МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с основными понятиями и приемами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++. Сопоставить рекурсивное решение с итеративным решением задачи, сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Формулировка задачи: Вариант 4. Напечатать все перестановки заданных n различных натуральных чисел (или символов).

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

Для решения поставленной задачи было принято создать класс Permutations, содержащий поля с количеством элементов в каждой перестановке, указателем на массив и вектор из элементов, участвующих в перестановках, а также методы для печати текущей перестановки на экран void Permutations::print_current(int *el), поиска следующей перестановки bool Permutations::next_permutation() и перебора всех перестановок рекурсивным void Permutations::print_recursively(int c) и итеративным void Permutations::print_iteratively() способами.

```
class Permutations {
private:
    int n;
    int *it;
    int *tmp;
    vector <int> elements;
private:
    void print_current(int *el);
    bool next_permutation();
    void print_iteratively();
    void print_recursively(int c = 0);
public:
    void run();
    Permutations(int n, int* el);
};
```

Метод void Permutations::print_recursively(int c) на каждом шаге извлекает поочередно доступные элементы из вектора, добавляет их во временный массив и переходит на следующий шаг рекурсии, после чего возвращает пройденный элемент в список доступных. При достижении глубины, когда доступен всего один элемент, метод добавляет его во временных массив и вызывает функцию печати текущей перестановки.

Метод void Permutations::print_iteratively() считает количество всех возможных перестановок, после чего в цикле поочередно вызывает функции печати текущей перестановки и поиска следующей перестановки.

Метод bool Permutations::next_permutation() находит и сохраняет в массив следующую перестановку с помощью алгоритма Нарайаны.

Сравнение рекурсивного и итеративного способов было решено осуществлять по времени выполнения, для этого был написан метод void Permutations::run(), который осуществлял подсчет времени выполнения

каждой реализации, а также разницу во времени с помощью функции clock_t clock().

Пример работы алгоритма представлен на Таблице 1.

Глубина рекурсии	Массив элементов	Временный массив	Вывод
0	1, 2, 3		
1	2, 3	1	
2	3	1, 2	
3		1, 2, 3	1 2 3
2	3	1, 2	
1	2, 3	1	
2	2	1, 3	
3		1, 3, 2	1 3 2
2	2	1, 3	
1	2, 3	1	
0	1, 2, 3		
1	1, 3	2	
2	3	2, 1	
3		2, 1, 3	2 1 3
2	3	2, 1	
1	1, 3	2	
2	1	2, 3	
3		2, 3, 1	2 3 1
2	1	2, 3	
1	1, 3	2	
0	1, 2, 3		
1	1, 2	3	
2	2	3, 1	
3		3, 1, 2	3 1 2
2	2	3, 1	
1	1, 2	3	
2	1	3, 2	
3		3, 2, 1	3 2 1

3. ТЕСТИРОВАНИЕ

3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ

Программа собрана в операционной системе Ubuntu 18.04.1 LTS bionic компилятором g++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0. В других ОС и компиляторах тестирование не проводилось.

3.2 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестовые случаи представлены в Приложении А.

Во время тестирования была обнаружена ошибка: программа уходила в бесконечный цикл при вводе символов, отличных от цифр. В программу были добавлены необходимые проверки, чтобы исключить данную ошибку.

По результатам тестирования было выявлено, что при достаточно низком количестве элементов рекурсивная реализация не сильно уступает по скорости работы итеративной, однако существенно проигрывает при большом количестве элементов.

4. ВЫВОД

В ходе выполнения данной работы были изучены основные принципы рекурсивного программирования. Были созданы функции для решения поставленной задачи рекурсивным и итеративным методом. Рекурсивная реализация оказалась незначительно медленнее при переборе перестановок из небольшого количества элементов (меньше 10), однако оказалась значительно медленнее в ситуациях, когда количество элементов в каждой перестановке превышало 10. При 9 элементах рекурсивная реализация оказалась медленнее примерно на 0.2 секунды быстрее, тогда как при 11 отставание составило примерно 77 секунд.

5. ПРИЛОЖЕНИЕ А: ТЕСТОВЫЕ СЛУЧАИ

Количество элементов	Вывод рекурсивно	Вывод итеративно	Время рекурсивно	Время итеративно	Разница во времени
-3	Something went wrong.				
0	Something went wrong.				
2	0 1 1 0	0 1 1 0	35	24	11 мкс.
3	0 1 2 0 2 1 1 0 2 1 2 0 2 0 1 2 1 0	0 1 2 0 2 1 1 0 2 1 2 0 2 0 1 2 1 0	102	55	47 мкс.
4	0 1 2 3 0 1 3 2 0 2 1 3 0 2 3 1 0 3 1 2 0 3 2 1 1 0 2 3 1 0 3 2 1 2 0 3 1 2 3 0 1 3 0 2 1 3 2 0 2 0 1 3 2 0 3 1 2 1 0 3 2 1 3 0 2 3 0 1 2 3 1 0 3 0 1 2 3 0 2 1 3 1 0 2 3 1 2 0 3 2 0 1 3 2 1 0	0 1 2 3 0 1 3 2 0 2 1 3 0 2 3 1 0 3 1 2 0 3 2 1 1 0 2 3 1 0 3 2 1 2 0 3 1 2 3 0 1 3 0 2 1 3 2 0 2 0 1 3 2 0 3 1 2 1 0 3 2 1 3 0 2 3 0 1 2 3 1 0 3 0 1 2 3 0 2 1 3 1 0 2 3 1 2 0 3 2 0 1 3 2 1 0	355	251	104 мкс.
9			32 с.	29,8 с.	0.21с.
11			395 с.	318 с.	77 с.

6. ПРИЛОЖЕНИЕ Б: ИСХОДНЫЙ КОД

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <ctime>

using namespace std;

class Permutations {
private:
    int n;
    int *it;
    int *tmp;
    vector <int> elements;

private:
    void print_current(int *el);
    bool next_permutation();
    void print_iteratively();
    void print_recursively(int c = 0);

public:
    void run();
    Permutations(int n, int* el);
};

int get_n ()
{
    string s;
    cout << "Enter amount of elements: ";
    cin >> s;
    if (!isdigit(s[0]) || stoi(s) <= 0)
    {
        cout << "Invalid value: n = " << s << endl;
        getline(cin, s);
        return 0;
    }
    return stoi(s);
}

int get_el()
```

```
{
    string s;
    cin >> s;
    if (!isdigit(s[0]) || stoi(s) <= 0)
    {
        return 0;
    }
    return stoi(s);
}

int main()
{
    int n, c;
    int *el;
    while(true)
    {
        cout << "Press 1 to get input from a file\n" <<
                "Press 2 to enter each element by yourself\n" <<
                "Press 3 to enter only amount of elements\n" <<
                "Press 4 to exit." << endl;
        cin >> c;
        switch (c)
        {
            case 1:
                break;
            case 2:
                if ((n = get_n()) == 0)
                    continue;
                el = new int[n];
                cout << "Enter elements:";
                for (int i = 0; i < n; i++)
                {
                    el[i] = get_el();
                }
                break;
            case 3:
                if ((n = get_n()) == 0)
                    continue;
                el = new int[n];
                for (int i = 0; i < n; i++)
                    el[i] = i;
                break;
            case 4:
                return 0;
```

```
    default:
        cout << "Something went wrong. try again!" << endl;
        continue;
}
if (c == 1)
{
    string s;
    cout << "Enter file name: ";
    cin >> s;
    ifstream f;
    f.open(s);
    if (!f)
    {
        cout << "Unable to open the file!" << endl;
        continue;
    }
    while(!f.eof())
    {
        bool flag = false;
        n = 0;
        getline(f, s);
        if (!isdigit(s[0]))
            continue;
        stringstream ss;
        ss << s;
        ss >> s;
        n = stoi(s);
        if (n <= 0)
            continue;
        el = new int[n];
        for (int i = 0; i < n; i++)
        {
            ss >> s;
            if (s.empty() || !isdigit(s[0]))
            {
                flag = true;
                break;
            }
            el[i] = stoi(s);
            s.clear();
        }
        if (flag)
            continue;
        Permutations p(n, el);
```

```
                p.run();
                delete[] el;
            }
            f.close();
        }
        else
        {
            Permutations p(n, el);
            p.run();
            delete[] el;
        }
    }
    cin >> n;
}

void Permutations::run()
{
    clock_t t1s, t1e, t2s, t2e, d1, d2;
    cout << "Printing Iteratively" << endl;
    t1s = clock();
    print_iteratively();
    t1e = clock();
    cout << "Printing Recursively" << endl;
    t2s = clock();
    print_recursively();
    t2e = clock();
    d1 = t1e - t1s;
    d2 = t2e - t2s;
    cout << "Iteratively it takes " << d1 << " µs." << endl;
    cout << "Recursively it takes " << d2 << " µs." << endl;
    cout << (d2 < d1 ? "Recursively" : "Iteratively") << " is faster
on "
                                                    << abs(d2 - d1) <<
" µs.\n" << endl;
}

Permutations::Permutations(int n, int* el) : n(n), it(el) {
    tmp = new int[n];
    for (int i = 0; i < n; i++)
        elements.push_back(el[i]);
};

void Permutations::print_current(int *el)
{
```

```
    for (int i = 0; i < n; i++)
        cout << el[i] << " ";
    cout << endl;
}

bool Permutations::next_permutation()
{ // next permutation algorithm by Narayana
    int j = n - 2;
    int k = n - 1;
    // find first decreasing element from the right
    while(j >= 0 && it[j] >= it[j + 1])
        j--;
    // if j < 0 there is no next permutation
    if (j < 0)
        return false;
    // find first element that larger than it[j]
    while(it[j] >= it[k])
        k--;
    swap(it[j], it[k]);
    // reverse the elements that are to the right for it[j]
    int l = n - 1;
    int r = j + 1;
    while(r < l)
    {
        swap(it[r], it[l]);
        r++;
        l--;
    }
    return true;
}

void Permutations::print_iteratively()
{
    int j = 1;
    for (int i = 1; i <= n; i++)
        j *= i;
    for (j; j > 0; j--)
        {
            print_current(it);
            next_permutation();
        }
}

void Permutations::print_recursively(int c)
```

```
{
    if (c == n - 1)
    {
        tmp[c] = elements.back();
        print_current(tmp);
        return;
    }
    for (int i = 0; i < elements.size(); i++)
    {
        tmp[c] = elements[i];
        elements.erase(elements.begin() + i);
        print_recursively(c + 1);
        elements.push_back(tmp[c]);
        sort(elements.begin(), elements.end());
    }
}

Permutations::~Permutations()
{
    delete[] tmp;
}
```