Алгоритмы и структуры данных

практические занятия

Марквирер Владлена Дмитриевна
vdmarkvirer@hse.ru

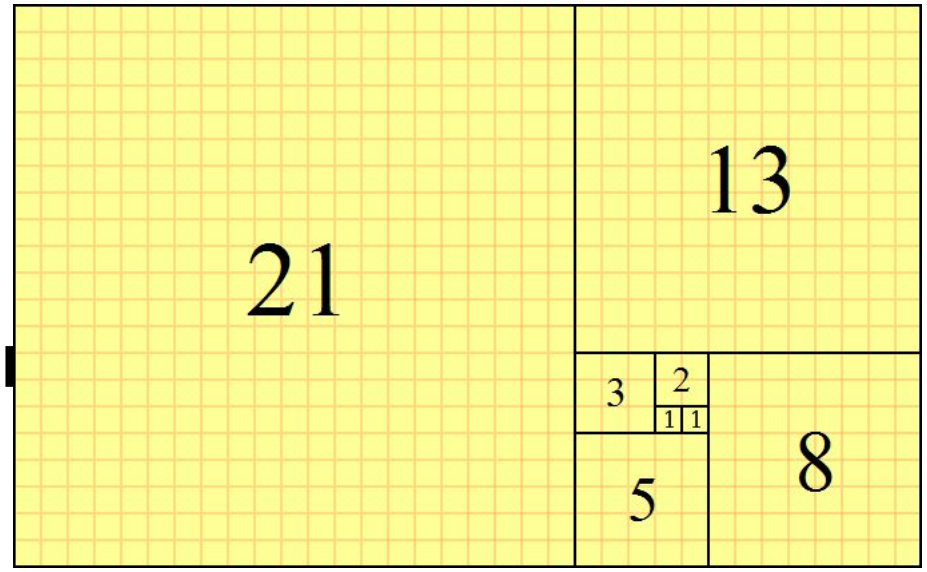
О практиках

- Работа в группах из 3-х человек.
- Постарайтесь сформировать группы, в которых будет хотя бы один уверенный программист.
- Каждая практика – решение основных задач + задачи повышенной сложности (для повышения баллов за практики).
- То, что не успеете решить – выносится на дз, с обязательной защитой в начале следующей практики, иначе работа принята не будет.
- Используем любой известный Вам язык программирования, но все алгоритмы пишем самостоятельно, не берём готовые библиотеки, и методы.
- Оценивание будет производиться за каждую задачу каждому человеку в команде по результатам защиты кода (по необходимости), алгоритма и тестов (по необходимости) в трёхзначной шкале (+ ; +/- ; -).

Практика №1

«Программирование рекурсивных процедур и функций»

Основная задача



- **Понять и реализовать 3 различны алгоритма нахождения чисел Фибоначчи.**
- **Подсказка:** один алгоритм рекурсивный, два – итерационных.
- Сравните эффективность (по времени, используемой памяти и т.п.) каждого алгоритма и докажите, какой будет лучше и почему.

Помните, что рекурсию не всегда можно свести к итерации, но эта задача – не тот случай, тут всё хорошо!

Задача повышенной сложности

- **Рекурсивно вычислить определитель матрицы разложением по строке/столбцу.**

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & a_{ij} & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

$$\det \mathbf{A} = a_{11} \cdot M_{11} - a_{12} \cdot M_{12} + a_{13} \cdot M_{13} - \ldots + (-1)^{1+n} a_{1n} \cdot M_{1n}$$

где a_{ij} – элемент матрицы **A**, стоящий на пересечении *i*-той строки с *j*-тым столбцом матрицы, а M_{ij} – его минор, т.е. определитель *n*–1-го порядка,

- Возможно ли вычислить итерационно?

Практика №2

«Продолжение работы с рекурсивными и итерационными алгоритмами»

Задача с прошлого занятия

- **Рекурсивно вычислить определитель матрицы разложением по строке/столбцу.**

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & a_{ij} & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

$$\det \mathbf{A} = a_{11} \cdot M_{11} - a_{12} \cdot M_{12} + a_{13} \cdot M_{13} - \ldots + (-1)^{1+n} a_{1n} \cdot M_{1n}$$

где a_{ij} – элемент матрицы **A**, стоящий на пересечении *i*-той строки с *j*-тым столбцом матрицы, а M_{ij} – его минор, т.е. определитель *n*–1-го порядка,

- Возможно ли вычислить итерационно?

Задача на определитель матрицы

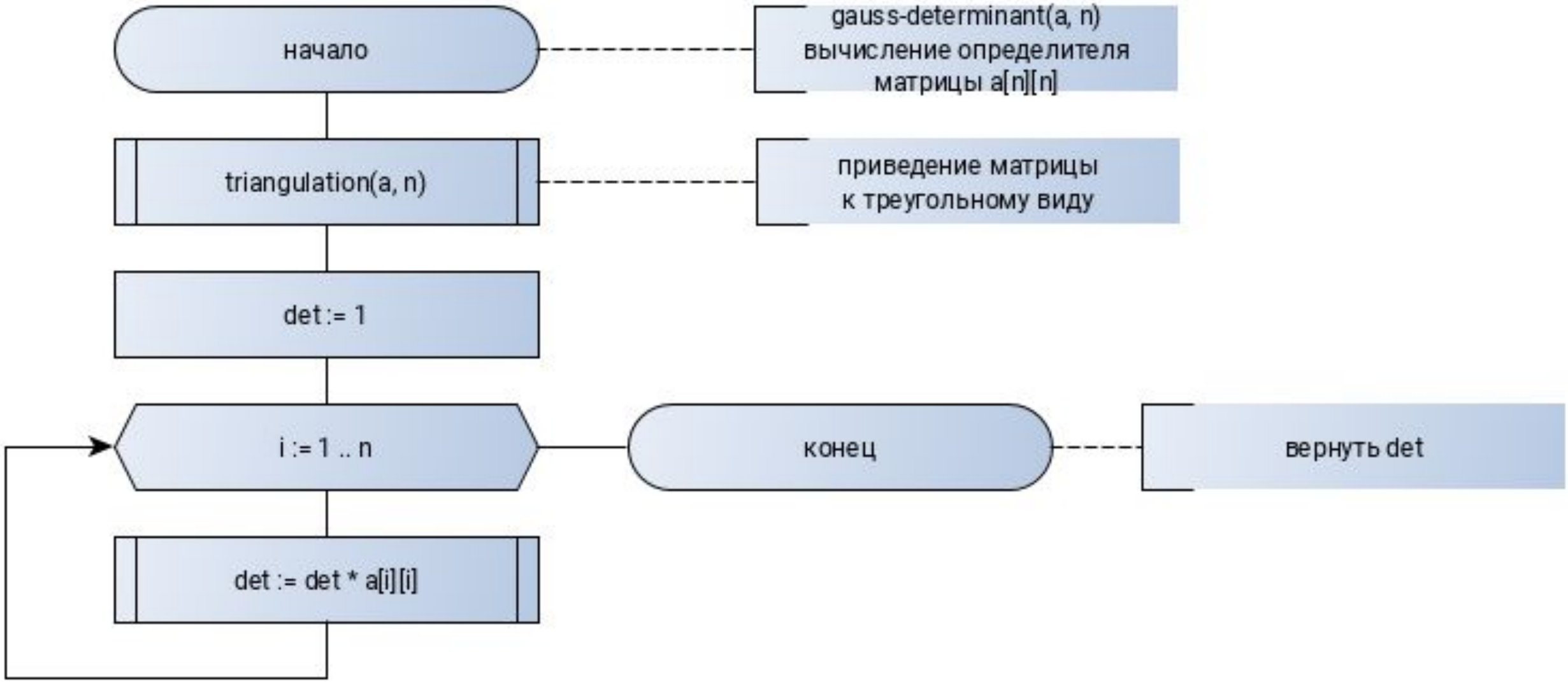
- **Вычислить рекурсивно определитель матрицы с помощью метода Гаусса.**

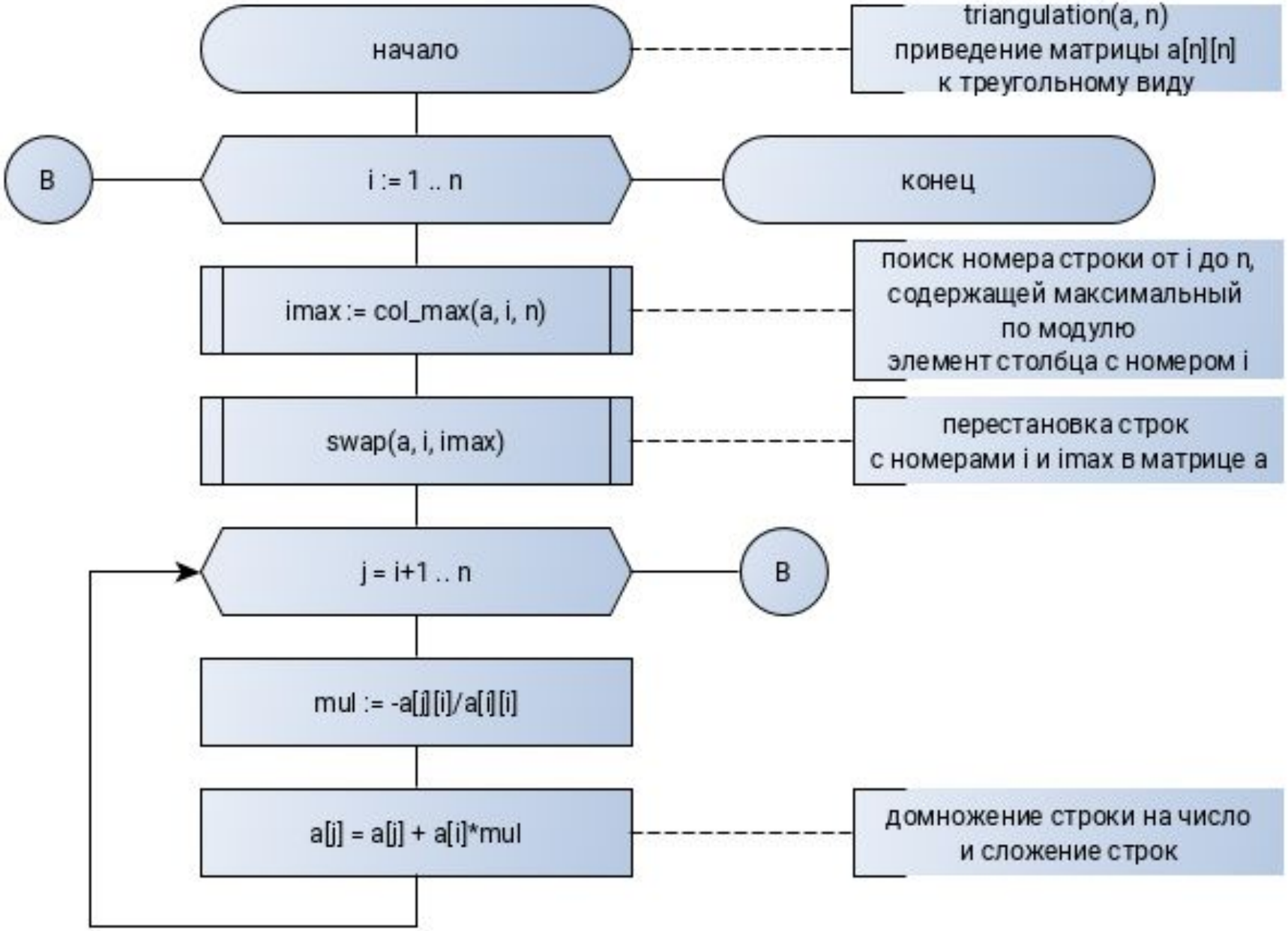
$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & a_{ij} & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

$$\begin{pmatrix} 73 & 7 & 6 \\ 110 & 16 & 19 \\ 148 & 10 & 7 \end{pmatrix} \Rightarrow \begin{pmatrix} 73 & 7 & 6 \\ 0 & 5{,}45 & 9{,}96 \\ 148 & 10 & 7 \end{pmatrix} \Rightarrow \begin{pmatrix} 73 & 7 & 6 \\ 0 & 5{,}45 & 9{,}96 \\ 0 & -4{,}19 & -5{,}16 \end{pmatrix} \Rightarrow \begin{pmatrix} 73 & 7 & 6 \\ 0 & 5{,}45 & 9{,}96 \\ 0 & 0 & 2{,}49 \end{pmatrix}$$

$$\Delta = 73 * 5{,}45 * 2{,}49 = 992$$

- Какой метод более точный и почему?
- Какой метод более быстрый и почему?


начало
gauss-determinant(a, n)
вычисление определителя
матрицы a[n][n]
triangulation(a, n)
приведение матрицы
к треугольному виду
det := 1
i := 1 .. n
конец
вернуть det
det := det * a[i][i]


начало
triangulation(a, n)
приведение матрицы a[n][n]
к треугольному виду
B
i := 1 .. n
конец
imax := col_max(a, i, n)
поиск номера строки от i до n,
содержащей максимальный
по модулю
элемент столбца с номером i
swap(a, i, imax)
перестановка строк
с номерами i и imax в матрице a
j = i+1 .. n
B
mul := -a[j][i]/a[i][i]
a[j] = a[j] + a[i]*mul
домножение строки на число
и сложение строк

Практика №3

«Рекуррентные соотношения и итерационный алгоритм»

Задача по генерации перестановок

- Рекурсивная генерация всех n-факториал перестановок. Параметр n задаётся от 0 до 9.
- n – входной параметр (количество разрядов).

- Два варианта (посчитать количество перестановок):
  - перестановка без повторений (всего n! перестановок);
  - перестановка с повторениями (всего n^n перестановок) – дополнительное задание.

- Вывод в лексикографическом порядке или другом (объяснить выбор).

Задача по генерации перестановок

- Например, найти все возможные перестановки для последовательности чисел 1, 2, 3.
  Существуют следующие перестановки:

```
1: 1 2 3
2: 1 3 2
3: 2 1 3
4: 2 3 1
5: 3 1 2
6: 3 2 1
```

Задача по генерации перестановок

- Дополнительно дублирую слайды с лекции:

Пример 2 (комбинаторика). Один из самых часто встречающихся комбинаторных объектов – это перестановка первых N натуральных чисел. Известно, что существует ровно N! таких перестановок. Вот, например, все 6 перестановок чисел 1,2,3, записанные в *лексикографическом* порядке:

(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).

При решении некоторых задач методом полного перебора требуется сгенерировать все N! перестановок, не пропустив случайно какую-нибудь, и не сгенерировав какую-нибудь перестановку дважды. Существует рекурсивный алгоритм, позволяющий это сделать. В отличие от рассмотренного выше рекурсивного алгоритма Евклида в данном примере мы будем иметь дело не с рекурсивной *функцией*, а с рекурсивной *процедурой*. На прошлой лекции у нас уже был пример двух рекурсивных процедур, вызывающих взаимно друг друга и генерирующих набор символьных строк в алфавите {0,1} с определёнными свойствами.

Рекурсивная процедура генерации всех перестановок из N элементов 1,2,3,…,N получает на вход один единственный параметр – натуральное число N. Она вызывает саму себя, но с параметром N – 1, и получает список всех (N – 1)! перестановок из элементов 1,2,3,…,N – 1 (ниже в таблице T_0 они перечислены в лексикографическом порядке):

1	2	3	…	N – 2	N – 1
1	2	3	…	N – 1	N – 2
…	…	…	…	…	…
N – 2	N – 1	N – 3		2	1
N – 1	N – 2	N – 3	…	2	1

Затем число N вставляется в каждую строку таблицы T_0 перед её первым элементом и получается таблица T_1:

N	1	2	3	…	N – 2	N – 1
N	1	2	3	…	N – 1	N – 2
N	…	…	…	…	…	…
N	N – 2	N – 1	N – 3		2	1
N	N – 1	N – 2	N – 3	…	2	1

Очевидно, что все строки таблицы T_1 отличаются между собой. Затем в таблице T_1 меняются местами первый и второй столбец. Получается таблица T_2:

1	N	2	3	…	N – 2	N – 1
1	N	2	3	…	N – 1	N – 2
…	N	…	…	…	…	…
N – 2	N	N – 1	N – 3		2	1
N – 1	N	N – 2	N – 3	…	2	1

Очевидно, что все её строки различаются между собой и отличаются от строк таблицы T_1. Далее в таблице T_2 меняются местами второй и третий столбец и получается таблица T_3:

1	2	N	3	…	N – 2	N – 1
1	2	N	3	…	N – 1	N – 2
…	…	N	…	…	…	…
N – 2	N – 1	N	N – 3		2	1
N – 1	N – 2	N	N – 3	…	2	1

Далее каждая следующая таблица T_{k+1} получается из предыдущей таблицы T_k перестановкой k-го и $(k+1)$-го столбцов. Последняя таблица T_N будет иметь вид:

1	2	3	…	N – 2	N – 1	N
1	2	3	…	N – 1	N – 2	N
…	…	…	…	…	…	N
N – 2	N – 1	N – 3		2	1	N
N – 1	N – 2	N – 3	…	2	1	N

Очевидно, что все строки таблиц $T_1, T_2, T_3, \ldots, T_N$ попарно различны, суммарное количество строк равно N! и все они представляют собой искомые перестановки чисел 1,2,3,…,N (без повторений и пропусков).

Задача по вычислению ленточного определителя

- Вычисление ленточного определителя (из презентации с лекции):
  - итерационный способ;
  - на основе выведенной математической формулы.

- Вспомогательные материалы по ленточной матрице:
  - Исходное состояние – Трёхдиагональная матрица: https://ru.wikipedia.org/wiki/Трёхдиагональная_матрица, решается с помощью метода прогонки: https://ru.wikipedia.org/wiki/Метод_прогонки. Можно попробовать подумать над вариантом пяти и более диагональных матриц (по желанию).
  - Математическая выкладка: https://scask.ru/i_book_alg_s.php?id=420&ysclid=l88lafgae453071270

- Далее продублированы слайды из презентации с лекции.

Пример 1. Требуется вычислить определитель n-го порядка

$$\begin{vmatrix} 6 & 9 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 6 & 9 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 6 & 9 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 6 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 6 & 9 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 6 & 9 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 6 \end{vmatrix}.$$

Решение. Обозначим искомый определитель через x_n. Тогда, вычисляя его разложением по первой строке, а затем возникший определитель $(n-1)$-го порядка – разложением по первому столбцу, приходим к равенству

$$x_n = \begin{vmatrix} 6 & 9 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 6 & 9 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 6 & 9 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 6 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 6 & 9 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 6 & 9 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 6 \end{vmatrix} = 6 \cdot \begin{vmatrix} 6 & 9 & 0 & \dots & 0 & 0 & 0 \\ 1 & 6 & 9 & \dots & 0 & 0 & 0 \\ 0 & 1 & 6 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 6 & 9 & 0 \\ 0 & 0 & 0 & \dots & 1 & 6 & 9 \\ 0 & 0 & 0 & \dots & 0 & 1 & 6 \end{vmatrix} - 9 \cdot \begin{vmatrix} 6 & 9 & 0 & \dots & 0 & 0 & 0 \\ 1 & 6 & 9 & \dots & 0 & 0 & 0 \\ 0 & 1 & 6 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 6 & 9 & 0 \\ 0 & 0 & 0 & \dots & 1 & 6 & 9 \\ 0 & 0 & 0 & \dots & 0 & 1 & 6 \end{vmatrix} =$$

$$= 6 \cdot x_{n-1} - 9 \cdot \begin{vmatrix} 6 & 9 & \dots & 0 & 0 & 0 \\ 1 & 6 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 6 & 9 & 0 \\ 0 & 0 & \dots & 1 & 6 & 9 \\ 0 & 0 & \dots & 0 & 1 & 6 \end{vmatrix} = 6 \cdot x_{n-1} - 9 \cdot x_{n-2}.$$

Таким образом мы получили линейное однородное рекуррентное соотношение с постоянными коэффициентами глубины 2. Чтобы его решить, надо добавить два начальных условия:

$$x_1 = 6,\ x_2 = \begin{vmatrix} 6 & 9 \\ 1 & 6 \end{vmatrix} = 6 \cdot 6 - 9 \cdot 1 = 27.$$

Применим описанный выше метод решения рекуррентных соотношений:

1) составим и решим характеристическое уравнение. Оно в данном случае будет иметь вид:

$$\lambda^n = 6 \cdot \lambda^{n-1} - 9 \cdot \lambda^{n-2},$$

или после упрощения

$$\lambda^2 - 6 \cdot \lambda + 9 = 0.$$

Его корни $\lambda_1 = 3$, $\lambda_2 = 3$, т.е. характеристическое уравнение имеет один корень, равный 3, кратности 2. Далее, согласно пункту 3, ответ к рекуррентному соотношению запишем в виде *формулы общего решения*

$$x_n = P_1(n) \cdot 3^n,$$

где $P_1(n)$ – полином от n степени 1, т.е. выражение вида $a \cdot n + b$. Константы a и b найдём из начальных условий $x_1 = 6$, $x_2 = 27$. Подставляя вместо n числа 1 и 2 в формулу общего решения

$$x_n = (a \cdot n + b) \cdot 3^n,$$

получим систему уравнений для нахождения a и b:

$$\begin{cases} 6 = (a + b) \cdot 3, \\ 27 = (2a + b) \cdot 9. \end{cases}$$

Эта система имеет единственное решение $a = b = 1$. В итоге получаем окончательный ответ, т.е. формулу *частного решения*

$$x_n = (n+1) \cdot 3^n.$$

Сделаем проверку, вычислив x_3 по нашей формуле и исходя из постановки задачи. Согласно полученной формуле $x_3 = 4 \cdot 3^3 = 108$. Исходя из постановки задачи, получаем такой же результат

$$x_3 = \begin{vmatrix} 6 & 9 & 0 \\ 1 & 6 & 9 \\ 0 & 1 & 6 \end{vmatrix} = 6 \cdot \begin{vmatrix} 6 & 9 \\ 1 & 6 \end{vmatrix} - 9 \cdot \begin{vmatrix} 1 & 9 \\ 0 & 6 \end{vmatrix} = 6 \cdot (36 - 9) - 9 \cdot 6 = 162 - 54 = 108.$$

В рассмотренном примере корни характеристического уравнения оказались совпадающими, т.е. был один корень, равный 3, кратности 2. Если бы корни были разные, например, 3 и (–3), то *формула общего решения* была бы такая

$$x_n = a \cdot 3^n + b \cdot (-3)^n,$$

где коэффициенты a и b далее нужно было бы вычислить, исходя из начальных условий $x_1 = 6$, $x_2 = 27$.

Практика №4

«Рекурсивное (по двум параметрам) и итерационное вычисление НОД и НОК (быстрый и медленный алгоритм Евклида)»

Задача по вычислению НОД и НОК

- Вычислить НОД (англ. greatest common divisor) и НОК (англ. least common multiple):
  - реализовать итерационный алгоритм;
  - реализовать рекурсивный алгоритм.
  - оцените время работы алгоритмов.

- Алгоритм Евклида для нахождения НОД - один из первых алгоритмов в истории, использовался ещё в Древней Греции, и дошёл до наших дней. В изначальном виде он назывался “взаимным вычитанием”, так как заключался в поочерёдном вычитании меньшего числа из большего, пока одно из них не станет равным 0. Сегодня чаще всего вместо вычитания используется взятие остатка от деления, но суть алгоритма сохранилась. Алгоритм основан на построении ряда чисел следующего вида (a>b $a, b, r_1, r_2, \ldots, r_n$

Задача по вычислению НОД и НОК

Где каждое последующее число является остатком от деления предпредыдущего на пр

$$r_1 = a \bmod b$$
$$r_2 = b \bmod r_1$$
$$\dots$$
$$r_n = r_{n-2} \bmod r_{n-1}$$

Ряд заканчивается, когда остаток от деления предпоследнего числа на последнее становится равным 0:

$$r_{n-1} \bmod r_n = 0$$

В таком случае утверждается, что:

$$\gcd(a, b) = r_n$$

Задача по вычислению НОД и НОК

- Дополнительно можете доказать описанное ранее утверждение, а также попробуйте реализовать другой / другие алгоритмы.

- Для вычисления НОК используется нахождение НОД, тогда в общем виде фор[illegible]ОК будет выглядеть так:

$$lcm(a, b) = \frac{a * b}{\gcd(a, b)}$$

Практика №5
«Взаимная и двойная рекурсия»

Задача по взаимной рекурсии

- Необходимо сгенерировать все строки длины n в алфавите {0, 1}, в которых нет двух подряд идущих нулей.
- Например, при n = 3, результатом будет {010, 011, 110, 101, 111}.
- Задача: реализовать рекурсивный алгоритм генерации таких строк, при этом если будет выбран переборный алгоритм, то максимальная оценка за это задание 0,4 балла (т.к. сложность алгоритма экспоненциальная). Перебор можно использовать для проверки.
- Воспользуйтесь материалами из лекции по взаимной рекурсии.

Задача по двойной рекурсии

- Рекурсивным методом вычислить число сочетаний C_n^k, где n = 0, 1, 2, …; k = 0, 1, 2, …, n.
- **Первый метод (треугольник Паскаля):** использовать свойство биномиальных коэффициентов, где первое слагаемое вычисляется из предыдущих, а второе – через первое. Осуществляется два вызова функции. $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$
- **Второй метод:** через рекуррентное соотношение. $C_n^k = \frac{n}{k} \cdot C_{n-1}^{k-1}$.
- Сравнить время работы двух методов.

Задача по двойной рекурсии (продолжение)

- Дополнительные задания:
- Реализовать итерационные методы вычисления числа сочетаний C_n^k, где n = 0, 1, 2, …; k = 0, 1, 2, …, n.
- **1 способ:** использовать двумерную матрицу.
- **2 способ:** использовать одномерную матрицу и вспомогательные переменные.
- Сравнить время работы алгоритмов как между собой, так и относительно рекурсивных алгоритмов.
- Используйте материалы лекции по двойной рекурсии.

Спасибо за внимание!