

Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

## Курсовая работа

Дисциплина: Программирование на языках высокого уровня

Тема: Алгоритм Нарайаны

Выполнил

студент гр. 3331506/70401

Жернаков А. А.

Преподаватель

Ананьевский М. С.

«    » \_\_\_\_\_ 2020 г.

Санкт-Петербург

2020 г.

# Оглавление

|  |   |
|--|---|
| 1. Введение .....                                      | 3 |
| 1.1 Формулировка задачи, которую решает алгоритм ..... | 3 |
| 1.2 Словесное описание алгоритма .....                 | 3 |
| 2. Реализация алгоритма .....                          | 4 |
| 3. Анализ алгоритма.....                               | 5 |
| 3.1 Анализ сложности алгоритма .....                   | 5 |
| 3.2 Численный анализ алгоритма .....                   | 5 |
| 4. Применение алгоритма .....                          | 7 |
| 5. Заключение .....                                    | 8 |
| Список литературы .....                                | 9 |

# 1. Введение

## 1.1 Формулировка задачи, которую решает алгоритм

В комбинаторике зачастую возникает задача генерации всех возможных перестановок из  $n$  элементов. Существует множество способов решить эту задачу.

В данной работе рассмотрен алгоритм Нарайаны, приведена его реализация на языке программирования, произведена оценка сложности и численный анализ алгоритма, описано его применение.

Алгоритм Нарайаны - рекурсивный алгоритм, генерирующий по данной перестановке следующую за ней перестановку (в лексикографическом порядке). Придуман индийским математиком Пандитом Нарайаной в XIV веке.

Особенностью алгоритма является то, что для генерации всех перестановок необходимо запоминать только одну текущую перестановку.

## 1.2 Словесное описание алгоритма

Пусть дана  $n$ -элементная перестановка  $a = \{a_1, a_2, a_3, \dots, a_n\}$ .

Шаг 1: Найти такой наибольший  $j$ , для которого  $a_j < a_{j+1}$ . Если такого элемента не существует, то  $a$  – наибольшая  $n$ -элементная перестановка. Работа алгоритма завершена.

Шаг 2: Увеличить  $a_j$ . Для этого надо найти наибольшее  $l > j$ , для которого  $a_l > a_j$ . Затем поменять местами  $a_j$  и  $a_l$ .

Шаг 3: Записать последовательность  $a_{j+1}, \dots, a_n$  в обратном порядке.

## 2. Реализация алгоритма

Алгоритм был реализован на языке C++.

Функция, реализующая алгоритм Нарайаны, представлена на рисунке 1.

```
void narayana(int array[], const int n)
{
    // реализация первого шага
    int i = n - 2;
    while (i >= 0 && array[i] > array[i + 1])
        i--;
    // реализация второго шага
    int j = n - 1;
    while (array[j] < array[i])
        j--;
    swap(array[i], array[j]);
    // реализация третьего шага
    for (int t = i + 1, k = n - 1; t < k; t++, k--)
        swap(array[t], array[k]);
    // вывод новой перестановки в консоль
    for (i = 0; i < n; i++)
        cout << array[i];
}
```

Рисунок 1 – Алгоритм Нарайаны

## 3. Анализ алгоритма

### 3.1 Анализ сложности алгоритма

Наилучшим является случай, когда элементы расположены по возрастанию. В этом случае произойдет 2 сравнения и 1 обмен.

Наихудшей является ситуация, при которой первый элемент меньше второго, а все последующие (с третьего - по последний) меньше первого и расположены в порядке убывания. Если всего в перестановке  $n$  элементов, то произойдет  $2(n - 1)$  сравнений и  $n/2$  обменов при четном  $n$  и  $(n + 1)/2$  обменов при нечетном  $n$ .

В результате сложность алгоритма можно оценить как  $O(n)$ .

### 3.2 Численный анализ алгоритма

Посчитаем время генерации всех возможных перестановок для массивов разных размеров.

На рисунке 2 приведен график зависимости времени выполнения от количества элементов (время указано в мс).



Рисунок 2 – Численный анализ

Численные значения приведены в таблице 1.

Таблица 1 – Численный анализ

| Число<br>элементов         | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 10  | 11   | 12   | 13     |
|----------------------------|---|---|---|---|---|---|---|---|----|-----|------|------|--------|
| Время<br>выполнения,<br>мс | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 6 | 56 | 618 | 5790 | 6456 | 874519 |

Как видно из графика, для перестановок, состоящих от 1 до 6 элементов, время приблизительно равно 0. С последующим увеличением количества элементов время начинает резко возрастать. Таким образом, для перестановки из 13 элементов время выполнения составляет уже 14,5 минут. Такие затраты времени неприемлемы. Для генерации перестановок с большим количеством элементов потребуются более производительные мощности.

## 4. Применение алгоритма

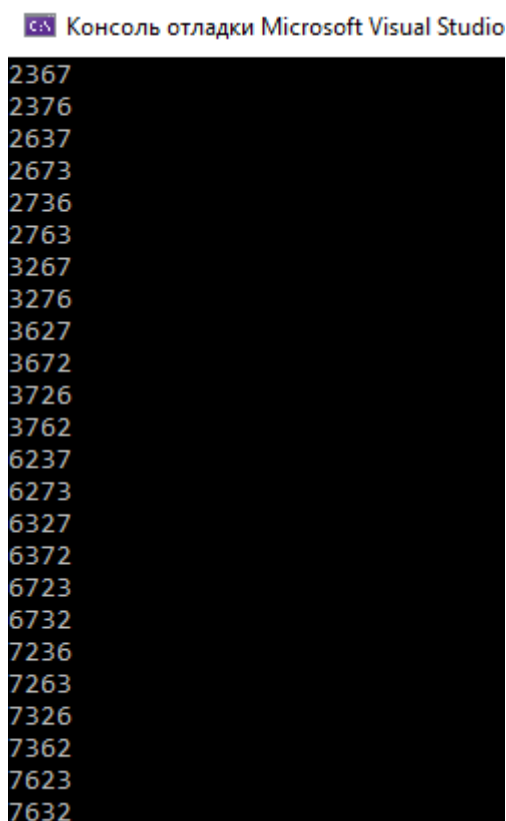
Основным применением алгоритма Нарайаны является вывод всех возможных перестановок из  $n$  элементов.

На рисунке 3 представлена функция, которая реализует это применение.

```
// применение алгоритма Нарайаны
void application(int array[], int n)
{
    bubble_Sort(array, n); // сортировка массива по возрастанию
    for (int g = 0; g < n; g++)
        cout << array[g];
    int f = factorial(n); // нахождение количества возможных перестановок
    for (int r = 1; r < f; r++)
    {
        cout << "\n";
        narayana(array, n);
    }
}
```

Рисунок 3 – Применение алгоритма Нарайаны

Результат выполнения данной функции представлен на рисунке 4.



Консоль отладки Microsoft Visual Studio

```
2367
2376
2637
2673
2736
2763
3267
3276
3627
3672
3726
3762
6237
6273
6327
6372
6723
6732
7236
7263
7326
7362
7623
7632
```

Рисунок 4 – Результат

## **5. Заключение**

В ходе выполнения работы было рассмотрены принцип работы алгоритма Нарайаны, его реализация на языке программирования C++, проведен анализ сложности и численный анализ алгоритма, а также рассмотрено его применение для решения определенных задач.

Таким образом, Алгоритм Нарайаны является очень простым и эффективным при решении задачи генерации всех возможных перестановок из  $n$  элементов, для чего он используется и в настоящее время.



## **Список литературы**

1. Knuth, D. E. The Art of Computer Programming. — Addison-Wesley, 2005. — Vol. 4. — ISBN 0-201-85393-0.