорема 1.

$$\frac{e+1}{e-1} = 2 + \sqrt[1]{6} + \sqrt[1]{10} + \ldots + \sqrt[1]{4n+2} + \ldots$$

Доказательство. Определим $f_n(x)$ $(n = 0,\ 1,\ 2,\ \ldots)$ как сумму ряда:

$$f_n(x) = \frac{n!}{(2n)!} + \frac{(n+1)!}{1!(2n+2)!}x^2 + \frac{(n+2)!}{2!(2n+4)!}x^4 + \ldots = \sum_{s=0}^{\infty} \frac{(n+s)!}{s!2n+2s!}x^{2s}.$$

Этот ряд сходится при любых значениях x; однако мы будем рассматривать только значение x, лежащие в интервале $(0; 1)$.

Легко проверить что имеет место тождество

$$f_n(x) - (4n+2)f_{n+1}(x) = 4x^2 f_{n+2}(x). \tag{14}$$

Действительно, коэффициент при x^{2k} в левой части равенства (14) равен

$$\frac{(n+k)!}{k!(2n+2k)!} - (4n+2)\frac{(n+k+1)!}{k!(2n+2k+2)!} =$$
$$= \frac{(n+k)!}{k!(2n+2k)!}(1 - \frac{2n+1}{2n+2k+1}) = \frac{2(n+k)!}{(k-1)!(2n+2k+1)!}.$$

а в правой части равенства (14) он равен

$$\frac{4(n+k+1)!}{(k-1)!(2n+2k+2)!} = \frac{2(n+k)!}{(k-1)!(2n+2k+1)!}$$

*example@mail.com

так что (14) верно. Обозначим $\dfrac{f_n(\frac{1}{2})}{f_{n+2}(\frac{1}{2})}$ через a_n. В частности, поскольку

$$f_0(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots = \frac{1}{2}(e^x + e^{-x})$$

$$f_1(x) = \frac{1}{2x} + (x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots) = \frac{1}{4x}(e^x - e^{-x}).$$

то

$$\alpha_0 = \frac{f_0(\frac{1}{2})}{f_1(\frac{1}{2})} = \frac{e^{\frac{1}{2}} + e^{-\frac{1}{2}}}{e^{\frac{1}{2}} - e^{-\frac{1}{2}}} = \frac{e+1}{e-1}$$

Из тожественного равенства (14) при $x = \frac{1}{2}$ получаем:

$$\alpha = (4n+2) + \frac{1}{\alpha_{n+1}}. \tag{15}$$

Поскольку α_{n+1} положительно, равенство (15) показывает, что при всех n $\alpha_n > 4n+2 > 1, \dfrac{1}{\alpha_{n+1}} < 1$ т.е. $4n+2 = [\alpha_n]$ и последовательность отношений (15) при $n = 0, 1, 2 \ldots$

$$\begin{aligned}
\alpha_0 &= 2 + \frac{1}{\alpha_1} \\
\alpha_1 &= 6 + \frac{1}{\alpha_2} \\
\alpha_2 &= 10 + \frac{1}{\alpha_3} \\
&\ldots\ldots\ldots
\end{aligned}$$

даёт разложение α_0 в цепную дробь:

$$\frac{e+1}{e-1} = \alpha_0 = 2 + \sqrt[1]{6} + \sqrt[1]{10} + \ldots + \sqrt[1]{4n+2} + \ldots \tag{16}$$

Теорема 2.

$$e = 2 + \sqrt[1]{1} + \sqrt[1]{2} + \sqrt[1]{1} + \sqrt[1]{1} + \sqrt[1]{1} + \sqrt[1]{4} + \sqrt[1]{1} + \sqrt[1]{6} + \ldots \tag{17}$$

т.е. элемент α_n разложения e в цепную дробь имеют вид:

$$\alpha_0 = 2, \alpha_{3n} = \alpha_{3n+1} = 1 и \alpha_{3n-1} = 2n$$

Доказательство. Обозначим подходящие дроби к правой части (17) через $\frac{P_n}{Q_n}$, а подходящие дроби к (16) через $\frac{R_n}{S_n}(n = 0, 1, 2, \ldots)$. Докажем, что

$$\frac{R_n}{S_n} = \frac{P_{3n+1} + Q_{3n+1}}{P_{3n+1} - Q_{3n+1}}.$$

Принимая во внимание значение элементов цепной дроби (17), имеем:

$$P_{3n+1} = P_{3n} + P_{3n-1}, P_{3n} = P_{3n-1} + P_{3n-2},$$
$$P_{3n-1} = 2nP_{3n-2} + P_{3n+3},$$
$$P_{3n-2} = P_{3n-3} + P_{3n-4}, P_{3n-3} = P_{3n-4} + P_{3n-5},$$

откуда находим:

$$P_{3n+1} = 2P_{5n-1} + P_{3n+-2} = (4n+1)P_{3n-2} + 2P_{3n-3} =$$
$$= (4n+2)P_{3n-2} + P_{8n-3} - P_{8n-4} = (4n+2)P_{8n-2} + P_{3n-5}.$$

Аналогичное соотношение имеем и для Q_{3n+1}, так что

$$\left.\begin{aligned} P_{3n+1} &= (4n+2)P_{3n-2} + P_{3n-5} \\ Q_{3n+1} &= (4n+2)Q_{3n-2} + Q_{3n-5} \end{aligned}\right\} \tag{18}$$

Докажем индукцией по n, что

$$R_n = \frac{1}{2}(P_{3n+1} + Q_{3n+1}). \tag{19}$$

Из (16) и (17) непосредственно вычисляем $R_0 = 2, R_1 = 13, P_1 = 3, P_4 = 19, Q_1 = 1, Q_4 = 7$, так что соотношение (19) верно для всех R с номерами, меньшими чем n, где $n \geqslant 2$ т.е., в частности,

$$R_n = (4n+2)R_{n-1} + R_{n-2} = \frac{1}{2}\{(4n+2)(P_{3n-5} + Q_{3n-5})\};$$

тогда, используя равенство (18), получаем:

$$R_n = (4n+2)R_{n-1} + R_{n-2} = \frac{1}{2}\{(4n+2)(P_{3n-2} + Q_{3n-2}) +$$
$$+ P_{3n-5} + Q_{3n-5}\} = \frac{1}{2}(P_{3n+1} + P_{3n+1}).$$

Согласно принципу полной математической индукции равенство (19) верно для всех n. Совершенно аналогично доказывается, что

$$S_n = \frac{1}{2}(P_{3n+1} - Q_{3n-1}).$$

Рассмотрим теперь предел отношений величин R_n и S_n, находим:

$$\frac{\lim\limits_{n\to\infty}\dfrac{P_{3n+1}}{Q_{3n+1}}+1}{\lim\limits_{n\to\infty}\dfrac{P_{3n+1}}{Q_{3n+1}}-1}=\lim_{n\to\infty}\frac{P_{3n+1}+Q_{3n+1}}{P_{3n+1}-Q_{3n+1}}=\lim_{n\to\infty}\frac{R_n}{S_n}=\frac{e+1}{e-1}$$

т.е.

$$\lim_{n\to\infty}\frac{P_{3n+1}}{Q_{3n+1}}=e.$$

Поскольку цепная дробь в правой части (17) сходится, мы будем иметь также, что вообще $\lim\limits_{n\to\infty}\dfrac{P_n}{Q_n}=e$, а это доказывает теорему.

Литература

[1] Бухштаб, А. А. Теория чисел / А. А. Бухштаб. — Москва : ПРОСВЕЩЕНИЕ, 1966. —387 с.

Пирамидальная сортировка

Уильямс Дж. *

Аннотация. На основе исходных данных строится двоичная куча, в которой последовательно собираются минимальные значения

Ключевые слова: Сортировка

Пирамидальная сортировка — алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае (то есть гарантированно) за $O(n\log n)$ операций при сортировке n элементов. Количество применяемой служебной памяти не зависит от размера массива (то есть, $O(1)$).

Может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает (min-heap) / тонет (max-heap) по многим путям.

*example@mail.com

Алгоритм

Сортировка пирамидой использует бинарное сортирующее дерево. Сортирующее дерево — это такое дерево, у которого выполнены условия:

1 Каждый лист имеет глубину либо d, либо $d-1$, d — максимальная глубина дерева.

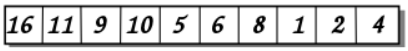


Рисунок 1: Структура хранения данных сортирующего дерева

2 Значение в любой вершине не меньше (другой вариант — не больше) значения её потомков.

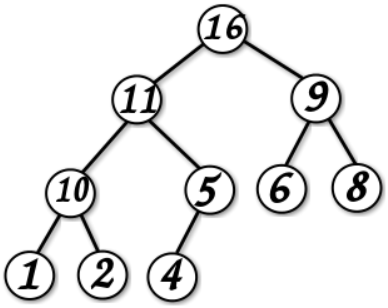


Рисунок 2: Пример сортирующего дерева

Алгоритм сортировки будет состоять из двух основных шагов:

1 Выстраиваем элементы массива в виде сортирующего дерева.

$Array[i] \geqslant Array[2i+1]$

$Array[i]$ и $Array[2i+2]$.

При $0 \leqslant i \leqslant n/2$

Этот шаг требует $O(n)$ операций.

2 Будем удалять элементы из корня по одному за раз и перестраивать дерево. То есть на первом шаге обмениваем $Array[0]$ и

$Array[n-1]$, преобразовываем $Array[0]$, $Array[1], \ldots, Array[n-2]$ в сортирующее дерево. Затем переставляем $Array[0]$ и $Array[n-2]$, преобразовываем $Array[0], \ldots, Array[n-3]$ в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда $Array[0]$, $Array[1], \ldots, Array[n-1]$ — упорядоченная последовательность.

Этот шаг требует $O(n \log n)$ операций.

Достоинства и недостатки

Достоинства

- Имеет доказанную оценку худшего случая $O(n \cdot \log n)$.
- Сортирует на месте, то есть требует всего $O(1)$ дополнительной памяти (если дерево организовывать так, как показано выше).

Недостатки

- Неустойчив — для обеспечения устойчивости нужно расширять ключ.
- На почти отсортированных массивах работает столь же долго, как и на хаотических данных.
- На одном шаге выборку приходится делать хаотично по всей длине массива — поэтому алгоритм плохо сочетается с кэшированием и подкачкой памяти.
- Методу требуется «мгновенный» прямой доступ; не работает на связанных списках и других структурах памяти последовательного доступа.
- Не распараллеливается.

Код

Java

Листинг 1: Пирамидальная сортировка на Java

```
public class HeapSort
{
    public void sort(int arr[])
    {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i=n-1; i>=0; i--)
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
     void heapify(int arr[], int n, int i)
    {
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;

        if (l < n && arr[l] > arr[largest])
            largest = l;
        if (r < n && arr[r] > arr[largest])
            largest = r;
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }

    static void printArray(int arr[])
```

```
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    public static void main(String args[])
    {
        int arr[] = {12, 11, 13, 5, 6, 7};
        int n = arr.length;
        HeapSort ob = new HeapSort();
        ob.sort(arr);
        System.out.println("Sorted array is");
        printArray(arr);
    }
}
```

C++

Листинг 2: Пирамидальная сортировка на C++

```
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
```

```
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i=n-1; i>=0; i--)
    {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    heapSort(arr, n);
    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Литература

[1] Левитин, А. В. Алгоритмы. Введение в разработку и анализ / А. В. Левитин. — Москва : Вильямс, 2006. — 576 с.

[2] Кормен, Е. Алгоритмы: построение и анализ = Introduction to Algorithms / Е. Кормен. — Москва : Вильямс, 2005.

Матрица поворота

Лурье А. И. *

Аннотация. ортогональная матрица, которая используется для выполнения собственного ортогонального преобразования в евклидовом пространстве.

Ключевые слова: Матрица, евклидово пространство, умножение матриц

Обычно считают, что в отличие от матрицы перехода при повороте системы координат (базиса), при умножении на матрицу поворота вектора-столбца координаты вектора преобразуются в соответствии с поворотом самого вектора (а не поворотом координатных осей; то есть при этом координаты повернутого вектора получаются в той же, неподвижной системе координат). Однако отличие той и другой матрицы лишь в знаке угла поворота, и одна может быть получена из другой заменой угла поворота на противоположный; та и другая взаимно обратны и могут быть получены друг из друга транспонированием.

Матрица поворота в двумерном пространстве

В двумерном пространстве поворот можно описать одним углом θ со следующей матрицей линейного преобразования в декартовой системе координат:

$$M(\theta) = \begin{pmatrix} \cos\theta & \mp\sin\theta \\ \pm\sin\theta & \cos\theta \end{pmatrix}$$

или

$$M(\theta) = exp\left(\theta \cdot \begin{bmatrix} 0 & \mp\sin\theta \\ \pm 1 & 0 \end{bmatrix}\right)$$

Поворот выполняется путём умножения матрицы поворота на вектор-столбец, описывающий вращаемую точку:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \mp\sin\theta \\ \pm\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

*example@mail.com

Координаты $(x^{'}, y^{'})$ в результате поворота точки (x, y) имеют вид:

$$x^{'} = x\cos\theta \mp y\sin\theta, y^{'} = \pm x\sin\theta + y\cos\theta.$$

Конкретные знаки в формулах зависят от того, является ли система координат правосторонней или левосторонней, и выполняется ли вращение по или против часовой стрелки. Верхний знак указан для обычного соглашения: правосторонняя система координат и положительное направление вращения против часовой стрелки (тот же знак верен для левосторонней координатной системы при выборе положительного направления вращения по часовой стрелке; в оставшихся двух комбинациях — нижний знак).

Матрица поворота в трёхмерном пространстве

Любое вращение в трёхмерном пространстве может быть представлено как композиция поворотов вокруг трёх ортогональных осей (например, вокруг осей декартовых координат). Этой композиции соответствует матрица, равная произведению соответствующих трёх матриц поворота. Матрицами вращения вокруг оси декартовой системы координат на угол α в трёхмерном пространстве с неподвижной системой координат являются:

- Вращение вокруг оси x:

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

- Вращение вокруг оси y:

$$M_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

- Вращение вокруг оси z:

$$M_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Положительным углам при этом соответствует вращение вектора против часовой стрелки в правой системе координат, и по часовой стрелке в левой системе координат, если смотреть против направления соответствующей оси. Например, при повороте на угол $\alpha = 90^\circ$ вокруг оси z ось x переходит в y: $M_z(90^\circ) \cdot e_x = e_y$. Аналогично, $M_y(90^\circ) \cdot e_z = e_x$ и $M_x(90^\circ) \cdot e_y = e_z$. Правая система координат связана с выбором правого базиса.

Перестановочность поворотов

Если M_1 — матрица поворота вокруг оси с ортом n на угол α , M_2 — матрица поворота вокруг оси с ортом m на угол β , то $M_2 \cdot M_1$ — матрица, описывающая поворот, являющийся результатом двух последовательно осуществленных поворотов M_1 и M_2), поскольку

$$(M_2 \cdot M_1) \cdot r = M_2 \cdot (M_1 \cdot r).$$

При этом последовательность поворотов можно поменять, видоизменив поворот M_1:

$$M_2 \cdot M_1 = M_1' \cdot M_2,$$

где матрица $M_1^{'}$ — матрица поворота на угол α вокруг оси с ортом $n^{'}$, повернутым с помощью поворота M_2:

$$n' = M_2 \cdot n, \qquad M_1' = M_2 \cdot M_1 \cdot M_2^T,$$

поскольку $M_2^T \cdot M_2 = E$ ак как матрица поворота является ортогональной матрицей (E — единичная матрица). Заметим, что коммутативности поворотов в обычном смысле нет, то есть

$$M_2 \cdot M_1 \neq M_1 \cdot M_2.$$

Матрица поворота вокруг произвольной оси

Пусть ось вращения задана единичным вектором $\hat{v} = (x, y, z)$, а угол поворота θ .

Тогда матрица поворота в декартовых координатах имеет вид:

$$M(\hat{v},\theta) =$$

$$\begin{pmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta xy) - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)yx + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz + (\sin\theta)x \\ (1-\cos\theta)zx + (\sin\theta)y & (1-\cos\theta)zy + (\sin\theta)x & (1-\cos\theta) + (\sin\theta)z^2 \end{pmatrix}$$

Выражение матрицы поворота через кватернион

Если задан кватернион $q = (w, x, y, z)$, то соответствующая матрица поворота имеет вид:

$$Q = \begin{bmatrix} 1-2y^2-2x^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{bmatrix}$$

Литература

[1] Гантмахер, Ф. Р. Теория матриц / Ф. Р. Гантмахер. — Изд. 5-е. — Москва : Физматлит, 2004. — 560 с.

[2] Курош, А. Г. Курс высшей алгебры / А. Г. Курош. — Изд. 9-е. — Москва : Наука, 1968. — 432 с.

[3] Ленг, С. Алгебра / С. Ленг. — Москва : Мир, 1968. — 564 с.

[4] Хорн, Р. Матричный анализ / Р. Хорн. — Москва : Мир, 1989. — 655 с.

Муравьиный алгоритм

*Грасс П. *, Дориго М.***

Аннотация.

Полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Ключевые слова: Оптимизация,алгоритм

Обзор

В основе алгоритма лежит поведение муравьиной колонии — маркировка более удачных путей большим количеством феромона. Работа начинается с размещения муравьёв в вершинах графа (городах), затем начинается движение муравьёв — направление определяется вероятностным методом, на основании формулы вида:

$$P_i = \frac{l_i^q \cdot f_i^p}{\sum\limits_{k=0}^{N} l_k^q \cdot f_k^p}$$

где: P_i — вероятность перехода по пути i,

l_i — величина, обратная весу (длине) i-го перехода,

f_i — количество феромона на i-м переходе,

q — величина, определяющая «жадность» алгоритма,

p — величина, определяющая «стадность» алгоритма.

Решение не является точным и даже может быть одним из худших, однако, в силу удачно подобранных эвристик, итерационное применение алгоритма обычно даёт результат, близкий к оптимальному.

Краткое изложение

*example@mail.com

**example@mail.com

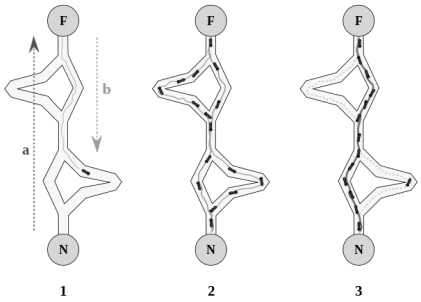


Рисунок 1: Работа алгоритма

В реальном мире муравьи (первоначально) ходят в случайном порядке и по нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым, и, как следствие, плотность феромонов остаётся высокой. Испарение феромонов также имеет функцию избежания стремления к локально-оптимальному решению. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным.

Таблица 1

Сравнение генетического и муравьиного алгоритма

	Тесты + LTL формулы	
Алгоритм	Генетический	Муравьиный
Среднее	52405	25976
Сандартное отклонение	61440	23389
Медиана	31507	18680
Минимум	2875	2392
Максимум	498365	154802

В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.

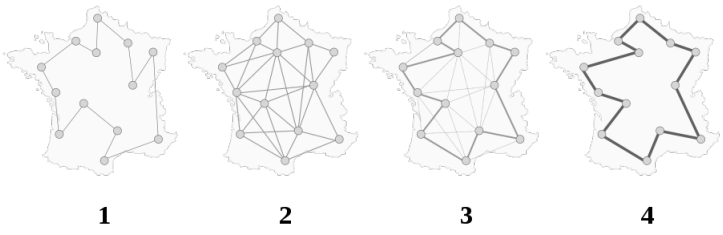


Рисунок 2: Пример нахождения минимального расстояния

Вариации алгоритма

Элитарная муравьиная система

Из общего числа муравьёв выделяются так называемые «элитные муравьи». По результатам каждой итерации алгоритма производится усиление лучших маршрутов путём прохода по данным маршрутам элитных муравьёв и, таким образом, увеличение количества феромонов на данных маршрутах. В такой системе количество элитных муравьёв является дополнительным параметром, требующим определения. Так, для слишком большого числа элитных муравьёв алгоритм может «застрять» на локальных экстремумах.

MMAS (Max-Min муравьиная система)

Добавляются граничные условия на количество феромонов (T_{min}, T_{max}). Феромоны откладываются только на глобально лучших или лучших в итерации путях. Все рёбра инициализируются значением T_{max}.

Ранговая муравьиная система (ASrank)

Все решения ранжируются по степени их пригодности. Количество откладываемых феромонов для каждого решения взвешено так, что более подходящие решения получают больше феромонов, чем менее подходящие.

Длительная ортогональная колония муравьёв (COAC)

Механизм отложения феромонов COAC позволяет муравьям искать решения совместно и эффективно. Используя ортогональный метод, муравьи в выполнимой области могут исследовать их выбранные области быстро и эффективно, с расширенной способностью глобального поиска и точностью. проблем.

Листинг 3: Муравьиный алгоритм на Python

```
import numpy as np
import random as rd

def lengthCal(antPath,distmat):
    length =[]
    dis = 0
    for i in range(len(antPath)):
        for j in range(len(antPath[i]) - 1):
            dis += distmat[antPath[i][j]]
            [antPath[i][j + 1]]
        dis += distmat[antPath[i][-1]]
        [antPath[i][0]]
        length.append(dis)
        dis = 0
    return length

distmat = np.array
([[0,35,29,67,60,50,66,44,72,41,48,97],
[35,0,34,36,28,37,55,49,78,76,70,110],
[29,34,0,58,41,63,79,68,103,69,78,130],
[67,36,58,0,26,38,61,80,87,110,100,110],
[60,28,41,26,0,61,78,73,103,100,96,130],
[50,37,63,38,61,0,16,64,50,95,81,95],
[66,55,79,61,78,16,0,49,34,82,68,83],
[44,49,68,80,73,64,49,0,35,43,30,62],
[72,78,103,87,103,50,34,35,0,47,32,48],
[41,76,69,110,100,95,82,43,47,0,26,74],
[48,70,78,100,96,81,68,30,32,26,0,58],
[97,110,130,110,130,95,83,62,48,74,58,0]])
```

```
antNum = 12
alpha = 1
beta = 3
pheEvaRate = 0.3
cityNum = distmat.shape[0]
pheromone = np.ones((cityNum,cityNum))
heuristic = 1 / (np.eye(cityNum) + distmat)
- np.eye(cityNum)
iter,itermax = 1,100

while iter < itermax:
    antPath = np.zeros((antNum, cityNum)).
    astype(int) - 1
    firstCity = [i for i in range(12)]
    rd.shuffle(firstCity)
    unvisted = []
    p = []
    pAccum = 0
    for i in range(len(antPath)):
        antPath[i][0] = firstCity[i]
    for i in range(len(antPath[0]) - 1):
        for j in range(len(antPath)):
            for k in range(cityNum):
                if k not in antPath[j]:
                    unvisted.append(k)
            for m in unvisted:
                pAccum += pheromone
                [antPath[j][i]][m] **
                alpha * heuristic
                [antPath[j][i]][m] **
                beta
            for n in unvisted:
                p.append(pheromone
                [antPath[j][i]][n] **
                alpha * heuristic
                [antPath[j][i]][n] **
                beta / pAccum)
            roulette = np.array(p).cumsum()
            r = rd.uniform(min(roulette),
            max(roulette))
```

```
            for x in range(len(roulette)):
                if roulette[x] >= r:
                    antPath[j][i + 1] = unvisted[x]
                    break
            unvisted = []
            p = []
            pAccum = 0
    pheromone = (1 - pheEvaRate) * pheromone
    length = lengthCal(antPath,distmat)
    for i in range(len(antPath)):
        for j in range(len(antPath[i]) - 1):
            pheromone[antPath[i][j]]
            [antPath[i][j + 1]] += 1 / length[i]
        pheromone[antPath[i][-1]]
        [antPath[i][0]] += 1 / length[i]
    iter += 1
print("Shortest distance :")
print(min(length))
print("Most shortest distance :")
print(antPath[length.index(min(length))])
```

Литература

[1] Dorigo, M. Ant colony optimization / M. Dorigo. — Cambridge, Mass : MIT Press, 2004.

[2] Dorigo, M. Optimization, Learning and Natural Algorithms / M. Dorigo. — Rome : Politecnico di Milano, 1992.

[3] Кирсанов, М. Н. Графы в Maple / М. Н. Кирсанов. — Москва : Физматлит, 2007. — 168 с.

[4] Кажаров, А. А. Муравьиные алгоритмы для решения транспортных задач / А. А. Кажаров // Теория и системы управления. — 2010. — Т. 1. — С. 32-45.

Быстрый обратный квадратный корень

*Уолш Г. *, Моулер К.***

Аннотация. быстрый приближённый алгоритм вычисления обратного квадратного корня$y = \frac{1}{\sqrt{x}}$ для положительных 32-битных

Ключевые слова: квадратный корень, трёхмерная графика

Алгоритм использует целочисленные операции «вычесть» и «битовый сдвиг», а также дробные «вычесть» и «умножить» — без медленных операций «разделить» и «квадратный корень». Несмотря на «хакерство» на битовом уровне, приближение монотонно и непрерывно: близкие аргументы дают близкий результат. Точности (менее $0,2\%$ в меньшую сторону и никогда — в большую) не хватает для настоящих численных расчётов, однако вполне достаточно для трёхмерной графики.

Рисунок 1: При расчёте освещения OpenArena вычисляет углы падения и отражения через быстрый обратный квадратный корень.

Алгоритм

Алгоритм принимает 32-битное число с плавающей запятой (одинарной точности в формате IEEE 754) в качестве исходных данных и производит над ним следующие операции:

*example@mail.com
**example@mail.com

1. Трактуя 32-битное дробное число как целое, провести операцию $y0 = 5F3759DF_{16}$ - $(x >> 1)$, где $>>$ — битовый сдвиг вправо. Результат снова трактуется как 32-битное дробное число.

2. Для уточнения можно провести одну итерацию метода Ньютона: $y1 = y0(1,5 - 0,5xy_{0^2})$.

Листинг 4: Реализация на языке C

```
 float Q_rsqrt( float number )
{
        const float x2 = number * 0.5F;
        const float threehalfs = 1.5F;
        union {
                float f;
                uint32_t i;
        } conv = {number};.
        conv.i = 0x5f3759df - ( conv.i >> 1 );
        conv.f *= threehalfs - x2 * conv.f * conv.f;
        return conv.f;}
```

Реализация из Quake III: Arena считает, что *float* по длине равен *long*, и использует для преобразования указатели (может ошибочно сработать оптимизация «если изменился *float*, ни один *long* не менялся»; на GCC при компиляции в «выпуск» срабатывает предупреждение).

Анализ и погрешность

Имеем дело только с положительными числами (знаковый бит равен нулю), не денормализованными, не ∞ и не NaN. Такие числа в стандартном виде записываются как $1,mmmm_2 2^e$. Часть $1,mmmm$ называется мантиссой, e — порядком. Головную единицу не хранят (неявная единица), так что величину $0,mmmm$ назовём явной частью мантиссы. Кроме того, у машинных дробных чисел смещённый порядок: 20 записывается как $011.1111.1_2$.

На положительных числах биекция «дробное $\leftrightarrow$ целое» (ниже обозначенная как I_x) непрерывна как кусочно-линейная функция и

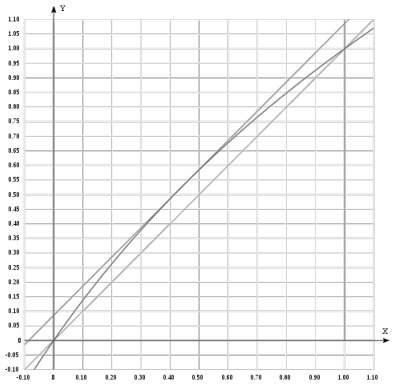


Рисунок 2: $\log_2(1+m_x) \approx m_x + \sigma$. Приведены крайние случаи—$\sigma = 0$ и $0,086$

монотонна. Отсюда сразу же можно заявить, что быстрый обратный корень, как комбинация непрерывных функций, непрерывен. А первая его часть — сдвиг-вычитание — к тому же монотонна и кусочно-линейна. Биекция сложна, но почти «бесплатна»: в зависимости от архитектуры процессора и соглашений вызова, нужно или ничего не делать, или переместить число из дробного регистра в целочисленный. Например, двоичное представление 16-ричного целого числа $0x5F3759DF$ есть $0|101.1111.0|011.0111.0101.1001.1101.1111_2$ (Точки — границы полубайтов, вертикальные линии — границы полей компьютерного дробного). Порядок 10111110_2 равен 190_{10}, после вычитания смещения 127_{10} получаем показатель степени 63_{10}. Явная часть мантиссы $01101110101100111011111_2$ после добавления неявной ведущей единицы превращается в $1,01101110101100111011111_2 = 1,432430148\ldots_{10}$. С учётом реальной точности компьютерных дробных $0x5F3759DF \leftrightarrow 1,4324301102^{63}$.

Обозначим $m_x \in [0,1)$ явную часть мантиссы числа $x, e_x \in \mathbb{Z}$ — несмещённый порядок, $L = 2^{23}$ — разрядность мантиссы, $B = 127$ — смещение порядка. Число $x \equiv 2^{e_x}(1+m_x)$, записанное в линейно-логарифмической разрядной сетке компьютерных дробных, можно приблизить логарифмической сеткой как $x \equiv e_x + \log_2(1+m_2) \approx e_x + m_x + \sigma$где σ — параметр, используемый для настройки точности приближения. Этот параметр варьируется от 0 (формула точна при $m_x = 0$ и 1) до 0,086 (точна в одной точке, $m_x = 0{,}443$).

Воспользовавшись этим приближением, целочисленное представление числа x можно приблизить как

$$I_x \equiv L(e_x + B + m_x) \approx L \log_2 x + L(B - \sigma)$$

Соответственно $\log_2 x \approx \frac{I_x}{L} - (B - \sigma)$.

Проделаем это же для $y = \frac{1}{\sqrt{x}}$ (соответственно $\log_2 y = -\frac{1}{2}\log_2 x$), и получим

$$I_y \approx \frac{3}{2}L(B - \sigma) - \frac{1}{2}I_x$$

$$y \approx I^{-1}\left[\frac{3}{2}L(B - \sigma) - \tfrac{1}{2}I_x\right]$$

Магическая константа $\frac{3}{2}L(B - \sigma)$, с учётом границ σ, в арифметике дробных чисел имеет вид $c \cdot 2^{63}$, где $c = 1,5 - 1,5\sigma \in (1,37; 1,5)$, а в двоичной записи — $0|101.1111.0|01_1 \ldots$ (Маленькая единица крайне вероятна, но не гарантирована нашими прикидочными расчётами.)

Можно вычислить, чему равняется первое кусочно-линейное приближение (в источнике используется не сама мантисса, а её явная часть $t = c - 1$):

- Для $x \in [0,5; c - 0,5) : y_{01} = -x + t + \frac{3}{2} = -x + c + \frac{1}{2}$;
- Для $x \in [c - 0,5; 1) : y + 02 = -\frac{1}{2}x + \frac{1}{2}t + \frac{5}{4} = -\frac{1}{2}x + \frac{1}{2}c + \frac{3}{4}$;
- Для $x \in [1 : 2) : y_{03} = -\frac{1}{4}x + \frac{1}{2}t + 1 = -\frac{1}{4} + \frac{1}{2}c + \frac{1}{2}$

На больших или меньших x результат пропорционально меняется: при учетверении x результат уменьшается ровно вдвое. Метод Ньютона даёт $f(y) = \frac{1}{y^2} - x, f^{'}(y) = -\frac{2}{y^3}$, и $y_{n+1} = y_n - \frac{f(y_n)}{f^{'}(y_n)} = \frac{y_n(3 - xy_n^2)}{2} = y_n(1,5 - 0,5)$. Функция $f(y)$ убывает и выпукла вниз, на таких функциях метод Ньютона подбирается к истинному значению слева — потому алгоритм всегда занижает ответ. Неизвестно, откуда взялась константа $0x5F3759DF \leftrightarrow 1,43243012^{63}$.

Перебором Крис Ломонт и Мэттью Робертсон выяснили, что наилучшая по предельной относительной погрешности константа для $float$ — $0x5F375A86 \leftrightarrow 1,43245002^{63}$, для $double$ —

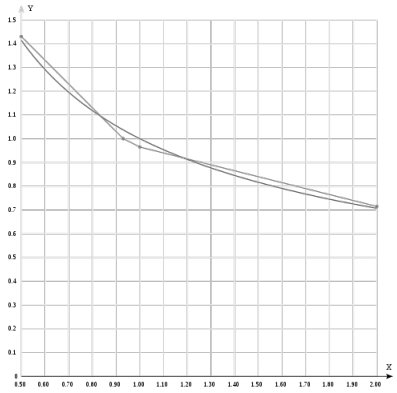


Рисунок 3: Первое (кусочно-линейное) приближение быстрого обратного квадратного корня ($c = 1,43$)

$0x5FE6EB50C7B537A9$. Правда, для double алгоритм бессмысленный (не даёт выигрыша в точности по сравнению с $float$). Константу Ломонта удалось получить и аналитически ($c = 1,43245008479014264217 9$), но расчёты довольно сложны. Эта цифра округляется до $1,4324500$, потому что единица младшего разряда равняется $1,1910^{-7}$, и следующее число округляется до $1,4324502$.

После одного шага метода Ньютона результат получается довольно точный ($+0\% - 0,18\%$), что для целей компьютерной графики более чем подходит ($1/256 \approx 0,39\%$). Такая погрешность сохраняется на всём диапазоне нормированных дробных чисел. Два шага дают точность в 5 цифр, после четырёх достигается погрешность $double$.

Метод Ньютона не гарантирует монотонности, но компьютерный перебор показывает, что монотонность всё-таки есть.

Литература

[1] Соммервил И. Инженерия. программного обеспечения. [Текст] : 6–е изд. / пер. с англ.; М.: Вильямс, 2002. – 624с.

[2] Przybylek Adam. Post object-oriented paradigms in software development: a comparative analysis [Текст] // Proceedings of the 1st Workshop on Advanced in Programming Languages at International Multiconference on Computer Science and Information Technology, October 15 – 17, 2007. Wisła, Poland. – pp. 1009-1020.

[3] Bonferroni C.E. Il calcolo delle assi curazioni su gruppi di test. in Studi Onore del Professore Salvatore Ortu Carboni. Rome, Italy, 1935. P.13-60.

[4] Пекельник Н. М. Замечание об одном интегральном представлении // Н. М. Пекельник, О. И. Хаустова, Н. И. Попова, И. А. Трефилова. – Актуальные проблемы гуманитарных и естественных наук. – 2016. – № 2-1. – С. 33–37.

[5] Esseen C.-G. A moment inequality with an application to the central limit theorem. Scand.Aktuarietidskr. J. – 1956. – V. 3-4. – P. 160-170.

SHA-3

Дамен Й., Раймен В.***

Аннотация. Алгоритм хеширования переменной разрядности преемки SHA-2

Ключевые слова: Криптография, SHA

Алгоритм

Хеш-функции семейства $SHA-3$ построены на основе конструкции криптографической губки, в которой данные сначала «впитываются» в губку, при котором исходное сообщение M подвергается многораундовым перестановкам f, затем результат Z «отжимается» из губки. На этапе «впитывания» блоки сообщения суммируются по модулю 2 с подмножеством состояния, после чего всё состояние

преобразуется с помощью функции перестановки f. На этапе «отжимания» выходные блоки считываются из одного и того же подмножества состояния, изменённого функцией перестановок f. Размер

*daemen.j@protonworld.com

**vincent.rilnmen@estat.kuleuven.com

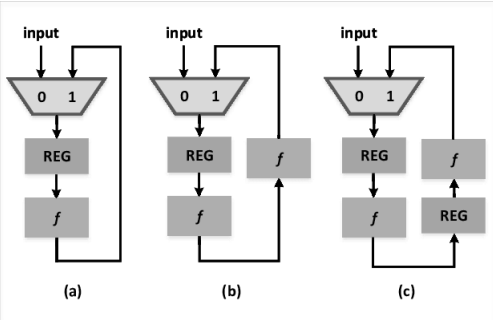


Рисунок 1: Архитектура SHA-3

части состояния, который записывается и считывается, называется «скоростью» и обозначается r, а размер части, которая нетронута вводом / выводом, называется «ёмкостью» и обозначается c.

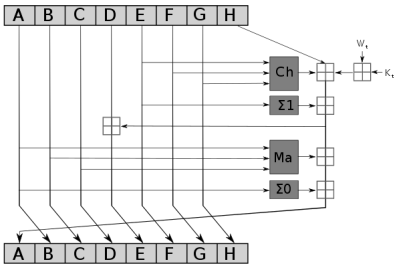
Алгоритм получения значения хеш-функции можно разделить на несколько этапов:

- Исходное сообщение M дополняется до строки P длины, кратной r, с помощью функции дополнения (pad-функции);
- Строка P делится на n блоков длины r: $P_0, P_1, \ldots, P_{n-1}$;
- «Впитывание»: каждый блок P_i дополняется нулями до строки длины b бит и суммируется по модулю 2 со строкой состояния S, где S — строка длины b бит ($b = r + c$). Перед началом работы функции все элементы S равны нулю. Для каждого следующего блока состояние — строка, полученная применением функции перестановок f к результату предыдущего шага;
- «Отжимание»: пока длина Z меньше d (d — количество бит в результате хеш-функции), к Z добавляется r первых бит состояния S, после каждого прибавления к S применяется функция перестановок f. Затем Z обрезается до длины d бит;
- Строка Z длины d бит возвращается в качестве результата.

Благодаря тому, что состояние содержит c дополнительных бит, алгоритм устойчив к атаке удлинением сообщения, к которой восприимчивы алгоритмы $SHA-1$ и $SHA-2$.

В $SHA-3$ состояние S — это массив 55 слов длиной $w = 64$ бита, всего $5564 = 1600$ бит. Также в Keccak могут использоваться длины w, равные меньшим степеням 2 (от $w = 1$ до $w = 32$).

Дополнение



[h]

Рисунок 2: Схема итерации алгоритма SHA-2

Для того, чтобы исходное сообщение M можно было разделить на блоки длины r, необходимо дополнение. В $SHA-3$ используется паттерн $pad10*1$: к сообщению добавляется 1, после него — 0 или больше нулевых битов (до $r-1$), в конце — 1.

$r-1$ нулевых битов может быть добавлено, когда последний блок сообщения имеет длину $r-1$ бит. Этот блок дополняется единицей, следующий блок будет состоять из $r-1$ нулей и единицы.

Два единичных бита добавляются и в том случае, если длина исходного сообщения M делится на r. В этом случае к сообщению добавляется блок, начинающийся и оканчивающийся единицами, между которыми $r-2$ нулевых битов. Это необходимо для того, чтобы для сообщения, оканчивающегося последовательностью битов как в функции дополнения, и для сообщения без этих битов значения хеш-функции были различны.

Первый единичный бит необходим для того, чтобы результаты хеш-функции от сообщений, различающихся несколькими нулевыми битами в конце, были различны.

Функция перестановок

Функция перестановок, используемая в $SHA-3$, включает в себя исключающее «ИЛИ» (XOR), побитовое «И» (AND) и побитовое отрицание (NOT). Функция определена для строк длины-степени 2 $w = 2^l$. В основной реализации $SHA-3$ $w = 64$ ($l = 6$).

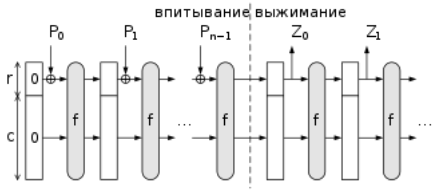


Рисунок 3: Конструкция функции губки, использованная в хеш-функции.

Состояние S можно представить в виде трёхмерного массива A размером $55w$. Тогда элемент массива $A[i][j][k]$ - это $(5i + j)xw + k$ бит строки состояния S.

Функция содержит несколько шагов: $\theta, \rho, \pi, \chi, \iota$, которые выполняются несколько раундов. На каждом шаге обозначим входной массив A, выходной массив $A^{'}$.

Шаг θ

Для всех i и k, таких, что $0 \leqslant i < 5$, $0 \leqslant k < w$, положим
$C(i,k) = A[i,0,k] \oplus A[i,1,k] \oplus A[i,2,k] \oplus A[i,3,k] \oplus A[i,4,k]$
$D(i,k) = C[(i-1) \mod 5, k] \oplus C[(i+1) \mod 5, (k-1) \mathrm{mod} w]$
Для всех (i,j,k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$,
$A^{'}[i,j,k] = A[i,j,k] \oplus D[i,k]$

Шаг ρ

Для всех k, таких, что $0 \leqslant k < w$, $A^{'}[0,0,k] = A[0,0,k]$
Пусть в начале $(i,j) = (1,0)$. Для t от 0 до 23:

1. Для всех k, таких, что $0 \leqslant k < w$, $A^{'}[i,j,k] = A[i,j,(k-(t+1)(t+2)/2) \mathrm{mod} w]$
2. $(i,j) = (j, (2i+3j) \mod 5)$

Шаг π

Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$
$A'[i, j, k] = A[(i + 3j) \mod 5, i, k]$

Шаг χ

Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$,
$A[i, j, k] = A[i, j, k] \oplus ((A[(i + 1) \mod 5, j, k] \oplus 1) \cdot A[(i + 2) \mod 5, j, k])$

Шаг ι

Введем дополнительную функцию $rc(t)$, где вход — целое число t, а на выходе — бит.

Алгоритм $rc(t)$

1. Если $t \mod 255 = 0$, то возвращается 1
2. Пусть $R = [10000000]$
3. Для i от 1 до $t \mod 255$:
   (a) $R = 0||R$
   (b) $R[0] = R[0] \oplus R[8]$
   (c) $R[4] = R[4] \oplus R[8]$
   (d) $R[5] = R[5] \oplus R[8]$
   (e) $R[6] = R[6] \oplus R[8]$
   (f) $R = Trunc_8[R]$
4. Возвращается $R[0]$.

Алгоритм $\iota(A, i_r)$

i_r — номер раунда.

1. Для всех (i, j, k), таких, что $0 \leqslant i < 5$, $0 \leqslant j < 5$, $0 \leqslant k < w$ $A'[i, j, k] = A[i, j, k]$
2. Пусть RC — массив длины w, заполненный нулями.
3. Для i от 0 до l: $RC[2^i - 1] = rc(i + 7i_r)$
4. Для всех k, таких, что $0 \leqslant k < w$, $A^{'}[0, 0, k] = A^{'}[0, 0, k] \oplus RC[k]$

Алгоритм перестановок

1. Перевод строки S в массив A
2. Для i_r от $12 + 2l - n_r$ до $12 + 2l - 1$ $A^{'} = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$
3. Перевод массива $A^{'}$ в строку $S^{'}$ длины b

Настройки

Оригинальный алгоритм Keccak имеет множество настраиваемых параметров с целью обеспечения оптимального соотношения криптостойкости и быстродействия для определённого применения алгоритма на определённой платформе. Настраиваемыми величинами являются: размер блока данных, размер состояния алгоритма, количество раундов в функции $f()$ и другие.

Таблица 1

Параметры типов SHA-3

Тип	Длинна вывода	Частота	Объём
$SHA-3$-224	224	1152	448
$SHA-3$-256	256	1088	512
$SHA-3$-384	384	832	768
$SHA-3$-512	512	576	1024

На протяжения конкурса хеширования Национального института стандартов и технологий участники имели право настраивать свои алгоритмы для решения возникших проблем. Так, были внесены некоторые изменения в Keccak: количество раундов было увеличено с 18 до 24 с целью увеличения запаса безопасности.

Авторы Keccak учредили ряд призов за достижения в криптоанализе данного алгоритма.

Версия алгоритма, принятая в качестве окончательного стандарта $SHA-3$, имеет несколько незначительных отличий от оригинального предложения Keccak на конкурс. В частности, были ограничены некоторые параметры (отброшены медленные режимы $c = 768$ и $c = 1024$), в том числе для увеличения производительности. Также в стандарте были введены «функции с удлиняемым результатом» (XOF, Extendable Output Functions) $SHAKE128$ и $SHAKE256$, для

Таблица 2

Соответсвие параметров между SHA-3 и Keccak

Функции	Формулы
SHA-224(M)	KECCAK[448](M||01,224)
SHA3-256(M)	KECCAK[512](M||01,256)
SHA3-384(M)	KECCAK[768](M||01,384)
SHA3-512(M)	KECCAK[1024](M||01,512)
SHAKE128(M,d)	KECCAK[256](M||1111,d)
SHAKE256(M,d)	KECCAK[512](M||1111,d)

чего хешируемое сообщение стало необходимо дополнять «суффиксом» из 2 или 4 бит, в зависимости от типа функции.

Криптоанализ

Таблица 3

Результаты криптоанализа

Цель	**Тип атаки**	**Выход**	**Вариант**	**CF Call**
Хеш-функция	Коллизия	160	1, 2 раунда	
Хеш-функция	Нахождение прообраза	80	1, 2 раунда	
Перестановки	Атака-различитель	Все	24 раунда	2^{1579}
Перестановки	Дифференциальное свойство	Все	8 раундов	$2^{491.47}$
Хеш-функция	Атака-различитель	224, 256	4 раунда	2^{25}
Хеш-функция	Коллизия	224, 256	2 раунда	2^{33}

На следующей странице

Таблица 3 – *Продолжение таблицы 3*

Цель	**Тип атаки**	**Выход**	**Вариант**	**CF Call**
Хеш-функция	Нахождение второго прообраза	224, 256	2 раунда	2^{33}
Хеш-функция	Нахождение второго прообраза	512	6 раундов	2^{506}
Хеш-функция	Нахождение второго прообраза	512	7 раундов	2^{507}
Хеш-функция	Нахождение второго прообраза	512	8 раундов	$2^{511.5}$
Хеш-функция	Коллизия	224, 256	4 раунда	

Литература

[1] Соммервил И. Инженерия. программного обеспечения. [Текст] : 6–е изд. / пер. с англ.; М.: Вильямс, 2002. – 624с.

[2] Przybylek Adam. Post object-oriented paradigms in software development: a comparative analysis [Текст] // Proceedings of the 1st Workshop on Advanced in Programming Languages at International Multiconference on Computer Science and Information Technology, October 15 – 17, 2007. Wisła, Poland. – pp. 1009-1020.

[3] Bonferroni C.E. Il calcolo delle assi curazioni su gruppi di test. in Studi Onore del Professore Salvatore Ortu Carboni. Rome, Italy, 1935. P.13-60.

Авторский указатель

Научное издание
Сборник статей
2022
Дата выхода в свет: 04.05.2022. Формат 210 x 148
Усл. печ. л. 16. Тираж 1 экз. Цена 0 руб. Заказ 1