Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт прикладных информационных

технологий и коммуникаций

Кафедра Информационная безопасность

автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

**Расчётно-графическая работа**

**по дисциплине «Языки программирования»**

Тема: «Лабиринт. Поиск выхода»

Выполнил:

Студент 1 курса ИнПит

группы с-Ибс11

Евсеев А.С.

Проверил: ассистент каф. ИБС

Романчук С. П.

Саратов 2020

**Аннотация**

В работе реализован лабиринт – структура, состоящая из запутанных путей, ведущих к выходу (ну или же в тупик). Программа строит на экране изображение лабиринта из файла, после чего определяет наиболее оптимальный маршрут от начального положения до выхода.

Данная работа сводится к нахождению кратчайшего пути между точками в пространстве.

**Содержание**

* Введение
* Теоретическая часть
* Практическая часть
* Заключение
* Приключение
* Литература

**Введение**

**Лабиринт** — какая-либо структура (обычно в двухмерном или трехмерном пространстве), состоящая из запутанных путей к выходу (и/или путей, ведущих в тупик). Под лабиринтом у древних [греков](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B5%D1%86%D0%B8%D1%8F) и [римлян](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%BC) подразумевалось более или менее обширное пространство, состоящее из многочисленных залов, камер, дворов и переходов, расположенных по сложному и запутанному плану, с целью запутать и не дать выхода несведущему в плане лабиринта человеку.

Именно подобию лабиринта я попытался воссоздать в своей расчетно-графической работе.

**Тема:**

Лабиринт. Поиск выхода

**Цель:**

Разработать программу для нахождения кратчайшего пути между точками в пространстве, используя бесплатную кроссплатформенную среду разработки Code::Blocks на одном из самых популярных языков программирования C++.

**Задачи:**

1. Создать лабиринт
2. Изучить алгоритм Дейкстры
3. Реализовать алгоритм Дейкстры
4. Оформить созданную программу (дизайн)
5. Составить пояснительную записку

**По итогу программа должна содержать:**

* Рабочий код, удовлетворяющий требованию задания
* Дизайн и удобство работы с программой

**Теоретическая часть**

**Язык программирования:**

**C++** — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения.

Поддерживает такие [парадигмы программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как [процедурное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [обобщённое программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).[[3]](https://ru.wikipedia.org/wiki/C%2B%2B#cite_note-_cd050e120e51c572-3)[[4]](https://ru.wikipedia.org/wiki/C%2B%2B#cite_note-_59ba0da17491157e-4) В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).[[4]](https://ru.wikipedia.org/wiki/C%2B%2B#cite_note-_59ba0da17491157e-4)

**C++** широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования[[мнения 1]](https://ru.wikipedia.org/wiki/C%2B%2B#cite_note-langpop-5)[[мнения 2]](https://ru.wikipedia.org/wiki/C%2B%2B#cite_note-tiobe-6). Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе [x86](https://ru.wikipedia.org/wiki/X86) это [GCC](https://ru.wikipedia.org/wiki/GNU_Compiler_Collection), [Visual C++](https://ru.wikipedia.org/wiki/Visual_C%2B%2B), [Intel C++ Compiler](https://ru.wikipedia.org/wiki/Intel_C%2B%2B_Compiler), [Embarcadero (Borland) C++ Builder](https://ru.wikipedia.org/wiki/Embarcadero_C%2B%2B_Builder) и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на [Java](https://ru.wikipedia.org/wiki/Java),[C#](https://ru.wikipedia.org/wiki/C_Sharp), [Golang](https://ru.wikipedia.org/wiki/Golang), [Solidity](https://ru.wikipedia.org/wiki/Solidity), [PHP](https://ru.wikipedia.org/wiki/PHP).

Синтаксис **C++** унаследован от языка [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как [компиляторами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

**Среда разработчика:**

**Code::Blocks** — [свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [кроссплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [среда разработки](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F). **Code::Blocks** написана на [С++](https://ru.wikipedia.org/wiki/%D0%A1%2B%2B) и использует библиотеку [wxWidgets](https://ru.wikipedia.org/wiki/WxWidgets). Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования [С](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [С++](https://ru.wikipedia.org/wiki/%D0%A1%2B%2B), [D](https://ru.wikipedia.org/wiki/D_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) (с ограничениями), [Fortran](https://ru.wikipedia.org/wiki/Fortran).

**Code::Blocks** разрабатывается для [Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows), [Linux](https://ru.wikipedia.org/wiki/Linux) и [Mac OS X](https://ru.wikipedia.org/wiki/Mac_OS_X). Среду можно собрать из исходников практически под любую [Unix](https://ru.wikipedia.org/wiki/Unix)-подобную систему, например [FreeBSD](https://ru.wikipedia.org/wiki/FreeBSD), [PC-BSD](https://ru.wikipedia.org/wiki/PC-BSD).

**GNU Compiler Collection** (обычно используется сокращение **GCC**) — набор [компиляторов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) для различных [языков программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), разработанный в рамках [проекта GNU](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_GNU). GCC является [свободным программным обеспечением](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), распространяется [фондом свободного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BD%D0%B4_%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (FSF) на условиях [GNU GPL](https://ru.wikipedia.org/wiki/GNU_GPL) и [GNU LGPL](https://ru.wikipedia.org/wiki/GNU_Lesser_General_Public_License) и является ключевым компонентом [GNU toolchain](https://ru.wikipedia.org/wiki/GNU_toolchain). Он используется как стандартный компилятор для свободных [UNIX-подобных операционных систем](https://ru.wikipedia.org/wiki/Unix-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0).

Изначально названный **GNU C Compiler** поддерживал только язык [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Позднее GCC был расширен для компиляции исходных кодов на таких языках программирования, как [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Objective-C](https://ru.wikipedia.org/wiki/Objective-C), [Java](https://ru.wikipedia.org/wiki/Java), [Фортран](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD), [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Go](https://ru.wikipedia.org/wiki/Go), [GAS](https://ru.wikipedia.org/wiki/GNU_Assembler) и [D](https://ru.wikipedia.org/wiki/D_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

С версии 4.2.2 GCC перешёл на лицензию [GPLv3](https://ru.wikipedia.org/wiki/GNU_General_Public_License#GPL_v3).

**Алгоритм:**

**Алгори́тм Де́йкстры** — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) на [графах](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), изобретённый нидерландским учёным [Эдсгером Дейкстрой](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0,_%D0%AD%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%92%D0%B8%D0%B1%D0%B5) в [1959 году](https://ru.wikipedia.org/wiki/1959_%D0%B3%D0%BE%D0%B4). Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без [рёбер](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B1%D1%80%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)) отрицательного [веса](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C_%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%BE%D0%B2_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#%D0%92). Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации [OSPF](https://ru.wikipedia.org/wiki/OSPF) и [IS-IS](https://ru.wikipedia.org/wiki/IS-IS).

**Практическая часть**

В коде программы комментариями описана основная структура и принцип работы программы.

Основной алгоритм, который был использован для нахождения кратчайшего пути - алгоритм Дейкстры.

**Описание/принцип работы программы:**

В самом начале программы мы вводим название файла, содержащего так называемую «карту» лабиринта, откуда построчно считываем структуру лабиринта, которую записываем в массив строк labyrinth, по ходу считывания проверяя, что у нас в файле нет лишних символов, неизвестных нам. Массив строк, а не массив массивов чисел был использован для того, чтобы можно было более просто считывать и выводить структуру лабиринта.

После считывания и записи лабиринта в массив, мы проходимся по клеткам лабиринта и ищем начальную и конечные точки в лабиринте, их записываем. После инициализации массива дистанций от начальной точки до какой-либо точки (изначально бесконечность, т.е. невозможно добраться), массива родителей и массива использованных точек для алгоритма дейкстры, мы выставляем расстояние dist[start.x][start.y] в ноль, так как из начальной точки в саму себя можно попасть за 0 единиц расстояния. Для записи расстояния я сделал массив массивов, в котором [i][j] обозначает расстояние от начала до точки i,j. Массив родителей является массивом массивов точек, в котором в [i][j] записан родитель этой точки, то есть из какого места мы в нее пришли. Это обновляется по ходу работы алгоритма и далее используется для восстановления и отрисовки пройденного нами пути. В массиве used (массив использованных точек) будет написано, использовали ли мы эту точку или еще нет, то есть true/false.

После всех инициализаций мы запускаем сам алгоритм дейкстры, который работает по следующему принципу:

Запустим основную часть алгоритма ширина\*высота раз, чтобы была возможность попасть в КАЖДУЮ клетку

в цикле пройдемся по всем точкам и найдем точку, до которой на данный момент самое короткое расстояние и которую мы еще не использовали (на первом шаге всегда будет начальная, т.к. до нее расстояние 0)

пометим выбранную вершину как использованную

попробуем от этой вершины пройти в каждую из 4-х сторон

если в эту сторону пройти можно, т.е. если она на карте и там нет стены, И если расстояние до нее уменьшится если мы сделаем этот ход, то обновим расстояние следующей точки и выставим нашу точку как ее родителя

После алгоритма дейкстры проверяем, что мы смогли дойти до конечной вершины, то есть проверяем наличие пути до нее, если такого не существует - сообщаем об ошибки и завершаем программу.

Далее создается итоговая карта лабиринта, с дополнительными клетками, обозначающими края лабиринта. Изначально инициализируем эти клетки так же, как и во входном лабиринте, помечая пустые клетки пробелами, а занятые – «П». После этого мы, начиная от последней точки, идем в обратную сторону по родителям точек, параллельно заполняя карту лабиринта символами '\*', обозначающие наш путь, после чего просто построчно выводим итоговую карту.

**Используемые библиотеки:**

**#include <iostream> -** заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++. Он включён в стандартную библиотеку C++.

**#include <vector> -** это библиотека, в которой находится шаблон класса для контейнеров последовательности. Вектор хранит элементы заданного типа в линейном упорядочении и обеспечивает быстрый произвольный доступ к любому элементу. Вектор является предпочтительным контейнером для последовательности, когда производительность произвольного доступа имеет уровень "Премиум".

**#include <fstream>** - заголовочный файл из стандартной библиотеки C++, включающий набор классов, методов и функций, которые предоставляют интерфейс для чтения/записи данных из/в файл.

**#include <iomanip> -** реализует инструменты для работы с форматированием вывода, например базу, используемую при форматировании целых и точных значений чисел с плавающей запятой.

**Заключение**

У меня получилось создать программу, для нахождения оптимального выхода из лабиринта, в этом мне помог алгоритм Дейкстры.

**Приложения**

#include <iostream>

#include <vector>

#include <fstream>

#include <iomanip>

using namespace std;

typedef pair<int, int> point; // тип данных точка на карте

const char wallChar = 'П'; // символ обозначающий стену

const char emptyChar = ' '; // символ обозначающий проход

const char moveChar = '\*'; // символ обозначающий путь

// все возможные движения, без диагонали

vector<point> moves = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

int main(int argc, char \*argv[]) {

setlocale (0, "");

string filename;

cout<<"Введите имя файла (по умолчанию это labirint.txt):";

cin>>filename;

vector<string> labyrinth; // здесь будем хранить сам лабиринт

int width, height; // сохраняем размеры лабиринта

ifstream labFile;

labFile.open(filename, ifstream::in);

if (labFile.is\_open()) {

string temp;

labFile >> temp; // считаем первую строку лабиринта

width = temp.size();

cout << "Ширина лабиринта = " << width << endl;

do {

for (char &c: temp) {

switch (c) {

case '0':

case '1':

case '2':

case '3':

break; // все в порядке

default: { // мы встретили неправильный символ

cout << "Лабиринт записан в некорректном формате, встретился непонятный символ."

<< endl;

return 0;

}

}

}

if (temp.size() != width) {

cout << "Строки в лабиринте имеют разную длину, некорректный формат." << endl;

return 0;

}

labyrinth.push\_back(temp); // а в самом теле цикла будем записывать уже считанную строку

} while (labFile >> temp); // будем считывать оставшиеся строки лабиринта

height = labyrinth.size();

cout << "Высота лабиринта = " << height << endl;

} else { // если не существует файла с лабиринтом то выйдем из программы

cout << "Такого файла не существует." << endl;

return 0;

}

// вектор будет хранить информацию о том использована/не использована вершина

vector<vector<bool>> used(height, vector<bool>(width));

// вектор будет хранить минимальные расстояния для алгоритма дейкстры

vector<vector<int>> dist(height, vector<int>(width));

// вектор будет хранить родителей вершин, то есть откуда мы пришли, будем записывать сюда

// по ходу обновления расстояний в алгоритме дейкстры

vector<vector<point>> parent(height, vector<point>(width));

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

parent[i][j] = {-1, -1}; // изначально нет родителя

dist[i][j] = INT\_MAX; // бесконечное расстояние до вершины, то есть добраться до нее нельзя

used[i][j] = 0; // не использована ни одна вершина

}

}

pair<int, int> startVert; // начальная вершина

pair<int, int> endVert; // начальная вершина

// пройдемся по всем вершинам и найдем вершину из которой мы начинаем

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

if (labyrinth[i][j] == '2') { // нашли то откуда мы начинаем

startVert = {i, j};

break;

}

}

}

// пройдемся по всем вершинам и найдем вершину в которой мы заканчиваем

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

if (labyrinth[i][j] == '3') { // нашли то откуда мы начинаем

endVert = {i, j};

break;

}

}

}

cout << "Начальная вершина - " << startVert.first << "," << startVert.second << endl;

cout << "Конечная вершина - " << endVert.first << "," << endVert.second << endl;

dist[startVert.first][startVert.second] = 0; // расстояния до начальной вершини равно нулю

cout << "Запускаем алгоритм дейкстры" << endl;

// алгоритм дейкстры

// проитерируемся столько раз, сколько всего вершин

for (int count = 0; count < width \* height; count++) {

point curItem;

int minD = INT\_MAX;

// найдем вершину с минимальным расстоянием на этом шаге

// на первом шаге это всегда будет начальная вершина

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

if (!used[i][j] and dist[i][j] < minD) {

curItem = {i, j}; // если мы еще не использовали и расстояние для нее меньше найденного ранее, то возьмем ее

minD = dist[i][j];

}

}

}

used[curItem.first][curItem.second] = true; // мы использовали выбранную вершину на этом шаге

for (auto &move: moves) {

point next = {curItem.first + move.first, curItem.second + move.second};

// если мы выходим за границы лабиринта либо если входим в стену, то пропустить этот ход

if (next.first < 0 || next.second < 0 || next.first >= height || next.second >= width ||

labyrinth[next.first][next.second] == '1')

continue;

// если мы сделаем ход от нашей вершины до этой и расстояние уменьшится, то обновим

if (dist[curItem.first][curItem.second] + 1 < dist[next.first][next.second]) {

dist[next.first][next.second] = dist[curItem.first][curItem.second] + 1;

parent[next.first][next.second] = curItem;

}

}

}

// дойти до конечной вершины нельзя

if (dist[endVert.first][endVert.second] == INT\_MAX) {

cout << "Невозможно добраться до конечной вершины, завершение программы." << endl;

return 0;

}

// заполним итоговую карту символами стен и пустоты

vector<string> resultMap(height + 2, string(width + 2, 0));

for (int i = 0; i < height + 2; i++) {

for (int j = 0; j < width + 2; j++) {

if (i > 0 && i <= height && j > 0 && j <= width)

resultMap[i][j] = (labyrinth[i - 1][j - 1] == '1' ? wallChar : emptyChar);

else

resultMap[i][j] = wallChar;

}

}

auto curVert = endVert;

// пройдемся по родителям вершин от последней вершины до самой первой

while (curVert != startVert) {

resultMap[curVert.first + 1][curVert.second + 1] = moveChar;

curVert = parent[curVert.first][curVert.second];

}

resultMap[curVert.first + 1][curVert.second + 1] = moveChar;

// выведем конечную карту

cout << "Карта движения для оптимального пути:" << endl;

for (string &cur: resultMap) {

cout << cur << endl;

}

return 0;

}

**Литература**

<https://ru.wikipedia.org/wiki/C%2B%2B>

<https://ru.wikipedia.org/wiki/Code::Blocks>

<https://prog-cpp.ru/deikstra/>

<https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры>

<https://ru.wikipedia.org/wiki/Стандартная_библиотека_языка_C%2B%2B>

<http://progopedia.ru/implementation/gpp/>