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

Студент гр.3331506\80001 Ющенко Д.В.

Преподаватель Ананьевский М.С.

Санкт-Петербург

2021

Оглавление

[1.Введение 3](#_Toc71899883)

[1.1.Применение алгоритма 3](#_Toc71899884)

[2.Работа алгоритма 4](#_Toc71899885)

[2.1.Принцип работы алгоритма 4](#_Toc71899886)

[3.Анализ алгоритма 5](#_Toc71899887)

[3.1.Оценка скорости работы алгоритма 5](#_Toc71899888)

[3.2.Численное исследование алгоритма 5](#_Toc71899889)

[Список литературы 6](#_Toc71899890)

[Приложение 7](#_Toc71899891)

# 1.Введение

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

Алгоритм Нарайаны – это нерекурсивный 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.

## 1.1.Применение алгоритма

В области математики, называемой комбинаторика, возникает необходимость генерации всех возможных перестановок из n элементов, что можно свести к следующей задаче: по данной перестановке сгенерировать следующую за ней перестановку (например, в лексикографическом порядке). Таким образом, основное применением алгоритма Нарайаны - генерация всех возможных перестановок из n элементов.

# 2.Работа алгоритма

## 2.1.Принцип работы алгоритма

Алгоритм состоит из трёх шагов:

1. Поиск , для которого . Установить . Если , то необходимо уменьшить на 1 повторно, пока не выполнится условие . Если окажется, что , завершить алгоритм.

* На втором шаге является наименьшим индексом, для которого были посещены все перестановки, начиная с . Следовательно, лексикографически следующая перестановка увеличит значение .

1. Увеличение . Необходимо найти наибольшее , для которого , уменьшить на 1, пока не выполнится условие . Затем необходимо поменять местами .

* Поскольку элемент является наименьшим элементом, который больше , и который может следовать за в перестановке. Перед заменой выполнялись отношения , а после замены – выполняются .

1. Необходимо записать последовательность в обратном порядке.

# 3.Анализ алгоритма

## 3.1.Оценка скорости работы алгоритма

В лучшем случае – то есть когда элементы расположены по возрастанию произойдёт 2 сравнения и 1 обмен. В худшем случае, при котором первый элемент меньше второго, а все последующие (с третьего - по последний) меньше первого и расположены в порядке убывания и, если всего в перестановке n элементов, то произойдет сравнений и обменов при четном n и обменов при нечетном n. В результате сложность алгоритма можно оценить как O(n)

## 3.2.Численное исследование алгоритма

Таблица 1 – Численное исследование

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Время, мс | 0 | 0 | 0 | 0 | 8 | 620 | 6465 |
| Количество элементов | 1 | 2 | 3 | 5 | 8 | 10 | 12 |

Рисунок 1 – График зависимости времени от количества элементов

Как можно видеть из графика, время необходимое на поиск полного количества перестановок после 10 элементов начинает резко возрастать

# Список литературы

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

2.Алгоритм Нарайаны // [Wikipedia]. URL: https://ru.wikipedia.org/wiki/Алгоритм \_Нарайаны

# Приложение

void narayana(int \*array, const int n) {

int i, j, k, l;

for(i = n - 2; (i > 0) && (array[i] > array [i+1]); i--); //Реализация первого шага

for(j = n - 1; array[j] < array[i]; j--); //Реализация второго шага

swap(array[i], array[j]);

for(l = i + 1, k = n - 1; l < k; l++, k--) { //Реализация третьего шага

swap(array[l], array[k]);

}

}