Лабораторная работа №1
Рекурсивные и итеративные алгоритмы

Краткая теоретическая справка

Числа Фибоначчи

Числовая последовательность, исследование которой приписывают Леонардо Пизанскому (известного как Фибоначчи), часто встречается при изучении различных структур данных и алгоритмов. Для задачи вычисления членов данной последовательности имеется несколько алгоритмов решения, которые обладают различной сложностью. Последний факт придаёт данной задаче большой интерес с точки зрения анализа алгоритмов.

Подробное описание различных свойств данной последовательности читатель может найти в монографии Кнута [1].

Определение 1.1. *(Последовательность Фибоначчи) Последовательность целых чисел, заданная соотношениями*

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad \forall n > 2,$$

называется последовательностью Фибоначчи.

Используя определение, можно записать рекурсивный алгоритм вычисления n-го члена последовательности.

```
FIB(n)
1  ▷ возвращает F_n
2  if n ≤ 1
3      then return(n)
4  else return(FIB(n − 1) + FIB(n − 2))
```

Утверждение 1.1. *Рекурсивный алгоритм вычисления значения n-го числа Фибоначчи имеет сложность $O(\phi^n)$, где $\phi = \frac{1+\sqrt{5}}{2}$ — число Фидия (золотое сечение).*

Доказательство. Пусть $T(n)$ — количество операций, необходимых для вычисления n-го числа Фибоначчи, тогда в соответствии с алгоритмом для $T(n)$ можно записать следующие соотношения

$$T(n) = \begin{cases} c_1, & n \leq 1 \\ T(n-1) + T(n-2) + c_2, & n > 1. \end{cases} \tag{1}$$

Здесь c_1 и c_2 количество операций для проверки условия и сложения двух чисел.

Будем искать решение в виде $T(n) = x^n$, при $n \gg 1$. Это позволяет переписать (1) в виде

$$x^n = x^{n-1} + x^{n-2} + c_2. \tag{2}$$

Разделим (2) на x^n и, переходя к пределу при $n \to \infty$ с учётом того, что $\lim\limits_{n\to\infty} x^n = \infty$, находим

$$x = x^{-1} + x^{-2}.$$

Разрешая уравнение относительно x, получаем два решения $x = \frac{1\pm\sqrt{5}}{2}$. Поскольку все члены последовательности неотрицательны, то следует рассмотреть только положительный корень. Решение можно записать в виде $T(n) = x^n = \phi^n$, при $n \gg 1$, откуда следует, что $T(n) = O(\phi^n)$. Полученный результат говорит о том, что данный алгоритм не является эффективным. Рассмотрим итеративный алгоритм решения этой задачи.

```
FIB(n)
1  ▷ возвращает F_n
2  if n ≤ 1
3     then return(n)
4  f ← 0,  f_1 ← 1,  f_2 ← 0
5  while n > 1
6        do f ← f_1 + f_2
7           f_2 ← f_1,  f_1 ← f
8           n ← n − 1
9  return(f)
```

Анализ итеративного алгоритма достаточно легко провести путём прямого подсчёта количества операций.

Строки 2, 4, а также 3 или 10 выполняются 1 раз при любых $n > 1$. Обозначим количество действий, необходимых для их выполнения, как c_1. Тело цикла (7-8) и условие (5) выполняются $n-1$ раз. Обозначим количество шагов для каждой итерации цикла как c_2. Собирая всё вместе, получаем следующее представление для общего количества шагов

$$T(n) = c_2(n-1) + c_1 = O(n).$$

Сравнение двух алгоритмов вычисления чисел Фибоначчи показывает, что итеративный алгоритм для этой задачи является более эффективным.

Отдельно следует отметить, что существует алгоритм вычисления чисел Фибоначчи, работающий за $O(\log n)$ шагов [2, 9].

Поиск максимума

Рекурсивный алгоритм

```
MAXIMUM(A[1..n])
1  ▷ Возвращает max A[1..n]
2  if n ≤ 1
3     then return(A[1])
4  return(max(MAXIMUM(A[1..n − 1]), A[n]))
```

Итеративный алгоритм

```
MAXIMUM(A[1..n])
1  ▷ Возвращает max A[1..n]
2  m ← A[1], t ← 1
3  while i ≤ n
4        do if A[i] > m
5              then m ← i
6  return(m)
```

Приведённые алгоритмы имеют одинаковую сложность, но при этом существуют различие в процессах, порождаемых их программной реализацией. Так использование отложенных вычислений в рекурсивном алгоритме приводит к тому, что процесс использует $O(n)$ памяти для запоминания операций (память расходуется на поддержание стека вызовов подпрограмм). Такое поведение хорошо просматривается на следующем примере:

```
 1  MAXIMUM([1, 5, 7, 2, 3])
 2  max(MAXIMUM([1, 5, 7, 2]), 3)
 3  max(max(MAXIMUM([1, 5, 7]), 2), 3)
 4  max(max(max(MAXIMUM([1, 5]), 7), 2), 3)
 5  max(max(max(max(MAXIMUM([1]), 5), 7), 2), 3)
 6  max(max(max(max(1, 5), 7), 2), 3)
 7  max(max(max(5, 7), 2), 3)
 8  max(max(7, 2), 3)
 9  max(7, 3)
10  7
```

В противоположность этому, итеративный алгоритм использует объём памяти $O(1)$. Что делает его более выгодным для применения.

Стоит отметить, что использование рекурсивного вызова функции не всегда порождает рекурсивный процесс. Так, например, использование хвостовой рекурсии в некоторых языках программирования (*Scheme, Haskell*) порождает итеративный, а не рекурсивный процесс.

```
MAXIMUM-ITER(A[1..n], m)
1  ▷
2  if n = 0
3      then return(m)
4  else return(MAXIMUM-ITER(A[1..n − 1], max(A[n], m)))

MAXIMUM(A[1..n])
1  return(MAXIMUM-ITER(A[1..N-1], A[N]))
```

Рассмотрим поведение процесса, порождённого данным алгоритмом:

```
1  MAXIMUM([1, 5, 7, 2])
2  MAXIMUM-ITER([1, 5, 7], 2)
3  MAXIMUM-ITER([1, 5], 7)
4  MAXIMUM-ITER([1], 7)
5  MAXIMUM-ITER([], 7)
6  7
```

Более подробно об реализации итеративных алгоритмов с помощью рекурсивных функций в языке *Scheme* можно познакомиться в [2].

Ханойские башни

По легенде в великом храме города Бернас под собором, отмечающим середину мира, находится бронзовый диск, на котором укреплены 3 алмазных стержня высотой в один локоть и толщиной с пчелу. Давным-давно, в самом начале времён, монахи этого монастыря провинились перед богом Брахмой. Разгневанный Брахма на один из стержней возложил 64 диска из чистого золота, причём так, что каждый меньший диск лежит на большем. Монахам был дан наказ — переложить все диски с одного стрежня на другой. При этом *нельзя* допускать, чтобы больший по диаметру диск оказался выше меньшего.

Как только все 64 диска будут переложены со стержня, на который Брахма возложил их при создании, на другой стержень, башня вместе с храмом обратится в пыль и под громовые раскаты погибнет мир [3].

Рассмотрим рекурсивный алгоритм решения задачи о ханойских башнях. Обозначим операцию перекладывания одного диска с башни A на башню B как $B \Leftarrow A$.

```
HANOI(n, F, T, V)
1  if n = 1
2      then T ⇐ F
3      else HANOI(n − 1, F, V, T) ▷ n − 1 диск с F на V
4           T ⇐ F
5           HANOI(n − 1, V, T, F) ▷ n − 1 диск с V на T
```

Итеративный алгоритм решения данной задачи не будет более эффективным в плане вычислительной сложности. Преимущество итеративной версии в плане объёма потребляемой памяти в данном случае не будет существенным, поскольку уже при $n = 64$ время решения данной задачи на современных компьютерах может исчисляться годами. При этом итеративный алгоритм не является столь прозрачным.

Умножение натуральных чисел

Несмотря на простую формулировку задачи об умножении двух чисел существует множество различных алгоритмов её решения [4]. В отличие от предыдущих рассмотренных задач размер входных данных задан не явно, а определяется количеством разрядов, необходимых для записи множителей в какой-либо позиционной СС, с основанием, отличным от 1. Здесь и далее будем подразумевать, что для записи чисел используется двоичная СС.

Рекурсивный алгоритм

MULTIPLY(x, y)
1 **if** $y = 0$
2 **then return**(0)
3 **elseif** y нечётно
4 **then return**(MULTIPLY$(2x, \lfloor y/2 \rfloor) + x)$
5 **else return**(MULTIPLY$(2x, \lfloor y/2 \rfloor)$)

Итеративный алгоритм

MULTIPLY(x, y)
1 $p \leftarrow 0$
2 **while** $y > 0$
3 **do if** y нечётно
4 **then** $p \leftarrow p + x$
5 $x \leftarrow 2x$
6 $y \leftarrow \lfloor y/2 \rfloor$
7 **return**(p)

Задание на лабораторную работу

1. Создайте программы, реализующие рекурсивный и итеративный алгоритмы вычисления n-ого числа Фибоначчи. Сравните время их работы для $n = 10, 20, 30, \ldots, 90$. Чтобы избежать переполнения, для $n < 100$ используйте беззнаковое представление чисел размером в 8 байт (тип `unsigned long long` или `uint64_t` в языке C и `uint64` в языке Pascal). Результаты сравнения оформить в виде таблицы и виде графика.

2. Проанализируйте вычислительную и пространственную сложности алгоритмов умножения чисел. На основе анализа проведите сравнение данных алгоритмов.
3. Разработайте два алгоритма возведения числа в целую неотрицательную степень $a^n, \quad n \in \mathbb{Z}^+$ различающиеся по сложности. Обратите внимание, что вычисление a^{15} может потребовать только 6 операций умножения
$$a^{15} = a\left(a^7\right)^2 = a\left(a\left(a^3\right)^2\right)^2 = a\left(a\left(a\left(a^2\right)\right)^2\right)^2.$$
А для вычисления a^{100} может потребоваться всего 14 операций умножения. Для разработанных алгоритмов определите вычислительную сложность и проведите сравнительный анализ.
4. Используя следующее выражение
$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix},$$
разработайте алгоритм вычисления n-ого числа Фибоначчи, работающий за $O(\log n)$ шагов.

Вопросы для самопроверки

1. Что такое алгоритм?
2. Какие существуют свойства алгоритмов?
3. Что такое анализ алгоритма? Приведите пример анализа алгоритма.
4. Может ли одна задача решаться с помощью разных алгоритмов?
5. Может ли один алгоритм решать разные задачи?
6. Как между собой связаны задача, алгоритм, программа и процесс?
7. Приведите пример алгоритма типа «разделяй и властвуй».
8. Приведите пример рекурсивного и итеративного алгоритма.
9. Как провести анализ рекурсивного алгоритма?
10. Приведите примеры классов сложности алгоритмов? Как они соотносятся друг с другом?
11. Дайте определение для нотаций $O(f(n))$, $\Theta(f(n))$, $\Omega(f(n))$, $o(f(n))$.
12. Чем отличаются рекурсивный и итеративный процессы?
13. Покажите, что сложность алгоритма умножения натуральных чисел не зависит от основания СС, в которой они записаны.

Лабораторная работа № 2
Алгоритмы решения задачи о рюкзаке

Краткая теоретическая справка

Неформально задачу о рюкзаке можно описать следующим способом. Пусть задан набор предметов количеством n штук. Каждый предмет имеет свой вес, равный $a_i,\ i = 1 \dots n$ кг. Требуется выбрать такой набор предметов, который можно унести в рюкзаке, выдерживающий вес в s кг. При этом необходимо полностью загрузить рюкзак.

Исследование данной задачи представляет большой интерес, поскольку она принадлежит к классу NP задач [5].

Приведём более формальное определение данной задачи.

Определение 2.1. *(Задача о рюкзаке). Необходимо построить алгоритм, вход и выход которого заданы следующим образом:*

▷ ***INPUT****: подмножество натуральных чисел* $\{a_1, \dots, a_n\}$ *и число* s
▷ ***OUTPUT****:* $\epsilon_i \in \{0, 1\}, 1 \leqslant i \leqslant n$*, такие, что* $\sum_{i=1}^{n} \epsilon_i a_i = s$

Наивный алгоритм

Проведём анализ наивного алгоритма. Всего существует 2^n различных векторов $(\epsilon_1, \dots, \epsilon_n) \in \mathbb{Z}_2^n$. Следовательно в худшем случае потребуется 2^n итераций цикла. Для каждой итерации необходимо вычислить сумму элементов l, что требует n шагов. В конечном счёте получаем, что алгоритм требует $O\left(n2^n\right)$ шагов и поэтому не является эффективным.

SubSet-Sum1(A)
1 ▷ **INPUT**:подмножество натуральных чисел $\{a_1, \dots, a_n\}$ и натуральное число s
2 ▷ **OUTPUT**: $\epsilon_i \in \{0, 1\}, 1 \leqslant i \leqslant n$, такие, что $\sum_{i=1}^{n} \epsilon_i a_i = s$
3 Для каждого вектора $(\epsilon_1, \epsilon_2, ..., \epsilon_n) \in {\mathbb{Z}_2}^n$
4 **do** $l = \sum_{i=1}^{n} \epsilon_i a_i$
5 **if** $s = l$
6 **return**$(\epsilon_1, \epsilon_2, ..., \epsilon_n)$ — решение
7 **return**(не существует решения)

Алгоритм Meet-in-the-middle

Попытка поиска более эффективных алгоритмов решения задачи о рюкзаке приводит нас к алгоритмам типа «разделяй и властвуй». Основная идея заключается в разделении рюкзака $A = (a_1, \dots, a_n)$ на две части $A_1 = (a_1, \dots, a_t),\ A_2 = (a_{t+1}, \dots, a_n)$, где $t = \lfloor n/2 \rfloor$. Далее строится таблица

всех возможных сумм для одной части рюкзака s_1, которая сортируется по первой компоненте. На следующем шаге алгоритма вычисляются все возможные суммы другой части рюкзака s_2 и в построенной таблице с помощью бинарного поиска ищется сумма, равная $s_1 = s - s_2$.

Ниже приведён псевдокод для данного алгоритма.

SubSet-Sum2(A)

1 ▷ **INPUT**: подмножество натуральных чисел $\{a_1, \ldots, a_n\}$ и натуральное число s
2 ▷ **OUTPUT**: $\epsilon_i \in \{0,1\}, 1 \leqslant i \leqslant n$, такие, что $\sum_{i=1}^{n} \epsilon_i a_i = s$
3 Положить $t \leftarrow \lfloor n/2 \rfloor$
4 Создать таблицу $\left(\sum_{i=1}^{t} \epsilon_i a_i, (\epsilon_1, \epsilon_2, ..., \epsilon_t)\right)$ для $(\epsilon_1, \epsilon_2, ..., \epsilon_t) \in {\mathbb{Z}_2}^t$.
Отсортировать ее по первой компоненте.
5 Для каждого $(\epsilon_{t+1}, \epsilon_{t+2}, ..., \epsilon_n) \in {\mathbb{Z}_2}^{n-t}$
6 **do** Вычислить $l = s - \sum_{i=t+1}^{n} \epsilon_i a_i$,
7 проверить с помощью бинарного поиска, является ли l первой компонентой какой-либо ячейки таблицы
8 **if** $l = \sum_{i=1}^{t} \epsilon_i a_i$
9 **return**$(\epsilon_1, \epsilon_2, ..., \epsilon_n)$ — решение
10 **return**(не существует решения)

Анализ алгоритма Meet-in-the-middle можно провести в два этапа. На первом этапе алгоритма формируется таблица сумм для первой части рюкзака размером $2^{n/2}$ элементов для вычисления каждой суммы требуется $O(n/2)$ операций. Для сортировки k элементов можно применить алгоритм с сложностью $O(k \log k)$ [6, 7]. Таким образом количество шагов для первого этапа $O(\frac{n}{2}2^{n/2})$. На втором шаге в худшем случае необходимо вычислить все $2^{n/2}$ сумм для оставшейся части рюкзака и провести бинарный поиск в построенной таблице размером $2^{n/2}$, что потребует $O(\frac{n}{2}2^{n/2})$. Складывая количество шагов первого и второго этапов, получаем $O(n2^{n/2})$.

Несмотря на то, что данный алгоритм требует меньшего количества шагов, наличие экспоненциальной зависимости от n характеризует алгоритм как неэффективный. При этом в отличие от наивного алгоритма Meet-in-the-middle требует $O(2^{n/2})$ дополнительной памяти для хранения таблицы.

Сверхвозрастающий рюкзак

Определение 2.2. *(Сверхвозрастающий ранец). Ранец называется сверхвозрастающим, если в нём каждый элемент, начиная со второго, больше суммы всех предыдущих, т.е.*

$$a_i > \sum_{j=1}^{i-1} a_j, \quad \forall i \geq 2.$$

Интересным является то, что при для задачи о сверхвозрастающем ранце существует эффективный алгоритм решения.

SubSet-Sum3(B)

```
1  ▷ INPUT: Сверхвозрастющий рюкзак B = {b_1, ..., b_n}
   и натуральное число s, которое является суммой элементов из B
2  ▷ OUTPUT: ε_i ∈ {0,1}, 1 ⩽ i ⩽ n, такие, что Σ_{i=1}^{n} ε_i b_i = s
3  Положить i ← n
4  while i ≥ 1
5        do if s ≥ b_i
6              then ε_i ← 1
7                   s ← s − b_i
8              else  ε_i ← 0
9           i ← i − 1
10 return(ε_1, ..., ε_n).
```

Данный пример показывает, что некоторые задачи при небольшой модификации могут переходить между классами P и NP.

Задание на лабораторную работу

1. Реализуйте приведённые алгоритмы и сравните их производительность в зависимости от n.
2. Рассмотрите оптимизационный вариант задачи о ранце с целевой функцией $AE^T \to \max$ при ограничениях $AE^T \leq s$ [6]. Модифицируйте приведённые выше алгоритмы для решения данной задачи и проведите их анализ.
3. Рассмотрите непрерывный вариант предыдущей задачи [5] при $\epsilon_i \in [0, 1]$, $i = 1 \ldots n$. Найдите алгоритм её решения и проведите его анализ.

Вопросы для самопроверки

1. Что такое эффективность алгоритма? Какие алгоритмы называются эффективными?
2. Единственно ли решение задачи о сверхвозрастающем ранце?
3. Что такое классы P и NP [5, 8]?
4. Приведите примеры задач из класса P. Ответ обоснуйте.
5. Приведите примеры задач их класса NP.
6. Поясните суть проблемы N = NP [8].

Лабораторная работа №3
Алгоритмы в $\mathbb{Z}$

Краткая теоретическая справка

При изучении теоретико-числовых алгоритмов важную роль играет терминология теории чисел [9, 10]. Здесь мы приведём основные необходимые определения и описание наиболее важных алгоритмов.

Определение 3.1 (делимость)**.** *Пусть* $a, b \in \mathbb{Z}$ *и* $a \neq 0$*. Говорят, что* b *делится на* a *(или* a *делит* b*, или* b *кратно* a*, или* a *является делителем* b*) и пишут* $a|b$*, если*

$$\exists c \in \mathbb{Z} : b = ca.$$

Определение 3.2 (общий делитель)**.** $c \in \mathbb{Z}$ *называется общим делителем* a *и* b*, если* $c|a$ *и* $c|b$*.*

Наибольший общий делитель (НОД) или Greater Common Divisor (GCD) обозначается следующими символами:

$$(a, b), \quad gcd(a, b), \quad \text{НОД}(a, b).$$

Задача вычисления $gcd(a, b)$ решается алгоритмов Евклида, который является одним из самых древних известных алгоритмов [1]. Основная идея алгоритма основана на том, что $gcd(a, b) = gcd(b, a \mod b)$, $\forall a, b \in \mathbb{Z}$.

Алгоритм Евклида

Данный алгоритм вычисляет НОД для двух неотрицательных целых чисел. Ниже приведён псевдокод данного алгоритма.

GCD(a, b)
1 ▷ **INPUT**: два неотрицательных целых числа $a \geq b \geq 0$
2 ▷ **OUTPUT**: наибольший общий делитель a, b
3 **while** $b \neq 0$
4 **do**
5 $r \leftarrow a \mod b$
6 $a \leftarrow b$
7 $b \leftarrow r$
8 **return**(a)

Сложность алгоритма $O(\lg^2 n)$ битовых операций [1]. Стоит отметить, что данный алгоритм позволяет вычислять не только НОД для двух натуральных чисел, но может быть применён для других алгебраических структур (поля полиномов, вычетов и др.).

Расширенный алгоритм Евклида

Модификация алгоритма Евклида позволяет не только вычислять $d = gcd(a, b)$, но и находит целочисленное решение уравнения

$$ax + by = d, \quad x, y \in \mathbb{Z}.$$

Такая модификация называется расширенным алгоритмом Евклида (РАЕ) [1].

Определение 3.3. *(простые числа) Число $p \in \mathbb{Z}$, $p \geq 2$ называется простым, если оно делится только на 1 и на само себя.*

Определение 3.4. *(взаимно простые числа) $a, b \in \mathbb{Z}$ называются взаимно простыми, если $gcd(a, b) = 1$.*

EXTGCD(a, b)

```
 1  ▷ INPUT: два неотрицательных целых числа a ≥ b ≥ 0
 2  ▷ OUTPUT: d — НОД a, b и x, y ∈ Z: ax + by = d
 3  if b = 0
 4     then d ← a
 5          x ← 1, y ← 0
 6          return(d, x, y)
 7  x_2 ← 1, x_1 ← 0
 8  y_2 ← 0, y_1 ← 1
 9  while b > 0
10        do
11          q ← ⌊a/b⌋
12          r ← a − qb
13          x ← x_2 − qx_1, x_2 ← x_1, x_1 ← x
14          y ← y_2 − qy_1, y_2 ← y_1, y_1 ← y
15          a ← b, b ← r
16  x ← x_2
17  y ← y_2
18  return(a, x, y)
```

Сложность алгоритма $O(\lg^2 n)$ битовых операций.

Квадратичные вычеты и корни по модулю n

Рассмотрим уравнение

$$x^2 \equiv a \mod n \quad \text{в} \quad \mathbb{Z}_n^*. \tag{3}$$

Если (3) разрешимо, то a называется *квадратичным вычетом* по модулю n, а x — *квадратным корнем* по модулю n для a.

Пусть $Q_n \subset \mathbb{Z}_n^*$ — множество всех квадратичных вычетов по модулю n. Имеем

$$Q_n = (\mathbb{Z}_n^*)^2.$$

Положим

$$\overline{Q}_n = \mathbb{Z}_n^* \setminus Q_n.$$

В дальнейшем число n предполагается нечетным.

Теорема 1 о квадратичных вычетах и квадратных корнях по модулю простого нечетного числа $n = p$.

(i) $|Q_p| = \frac{p-1}{2}$. В частности,

$$|Q_p| = \left|\overline{Q}_p\right| = \frac{p-1}{2}.$$

(ii) Если x — квадратный корень по модулю p для a, то $-x$ также квадратный корень по модулю p для a, и любой квадратный корень y для a удовлетворяет тождеству $y \equiv \pm x \mod p$

(iii)

$$a^{\frac{p-1}{2}} \equiv \pm 1 \mod p \quad \text{для всех} \quad \mathbb{Z} \ni a \neq 0 \mod p.$$

(iv)

$$a \in Q_p \Leftrightarrow a^{\frac{p-1}{2}} \equiv 1 \mod p.$$

Алгоритм 1 нахождения квадратных корней для a по модулю простого нечетного числа $n = p$.

INPUT: нечетное простое число p и целое число a: $1 \leq a \leq p-1$.

OUTPUT: два квадратных корня для a по модулю p при условии $a \in Q_p$.

1. Вычислить символ Лежандра $\left(\frac{a}{p}\right) =: a^{\frac{p-1}{2}} \mod p$. Если $\left(\frac{a}{p}\right) = -1$, то a не имеет квадратных корней (т.е. $a \notin Q_p$) и остановиться.

2. Выбрать случайно $b \in \mathbb{Z}$: $1 \leq b \leq p-1$, пока не будет найдено b: $\left(\frac{b}{p}\right) = -1 \Leftrightarrow b \in \overline{Q}_p$.

3. Записать $p - 1 = 2^s t$, где t нечетно.

4. Найти $a^{-1} \mod p$ используя РАЭ.

5. Положить $c \leftarrow b^t \mod p$ и $r \leftarrow a^{(t+1)/2} \mod p$.

6. For $1 \leq i \leq s-1$ выполнить след.:

6.1 Вычислить $d = (r^2 \cdot a^{-1})^{2^{s-i-1}} \mod p$.

6.2 Если $d = -1 \mod p$, то положить $r \leftarrow r \cdot c \mod p$.

6.3 Положить $c \leftarrow c^2 \mod p$.

7. Return$(r, -r)$.

Алгоритм 1 при $s = 1$ сводится к алгоритму 2

Алгоритм 2 нахождения квадратных корней когда $p \equiv 3 \mod 4$
INPUT: нечетное простое число $p \equiv 3 \mod 4$ и целое число a:
$1 \leq a \leq p-1$.
OUTPUT: два квадратных корня для a по модулю p, при условии $a \in Q_p$.
1. Вычислить символ Лежандра $\left(\frac{a}{p}\right) =: a^{\frac{p-1}{2}} \mod p$. Если $\left(\frac{a}{p}\right) = -1$, то a не имеет квадратных корней (т.е. $a \notin Q_p$) и остановиться.
2. Вычислить $r = a^{(p+1)/4} \mod p$.
3. Return$(r, -r)$.

Алгоритм 1 при $s = 2$ сводится к алгоритму 3

Алгоритм 3 нахождения квадратных корней когда $p \equiv 5 \mod 8$
INPUT: нечетное простое число $p \equiv 5 \mod 8$ и целое число a: $1 \leq a \leq p-1$.
OUTPUT: два квадратных корня для a по модулю p, при условии $a \in Q_p$.
1. Вычислить символ Лежандра $\left(\frac{a}{p}\right) =: a^{\frac{p-1}{2}} \mod p$. Если $\left(\frac{a}{p}\right) = -1$, то a не имеет квадратных корней (т.е. $a \notin Q_p$) и остановиться.
2. Вычислить $d = a^{(p-1)/4} \mod p$.
3. Если $d = 1$, тогда вычислить $r = a^{(p+3)/8} \mod p$.
4. Если $d = p-1$, тогда вычислить
$r = 2a(4a)^{(p-5)/8} \mod p$.
5. Return$(r, -r)$.

Пример. $p = 331$, $a = 214$.
Найти $x : x^2 = 214 \mod 331$.
$331 = 3 \mod 4 \Rightarrow$ **Алгоритм 2**
$p - 1 = 330 = 2 \cdot 165$.
1.

$$\begin{aligned}\left(\frac{214}{331}\right) &= (214)^{\frac{331-1}{2}} \mod 331 = (214)^{165} \mod 331 \\ &= ((214)^2)^{82}(214) \mod 331 \\ &= (118)^{82}(214) \mod 331 \\ &= ((118)^2)^{41}(214) \mod 331 \\ &= (22)^{41}(214) \mod 331 \\ &= ((22)^4)^{10}(22)(214) \mod 331 \\ &= (239)^{10}(22)(214) \mod 331 \\ &= ((239)^2)^5(22)(214) \mod 331 \\ &= (189)^5(22)(214) \mod 331 \\ &= ((189)^2)^2(189)(22)(214) \mod 331 = (304)^2(189)(22)(214) \mod 331 \\ &= (67)(189)(22)(214) \mod 331 = 1 \mod 331 \Leftrightarrow 214 \in Q_{331}\end{aligned}$$

2.

$$\begin{aligned} r &= (214)^{\frac{331+1}{4}} \bmod 331 = (214)^{83} \bmod 331 \\ &= ((214)^2)^{41}(214) \bmod 331 \\ &= (118)^{41}(214) \bmod 331 \\ &= ((118)^2)^{20}(118)(214) \bmod 331 \\ &= (22)^{20}(118)(214) \bmod 331 \\ &= ((22)^4)^5(118)(214) \bmod 331 \\ &= (239)^5(118)(214) \bmod 331 \\ &= ((239)^2)^2(239)(118)(214) \bmod 331 \\ &= (189)^2(239)(118)(214) \bmod 331 \\ &= (304)(239)(118)(214) \bmod 331 = 144 \bmod 331 \end{aligned}$$

3. Корни $r = 144$ и $r = -144 = 187 \bmod 331$

Пример. $p = 277$, $a = 63$.
Найти $x : x^2 = 63 \mod 277$.
$277 = 5 \bmod 8 \Rightarrow$ **Алгоритм 3**
$p - 1 = 276 = 2^2 \cdot 3 \cdot 23$.

1.

$$\begin{aligned} \left(\frac{63}{277}\right) &= 63^{\frac{277-1}{2}} \bmod 277 = 63^{138} \bmod 277 \\ &= ((63)^2)^{69} \bmod 277 = (91)^{69} \bmod 277 \\ &= ((91)^2)^{34}(91) \bmod 277 = ((248)^2)^{17}(91) \bmod 277 \\ &= (10)^{17}(91) \bmod 277 \\ &= (((10)^2)^2)^4(10)(91) \bmod 277 \\ &= (28)^4(10)(91) \bmod 277 \\ &= (270)(10)(91) \bmod 277 = 1 \bmod 277. \end{aligned}$$

2.

$$\begin{aligned} d &= 63^{\frac{277-1}{4}} \bmod 277 = 63^{69} \bmod 277 \\ &= ((63)^2)^{34}(63) \bmod 277 = (91)^{34}(63) \bmod 277 \\ &= ((91)^2)^{17}(63) \bmod 277 \\ &= (248)^{17}(63) \bmod 277 \\ &= ((248)^2)^8(248)(63) \bmod 277 \\ &= (28)^2(248)(63) \bmod 277 \\ &= (230)(248)(63) \bmod 277 = 276 \bmod 277 = p - 1 \bmod 277. \end{aligned}$$

3.

$$\begin{aligned} r &= 2(63)(4\cdot 63)^{\frac{277-5}{8}} \bmod 277 \\ &= 2(63)(4\cdot 63)^{34} \bmod 277 \\ &= 2(63)((252)^2)^{17} \bmod 277 \\ &= 2(63)(71)^{17} \bmod 277 \\ &= 2(63)(71)((71)^2)^8 \bmod 277 \\ &= 2(63)(71)(55)^8 \bmod 277 \\ &= 2(63)(71)((55)^2)^4 \bmod 277 \\ &= 2(63)(71)(255)^4 \bmod 277 \\ &= 2(63)(71)((255)^2)^2 \bmod 277 \\ &= 2(63)(71)(207)^2 \bmod 277 \\ &= 2(63)(71)(191) \bmod 277 \\ &= 150 \bmod 277 \end{aligned}$$

4. Корни $r = 150$ и $r = -150 = 127 \bmod 277$

Теорема 2 о квадратичных вычетах и квадратных корнях по модулю составного числа $n = pq$, где p, q — различные нечетные простые числа.
(i) $|Q_p| = \frac{\varphi(n)}{4} = \frac{(p-1)(q-1)}{4}$. В частности,

$$\left|\overline{Q}_p\right| = \frac{3\varphi(n)}{4} = \frac{3(p-1)(q-1)}{4}.$$

(ii) Если $a \in Q_n$, то a имеет в $\mathbb{Z}_n^*$ 4 различных квадратных корня.
(iii) Имеем

$$a \in Q_n \;\Leftrightarrow\; a \bmod p \in Q_p \quad \text{и} \quad a \bmod q \in Q_q.$$

Алгоритм 4 нахождения квадратных корней для a по модулю составного числа $n = pq$, где p, q — различные нечетные простые числа.

INPUT: целое число n с простыми факторами p, q и $a \in Q_n$.
OUTPUT: 4 квадратных корня для a по модулю n.

1. С помощью алг.1 (или алг.2 или алг.3 если они применимы) найти два квадратных корня r, $-r$ для a по модулю p.
2. С помощью алг.1 (или алг.2 или алг.3 если они применимы) найти два квадратных корня s, $-s$ для a по модулю q.
3. С помощью РАЕ найти $c, d \in \mathbb{Z}$: $cp + dq = 1$.

4. Положить $x \leftarrow (rdq + scp) \mod n$ и
$y \leftarrow (rdq - scp) \mod n$.
5. Return$(\pm x \mod n, \pm y \mod n)$.

Пример $p = 331$, $q = 277$, $n = pq = 91687$,
$a = 62111$. Найти $x : x^2 = 62111 \mod 91687$.
1. $62111 = 214 \mod 331 \Rightarrow$ (см. прим.3)
$r = 114 \mod 331$, $-r = 187 \mod 331$
2. $62111 = 63 \mod 277 \Rightarrow$ (см. прим.4)
$s = 150 \mod 277$, $-s = 127 \mod 277$
3. $c = 118$, $d = -141$
4. $x = (rdq + scp) \mod n = 51118 \Rightarrow$
$r_1 = x = 51118$
$r_2 = -x \mod 91687 = 40569$.
$y = (rdq - scp) \mod n = 69654 \Rightarrow$
$r_3 = y = 69654$
$r_4 = -y \mod 91687 = 22033$.
5. 4 квадратных корня
$r_1 = x = 51118$, $r_2 = 40569$, $r_3 = 69654$, $r_4 = 22033$.

Задание на лабораторную работу

1. Разработайте программную реализация расширенного алгоритма Евклида.
2. Используя результаты первой лабораторной работы, напишите программу для вычисления символа Лежандра.
3. Разработайте программу для решения уравнения (3). Алгоритм решения определяется согласно варианту задания.
4. Для $\mathbb{Z}_n^*$, где n определяется исходя из варианта задания, определить все квадратичные вычеты и соответствующие им корни уравнения (3).

Вопросы для самопроверки

1. Приведите пример самого большого простого числа, которые вы знаете.
2. Приведите пример взаимно простых чисел.
3. Обоснуйте утверждение о том, что $gcd(a, b) = gcd(b, a \mod b)$.
4. Что означает запись $\mathbb{Z}_n^*$?
5. Что такое модульная арифметика (арифметика остатков)?
6. Всегда ли разрешимо уравнение $ax = b \mod n$ в $\mathbb{Z}_n$, $\mathbb{Z}_n^*$?
7. Как найти решение уравнения $ax = b \mod n$ в $\mathbb{Z}_n^*$?
8. Всегда ли разрешимо уравнение $x^2 = a \mod n$ в $\mathbb{Z}_n^*$?
9. Как найти решение уравнения $x^2 = a \mod n$ в $\mathbb{Z}_n^*$?
10. Покажите как алгоритмы 2 и 3 получаются из алгоритма 1.

Лабораторная работа №4
Анализ алгоритмов сортировки

Краткая теоретическая справка

Задача сортировки последовательности богата существующими алгоритмами её решения [6, 7, 11]. В данной работе предлагается рассмотреть и реализовать некоторые из алгоритмов сортировки и сравнить их производительность.

Вначале сформулируем задачу сортировки.

Определение 4.1. *(задача сортировки) Задача сортировки заключается в построении алгоритма со следующими входом и выходом:*
INPUT: *последовательность* $A = (a_1, \cdots, a_n)$ *сравнимых элементов некоторого множества.*
OUTPUT: *перестановка* $A' = (a'_1, \cdots, a'_n)$, *в которой* $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Все алгоритмы сортировки можно разделить на четыре класса:

- вставками;
- выбором;
- обменные;
- слиянием;

Далее рассмотрим по одному представителю их каждого класса.

Сортировка вставками

В данной сортировке предлагается, продвигаясь от начала последовательности к её концу, сохранять на каждом i-м шаге следующий инвариант $a'_1 \leq a'_2 \leq \cdots \leq a'_i$, что на последнем шаге при $i = n$ приводит к искомой перестановке.

Ниже приведёно описание алгоритма на псевдокоде.

INSERTION-SORT(A)

```
1  for j ← 2 to length[A]
2        do key ← A[j]
3           ▷ добавить A[j] к отсортированной части A[1 .. j − 1].
4           i ← j − 1
5           while i > 0 and A[i] > key
6                 do A[i + 1] ← A[i]
7                    i ← i − 1
8           A[i + 1] ← key
```

Сортировка прямым выбором

Идея сортировки выбором заключается в поиске максимального элемента в неупорядоченной части последовательности и его последующего исключения путём записи в конец.

SELECTION-SORT(A)

```
1  for i ← length[A] to 2
2        do
3           m ← MAX(A[1..i])
4           A[m] ↔ A[i] ▷ обменять местами
```

Пузырьковая сортировка

При пузырьковой сортировке в последовательности сравниваются соседние элементы и если они не упорядоченны, то меняются местами.

BUBBLE-SORT(A)

```
1  for i ← 1 to length[A] − 1
2        do for j ← length[A] downto i + 1
3                 do if A[j] < A[j − 1]
4                       then A[j] ↔ A[j − 1]
```

Сортировка слиянием

Данный алгоритм сортировки относится к классу «разделяй и властвуй». На каждом шаге алгоритма последовательность делится пополам и каждая часть сортируется отдельно. Затем последовательности сливаются с сохранением упорядоченности с помощью процедуры MERGE.

MERGE(A, p, q, r)

```
 1  n1 ← q − p + 1, n2 ← r − q
 2  ▷ Создаем массивы L[1..n1 + 1] и R[1..n2 + 1]
 3  for i ← 1 to n1
 4        do L[i] ← A[p + i − 1]
 5  for j ← 1 to n2
 6        do R[j] ← A[q + j]
 7  L[n1 + 1] ← ∞, R[n2 + 1] ← ∞
 8  i ← 1, j ← 1
 9  for k ← p to r
10        do if L[i] ≤ R[j]
11              then A[k] ← L[i]
12                   i ← i + 1
13              else  A[k] ← R[j]
14                   j ← j + 1
```

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

Задание на лабораторную работу

1. Для каждого из приведённых алгоритмов найдите оценку для количества шагов и количества требуемой памяти.
2. Реализуйте алгоритм сортировки согласно варианту задания.
3. Реализуйте более эффективные алгоритмы сортировки [5, 6, 7, 11] согласно варианту задания.
4. Сравните производительность различных алгоритмов.

Вопросы для самопроверки

1. Для элементов каких множеств можно корректно поставить задачу сортировки?
2. Что такое стабильная сортировка?
3. Сколько существует всевозможных перестановок из n элементов?
4. Что такое беспорядок?
5. Каково максимально возможное число беспорядков в последовательности из n элементов?
6. Определите временную и простанственную сложности для приведённых алгоритмов.
7. Какова сложность оптимального алгоритма сортировки в худшем случае?
8. Всегда ли существует решение задачи сортировки?
9. Единственно ли решение задачи сортировки?

Список литературы

1. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. — 3-е изд. — М.:Вильямс, 2006.

2. Абельсон Х., Сассман Дж., Сассман Дж. Структура и интерпретация компьютерных программ. — М.:Добросвет, 2010.

3. Кнут Д., Грэхем Р., Паташник О. Конкретная математика. Основание информатики. — 2-е изд. — М.:Мир, 2009.

4. Кнут Д. Искусство программирования, том 2. Получисленные алгоритмы. — 3-е изд. — М.:Вильямс, 2007.

5. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. — 2-е изд. — М.:Вильямс, 2011.

6. Вирт Н. Алгоритмы и структуры данных. — М.:ДМК Пресс, 2011.

7. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. — М.:Вильямс, 2010.

8. Босс В. Перебор и эффективные алгоритмы. — М.:URSS, 2008.

9. Босс В. Теория чисел. — М.:URSS, 2009.

10. Menezes A., Oorschot P., Vanstone S. Handbook of applied cryptography. — CRC Pr., 1996.

11. Кнут Д. Искусство программирования, том 3. Сортировка и поиск. — 2-е изд. — М.:Вильямс, 2007.

Содержание