Лекция 13. Методы создания самотестирующихся и самокорректирующихся программ

Цель лекции: изучить методы создания и области применения самотестирующихся и самокорректирующихся программ.

План лекции:
Введение.
1 Общие положения
2 Пример самотестирующейся/самокорректирующейся программной пары для функции дискретного экспоненцирования
3 Области применения самотестирующихся и самокорректирующихся программ и их сочетаний
Заключение
Контрольные вопросы

Ключевые слова: [выбрать самостоятельно].

Содержание лекции:

Введение

1 Общие положения

Принцип самотестирования программы. Пусть требуется написать программу P для вычисления функции f так, чтобы P(x) = f(x) для всех значений x. Традиционные методы верификационного анализа и тестирования программ не позволяют убедиться с вероятностью близкой к единице в корректности результата выполнения программы, хотя бы потому, что тестовый набор входных данных, как правило, не перекрывает весь их возможный спектр. Один из методов решения данной проблемы заключается в создании так называемых самокорректирующихся и самотестирующихся программ, которые позволяют оценить вероятность некорректности результата выполнения программы, т.е. вероятность того, что $P(x) \neq f(x)$, и корректно вычислить f(x) для любых x в том случае, если сама программа P на большинстве наборах своих входных данных (не на всех наборах) работает корректно.

Чтобы добиться корректного результата выполнения программы P, вычисляющей функцию f, нам необходимо написать такую программу T_j, которая позволяла бы оценить вероятность того, что $P(x) \neq f(x)$ для любых x. Такая вероятность будет называться вероятностью ошибки выполнения программы P. При этом T_j может обращаться к P как к своей подпрограмме.

Обязательным условием для программы T_j является ее принципиальное временное отличие от любой корректной программы вычисления функции f в том смысле, что время выполнения программы T_j, не учитывающее время вызовов программы P, должно быть значительно меньше, чем время выполнения любой корректной программы для вычисления f. В этом случае, вычисления согласно T_j некоторым количественным образом должны отличаться от вычислений функции f и самотестирующаяся программа может рассматриваться как независимый шаг при верификации программы P, которая предположительно вычисляет функцию f. Кроме того, желательное свойство для T_j должно заключаться в том, чтобы ее код был, насколько это возможно, более простым. Таким образом, программа T_j должна быть эффективной в том смысле, что время выполнения T_j даже с учетом времени, затраченного на вызовы P, должно составлять константный мультипликативный фактор от времени выполнения P. Итак, самотестирование должно лишь незначительно замедлять время выполнения программы P.

Пусть π означает некоторую вычислительную задачу и/или некоторую задачу поиска решения. Для x, рассматриваемого в качестве входа задачи, пусть π(x) обозначает результат решения задачи к. Пусть P – программа (предположительно предназначенная) для решения задачи π, которая останавливается (например, не имеет зацикливаний) на всех входах задачи π. Будем говорить, что P имеет дефект, если для некоторого входа x задачи π имеет место $P(x) \neq \pi(x)$.

Эффективный программный чекер C_π для задачи π определим следующим образом. Чекер $C_\pi^P(I,k)$ является произвольной вероятностной машиной Тьюринга, удовлетворяющей следующим условиям. Для любой программы P (предположительно решающей задачу π), выполняемой на всех входах задачи π, для любого элемента I задачи π и для любого положительного k (параметра безопасности) имеет место следующее:

- ✓ *если программа P не имеет дефектов, т.е. $P(x) = \pi(x)$ для всех входов x задачи π, тогда $C_\pi^P(I,k)$ выдаст на выходе ответ «Норма» с вероятностью не менее $1 - 1/2^k$;*
- ✓ *если программа P имеет дефекты, т.е. $P(x) \neq \pi(x)$ для всех входов x задачи π, тогда $C_\pi^P(I,k)$ выдаст на выходе ответ «Сбой» с вероятностью не менее $1 - 1/2^k$.*

Самокорректирующаяся программа – это вероятностная программа C_j, которая помогает программе P скорректировать саму себя, если только P выдает корректный результат с низкой вероятностью ошибки, т.е. для любого x программа C_j вызывает программу P для корректного вычисления f(x), в то время как собственно сама P обладает низкой вероятностью ошибки.

Самотестирующейся/самокорректирующейся программной парой называется пара программ вида (T_j, C_j). Предположим, что пользователь может взять любую программу P, которая целенаправленно вычисляет f и тестирует саму себя при помощи программы T_j. Если P проходит такие тесты, тогда по любому x пользователь может вызвать программу C_j, которая, в свою очередь, вызывает P для корректного вычисления f(x). Даже если программа P вычисляет значение функции/некорректно для некоторой небольшой доли входных значений, ее в данном случае все равно можно уверенно использовать для корректного вычисления f(x) для любого x. Кроме того, если удастся в будущем написать программу P' для вычисления f, тогда некоторая пара (T_j, C_j) может использоваться для самотестирования и самокоррекции P' без какой-либо ее модификации. Таким образом, имеет смысл тратить определенное количество времени для разработки самотестирующейся/самокорректирующейся программной пары для вычислительных функций, используемых в прикладных программах.

Вероятностная оракульная программа. Перед тем как перейти к более формальному описанию определений самотестирующихся и самокорректирующихся программ, необходимо дать определение вероятностной оракульной программы (по аналогии с вероятностной оракульной машиной Тьюринга).

Вероятностная программа M является вероятностной оракульной программой, если она может вызывать другую программу, которая является исполнимой во время выполнения M. Обозначение M^A означает, что M может делать вызовы программы A.
Пусть P – программа, которая предположительно вычисляет функцию f, где функция имеет один аргумент (вход), выбираемый из некоторого универсального множества I. Пусть I является объединением подмножеств I_n где $n \in N$, и пусть $D^p = \{D_n \mid n \in N\}$ есть ансамбль распределений вероятностей D_n над I_n. Далее, пусть err(P, f, D_n) – ошибка программы, т.е. вероятность того, что $P(x) \neq f(x)$, где x выбрано случайно в соответствии с распределением D_n из подмножества I_n. Пусть $\beta > 0$ есть некоторый параметр безопасности. Тогда (ε_1, ε_2) – самотестирующейся программой для функции f в отношении D^p с параметрами

$$0 \leq \varepsilon_1 < \varepsilon_2 \leq 1$$

называется вероятностная оракульная программа T_f которая для параметра безопасности β и любой программы P на входе n имеет следующие свойства:

- ✓ *если $err(P, f, D_n) \le \varepsilon_1$; тогда программа $T^P{}_f$ выдаст на выходе ответ «Норма» с вероятностью не менее $1 - \beta$;*
- ✓ *если $err(P, f, D_n) \ge \varepsilon_2$, тогда программа $T^P{}_f$ выдаст на выходе ответ «Сбой» с вероятностью не менее $1 - \beta$.*

Оракульная программа C_f с параметром $0 \le \varepsilon < 1$ называется ε-самокорректирующейся программой для функции f в отношении множества распределений D^p, которая имеет следующее свойство по входу n, $x \in I_n$, и параметру безопасности β: если $err(P, f, D_n) \le \varepsilon$, тогда $C_f^P = f(x)$ с вероятностью не менее $1 - \beta$.

Программной парой $(\varepsilon_1, \varepsilon_2, \varepsilon)$-самотестирующейся/самокорректирующейся для функции f называется пара вероятностных программ (T_f, C_f), такая, что существуют константы $0 \le \varepsilon_1 < \varepsilon_2 \le 1$ и множество распределений D^p, при которых T_f есть $(\varepsilon_1, \varepsilon_2)$-самотестирующаяся программа для функции f в отношении D^p и C_f есть ε-самокорректирующаяся программа для функции f в отношении распределения D^p.

Свойство случайной самосводимости. Пусть $x \in I_n$, и пусть $c > 1$ есть целое число. Свойство случайной самосводимости заключается в том, что существует два алгоритма:

- ✓ *алгоритм A_1; который работает за время, пропорциональное $n^{O(1)}$, и посредством которого функция $f(x)$ может быть выражена через вычислимую функцию F от x, $a_1, ..., a_C$ и $f(a_1), ..., f(a_C)$;*
- ✓ *алгоритм A_2, который работает за время, пропорциональное $n^{O(1)}$, и посредством которого по данному x можно вычислить $a_1; ..., a_C$, где каждое a_i является случайно распределенным над I_n в соответствии с D^p.*

2 Пример самотестирующейся/самокорректирующейся программной пары для функции дискретного экспоненцирования

Обозначения и определения для функции дискретного возведения в степень вида A^x modulo N

Пусть $I_n = Z_q$ представляет собой множество {1, ..., q}, где $q = \varphi(M)$ – значение функции Эйлера для модуля M, а Z^*_M – множество вычетов по модулю M, где $n = [\log_2 M]$. Пусть распределение D^p есть равномерное распределение вероятностей. Оракульной программой в данном случае является программа вычисления функции g^x modulo M по отношению к исследуемым самотестирующейся и самокорректирующейся программам.

Алгоритм A^x modulo N можно вычислить многими способами ([1]4] и др.). Среди них один из наиболее известных и широко применяемых на практике – это алгоритм, основанный на считывании индекса (значения степени) слева направо. Этот метод достаточно прост и нагляден, история его восходит к 200 г. до н.э. Метод известен также как русский крестьянский метод.

Пусть x[1, ..., n] – двоичное представление положительного числах и A, B и N – положительные целые числа в r-ичной системе счисления, тогда псевдокод алгоритма A^x modulo N, реализованного программой Exp(·), имеет следующий вид.

Листинг 13.1. Псевдокод алгоритма A^x modulo N

```
Program Exp (x, N, A, R);  {вход – x, N, A, выход – R}
begin
        B := 1;
        For I := 1 to n do
                begin
                        B := (B*B) mod N;
                        if [i] = 1 then B := (A*B) mod N;
                end;
        R := B;
```

[1] Кнут Д. Искусство программирования для ЗВМ: в 3 т. / Д. Кнут. – Мир, 1976.

```
end.
```

Из описания алгоритма следует, что число обращений к операции вида A*B modulo N будет log x + β(x), где β(x) – число единиц в двоичном представлении операнда x или вес Хэмминга x.

Построение самотестирующейся/самокорректирующейся программной пары для функции дискретного экспоненцирования

Прежде всего рассмотрим следующие четыре алгоритма (см. листинги 13.2 – 13.5).

Листинг 13.2. Псевдокод алгоритма S_K_exp

```
Program S_K_exp(x, M, q, q, Rk); {вход – n, x, M, q, q, выход – Rk}
    begin
        for l = 1 to 12ln(1/β)
            begin
                x1 := random(q); {random – функция случайного равнове-
                роятного выбора из целочисленного отрезка [1, …, q – l]}
                x2 := (x – x1) mod q;
                Exp(g, x1, M, R1); {Exp – процедура вычисления gx modulo
                M=R}
                Exp(g, x2, M, R2);
                R0 := (R1·R2) mod M;
            end;
            Rk = choice(R0(l)); {choice - функция выбора из массива, состоя-
            щего из 12ln(1/β) элементов, ответов, который повторяется наи-
            большее количество раз}
    end.
```

Листинг 13.3. Псевдокод алгоритма S_T_exp

```
Program S_T_exp(x, M, q, g, β); {вход – x, M, q, g, выход – значение предиката
output}
    begin
        t1 := 0; t2 := 0;
        for l = 1 to [576ln(4/β)]
            begin
                L_T(g, M, q, R1); {L_T – процедура, реализующая тест
                линейной состоятельности, выход – R1}
                t1 := t1 + R1;
            end;
        if t1/[576ln(4/β)] > 1/72 then output := “false”;
        for l = 1 to [32ln(4/β)]
            begin
                N_T(g, M, q, Re); {N_T – процедура, реализующая тест
                единичной состоятельности, выход – Re}
                t2 := t2 + Re;
            end;
        if t1/[321ln(4/β)] > 1/4 then output := “false”;
            else output := “true”
    end.
```

Листинг 13.4. Псевдокод алгоритма теста линейной состоятельности L_T

```
Program L_T(g, M, q, R1);  {вход – g, M, q, выход – R1}
    begin
```

```
                x1 := random(q);
                x1 := random(q);
                Exp(g, x1, M, R1);
                Exp(g, x2, M, R2);
                Exp(g, x, M, R);
                If R1·R2 = R then Rl := 1
                    else R1 := 0;
            end.
```

Листинг 13.5. Псевдокод алгоритма теста единичной состоятельности N_T

```
        Program N_T(g, M, q, Re);  {вход – g, M, q, выход – Re}
            begin
                x1 := random(q);
                x1 := (x1 + 1) mod q;
                Exp(g, x1, M, R1);
                Exp(g, x2, M, R2);
                If R1·g = R2 then Re := 1
                    else Re := 0;
            end.
```

Для доказательства полноты и безопасности указанной самотестирующейся/ самокорректирующейся программной пары доказывается следующая теорема.

Теорема 13.1. Пара программ S_K_exp(x, M, q, g, β) и S_T_exp(x, M, q, g, β) является (1/288, 1/8, 1/8)-самокорректирующейся/самотестирующейся программной парой для функции g^x modulo M с входными значениями, выбранными случайно и равновероятно из множества I_n.

Для доказательства теоремы необходимо доказать две леммы.

Лемма 13.1. Программа S_K_exp(M, q, g, β) является (1/8)-самокорректирующейся программой для вычисления функции g^x modulo M в отношении равномерного распределения D_n.

Доказательство. Сначала покажем, что если оракульная программа P(x), обозначаемая как Exp(·), выполняется корректно на большинстве входов, то и самокорректирующаяся программа S_K(·) будет выполняться корректно. В данном случае это очевидно. Если P(x) корректно вычислима, то из $[P_{M,g}(x_1)] \cdot P_{M,g}(x_2)]$ (mod M) следует, что

$$f_{M,g}(x) = f_{M,g}(x_1)^{\circ} f_{M,g}(x_2) = g^{[x_1+x_2](mod\ \varphi(M))}(mod\ M) \equiv g^x(mod\ M) \equiv R_k.$$

Для доказательства того, что на большинстве входов программа S_K(·) выдает корректный результат, необходимо отметить, что так как $x_1 \in {}_R Z_q$, то и x_2 имеет равномерное распределение вероятностей над Z_q. Так как вероятность ошибки $\varepsilon \leq 1/8$, то в одном цикле вероятность $Prob[R_k = f_{M,g}(x)] \geq 3/4$. Чтобы вероятность корректного ответа была не менее чем 1 – β, исходя из оценки Чернова, необходимо выполнить не менее 12ln(1/β) циклов.

Лемма 13.2. Программа S_T_exp(n, M, q, g, (3) является (1/288, 1/8)-самотестирующейся программой, которая контролирует результат вычисления значения функции gx modulo M с любым модулем M.

Доказательство. Корректность программы S_T(·) доказывается аналогично доказательству корректности в лемме 13.1, где $x_1, x_2 \in {}_R Z_q$,. Корректное выполнение теста единичной состоятельности $[P_{M,g}(x_1)] \cdot P_{M,g}(1)]$ (mod M) соответствует вычислению функции

$$f_{M,g}(x) = f_{M,g}(x_1)^\circ f_{M,g}(1) = g^{[x_1+1](mod\ \varphi(M))}(mod\ M) \equiv$$
$$\equiv g^{x_1} \cdot g(mod\ M) \equiv g^x(mod\ M) \equiv R_e.$$

Для доказательства условия самотестируемости необходимо отметить, что, как и в лемме 13.1, для того чтобы вероятность корректных ответов R_1 и R_e в обоих тестах была не менее чем 1 – β, достаточно выполнить тест линейной состоятельности [576ln(4/β)] раз и тест единичной состоятельности [32ln(4/β)] раз.

Можно показать, основываясь на общих положениях теории групп, что возможно обобщение программы S_T(·) и для других групп (вышеописанные алгоритмы основываются на вычислениях в мультипликативной группе вычетов над конечным полем), т.е. для всех y ∈ G, P(y) ∈ G*, где G* представляет собой любую группу, кроме групп G**. К группам последнего вида относятся бесконечные группы, не имеющие конечных подгрупп за исключением {O'}, где O' – тождество группы. Таким образом, можно показать (если параметры выбираются независимо, равновероятно и случайным образом), что программа вида S_T(·) является (ε/36, ε)-самотестирующейся программой.

Исходя из определения самотестирующейся/самокорректирующейся программной пары и основываясь на результатах доказательств лемм 13.1 и 13.2, очевидным образом следует доказательство теоремы 13.1.

Замечания. Как следует из псевдокода алгоритма A^x modulo N, в нем, как уже говорилось, используется операция AB modulo N. Используя ту же технику доказательств, что и для функции дискретного возведения в степень, можно построить (1/576, 1/36, 1/36)-самокорректирующуюся/самотестирующуюся программную пару для вычисления функции модулярного умножения. Это справедливо исходя из следующих соображений. Вычисление функции

$$f_M(ab) = f_M\big((a_1 + a_2)(b_1 + b_2)\big)$$

следует из корректного выполнения программы с 4-кратным вызовом оракульной программы P(a, b), т.е.

$$[P_M(a_1, b_1) + P_M(a_1, b_2) + P_M(a_2, b_1) + P_M(a_2, b_2)](mod\ M),$$

Алгоритм вычисления A^x modulo N выполняется для c = 2. Однако декомпозиция x, как следует из свойства самосводимости функции A^x modulo N, может осуществляться на большее число слагаемых. Хотя это приведет к гораздо большему количеству вызовов оракульной программы, но в то же время позволит значительно снизить вероятность ошибки.

Таким образом, мы рассмотрели возможность создания самотестирующихся программ с эффективным тестирующим модулем. Такой модуль осуществляет автономное тестирование программ на предмет отсутствия/наличия преднамеренных и/или непреднамеренных программных дефектов и использует ST-пару функций (g_c, f_c), таких, что $Y = g_c(f(a_1), ..., f(a_c))$ и $X = h_c(f_1, ..., a_c)$ для некоторого входного вектора X, которые используют свойство рандомизированной самосводимости функции Y = f(X), вычисляемой посредством программы P. Для этого реализуется алгоритм, блок-схема которого приведена на рис. 13.1.

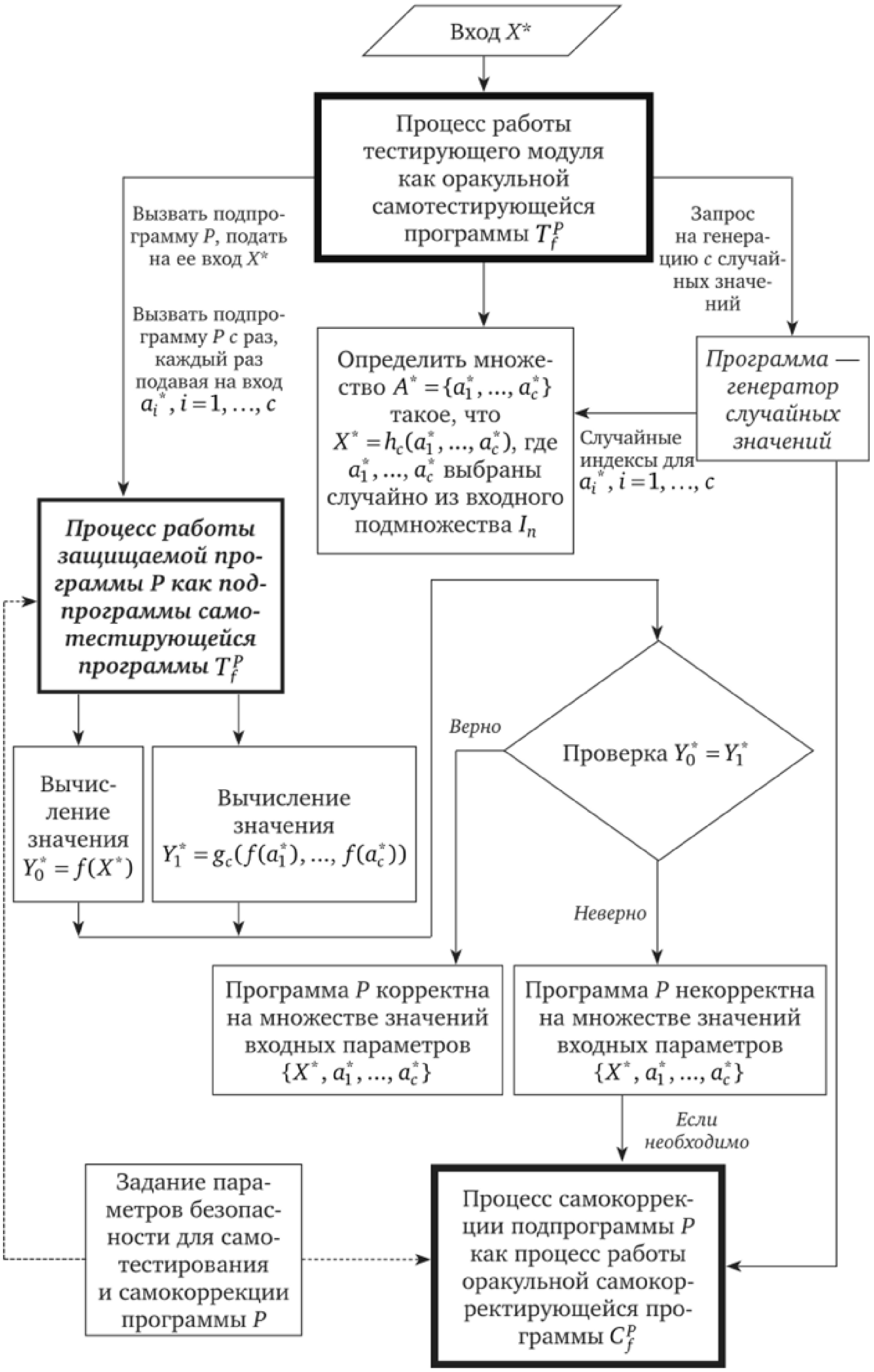

Рис. 13.1 Схема работы самотестирующейся/самокорректирующейся программной пары

3 Области применения самотестирующихся и самокорректирующихся программ и их сочетаний

Вычислительная математика

Общие замечания. Применение чекеров, самотестирующихся, самокорректирующихся программ и их сочетаний возможно в самых разных областях вычислительной математики, а, следовательно, в самых разнообразных областях человеческой деятельности, где широко применяются вычислительные методы. К ним относятся такие направления, как цифровая обработка сигналов (решение задач в системах распознавания изображений и голоса в радио- и гидроакустике) , а также методы математического моделирования процессов изменения народонаселения, экономических процессов, процессов изменения погоды и т.п. Идеи самотестирования могут найти самое широкое применение в системах защиты информации (например, в системах открытого распределения ключей, в криптосистемах с открытым ключом, в схемах идентификации пользователей вычислительных систем и аутентификации данных), где базовые вычислительные алгоритмы обладают некоторыми алгоритмическими свойствами, например свойством случайной самосводимости, описанным выше.

Активными исследованиями в области создания самотестирующихся и самокорректирующихся программ ученые и практики начали заниматься с начала 1990-х гг. В этот период были разработаны программные чекеры для ряда теоретико-числовых и теоретико-групповых задач, для решения задач с матрицами, полиномами, линейными уравнениями и рекуррентными соотношениями.

Далее приведем наиболее интересные и необходимые схемы, протоколы, теоремы и их доказательства.

Целочисленная арифметика и арифметика многократной точности. Пусть M(n) – время выполнения программы P на входе размером n. В 1990-х гг. были разработаны программные чекеры для функций, представленных в табл. 13.1.

Таблица 13.1 Программные чекеры для некоторых функций

Функция	Время выполнения	
	без учета времени вызова программы	общее
A mult B	N	M(n)
A div B	n log n	M(n)) log n
A modulo N	N	M(n)
AB modulo N	N	M(n)
A^b modulo N (с известной факторизацией модуля)	N	M(n)
A^b modulo N (с неизвестной факторизацией модуля)	n $\log^4$n	M(n)$\log^4$n

В табл. 13.1 приведены время (затраты) на выполнение самотестирующейся/ самокорректирующейся программной пары для указанных целочисленных (в том числе модулярных с операндами многократной точности) функций. Во второй колонке показано время выполнения самотестирующейся/ самокорректирующейся программной пары без учета времени вызова программы, реализующей функции, приведенные в первой колонке. В третьей колонке приведено общее время выполнения с учетом времени вызовов программы P. В общее время не включается время выполнения программ реализации функций, зависящее от параметра безопасности k, который обычно составляет O(log(1/k)).

Теоретико-групповые и теоретико-числовые вычисления. Существуют чекеры для решения некоторых теоретико-групповых и теоретико-числовых проблем, некоторые из которых приведены ниже.

Проблема эквивалентного поиска. Пусть S – множество и G – группа, групповые действия в которой, осуществляются над элементами множества S. Для a, b ∈ S элемент $a \equiv_G b$ тогда и только тогда, когда g(a) = b для некоторого g из G. Проблема эквивалентного поиска заключается в нахождении g такого, что g(a) = b, если $a \equiv_G b$ для a и b, принадлежащих множеству S. Если существует эффективный вероятностный алгоритм поиска g ∈ G, тогда существует эффективный программный чекер для данной проблемы. Примеров решения задач эффективного эквивалентного поиска достаточно много. К ним относятся проблема поиска изоморфизма графов (см. далее), решение задачи квадратных вычетов, обобщенная проблема дискретных логарифмов, задачи подобные «Кубику Рубика», ряд задач из теории кодирования и др.

Проблема пересечения групп. Пусть G и H – группы перестановок, определенные некоторыми генераторами групп. Генераторы представляются как перестановки над [1, …, n]. Проблема пересечения групп заключается в нахождении образующих для $G \cap H$. Существует возможность построения программного чекера для решения данной проблемы.

Проблема расширенного нахождения НОД. Проблема расширенного нахождения наибольшего общего делителя (которая отличается от нахождения НОД посредством алгоритма Евклида) заключается в нахождении для двух положительных целых a и b целого d = НОД(a, b) и целых u и v, таких, что au + bv = d.

Чекер для решения расширенного нахождения НОД по входу двух положительных целых a и b, целого d и целых u и v выдаст ответ «Сбой», если d не делит a, или d не делит b, или $au + bv \neq d$. В противном случае чекер выдаст ответ «Норма». Эффективность и корректность данного чекера легко доказывается.

Вычисления над полиномами. Существует достаточно простой способ построения самокорректирующейся программы, который основывается на существовании следующего интерполяционного тождества, соответствующего значениям функций между точками: для всех одномерных полиномов f степени не более d, для всех $x, t \in F$

$$\sum_{i=1}^{d+1} \alpha_i f(x + a_i t) = 0,$$

где α зависит от F, d и не зависит от x, t; $\alpha_0 = -1$; α_i – различные элементы из F.

Тогда самокорректирующаяся программа для вычисления $f(\vec{x}) = f(x_1, \ldots, x_n)$ заключается в выполнении следующего алгоритма. Случайно и равновероятно выбирается

$$\vec{t} = (t_1, \ldots, t_n) \in F^n,$$

и выдается

$$\sum_{i=0}^{d+1} \alpha_i P(\vec{x} + i \cdot \vec{t}) = 0.$$

С вероятностью не менее 2/3 все вызовы программы будут возвращать корректные результаты, и, следовательно, выход программы будет корректным. Листинг 13.6 демонстрирует самотестирующуюся программу для полинома f.

Листинг 13.6. Псевдокод алгоритма самотестирующейся программы для полинома/

```
Program P_S_T(P, ε, β, x, f(x)); {вход – P, ε, β, (x1, f(x1), …, xd + 1, f(xd + 1)), вы-
ход – («Норма», «Сбой»)}
    begin
        for i = 1 to O((1/k)log(1/β)) do
            begin
                x, t := random(Zp); {random –  функция случайного равно-
                вероятного выбора из множества вычетов по модулю p};
                if ∑_{i=0}^{d+1} α_i P(x + it) ≠ 0 (более чем ε итерациях)
                then output «Сбой»;
            end;
```

```
            output «Норма»;
            for j = 0 to d do ∑_{i=0}^{d+1} α_i P(x + it) ≠ 0
                begin
                    for i = 1 to O(log(d/β)) do
                        begin
                            t := random(Z_p);
                            {random – функция случайного равновероят-
                            ного выбора из множества вычетов по моду-
                            лю p};
                            if ∑_{i=0}^{d+1} α_i P(x_j + it) ≠ f(x_j) (более чем в ¼
                            итерациях) then output «Сбой»;
                        end;
                end;
            output «Норма»;
        end.
```

Вычисления над матрицами. Одной из первых в области вероятностных алгоритмов, в конечном счете ставшей одной из основополагающих в области методологии самотестирования, была работа Р. Фрейвалдса, написанная им еще в 1979 г. Он предложил простой и элегантный чекер для задачи умножения матриц. Рассмотрим его.

Пусть матрицы A и B – матрицы размером n×n определены над конечным полем F.

Время выполнения программы, реализующей чекер Фрейвалдса (обозначаемый в программе как C_F), составляет $O(n^2[\log(l/k)])$.

Используя чекер Фрейвалдса, можно построить самотестирующуюся/ самокорректирующуюся программную пару для умножения матриц, что и демонстрируют листинги 13.7 – 13.9.

Листинг 13.7. Псевдокод алгоритма, реализующего чекер Фрейвалдса

```
    Program C_F(A, B, C, k); {вход – A, B, C, k, выход – («Норма», «Сбой»)}
        begin
            for i = 1 to [log(1/k)] do
                begin
                    R := random(F); {random – функция случайного равноверо-
                    ятного выбора 0-1-вектора размером (n×1) из F};
                    If C·R ≠ A·(B·R) then output «Сбой»;
                end;
            output «Норма»;
        end.
```

Листинг 13.8. Псевдокод алгоритма самокорректирующейся программы умножения матриц

```
    Program S_K_mult AB(A, B, k); { вход – A, B, C, k, выход – C}
        begin
            for i = 1 to ∞ do
                begin
                        A1 := random(F); {random – функция случайного рав-
                        новероятного выбора матрицы размером (n×n) из F};
                        B1 := random(F); {random – функция случайного рав-
                        новероятного выбора матрицы размером (n×n) из F};
                A2 := A – A1;
                B2 := B – B1;
                C := P(A1, B1) + P(A1, B2) + P(A2, B1) + P(A2, B2);
```

```
                    if C_F(A, B, C, k) = «Норма» then output C and go to 1;
                    end;
            1:end.
```

Время выполнения программы S_K_mult AB составляет $O(M(n) + n^2\log(l/k))$, где M(n) – время выполнения программы умножения матриц размером n×n.

Самотестирующаяся программа для умножения матриц строится достаточно просто.

Листинг 13.9. Псевдокод алгоритма самотестирующейся программы умножения матриц

```
        Program S_K_mult AB(A, B, k); { вход – A, B, C, k, выход – («Норма», «Сбой»)}
            begin
                for i = 1 to O(log(1/k))∞ do
                    begin
                            A := random(F); {random – функция случайного рав-
                            новероятного и независимого выбора матрицы раз-
                            мером (n×n) из F};
                            B := random(F); {random – функция случайного рав-
                            новероятного и независимого выбора матрицы раз-
                            мером (n×n) из F};
                    C := P(A, B);
                    if C_F(A, B, C, 1/4) = «Норма» then output 0 and go to 1;
                    if C_F(A, B, C, 1/4) = «Сбой» then output 1 and go to 1;
                    end;
            1:end.
```

Можно легко удостовериться, что, если $err(P, f, U_n) \geq 1/8$, то количество единиц на выходе будет не менее 1/16 с вероятностью не менее 1 – k, и если $err(P, f, U_n) \leq 1/32$, то количество единиц будет не менее 1/16 с вероятностью не более 1 – k. Таким образом, выше приведенная программа будет (1/32, 1/8)-самотестирующейся программой для умножения матриц.

Таблица 13.2 Программные чекеры для некоторых операций над матрицами

Функция	Время выполнения	
	без учета времени вызова программы	общее
Умножение матриц	n	M(n)
Определение детерминанта	n	M(n)
Инверсия матрицы	n	M(n)
Определение ранга матрицы C	n	M(n)
Определение ранга матрицы T	$n\sqrt{n}$	$M(n)\sqrt{n}$

Аналогичным образом строятся самотестирующиеся/самокорректирующиеся программные пары для других операций над матрицами. Данные по временным затратам сведены в табл. 13.2, обозначения в которой те же, что в табл. 13.1.

Линейные рекуррентные соотношения. Исследовались вопросы построения самотестирующихся и самокорректирующихся программ для линейных рекуррентных соотношений, т.е. соотношений вида

$$f(n) = \sum_{i=1}^{d} c_i f(n-i).$$

Такие последовательности являются основными для многих комбинаторных и теоретико-числовых последовательностей, таких как последовательность Фибоначчи и последовательность Лукаша.

Линейные рекуррентные соотношения часто рассматриваются в неявном виде в качестве однородных линейных уравнений вида

$$\sum_{i=0}^{d} c_i f(n + d - i) = 0$$

и часто используются в таких прикладных областях, как моделирование динамики изменения народонаселения, различных экономических процессов, при анализе различных трафиков (потоков всевозможных данных, процессов) и т.п., а также при описании различных процессов в робототехнике и цифровой обработке сигналов.

Криптография, интерактивные доказательства

Вводные замечания. Основная идея использования задач самотестирования в криптографии заключается в девизе «Защитить самих защитников!». Так как криптографические методы используются для высокоуровневого обеспечения конфиденциальности и целостности информации, собственно программно-техническая реализация этих методов должна быть свободна от программных и/или аппаратных дефектов. Таким образом, самотестирование и самокоррекция программ может эффективно применяться в современных системах защиты информации от несанкционированного доступа.

Распределение ключей, цифровая подпись, схемы аутентификации.

Функция дискретного экспоненцирования (см. выше) широко используется в современной криптографии, в частности, при открытом распределении ключей Диффи – Хеллмана, для генерации и верификации подписей в схемах электронной цифровой подписи, для построения различных схем аутентификации сообщений, идентификации пользователей вычислительных систем и т.п. Следовательно, существует принципиальная возможность построения программных чекеров, самотестирующихся, самокорректирующихся программ для криптографических схем, использующих эту функцию. Продемонстрируем это на примере схемы цифровой подписи RSA (листинг 13.10).

Пусть программа P предположительно вычисляет RSA-функцию и для $x, y, z \in Z > 0$ с $x, y < z$ и НОД$(x, z) = 1$. Тогда чекер $C^{P}_{RSA}(x, y, z, k)$ работает следующим образом.

Листинг 13.10. Псевдокод алгоритма RSA-чекера

```
Program C_RSA_(x, y, z, k); {вход – x, y, z, k, выход – («Норма», «Сбой»)}
    begin
        t1 := [-klog99/1002];
        t2 := [-klog4/52];
        for l = 1 to t1 do
            begin
                    i := random(Z); {random – функция случайного рав-
                    новероятного выбора из [1, …, z]};
                if P(x, i, z) ≡ 0 (mod z) output «Сбой» and STOP;
                    i, j := random(Z); {random – функция случайного рав-
                    новероятного выбора из [1, …, z]};
                if P(x, i, z)P(x, j, z) ≠ P(x, i + j, z) (mod z) ) output «Сбой»
                and STOP;
                    i := random(Z); {random – функция случайного рав-
                    новероятного выбора из [1, …, z]};
```

```
                        if P(x, i, z) ≡ P(x, l, z) ≠ P(x, i + l, z) (mod z) ) output «Сбой»
                        and STOP;
                    end;
                for l = 1 to t2 do
                    begin
                            r := random(Z); {random – функция случайного рав-
                            новероятного выбора из [1, …, z]};
                        if P(x, y, z)P(x, r, z) ≠ P(x, y + r, z) (mod z) ) output «Сбой»
                        and STOP;
                    end;
                output «Норма»;
            end.
```

Заключение

Контрольные вопросы

Смотри руководство по организации самостоятельной работы магистрантов.