ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | В. А. Кузнецов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3.2 |
| АЛГОРИТМ С ИСПОЛЬЗОВАНИЕМ РЕКУРСИИ И ЦИКЛОВ |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#__RefHeading___Toc801_316780651)

[2 Схема алгоритма решения 4](#__RefHeading___Toc809_316780651)

[3 Полное описание реализованных функций 5](#__RefHeading___Toc807_316780651)

[3.1 get\_partial\_permutations\_with\_debug 5](#__RefHeading___Toc3907_3112333877)

[3.2 write\_to\_file 5](#__RefHeading___Toc3913_3112333877)

[3.3 read\_from\_file 6](#__RefHeading___Toc3913_31123338771)

[3.4 print\_permutations 6](#__RefHeading___Toc3913_31123338772)

[3.5 main 7](#__RefHeading___Toc3913_311233387721)

[4 Листинг программы 8](#__RefHeading___Toc805_316780651)

[5 Результаты тестирования программы 10](#__RefHeading___Toc4883_751357292)

[6 Вывод по результату тестирования 13](#__RefHeading___Toc803_3167806511)

1. Постановка задачи

Задача: реализовать программную функцию на языке C/С++, выполняющую поставленную задачу с использованием рекурсии. Вариант задания, пример входных и выходных данных представлен в таблице 1.

* Написать код функции, принимающей в качестве аргументов и возвращающей все необходимые параметры, без использования глобальных переменных. Допустимо использование дополнительных функций.
* Протестировать функцию для всех возможных исключительных ситуаций, особое значение придается тестам на возникновение ошибок в ходе работы программы.
* Из наименования функции и принимаемых аргументов должно быть ясно их назначение.
* В ходе тестирования функции при каждом вызове рекурсивной функции необходимо вывести отладочную информацию: порядковый номер вызова рекурсивной функции, значения изменяющегося аргумента и возвращаемого значения, если они присутствуют. Привести глубину рекурсии для каждого тестового примера.

Таблица 1 – Вариант

|  |  |  |  |
| --- | --- | --- | --- |
| N | Текст задания | Вход | Выход |
| 2 | Дан одномерный массив A размерностью N. Сохранить бинарно в файл все возможные размещения элементов массива из K элементов. | N: 3  A: [1,2,3]  K: 2 | [1,2]  [1,3]  [2,1]  [2,3]  [3,1]  [3,2] |

1. Схема алгоритма решения

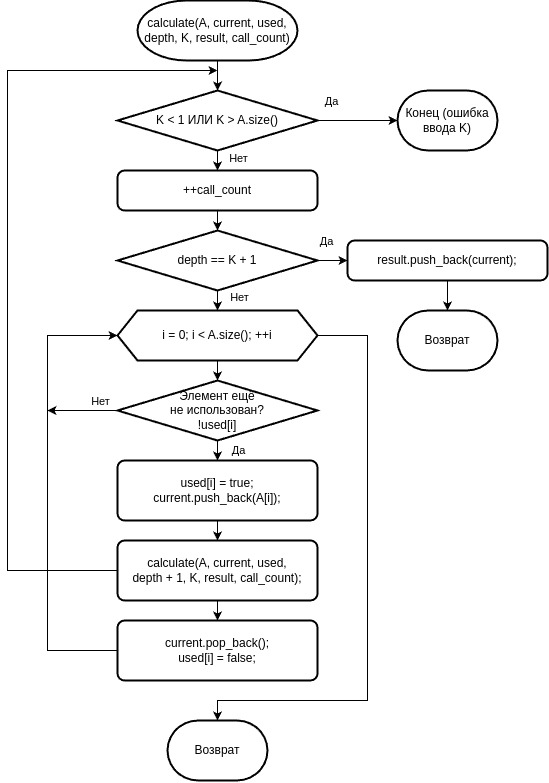


Рисунок 1 – Блок-схема алгоритма

1. Полное описание реализованных функций
   1. get\_partial\_permutations\_with\_debug

Функция get\_partial\_permutations\_with\_debug генерирует все возможные размещения длины K из элементов массива A и сохраняет их в вектор result. В процессе работы функция выводит отладочную информацию о каждом вызове рекурсии, глубине вызова и состоянии текущего размещения. Принимает следующие аргументы:

1. const std::vector<int> &A: Исходный массив элементов.
2. std::vector<int> &current: Вектор для хранения текущего частичного размещения.
3. std::vector<bool> &used: Вектор флагов для отслеживания использования элементов.
4. int depth: Текущая глубина рекурсивного вызова.
5. int K: Длина размещения.
6. std::vector<std::vector<int>> &result: Вектор для хранения всех найденных размещений.
7. int &call\_count: Счётчик вызовов функции для отладки.

Ничего не возвращает, результат записывается в result. Работа функции происходит следующим образом:

1. Проверяет корректность значений K.
2. Увеличивает счётчик вызовов и выводит отладочную информацию.
3. Если текущая глубина равна K+1, сохраняет текущее размещение и возвращается из рекурсии.
4. В противном случае, перебирает все элементы массива A и, если элемент ещё не использован, добавляет его в текущее размещение и вызывает рекурсивную функцию с увеличенной глубиной.
5. После возвращения из рекурсивного вызова удаляет последний элемент из текущего размещения и отмечает элемент как неиспользованный.
   1. write\_to\_file

Функция write\_to\_file записывает в бинарный файл все размещения, переданные в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::string &filename: Имя файла для записи.
2. const std::vector<std::vector<int>> &permutations: Вектор векторов, содержащий все размещения.

Ничего не возвращает. Работа функции происходит следующим образом:

1. Открывает файл для записи в бинарном режиме.
2. Проверяет успешность открытия файла, если неудачно — выводит сообщение об ошибке и завершает программу.
3. Пишет каждое размещение в файл, используя метод write для побайтовой записи данных.
4. Закрывает файл.
   1. read\_from\_file

Функция read\_from\_file читает из бинарного файла все размещения и возвращает их в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::string &filename: Имя файла для чтения.
2. int K: Длина размещения.

Возвращает вектор векторов, содержащий все прочитанные размещения. Работа функции происходит следующим образом:

1. Открывает файл для чтения в бинарном режиме.
2. Проверяет успешность открытия файла, если неудачно — выводит сообщение об ошибке и завершает программу.
3. Читает данные из файла в буфер фиксированного размера (размера K), пока файл не закончится.
4. Сохраняет каждое прочитанное размещение в результирующий вектор.
5. Закрывает файл и возвращает результирующий вектор.
   1. print\_permutations

Функция print\_permutations выводит все размещения, переданные в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::vector<std::vector<int>> &permutations: Вектор векторов, содержащий все размещения.

Ничего не возвращает. Работа функции происходит следующим образом:

1. Перебирает каждое размещение из переданного вектора.
2. Для каждого размещения выводит его элементы в консоль.
   1. main
3. Вводит массив элементов и значение K.
4. Генерирует все возможные размещения длины K из элементов массива A с помощью функции get\_partial\_permutations\_with\_debug.
5. Записывает найденные размещения в бинарный файл permutations.bin.
6. Читает размещения из файла и выводит их на экран.
7. Выводит глубину рекурсии для данной задачи.
8. Листинг программы

Листинг 1

#include <iostream>  
#include <fstream>  
#include <vector>  
#include <sstream>  
  
void get\_partial\_permutations\_with\_debug(const std::vector<int> &A, std::vector<int> &current,  
 std::vector<bool> &used, int depth, int K,  
 std::vector<std::vector<int>> &result, int &call\_count) {  
 if (K < 1) {  
 std::cout << "Ошибка: K < 1";  
 exit(1);  
 }  
 if (K > A.size()) {  
 std::cout << "Ошибка: элементов меньше, чем K";  
 exit(1);  
 }  
  
 int current\_call\_count = ++call\_count;  
 std::cout << "Вызов #" << current\_call\_count << ": depth=" << depth << ", current={ ";  
 for (int i : current) std::cout << i << " ";  
 std::cout << "}" << std::endl;  
  
 // Базовый случай: прекращаем рекурсию  
 if (depth == K + 1) {  
 result.push\_back(current);  
 std::cout << "- Возврат из вызова #" << current\_call\_count << std::endl;  
 return;  
 }  
  
 // Рекурсивный случай: продолжаем составление  
 for (size\_t i = 0; i < A.size(); ++i)  
 if (!used[i]) {  
 used[i] = true;  
 current.push\_back(A[i]);  
 get\_partial\_permutations\_with\_debug(A, current, used, depth + 1, K, result, call\_count);  
 current.pop\_back();  
 used[i] = false;  
 }  
  
 std::cout << "- Возврат из вызова #" << current\_call\_count << std::endl;  
 return;  
}  
  
void write\_to\_file(const std::string &filename, const std::vector<std::vector<int>> &permutations) {  
 std::ofstream file(filename, std::ios::binary);  
 if (!file) {  
 std::cout << std::endl << "Ошибка открытия файла для записи";

Продолжение листинга 1

exit(1);  
 }  
 for (const auto &permutation : permutations)  
 file.write(reinterpret\_cast<const char \*>(permutation.data()), permutation.size() \* sizeof(int));  
 file.close();  
}  
  
std::vector<std::vector<int>> read\_from\_file(const std::string &filename, int K) {  
 std::ifstream file(filename, std::ios::binary);  
 if (!file) {  
 std::cout << std::endl << "Ошибка открытия файла для чтения";  
 exit(1);  
 }  
 std::vector<std::vector<int>> permutations;  
 std::vector<int> permutation(K);  
 while (file.read(reinterpret\_cast<char \*>(permutation.data()), K \* sizeof(int)))  
 permutations.push\_back(permutation);  
 file.close();  
 return permutations;  
}  
  
void print\_permutations(const std::vector<std::vector<int>> &permutations) {  
 for (const auto &permutation : permutations) {  
 std::cout << "{ ";  
 for (int item : permutation) std::cout << item << " ";  
 std::cout << "}" << std::endl;  
 }  
}  
  
int main() {  
 // Ввод массива  
 std::string input;  
 std::vector<int> A;  
 std::cout << "Массив: ";  
 std::getline(std::cin >> std::ws, input);  
 std::stringstream ss(input);  
 std::string word;  
 while (ss >> word) A.push\_back(std::stoi(word));  
  
 int K;  
 std::cout << "K: ";  
 std::cin >> K;  
  
 std::vector<std::vector<int>> permutations;  
 std::vector<int> current;  
 std::vector<bool> used(A.size(), false);  
 int call\_count = 0;  
  
 get\_partial\_permutations\_with\_debug(A, current, used, 1, K, permutations, call\_count);  
  
 // Запись размещений в файл  
 std::cout << "Запись в файл permutations.bin...";

Продолжение листинга 1

std::string filename = "/home/grigorijtomczuk/Desktop/suai/op/lab3.2/permutations.bin";  
 write\_to\_file(filename, permutations);  
 std::cout << " ✔" << std::endl;  
  
 // Чтение и вывод размещений из файла  
 std::cout << "Чтение из файла permutations.bin...";  
 std::vector<std::vector<int>> readPermutations = read\_from\_file(filename, K);  
 std::cout << " ✔" << std::endl;  
 print\_permutations(readPermutations);  
  
 // Глубина рекурсии для данной задачи всегда равна K+1  
 std::cout << "Глубина: " << K + 1;  
  
 return 0;  
}

1. Результаты тестирования программы

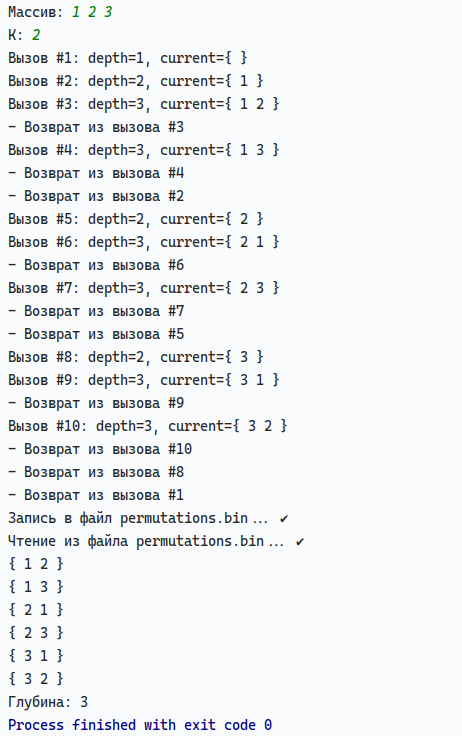


Рисунок 2

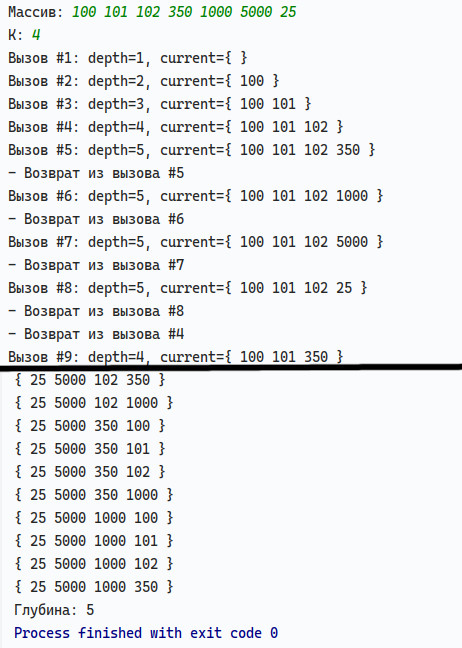


Рисунок 3

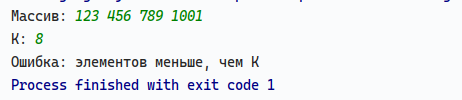


Рисунок 4

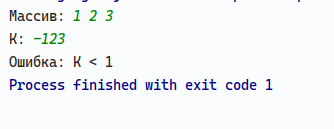


Рисунок 5

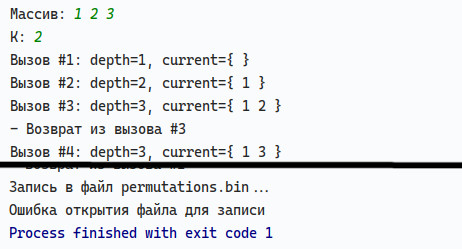


Рисунок 6

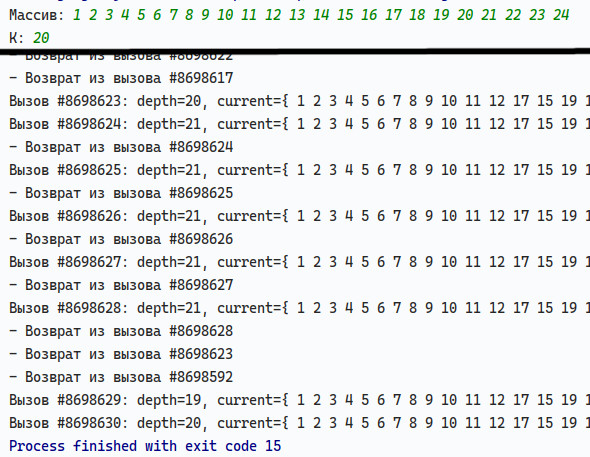


Рисунок 7

1. Вывод по результату тестирования

**Нормальные значения.** Функция корректно вычисляет результат для нормальных значений (рис. 2, 3).

**Отрицательные и нулевые значения.** При значениях K < 1 и K больше размера A работа программы прекращается, так как составить размещения при таком значении невозможно (рис. 4, 5).

**Ошибка открытия файла.** При ошибке открытия файла работа программы прекращается, так как невозможно записать результат (рис. 6).

**Большие значения.** За счет того, что глубина рекурсии никогда не превышает K+1, большие значения и объемные вычисления не прерывают работу программы, однако вычисление результата занимает слишком много времени (рис. 7). Для того, чтобы программа оборвалась переполнением стека, необходимо подать на вход большое значение K, а следовательно и очень большой массив, чтобы глубина рекурсии достигла слишком большого значения.

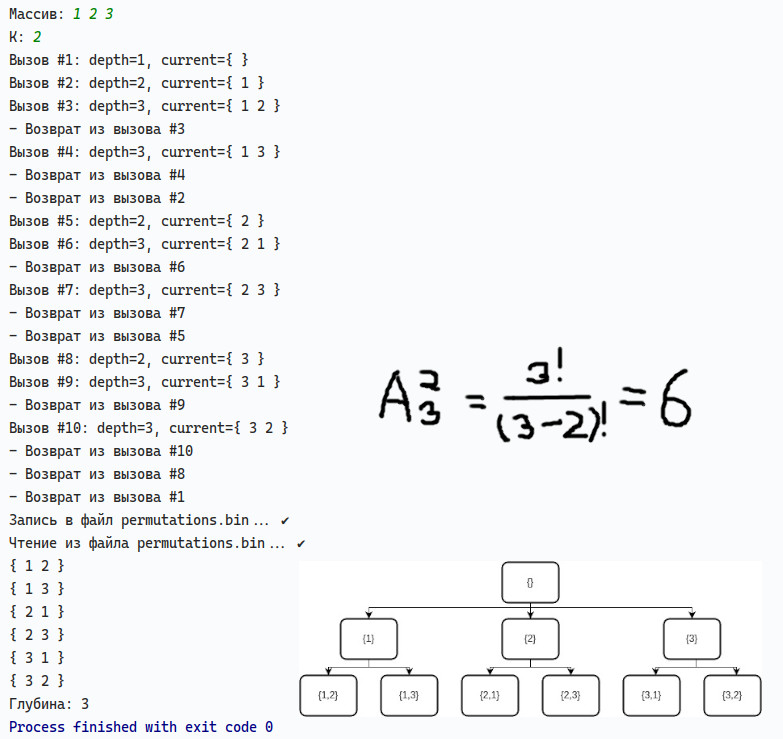


Рисунок 8

Из рис. 8 видно, что функция алгоритмически корректно решает задачу.