Министерство образования и науки Российской Федерации
Пензенский государственный университет
Факультет вычислительной техники

Пояснительная записка

К курсовому проектированию по курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Реализация алгоритма генерирования всех перестановок заданного множества в антилексикографическом порядке»

Выполнил:
Студент группы 24ВВВ3
Масюк А.Р.

Принял:
Юрова О.В.

Пенза 2025

Содержание

Реферат

Отчет содержит 27 страниц, 7 рисунков, 2 таблицы, 6 источников, 1 приложение.

ПЕРЕСТАНОВКА ЭЛЕМЕНТОВ МНОЖЕСТВА, АНТИЛЕКСИКОГРАФИЧЕСКИЙ ПОРЯДОК, МНОЖЕСТВА

Цель исследования – разработка программы, способной найти все множества перестановок в антилексиграфическом порядке.

В работе рассмотрены 3вида перестановок в виде только буквы, только цифры, и буквы, и символы.

Введение

Перестановки имеют фундаментальное значение в различных сферах математики, компьютерных и прикладных наук. Они находят применение в комбинаторике, криптографии, задачах оптимизации, а также во многих алгоритмах обработки информации. Генерация всех перестановок заданного множества формулируется как проблема, обладающая значительной теоретической и практической ценностью. Особый интерес вызывает создание алгоритмов, способных формировать перестановки в специфическом порядке, таком как антилексикографический.

Антилексикографический порядок является инверсией лексикографического. В этом порядке перестановки упорядочены от наибольшей к наименьшей с точки зрения их лексикографического значения. Данный подход важен для задач, в которых требуется анализировать последовательности с конца упорядоченного множества или выполнять специализированные вычисления.

Целью данной работы выступает исследование и практическая реализация алгоритма генерации всех перестановок заданного множества в антилексикографическом порядке. В рамках проекта будут изучены теоретические основы вопроса, разработан и реализован соответствующий алгоритм, а также проведена оценка его эффективности.

Актуальность темы объясняется широким применением алгоритмов генерации перестановок в сферах анализа данных, исследования методов полного перебора и решения комбинаторных проблем. Создание эффективного алгоритма может быть полезно для разработки программного обеспечения, направленного на решение задач оптимизации и аналитики.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – C. Целью данной курсовой работы является разработка программы на языке C, который является широко используемым.

1. Постановка задачи

Требуется разработать программу, которая осуществит перестановку всех множеств.

a) Анализ существующих методов генерации перестановок.

b) Разработка алгоритма для генерации перестановок в антилексикографическом порядке.

c) Реализация алгоритма на выбранном языке программирования(Си).

d) Тестирование и оценка эффективности алгоритма на различных наборах данных.

Программа должна работать так, чтобы пользователь вводил символы, для которых необходима перестановка. После обработки этих данных на экран должна выводиться перестановка всех множеств. Необходимо предусмотреть различные исходы алгоритмов, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

2. Теоретическая часть задания

В данной работе рассматривается задача генерации всех перестановок заданного множества. Для начала напомним основные определения, используемые в комбинаторике, без углубления в теорию множеств и их свойств.

Перестановкой называется упорядоченное расположение всех элементов множества. Если множество состоит из элементов, общее количество перестановок равно.

Одним из основных способов генерации перестановок является метод последовательного обмена элементов. Принцип работы данного подхода можно описать следующим образом:

- Выбирается текущий элемент множества, начиная с первой позиции.
- Происходит последовательный обмен текущего элемента с каждым из оставшихся, после чего вызывается рекурсивная функция для оставшейся части множества.
- После завершения вызова рекурсивной функции элементы возвращаются на исходные позиции для сохранения корректности последующих итераций.

Для реализации данного метода был разработан алгоритм на языке программирования C, который позволяет генерировать все перестановки заданного множества и сохранять их в файл. Ключевые этапы работы алгоритма:

a) **Инициализация и ввод данных:** Пользователь может задать множество вручную или выбрать его автоматическую генерацию. В последнем случае элементы множества формируются случайным образом, исходя из заданных пользователем параметров (буквы, цифры, комбинированный тип).

b) **Рекурсивная генерация перестановок:** Основной частью алгоритма является рекурсивная функция, которая поочерёдно фиксирует элемент множества на текущей позиции и вызывает саму себя для оставшихся элементов.

c) **Запись результатов:** Каждая сгенерированная перестановка выводится на экран и записывается в текстовый файл для дальнейшего анализа.

d) **Оценка производительности:** Алгоритм замеряет время выполнения с использованием системного таймера, что позволяет оценить его эффективность на различных входных данных.

Реализация алгоритма была выполнена на языке программирования C. Код включает в себя следующие ключевые компоненты:

– Функция swap, осуществляющая обмен двух элементов массива.

– Рекурсивная функция generatePermutations, генерирующая перестановки путём последовательного обмена элементов и записи результатов.

– Главная функция main, которая обеспечивает взаимодействие с пользователем, управление входными данными и обработку выходных результатов.

Пример работы программы:

a) Пользователь вводит строку "ABC". Программа генерирует перестановки: "ABC", "ACB", "BAC", "BCA", "CAB", "CBA".

b) Все перестановки сохраняются в файл results.txt.

c) Выводится время выполнения операции для оценки производительности.

Данный подход демонстрирует один из базовых способов решения задачи генерации перестановок. Его дальнейшая оптимизация или использование альтернативных методов, таких как алгоритм Лемера или подходы на основе битовых операций, могут существенно повысить производительность при работе с большими наборами данных.

3. Описание алгоритма программы

Алгоритм генерации перестановок включает следующие ключевые этапы:

a) **Обмен элементов:** На каждом шаге текущий элемент обменивается с каждым из оставшихся. Это позволяет сформировать множество всех возможных комбинаций для текущего набора элементов. Данный процесс реализуется итеративно или рекурсивно, с обеспечением обратного обмена для сохранения исходного состояния массива.

b) **Рекурсивный вызов:** Алгоритм использует рекурсию для перебора всех возможных вариантов расположения элементов. После фиксирования текущего элемента вызывается функция для подмассива, исключающего уже зафиксированные элементы. Рекурсия продолжается до тех пор, пока не будет достигнут базовый случай.

b.1. О**собенности генерации в антилексикографическом порядке:**

В данной программе антилексикографический порядок достигается **непосредственно в процессе генерации перестановок**, без применения дополнительной сортировки результатов.

Перед началом рекурсивного перебора исходное множество символов сортируется по убыванию. Далее рекурсивный алгоритм перебирает элементы массива в обратном порядке — от последнего к первому. Такой подход обеспечивает формирование перестановок от лексикографически наибольшей к наименьшей.

Благодаря данной модификации алгоритм сразу формирует корректную последовательность перестановок, что повышает эффективность и упрощает логику обработки результатов.

c) **Базовый случай:** Когда остаётся только один элемент для размещения, формируется полная перестановка. На этом этапе алгоритм записывает её в выходной файл и/или выводит на экран. Базовый случай обеспечивает завершение рекурсии.

d) **Обратный ход рекурсии:** По завершении обработки текущего уровня рекурсии элементы возвращаются на свои исходные позиции. Этот процесс называется откатом *(backtracking)* и позволяет алгоритму продолжить перебор на предыдущих уровнях.

e) **Сохранение результатов:** Для хранения всех сгенерированных перестановок используется текстовый файл. Это даёт возможность анализа результата после выполнения программы, а также исключает ограничения на объём оперативной памяти.

f) **Измерение времени выполнения:** Алгоритм замеряет время выполнения от начала генерации до завершения последней перестановки. Это позволяет оценить эффективность метода и сопоставить результаты при различных параметрах входных данных (размер множества, тип элементов).

g) **Гибкость ввода данных:** Алгоритм поддерживает как ручной ввод элементов множества, так и автоматическую генерацию. Автоматический режим позволяет задавать тип элементов (буквы, цифры, или их комбинации), а также их количество, что упрощает использование программы для тестирования и анализа.

Преимущества и ограничения

Описанный алгоритм обладает рядом преимуществ, включая простоту реализации, универсальность, и возможность гибкой настройки параметров. Однако он имеет и ограничения, связанные с экспоненциальным ростом количества перестановок при увеличении размера множества. Это делает алгоритм непрактичным для работы с большими наборами данных, что следует учитывать при его использовании.

Таким образом, описанный подход к генерации перестановок представляет собой эффективное решение для задач малого и среднего масштаба, требующих полного перебора всех возможных вариантов.

4. Описание программы

Для написания данной программы использован язык программирования C - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C). Программа реализует алгоритм генерации всех перестановок заданного множества элементов в антилексикографическом порядке. Она предоставляет пользователю возможность гибкого ввода данных и сохраняет результаты работы в текстовый файл. Программа ориентирована на решение задач, связанных с полным перебором вариантов, и может применяться в комбинаторных расчётах, оптимизационных задачах и анализе данных.

Основные возможности программы:

а) **Выбор режима ввода данных** (смотреть Рисунок 1):

– **Ручной ввод**: пользователь самостоятельно задаёт элементы множества.

– **Автоматическая генерация**: программа формирует множество элементов случайным образом на основе выбранных параметров (тип данных: буквы, цифры или их комбинация; количество элементов).

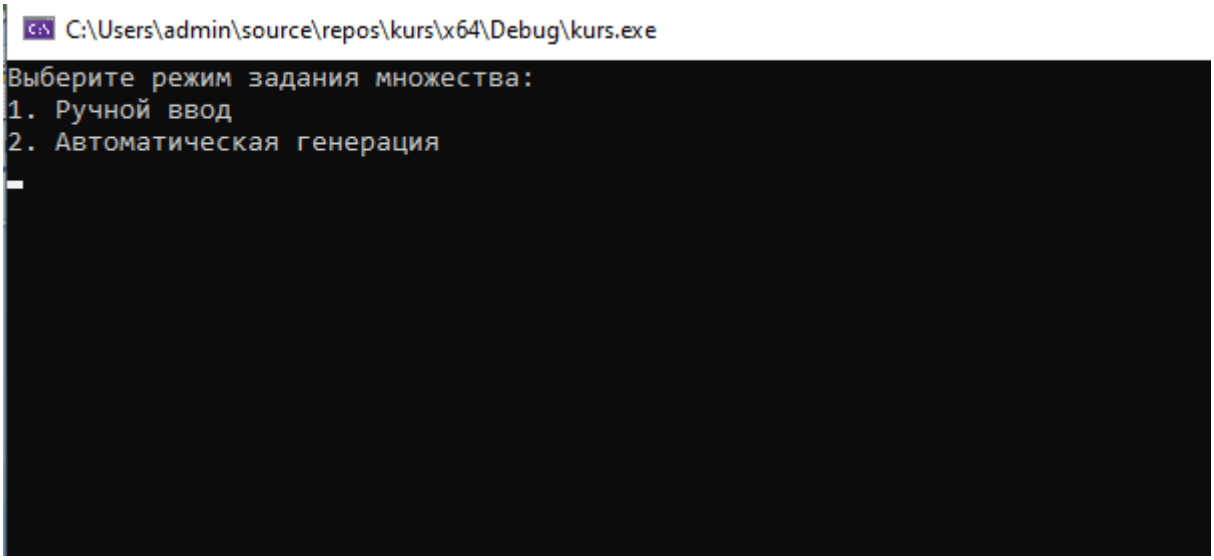


Рисунок 1

b) **Генерация перестановок**:

– Реализуется с использованием рекурсивного алгоритма, который фиксирует элементы на текущей позиции, создавая перестановки для оставшихся элементов.

– На каждом шаге алгоритм выполняет обмен элементов и откат изменений для обеспечения корректности дальнейших вычислений.

– Перед запуском рекурсивного алгоритма исходный массив элементов сортируется по убыванию. В процессе рекурсивного перебора элементы фиксируются в обратном порядке, что обеспечивает генерацию перестановок в антилексикографической последовательности (смотреть Рисунок 2).

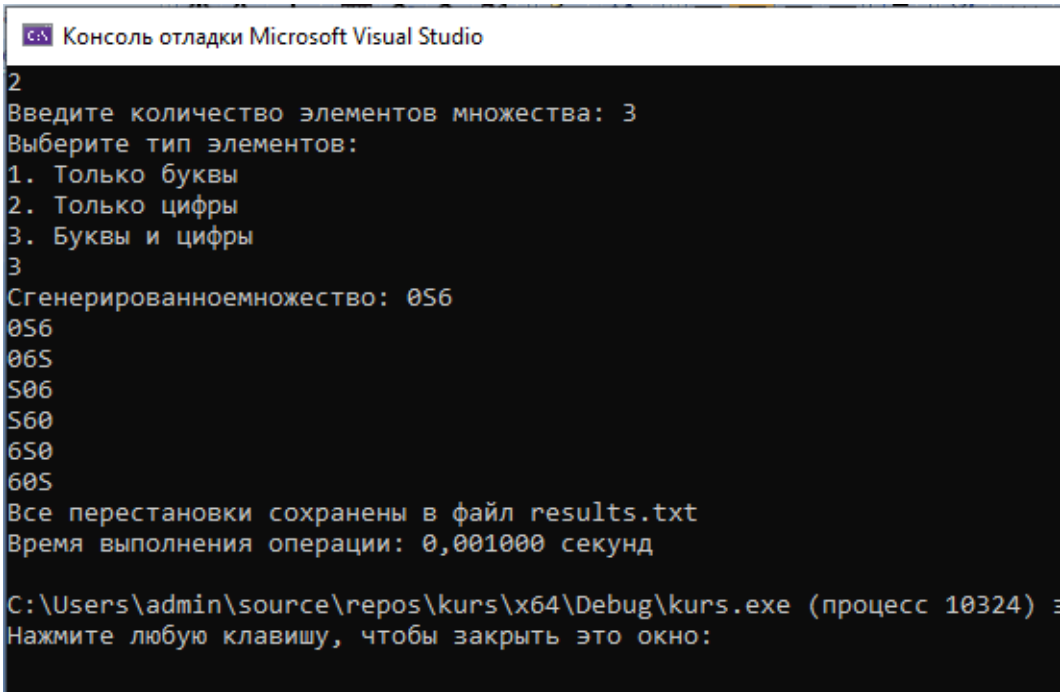


Рисунок 2

c) **Вывод результатов**:

– Каждая сгенерированная перестановка отображается на экране (Рисунок 3).

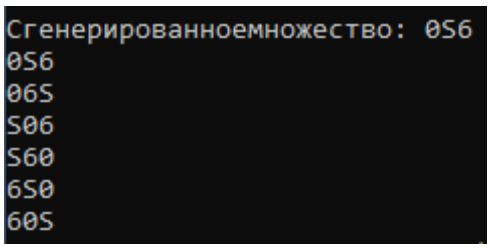


Рисунок 3

– Результаты записываются в текстовый файл results.txt для последующего анализа (принцип работы показан ниже в Рисунке 4).

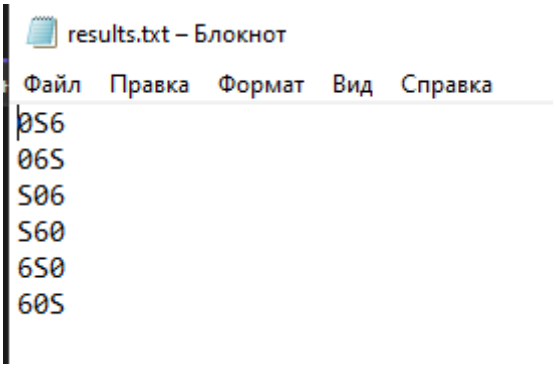


Рисунок 4

d) **Замер времени выполнения**:

– Программа измеряет время выполнения алгоритма, что позволяет пользователю оценить производительность для заданных параметров множества (это можно видеть на Рисунке 5).

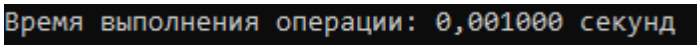
– 

Рисунок 5

Ключевые этапы работы программы:

1. **Инициализация**: Программа предлагает пользователю выбрать режим работы и вводит данные множества в зависимости от его выбора.

2. **Реализация алгоритма**: Алгоритм начинает генерацию перестановок с первого элемента и рекурсивно продолжает, пока не будут рассмотрены все возможные комбинации.

3. **Сохранение результатов**: Каждая перестановка записывается в файл, что позволяет избежать ограничения оперативной памяти при большом объёме данных.

4. **Оценка эффективности**: После завершения работы алгоритма программа выводит на экран общее время выполнения, что может быть полезно при оптимизации или сравнении производительности различных реализаций.

Преимущества программы:

Удобство использования: простота ввода данных и возможность автоматической генерации упрощают работу пользователя.

Универсальность: программа подходит для решения широкого спектра задач, связанных с генерацией перестановок.

Анализ производительности: встроенная функция замера времени выполнения помогает оценить эффективность алгоритма.

Экспоненциальный рост времени выполнения: при увеличении размера множества количество перестановок растёт факториально, что может привести к длительному выполнению программы.

Ограничение объёма данных: при генерации множества с большим количеством элементов размер выходного файла может стать значительным.

Меню содержатся, как отдельные функция. Они нужны для фиксации выбора пользователя, и дальнейшего выбора решения.

```
int main() {
setlocale(LC_ALL, "Rus");
srand((unsigned)time(NULL));

showTitleScreen();

system("cls");

printf("Выберите режим задания множества:\n");
printf("1. Ручной ввод\n");
printf("2. Автоматическая генерация\n");

int mode;
scanf("%d", &mode);

if (mode < 1 || mode > 2) {
printf("Неверный выбор. Завершение программы.\n");
return 1;
 }
```

```
char array[100];
int length;

if (mode == 1) {
rintf("Введите элементы множества (без пробелов): ");
scanf("%s", array);
length = strlen(array);
}
else {
printf("Введите количество элементов множества: ");
scanf("%d", &length);

if (length <= 0 || length > 99) {
printf("Некорректное количество элементов.\n");
return 1;
      }

printf("Выберите тип элементов:\n");
printf("1. Только буквы\n");
printf("2. Только цифры\n");
printf("3. Буквы и цифры\n");

int type;
scanf("%d", &type);

if (type < 1 || type > 3) {
printf("Неверный выбор.\n");
return 1;
}

for (int i = 0; i < length; i++) {
if (type == 1)
array[i] = 'A' + rand() % 26;
else if (type == 2)
array[i] = '0' + rand() % 10;
else
array[i] = (rand() % 2 == 0) ? ('A' + rand() % 26) : ('0' +        rand() %                    10);
}
array[length] = '\0';

printf("Сгенерированное множество: %s\n", array);
}

sortDescending(array, length);

FILE* file = fopen("results.txt", "w");
if (!file) {
printf("Ошибка открытия файла results.txt\n");
return 1;
}

clock_t start_time = clock();
generatePermutationsAntiLex(array, 0, length - 1, file);
clock_t end_time = clock();

fclose(file);
```

Выбор типа генерации:

Только буквы: генерируется символ из диапазона заглавных латинских букв ('A' - 'Z').

Только цифры: генерируется случайная цифра ('0' - '9').

Буквы и цифры: случайным образом выбирается либо буква, либо цифра.

Завершение строки:

После заполнения массива добавляется нуль-символ (\0) для обозначения конца строки, что необходимо для корректной работы с данными как с C-строкой.

Назначение

Этот код предназначен для создания тестового множества символов, которое используется в задаче генерации перестановок. Гибкость параметров позволяет пользователю выбирать тип символов, что делает программу универсальной для различных случаев применения.

```
for (int i = 0; i < length; i++) {
if (type == 1) {
array[i] = 'A' + rand() % 26;
            }
elseif (type == 2) {
array[i] = '0' + rand() % 10;
            }
else {
if (rand() % 2 == 0) {
array[i] = 'A' + rand() % 26;
                }
else {
array[i] = '0' + rand() % 10;
}
            }
        }
array[length] = '\0';
```

Этот код представляет собой рекурсивную функцию для генерации всех перестановок символов в строке и их сохранения в файл. Он состоит из двух основных функций:

```
void swap(char* a, char* b) {
char temp = *a;
   *a = *b;
   *b = temp;
}

void generatePermutations(char* array, intstart, intend, FILE* file) {
if (start == end) {
printf("%s\n", array); // Вывод на экран
fprintf(file, "%s\n", array); // Сохранение в файл
return;
   }

for (int i = start; i <= end; i++) {
swap(&array[start], &array[i]);
generatePermutations(array, start + 1, end, file);
swap(&array[start], &array[i]);
}
}
```

a) **swap** – Функция для обмена значениями двух символов. Она принимает два указателя на символы (char* a и char* b), и меняет их местами. Для этого создается временная переменная temp, в которой хранится значение одного из символов, чтобы не потерять его.

b) **generatePermutations** – Рекурсивная функция для генерации всех возможных перестановок символов строки. Она принимает следующие параметры:

– array – массив символов (строка), для которой нужно найти все перестановки.

– start и end – индексы начала и конца текущего сегмента строки, который обрабатывается.

– file – указатель на файл, в который записываются все найденные перестановки.

Функция работает следующим образом:

– Если start == end, это означает, что строка переставлена полностью, и ее нужно вывести на экран с помощью printf и записать в файл с помощью fprintf.

– В цикле для каждого индекса от start до end производится обмен текущего символа с символом на позиции start. После этого рекурсивно вызывается generatePermutations для следующей части строки (от позиции start + 1 до end).

– После рекурсивного вызова происходит еще один обмен, чтобы вернуть строку в исходное состояние, обеспечивая возможность дальнейших перестановок.

Таким образом, код генерирует все возможные перестановки строки и сохраняет их как в файл, так и выводит на экран.

5. Тестирование

Среда разработки *Microsoft Visual Studio 2022* предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

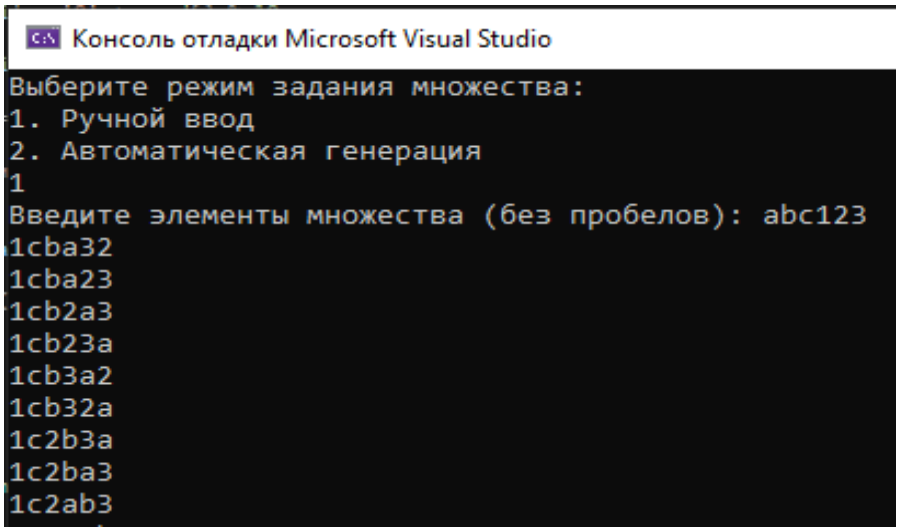
Таблица 1 – Описание поведения программы при тестировании

№	Описание теста	Предусловие	Тестирование	Ожидаемый результат
1	Запуск программы	Нет	Запуск программы с помощью Miscrosoft Visual Studio 2022	Успешный запуск метода main, отображение стартового экрана и главного меню
2	Создание множества	Программа запущена	Выбор режима ввода (ручной или автоматический), ввод элементов множества	Инициализация массива символов заданного размера
3	Просмотр сгенерированных перестановок	Множество успешно создано	Запуск генерации перестановок	Корректный вывод всех перестановок в антилексикографическом порядке

Продолжение таблицы 1

4	Анализ корректности алгоритма	Перестановки сгенерированы	Проверка количества и уникальности перестановок	Количество перестановок соответствует n!, все перестановки уникальны
5	Сохранение результатов	Перестановки успешно сгенерированы	Подтверждение сохранения результатов	Успешное создание файла results.txt с полным списком перестановок

Ниже продемонстрирован результат(Рисунок 6) тестирования программы при ручном вводе пользователем различных количеств вершин.



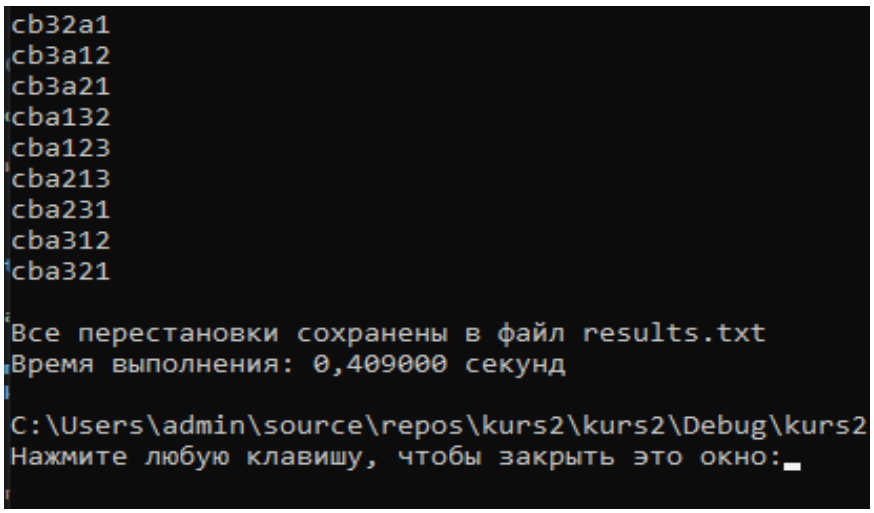


Рисунок 6 - Результат работы с ручным вводом

И также при автоматической генерации (Рисунок 7).

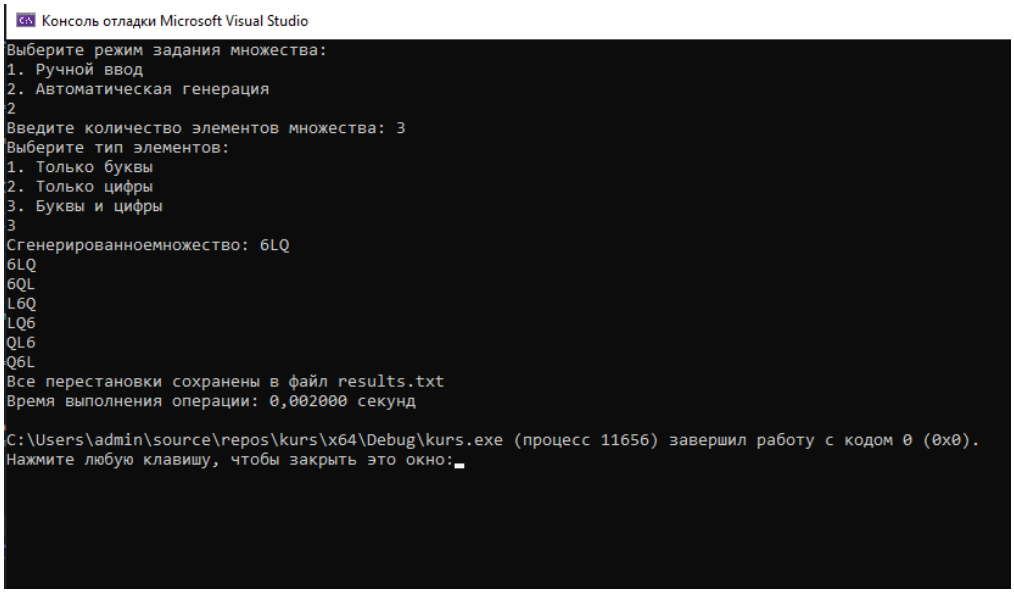


Рисунок 7 - Результат работы с автоматической генерацией и букв и чисел

Таблица 2 – Итоги тестирования

№	Описание теста	Полученный результат
1	Запуск программы	Верно
2	Создание множества	Верно
3	Просмотр сгенерированных перестановок	Верно
4	Анализ корректности алгоритма	Верно
5	Сохранение результатов	Верно

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

6. Ручной расчет задачи

Для проверки корректности работы программы выполнен ручной расчёт для множества, состоящего из трёх элементов.

Для проверки корректности работы программы выполним ручной расчёт для небольшого множества, состоящего из трёх элементов:

{A,B,C}

Количество возможных перестановок определяется по формуле:

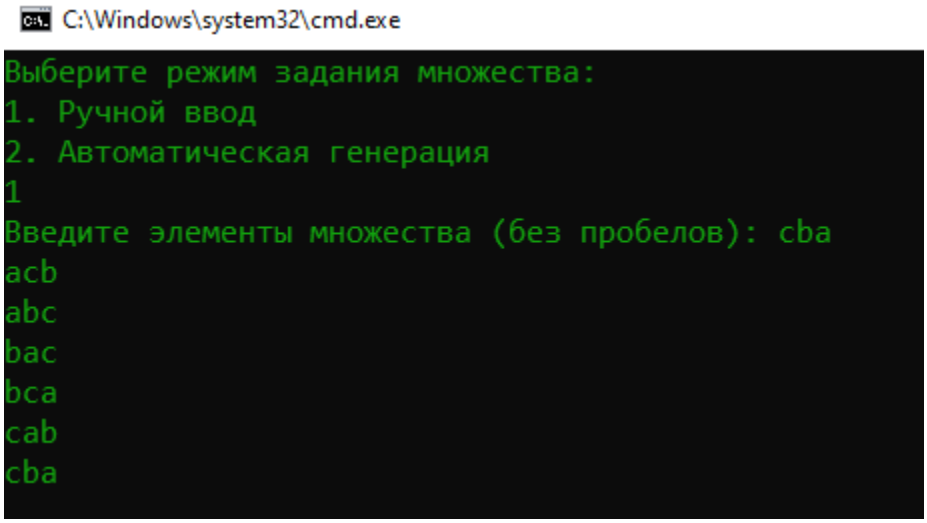
n!=3!=6

Сначала расположим элементы в антилексикографическом порядке, то есть от наибольшего к наименьшему. Для этого исходное множество упорядочивается по убыванию:

C,B,A

Далее вручную получим все перестановки данного множества в антилексикографическом порядке:

1. ACB
2. ABC
3. BAC
4. BCA
5. CAB
6. CBA



Таким образом, вручную получено 6 перестановок, что соответствует теоретическому значению 3!.

Работа программы для данного входного множества даёт аналогичный результат: выводятся те же 6 перестановок в том же антилексикографическом порядке. Это подтверждает корректность реализованного алгоритма генерации перестановок и правильность его работы.

Заключение

Таким образом, в ходе курсового проекта была разработана и реализована на языке С программа для генерации всех перестановок заданного множества в среде разработки Microsoft Visual Studio 2022.

В процессе выполнения данной курсовой работы были сформированы умения создания программного обеспечения и освоены методы работы с перестановками. Получены практические навыки реализации алгоритма генерации перестановок множеств в антилексикографическом порядке. Углублено понимание языка программирования С.

К недостаткам разработанного приложения можно отнести простой интерфейс взаимодействия с пользователем. Это объясняется тем, что программа функционирует в консольном режиме, что исключает сложности, связанные с созданием оконного графического интерфейса.

Программа имеет небольшой, но достаточный для использованияфункционал возможностей.

Программа попадает в категорию NP-полных, поэтому результат может отличаться от ручных расчётов.

Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с
2. www.wikipedia.com
3. З. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с
4. Хар Прата С. Язык программирования С. Лекции и упражнения. — М.: Вильямс, 2022.
5. Герб Керниган Б.У., Ритчи Д.М. Язык программирования С.
6. Робинсон Д., Вильямс У. Дискретная математика для программистов. — М.: ДМК Пресс, 2021.

Приложение А

Листинг программы

Файл Kurs.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <locale.h>


void swap(char* a, char* b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}

void sortDescending(char* array, int length) {
    for (int i = 0; i < length - 1; i++) {
        for (int j = i + 1; j < length; j++) {
            if (array[i] < array[j]) {
                swap(&array[i], &array[j]);
            }
        }
    }
}


void generatePermutationsAntiLex(char* array, int start, int end, FILE* file) {
    if (start == end) {
        printf("%s\n", array);
        fprintf(file, "%s\n", array);
        return;
    }

    for (int i = end; i >= start; i--) {
        swap(&array[start], &array[i]);
        generatePermutationsAntiLex(array, start + 1, end, file);
        swap(&array[start], &array[i]);
    }
}


void showTitleScreen() {
    system("cls");

    printf("============================================================\n");
    printf("Министерство образования и науки Российской Федерации\n");
    printf("Пензенский государственный университет\n");
    printf("Факультет вычислительной техники\n\n");
    printf("Курсовая работа по дисциплине:\n");
    printf("«Логика и основы алгоритмизации в инженерных задачах»\n");
    printf("Тема:\n");
    printf("Реализация алгоритма генерации всех перестановок\n");
    printf("заданного множества в антилексикографическом порядке\n\n");
    printf("Выполнил: студент группы 24ВВВ3\n");
    printf("Масюк А.Р.\n");
    printf("============================================================\n\n");
    printf("Нажмите Enter для продолжения...");

    getchar();
}


int main() {
```

```
setlocale(LC_ALL, "Rus");
srand((unsigned)time(NULL));

showTitleScreen();

system("cls");

printf("Выберите режим задания множества:\n");
printf("1. Ручной ввод\n");
printf("2. Автоматическая генерация\n");

int mode;
scanf("%d", &mode);

if (mode < 1 || mode > 2) {
    printf("Неверный выбор. Завершение программы.\n");
    return 1;
}

char array[100];
int length;

if (mode == 1) {
    printf("Введите элементы множества (без пробелов): ");
    scanf("%s", array);
    length = strlen(array);
}
else {
    printf("Введите количество элементов множества: ");
    scanf("%d", &length);

    if (length <= 0 || length > 99) {
        printf("Некорректное количество элементов.\n");
        return 1;
    }

    printf("Выберите тип элементов:\n");
    printf("1. Только буквы\n");
    printf("2. Только цифры\n");
    printf("3. Буквы и цифры\n");

    int type;
    scanf("%d", &type);

    if (type < 1 || type > 3) {
        printf("Неверный выбор.\n");
        return 1;
    }

    for (int i = 0; i < length; i++) {
        if (type == 1)
            array[i] = 'A' + rand() % 26;
        else if (type == 2)
            array[i] = '0' + rand() % 10;
        else
            array[i] = (rand() % 2 == 0) ? ('A' + rand() % 26) : ('0' + rand() % 10);
    }
    array[length] = '\0';

    printf("Сгенерированное множество: %s\n", array);
}

sortDescending(array, length);

FILE* file = fopen("results.txt", "w");
if (!file) {
    printf("Ошибка открытия файла results.txt\n");
    return 1;
}
```

```
    clock_t start_time = clock();
    generatePermutationsAntiLex(array, 0, length - 1, file);
    clock_t end_time = clock();

    fclose(file);

    printf("\nВсе перестановки сохранены в файл results.txt\n");
    printf("Время выполнения: %.6f секунд\n",
        (double)(end_time - start_time) / CLOCKS_PER_SEC);

    return 0;
}
```