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

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 j, для которого $a_j < a_{j+1}$. Установить $j \leftarrow n-2$. Если $a_j > a_{j+1}$, то необходимо уменьшить j на 1 повторно, пока не выполнится условие $a_j < a_{j+1}$. Если окажется, что $j = 0$, завершить алгоритм.
    - На втором шаге j является наименьшим индексом, для которого были посещены все перестановки, начиная с $a_1, \dots, a_j$. Следовательно, лексикографически следующая перестановка увеличит значение a_j.
2) Увеличение a_j. Необходимо найти наибольшее $l > j$, для которого $a_j > a_l$, уменьшить l на 1, пока не выполнится условие $a_j < a_l$. Затем необходимо поменять местами $a_j \leftrightarrow a_l$.
    - Поскольку $a_{j+1} < a_n$ элемент a_l является наименьшим элементом, который больше a_j, и который может следовать за $a_1, \dots, a_{j-1}$ в перестановке. Перед заменой выполнялись отношения $a_{j+1} > \cdots > a_{l-1} > a_l > a_j > a_{l+1} > \cdots > a_n$, а после замены – выполняются $a_{j+1} > \cdots > a_{l-1} > a_j > a_l > a_{l+1} > \cdots > a_n$.
3) Необходимо записать последовательность $a_{j+1,}, \dots, a_n$ 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 $2(n - 1)$ 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.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

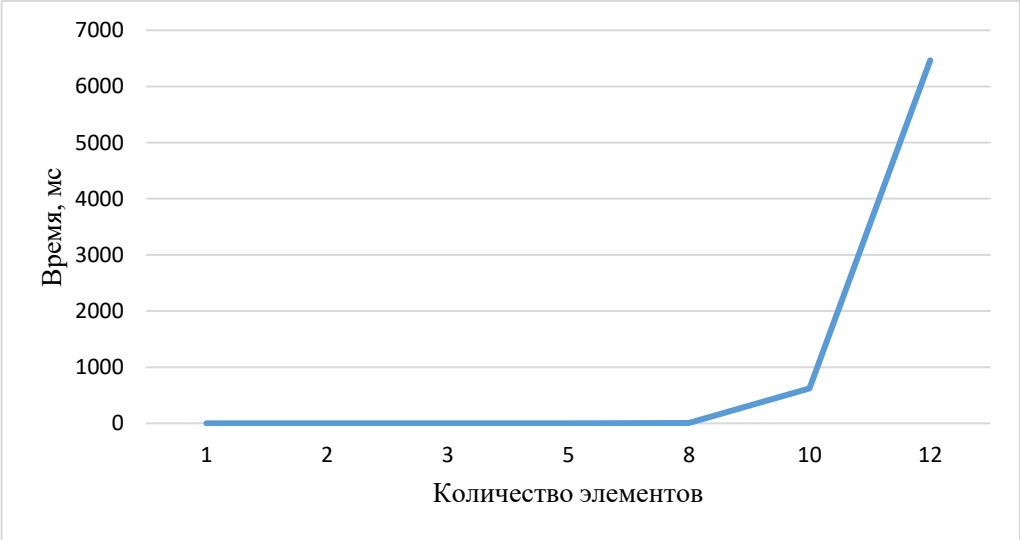

Рисунок 1 – График зависимости времени от количества элементов

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]);
    }
}
```