ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПБГУТ)

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ (ИКСС)

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ (ПИ И ВТ)

ДИСЦИПЛИНА: «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

ЛАБОРАТОРНАЯ РАБОТА №6.

ТЕМА: «ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ (ЛАБИРИНТЫ)»

Выполнили: Молошников Ф.А., Мартынюк А.А.

Студенты группы ИКПИ-05 Подпись_____

Принял: Дагаев А.В.

Доцент кафедры ПИиВТ Подпись_____

«_____»_____2022

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Ознакомление с эвристическими алгоритмами поиска пути в лабиринте и методикой оценки их эффективности. Для сравнения были выбраны:

1) Алгоритмы генерации лабиринта:

    1) Уилсона

    2) Уилсона (модифицированный)

    3) Олдоса-Бродера

    4) Бинарного дерева

2) Алгоритмы поиска пути в лабиринте:

    1) Ли

    2) Ли (модификация с 2 волнами)

    3) Дейкстры

    4) A* (AStar)

Первые два алгоритма поиска пути работают на ДРП (дискретном рабочем поле), где, с точки зрения прокладки путей, все ячейки одинаковы. 3 и 4 алгоритмы работают на графах. Мы используем их модификацию для работы на ДРП (частный случай графа), где ячейки могут иметь разные веса. Вес означает путь в эту ячейку из соседней ячейки. В общем случае, вес ячейки соответствует ребру соответствующего ориентированного графа.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание программы

Для исследования алгоритмов генерации лабиринтов и поиска в них пути были разработаны классы maze и mazeWeighted, которые реализуют хранение лабиринтов и доступ к ним. Для сбора информации о времени генерации лабиринтов и о времени поиска в них путей были разработаны классы StatisticsMaze и StatisticsMazeSearch. В работе использовались классы presenthandler и plothandler, скрипт PlotScript.bash, разработанные в рамках предыдущих лабораторных работ и предназначенные для упрощения вывода промежуточных и конечных результатов работы алгоритмов. Работа также содержит вспомогательные функции для ввода и вывода. Помимо двух основных целей, которые собираются из файлов StatisticsMaze_main.cpp и StatisticsMazeSearch_main.cpp и служат для исследования времени генерации лабиринтов и поиска путей в них соответственно, работа содержит отдельные цели для демонстрации алгоритмов, а так же для демонстрации и отладки разработанных классов.

1) Язык: C++

2) Среда: VS Code

3) ОС: Debian 11

4) Оболочка: xfce 4

5) Библиотеки: -

6) Фреймворк: -

2.2. Структура программы

Таким образом, работа состоит из следующих файлов:

1) Файлы с реализацией основных возможностей

- Файлы системы сборки:

    - CMakeLists.txt

- Файлы с классами и функциями над лабиринтами

- maze.h
- MazeGenerationAlgs.h
- MazeSearchAlg.h

- Файлы с реализацией сбора времени выполнения алгоритмов
  - StatisticsMaze.inl
  - StatisticsMazeSearch.inl

- Файлы со вспомогательными функциями
  - Funcs.h
  - presenthandler.h
  - plothandler.h

- bush-скрипты
  - PlotScript.bash

2) Файлы с реализацией примеров использования основных возможностей
  - maze_debug_main.cpp
  - mazeWeighted_debug_main.cpp
  - Wilson_main.cpp
  - AldousBroder_main.cpp
  - BinaryTree_main.cpp
  - Lee_main.cpp
  - Lee2Waves_main.cpp
  - Dijkstra_main.cpp
  - AStar_main.cpp
  - StatisticsMaze_main.cpp
  - StatisticsMazeSearch_main.cpp

2.3. Разработка классов и функций

Остановимся подробнее на некоторых классах и функциях, существенных в данной работе:

2.3.1. maze

Класс, реализующий лабиринт. Позволяет хранить лабиринт и обращаться к нему.

Лабиринт хранится внутри вектора BaseVector в виде символов. Нечетные строки хранят информацию о верхних и нижних стенах ячеек, четные – о боковых стенах ячеек и значения, хранящиеся в ячейках. Используются следующие символы:

1) # - стена между ячейками

2) ? – отсутствие стены между ячейками

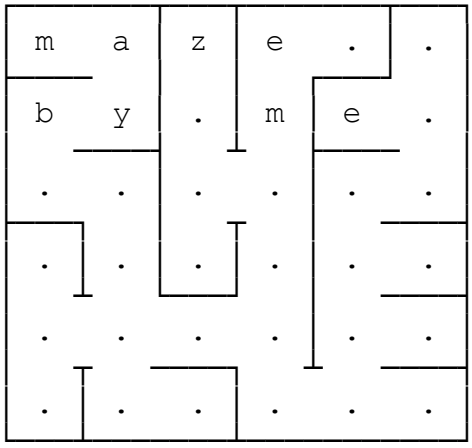
3) + - угол ячейки (Служит для удобства представления. Никак не используется)

4) . – Значение ячейки по умолчанию.

Такой подход к хранению лабиринта нельзя назвать оптимальным по используемой памяти, т.к. треть памяти занята символами «+», которые нигде не используются (например, более оптимальным было бы хранить в массиве только нижнюю и левую стены), однако он обеспечивает довольно быстрое обращение к своим элементам, поскольку использует контейнер vector для хранения элементов.

Также он позволяет в относительно наглядном виде просматривать лабиринт и относительно просто его редактировать (для замены стены на проем или проем на стену достаточно поменять символ # на ? или ? на #).

Например, этот лабиринт

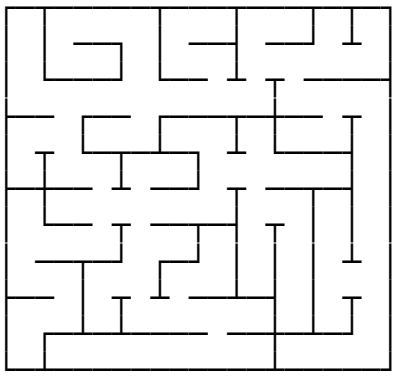


Будет иметь следующее внутреннее представление:

```
+#+#+#+#+#+#+
#m?a#z#e?.#.#
+#+?+?+?+#+?+
#b?y#.#m#e?.#
+?+#+?+?+#+?+
#.?.#.?.#.?.#
+#+?+?+?+?+#+
#.#.#.#.#.?.#
+?+?+#+?+?+#+
#.?.?.?.#.?.#
+?+?+#+?+?+#+
#.#.?.#.?.?.#
+#+#+#+#+#+#+
```

Доступ к ячейкам реализован по индексам i j. При этом нумерация ячеек аналогична нумерации элементов в матрицах: Увеличение индекса строк i идет сверху вниз, увеличение индекса столбцов j слева направо.

Вывод лабиринтов в декоративном виде осуществляется с помощью символов Unicode и доступен в двух вариантах: с мелкими и крупными ячейками. Вариант с крупными ячейками уже был показан выше, вариант с мелкими ячейками показан ниже:



2.3.1.1. Свойства:

`int n=0`	высота лабиринта
`int m=0;`	ширина лабиринта
`std::vector<std::vector<char>> BaseVector;`	Вектор, в котором хранится лабиринт

2.3.1.2. Методы:

maze(int Nn, int Nm)	Конструктор с параметрами. Создает лабиринт высотой Nn и шириной Nm, ячейки которого изолированы друг от друга стенами.
void SetCellValue(int i, int j, char c)	Устанавливает значение в ячейке
void ResetValues()	Сбрасывает значения ячеек (по умолчанию «.»)
char GetCellValue(int i, int j)	Возвращает значение в ячейке
void SetCellWalls(int i, int j, int alpha, bool HasWall, bool Protected=false)	Позволяет установить или убрать стену между ячейками. alpha – полярный угол, под которым радиус вектор из центра ячейки встречает соответствующую стену (измеряется в pi/4). Т.е. если alpha=0, то будет изменено состояние правой стены ячейки, если alpha=1, то верхней стены ячейки, если alpha=2, то левой стены, если alpha=3, то нижней стены и т.д. Флаг Protected позволяет замораживать внешние стены лабиринта, не позволяя изменять их состояние.
bool HasWall(int i, int j, int alpha)	Позволяет определять наличие стены вокруг ячейки. alpha имеет тот же смысл, что и в методе SetCellWalls
int GetNeighborI(int i, int alpha) int GetNeighborJ(int j, int alpha)	Позволяют получить координаты (в I, j) соседней ячейки, радиус вектор до которой находится под углом alpha.
std::string Show(char* filename=(char*)"cin")	Не используется
void CharFromWallFlags(std::stringstream& ss, std::string& WallFlags)	Вспомогательная функция для получения символа по флагу, т.е. по его описанию.
GetCellValueFromStruct 3 перегрузки std::string StrFromInt(T value)	Вспомогательные функции для организации вывода в декоративном виде данных внутри ячеек.
template <typename T=const std::string> void ShowDecorate(char* filename=(char*)"cout", int Mode=0, int Scale=1, bool IsWithValues=false, T& Values=std::string("GetCellValue"), int member=0)	Шаблонный метод для вывода лабиринта в терминал или файл. Первый аргумент – имя файла для вывода (по умолчанию в терминал), второй аргумент – режим вывода 0 – внутреннее представление и декоративный вид. 1 – только декоративный вид. Scale – 1 мелкие ячейки. (высота каждой ячейки 0.5+0.5=1 строка; ширина – 0.5+1+0.5=2 символа), 2 крупные ячейки. В случае крупных ячеек возможен вывод внутри ячеек значений из внешних контейнеров различных типов. member служит для выбора поля внутри элемента контейнера для представления внутри ячейки.

	Для того чтобы выводить какой-либо свой тип контейнера внутри ячейки, нужно написать перегрузку функции GetCellValueFromStruct. В ячейку можно выводить значения длиной до 3х символов. Если выводимое значение имеет большую длину, то можно, например, выводить остаток этого значения от деления на 1000 либо отсечь часть символов.

2.3.2. mazeWeighted

Наследует класс maze. Позволяет создавать лабиринты с весами ячеек.

2.3.2.1. Свойства:

`std::vector<std::vector<int>> Weights;`	Контейнер с весами ячеек

2.3.2.2. Методы:

mazeWeighted(int Nn, int Nm, int FillWeight=0)	Конструктор с параметрами. Создает лабиринт из изолированных друг от друга ячеек с весом по умолчанию равным нулю.
void WeightsToValues()	Не используется. Лучше использовать ShowDecorate с параметром Weights. Копирует веса в значения ячеек для удобного представления.

2.3.3. statisticsMaze

Класс для исследования функций, выполняющих генерацию лабиринтов.

2.3.3.1. Свойства:

`int CurrentN=0;`	Текущая высота лабиринта
`int CurrentM=0;`	Текущая ширина лабиринта
`int NStart=0, NEnd=0, NStep=0;`	Начальное, конечное значения высоты лабиринта и шаг её изменения
`double MRatio=1;`	M/N отношение ширины к длине
`int NumberOfRuns=1;`	число прогонов для текущих значений N M

2.3.3.2. Методы:

StatisticsMaze(int NNStart, int NNEnd, int NNStep, double NMRatio, int NNumberOfRuns, std::default_random_engine& generator, presenthandler& PresentHandler, std::string filename, void (*CallBackGenerate)(maze&, std::default_random_engine&, presenthandler&, CallBackParamsTail&...), CallBackParamsTail& ...callbackparamstail)	Конструктор. Запускает функцию генерации лабиринта со значениями высоты от NNStart до NNEnd с шагом NNStep и отношением ширины к высоте NMRatio. Записывает время генерации в файл.

void printLabel(std::string filename)	Запись информации об условиях выполнения сортировки в файл
void printLabel(std::string filename)	Запись искомых значений времени в файл

2.3.1. statisticsMazeSearch

Аналогичен классу statisticsMaze. Следует отметить, что передача в конструктор генератора позволяет для различных алгоритмов поиска генерировать одинаковую последовательность лабиринов, причем последовательности стартовых и финишных ячеек так же будут совпадать.

2.3.1.1. Свойства:

`int CurrentN=0;`	Текущая высота лабиринта
`int CurrentM=0;`	Текущая ширина лабиринта
`int NStart=0, NEnd=0, NStep=0;`	Начальное, конечное значения высоты лабиринта и шаг её изменения
`double MRatio=1;`	M/N отношение ширины к длине
`int NumberOfRuns=1;`	число прогонов для текущих значений N M

2.3.1.2. Методы:

void StatisticsMazeSearchFunc(int NNStart, int NNEnd, int NNStep, double NMRatio, int NNumberOfRuns, std::default_random_engine& generator, presenthandler& PresentHandler, std::string filename, void (*CallBackSearch)(T&, int starti, int startj, int finishi, int finishj, std::vector<std::pair<int,int>>& Path, presenthandler& PrHandler))	Генерирует лабиринт со значениями высоты от NNStart до NNEnd с шагом NNStep и отношением ширины к высоте NMRatio. Запускает для него функцию поиска между случайно выбранными ячейками и записывает время поиска в файл
void printLabel(std::string filename)	Запись информации об условиях выполнения сортировки в файл
void printLabel(std::string filename)	Запись искомых значений времени в файл

3. ОПИСАНИЕ АЛГОРИТМОВ

3.1. Алгоритмы генерации лабиринта

В данной работе реализованы алгоритмы генерации идеальных лабиринтов, т.е. лабиринтов без недостижимых областей и без циклов. Генерация происходит методом вырезания проходов.

3.1.1. Олдоса-Бродера

3.1.1.1. Теоретические сведения

Тип(Дерево/Множество): Дерево

Фокус(Возможность вырезать/добавлять стены): Оба

Смещенность(одинаково ли легко перемещаться по лабиринту в различных направлениях): отсутствует

Однородность(Генерирует ли алгоритм все возможные лабиринты с равной вероятностью): Да

Память: 0

3.1.1.2. Алгоритм

1) Лабиринт состоит из изолированных стенами ячеек

2) Выбирается случайная ячейка и отмечается как посещенная

3) Счетчик посещенных вершин k устанавливается равным 1

4) Пока k не равен числу ячеек в лабиринте

   1) Выбирается случайное направление движения

   2) Если это направление не направлено в стену, то

      1) проверяем, была ли уже посещена эта ячейка. Если нет, то прорезаем стену к ней, переходим к ней и помечаем её как посещенную

   3) В противном случае просто переходим к ней

3.1.1.3. Пример работы

Символом «1» функция помечает посещенные ячейки, «.» - непосещенные. Также будем обозначать символом «*» текущую ячейку

Система координат: (i, j) i увеличивается сверху вниз. j-слева направо. Счет с нуля.

1) Начальная ячейка была выбрана с координатами (1, 1)

.	.	.
.	1*	.
.	.	.

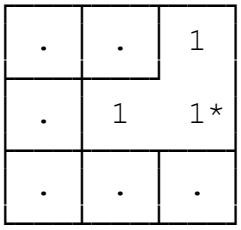
2) Случайным образом было выбрано направление в правую сторону. Поскольку ячейка справа ещё не была посещена, то стена к ней прорезается. Происходит переход к этой ячейке ячейке (1, 2)

.	.	.
.	1	1*
.	.	.

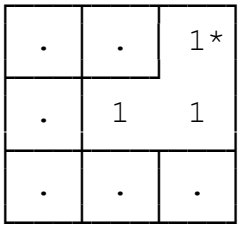
3) Аналогично проходится ячейка (0,2)

.	.	1*
.	1	1
.	.	.

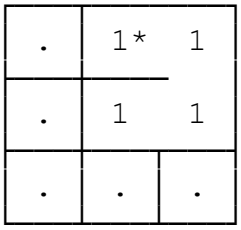
4) Выбирается направление вниз. Так как нижняя ячейка уже была посещена, то стена к ней не прорезается(в случае, если бы она там была, это было бы важно) и происходит переход к этой ячейке



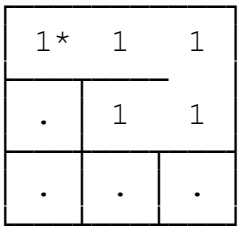
5) Аналогично 4) возвращаемся к ячейке (0, 2)



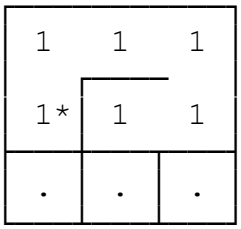
6) Аналогично шагу 2) шаги 6)-11)



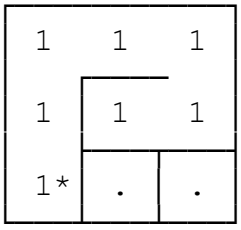
7)



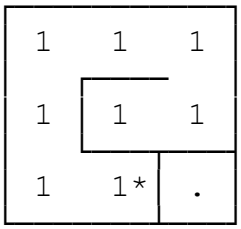
8)



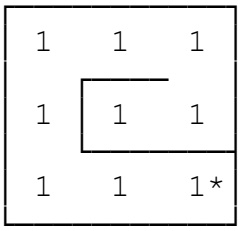
9)



10)



11) После 11 шага работа алгоритма завершается, т.к. количество посещенных ячеек стало равно количеству ячеек в лабиринте.



3.1.1.4. Примеры лабиринтов

Рисунок 1 – фрагмент лабиринта, сгенерированного алгоритмом Олдоса-Бродера (2500x2500 ячеек)

Рисунок 2 – лабиринт, сгенерированный алгоритмом Олдоса-Бродера (100x100 ячеек)

3.1.2. Уилсона

3.1.2.1. Теоретические сведения

Тип(Дерево/Множество): Дерево

Фокус(Возможность вырезать/добавлять стены): Оба

Смещенность(одинаково ли легко перемещаться по лабиринту в различных направлениях): отсутствует

Однородность(Генерирует ли алгоритм все возможные лабиринты с равной вероятностью): Да

Память: N^2

3.1.2.2. Алгоритм

1) Лабиринт состоит из изолированных стенами ячеек

2) Выбрать случайную ячейку и добавить её в множество UST (uniform spanning trees — однородные остовные деревья).

3) Пока есть ячейки, не входящие в UST, выбрать случайную ячейку и совершить от неё случайную прогулку, пока не будет встречена какая-нибудь ячейка из UST. Во время прогулки в ячейки записываются путевые координаты. Причем если во время такой прогулки мы наткнемся на ячейку, в которой уже были, т.е. сделаем петлю, то её путевые координаты перезапишутся, т.е. эта петля не попадет в итоговый путь (по-видимому, это одна из причин, по которой этот алгоритм такой медленный)

4) Проходим по путевым координатам от ячейки, с которой мы начали прогулку, до ячейки из UST по путевым координатам, прорезая стены (сразу это нельзя сделать из-за петель, которые срезает алгоритм во время первого прохода).

3.1.2.3. Пример работы

Символом «U» будем помечать ячейки в UST, «>»^»«<»«v» - путевые координаты. «.» - непосещенные ячейки. Также будем обозначать символом «*» текущую ячейку

Система координат: (i, j) i увеличивается сверху вниз. j-слева направо. Счет с нуля.

1) Начальная ячейка была выбрана с координатами (1, 2). Добавляем её в UST.

.	.	.
.	.	U*
.	.	.

2) Выберем случайную ячейку, которой нет в UST. Например, (2, 0)

.	.	.
.	.	U*
*	.	.

3) Начнем блуждание, пока не дойдем до ячеек, которые есть в UST

.	.	.
.	.	U
^*	.	.

4)

.	.	.
^*	.	U
^	.	.

5)

>*	.	.
^	.	U
^	.	.

6)

>	v*	.
^	.	U
^	.	.

7)

>	v	.
^	<*	U
^	.	.

8) Путевую координату ячейки (1, 0) пезезаписали, т.к. мы случайным образом выбрали направление вниз, а не вверх.

>	v	.
v*	<	U
^	.	.

9) Путевую координату ячейки (2, 0) пезезаписали, т.к. мы случайным образом выбрали направление вправо, а не вверх.

>	v	.
v	<	U
>*	.	.

10) Продолжаем блуждание

>	v	.
v	<	U
>	^*	.

11)

>	v	.
v	>*	U
>	^	.

12) Мы дошли до ячейки в UST.

>	v	.
v	>	U*
>	^	.

13) Прорезаем путь от исходной ячейки до ячейки в UST. Добавляя ячейки в UST.

>	v	.
v	>	U
U*	^	.

14)

>	v	.
v	>	U
U	U*	.

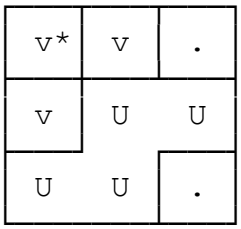
15)

>	v	.
v	U*	U
U	U	.

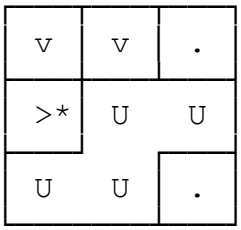
16) Путь прорезан. Путевые координаты ячеек, которые были в петлях, оставляем. На работу алгоритма никакого влияния они не окажут.

>	v	.
v	U	U*
U	U	.

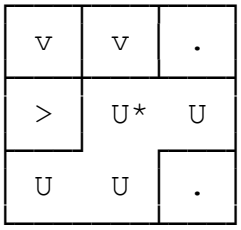
17) Аналогично включаем в UST оставшиеся ячейки.



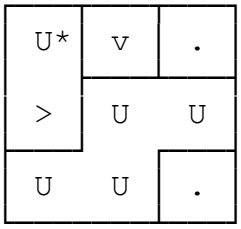
18)



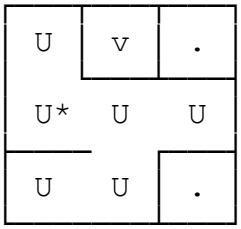
19)



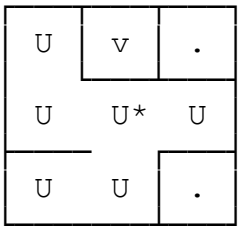
20)



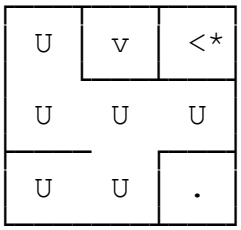
21)



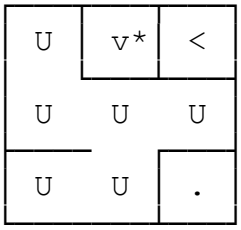
22)



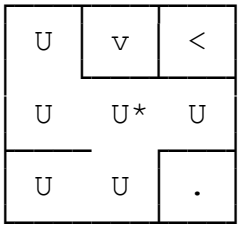
23)



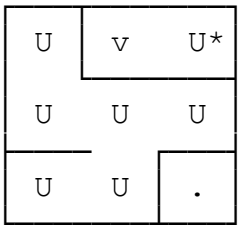
24)



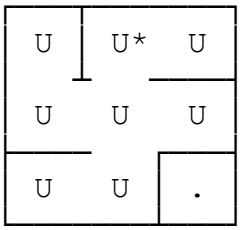
25)



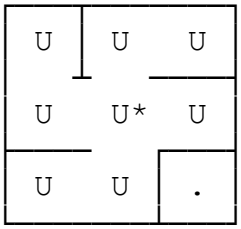
26)



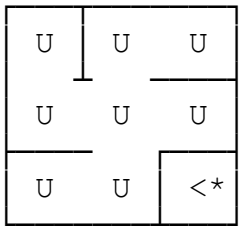
27)



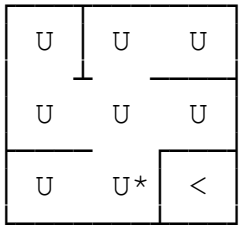
28)



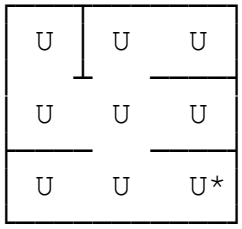
29)



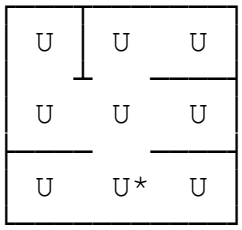
30)



31)



32)



3.1.2.4. Примеры лабиринтов

Лабиринты, генерируемые алгоритмом Уилсона, по характеру не отличимы от лабиринтов, генерируемых алгоритмом Олдоса-Бродера.

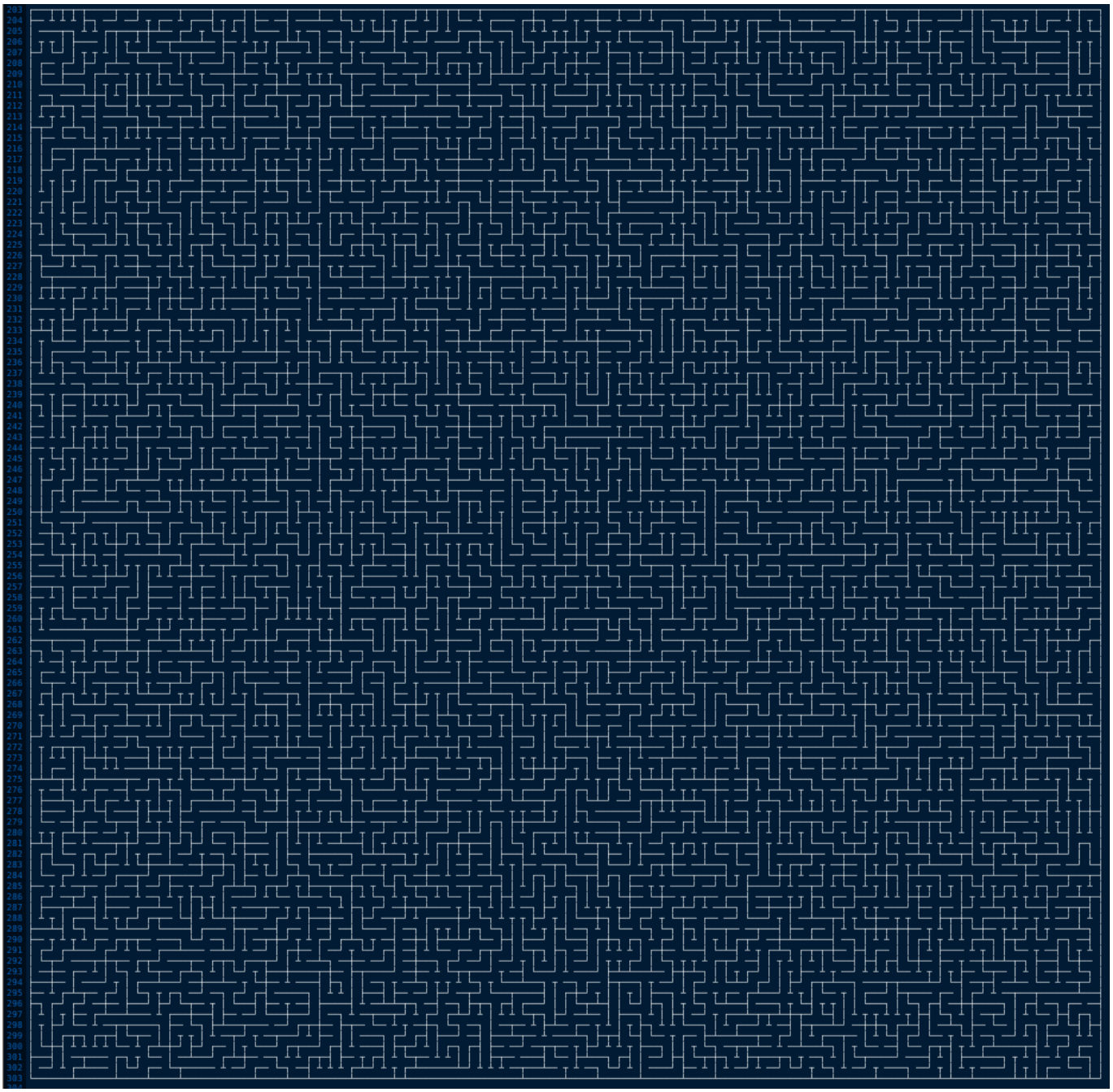


Рисунок 3 – Лабиринт, сгенерированный алгоритмом Уилсона. 100x100 ячеек

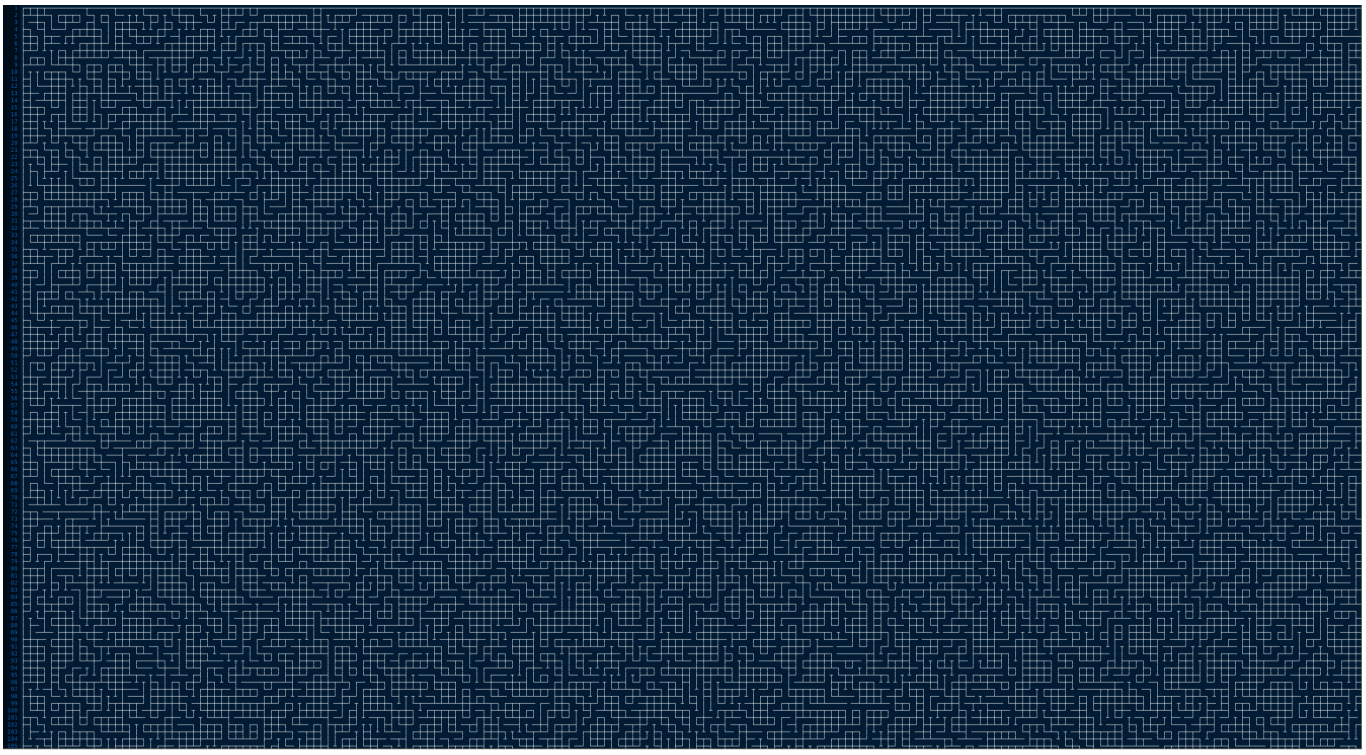


Рисунок 4 – Фрагмент лабиринта во время генерации алгоритмом Уилсона. 1000x1000 ячеек

3.1.3. Уилсона (модификация)

Данный алгоритм отличается от предыдущего методом выбора ячейки, которой ещё нет в множестве UST. В отличие от предыдущего, здесь ячейки выбираются последовательно от начала лабиринта. Отличие наглядно показано на рисунке:

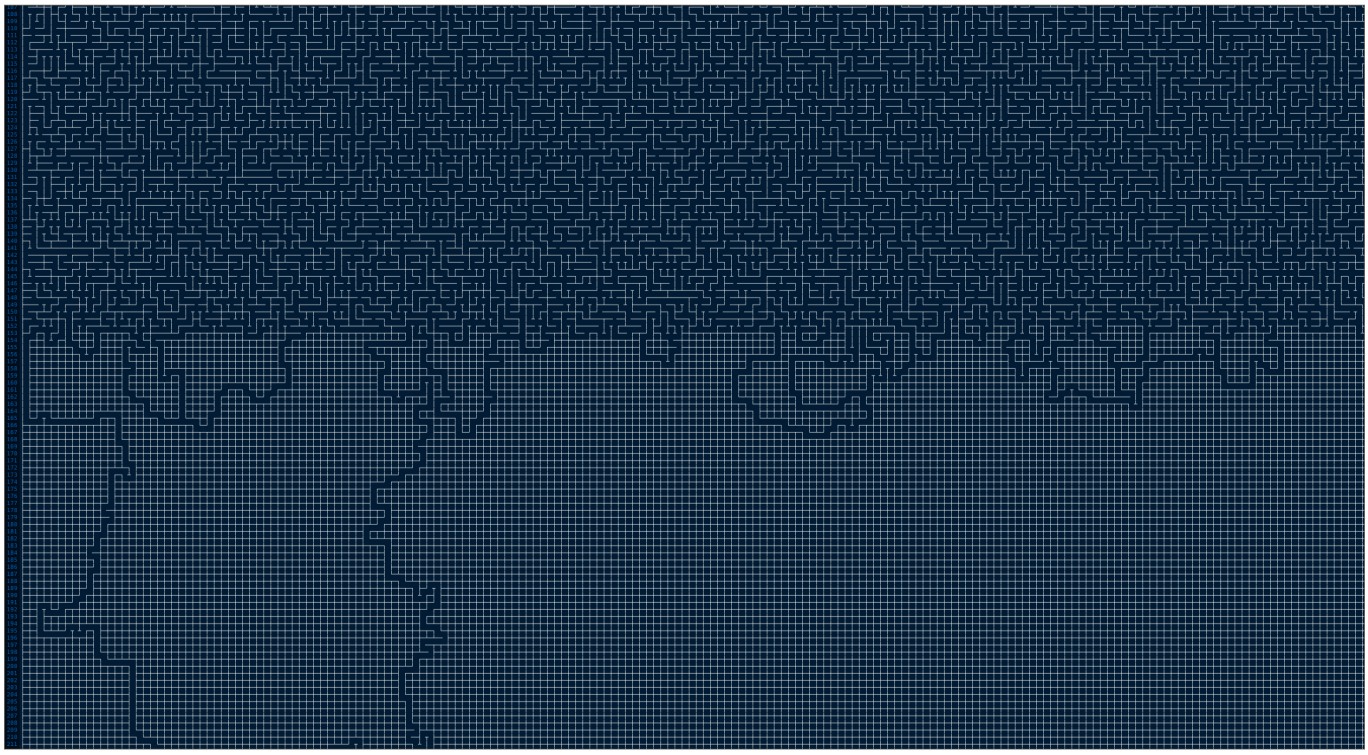


Рисунок 5 – Фрагмент лабиринта во время генерации алгоритмом Уилсона (модификация). 500x500 ячеек

3.1.4. Двоичным деревом

3.1.4.1. Теоретические сведения

Тип(Дерево/Множество): Множество

Фокус(Возможность вырезать/добавлять стены): Оба

Смещенность(одинаково ли легко перемещаться по лабиринту в различных направлениях): Присутствует.

Однородность(Генерирует ли алгоритм все возможные лабиринты с равной вероятностью):Никогда

Память: 0*

3.1.4.2. Алгоритм

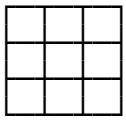
1) Лабиринт состоит из изолированных стенами ячеек

2) Для каждой ячейки в сетке случайным образом разделяем проход на север или запад (север или восток / юг или запад / юг или восток) двигаясь построчно. В данной реализации сразу вырезается один из столбцов (такой же результат можно было бы получить, двигаясь построчно, пробегая все столбцы).

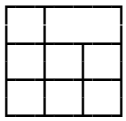
3.1.4.3. Пример работы

Продемонстрируем генерацию для выбранного направления север-восток

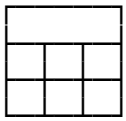
1) Изначальное состояние поля



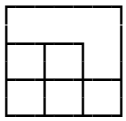
2) Вырезается первая/последняя строка (при выбранном направлении север/юг)



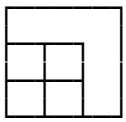
3)



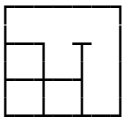
4) Вырезается левый/правый столбец, согласно выбранному направлению (запад/восток)



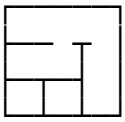
5)



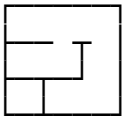
6) Далее построчно (для данных направлений построчно сверху вниз)



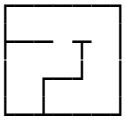
7)



8)



9)

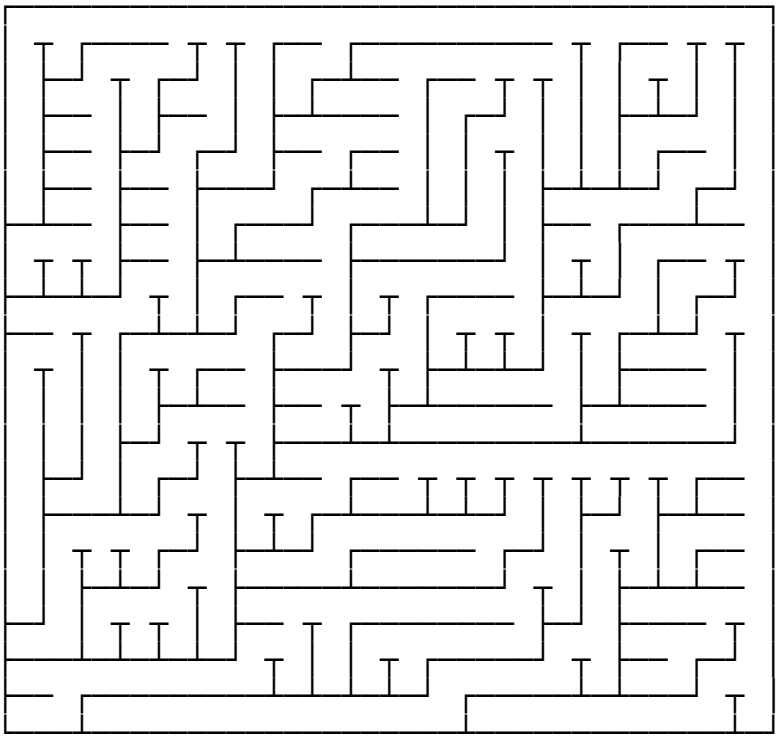


3.1.4.4. Примеры лабиринтов

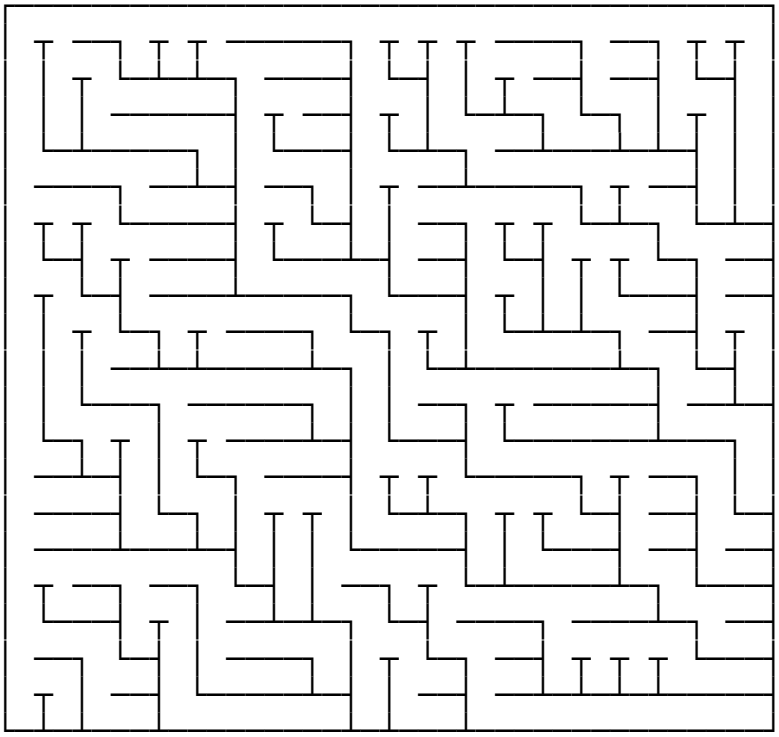
Лабиринты, сгенерированные этим алгоритмом, имеют заметную на глаз смещенность.

Приведем сгенерированные лабиринты (20x20) для различных выбранных направлений:

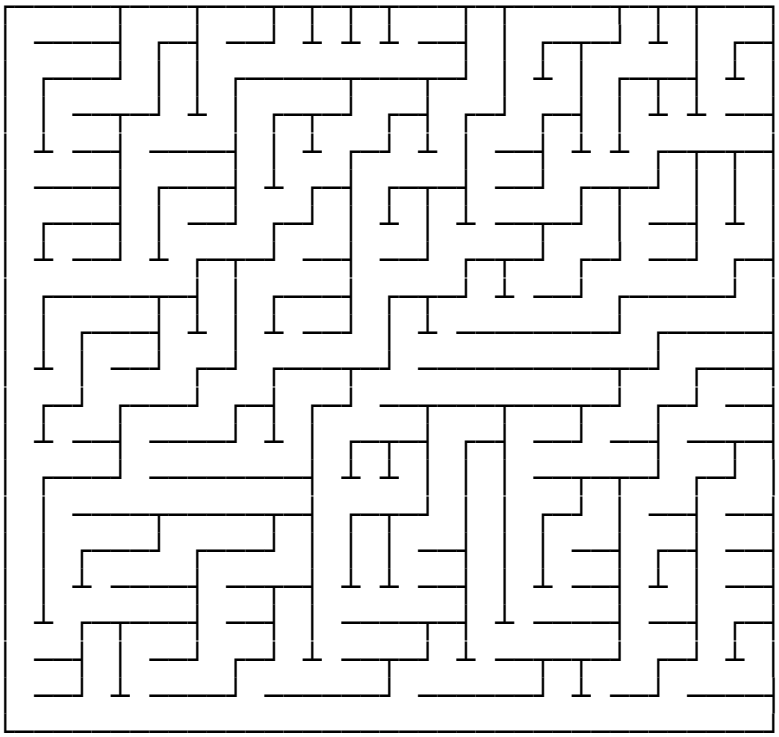
Север-восток:



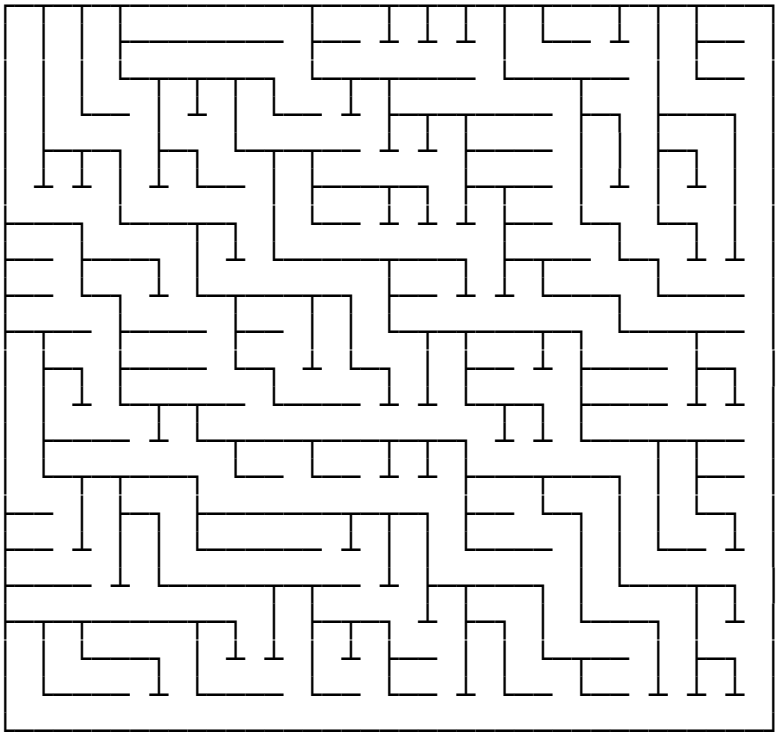
Север-запад:



Юг-запад:



Юг-восток:



3.2. Алгоритмы поиска пути в лабиринте

3.2.1. Подготовка лабиринта

Алгоритмы генерации, реализованные ранее, создают идеальные лабиринты. Это значит, что между любыми двумя ячейками существует путь, причем единственный. При поиске путей более интересной, на мой взгляд, является ситуация, когда между ячейками могут существовать альтернативные пути, причем стоимость (вес) пути между различными ячейками может быть различным.

Поэтому был создан класс mazeWeighted и написаны следующие функции:

WallsReduce – для разряжения лабиринта, т.е. случайного удаления его стен с заданной вероятнсотью.

```
void RandomCircules(mazeWeighted& MazeWeighted, std::default_random_engine&
generator, int minWeight, int maxWeight, double probability, int meanRadius,
double stddevRadiusRatio)
```

– для генерации весов лабиринта в виде кругов со случайным радиусом (имеющим нормальное распределение вероятности) и находящихся в случайных ячейках. мат.ожидание радиуса задается с помощью meanRadius, stddevRadiusRatio – среднеквадратическое отклонение радиуса круга, деленное на матожидание круга. Примеры получившихся лабиринтов можно видеть в примерах алгоритмов поиска.

3.2.2. Ли

3.2.2.1. Теоретические сведения

Алгори́тм волново́й трассиро́вки (волновой алгоритм, алгоритм Ли) — алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

В основном используется при компьютерной трассировке (разводке) печатных плат, соединительных проводников на поверхности микросхем. Другое применение волнового алгоритма — поиск кратчайшего расстояния на карте в компьютерных стратегических играх.

Волновой алгоритм в контексте поиска пути в лабиринте был предложен Э. Ф. Муром. Ли независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году.

Алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

3.2.2.2. Алгоритм

В данной работе дискретное поле не содержит непроходимых ячеек, однако между ячейками на этом поле могут быть стены. Стоит отметить, что в данной реализации не используется очередь, характерная для алгоритмов поиска в ширину. Что, предположительно, может являться причиной того, что этот алгоритм более быстрый, чем алгоритм Дейкстры (который, будучи примененным для невзвешенных полей, должен давать аналогичный результат за, возможно, чуть большее время) на полях небольших размеров и значительно более медленный на полях больших размеров. (вместо извлечения элементов из очереди алгоритм пробегает ячейки в прямоугольной области, ограниченной ячейками фронта волны)

1) Вводится матрица весов путей (стоимости путей от стартовой ячейки до текущей)

2) Матрица так же используется для отличия посещенных ячеек от непосещенных (для непосещенных ячеек устанавливается вес, максимальный для int на данной платформе)

3) Начинается распространение волны от стартовой ячейки: ей присваивается нулевой вес.

4) Далее идет распространение волны: для каждой ячейки фронта волны просматриваются соседи. Если соседи ещё не были посещены, то им присваивается вес. Также отслеживается, достигнута ли финишная ячейка и удалось ли продвинутся фронту волны.

5) Если финишная ячейка достигнута, либо же фронт волны больше не может распространятся, то происходит завершение распространения волны. Если завершение было связано с достижением финишной ячейки, то начинается восстановление пути, в противном случае выдается сообщение о несуществовании пути, и происходит выход из функции.

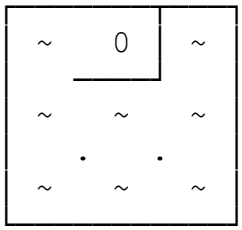
6) Восстановление пути идет из финишной в начальную ячейку. Для того, чтобы восстановить путь нужно двигаться из финишной в начальную ячейку так, чтобы при движении вес пути уменьшался на единицу.

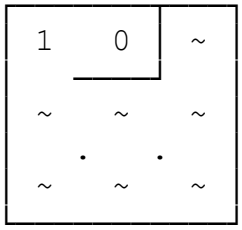
3.2.2.3. Пример работы

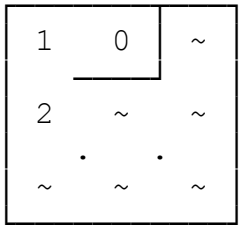
Пример, когда путь существует. (цифры – веса пути; ~ - непосещенные ячейки)

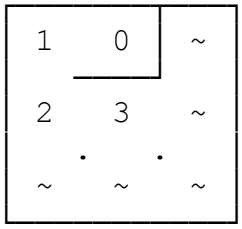
Поиск пути из (0, 1) в (1, 1)

1) Распространение волны от (0, 1)

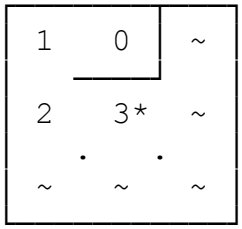




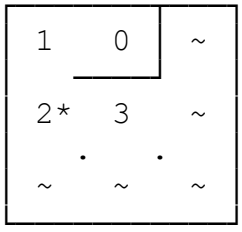




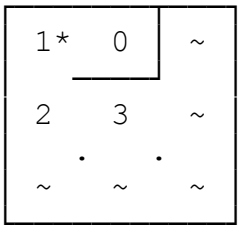
2) Восстановление пути с конца.



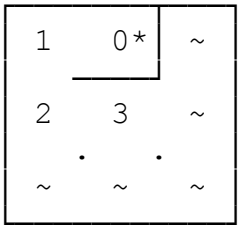

```
Way=  (1, 1)
```




```
Way=  (1, 0)  (1, 1)
```




```
Way=  (0, 0)  (1, 0)  (1, 1)
```




```
Way=  (0, 1)  (0, 0)  (1, 0)  (1, 1)
```
– Искомый путь.

3.2.3. Ли (модификация с двумя волнами)

Более сложная реализация алгоритма Ли. Отличие заключается в том, что помимо волны, распространяющейся от стартовой ячейки, создается волна, распространяющаяся от финишной ячейки. Фронты волн продвигаются по дискретному полю поочередно, пока волны не встретятся, или пока одна из волн не сможет продвигаться дальше. В этой реализации помимо матрицы весов путей вводится матрица идентификаторов волн.

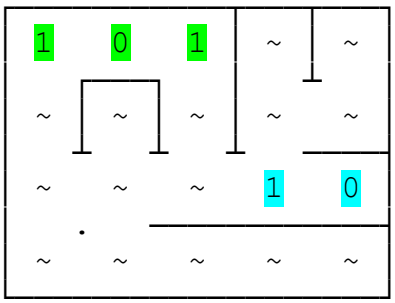
Если волны встретились, то путь восстанавливается в два этапа: сначала от стартовой ячейки до ячейки пересечения волн, затем от ячейки пересечения волн до финишной ячейки.

Данная реализация позволяет посещать примерно в два раза меньшее количество ячеек, чем вариант с одной волной.

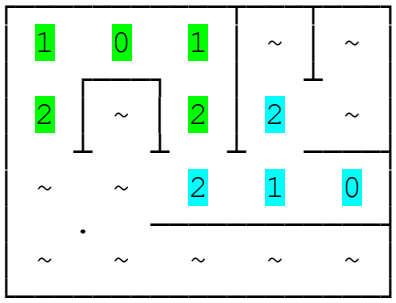
3.2.3.1. Пример работы

Поиск пути от (0, 1) до (2, 4). Зеленый – волна от стартовой ячейки, голубой – от финишной.

```
Wave1 CurrDist=1
Wave2 CurrDist=1
```

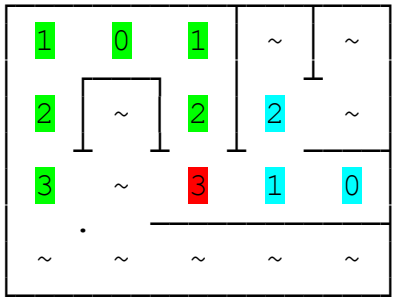


```
Wave1 CurrDist=2
Wave2 CurrDist=2
```

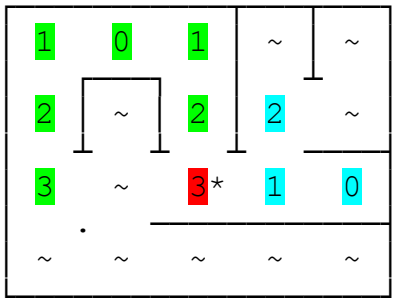


Волны пересеклись в ячейке (2, 2). Обозначим её красным цветом.

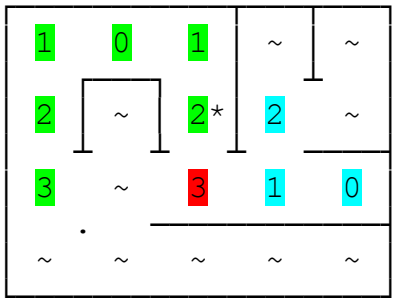
```
Wave1 CurrDist=3
```



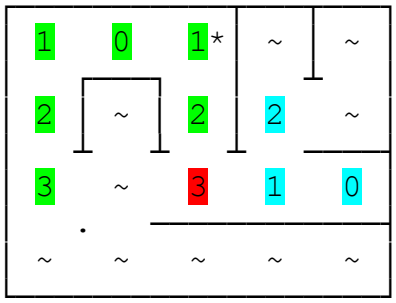
Рассмотрим теперь восстановление пути:
От стартовой до ячейки пересечения:



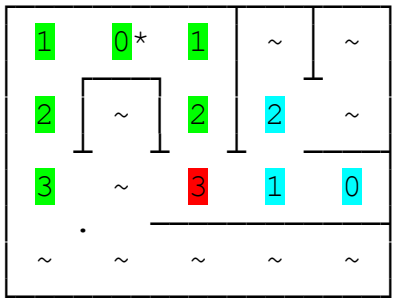

```
Way= (2, 2)
```




```
Way= (1, 2)  (2, 2)
```

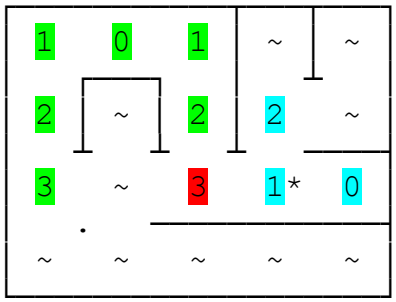


Way= (0, 2) (1, 2) (2, 2)

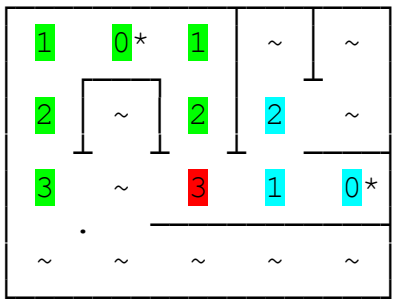


Way= (0, 1) (0, 2) (1, 2) (2, 2)

От ячейки пересечения до финишной:



Way= (0, 1) (0, 2) (1, 2) (2, 2) (2, 3)



Way= (0, 1) (0, 2) (1, 2) (2, 2) (2, 3) (2, 4) – Искомый путь

3.2.4. Дейкстры

3.2.4.1. Алгоритм

Алгоритм Дейкстры – это алгоритм на графе. На ДРП этот алгоритм применяется, когда веса (стоимости) перехода (не путать с путевыми весами) из одной ячейки в другую могут быть различны. Если все веса одинаковы, то алгоритм Дейкстры будет давать тот же результат, как и поиск в ширину.

Основное отличие алгоритма Дейкстры от алгоритма поиска в ширину – использование очереди с приоритетами.

Итак, при работе алгоритма будут введены: PathCoordWeights – матрица, каждый элемент которой – это путевые координаты ячейки и её путевой вес.

PriorQueue – очередь с приоритетами, меньший вес – больший приоритет. Будет хранить в себе координаты ячеек.

1) Устанавливаем путевой вес стартовой ячейки равным 0.

2) Помещаем стартовую ячейку в очередь

3) Извлекаем ячейку из очереди

4) Проверяем всех её соседей (всего 4 соседа) :Если сосед ни разу не был посещен, то устанавливаем для него путевые координаты(с какого направления к нему пришли) и путевые веса. Если сосед уже был посещен, то находим для него новый путевой вес и сравниваем с текущим. Если новый вес оказался не меньше текущего, то не делаем ничего, если же новый вес оказался меньше текущего, то посещенному соседу присваиваем новый вес, новые путевые координаты и помещаем в очередь (он оказывается в ней уже повторно)

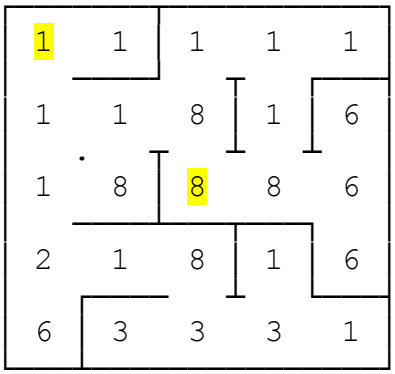
5) Продолжаем выполнять пункты 3-4, пока извлеченный элемент не окажется финишной ячейкой, либо пока в очереди не останется элементов.

6) Узнав причину остановки, делаем вывод о том, существует ли путь или нет.

7) Если путь существует, то восстанавливаем его, двигаясь по путевым координатам от финиша к старту.

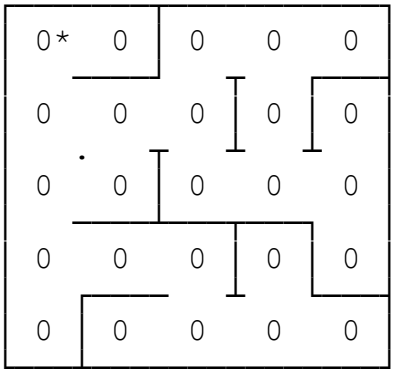
3.2.4.2. Пример работы

Пусть исходный лабиринт выглядит так:

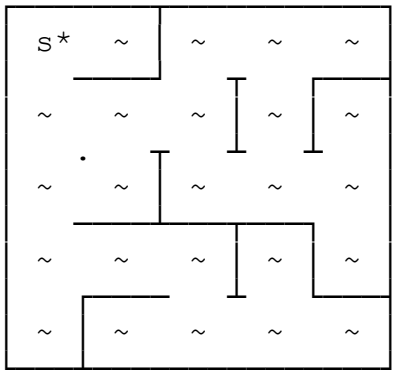


Цифры означают веса ячеек, т.е. стоимость при перемещении в данную ячейку из соседней.
Найдем путь из ячейки (0, 0,) в ячейку (2, 2). Отмечены желтым.
1) Выставим атрибуты первой ячейки и поместим её в очередь.

Путевые веса :

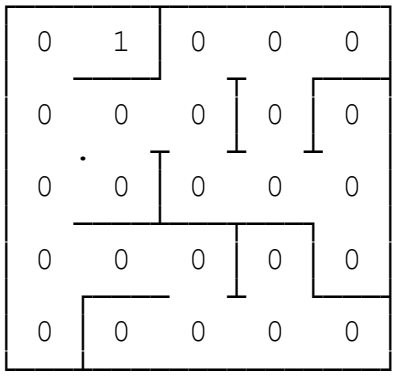


Путевые координаты :

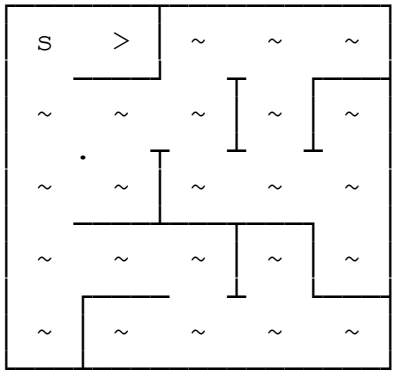


2) Извлечем эту ячейку из очереди и пройдемся по её соседям. Вначале посмотрим левого соседа:

`PathWeights % 1000:`

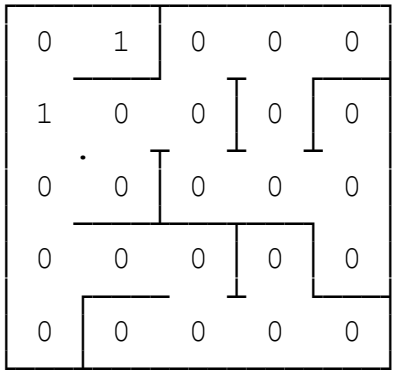


`PathCoordinates:`

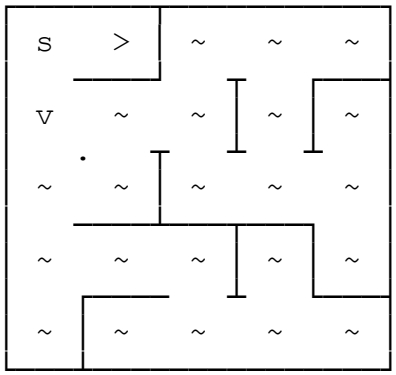


3) Потом нижнего соседа

`PathWeights % 1000:`

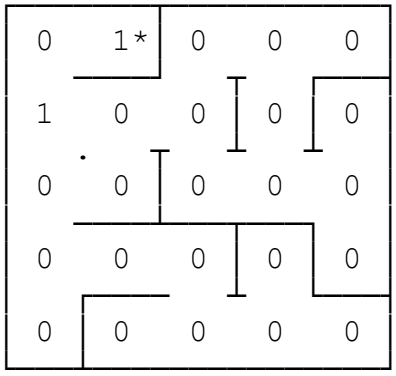


PathCoordinates:

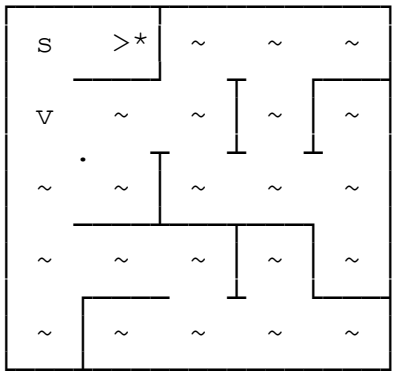


4) Теперь в очереди находятся 2 ячейки с одинаковым приоритетом 1. Это ячейки (0, 1) и (1, 0). Извлечем из очереди ячейку (0, 1). Она имеет единственного соседа, неотделенного стеной – это ячейка (0, 0). Найдем новый вес для неё: новый путевой вес(0, 0)=путевой вес(0, 1)+вес перемещения(0, 0)=1+1=2 Это больше, чем 0. Значит, с ячейкой (0, 0) мы ничего не делаем.

PathWeights % 1000:

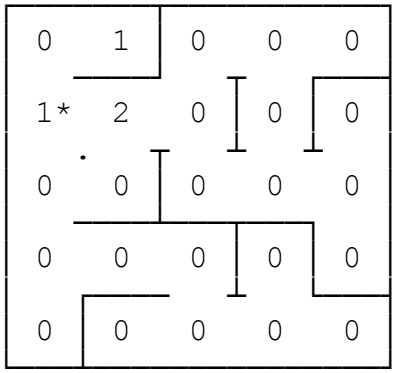


PathCoordinates:

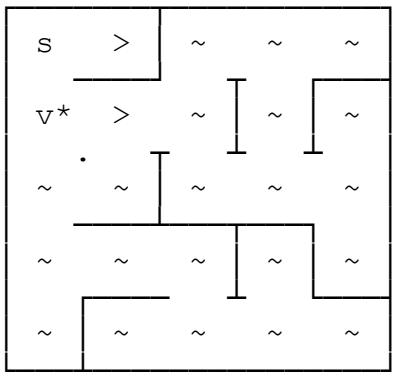


5) Теперь в очереди находится единственная ячейка (1, 0). Она имеет трех соседей. Обойдем их последовательно.

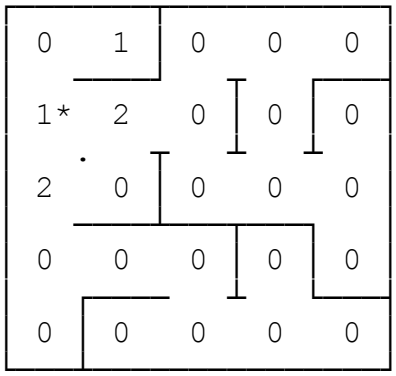
PathWeights % 1000:



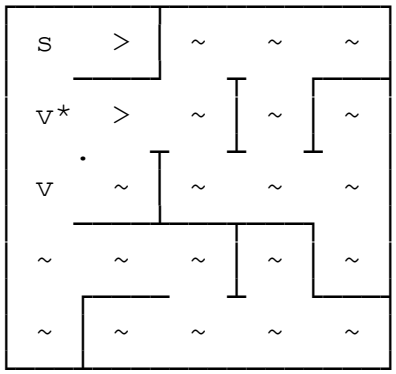
PathCoordinates:



PathWeights % 1000:

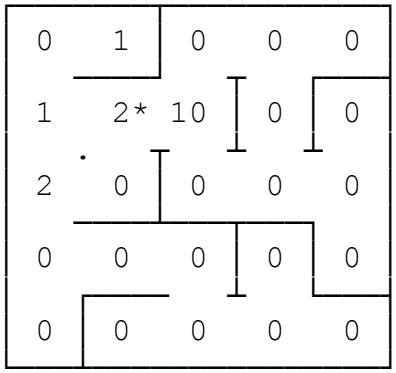


PathCoordinates:

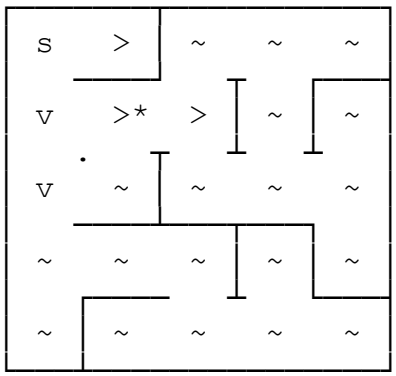


6) В очереди теперь две ячейки (1, 1) и (2, 0). Обе эти ячейки имеют приоритет 2. Извлечем из очереди ячейку (1, 1) и пройдемся по её соседям.

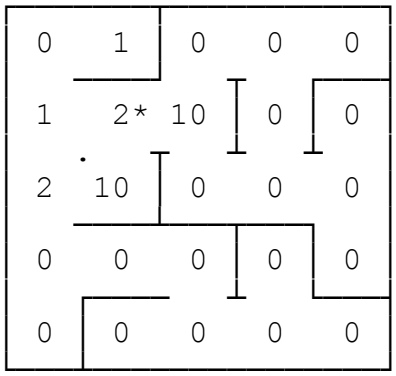
PathWeights % 1000:



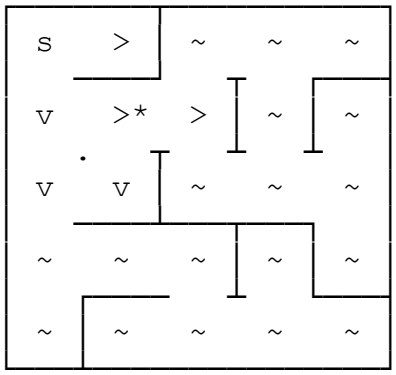
PathCoordinates:



PathWeights % 1000:



PathCoordinates:



7) Теперь в очереди лежит по прежнему ячейка (2, 0) с приоритетом 2 и лежат ячейки (1, 2) и (2, 1) с приоритетами 10. Выберем наиболее приоритетную ячейку, т.е. ячейку (2, 0). Пройдемся по её соседям.

PathWeights % 1000:

0	1	0	0	0
1	2	10	0	0
2*	10	0	0	0
4	0	0	0	0
0	0	0	0	0

PathCoordinates:

s	>	~	~	~
v	>	>	~	~
v*	v	~	~	~
v	~	~	~	~
~	~	~	~	~

8) Продолжаем в том же духе

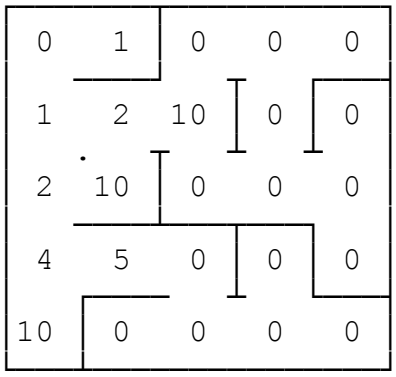
PathWeights % 1000:

0	1	0	0	0
1	2	10	0	0
2	10	0	0	0
4	5	0	0	0
0	0	0	0	0

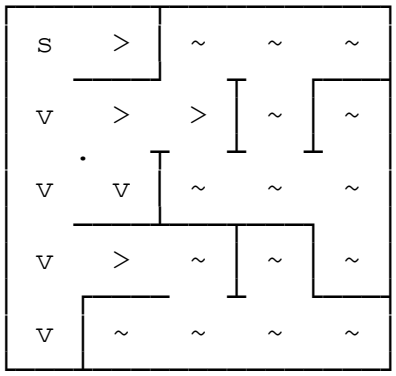
PathCoordinates:

s	>	~	~	~
v	>	>	~	~
v	v	~	~	~
v	>	~	~	~
~	~	~	~	~

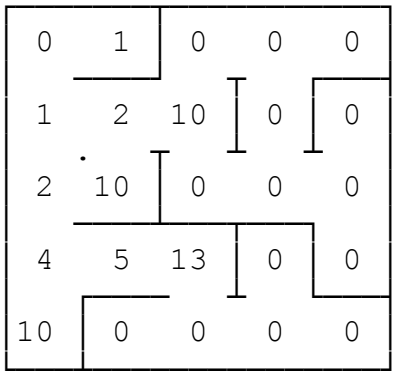
PathWeights % 1000:



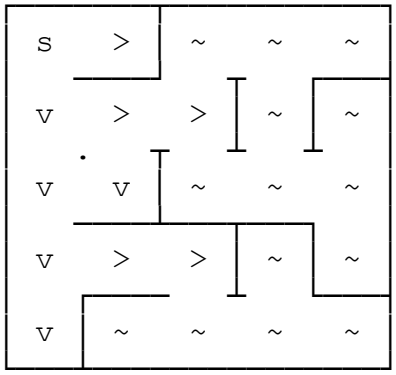
PathCoordinates:



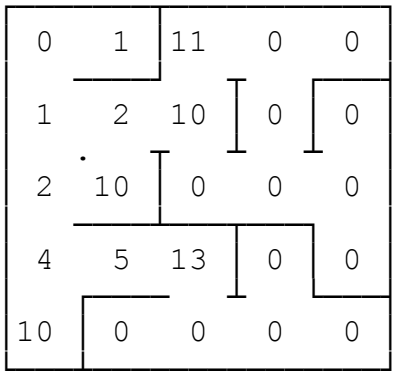
PathWeights % 1000:



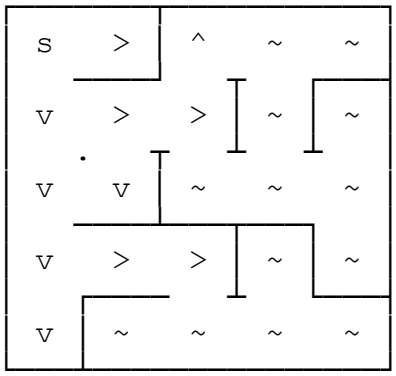
PathCoordinates:



PathWeights % 1000:

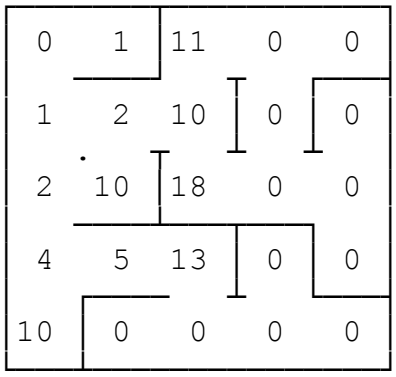


PathCoordinates:

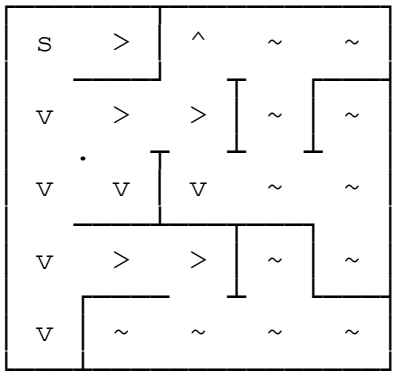


9) На этом шаге мы уже добрались до финишной ячейки. Но она не самая приоритетная в очереди, поэтому поиск продолжается (существуют случаи, когда удается найти путь короче, чем тот, который бы мы получили, сразу остановив поиск. Более того, когда мы дождемся, когда «фронт волны» полностью прокатится через полученный вес, то найденный путь окажется наикратчайшим из возможных. Т.е. алгоритм Дейкстры (в данной интерпретации, т.е. когда мы можем многократно ставить в очередь уже просмотренные элементы) гарантированно находит наикратчайший путь.).

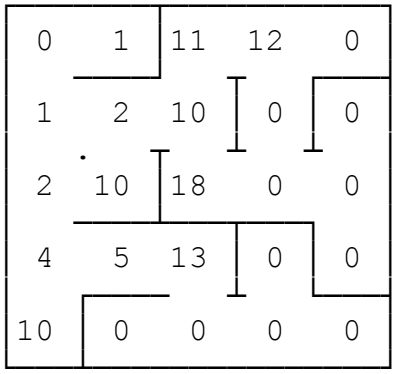
PathWeights % 1000:



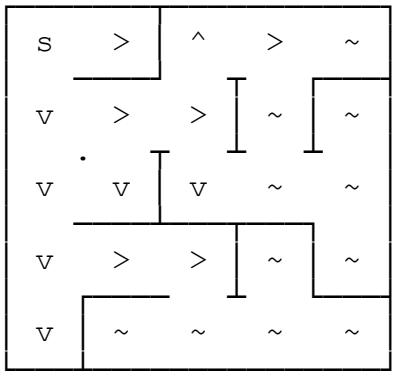
PathCoordinates:



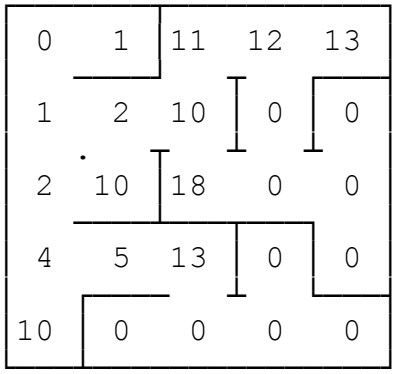
PathWeights % 1000:



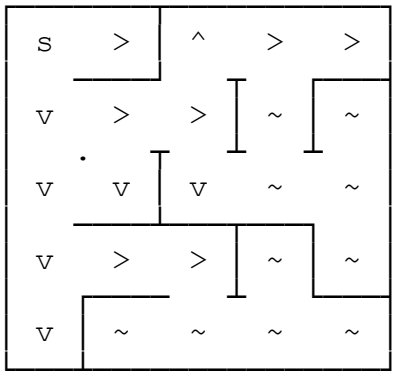
PathCoordinates:



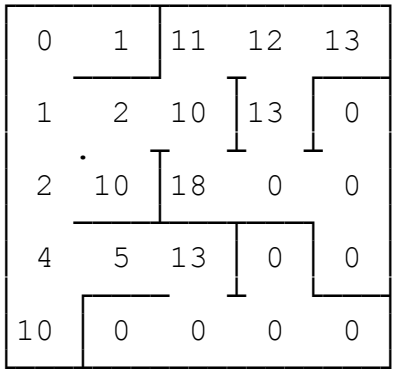
PathWeights % 1000:



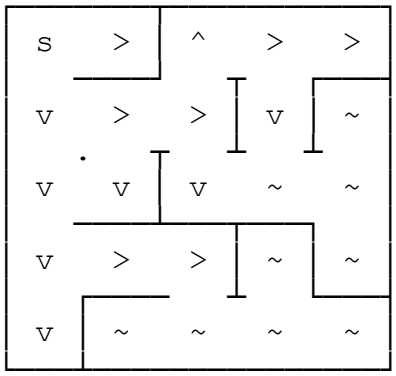
PathCoordinates:



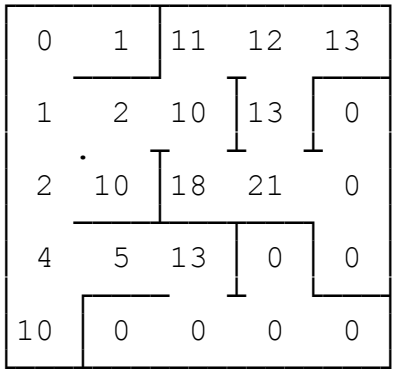
PathWeights % 1000:



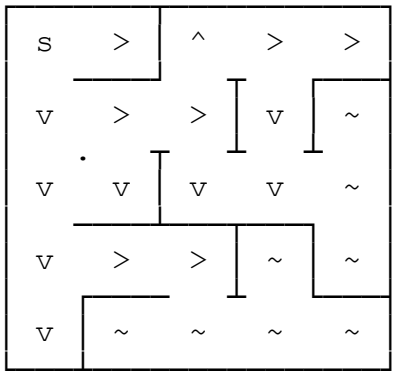
PathCoordinates:



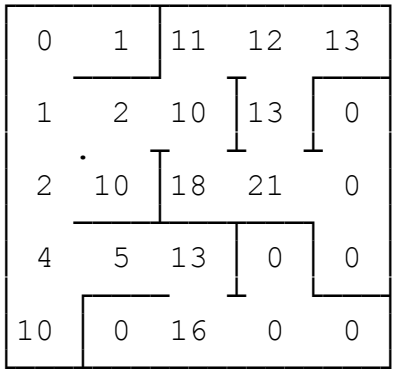
PathWeights % 1000:



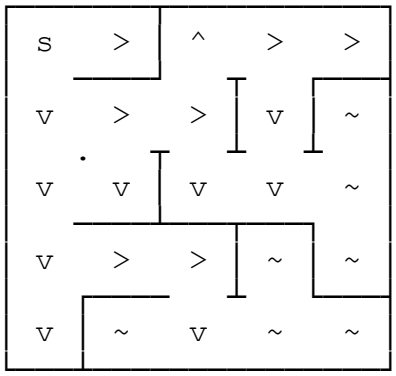
PathCoordinates:



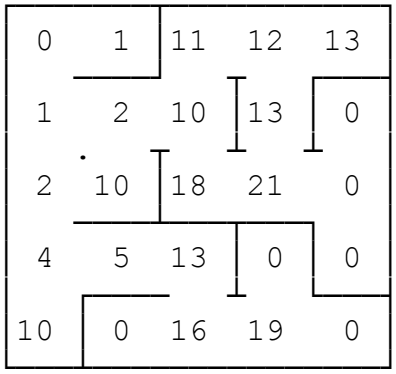
PathWeights % 1000:



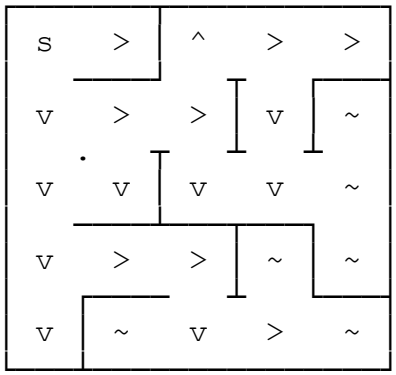
PathCoordinates:



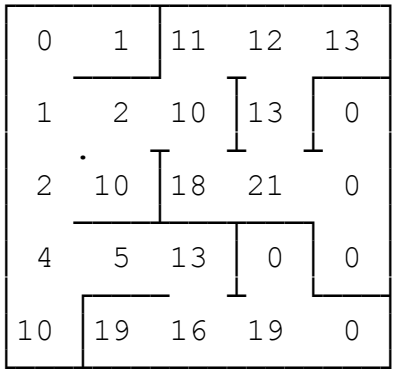
PathWeights % 1000:



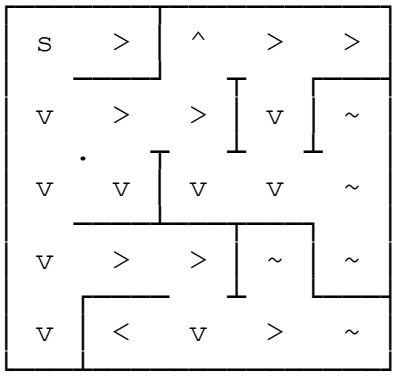
PathCoordinates:



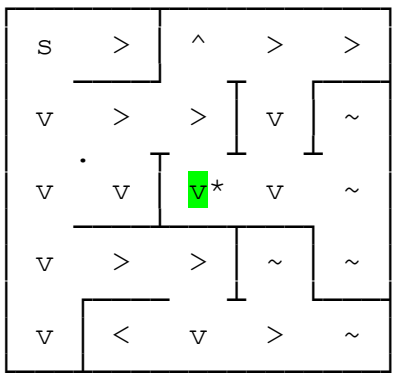
PathWeights % 1000:



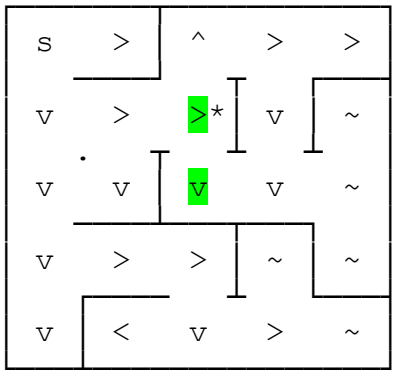
PathCoordinates:



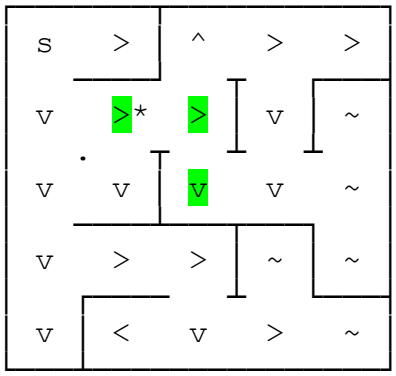
10) Наконец, очередной элемент, извлеченный из очереди, оказывается финишной ячейкой. На этом поиск прекращается, и по путевым координатам восстанавливается путь.(с конца в начало)



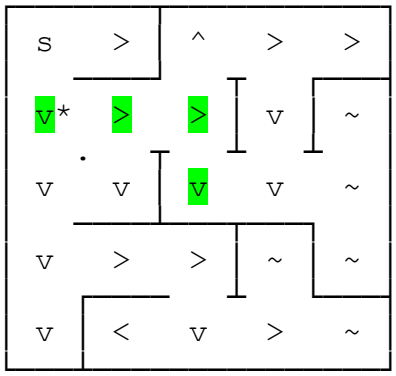
Path= (2, 2)



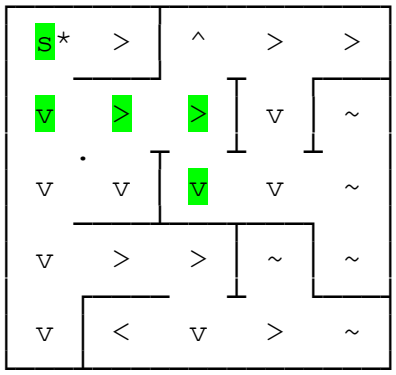
Path= (1, 2) (2, 2)



Path=(1, 1) (1, 2) (2, 2)



Path=(1, 0) (1, 1) (1, 2) (2, 2)



Path=(0, 0) (1, 0) (1, 1) (1, 2) (2, 2) – искомый путь

Вершина очереди при «распространении волны» на каждой итерации выглядит так (приоритет_он_же_путевой_вес, (I, j)):

```
PriorQueue all items:
PriorQueue.top()=(0, (0, 0))
PriorQueue.top()=(1, (0, 1))
PriorQueue.top()=(1, (1, 0))
PriorQueue.top()=(2, (1, 1))
PriorQueue.top()=(2, (2, 0))
PriorQueue.top()=(4, (3, 0))
PriorQueue.top()=(5, (3, 1))
PriorQueue.top()=(10, (1, 2))
PriorQueue.top()=(10, (2, 1))
PriorQueue.top()=(10, (4, 0))
PriorQueue.top()=(11, (0, 2))
PriorQueue.top()=(12, (0, 3))
PriorQueue.top()=(13, (0, 4))
PriorQueue.top()=(13, (1, 3))
PriorQueue.top()=(13, (3, 2))
PriorQueue.top()=(16, (4, 2))
PriorQueue.top()=(18, (2, 2))
```

3.2.5. A* (AStar; A со звездой)

3.2.5.1. Теоретические сведения

Поиск A* (произносится «А звезда» или «А стар», от англ. A star) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как f(x)). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как h(x)).

Функция h(x) должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации h(x) может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

Этот алгоритм был впервые описан в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем. Это по сути было расширение алгоритма Дейкстры, созданного в 1959 году. Новый алгоритм достигал более высокой производительности (по времени) с помощью эвристики. В их работе он упоминается как «алгоритм A». Но так как он вычисляет лучший маршрут для заданной эвристики, он был назван A*.

Обобщением для него является двунаправленный эвристический алгоритм поиска.

3.2.5.2. Алгоритм

Реализация данного алгоритма мало отличается от реализации алгоритма Дейкстры.

Для оценки расстояния от текущей ячейки до финишной вводится эвристическая функция, состоящая из одного оператора:

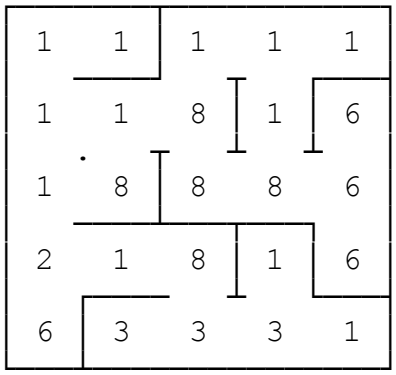
return abs(finishi-Currenti)+abs(finishj-Currentj);

Отличие от алгоритма Дейкстры в том, что в качестве приоритета в очереди берутся не путевые веса, а сумма путевого веса и эвристической оценки расстояния до финишной ячейки.

3.2.5.3. Пример работы

Исходный лабиринт с весами ячеек:

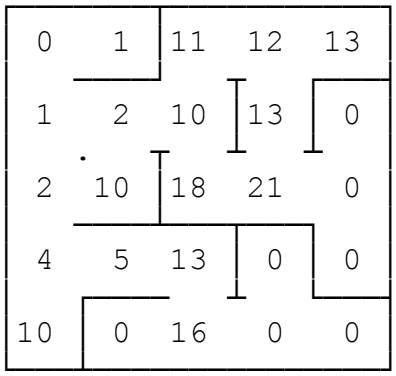
```
Weights:
```



Вершина очереди на каждой итерации:

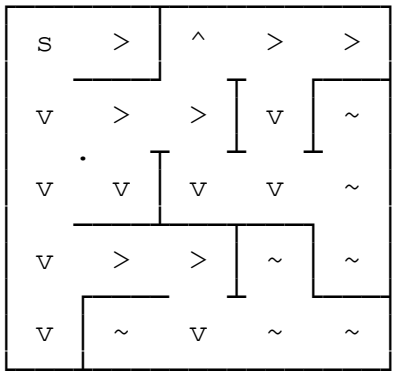
```
PriorQueue all items:
PriorQueue.top()=(4, (0, 0))
PriorQueue.top()=(4, (0, 1))
PriorQueue.top()=(4, (1, 0))
PriorQueue.top()=(4, (1, 1))
PriorQueue.top()=(4, (2, 0))
PriorQueue.top()=(7, (3, 0))
PriorQueue.top()=(7, (3, 1))
PriorQueue.top()=(11, (1, 2))
PriorQueue.top()=(11, (2, 1))
PriorQueue.top()=(13, (0, 2))
PriorQueue.top()=(14, (3, 2))
PriorQueue.top()=(14, (4, 0))
PriorQueue.top()=(15, (0, 3))
PriorQueue.top()=(15, (1, 3))
PriorQueue.top()=(17, (0, 4))
PriorQueue.top()=(18, (2, 2))

PathWeights % 1000:
```



Как видим, путевые веса аналогичны путевым весам в алгоритме Дейкстры, но приоритеты внутри очереди стали другими. Например, ячейка (4, 2) имела в очереди приоритет 18, т.е. такой же, как и у финишной ячейки (в вершину очереди эта ячейка не попала). При сравнении с алгоритмом Дейкстры можно увидеть, что при работе A* из очереди было извлечено число ячеек на один меньше, чем при работе алгоритма Дейкстры.

```
PathCoordinates:
```



Построение пути для A* аналогично построению пути в алгоритме Дейкстры.

```
The finish cell is reached!
Path=  (0, 0)  (1, 0)  (1, 1)  (1, 2)  (2, 2)
```

4. РЕЗУЛЬТАТЫ РАБОТЫ

4.1. Графики

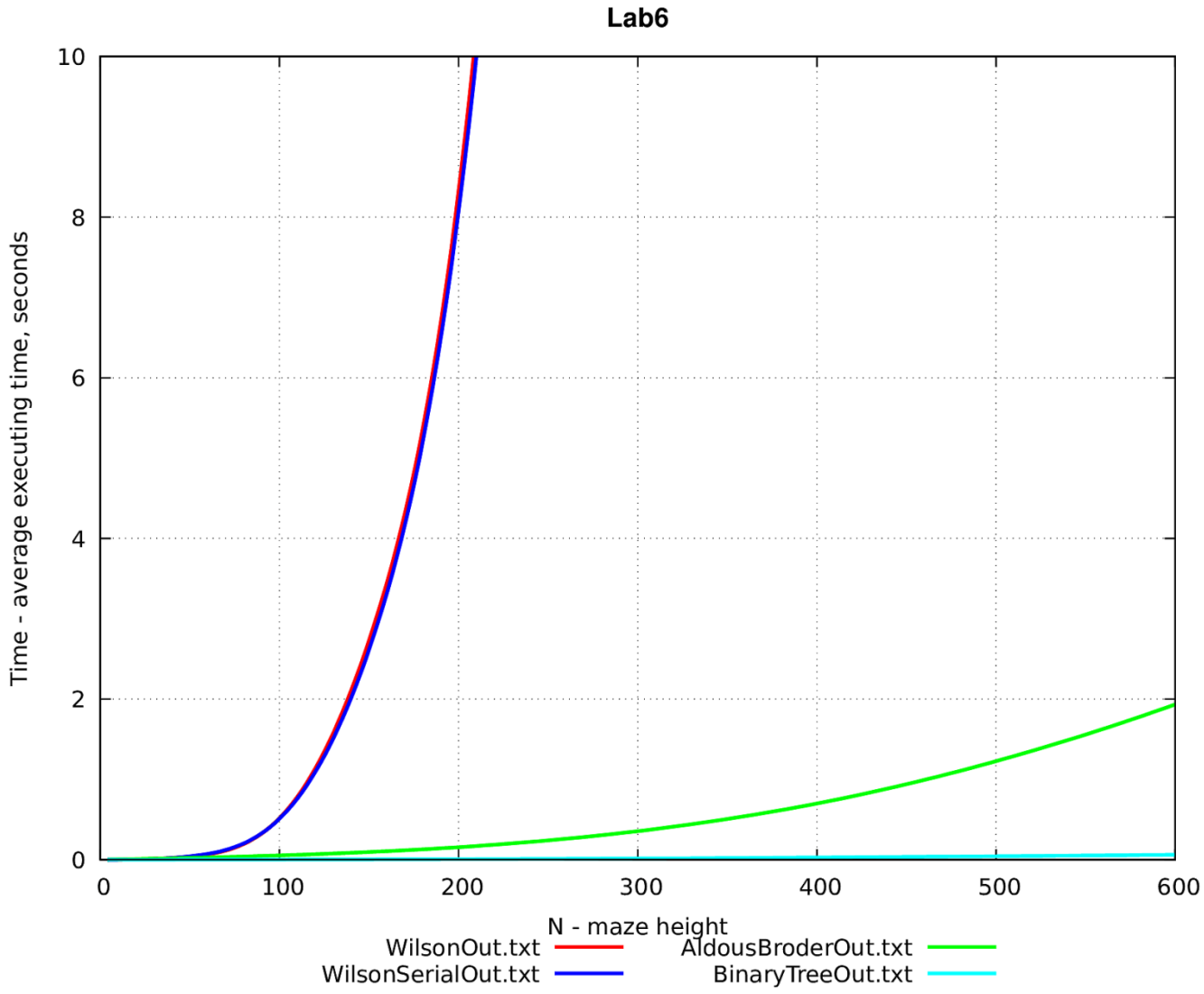


Рисунок 6 – Время генерации лабиринта от высоты лабиринта

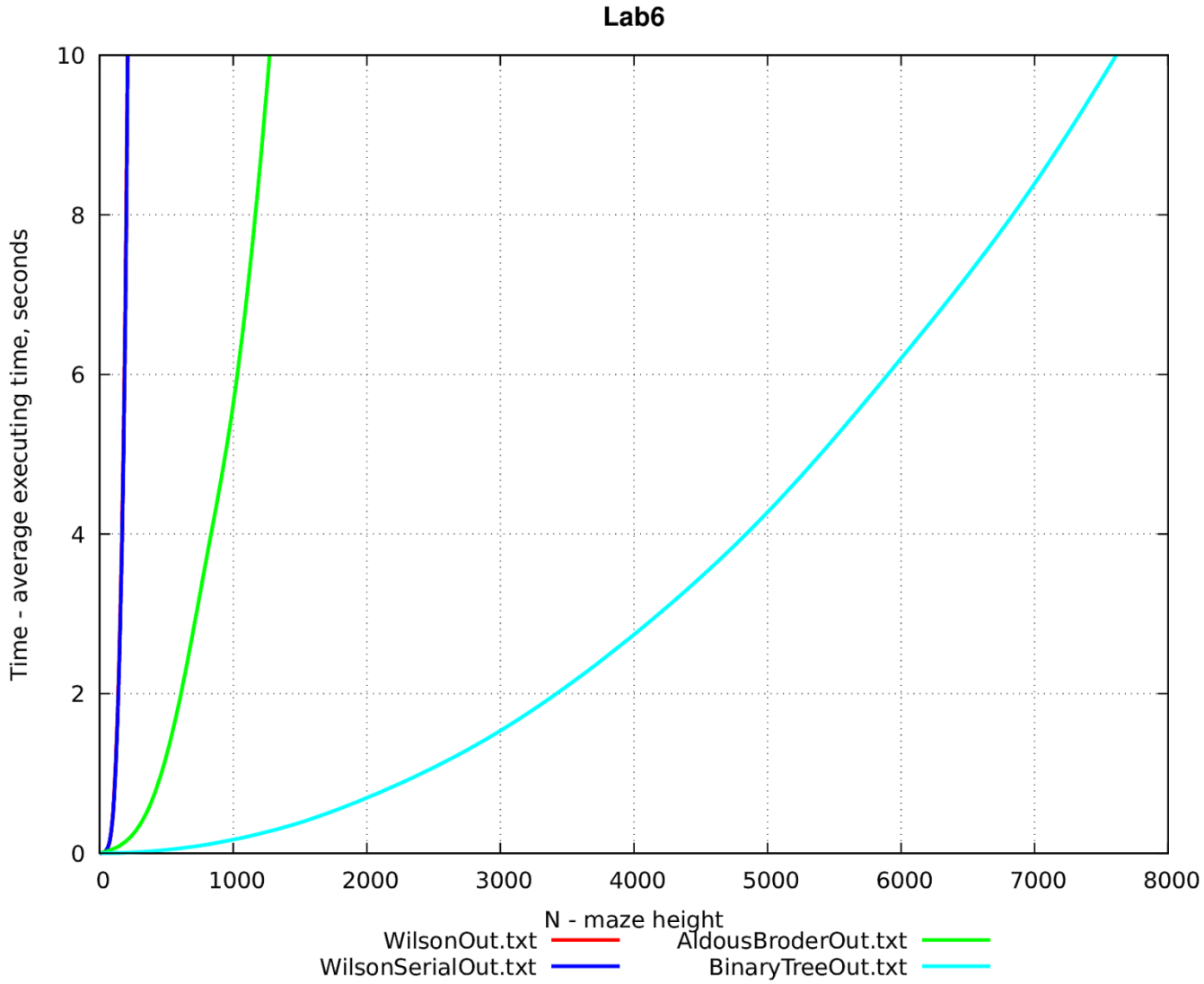


Рисунок 7 – Время генерации лабиринта от высоты лабиринта

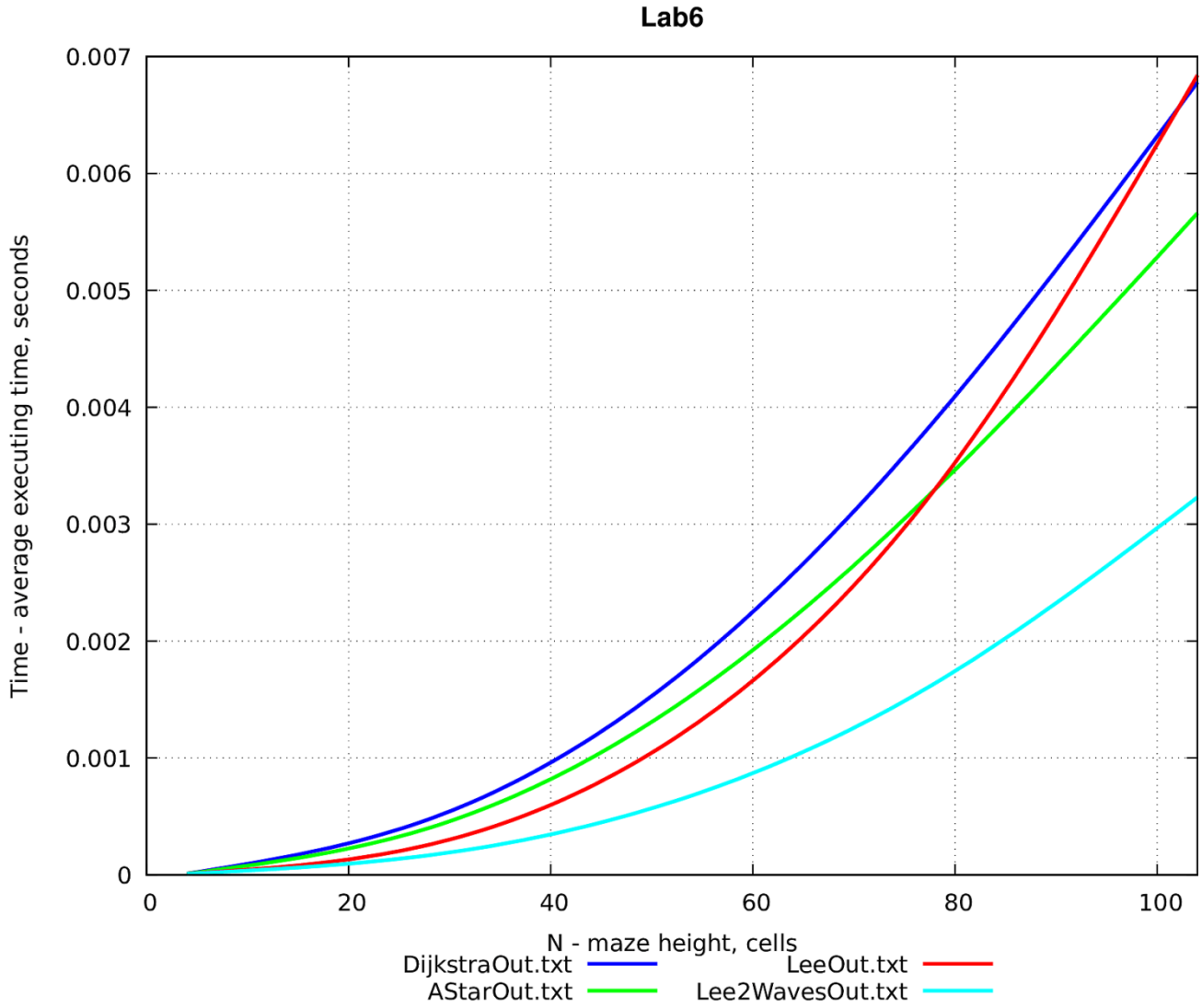


Рисунок 8 – Время поиска в лабиринте от высоты лабиринта

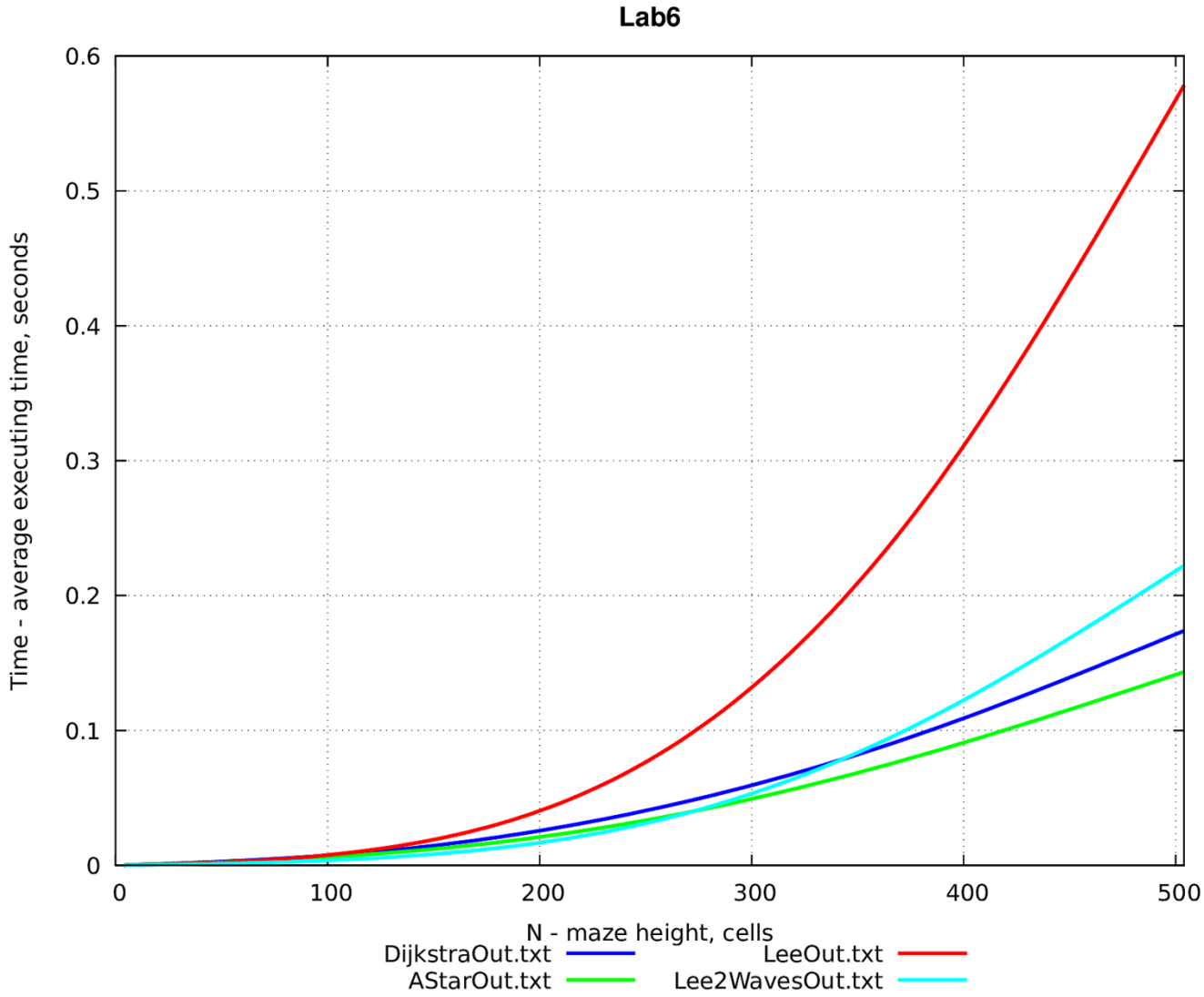


Рисунок 9 – Время поиска в лабиринте от высоты лабиринта

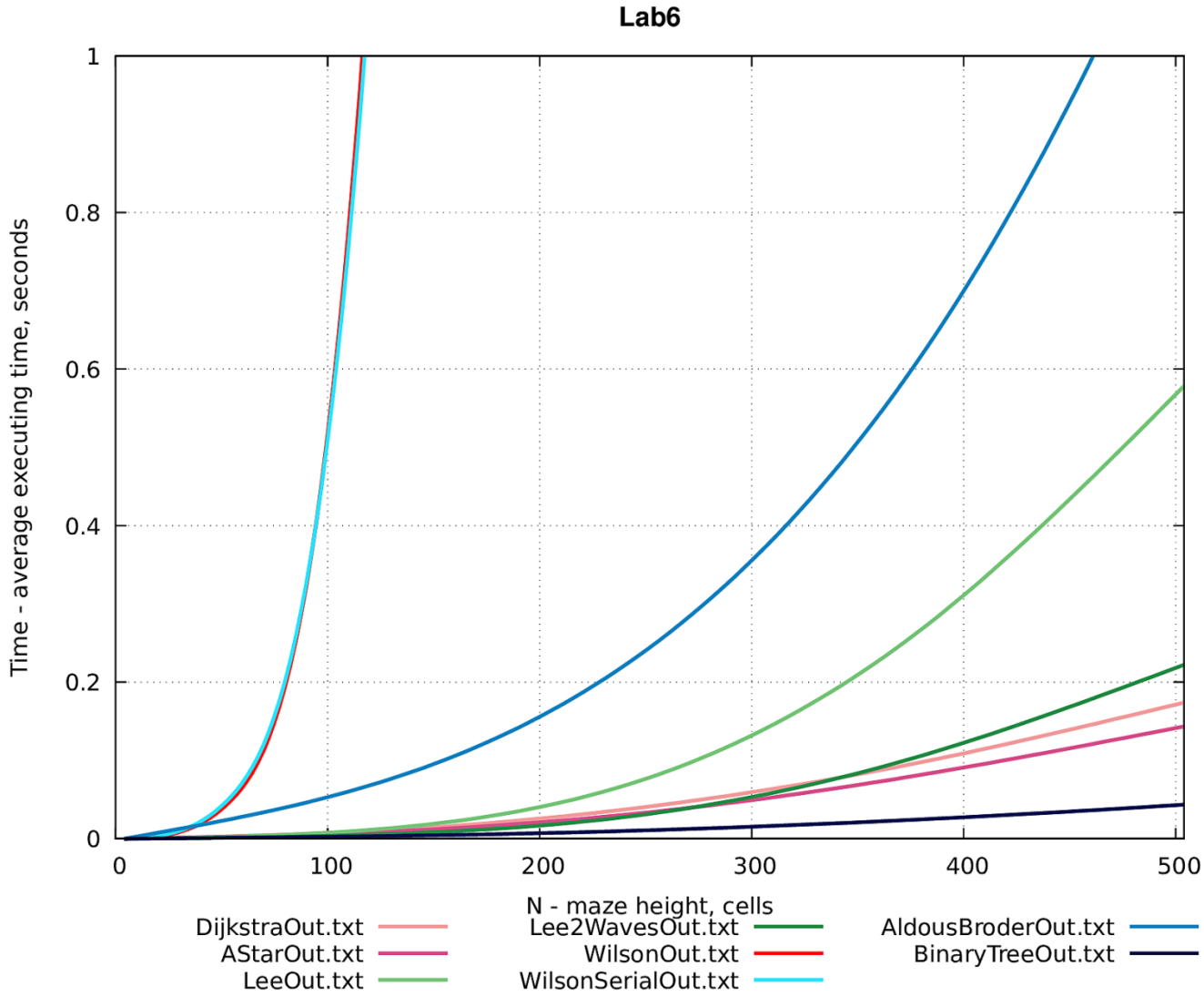


Рисунок 10 – Сравнение времени генерации и времени поиска в лабиринтах

4.2. Анализ графиков

4.2.1. Алгоритмы генерации лабиринтов

Как видно из рис 6 и 7, Алгоритм Уилсона оказался самым медленным из всех, сильно уступая даже алгоритму Олдоса-Бродера, который является более «глупым» (за 10 секунд алгоритм Олдоса-Бродера успевает сгенерировать примерно в 25 раз больше ячеек, чем алгоритм Уилсона). Это, по-видимому, связано с тем, что срезая петли, алгоритм многократно проходится по одним и тем же ячейкам, в отличие от Олдоса-Бродера, который добавляет в лабиринт все ячейки, на которые наткнется.

Его модификация с последовательным выбором ячеек, не входящих в UST, как оказалось, практически не влияет на время его работы.

Алгоритм Двоичного дерева оказался намного быстрее алгоритмов Уилсона (за 10 секунд он успевает сгенерировать примерно в 900 раз больше ячеек!) и Олдоса-Бродера(за 10 секунд в 36 раз больше ячеек). Но, как известно, лабиринты, сгенерированные им, имеют сильную смещенность и неоднородность, что заметно невооруженным взглядом.

4.2.2. Алгоритмы поиска в лабиринтах

Как видно из рис 8-9, Модификация Ли с двумя волнами работает примерно в 2 раза быстрее, чем вариант с одной волной. Причем при больших значениях лабиринта выигрыш во времени увеличивается.

Алгоритм A* быстрее алгоритма Дейкстры в 1,19 раз для высоты лабиринта в 104 ячейки и в 1,21 раз для высоты лабиринта в 504 ячейки.

Алгоритм Ли быстрее алгоритма A* на лабиринтах с высотой до 80 ячеек и быстрее алгоритма Дейкстры на лабиринтах с высотой до 100 ячеек. Причем при высоте в 504 ячейки алгоритм Ли становится медленнее в 3,32 раза.

Модификация Ли с двумя волнами оказывается быстрее алгоритма Дейкстры (приблизительно до 340 ячеек по высоте).

Это связано с отличиями реализации этих алгоритмов. Алгоритм Дейкстры использует очередь с приоритетами, работа с которой требует затрат времени. Очереди – обычный подход к реализации алгоритмов, основанных на поиске в ширину. В этой работе алгоритм Ли реализован без использования очередей. При каждой итерации распространения фронта волны ячейки фронта волны ищутся в прямоугольной области поля лабиринта. Эта область ограничена наиболее удаленными друг от друга ячейками фронта волны. Такой подход оказывается быстрее для относительно небольших лабиринтов (до 640 ячеек для Ли и A*) и значительно более медленным при больших лабиринтах. В прочем, для сравнения нужна реализация алгоритма Ли с очередью. Может оказаться, что очередь без приоритетов, которая нужна для реализации Ли окажется быстрее очереди с приоритетами настолько, что реализация без очередей не будет иметь преимуществ.

4.2.3. Сравнение алгоритмов генерации и поиска

Скорость генерации и поиска сопоставимы друг с другом. Из рассмотренных алгоритмов генерации быстрее алгоритмов поиска оказался только алгоритм генерации Двоичным деревом.

Так, алгоритм Олдоса-Бродера медленнее алгоритма Ли примерно в 2 раза для лабиринта с высотой 400 ячеек.

5. ВЫВОДЫ

1) В ходе поделанный работы были изучены алгоритмы Уилсона, Уилсона(модификация), Олдоса-Бродера, Бинарного дерева для генерации лабиринтов, алгоритмы Ли, Ли с двумя волнами, Дейкстры, A* для поиска пути в лабиринтах.

2) Был освоен Класс-шаблон адаптера контейнера priority_queue в стандартной библиотеке шаблонов (STL) языка C++, способы вывода данных с использованием псевдографики Unicode.

3) Были разработаны классы для хранения, доступа и изменения лабиринтов без весов ячеек и с весами ячеек. Реализован вывод лабиринтов в пригодном для редактирования вида, а так же в декоративном виде в двух вариантах: с мелкими и крупными ячейками. Были разработаны классы для сбора времени выполнения алгоритмов генерации и поиска в лабиринтах

4) Анализ показал, что алгоритмы поиска на лабиринтах по времени сопоставимы с алгоритмами генерации лабиринтов. Самым медленным является алгоритм Уилсона, самым быстрым – алгоритм Бинарного дерева. Использование эвристики уменьшает время поиска пути в лабиринте приблизительно в 1,2 раз.

6. ИСХОДНЫЙ КОД

6.1. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.0)
project(Lab6 VERSION 0.1.0)

#include(CTest)
#enable_testing()

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -pthread")
```

```
    #Разработка класса лабиринта
    add_executable(maze_debug_main maze_debug_main.cpp)
    add_executable(mazeWeighted_debug_main
mazeWeighted_debug_main.cpp)
    #Алг генерации лабиринтов
    add_executable(AldousBroder AldousBroder_main.cpp)
    add_executable(Wilson Wilson_main.cpp)
    add_executable(BinaryTree BinaryTree_main.cpp)
    #Сбор статистики с генерации лабиринтов
    add_executable(StatisticsMaze StatisticsMaze_main.cpp)

    #Алг поиска пути в лабиринте
    add_executable(Lee Lee_main.cpp)
    add_executable(Lee2Waves Lee2Waves_main.cpp)
    #Алг поиска пути в лабиринте, основанные на поиске на графах с
ребрами с различными весами.
    add_executable(Dijkstra Dijkstra_main.cpp)
    add_executable(AStar AStar_main.cpp)
    #Сбор статистики с генерации лабиринтов
    add_executable(StatisticsMazeSearch_main
StatisticsMazeSearch_main.cpp)

    install(PROGRAMS            PlotScript.bash            DESTINATION
${CMAKE_CURRENT_BINARY_DIR})

    set(CPACK_PROJECT_NAME ${PROJECT_NAME})
    set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
    include(CPack)
```

6.2. maze.h

```
    #ifndef MAZE_H
    #define MAZE_H

    #include <iostream>
    #include <vector>
    #include <fstream>
    #include <sstream>

    #include <type_traits> //Будем сравнивать типы в шаблонном методе
    #include <cassert>

    #define WALL '#' //Стена присутствует
    #define NOWALL '?' //Стена отсутствует (половина строк содержит
только информацию о стенах)
    #define EMPTYCELL '.' //Сама ячейка
    #define CORNER '+' //Не несет смысла. Стоит по углам ячеек.

    #define HORIZWALL "\u2500" // '-'
    #define VERTWALL "\u2502" // '|'

    #define CORTL "\u250C" // 'Г'
    #define CORTR "\u2510" // '~.'
```

```
    #define CORBL "\u2514" // 'L'
    #define CORBR "\u2518" // '_|'

    #define TOPWALL "\u252C" // '~.Г'
    #define BOTTOMWALL "\u2534" // '_L'
    #define LWALL "\u251C" // '|-'
    #define RWALL "\u2524" // '-|'

    #define FULLWALL "\u253C" // '-|-'
    #define DOT "." //для случаев, когда нет всех четырех стен
    #define NOHORIZ " "

    class maze
    {
    private:
    public:
        int n=0, m=0; //высота и ширина лабиринта
        std::vector<std::vector<char>> BaseVector;
        maze()=default;
        maze(int Nn, int Nm): n(Nn), m(Nm)
        {
            BaseVector.resize(n*2+1);
            for(auto& i: BaseVector) i.resize(m*2+1);

            for(int  i=0;  i<BaseVector.size();  i+=2)  //устанавливаем
углы ячеек
                for(int j=0; j<BaseVector.at(i).size(); j+=2)
                {
                    BaseVector.at(i).at(j)=CORNER;
                }
            for(int i=0; i<n; i++)
                for(int j=0; j<m; j++)
                {
                    SetCellValue(i, j, EMPTYCELL);
                    for(int alpha=0; alpha<4; alpha++)
                        SetCellWalls(i, j, alpha, true);
                }
        }
        void SetCellValue(int i, int j, char c)
        {
            BaseVector.at(i*2+1).at(j*2+1)=c;
        }
        void ResetValues()
        {
            for(int i=0; i<n; i++)
            {
                for(int j=0; j<m; j++)
                {
                    SetCellValue(i,j, EMPTYCELL);
                }
            }
        }
        char GetCellValue(int i, int j)
```

```
        {
            return BaseVector.at(i*2+1).at(j*2+1);
        }
        void SetValuesByStr(std::string str)
        {
            int k=0;
            for(int i=0; i<n && k<str.size(); i++)
            {
                for(int j=0; j<m && k<str.size(); j++)
                {
                    if (str.at(k)!=' ') SetCellValue(i,j, str.at(k));
                    k++;
                }
            }
        }
        void SetCellWalls(int i, int j, int alpha, bool HasWall, bool
Protected=false) //alpha - угол, под которым находится радиус вектор,
указывающий на данную стену из центра ячейки. (как в тригонометрии).
Значения от 0 до 3. Protected защищать ли стены области. true - не
позволять убирать стенки области лабиринта.
        {
            int     di=(alpha%2)*((alpha%4)*(alpha%2)-2);     //формулы
получены из графиков
            int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);
            if(Protected)
            {
                if((i+di<n)&&(i+di>=0)&&(j+dj<m)&&(j+dj>=0))     //если
смежная по этой стене ячейка находится внутри области лабиринта
                {
                    (HasWall)?
BaseVector.at(i*2+1+di).at(j*2+1+dj)=WALL                              :
BaseVector.at(i*2+1+di).at(j*2+1+dj)=NOWALL;
                }
            }
            else
            {
                (HasWall)? BaseVector.at(i*2+1+di).at(j*2+1+dj)=WALL :
BaseVector.at(i*2+1+di).at(j*2+1+dj)=NOWALL;
            }
        }
        //Если будем обращаться к несуществующим стенам (из
несуществующих ячеек), то будем возвращать false, т.е. что такой стены
нет. Это позволит пробегать все стены, итерируясь по несуществующим
ячейкам
        bool HasWall(int i, int j, int alpha) //прикол в том, что мы
можем узнать, существуют ли стены у ячейки, даже если самой ячейки не
существует. Главное, чтобы сами стены существовали в лабиринте.
        {
            int     di=(alpha%2)*((alpha%4)*(alpha%2)-2);     //формулы
получены из графиков
            int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);
            if((i*2+1+di>=0)  &&  (i*2+1+di<n*2+1)  &&  (j*2+1+dj>=0)  &&
(j*2+1+dj<m*2+1))     //Лишние     условные     операторы     программе
```

производительности не прибавляют, но так будет легче выводит лабиринт в графическом виде. Другой способ - создание лишних ячеек по углам

```
            {
                if(BaseVector.at(i*2+1+di).at(j*2+1+dj)==WALL)   return
true;
                else return false;
            }
            else
            {
                return false;
            }
        }
        int GetNeighborI(int i, int alpha)
        {
            int     di=(alpha%2)*((alpha%4)*(alpha%2)-2);     //формулы
получены из графиков
            return i+di;
        }
        int GetNeighborJ(int j, int alpha)
        {
            int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);
            return j+dj;
        }
        std::string Show(char* filename=(char*)"cin")
        {
            std::stringstream ss;
            for(auto& i: BaseVector)
            {
                for(auto& j: i)
                {
                    ss<<j;
                }
                ss<<std::endl;
            }
            std::cout<<ss.str();
            return ss.str();
        }
        void  CharFromWallFlags(std::stringstream&  ss,  std::string&
WallFlags) //Вспомогательная функция для получения символа по флагу,
т.е. по его описанию.
        {
            switch (std::stoi(WallFlags))
            {
            case 0:
                ss<<DOT;
                break;
            case 1:
                ss<<HORIZWALL;
                break;
            case 10:
                ss<<BOTTOMWALL;
                break;
            case 11:
```

```
                ss<<CORBL;
              //  std::cout<<"char:CORBL"<<std::endl;
                break;
            case 100:
                ss<<HORIZWALL;
                break;
            case 101:
                ss<<HORIZWALL;
                break;
            case 110:
                ss<<CORBR;
                break;
            case 111:
                ss<<BOTTOMWALL;
                break;
            case 1000:
                ss<<TOPWALL; //если ставить вертикальную черту, то
может возникнуть ситуация, когда закроется проход
                break;
            case 1001:
                ss<<CORTL;
                break;
            case 1010:
                ss<<VERTWALL;
                break;
            case 1011:
                ss<<LWALL;
                break;
            case 1100:
                ss<<CORTR;
                break;
            case 1101:
                ss<<TOPWALL;
                break;
            case 1110:
                ss<<RWALL;
                break;
            case 1111:
                ss<<FULLWALL;
                break;
            default:
                break;
            }
        }
         //Сейчас попробуем переделать это, чтобы scale работал.
        void ShowDecorateOld(char* filename=(char*)"cout", int Mode=0,
int Scale=1, bool IsWithValues=false) //Mode - 0 выводить внутреннее
представление лабиринта и его декоративный вариант. 1 - только
декоративный. Scale 1 и 2 возможные масштабы лабиринта. bool
IsWithValues Для масштаба 2 можно вывести значения в ячейках
        {
            std::stringstream ss;
            std::string str;
```

```
        //
ss<<HORIZWALL<<VERTWALL<<CORTL<<CORTW<<CORBL<<CORBR<<TOPWALL<<BOTTOMWA
LL<<LWALL<<RWALL<<FULLWALL<<    std::endl;    //проверка    отображения
символов.
        //    std::cout<<ss.str();
            std::string WallFlags;

            WallFlags.clear();
            // Тут просто корректируем граничные условия
            if(HasWall(0,0,    1))    WallFlags.insert(0,"1");    else
WallFlags.insert(0,"0");
            WallFlags.insert(0,"0"); // наверх стены не продолжаются
            WallFlags.insert(0,"0");
            if(HasWall(0,0,    2))    WallFlags.insert(0,"1");    else
WallFlags.insert(0,"0");
            CharFromWallFlags(ss, WallFlags);


            for(int j=0;j<m-1;j++)
            {
                if(HasWall(0,0, 1)) ss<<HORIZWALL; else ss<<NOHORIZ;
                WallFlags.clear();
                if(HasWall(0,j+1,  1))  WallFlags.insert(0,"1");  else
WallFlags.insert(0,"0");
                WallFlags.insert(0,"0");    //    наверх    стены    не
продолжаются
                if(HasWall(0,j,   1))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
                if(HasWall(0,j,   0))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
                CharFromWallFlags(ss, WallFlags);
            }

            if(HasWall(0,0, 1)) ss<<HORIZWALL; else ss<<NOHORIZ;
            WallFlags.clear();
            WallFlags.insert(0,"0");  //  вправо  и  наверх  стены  не
продолжаются
            WallFlags.insert(0,"0");
            if(HasWall(0,m-1,    1))    WallFlags.insert(0,"1");    else
WallFlags.insert(0,"0");
            if(HasWall(0,m-1,    0))    WallFlags.insert(0,"1");    else
WallFlags.insert(0,"0");
            CharFromWallFlags(ss, WallFlags);

            ss<<std::endl;
            // /////
            for(int i=0; i<n; i++)
            {
                // Тут просто корректируем граничные условия
                WallFlags.clear();
                if(HasWall(i,0,   3))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
```

```
                if(HasWall(i,0,   2))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
                WallFlags.insert(0,"0"); //слева ячеек нет, и значит
стен, уходящих влево тоже нет
                if(i+1<n)             if(HasWall(i+1,0,             2))
WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");
                CharFromWallFlags(ss, WallFlags);
              //   std::cout<<"i="<<i<<" WallFlags="<<WallFlags<<" "<<
std::stoi(WallFlags)<<"         HasWall(i,0,         2)="<<HasWall(i,0,
2)<<CORBL<<std::endl;
                // /////
                for(int j=0; j<m-1; j++)
                {
                    if(HasWall(i,j,     3))     ss<<HORIZWALL;     else
ss<<NOHORIZ;
                    WallFlags.clear();
                    if(HasWall(i+1,j+1,   1))   WallFlags.insert(0,"1");
else WallFlags.insert(0,"0");
                    if(HasWall(i,j,  0))  WallFlags.insert(0,"1");  else
WallFlags.insert(0,"0");
                    if(HasWall(i,j,  3))  WallFlags.insert(0,"1");  else
WallFlags.insert(0,"0");
                    if(i+1<n)           if(HasWall(i+1,j+1,          2))
WallFlags.insert(0,"1");     else     WallFlags.insert(0,"0");     else
WallFlags.insert(0,"0");

                    CharFromWallFlags(ss, WallFlags);

                }
                // Тут просто корректируем граничные условия
                if(HasWall(i,m-1, 3)) ss<<HORIZWALL; else ss<<NOHORIZ;
                WallFlags.clear();
                WallFlags.insert(0,"0"); //справа ячеек нет, и значит
стен, уходящих вправо тоже нет
                if(HasWall(i,m-1,   0))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
                if(HasWall(i,m-1,   3))   WallFlags.insert(0,"1");   else
WallFlags.insert(0,"0");
                if(i+1<n)             if(HasWall(i+1,m-1,             0))
WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");
                CharFromWallFlags(ss, WallFlags);
                // /////
                ss<<std::endl;
            }
            if(filename==(char*)"cout")  //пошел вывод в терминал или
файл
            {
                switch(Mode)
                {
                    case 0:
                    {
                        std::stringstream ss1;
                        for(auto& i: BaseVector)
```

```
                {
                    for(auto& j: i)
                    {
                        ss1<<j;
                    }
                    ss1<<std::endl;
                }
                std::cout<<ss1.str();
                std::cout<<"------------"<<std::endl;
                std::cout<<ss.str();
            }break;
            case 1:
            {
                std::cout<<ss.str();
            }break;
        }
    }
    else
    {
        std::ofstream fout(filename);
        if(!fout.is_open())
        {
            std::stringstream ss;
            ss << "Can not open file:"<<filename<<std::endl;
            throw(ss.str());
        }
        switch(Mode)
        {
            case 0:
            {
                std::stringstream ss1;
                for(auto& i: BaseVector)
                {
                    for(auto& j: i)
                    {
                        ss1<<j;
                    }
                    ss1<<std::endl;
                }
                fout<<ss1.str();
                fout<<"------------"<<std::endl;
                fout<<ss.str();
            }break;
            case 1:
            {
                fout<<ss.str();
            }break;

        }
        fout.close();

    }
}
```

```
        char GetCellValueFromStruct(int i, int j, const std::string&
str, int member) //перегрузки.
        {
            //
            if(str==std::string("GetCellValue"))
                return GetCellValue(i, j);
            assert(false);
            return -1;
        }
        int       GetCellValueFromStruct(int       i,       int       j,
std::vector<std::vector<int>>& Weights, int member)
        {
            return Weights.at(i).at(j);
        }
        int       GetCellValueFromStruct(int       i,       int       j,
std::vector<std::vector<std::pair<char,  int>>>  PathCoordWeights,  int
member) //member - какой член выбрать в PathCoordWeights
        {
            switch (member)
            {
            case 1:
                return PathCoordWeights.at(i).at(j).first;
                break;
            case 2:
                return PathCoordWeights.at(i).at(j).second % 1000;
                break;
            default:
                break;
            }
            assert(false);
            return -1;
        }
        template<typename T>
        std::string StrFromInt(T value)
        {
            std::stringstream ss1, ss2;
            ss1<<value;
            switch (ss1.str().length())
            {
            case 1:
                ss2<<NOHORIZ<<ss1.str()<<NOHORIZ;
                break;
            case 2:
                ss2<<ss1.str()<<NOHORIZ;
                break;
            case 3:
                ss2<<ss1.str();
                break;
            default:
                ss2<<NOHORIZ<<'?'<<NOHORIZ;
                break;
```

```
            }
            return ss2.str();
        }
        template <typename T=const std::string> //константные ссылки могут быть привязаны к временным объектам, причем время жизни объекта расширяется до времени жизни константной ссылки
        void ShowDecorate(char* filename=(char*)"cout", int Mode=0, int Scale=1, bool IsWithValues=false, T& Values=std::string("GetCellValue"), int member=0) //Mode - 0 выводить внутреннее представление лабиринта и его декоративный вариант. 1 - только декоративный. Scale 1 и 2 возможные масштабы лабиринта. bool IsWithValues Для масштаба 2 можно вывести значения в ячейках
        {
            std::stringstream ss;
            std::string str;
        //
ss<<HORIZWALL<<VERTWALL<<CORTL<<CORTW<<CORBL<<CORBR<<TOPWALL<<BOTTOMWALL<<LWALL<<RWALL<<FULLWALL<< std::endl; //проверка отображения символов.
        //    std::cout<<ss.str();
            std::string WallFlags;

            for(int i=0; i<n+1; i++)
            {

                for(int j=0; j<m+1; j++)
                {
                    //рисуем левый вехний угл ячейки
                    WallFlags.clear();
                    if(HasWall(i,j, 1)) WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");
                    if(HasWall(i-1,j-1, 0)) WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");
                    if(HasWall(i-1,j-1, 3)) WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");
                    if(HasWall(i,j, 2)) WallFlags.insert(0,"1"); else WallFlags.insert(0,"0");

                    if(Scale==1)
                    {
                        CharFromWallFlags(ss, WallFlags);
                        //рисуем верхнюю стену ячейки
                        if(HasWall(i,j, 1)) ss<<HORIZWALL; else ss<<NOHORIZ;
                    }
                    else if(Scale==2)
                    {
                        CharFromWallFlags(ss, WallFlags);
                        //рисуем верхнюю стену ячейки
                        if(HasWall(i,j, 1)) ss<<HORIZWALL<<HORIZWALL<<HORIZWALL; else ss<<NOHORIZ<<NOHORIZ<<NOHORIZ;
                    }
```

```
                }
                ss<<std::endl;
                if(Scale==2 && i!=n)
                {
                    //рисуем левые стенки
                    for(int j=0; j<m; j++)
                    {
                        if(HasWall(i,j,    2))    ss<<VERTWALL;    else
ss<<NOHORIZ;
                    //                                    if(IsWithValues)
ss<<NOHORIZ<<GetCellValue(i, j)<<NOHORIZ;
                    if(IsWithValues)
                    {
                        ss<<StrFromInt(GetCellValueFromStruct(i,j,
Values, member));
                    //          std::cout<<"GetCellValueFromStruct(i,j,
Values)="<<GetCellValueFromStruct(i,j, Values)<<std::endl;
                    }
                    else ss<<NOHORIZ<<NOHORIZ<<NOHORIZ;
                    }
                    //Граничные условия
                    if(HasWall(i,    m,    2))    ss<<VERTWALL;    else
ss<<NOHORIZ;
                    ss<<std::endl;
                }
            }

            if(filename==(char*)"cout")  //пошел  вывод  в  терминал  или
файл
            {
                switch(Mode)
                {
                    case 0:
                    {
                        std::stringstream ss1;
                        for(auto& i: BaseVector)
                        {
                            for(auto& j: i)
                            {
                                ss1<<j;
                            }
                            ss1<<std::endl;
                        }
                        std::cout<<ss1.str();
                        std::cout<<"------------"<<std::endl;
                        std::cout<<ss.str();
                    }break;
                    case 1:
                    {
                        std::cout<<ss.str();
                    }break;
                }
            }
```

```
            else
            {
                std::ofstream fout(filename);
                if(!fout.is_open())
                {
                    std::stringstream ss;
                    ss << "Can not open file:"<<filename<<std::endl;
                    throw(ss.str());
                }
                switch(Mode)
                {
                    case 0:
                    {
                        std::stringstream ss1;
                        for(auto& i: BaseVector)
                        {
                            for(auto& j: i)
                            {
                                ss1<<j;
                            }
                            ss1<<std::endl;
                        }
                        fout<<ss1.str();
                        fout<<"------------"<<std::endl;
                        fout<<ss.str();
                    }break;
                    case 1:
                    {
                        fout<<ss.str();
                    }break;

                }
                fout.close();

            }
        }
    };

    class mazeWeighted : public maze
    {
    public:
        std::vector<std::vector<int>> Weights;
        mazeWeighted()=default;
        mazeWeighted(int Nn, int Nm, int FillWeight=0): maze(Nn, Nm)
//FillWeight заливка всего поля этим весом
        {
            Weights.resize(Nn);
            for(auto& i : Weights)
            {
                i.resize(Nm,   FillWeight);   //Должны быть нулевые
значения (если без второго аргумента). Это связано с реализацией
вектора.
            }
```

```
        }
        void WeightsToValues() //Копирует веса в значения ячеек для
удобного представления.
        {
            for(int i=0;i<Weights.size(); i++)
            {
                for (int j=0;j<Weights.at(i).size(); j++)
                {
                    SetCellValue(i, j, Weights.at(i).at(j)+48);
                }
            }
        }
    };


    #endif /* MAZE_H */
```

6.3. MazeGenerationAlgs.h

```
    #ifndef MAZEGENERATIONALGS_H
    #define MAZEGENERATIONALGS_H

    #include "maze.h"
    #include "random"
    #include "presenthandler.h"

    #define VISITED '1' //Для AldousBroder

    #define UST '1' //Для Wilson
    #define ARROWUP '^'
    #define ARROWDOWN 'v'
    #define ARROWLEFT '<'
    #define ARROWRIGHT '>'

    //PresentHandler.Mode 0-не влиять; 1-выводить лабиринт в файл
каждые 30 секунд; 2-пошаговый вывод лабиринта

    void AldousBroder(maze& Maze, std::default_random_engine&
generator, presenthandler& PrHandler)
    {
        std::uniform_int_distribution<int> DistrI(0, Maze.n-1);
        std::uniform_int_distribution<int> DistrJ(0, Maze.m-1);
        std::uniform_int_distribution<int> Distrdidj(0,3);
        int randI=DistrI(generator);
        int randJ=DistrJ(generator);

        Maze.SetCellValue(randI, randJ, VISITED);
        int k=1;
        int di=0, dj=0;
        bool IsSatisfying=true; //когда у стены некотрые случайные
значения не подходят
        int randcase=0;
        int t1=time(0);
        int t2=time(0);
```

```
while(k!=Maze.n*Maze.m)
{
    //Maze.ShowDecorate((char*)"cout", 1);
    if(PrHandler.Mode==1)
    {
    if(t2-t1>=30)
    {
        Maze.ShowDecorate((char*)"MazeOut.txt", 1);
        t1=t2;
    }
    }
    if(PrHandler.Mode==2)
    {
        Maze.ShowDecorate((char*)"cout", 1, 2, true);
    }
    randcase=Distrdidj(generator);
    switch (randcase)
    {
    case 0:
        if(randJ<Maze.m-1)
        {
            di=0;
            dj=1;
            IsSatisfying=true;
        }
        else IsSatisfying=false;
        break;
    case 1:
        if(randI>0)
        {
            di=-1;
            dj=0;
            IsSatisfying=true;
        }
        else IsSatisfying=false;
        break;
    case 2:
        if(randJ>0)
        {
            di=0;
            dj=-1;
            IsSatisfying=true;
        }
        else IsSatisfying=false;
        break;
    case 3:
        if(randI<Maze.n-1)
        {
            di=1;
            dj=0;
            IsSatisfying=true;
        }
        else IsSatisfying=false;
```

```
                break;

            default:
                break;
            }
            if(IsSatisfying) //если не уперлись в стену
            {
                if(Maze.GetCellValue(randI+di, randJ+dj)!=VISITED)
                {
                    Maze.SetCellWalls(randI, randJ, randcase, false);
                    randI+=di;
                    randJ+=dj;
                    Maze.SetCellValue(randI, randJ, VISITED);
                    k++;
                }
                else
                {
                    randI+=di;
                    randJ+=dj;
                }
            }
            t2=time(0);
        }
    }
```

//Реализацию нашел только на питоне. На си++ нету. Поэтому придется изобретать все велосипеды самостаятельно.

//Итак, доп память нам нужно только для возможности случайного выбора ячейки из ещё невыбранных, поскольку мы должны выбрать диапазон, в котором генерируется случайное число

//И как-то определять какая ячейка соответствует какому числу

//UPD сначала для этих целей я думал взять вектор или map (решил, что map более мне подходит, чем вектор, т.к. тормоза будут только при выборе случайной ячейки, но не при извлечении из него ячеек, как у вектора)

//Но сейчас я понял ---что все идет по плану--- что можно не использовать дополнительную память: Максимальное генерируемое число будет определятся счетчиком оставшихся ячеек(изначально n*m , далее каждая пометка ячейки о пройденности в лабиринте сопровождается уменьшением счетчика на единицу)

//А выбор ячейки будем осуществлять так: будем последовательно пробегать строки и считать число встретившихся ячеек, которые мы ещё не посетили. Когда это число станет равным нашему сгенерированному числу мы и попадем в соответствующую этому числу ячейку.

//Таким образом, мой алгоритм не требует дополнительной памяти! Кроме той, которая нужна для хранения самого лабиринта.

//В методичке написано, что он потребляеть память O(N^2). Я думаю, что речь идет о хранении информации о направлении стрелочек в ячейке. У меня на информацию о ячейке отведен отдельный символ.

//Причем каждая стена кодируется отдельным символом, что позволяет хранить 256 состояний ячейки. Не считая информации о стенах, её окружающих.

```
    //На самом деле, для хранения информации о ячейке достаточно всего одного символа. Ведь всего может быть 16 ситуаций со стенами для кажной ячейки.
    //Т.е. если мы храним лабиринт в виде символов, где каждым символом закодирована ячейка, то мы помимо информации о стенах можем кодировать этими символом и состояние самой ячейки.
    //более того, в каждой ячейке мы можем хранить лишь информацию о половине стен. (тут сложность будет в хранении информации о стенах по краям лабиринта. Можно ввести дополнительную строку и дополнительный столбец. В этом случае мы все равно экономим память. (на больших лабиринтах экономия до двух раз))
    //Мы же используем избыточное число символов для наглядности. Так мы легко можем редактировать лабиринт вручную, просматривать его состояние.
    //Если бы было 1 ячейка - 1 символ, то это было бы намного сложнее.
    //В общем, с реальным расоходом алгоритмом памяти в O(N^2) мы можем встретиться только в экзотических ситуациях: при жесткой экономии памяти для ячеек(например, по 2(не по 4, а по 2) бита на ячейку. Либо же структура данных, отведенная под лабиринт содержит в себе ещё сторонюю информацию). Или при экзотической геометрии ячеек.
    //В методичке они очищают информацию о стрелках после каждого блуждания. Мы этого не делаем, поскольку это ни на что не влияет. (единственное, так было бы проще отлаживать: лишние стрелки будут сбивать с толку)
    void Wilson(maze& Maze, std::default_random_engine& generator, presenthandler& PrHandler)
    {
        std::uniform_int_distribution<int> InitDistrI(0, Maze.n-1);
        std::uniform_int_distribution<int> InitDistrJ(0,Maze.m-1);
        int InitrandI=InitDistrI(generator); //выбираем первую случайную ячейку.
        int InitrandJ=InitDistrJ(generator);

        int t1=time(0);
        int t2=time(0);

        Maze.SetCellValue(InitrandI, InitrandJ, UST);
        int USTCount=1; //число ячеек в UST
        while(USTCount != Maze.n*Maze.m) //Каждая итерация включает в UST новую ветвь.
        {
            //Наглядное отображение
            if(PrHandler.Mode==2)
            {
                Maze.ShowDecorate();
                std::cin.ignore();
            }
            //Случайным образом выбираем ячейку, которой ещё нет в UST
            std::uniform_int_distribution<int> DistrIter(0, Maze.n*Maze.m-1-USTCount); //Maze.n*Maze.m-1 номер последей ячейки во всем лабиринте при счете с нуля.
```

```
            int  randIter=DistrIter(generator);  //выбираем  случайный
номер ячейки.
            int k=0; //счетчик ячеек, которых пока нет в UST
            int randI=0, randJ=0; //координаты искомой ячейки
            bool IsReached=false;
            for(int i=0; (i<Maze.n)&& (!IsReached); i++)
            {
                for(int j=0; (j<Maze.m)&& (!IsReached); j++)
                {
                    if(Maze.GetCellValue(i,  j)!=UST)  //Тут  позволим
небольшой изврат, чтобы k не инициировать отрицательным числом.
                    {
                        if(randIter==k)
                        {
                            randI=i;
                            randJ=j;
                            IsReached=true;
                        }
                        k++; //увеличиваем потом, а не в начале.
                    }
                }
            }
            //Теперь блуждаем от этой ячейки, пока не уткнемся в UST
            std::uniform_int_distribution<int> Distrdidj(0,3);
            int di=0, dj=0;
            bool IsSatisfying=true; //когда у стены некотрые случайные
значения не подходят
            int randcase=0;
            int WanderI, WanderJ; //Текущие i j
            WanderI=randI;
            WanderJ=randJ;
            while(Maze.GetCellValue(WanderI,  WanderJ)!=UST)  //Пока  не
дойдем до области UST
            {
                //Наглядное отображение
                if(PrHandler.Mode==1)
                {
                t2=time(0);
                if(t2-t1>=30)
                {
                    Maze.ShowDecorate((char*)"MazeOut.txt", 1);
                    std::cout<<"presenthandler:    file    has    been
rewrited"<<std::endl;
                    t1=t2;
                }
                }
                randcase=Distrdidj(generator);
                switch  (randcase)  //выбираем  направление  движения  с
учетом внешних стен лабиринта
                {
                case 0:
                    if(WanderJ<Maze.m-1)
                    {
```

```
                    di=0;
                    dj=1;
                    IsSatisfying=true;
                }
                else IsSatisfying=false;
                break;
            case 1:
                if(WanderI>0)
                {
                    di=-1;
                    dj=0;
                    IsSatisfying=true;
                }
                else IsSatisfying=false;
                break;
            case 2:
                if(WanderJ>0)
                {
                    di=0;
                    dj=-1;
                    IsSatisfying=true;
                }
                else IsSatisfying=false;
                break;
            case 3:
                if(WanderI<Maze.n-1)
                {
                    di=1;
                    dj=0;
                    IsSatisfying=true;
                }
                else IsSatisfying=false;
                break;

            default:
                break;
            }
            if(IsSatisfying) //если не уперлись в стену, то ставим
стрелку и переходим в соответствии с di dj
            {
                switch (randcase)
                {
                case 0:
                    Maze.SetCellValue(WanderI,             WanderJ,
ARROWRIGHT);
                    break;
                case 1:
                    Maze.SetCellValue(WanderI, WanderJ, ARROWUP);
                    break;
                case 2:
                    Maze.SetCellValue(WanderI,             WanderJ,
ARROWLEFT);
                    break;
```

```
                    case 3:
                        Maze.SetCellValue(WanderI,              WanderJ,
ARROWDOWN);
                        break;
                    default:
                        break;
                    }
                    WanderI+=di;
                    WanderJ+=dj;
                }
            }
            //Теперь возвращаемся к ячейке, с которой начали блуждания
и строим уже окончательную ветвь по стрелкам.
            WanderI=randI;
            WanderJ=randJ;
            while(Maze.GetCellValue(WanderI, WanderJ)!=UST) //Пока не
дойдем до области UST
            {
                switch (Maze.GetCellValue(WanderI, WanderJ)) //Убираем
стрелку на символ UST и переходим в ячейку по стрелке. Увеличиваем
счетчик ячеек в UST.
                {
                case ARROWRIGHT:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 0, false);
                    USTCount++;
                    WanderJ++;
                    break;
                case ARROWUP:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 1, false);
                    USTCount++;
                    WanderI--;
                    break;
                case ARROWLEFT:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 2, false);
                    USTCount++;
                    WanderJ--;
                    break;
                case ARROWDOWN:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 3, false);
                    USTCount++;
                    WanderI++;
                    break;
                }
            }
        }

    }
```

```
    void    WilsonSerial(maze&    Maze,    std::default_random_engine&
generator, presenthandler& PrHandler) //более простая модификация, где
выбирается первая неотмеченная ячейка вместо случайной.
    {
        std::uniform_int_distribution<int> InitDistrI(0, Maze.n-1);
        std::uniform_int_distribution<int> InitDistrJ(0,Maze.m-1);
        int    InitrandI=InitDistrI(generator);    //выбираем    первую
случайную ячейку.
        int InitrandJ=InitDistrJ(generator);

        int t1=time(0);
        int t2=time(0);

        Maze.SetCellValue(InitrandI, InitrandJ, UST);
        int USTCount=1; //число ячеек в UST

        while(USTCount != Maze.n*Maze.m) //Каждая итерация включает в
UST новую ветвь.
        {
            //Наглядное отображение
            if(PrHandler.Mode==2)
            {
                Maze.ShowDecorate();
                std::cin.ignore();
            }
            //Выбираем первую попавшуюся ячейку, которой ещё нет в UST
            int randI=0, randJ=0; //координаты искомой ячейки
            bool IsReached=false;
            for(int i=0; (i<Maze.n)&&(!IsReached); i++)
            {
                for(int j=0; (j<Maze.m)&&(!IsReached); j++)
                {
                    if(Maze.GetCellValue(i, j)!=UST)
                    {
                        randI=i;
                        randJ=j;
                        IsReached=true;
                    }

                }
            }
            //Теперь блуждаем от этой ячейки, пока не уткнемся в UST
            std::uniform_int_distribution<int> Distrdidj(0,3);
            int di=0, dj=0;
            bool IsSatisfying=true; //когда у стены некотрые случайные
значения не подходят
            int randcase=0;
            int WanderI, WanderJ; //Текущие i j
            WanderI=randI;
            WanderJ=randJ;
            while(Maze.GetCellValue(WanderI,  WanderJ)!=UST)  //Пока  не
дойдем до области UST
            {
```

```
                //Наглядное отображение
                if(PrHandler.Mode==1)
                {
                t2=time(0);
                if(t2-t1>=30)
                {
                    Maze.ShowDecorate((char*)"MazeOut.txt", 1);
                    std::cout<<"presenthandler:    file    has    been
rewrited"<<std::endl;
                    t1=t2;
                }
                }
                randcase=Distrdidj(generator);
                switch  (randcase)  //выбираем направление движения с
учетом внешних стен лабиринта
                {
                case 0:
                    if(WanderJ<Maze.m-1)
                    {
                        di=0;
                        dj=1;
                        IsSatisfying=true;
                    }
                    else IsSatisfying=false;
                    break;
                case 1:
                    if(WanderI>0)
                    {
                        di=-1;
                        dj=0;
                        IsSatisfying=true;
                    }
                    else IsSatisfying=false;
                    break;
                case 2:
                    if(WanderJ>0)
                    {
                        di=0;
                        dj=-1;
                        IsSatisfying=true;
                    }
                    else IsSatisfying=false;
                    break;
                case 3:
                    if(WanderI<Maze.n-1)
                    {
                        di=1;
                        dj=0;
                        IsSatisfying=true;
                    }
                    else IsSatisfying=false;
                    break;
```

```
                default:
                    break;
                }
                if(IsSatisfying) //если не уперлись в стену, то ставим
стрелку и переходим в соответствии с di dj
                {
                    switch (randcase)
                    {
                    case 0:
                        Maze.SetCellValue(WanderI,            WanderJ,
ARROWRIGHT);
                        break;
                    case 1:
                        Maze.SetCellValue(WanderI, WanderJ, ARROWUP);
                        break;
                    case 2:
                        Maze.SetCellValue(WanderI,            WanderJ,
ARROWLEFT);
                        break;
                    case 3:
                        Maze.SetCellValue(WanderI,            WanderJ,
ARROWDOWN);
                        break;
                    default:
                        break;
                    }
                    WanderI+=di;
                    WanderJ+=dj;
                }
            }
            //Теперь возвращаемся к ячейке, с которой начали блуждания
и строим уже окончательную ветвь по стрелкам.
            WanderI=randI;
            WanderJ=randJ;
            while(Maze.GetCellValue(WanderI, WanderJ)!=UST) //Пока не
дойдем до области UST
            {
                switch (Maze.GetCellValue(WanderI, WanderJ)) //Убираем
стрелку на символ UST и переходим в ячейку по стрелке. Увеличиваем
счетчик ячеек в UST.
                {
                case ARROWRIGHT:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 0, false);
                    USTCount++;
                    WanderJ++;
                    break;
                case ARROWUP:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 1, false);
                    USTCount++;
                    WanderI--;
                    break;
```

```
                case ARROWLEFT:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 2, false);
                    USTCount++;
                    WanderJ--;
                    break;
                case ARROWDOWN:
                    Maze.SetCellValue(WanderI, WanderJ, UST);
                    Maze.SetCellWalls(WanderI, WanderJ, 3, false);
                    USTCount++;
                    WanderI++;
                    break;
                }
            }
        }

    }

    void BinaryTree(maze& Maze, std::default_random_engine& generator, presenthandler& PrHandler, int& alpha) //Хорошо бы сделать универсальность без использования условий. Только за счет использования %. Это, скорее всего, можно сделать(нечто подобное получилось сделать в классе maze), но из соображений читабельности и ограниченности времени оставим как есть.
    {
        std::discrete_distribution<int> distr{0.5, 0.5};
        int StartI=0;
        int StartJ=0;
        switch (alpha%4) //направление вырезания стены. Вторую стену вырезаем против часовой стрелки
        {
        case 0: // L
            StartI=0;
            StartJ=Maze.m-1;
            for(int j=Maze.m-1-1; j>=0; j--) //крайние строку и столбец отдельно, чтобы было читабельно
            {
                Maze.SetCellWalls(0, j, alpha, false);
                if(PrHandler.Mode==2) Maze.ShowDecorate();
            }
            for(int i=1; i<Maze.n; i++)
            {
                Maze.SetCellWalls(i, Maze.m-1, alpha+1, false);
                if(PrHandler.Mode==2) Maze.ShowDecorate();
            }
            for(int i=1; i<Maze.n; i++)
            {
                for(int j=Maze.m-1-1; j>=0; j--)
                {
                    Maze.SetCellWalls(i, j, alpha+distr(generator), false);
                    if(PrHandler.Mode==2) Maze.ShowDecorate();
                }
```

```
            }
            break;
        case 1: // _|
            StartI=0;
            StartJ=0;
            for(int j=1; j<Maze.m; j++) //крайние строку и столбец
отдельно, чтобы было читабельно
            {
                Maze.SetCellWalls(0, j, alpha+1, false);
            }
            for(int i=1; i<Maze.n; i++)
            {
                Maze.SetCellWalls(i, 0, alpha, false);
            }
            for(int i=1; i<Maze.n; i++)
            {
                for(int j=1; j<Maze.m; j++)
                {
                    Maze.SetCellWalls(i,   j,   alpha+distr(generator),
false);
                }
            }
            break;
        case 2: // 7
            StartI=Maze.n-1;
            StartJ=0;
            for(int j=1; j<Maze.m; j++) //крайние строку и столбец
отдельно, чтобы было читабельно
            {
                Maze.SetCellWalls(Maze.n-1, j, alpha, false);
            }
            for(int i=Maze.n-1-1; i>=0; i--)
            {
                Maze.SetCellWalls(i, 0, alpha+1, false);
            }
            for(int i=Maze.n-1-1; i>=0; i--)
            {
                for(int j=1; j<Maze.m; j++)
                {
                    Maze.SetCellWalls(i,   j,   alpha+distr(generator),
false);
                }
            }
            break;
        case 3: // Г
            StartI=Maze.n-1;
            StartJ=Maze.m-1;
            for(int  j=Maze.m-1-1;  j>=0;  j--)  //крайние строку и
столбец отдельно, чтобы было читабельно
            {
                Maze.SetCellWalls(Maze.n-1, j, alpha+1, false);
            }
            for(int i=Maze.n-1-1; i>=0; i--)
```

```
            {
                Maze.SetCellWalls(i, Maze.m-1, alpha, false);
            }
            for(int i=Maze.n-1-1; i>=0; i--)
            {
                for(int j=Maze.m-1-1; j>=0; j--)
                {
                    Maze.SetCellWalls(i,   j,   alpha+distr(generator),
false);
                }
            }
            break;
        }

    }
```

//Есть идея по созданию алгоритма, который бы генерировал лабиринт от случайной точки до неизвестной заранее точки без тупиков. Т.е. змеевидного. При этом лабиринт должен охватывать все выделенное поле.

//Это можно сделать, если взять за основу алгоритм DFS. Т.е. пытающийся закончить начатое решение. (каждая пройденная ячейка будет содержать множество направлений, по которым из неё выходили, их количество(т.е. мы вернулись назад и выбрали другое возможное направление из 3 доступных. Исключение составит лишь исходная ячейка, из которой можно ходить во все 4 стороны, с этим случаем нужно отдельно подумать))

//Очевидно, что если ограничится только этим, то

//время алгоритма начнет быстро расти при увеличении поля. Поэтому нужно ввести дополнительные меры для отсечения заведомо не приводящих к решению варианты.

//Реализуем самое очевидное: если путь коснулся двух стен, то нужно определить, в каком состоянии находятся ячейки от линии, которыми лабиринт разбил поле на 2 области.

//В одной из областей ячейки либо должны все принадлежать лабиринту, либо, если есть ячейки, которые ему не принадлежат, то компонента связности этих ячеек должна иметь мощность, совпадающую с мощностью всех ячеек, которых нет в лабиринте.

//Аналогично можно поступить с ситуацией, когда лабиринт берет в петлю ячейки, которые не принадлежат лабиринту.

//Как только лабиринт коснулся себя, нужно взять эту ячейку и пройти по стрелке вперед, помечая ячейки на этом пути. Как только петля оказалось помеченой, нужно начать проверять состояние ячеек внутри петли аналогично случаю со стенами.

//Оказалось, что на все это добро не требуется дополнительная память. 8 бит, выделенных на состояние каждой ячейки достаточно, чтобы все это организовать.

//Действительно пусть есть один байт: 76543210 .Чтобы закодиновать множество всех уже совершенных переходов из данной ячейки(всего из ячейки можно ходить в 3 направления, в 4 направлении находится ячейка, из которой мы пришли, поэтому это направление не нужно кодировать)

//Нужно всего 4 бита (Всего 14 вариантов уже совершенных переходов: 4 варианта, когда мы совершили один переход из ячейки, 6 вариантов, когда 2 перехода из ячейки, и 4 варианта, когда совершили 3

перехода из ячейки. 4 перехода мы совершить уже не можем, т.к. одно из направлений занимет ячейка, из которой мы пришли)

//Поскольку 4 бита дают 16 вариантов, то у нас остается 2 неиспользованных вариантов. одним из этих вариантов закодируем случай, когда переходов небыло. Например, возьмем для этого код 0000. (Изначальная ячейка единственная, для которой нужно знать случай, когда мы совершали все 4 перехода и не смогли найти решение. Поэтому можем выделить ещё один свободный код под этот случай, например, код 1111. Действительно возьмем лабиринт 3x3. Выберем в качестве исходной ячейки левую среднюю. Не существует лабиринта, который бы являлся решением нашей задачи. Т.е. не для всехо полей существуют такие решения для каждой ячейки. Но если

// взять угловую ячейку, то для неё на любом поле можно найти решение.)

//Таким образом, у нас остается ещё 4 пустых бита. Их можно отвести на пометку ячеек во время провекри на связность (когда встречаем стены или образуем петлю.)

//В общем, алгоритм получается непростым, но вышесказаного достаточно, чтобы его реализовать, как мне кажется. Эффективность его, конечно, будет ещё ниже, чтем алгоритма Уилсона.

//Т.е. 1000x1000 сгенерировать им не получится, скорее всего. алг Уилсона работал для этого около 2х часов. Тут нужно добавлять ещё какие-то условия, помимо проверки на связность, как мне кажется.

//Например, проем в одну ячейку между лабиринтом и стеной или частями лабиринта - такая себе ситуация. Такое возможно только если в этом проеме находится конец всего лабиринта.

//Значит нужно отыскать конец этого единичного проема и строить кусок лабиринта с этого конца в обратном направлении.

//Что это значит? Начальный и конечный куски не могут соединяться, пока есть ячейки, не принадлежащие лабиринту. Значит это мы должны проверять.

//Мы не можем занимать ячейку перед конечным куском лабиринта, пока у нас ещё есть не принадлежащие лабиринту ячейки.

//Нужно ли всвязи с конечным куском лабиринта делать проверку связности я пока не понял. Возможно, того, что перед конечным куском должна быть незанятая ячейка достаточно, чтобы с этим справлялать проверка связности, о которой написано выша.

//Очевидно, что лабиринт не может иметь 2 конца. Поэтому если у нас уже образовался один конец, то дальнейший поиск решения должен осуществляться так, чтобы не образовывался ещё один конец.

//Поскольку на состояние ячейки у нас аж 4 бита (4 бита заняты под множество совершенных переходов), то для этого доп памяти так же не потребуется.

//Вообще, если мы упираемся в память, мы можем хранить лабиринт каким-то другим образом. Например, если хранить в ячейке информацию о станах только одного угла (с добавлением фиктивных ячеек с угла всей области лабиринта), то экономия памяти стремиться к уменьшению в 2 раза. Так, для лабиринта 10x10 экономия 1,408 раза, для 100x100 экономия в 1,923 раза. Для 1000x1000 1,992 раза. (общая формула 1/(1/2+2/n+1/n^2))

//где n - сторона квадратного лабиринта.

```
    //Посмотрел, какие бывают алгоритмы для генерирования таких лабиринтов: вставка стен в идеальный лабиринт (вход и выход рядом). С какой-то хитростью - вход и выход в противоположных концах. Ещё можно использовать Context-based Space Filling Curves, что бы это не значило.
    // //////////////////////////////////////

    //Алгоритм для проряжения стен лабиринта. Нужен, чтобы мы могли использовать с этим лабиринтом алгоритмы поиска кратчайшего пути. Фактически, если изначальный лабиринт идеальный, то происходит соединение случайных узлов дерева друг с другом, за счет чего оно перестает быть деревом.
    void WallsReduce(maze& Maze, double probability, std::default_random_engine& generator) //probability - вероятность разрушения каждой из стен. 1 - лабиринт без внутренних стен.
    {
        std::discrete_distribution<int> distr{probability, 1-probability};
        for(int i=0; i<Maze.n; i++)
        {
            for(int j=0; j<Maze.m; j++)
            {
                if(!distr(generator)) //Если стену нужно убрать
                    Maze.SetCellWalls(i,j, 0, false, true);
                if(!distr(generator))
                    Maze.SetCellWalls(i,j, 3, false, true);
            }
        }
    }

    //просто одна строка кода вместо двух строк кода.
    void WilsonReduced(maze& Maze, std::default_random_engine& generator, presenthandler& PrHandler, double probability)
    {
        Wilson(Maze, generator, PrHandler);
        WallsReduce(Maze, probability, generator);
    }

    //Вспомогательная функция. Тут все попростому. Дискретизация, как я понимаю, это отдельный вопрос.
    inline void WeightCircule(mazeWeighted& MazeWeighted, int Oi, int Oj, int Radius, int Weight)
    {
        for(int i=Oi-Radius; i<=Oi+Radius; i++)
        {
            for(int j=Oj-Radius; j<=Oj+Radius; j++)
            {
                if((i-Oi)*(i-Oi)+(j-Oj)*(j-Oj)<=Radius*Radius && i<MazeWeighted.Weights.size())
                {   if(j<MazeWeighted.Weights.at(i).size())
                        MazeWeighted.Weights.at(i).at(j)=Weight;
                }
            }
```

```
        }
    //    std::cout<<"Radius="<<Radius<<std::endl;
    }
    //Генерация весов ячеек для лабиринта с весами
    //Генерация меня сильно озадачила. Вначале я хотел сделать, чтобы генерация зависела от размеров лабиринта, но сейчас мне это не кажется хорошей идеей.
    void RandomCircules(mazeWeighted& MazeWeighted, std::default_random_engine& generator, int minWeight, int maxWeight, double probability, int meanRadius, double stddevRadiusRatio) //Генерируем круги. probability вероятность, что из данной ячейки будет построен круг с текущим весом. meanRadius - матожидание радиуса. stddevRatio чем меньше, тем меньше отклонение от матожидания
    {
    //    std::uniform_int_distribution<int> distr();
        std::normal_distribution<double> distrRadius(meanRadius, stddevRadiusRatio*meanRadius); //для генерации радиусов кругов
        std::discrete_distribution<int> distrAppearance{1-probability, probability}; //для принятия решения о генерации круга в ячейке

        auto PositiveRadius{[](int Radius)->int
            {
                if(Radius<0) Radius=0;
                return Radius;
            }
        };
        for(int k=minWeight; k<=maxWeight; k++)
        {
            for(int i=0; i<MazeWeighted.Weights.size(); i++)
            {
                for(int j=0; j<MazeWeighted.Weights.at(i).size(); j++)
                {
                    if(distrAppearance(generator))
                    {
                        WeightCircule(MazeWeighted, i, j, PositiveRadius(distrRadius(generator)), k);
                    }
                }
            }
        }
    //                    WeightCircule(MazeWeighted, 5, 5, PositiveRadius(distrRadius(generator)), 1);



    }
    #endif /* MAZEGENERATIONALGS_H */
```

6.4. MazeSearchAlg.h

```
#ifndef MAZESEARCHALG_H
#define MAZESEARCHALG_H

#include "maze.h"
```

```
    #include "presenthandler.h"
    #include <vector>
    #include <utility> //pair
    #include <limits> //будем помечать максимальным возможным значением непосещенные ячейки.

    #include <queue>
    #include "Funcs.h"
    #include <cmath>
    #define NOTVISITED '~'


    void DebugHandler(maze& Maze, std::vector<std::vector<std::pair<char, int>>>& CellDist, presenthandler& PrHandler); //Небольшие манипуляции, чтобы помочь отладить это.

    //Постараемся реализовать волновой алг и, может быть, его модификацию с двумя волнами. Так же есть идея немного его ускорить без какого-либо влияния на функциональность путем сокращения области поиска ячеек фронта на этапе распространения волны.
    void Lee(maze& Maze, int starti, int startj, int finishi, int finishj, std::vector<std::pair<int,int>>& Way, presenthandler& PrHandler)
    {
        std::vector<std::vector<int>> CellDist(Maze.n); //В векторах нулевые значения в стандартных контейнерах будут зануленными. Смотри value-initialization и тому подобное.
        for(int i=0; i<CellDist.size(); i++) //Храним в этой матрице веса во время распространения волны.
        {
            CellDist.at(i).resize(Maze.m, std::numeric_limits<int>::max()); //Макс значением int помечаем ещё не пройденные ячейки.
        }
        int CurrentDist=0;
        bool IsReachedfinish=false;
        //Распространяем волну, пока не попадем в точку назначения
        CellDist.at(starti).at(startj)=CurrentDist;
        if((starti==finishi)&&(startj==finishj)) IsReachedfinish=true;
        bool IsChanged=true; //Проверяем, продвинулась ли волна хотя бы на одну ячейку. Если нет, то значит, что пути не существует.

        int mini=starti, maxi=starti, minj=startj, maxj=startj; //Для сужения области поиска ячеек фронта волны. Больше ни для чего не будем эти переменные использовать
        while((!IsReachedfinish)&&IsChanged) //одна итерация-одно продвижение фронта волны. Пока не дойдем до финишной ячейки или пока есть куда ходить
        {

            IsChanged=false;
            CurrentDist++;
```

```
            for(int    i=mini;    (i<=maxi)&&(!IsReachedfinish);    i++)
//Пробегаем все ячейки фронта волны
            {
                for(int j=minj; (j<=maxj)&&(!IsReachedfinish); j++)
                {
                    if(CellDist.at(i).at(j)==CurrentDist-1)      //Если
данная ячейка - ячейка фронта волны
                    {
                        for(int                                  alpha=0;
(alpha<4)&&(!IsReachedfinish);   alpha++)   //Пытаемся   распространить
волну на соседние ячейки
                        {
                            if(!Maze.HasWall(i, j, alpha)) //Если нет
стенки
                            {
                                int  di=(alpha%2)*((alpha%4)*(alpha%2)-
2); //формулы получены из графиков
                                int          dj=((alpha-1)%2)*(((alpha-
1)%4)*((alpha-1)%2)-2);

if(CellDist.at(i+di).at(j+dj)==std::numeric_limits<int>::max())  //если
волна ещё не проходила через эти ячейки. Можно ли объединить этот цикл
с тем, в который он вложен, не ясно. Т.к. мы можем попадать в ячейки,
которых не сущетсвует.
                                {

CellDist.at(i+di).at(j+dj)=CurrentDist;
                                    //для сужения области поиска ячеек
фронта волны на поле лабиринта
                                    if(i+di<mini) mini=i+di;
                                    if(i+di>maxi) maxi=i+di;
                                    if(j+dj<minj) minj=j+dj;
                                    if(j+dj>maxj) maxj=j+dj;
                                    // /////

if((i+di==finishi)&&(j+dj==finishj)) //если волна дошла до финишной
ячейки, то прекращаем распространение волны
                                    {
                                        IsReachedfinish=true;
                                    }
                                    IsChanged=true;
                                }

                            }
                        }
                    }
                }
            }
            if(PrHandler.Mode==1)    //Небольшие   манипуляции,   чтобы
помочь отладить это.
            {
                for(int i=0; i<Maze.n; i++)
                {
```

```
                    for(int j=0; j<Maze.m; j++)
                    {

if(CellDist.at(i).at(j)==std::numeric_limits<int>::max())
                        {
                            Maze.SetCellValue(i, j, '~');
                        }
                        else
                        {
                            Maze.SetCellValue(i,                    j,
(char)CellDist.at(i).at(j)+48); //Если CellDist будет иметь большие
значения, то в char они не влезут. Но для отладки подойдет.
                        }
                    }
                }
                Maze.ShowDecorate((char*)"cout", 1, 2, true);
                std::cin.ignore();
            }
        }
        //Проверяем существование пути и восстанавливаем путь.
        if(!IsReachedfinish)
        {
            std::cout<<"The     way     between     starti="<<starti<<"
startj="<<startj<<"   and   finishi="<<finishi<<"   finishj="<<finishj<<"
does not exist!"<<std::endl; //you do not know de wey
            return;
        }
        //пока не дойдем до начала
        int Currenti=finishi, Currentj=finishj;
        Way.clear();
        Way.insert(Way.begin(),std::make_pair(Currenti, Currentj));
        while(!((Currenti==starti)&&(Currentj==startj)))
        {
            for(int alpha=0; alpha<4;alpha++)
            {
                int   di=(alpha%2)*((alpha%4)*(alpha%2)-2);   //формулы
получены из графиков
                int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);
                if(!Maze.HasWall(Currenti,Currentj, alpha)) //Если нет
стены
                {

if(CellDist.at(Currenti+di).at(Currentj+dj)==CellDist.at(Currenti).at(
Currentj)-1)
                    {
                        Way.insert(Way.begin(),
std::make_pair(Currenti+di, Currentj+dj));
                        Currenti+=di;
                        Currentj+=dj;
                        break;
                    }
                }
            }
```

```
        }
    }

    //Модификация алг Ли с 2 волнами (от стартовой и финишной точек)
    //Итерацию движения фронта нужно выделить в отдельную функцию. Фронты будут двигаться поочередно, пока не встретятся друг с другом. При встрече фронтов в ячейку, в которой они пересекутся, поместим наибольший вес из весов, которые соответствуют первой и второй волне.
    //Возвращает 2 значения: пересеклись ли волны и не застряла ли волна.
    inline void WaveFrontIteration(maze& Maze, std::vector<std::vector<std::pair<char, int>>>& CellDist, int CurrentDist, int& mini, int& maxi, int& minj, int& maxj, bool& IsReachedfinish, bool& IsChanged, char WaveID, int& Interseci, int& Intersecj)
    {
        IsChanged=false; //если застряла хотя бы одна волна, то поиск можно прекращать.
        for(int i=mini; (i<=maxi)&&(!IsReachedfinish); i++) //Пробегаем все ячейки фронта волны
        {
            for(int j=minj; (j<=maxj)&&(!IsReachedfinish); j++)
            {
                if((CellDist.at(i).at(j).second==CurrentDist-1)&&(CellDist.at(i).at(j).first==WaveID)) //Если данная ячейка - ячейка фронта волны
                {
                    for(int alpha=0; (alpha<4)&&(!IsReachedfinish); alpha++) //Пытаемся распространить волну на соседние ячейки
                    {
                        if(!Maze.HasWall(i, j, alpha)) //Если нет стенки
                        {
                            int di=(alpha%2)*((alpha%4)*(alpha%2)-2); //формулы получены из графиков
                            int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);

if(CellDist.at(i+di).at(j+dj).first!=WaveID) //если волна ещё не проходила через эти ячейки. Т.е. в эту ячейку можно распространить волну. Можно ли объединить этот цикл с тем, в который он вложен, не ясно. Т.к. мы можем попадать в ячейки, которых не сущетсвует.
                            {

if(CellDist.at(i+di).at(j+dj).first!=0) //если волна дошла до пересечения с другой волной, то прекращаем распространение волны. За 0 обозначим ячейки, по которым не прошла ни одна волна
                                {
                                    IsReachedfinish=true;
                                    Interseci=i+di;
                                    Intersecj=j+dj;

CellDist.at(i+di).at(j+dj).second=std::max(CurrentDist,
```

```
CellDist.at(i+di).at(j+dj).second);   //Для определенности назначим ячейке пересечения наибольший вес
                                    }
                                    else
                                    {

CellDist.at(i+di).at(j+dj).second=CurrentDist;

CellDist.at(i+di).at(j+dj).first=WaveID;
                                    }
                                    //для сужения области поиска ячеек фронта волны на поле лабиринта
                                    if(i+di<mini) mini=i+di;
                                    if(i+di>maxi) maxi=i+di;
                                    if(j+dj<minj) minj=j+dj;
                                    if(j+dj>maxj) maxj=j+dj;
                                    // /////

                                    IsChanged=true;
                                }

                            }
                        }
                    }
                }
            }
    //Инициализация волны. Отмечает ячейку, из которой распространяется волна. Если ячейка была посещена другой волной, то выставляет флаг IsReachedfinish. Присваивает начальные значения min/max/i/j для сужения области поиска ячеек фронта волны
    inline void WaveFrontInit(maze& Maze, std::vector<std::vector<std::pair<char, int>>>& CellDist, int CurrentDist, int& mini, int& maxi, int& minj, int& maxj, bool& IsReachedfinish, char WaveID, int& Interseci, int& Intersecj, int Initi, int Initj)
    {
        if(CellDist.at(Initi).at(Initj).first!=0) //Если встретились с другой волной. Т.е. случай, когда начальная и конечная ячейки сопадают
        {
            IsReachedfinish=true;
            Interseci=Initi;
            Intersecj=Initj;
            CellDist.at(Initi).at(Initj).second=std::max(CurrentDist, CellDist.at(Initi).at(Initj).second);  //Для определенности назначим ячейке пересечения наибольший вес
        }
        else
        {
            CellDist.at(Initi).at(Initj).second=CurrentDist;
            CellDist.at(Initi).at(Initj).first=WaveID;
        }
        mini=Initi;
```

```
        maxi=Initi;
        minj=Initj;
        maxj=Initj;
    }
    inline void ReconstructWay(maze& Maze, std::vector<std::vector<std::pair<char, int>>>& CellDist, char WaveID, int Interseci, int Intersecj, int Initi, int Initj, std::vector<std::pair<int, int>>& Way, bool Direction) //Восстановение пути после пересечения волн. WaveID-по какой волне идем; Interseci, Intersecj точка пересечения волн. int Initi, int Initj точка инициации волны; bool Direction - направление построение пути (false - начало пути true - конец пути).
    {
        int Currenti=Interseci, Currentj=Intersecj;
        while(!((Currenti==Initi)&&(Currentj==Initj)))
        {
            if((Currenti==Interseci)&&(Currentj==Intersecj)) //Такой поиск предыдущей ячейки в пути нужен только для ячейки пересечения. Для остальных ячеек поиск будет аналогичным поиску в немодифицированном алг Ли.
            {
                //Ищем соседа с минимальным весом из всех соседей для данной WaveID.
                int MinNeighbor=0;
                int minalpha=0; //Альфа, указывающая на минимального соседа.
                int Cutalpha=0; //Чуть чуть ускорим алг. Помимо поиска первого соседа, из которого могла быть распространена волна, мы ищем соответствующую ему альфу, чтобы вдальнейшем не перебирать варианты, в которых заведомо нет соседей.
                for(int alpha=0; alpha<4;alpha++) //Ищем первого попавшегося соседа, принадлежащего данной волне.
                {
                    int di=(alpha%2)*((alpha%4)*(alpha%2)-2); //формулы получены из графиков
                    int dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-2);
                    if(!Maze.HasWall(Currenti,Currentj, alpha)) //Если нет стены
                    {

if(CellDist.at(Currenti+di).at(Currentj+dj).first==WaveID) //Если ячейка принадлежит волне WaveID
                        {

MinNeighbor=CellDist.at(Currenti+di).at(Currentj+dj).second;
                            Cutalpha=alpha; //альфа, при которой мы нашли первого встретившегося соседа, принадлежащего данной волне.
                            minalpha=alpha;
                            break;
                        }
                    }
                }
```

```
            //    std::cout<<"minalpha="<<minalpha<<std::endl;

                for(int   alpha=Cutalpha+1;   alpha<4;alpha++)   //ищем
минимального соседа из всех соседей, принадлежащих данной волне.
                {
                    int           di=(alpha%2)*((alpha%4)*(alpha%2)-2);
//формулы получены из графиков
                    int  dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-
2);
                    if(!Maze.HasWall(Currenti,Currentj, alpha)) //Если
нет стены
                    {

if((CellDist.at(Currenti+di).at(Currentj+dj).first==WaveID)&&(CellDist
.at(Currenti+di).at(Currentj+dj).second<MinNeighbor))
                        {

MinNeighbor=CellDist.at(Currenti+di).at(Currentj+dj).second;
                            minalpha=alpha;
                        }
                    }
                }
                //Добавляем минимального соседа, принадлежащего данной
волне, в путь, и переходим к нему.
                int      di=(minalpha%2)*((minalpha%4)*(minalpha%2)-2);
//формулы получены из графиков
                int   dj=((minalpha-1)%2)*(((minalpha-1)%4)*((minalpha-
1)%2)-2);

                if(Direction) //Для конечного участка пути
                {
                    Way.push_back(std::make_pair(Currenti+di,
Currentj+dj));
                }
                else //Для начального участка пути
                {
                    Way.insert(Way.begin(),
std::make_pair(Currenti+di, Currentj+dj));
                }
                Currenti+=di;
                Currentj+=dj;
            }
            else
            {
                for(int alpha=0; alpha<4;alpha++)
                {
                    int           di=(alpha%2)*((alpha%4)*(alpha%2)-2);
//формулы получены из графиков
                    int  dj=((alpha-1)%2)*(((alpha-1)%4)*((alpha-1)%2)-
2);
                    if(!Maze.HasWall(Currenti,Currentj, alpha)) //Если
нет стены
                    {
```

```
if((CellDist.at(Currenti+di).at(Currentj+dj).first==WaveID)&&(CellDist
.at(Currenti+di).at(Currentj+dj).second==CellDist.at(Currenti).at(Curr
entj).second-1))
                        {
                            if(Direction) //Для конечного участка пути
                            {

Way.push_back(std::make_pair(Currenti+di, Currentj+dj));
                            }
                            else //Для начального участка пути
                            {
                                Way.insert(Way.begin(),
std::make_pair(Currenti+di, Currentj+dj));
                            }
                            Currenti+=di;
                            Currentj+=dj;
                            break;
                        }
                    }
                }
            }
        }
    }

    void Lee2Waves(maze& Maze, int starti, int startj, int finishi,
int  finishj,  std::vector<std::pair<int,int>>&  Way,  presenthandler&
PrHandler)
    {
        //Содзаем контейнер, в котором будем индекс волны, прошедшей
через ячейки и веса ячеек при движении этой волны. Поле внутри Maze
для хранения состояния ячейки не будем использоать в этом алгоритма,
дабы избежать путаницы. Оно будет использоваться только для наглядного
представления при отладке.
        std::vector<std::vector<std::pair<char,                  int>>>
CellDist(Maze.n);  //В  векторах  нулевые  значения  в  стандартных
контейнерах будут зануленными. Смотри value-initialization и тому
подобное.
        for(int i=0; i<CellDist.size(); i++) //Храним в этой матрице
веса во время распространения волны.
        {
            CellDist.at(i).resize(Maze.m,             std::make_pair(0,
std::numeric_limits<int>::max())); //Макс значением int помечаем ещё
не пройденные ячейки.
        }
        int CurrentDist=0;
        bool IsReachedfinish=false;
        bool  IsChanged=true;  //Проверяем,  продвинулась  ли  волна  хотя
бы на одну ячейку. Если нет, то значит, что пути не существует.
        int                  Interseci=std::numeric_limits<int>::max(),
Intersecj=std::numeric_limits<int>::max();//Для   упрощения   отладки
инициализируем самым большим числом
```

```
        int mini1, maxi1, minj1, maxj1; //Для сужения области поиска ячеек фронта волны. Больше ни для чего не будем эти переменные использовать
        WaveFrontInit(Maze, CellDist, CurrentDist, mini1, maxi1, minj1, maxj1, IsReachedfinish, 1, Interseci, Intersecj, starti, startj);
        int mini2, maxi2, minj2, maxj2;
        WaveFrontInit(Maze, CellDist, CurrentDist, mini2, maxi2, minj2, maxj2, IsReachedfinish, 2, Interseci, Intersecj, finishi, finishj);

        while((!IsReachedfinish)&&IsChanged)    //одна итерация-одно продвижение фронта волны. Пока волны не встретятся или не застрянут.
        {
            CurrentDist++;
            WaveFrontIteration(Maze, CellDist, CurrentDist, mini1, maxi1, minj1, maxj1, IsReachedfinish, IsChanged, 1, Interseci, Intersecj);
            if(PrHandler.Mode==1)                       std::cout<<"Wave1 CurrDist="<<CurrentDist<<std::endl;
            if((IsReachedfinish)||(!IsChanged))
            {
                DebugHandler(Maze, CellDist, PrHandler);
                break; //если достигли пересечения или если волна застряла
            }
            WaveFrontIteration(Maze, CellDist, CurrentDist, mini2, maxi2, minj2, maxj2, IsReachedfinish, IsChanged, 2, Interseci, Intersecj);
            if(PrHandler.Mode==1)                       std::cout<<"Wave2 CurrDist="<<CurrentDist<<std::endl;
            DebugHandler(Maze, CellDist, PrHandler);
        }

        //Проверяем существование пути и восстанавливаем путь.
        if(!IsReachedfinish)
        {
            std::cout<<"The     way     between     starti="<<starti<<" startj="<<startj<<"  and  finishi="<<finishi<<"  finishj="<<finishj<<" does not exist!"<<std::endl; //you do not know de wey
            return;
        }
        else
        {
            if(PrHandler.Mode==1)     std::cout<<"Intersection     cell between                waves:                Interseci="<<Interseci<<" Intersecj="<<Intersecj<<std::endl;
        }
        //Будем идти от точки пересечения в обе стороны. Вначале дойдем до точки старта, потом до точки финиша.
        int Currenti=Interseci, Currentj=Intersecj;
        Way.clear();
        Way.insert(Way.begin(),std::make_pair(Currenti, Currentj));
```

```
        //Восстанавливаем участок пути от стартовой ячейки до ячейки пересечения волн.
        ReconstructWay(Maze, CellDist, 1, Interseci, Intersecj, starti, startj, Way, false);
        // Теперь то же самое для конечного участка, т.е. для второй волны. Т.е. восстанавливаем второй кусок пути.
        ReconstructWay(Maze, CellDist, 2, Interseci, Intersecj, finishi, finishj, Way, true);

    }

    void DebugHandler(maze& Maze, std::vector<std::vector<std::pair<char, int>>>& CellDist, presenthandler& PrHandler) //Небольшие манипуляции, чтобы помочь отладить это.
    {
        if(PrHandler.Mode==1) //Небольшие манипуляции, чтобы помочь отладить это.
        {
            for(int i=0; i<Maze.n; i++)
            {
                for(int j=0; j<Maze.m; j++)
                {

if(CellDist.at(i).at(j).second==std::numeric_limits<int>::max())
                    {
                        Maze.SetCellValue(i, j, '~');
                    }
                    else
                    {
                        Maze.SetCellValue(i, j, (char)CellDist.at(i).at(j).second+48); //Если CellDist будет иметь большие значения, то в char они не влезут. Но для отладки подойдет.
                    }
                }
            }
            Maze.ShowDecorate((char*)"cout", 1, 2, true);
            std::cin.ignore();
        }
    }

    //Для алг AStar Dijkstra
    inline void DijkstraReconstructWay(mazeWeighted& MazeWeighted, std::vector<std::vector<std::pair<char, int>>>& PathCoordWeights, int finishi, int finishj, int starti, int startj, std::vector<std::pair<int, int>>& Path) //Восстановение пути.
    {
        int Currenti=finishi, Currentj=finishj;
        Path.insert(Path.begin(), std::make_pair(Currenti, Currentj));

        while(!((Currenti==starti)&&(Currentj==startj)))
        {
```

```
         //                          std::cout<<"Currenti="<<Currenti<<"
Currentj="<<Currentj<<std::endl;
            int       tempCurrenti=MazeWeighted.GetNeighborI(Currenti,
PathCoordWeights.at(Currenti).at(Currentj).first+2);      //Идем      в
противоположную сторону, из которой пришли
            Currentj=MazeWeighted.GetNeighborJ(Currentj,
PathCoordWeights.at(Currenti).at(Currentj).first+2);
            Currenti=tempCurrenti;

            Path.insert(Path.begin(),          std::make_pair(Currenti,
Currentj));
        }
    }
    //
    void Dijkstra(mazeWeighted& MazeWeighted, int starti, int startj,
int  finishi,  int  finishj,  std::vector<std::pair<int,  int>>&  Path,
presenthandler& PrHandler)
    {
        std::vector<std::vector<std::pair<char,                  int>>>
PathCoordWeights(MazeWeighted.n); //Путевые координаты и веса пути у
ячеек
        for(auto& i:PathCoordWeights)
        {
            i.resize(MazeWeighted.m,  std::make_pair(NOTVISITED,  0));
//NOTVISITED Будем так помечать непосещенные ячейки
        //    for(auto& j:i) j.first=NOTVISITED;
        }

        //Очередь  с  приоритетом.  Хранит  ячейки,  которые  нужно
посетить. В качестве приоритета будем брать расстояние от ячейки
старта + эвристика, зависящая от расстояния(на плоскости) до финишной
ячейки
        std::priority_queue<std::pair<int,    std::pair<int,    int>>,
std::vector<std::pair<int,           std::pair<int,           int>>>,
std::greater<std::pair<int, std::pair<int, int>>>> PriorQueue; //пара-
приоритет(вес пути); координаты ячейки

    /*  //Тестируем очередь с приоритетами
        std::vector<std::pair<int, std::pair<int, int>>> testVector;
        std::pair<int, std::pair<int, int>> testItem1, testItem2;

        testItem1.first=3;
        testItem1.second.first=1;
        testItem1.second.second=2;
        PriorQueue.emplace(testItem1);

        testItem1.first=0;
        testItem1.second.first=4;
        testItem1.second.second=5;
        PriorQueue.emplace(testItem1);
        while(!PriorQueue.empty())
        {
            testVector.push_back(PriorQueue.top());
```

```
            PriorQueue.pop();
        }
        PrintVector(testVector, "testVector");
    */
        //Устанавливаем атрибуты стартовой ячейки
        PathCoordWeights.at(starti).at(startj).first=4;       //Путевые
координаты пусть будут такими 0-3 значения alpha, если смотреть из
ячейки, из которой мы пришли. 4 - отстуствие координат(только для
исходной ячейки). NOTVISITED='~' ячейка не была посещена.
        PathCoordWeights.at(starti).at(startj).second=0; //Вес пути
        //Записываем в очередь стартовую ячейку

PriorQueue.push(std::make_pair(PathCoordWeights.at(starti).at(startj).
second, std::make_pair(starti, startj)));

        //Каждая итерация - просмотр одного элемента из очереди.
        bool IsFinishReached=false; //Достигли ли финишной ячейки
        if(PrHandler.Mode==1)
        {
            std::cout<<"PriorQueue all items:"<<std::endl;
        }
        while(!PriorQueue.empty())
        {
            //Извлекаем наиближайший элемент из очереди.  (т.е.
элемент, вес пути которого минимален из всех элементов в очереди)
            std::pair<int, std::pair<int, int>> CellFromQueue; //Для
элемента, извлеченного из очереди
            CellFromQueue=PriorQueue.top();

            if(PrHandler.Mode==1)
            {
            std::cout<<"PriorQueue.top()=";
            Print(PriorQueue.top());
            std::cout<<std::endl;
            }
            PriorQueue.pop();

            //Если извлеченным элементом оказалась финишная ячейка, то
прекращаем дальнейший обход ячеек (т.е. если это произошло, то значит
в других ячейках веса путей не меньше, чем в финишной, а значит, с
меньшим весом путь до финишной ячейки уже никак не может быть (случай
с отрицательными весами ребер исходного графа мы здесь не
рассматриваем))
            if(CellFromQueue.second==std::make_pair(finishi,finishj))
            {
                IsFinishReached=true;
                break;
            }

            //Просматриваем всех соседов извлеченной ячейки.
            for(int alpha=0; alpha<4; alpha++)
            {
```

```
                int
NeighborI=MazeWeighted.GetNeighborI(CellFromQueue.second.first,
alpha);
                int
NeighborJ=MazeWeighted.GetNeighborJ(CellFromQueue.second.second,
alpha);
              //
std::cout<<"NeighborI="<<NeighborI<<"NeighborJ"<<NeighborJ<<std::endl;
                if(!MazeWeighted.HasWall(CellFromQueue.second.first,
CellFromQueue.second.second, alpha) ) //Если у текущая ячейка не
отделена от соседней стеной
                {

if(PathCoordWeights.at(NeighborI).at(NeighborJ).first==NOTVISITED)
//Если этот сосед не был посещен ни разу
                    {

PathCoordWeights.at(NeighborI).at(NeighborJ).first=alpha;
//устанавливаем путевые координаты

PathCoordWeights.at(NeighborI).at(NeighborJ).second=PathCoordWeights.a
t(CellFromQueue.second.first).at(CellFromQueue.second.second).second +
MazeWeighted.Weights.at(NeighborI).at(NeighborJ);    //Вес    пути    в
соседней ячейке равен весу пути в текущей ячейке+вес исходный в
соседней ячейке

PriorQueue.emplace(std::make_pair(PathCoordWeights.at(NeighborI).at(Ne
ighborJ).second, std::make_pair(NeighborI, NeighborJ))); //Помещаем в
очередь (вес_он_же_приоритет, (i, j))

                        if(PrHandler.Mode==2)
                        {
                            std::cout<<"PathWeights                    %
1000:"<<std::endl;
                            MazeWeighted.ShowDecorate((char*)"cout",
1, 2, true, PathCoordWeights, 2);
                            //Наглядное      представление      путевых
координат
                            for(int i=0; i<MazeWeighted.n; i++)
                            {
                                for(int j=0; j<MazeWeighted.m; j++)
                                {
                                //
if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                                //                                  else
MazeWeighted.SetCellValue(i, j, PathCoordWeights.at(i).at(j).first);
                                    switch
(PathCoordWeights.at(i).at(j).first)
                                    {
                                    case 0:
```

```
                                        MazeWeighted.SetCellValue(i,
j, '>');
                                        break;
                                    case 1:
                                        MazeWeighted.SetCellValue(i,
j, '^');
                                        break;
                                    case 2:
                                        MazeWeighted.SetCellValue(i,
j, '<');
                                        break;
                                    case 3:
                                        MazeWeighted.SetCellValue(i,
j, 'v');
                                        break;
                                    case 4:
                                        MazeWeighted.SetCellValue(i,
j, 's');
                                        break;
                                    default:
                                        MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first);
                                        break;
                                    }
                                }
                            }
                            std::cout<<"PathCoordinates:"<<std::endl;
                            MazeWeighted.ShowDecorate((char*)"cout",
1, 2, true);
                            std::cin.ignore();
                        }
                    }
                    else //Если сосед уже был посещен
                    {
                        int
NewPathWeight=PathCoordWeights.at(CellFromQueue.second.first).at(CellF
romQueue.second.second).second                                      +
MazeWeighted.Weights.at(NeighborI).at(NeighborJ);

if(NewPathWeight<PathCoordWeights.at(NeighborI).at(NeighborJ).second)
//Если вес пути оказался меньше, чем уже найденный
                        {
                            //Делаем то же самое

PathCoordWeights.at(NeighborI).at(NeighborJ).first=alpha;
//устанавливаем путевые координаты

PathCoordWeights.at(NeighborI).at(NeighborJ).second=PathCoordWeights.a
t(CellFromQueue.second.first).at(CellFromQueue.second.second).second  +
MazeWeighted.Weights.at(NeighborI).at(NeighborJ);    //Вес    пути    в
соседней ячейке равен весу пути в текущей ячейке+вес исходный в
соседней ячейке
```

```
PriorQueue.emplace(std::make_pair(PathCoordWeights.at(NeighborI).at(Ne
ighborJ).second, std::make_pair(NeighborI, NeighborJ))); //Помещаем в
очередь (вес_он_же_приоритет, (i, j))
                            if(PrHandler.Mode==2)
                            {
                                std::cout<<"PathWeights                %
1000:"<<std::endl;

MazeWeighted.ShowDecorate((char*)"cout", 1, 2, true, PathCoordWeights,
2);
                                //Наглядное    представление    путевых
координат
                                for(int i=0; i<MazeWeighted.n; i++)
                                {
                                    for(int    j=0;    j<MazeWeighted.m;
j++)
                                    {
                                    //
if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                                    //                                 else
MazeWeighted.SetCellValue(i, j, PathCoordWeights.at(i).at(j).first);
                                        switch
(PathCoordWeights.at(i).at(j).first)
                                        {
                                        case 0:

MazeWeighted.SetCellValue(i, j, '>');
                                            break;
                                        case 1:

MazeWeighted.SetCellValue(i, j, '^');
                                            break;
                                        case 2:

MazeWeighted.SetCellValue(i, j, '<');
                                            break;
                                        case 3:

MazeWeighted.SetCellValue(i, j, 'v');
                                            break;
                                        case 4:

MazeWeighted.SetCellValue(i, j, 's');
                                            break;
                                        default:

MazeWeighted.SetCellValue(i, j, PathCoordWeights.at(i).at(j).first);
                                            break;
                                        }
                                    }
```

```
                                }
std::cout<<"PathCoordinates:"<<std::endl;

MazeWeighted.ShowDecorate((char*)"cout", 1, 2, true);
                                std::cin.ignore();
                            }
                        }
                    }
                }
            }
        }

        if(PrHandler.Mode==1)
        {
            //Вывод изначальных весов ячеек.
        //    MazeWeighted.WeightsToValues();
            std::cout<<"Weights:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout",1,     2,     true,
MazeWeighted.Weights);

            //Наглядное представление путевых весов
           /* for(int i=0; i<MazeWeighted.n; i++)
            {
                for(int j=0; j<MazeWeighted.m; j++)
                {
                    MazeWeighted.SetCellValue(i,                    j,
PathCoordWeights.at(i).at(j).second%10+48);
                //             if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                //             else   MazeWeighted.SetCellValue(i,   j,
PathCoordWeights.at(i).at(j).first);
                }
            }*/
            std::cout<<"PathWeights % 1000:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout",   1,   2,   true,
PathCoordWeights, 2);

            //Наглядное представление путевых координат
            for(int i=0; i<MazeWeighted.n; i++)
            {
                for(int j=0; j<MazeWeighted.m; j++)
                {
                //             if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                //             else   MazeWeighted.SetCellValue(i,   j,
PathCoordWeights.at(i).at(j).first);
                    switch (PathCoordWeights.at(i).at(j).first)
                    {
                    case 0:
                        MazeWeighted.SetCellValue(i, j, '>');
```

```
                        break;
                    case 1:
                        MazeWeighted.SetCellValue(i, j, '^');
                        break;
                    case 2:
                        MazeWeighted.SetCellValue(i, j, '<');
                        break;
                    case 3:
                        MazeWeighted.SetCellValue(i, j, 'v');
                        break;
                    case 4:
                        MazeWeighted.SetCellValue(i, j, 's');
                        break;
                    default:
                        MazeWeighted.SetCellValue(i,                j,
PathCoordWeights.at(i).at(j).first);
                        break;
                    }
                }
            }
            std::cout<<"PathCoordinates:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout", 1, 2, true);
        }

        //Проверяем, достигли ли мы финишной ячейки
        if(!IsFinishReached)
        {
            std::cout<<"The path from ("<<starti<<", "<<startj<< ") to
("<<finishi<<", "<<finishj<< ") does not exist!"<<std::endl;
            return;
        }
        if(PrHandler.Mode==1)
        {
            std::cout<<"The finish cell is reached!"<<std::endl;
        }
        DijkstraReconstructWay(MazeWeighted,          PathCoordWeights,
finishi, finishj, starti, startj, Path);
    }
    //Эвристическая функция для алг A*
    int HeuristicAStar(int Currenti, int Currentj, int finishi, int
finishj)
    {
        return abs(finishi-Currenti)+abs(finishj-Currentj);
    }
    void AStar(mazeWeighted& MazeWeighted, int starti, int startj, int
finishi,   int   finishj,   std::vector<std::pair<int,   int>>&   Path,
presenthandler& PrHandler)
    {
        std::vector<std::vector<std::pair<char,                  int>>>
PathCoordWeights(MazeWeighted.n); //Путевые координаты и веса пути у
ячеек
        for(auto& i:PathCoordWeights)
        {
```

```
            i.resize(MazeWeighted.m,  std::make_pair(NOTVISITED,  0));
//NOTVISITED Будем так помечать непосещенные ячейки
        //    for(auto& j:i) j.first=NOTVISITED;
        }

        //Очередь с приоритетом. Хранит ячейки, которые нужно посетить. В качестве приоритета будем брать расстояние от ячейки старта + эвристика, зависящая от расстояния(на плоскости) до финишной ячейки
        std::priority_queue<std::pair<int,    std::pair<int,    int>>,
std::vector<std::pair<int,           std::pair<int,           int>>>,
std::greater<std::pair<int, std::pair<int, int>>>> PriorQueue; //пара-приоритет(вес пути); координаты ячейки

    /*  //Тестируем очередь с приоритетами
        std::vector<std::pair<int, std::pair<int, int>>> testVector;
        std::pair<int, std::pair<int, int>> testItem1, testItem2;

        testItem1.first=3;
        testItem1.second.first=1;
        testItem1.second.second=2;
        PriorQueue.emplace(testItem1);

        testItem1.first=0;
        testItem1.second.first=4;
        testItem1.second.second=5;
        PriorQueue.emplace(testItem1);
        while(!PriorQueue.empty())
        {
            testVector.push_back(PriorQueue.top());
            PriorQueue.pop();
        }
        PrintVector(testVector, "testVector");
    */
        //Устанавливаем атрибуты стартовой ячейки
        PathCoordWeights.at(starti).at(startj).first=4;       //Путевые координаты пусть будут такими 0-3 значения alpha, если смотреть из ячейки, из которой мы пришли. 4 - отстуствие координат(только для исходной ячейки). NOTVISITED='~' ячейка не была посещена.
        PathCoordWeights.at(starti).at(startj).second=0; //Вес пути
        //Записываем в очередь стартовую ячейку

PriorQueue.push(std::make_pair(PathCoordWeights.at(starti).at(startj).
second+HeuristicAStar(starti,      startj,      finishi,      finishj),
std::make_pair(starti, startj)));

        //Каждая итерация - просмотр одного элемента из очереди.
        bool IsFinishReached=false; //Достигли ли финишной ячейки
        if(PrHandler.Mode==1)
        {
            std::cout<<"PriorQueue all items:"<<std::endl;
        }
        while(!PriorQueue.empty())
```

```
        {
            //Извлекаем наиближайший элемент из очереди. (т.е. элемент, вес пути=вес дейкстры+эвирстика  которого минимален из всех элементов в очереди)
            std::pair<int, std::pair<int, int>> CellFromQueue; //Для элемента, извлеченного из очереди
            CellFromQueue=PriorQueue.top();

            if(PrHandler.Mode==1)
            {
            std::cout<<"PriorQueue.top()=";
            Print(PriorQueue.top());
            std::cout<<std::endl;
            }
            PriorQueue.pop();

            //Если извлеченным элементом оказалась финишная ячейка, то прекращаем дальнейший обход ячеек (т.е. если это произошло, то значит в других ячейках веса путей не меньше, чем в финишной, а значит, с меньшим весом путь до финишной ячейки уже никак не может быть (случай с отрицательными весами ребер исходного графа мы здесь не рассматриваем))
            if(CellFromQueue.second==std::make_pair(finishi,finishj))
            {
                IsFinishReached=true;
                break;
            }

            //Просматриваем всех соседов извлеченной ячейки.
            for(int alpha=0; alpha<4; alpha++)
            {
                int NeighborI=MazeWeighted.GetNeighborI(CellFromQueue.second.first, alpha);
                int NeighborJ=MazeWeighted.GetNeighborJ(CellFromQueue.second.second, alpha);
             //
std::cout<<"NeighborI="<<NeighborI<<"NeighborJ"<<NeighborJ<<std::endl;
                if(!MazeWeighted.HasWall(CellFromQueue.second.first, CellFromQueue.second.second, alpha) ) //Если у текущая ячейка не отделена от соседней стеной
                {

if(PathCoordWeights.at(NeighborI).at(NeighborJ).first==NOTVISITED) //Если этот сосед не был посещен ни разу
                    {

PathCoordWeights.at(NeighborI).at(NeighborJ).first=alpha; //устанавливаем путевые координаты

PathCoordWeights.at(NeighborI).at(NeighborJ).second=PathCoordWeights.at(CellFromQueue.second.first).at(CellFromQueue.second.second).second +
```

```
MazeWeighted.Weights.at(NeighborI).at(NeighborJ);    //Вес    пути    в
соседней ячейке равен весу пути в текущей ячейке+вес исходный в
соседней ячейке

PriorQueue.emplace(std::make_pair(PathCoordWeights.at(NeighborI).at(Ne
ighborJ).second+HeuristicAStar(NeighborI,      NeighborJ,      finishi,
finishj), std::make_pair(NeighborI, NeighborJ))); //Помещаем в очередь
(вес_он_же_приоритет, (i, j))
                    }
                    else //Если сосед уже был посещен
                    {
                        int
NewPathWeight=PathCoordWeights.at(CellFromQueue.second.first).at(CellF
romQueue.second.second).second                                        +
MazeWeighted.Weights.at(NeighborI).at(NeighborJ);

if(NewPathWeight<PathCoordWeights.at(NeighborI).at(NeighborJ).second)
//Если вес пути оказался меньше, чем уже найденный
                        {
                            //Делаем то же самое

PathCoordWeights.at(NeighborI).at(NeighborJ).first=alpha;
//устанавливаем путевые координаты

PathCoordWeights.at(NeighborI).at(NeighborJ).second=NewPathWeight;
//Вес пути в соседней ячейке равен весу пути в текущей ячейке+вес
исходный в соседней ячейке

PriorQueue.emplace(std::make_pair(PathCoordWeights.at(NeighborI).at(Ne
ighborJ).second+HeuristicAStar(NeighborI,      NeighborJ,      finishi,
finishj), std::make_pair(NeighborI, NeighborJ))); //Помещаем в очередь
(вес_он_же_приоритет, (i, j))
                        }
                    }
                }
            }
        }

        if(PrHandler.Mode==1)
        {
            //Вывод изначальных весов ячеек.
        //    MazeWeighted.WeightsToValues();
            std::cout<<"Weights:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout",1,     2,     true,
MazeWeighted.Weights);

            //Наглядное представление путевых весов
           /* for(int i=0; i<MazeWeighted.n; i++)
            {
                for(int j=0; j<MazeWeighted.m; j++)
                {
                    MazeWeighted.SetCellValue(i,                    j,
PathCoordWeights.at(i).at(j).second%10+48);
```

```
                //              if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                //              else   MazeWeighted.SetCellValue(i,   j,
PathCoordWeights.at(i).at(j).first);
                }
            }*/
            std::cout<<"PathWeights % 1000:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout",   1,   2,   true,
PathCoordWeights, 2);

            //Наглядное представление путевых координат
            for(int i=0; i<MazeWeighted.n; i++)
            {
                for(int j=0; j<MazeWeighted.m; j++)
                {
                //              if(PathCoordWeights.at(i).at(j).first>=0
&&PathCoordWeights.at(i).at(j).first<=9)   MazeWeighted.SetCellValue(i,
j, PathCoordWeights.at(i).at(j).first+48);
                //              else   MazeWeighted.SetCellValue(i,   j,
PathCoordWeights.at(i).at(j).first);
                    switch (PathCoordWeights.at(i).at(j).first)
                    {
                    case 0:
                        MazeWeighted.SetCellValue(i, j, '>');
                        break;
                    case 1:
                        MazeWeighted.SetCellValue(i, j, '^');
                        break;
                    case 2:
                        MazeWeighted.SetCellValue(i, j, '<');
                        break;
                    case 3:
                        MazeWeighted.SetCellValue(i, j, 'v');
                        break;
                    case 4:
                        MazeWeighted.SetCellValue(i, j, 's');
                        break;
                    default:
                        MazeWeighted.SetCellValue(i,                j,
PathCoordWeights.at(i).at(j).first);
                        break;
                    }
                }
            }
            std::cout<<"PathCoordinates:"<<std::endl;
            MazeWeighted.ShowDecorate((char*)"cout", 1, 2, true);
        }

        //Проверяем, достигли ли мы финишной ячейки
        if(!IsFinishReached)
        {
```

```
            std::cout<<"The path from ("<<starti<<", "<<startj<< ") to
("<<finishi<<", "<<finishj<< ") does not exist!"<<std::endl;
            return;
        }
        if(PrHandler.Mode==1)
        {
            std::cout<<"The finish cell is reached!"<<std::endl;
        }
        DijkstraReconstructWay(MazeWeighted,          PathCoordWeights,
finishi, finishj, starti, startj, Path);
    }

    #endif /* MAZESEARCHALG_H */
```

6.5. StatisticsMaze.inl

```
    #ifndef STATISTICSMAZE_INL
    #define STATISTICSMAZE_INL

    /*
    #include <limits> //для определения максимального числа в типе
    */

    #include <string>
    #include <time.h> //для измерения времени
    #include <fstream> //для файлов
    #include <iomanip>      // std::setprecision
    #include <random>

    #include "maze.h"
    #include "presenthandler.h"

    //PresentHandler.Mode11 выводить лабиринты в терминал

    class StatisticsMaze
    {
    private:
        int CurrentN=0;
        int CurrentM=0;
        int NStart=0, NEnd=0, NStep=0;
        double MRatio=1; // M/N отношение ширины к длине
        int NumberOfRuns=1; //число прогонов для текущих значений N M

    public:
        template<typename... CallBackParamsTail> //CallBackGenerate m
WeightMax
        StatisticsMaze(int NNStart, int NNEnd, int NNStep, double
NMRatio, int NNumberOfRuns, std::default_random_engine& generator,
presenthandler&    PresentHandler,    std::string    filename,    void
(*CallBackGenerate)(maze&,                std::default_random_engine&,
presenthandler&,    CallBackParamsTail&...    ),    CallBackParamsTail&
...callbackparamstail): NStart(NNStart), NEnd(NNEnd), NStep(NNStep),
MRatio(NMRatio), NumberOfRuns(NNumberOfRuns)
        {
```

```
            printLabel(filename);
            for(CurrentN=NStart;    CurrentN<=NEnd;    CurrentN+=NStep)
//Цикл табуляции по высоте лабиринта N
            {
                double timeSeconds=0;
                for(int k=0; k<NumberOfRuns; k++) //Число прогонов для
данной высоты лабиринта N
                {
                    CurrentM= CurrentN*MRatio;
                    maze Maze(CurrentN, CurrentM);

                    std::cout<<"Maze generating started"<<std::endl;
                    clock_t timeStart=clock(); //число тиков с начала
выполнения программы
                    CallBackGenerate(Maze,  generator,  PresentHandler,
callbackparamstail...); //генерируем лабиринт
                    clock_t timeEnd=clock();
                    std::cout<<"Maze generating ended"<<std::endl;
                    if(PresentHandler.Mode==11) Maze.ShowDecorate();

                    timeSeconds+=double(timeEnd-
timeStart)/CLOCKS_PER_SEC; //время работы алгоритма в секундах
                    std::cout<<"CurrentN="<<  CurrentN  <<";  Iteration
"<<k<<" is complete"<<std::endl;
                }
                timeSeconds/=NumberOfRuns;
                printValue(CurrentN, timeSeconds, filename);
            }
        }

        void  printLabel(std::string  filename)  //пишем  подпись  оси
высоты лабиринта в файл
        {
            std::ofstream                                    fd(filename,
std::ios_base::out|std::ios_base::trunc); //для записи очистив
            fd<<   "#filename="<<   filename<<";   MRatio="<<MRatio<<";
NumberOfRuns="<<NumberOfRuns<<std::endl;
            fd.close();
        }
        void  printValue(int  CurrentSize,  double  value,  std::string
filename) //пишем время выполнения в файл
        {
            std::ofstream   fd(filename,   std::ios_base::app);   //для
записи в конец

            fd<<               CurrentN<<               "\t"<<std::fixed
<<std::setprecision(5)<<value<<std::endl;
           // fd<<std::endl;
            fd.close();
        }
    };
    /*
    int CurrentSize=0; //текущее число вершин в графе
```

```
        int CurrentM=0;

        int NStart=0, NEnd=0, NStep=0;
        double MRatio=0; //Отношение числа ребер к максимально возможному для данного графа (будет округляться до целого числа m)
        int WeightMax=0; //максимальный вес ребер графа
        int NumberOfRuns=1; //число прогонов для каждого размера(числа вершин) графа



    //в callback будут Vertices Edges SourceVert DestVert  параметры для результатов работы MPath MPathLenght для ФлойдаУоршелла  Lenght для Дейкстры и т.д.  PresentHandler
        template<typename... CallBackParamsTail> //CallBackGenerate m WeightMax
        StatisticsGraph(int NNStart, int NNEnd, int NNStep, double NMRatio, int NWeightMax, int NNumberOfRuns, std::default_random_engine& generator, presenthandler& PresentHandler, std::string filename, void (*CallBackGenerate)(Graph&, int, int, std::default_random_engine&, presenthandler&), void(*CallBackFind)(Graph&, CallBackParamsTail&...), void(*CallBackTailHandler)(int N, CallBackParamsTail&...), CallBackParamsTail& ...callbackparamstail): NStart(NNStart), NEnd(NNEnd), NStep(NNStep), MRatio(NMRatio), WeightMax(NWeightMax), NumberOfRuns(NNumberOfRuns)
        {
            printLabel(filename);

            for(CurrentSize=NStart; CurrentSize<=NEnd; CurrentSize+=NStep) //Цикл табуляции по числу вершин N
            {
                double timeSeconds=0;

                for(int k=0; k<NumberOfRuns; k++) //Число прогонов для данного числа вершин N
                {
                    Graph Graph1(CurrentSize);
                    CurrentM= (double)(Graph1.Edges.size()-1+0)*Graph1.Edges.size()/2 * MRatio; //максимально возможное количество ребер на интересующий нас процент
                    std::cout<<"Graph generating started"<<std::endl;
                    CallBackGenerate(Graph1, CurrentM, WeightMax, generator, PresentHandler); //генерируем подходящий граф
                    std::cout<<"Graph generating ended"<<std::endl;
                    if(PresentHandler.Mode>=1) Graph1.ShowPlot(false, "Pic");

                    CallBackTailHandler(CurrentSize, callbackparamstail...); //Управляемся с "лишними" параметрами функции поиска. В частности, изменяем размеры их массивов.
                    clock_t timeStart=clock(); //число тиков с начала выполнения программы
```

```
                    CallBackFind(Graph1, callbackparamstail...);
                    clock_t timeEnd=clock();

                    timeSeconds+=double(timeEnd-
timeStart)/CLOCKS_PER_SEC; //время работы алгоритма в секундах

                    std::cout<<"CurrentSize="<<    CurrentSize    <<";
Iteration "<<k<<" is complete"<<std::endl;
                }
                timeSeconds/=NumberOfRuns;
                printValue(CurrentSize, timeSeconds, filename);
            }
        }
        void  printLabel(std::string  filename)  //пишем  подпись  оси
размера массива в файл
        {
            std::ofstream                                    fd(filename,
std::ios_base::out|std::ios_base::trunc); //для записи очистив
            fd<<   "#filename="<<   filename<<";   MRatio="<<MRatio<<";
WeightMax="<<WeightMax<<"; NumberOfRuns="<<NumberOfRuns<<std::endl;
            fd.close();
        }
        void  printValue(int  CurrentSize,  double  value,  std::string
filename) //пишем время выполнения в файл
        {
            std::ofstream   fd(filename,   std::ios_base::app);   //для
записи в конец

            fd<<             CurrentSize<<             "\t"<<std::fixed
<<std::setprecision(5)<<value<<std::endl;
           // fd<<std::endl;
            fd.close();
        } */

    #endif //STATISTICSMAZE_INL
```

6.6. StatisticsMazeSearch.inl

```
#ifndef STATISTICSMAZE_INL
#define STATISTICSMAZE_INL

/*
#include <limits> //для определения максимального числа в типе
*/

#include <string>
#include <time.h> //для измерения времени
#include <fstream> //для файлов
#include <iomanip>      // std::setprecision
#include <random>

#include "maze.h"
#include "presenthandler.h"
```

```
    //PresentHandler.Mode11 выводить лабиринты в терминал

    class StatisticsMazeSearch
    {
    private:
        int CurrentN=0;
        int CurrentM=0;
        int NStart=0, NEnd=0, NStep=0;
        double MRatio=1; // M/N отношение ширины к длине
        int NumberOfRuns=1; //число прогонов для текущих значений N M

    public:
        StatisticsMazeSearch()=default;

     //    template<typename... CallBackParamsTail> //CallBackGenerate
m WeightMax
        void StatisticsMazeSearchFunc1()
        {

        }

     //    template<typename... CallBackParamsTail> //CallBackGenerate
m WeightMax
        template <typename T>
        void  StatisticsMazeSearchFunc(int  NNStart,  int  NNEnd,  int
NNStep, double NMRatio, int NNumberOfRuns, std::default_random_engine&
generator, presenthandler& PresentHandler, std::string filename, void
(*CallBackSearch)(T&,  int  starti,  int  startj,  int  finishi,  int
finishj,   std::vector<std::pair<int,int>>&   Path,   presenthandler&
PrHandler))
        {
            NStart=NNStart;  NEnd=NNEnd;  NStep=NNStep;  MRatio=NMRatio;
NumberOfRuns=NNumberOfRuns;
            printLabel(filename);
            for(CurrentN=NStart;   CurrentN<=NEnd;   CurrentN+=NStep)
//Цикл табуляции по высоте лабиринта N
            {
                CurrentM= CurrentN*MRatio;
                std::uniform_int_distribution<int>
undistrStartFinishI(0, CurrentN-1);
                std::uniform_int_distribution<int>
undistrStartFinishJ(0, CurrentM-1);
                double timeSeconds=0;
                for(int k=0; k<NumberOfRuns; k++) //Число прогонов для
данной высоты лабиринта N
                {
                    mazeWeighted MazeWeighted(CurrentN, CurrentM, 1);

                    //Генерируем лабиринт без недостижимых областей с
циклами. У ячеек устанавливаем веса от 1 до 10.
                    std::cout<<"Maze generation started"<<std::endl;
                //            WilsonReduced(MazeWeighted,   generator,
PresentHandler, 0.3);
```

```
                    int alpha=0;
                    BinaryTree(MazeWeighted,                    generator,
PresentHandler, alpha);
                    WallsReduce(MazeWeighted, 0.3, generator);
                    RandomCircules(MazeWeighted,   generator,   2,   10,
0.05, 1, 0.5); //1, 3, 0.03, 2, 0.5
                    std::cout<<"Maze generation ended"<<std::endl;
                    if(PresentHandler.Mode==11)
MazeWeighted.ShowDecorate((char*)"cout",       1,       2,       true,
MazeWeighted.Weights);

                    //Запускаем алг поиска по лабиринту
                    int starti=undistrStartFinishI(generator);
                    int startj=undistrStartFinishJ(generator);
                    int finishi=undistrStartFinishI(generator);
                    int finishj=undistrStartFinishJ(generator);
                    if(PresentHandler.Mode==11)
                    {
                        std::cout<<"start=("<<starti<<",  "<<startj<<")
finish=("<<finishi<<", "<<finishj<<")"<<std::endl;
                    }
                    std::vector<std::pair<int,int>> Path;

                    std::cout<<"Maze search started"<<std::endl;
                    clock_t  timeStart=clock(); //число тиков с начала
выполнения программы
                    CallBackSearch(MazeWeighted,    starti,    startj,
finishi, finishj, Path, PresentHandler);
                    clock_t timeEnd=clock();
                    std::cout<<"Maze search ended"<<std::endl;
                    if(PresentHandler.Mode==11)
                    {
                        PrintVector(Path, "Path");
                    }

                    timeSeconds+=double(timeEnd-
timeStart)/CLOCKS_PER_SEC; //время работы алгоритма в секундах
                    std::cout<<"CurrentN="<<  CurrentN  <<";  Iteration
"<<k<<" is completed"<<std::endl;
                }
                timeSeconds/=NumberOfRuns;
                printValue(CurrentN, timeSeconds, filename);
            }
        }

        void  printLabel(std::string  filename)  //пишем  подпись  оси
высоты лабиринта в файл
        {
            std::ofstream                                   fd(filename,
std::ios_base::out|std::ios_base::trunc); //для записи очистив
            fd<<   "#filename="<<   filename<<";   MRatio="<<MRatio<<";
NumberOfRuns="<<NumberOfRuns<<std::endl;
            fd.close();
```

```
        }
        void printValue(int CurrentSize, double value, std::string
filename) //пишем время выполнения в файл
        {
            std::ofstream fd(filename, std::ios_base::app); //для
записи в конец

            fd<< CurrentN<< "\t"<<std::fixed
<<std::setprecision(5)<<value<<std::endl;
           // fd<<std::endl;
            fd.close();
        }
    };

    #endif //STATISTICSMAZE_INL
```

6.7. Funcs.h

```
    #ifndef FUNCS_H
    #define FUNCS_H

    #include <iostream>
    #include <sstream>
    #include <fstream>
    #include <string>

    #include "maze.h"
    #include "MazeGenerationAlgs.h"
    #include <iomanip>


    void MazeFromFile(maze& Maze, char* filename)
    {
        std::ifstream fin(filename);
        if(!fin.is_open()) //проверка на успешность открытия файла
        {
            std::stringstream ss;
            ss << "Can not open file:"<<filename<<std::endl;
            throw(ss.str());
        }
        std::string str;
        std::stringstream ss;
        int n=0, m=0; //определяем размеры лабиринта
        while(std::getline(fin, str)) n++;
        if (n%2==0)
        {
            std::cout<<"MazeFromFile: bad n: n%2==0"<<std::endl;
            getchar();
        }
        n=(n-1)/2;
        fin.clear();
        fin.seekg(0, std::ios_base::beg);
        std::getline(fin, str);
        m=str.length();
```

```
        if (m%2==0)
        {
            std::cout<<"MazeFromFile: bad m: m%2==0"<<std::endl;
            getchar();
        }
        m=(m-1)/2;
        Maze=maze(n,m);
        fin.clear();
        fin.seekg(0, std::ios_base::beg);
        for(int i=0; i<n*2+1; i++)
        {
           std::getline(fin, str);
           for(int j=0; j<m*2+1; j++)
           {
               Maze.BaseVector.at(i).at(j)=str.at(j);
           }
        }
        fin.close();

    }
    template<typename T>
    void PrintMatrix(std::vector<std::vector<T>>& Matrix, const char*
MatrixName)
    {
        typename  std::vector<std::vector<T>>::iterator  iteri;  //два
итератора, для итерации по строкам и ячейкам(столбцам) .Без typename
вообще не компилируется, хотя казалось бы.
        typename std::vector<T>::iterator iterj;
        for(iteri=Matrix.begin(); iteri!=Matrix.end(); iteri++)
        {
            std::cout<<  MatrixName<<"["<<std::distance(Matrix.begin(),
iteri) <<"]=";
            for(iterj=iteri->begin(); iterj!=iteri->end(); iterj++)
            {
                std::cout<< (*iterj)<<"\t";
            }
            std::cout<<std::endl;
        }
        std::cout<<"----------"<<std::endl;
    }
    //Print будем перегружать для разных случаев
    void Print(std::pair<int,int> Pair);
    void Print(std::pair<int, std::pair<int, int>> Item);
    template<typename T>
    void PrintVector(std::vector<T>& Vector, const char* VectorName)
    {
        typename std::vector<T>::iterator iteri=Vector.begin();
            std::cout<< VectorName<<"=";
            for(iteri=Vector.begin(); iteri!=Vector.end(); iteri++)
            {
                std::cout<<std::setw(3);
            //    std::cout<< (*iteri)<<"\t";
            //    std::cout<< (*iteri);
```

```
            Print(*iteri);
        }
        std::cout<<std::endl;
//    std::cout<<"----------"<<std::endl;
}
void Print(std::pair<int,int> Pair)
{
    std::cout<<"("<<Pair.first<<", "<<Pair.second<<")";
}
void Print(std::pair<int, std::pair<int, int>> Item)
{
    std::cout<<"("<<Item.first<<",        ("<<Item.second.first<<",
"<<Item.second.second<<"))";
}
#endif /* FUNCS_H */
```

6.8. presenthandler.h

```
#ifndef PRESENTHANDLER_H_INCLUDED
#define PRESENTHANDLER_H_INCLUDED

#include <string>
class   presenthandler   //Класс   для   управления   наглядным
представлением шагов, совершаемых внутри функций и прочих.
{
private:
    int  CurrentFileNumber=0;  //Количество  уже  сгенерированных
файлов.
public:
    int Mode=0; //0 -не влиять на выполнение вызывающей функции; 1
-выводить шаги в командную строку; 2 -выводить шаги в файл(ы)

    std::string GetFileNumberAndIncrease()
    {
        CurrentFileNumber++;
        return std::to_string(CurrentFileNumber);
    }
    void ResetFileNumber()
    {
        CurrentFileNumber=0;
    }
};



#endif //PRESENTHANDLER_H_INCLUDED
```

6.9. plothandler.h

```
#ifndef PLOTHANDLER_H
#define PLOTHANDLER_H
//Имена файлов, из которых берутся данные для графиков

#include <string>
#include <vector>
```

```
#include <fstream>

class plothandler
{
private:
    std::vector<std::string> v;
public:
    std::string addandreturn(std::string filename)
    {
        v.push_back(filename);
        return filename;
    }
    void tofile(std::string filename="Plotfilelist.txt")
    {
        std::ofstream fd(filename);
        fd<<"filename=\t\"";
        for(auto& i: v)
        {
            fd<<i<<" ";
        }
        fd<<"\"";
        fd.close();
    }
};


#endif /* PLOTHANDLER_H */
```

6.10. PlotScript.bash

```
#! /usr/bin/bash
filelistVar=$(gawk -F '\t' '{printf "%s", $2}' Plotfilelist.txt)


cat > PlotScript.gpi << HEREDOC1
#! /usr/bin/gnuplot -persist

# Будем этим скриптом строить графики
#Тут неизменяемые настройки
title_f(f,a,u,v,b)=  sprintf('%s=%.12f*x*log2(%.12fx+%.12f)+%.12f',
f, a, u,v, b)

set grid
set datafile separator '\t'

#set terminal png size 1024, 768
#set output "LabPlot.png"
#set terminal svg enhanced size 3,2
#set output 'file.svg'

#Этот способ рабочий. Потом нужно преобразовать к .PNG с помощью
команды. and then use imagemagick to convert the vector format to a
.png:
```

```
    #convert -verbose -density 300 -trim file.pdf -quality 100 -
flatten -sharpen 0x1.0 file.png
    #set terminal pdfcairo size 6, 5 #этот неплохо работает. Размер в
дюймах
    #set output 'file.pdf'

    #для графиков данных
    set linestyle 1 lc rgb "#0000FF" lw 2.5
    set linestyle 2 lc rgb "#00FF00" lw 2.5
    set linestyle 3 lc rgb "#FF0000" lw 2.5
    set linestyle 4 lc rgb "#00FFFF" lw 2.5
    set linestyle 5 lc rgb "#FF00FF" lw 2.5
    set linestyle 6 lc rgb "#FFFF00" lw 2.5
    set linestyle 7 lc rgb "#00000F" lw 2.5
    set linestyle 8 lc rgb "#000F00" lw 2.5
    set linestyle 9 lc rgb "#0F0000" lw 2.5
    set linestyle 10 lc rgb "#0000F0" lw 2.5
    #для линий треда
    set linestyle 11 lc rgb "#000080" lw 1.25 dashtype 2
    set linestyle 12 lc rgb "#008000" lw 1.25 dashtype 2
    set linestyle 13 lc rgb "#800000" lw 1.25 dashtype 2


    #Тут изменяемые настройки
    #Положение легенды
    #set key left
    set key below

    set title "Lab6" font "Helvetica Bold, 14"
    set xlabel "N - maze height, cells"
    set ylabel "Time - average executing time, seconds"

    set yrange [0:*]
    #set xrange[-pi:pi]


    #plot sin(x) title "sinux"  lc rgb "red", cos(x)  title "cosinus"
lc rgb "green"
    #plot "HeapSortOut.txt" w lp lc 3 pt 7 ps 30 smooth bezier
    #plot "HeapSortOut.txt" w lp lc 2 pt 1 ps 2 smooth acsplines

    # The equation log(x) -логарифм натуральный
    #f1(x) = a1*x*log(u1*x+v1)/log(2)+b1
    #f1(x)=(a1*x*log(u1*x))/log(2)+b1
    # The fit
    #fit f1(x) "HeapSortOut.txt" u 1:2 via a1,b1, u1

    #f2(x) = a2*x*log(u2*x+v2)/log(2)+b2
    #fit f2(x) "QuickSortHoareOut.txt" u 1:2 via a2,b2, u2, v2

    #f3(x) = a3*x*log(u3*x+v3)/log(2)+b3
    #fit f3(x) "ShellSortOut.txt" u 1:2 via a3,b3, u3, v3
```

```
    #listsize=5

    array filelist1[5]
    filelist=$filelistVar
    i=1
    do for [file in filelist] {
    #filelist(t)="FloydWarshellOut.txt"
    #filelist(t)=gawk 'FNR=t{printf "%s", }' filelist.txt
    filelist1[i]=sprintf('%s', file);

    #plot file using 1:2 smooth csplines with lines linestyle i

    i=i+1
    }



    #plot \    "FloydWarshellOut.txt" using 1:2 notitle smooth csplines
with lines linestyle 1, \
    #    1 / 0 title "FloydWarshell" with lines linestyle 1
    #     f1(x) title title_f("f1", a1,u1, 0,b1) with lines linestyle
11,\
    #     \
    #     "QuickSortHoareOut.txt" using 1:2 notitle smooth csplines
with lines linestyle 2, \
    #     1 / 0 title "QuickSortHoare" with lines linestyle 2, \
    #     f2(x) title title_f("f2",a2, u2, v2,b2) with lines linestyle
12, \
    #     \
    #     "ShellSortOut.txt" using 1:2 notitle smooth csplines with
lines linestyle 3, \
    #     1 / 0 title "ShellSort" with lines linestyle 3, \
    #      f3(x)  title  title_f("f3",  a3,u3,  v3,  b3)  with  lines
linestyle 13

    plot for [j=1:i-1] filelist1[j] using 1:2 smooth csplines with
lines linestyle j title filelist1[j]



    #Справка
    # \ позволяет записать одну строку как несколько
    #  -persist позволяет держать окно открытым после завершения
скрипта
    #
    #
    #
    HEREDOC1
    chmod +x PlotScript.gpi
    ./PlotScript.gpi
```

6.11. maze_debug_main.cpp

```
    #include <iostream>
```

```
#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include <random>
#include "presenthandler.h"

int main(int, char**) {
 //   int seed=time(0);
    int seed=2;
    std::default_random_engine generator1(seed);
    presenthandler PrHandler;
    PrHandler.Mode=0;

    maze Maze(6,6);
 /*   for(int i=0; i<Maze.n; i++)
    {
        for(int j=0; j<Maze.m; j++)
        {
            Maze.SetCellWalls(i,j, 1, false, true);
            Maze.SetCellWalls(i,j, 2, false, true);
        }
    }
*/
 /*   try
    {
         MazeFromFile(Maze, (char*)"Maze.txt");
    }
    catch(std::string str)
    {
        std::cout << str << std::endl;
    }
*/
  //  WallsReduce(Maze, 0.5, generator1);
//    WilsonReduced(Maze, generator1, PrHandler, 0.3);
/*    Maze.SetCellWalls(0,0,3, false);
    Maze.SetCellWalls(0,1,3, false);
    Maze.SetCellWalls(1,0,3, false);
    Maze.SetCellWalls(1,1,3, false);
    Maze.SetCellWalls(1,0,0, false);
    Maze.SetCellValue(1,1, '@');*/

 //   Maze.ShowDecorate((char*)"MazeOut.txt");
    Wilson(Maze, generator1, PrHandler);
    Maze.ResetValues();
    std::stringstream ss;
    std::string str("maze  by me");
    char ch='N';
    int i=0, j=0;
/*    for(int k=0; (k<str.size())&& i<Maze.n && j<Maze.m; k++)
    {
        ch=str.c_str()[k];
```

```
        //    str.erase();
        //   str.pop_back();
            if (ch!=' ') Maze.SetCellValue(i,j, ch);
            j++;
            i+=j/Maze.m;
            j=j%Maze.m;

        }
        */
       Maze.SetValuesByStr(str);
        Maze.ShowDecorate((char*)"cout",0, 2, true);
     //   std::cout << ch<<std::endl;
    }
```

6.12. mazeWeighted_debug_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include <random>
#include "presenthandler.h"

int main(int, char**) {
  //  int seed=time(0);
    int seed=1;
    std::default_random_engine generator1(seed);
    presenthandler PrHandler;
    PrHandler.Mode=0;

    mazeWeighted MazeWeighted(3,3);
//    WeightCircule(MazeWeighted, 5, 5, 5, 2);
 //   for(int k=0; k<10; k++)
//    WilsonReduced(MazeWeighted, generator1, PrHandler, 0.3);
//       RandomCircules(MazeWeighted, generator1, 1, 3, 0.03, 2,
0.5);

//    MazeWeighted.WeightsToValues();
    MazeWeighted.ShowDecorate((char*)"cout",1, 2, true);
 //   std::cout << ch<<std::endl;
}
```

6.13. Wilson_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"

int main(int, char**) {
 //   int seed=time(0);
```

```
    int seed=2;
    std::default_random_engine generator1(seed);
    maze Maze(500,500);
 /*   try
    {
         MazeFromFile(Maze, (char*)"Maze.txt");
    }
    catch(std::string str)
    {
        std::cout << str << std::endl;
    }
*/
    presenthandler PrHandler;
    PrHandler.Mode=1;
//    Wilson(Maze, generator1, PrHandler);
    WilsonSerial(Maze, generator1, PrHandler);
    Maze.ShowDecorate((char*)"MazeOut.txt",0);
    Maze.ShowDecorate();
}
```

6.14. AldousBroder_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"

int main(int, char**) {
    int seed=time(0);
    //int seed=2;
    std::default_random_engine generator1(seed);

    maze Maze(3,3);
 /*   try
    {
         MazeFromFile(Maze, (char*)"Maze.txt");
    }
    catch(std::string str)
    {
        std::cout << str << std::endl;
    }
*/
    presenthandler PrHandler;
    PrHandler.Mode=2;
    AldousBroder(Maze, generator1, PrHandler);
    Maze.ShowDecorate((char*)"cout", 1, 2, true);
//    Maze.ShowDecorate((char*)"MazeOut.txt");
 //   Wilson(Maze, generator1, PrHandler);

}
```

6.15. BinaryTree_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"

int main(int, char**) {
    int seed=time(0);
    //int seed=2;
    std::default_random_engine generator1(seed);
    maze Maze(3,3);
 /*   try
    {
         MazeFromFile(Maze, (char*)"Maze.txt");
    }
    catch(std::string str)
    {
        std::cout << str << std::endl;
    }
*/
    presenthandler PrHandler;
    PrHandler.Mode=2;
    int alpha=0;
    BinaryTree(Maze, generator1, PrHandler, alpha);
 //   Maze.ShowDecorate();
   Maze.ShowDecorate();
}
```

6.16. Lee_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include <random>
#include "presenthandler.h"
#include "MazeSearchAlg.h"
#include <utility>

int main(int, char**) {
 //   int seed=time(0);
    int seed=2;
    std::default_random_engine generator1(seed);
    presenthandler PrHandler;
    PrHandler.Mode=1;

    maze Maze(3,3);

//    WallsReduce(Maze, 0.5, generator1);
```

```
   WilsonReduced(Maze, generator1, PrHandler, 0.3);
   std::vector<std::pair<int,int>> Way;
    std::vector<int> Way1(10, 33);
   Lee(Maze, 0, 1, 1, 1, Way, PrHandler);
   PrintVector(Way, "Way");
   //   Maze.ShowDecorate((char*)"cout",0);
 //   Maze.ShowDecorate((char*)"MazeOut.txt");
 //   Wilson(Maze, generator1, PrHandler);

}
```

6.17. Lee2Waves_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include <random>
#include "presenthandler.h"
#include "MazeSearchAlg.h"
#include <utility>

int main(int, char**) {
 //   int seed=time(0);
    int seed=2;
    std::default_random_engine generator1(seed);
    presenthandler PrHandler;
    PrHandler.Mode=1;

    maze Maze(4,5);
//   MazeFromFile(Maze, (char*)"Maze.txt");
//    WallsReduce(Maze, 0.5, generator1);
   WilsonReduced(Maze, generator1, PrHandler, 0.3);
   std::vector<std::pair<int,int>> Way;
    std::vector<int> Way1(10, 33);
   Lee2Waves(Maze, 0, 1, 2, 4, Way, PrHandler);
   PrintVector(Way, "Way");
//   Maze.ShowDecorate((char*)"cout",0);
 //   Maze.ShowDecorate((char*)"MazeOut.txt");
 //   Wilson(Maze, generator1, PrHandler);

}
```

6.18. Dijkstra_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include <random>
#include "presenthandler.h"
```

```
    #include "MazeSearchAlg.h"

    int main(int, char**) {
    //  int seed=time(0);
        int seed=2;
        std::default_random_engine generator1(seed);
        presenthandler PrHandler;
        PrHandler.Mode=2;

        mazeWeighted MazeWeighted(5,5, 1);
        //Генерация лабиринта
        WilsonReduced(MazeWeighted, generator1, PrHandler, 0.3);
        RandomCircules(MazeWeighted, generator1, 2, 9, 0.05, 1, 0.5);
//1, 3, 0.03, 2, 0.5
        //MazeWeighted.SetCellWalls(0,0,0,true);
        //MazeWeighted.SetCellWalls(0,0,3,true);
        //Вывод весов лабиринта
    //    MazeWeighted.WeightsToValues();
    //    std::cout<<"Weights:"<<std::endl;
    //    MazeWeighted.ShowDecorate((char*)"cout",1, 2, true);

        //Запуск алгоритма
        std::vector<std::pair<int,int>> Path;
        Dijkstra(MazeWeighted, 0, 0, 2, 2, Path, PrHandler);
        PrintVector(Path, "Path");

    }
```

6.19. AStar_main.cpp

```
    #include <iostream>


    #include "maze.h"
    #include "Funcs.h"
    #include "MazeGenerationAlgs.h"
    #include <random>
    #include "presenthandler.h"
    #include "MazeSearchAlg.h"

    int main(int, char**) {
    //  int seed=time(0);
        int seed=2;
        std::default_random_engine generator1(seed);
        presenthandler PrHandler;
        PrHandler.Mode=1;

        mazeWeighted MazeWeighted(10,10, 1);
        //Генерация лабиринта
        WilsonReduced(MazeWeighted, generator1, PrHandler, 0.3);
        RandomCircules(MazeWeighted, generator1, 2, 9, 0.05, 1, 0.5);
//1, 3, 0.03, 2, 0.5
        //MazeWeighted.SetCellWalls(0,0,0,true);
        //MazeWeighted.SetCellWalls(0,0,3,true);
```

```
        //Вывод весов лабиринта
    //    MazeWeighted.WeightsToValues();
    //    std::cout<<"Weights:"<<std::endl;
    //    MazeWeighted.ShowDecorate((char*)"cout",1, 2, true);

        //Запуск алгоритма
        std::vector<std::pair<int,int>> Path;
        AStar(MazeWeighted, 0, 0, 2, 2, Path, PrHandler);
        PrintVector(Path, "Path");

    }
```

6.20. StatisticsMaze_main.cpp

```
    #include <iostream>


    #include "maze.h"
    #include "Funcs.h"
    #include "MazeGenerationAlgs.h"
    #include "StatisticsMaze.inl"

    #include "presenthandler.h"
    #include "plothandler.h"

    int main(int, char**) {
        int seed=time(0);
        //int seed=2;
        std::default_random_engine generator1(seed);

        presenthandler PrHandler;
        PrHandler.Mode=11; //11 выводить генерируемые лабиринты
        plothandler PlotHandler;

        int NStart=4, NEnd=50, NStep=5, NumberOfRun=100;
        double MRatio=1;
     //     StatisticsMaze(NStart, NEnd, NStep, MRatio, NumberOfRun,
generator1,    PrHandler,    PlotHandler.addandreturn("WilsonOut.txt"),
Wilson); //200x200 около 9 секунд генерируются
     //     StatisticsMaze(NStart, NEnd, NStep, MRatio, NumberOfRun,
generator1,                                                PrHandler,
PlotHandler.addandreturn("WilsonSerialOut.txt"), WilsonSerial);
    //       StatisticsMaze(NStart, NEnd, NStep, MRatio, NumberOfRun,
generator1,                                                PrHandler,
PlotHandler.addandreturn("AldousBroderOut.txt"), AldousBroder);
        int alpha=0; //угол, куда смотрит вырез стены
        StatisticsMaze(NStart,   NEnd,   NStep,   MRatio,   NumberOfRun,
generator1,  PrHandler,  PlotHandler.addandreturn("BinaryTreeOut.txt"),
BinaryTree, alpha);

        PlotHandler.tofile("Plotfilelist.txt");
        system("./PlotScript.bash");
    }
```

6.21. StatisticsMazeSearch_main.cpp

```
#include <iostream>


#include "maze.h"
#include "Funcs.h"
#include "MazeGenerationAlgs.h"
#include "MazeSearchAlg.h"
#include "StatisticsMazeSearch.inl"

#include "presenthandler.h"
#include "plothandler.h"

#include <thread>

int main(int, char**) {
    int seed=time(0);
    //int seed=2;
    std::default_random_engine generator1(seed), generator2(seed),
generator3(seed), generator4(seed);

    presenthandler PrHandler;
    PrHandler.Mode=0; //11 выводить генерируемые лабиринты
    plothandler PlotHandler;

    int NStart=4, NEnd=504, NStep=125, NumberOfRun=500;
    double MRatio=1;

    StatisticsMazeSearch                    StatisticsMazeSearch1,
StatisticsMazeSearch2, StatisticsMazeSearch3, StatisticsMazeSearch4;
    std::thread
thread1(&StatisticsMazeSearch::StatisticsMazeSearchFunc<mazeWeighted>,
std::ref(StatisticsMazeSearch1),   NStart,   NEnd,   NStep,   MRatio,
NumberOfRun,        std::ref(generator1),        std::ref(PrHandler),
PlotHandler.addandreturn("DijkstraOut.txt"), Dijkstra);
    thread1.join();
    std::thread
thread2(&StatisticsMazeSearch::StatisticsMazeSearchFunc<mazeWeighted>,
std::ref(StatisticsMazeSearch2),   NStart,   NEnd,   NStep,   MRatio,
NumberOfRun,        std::ref(generator2),        std::ref(PrHandler),
PlotHandler.addandreturn("AStarOut.txt"), AStar);
    thread2.join();
    std::thread
thread3(&StatisticsMazeSearch::StatisticsMazeSearchFunc<maze>,
std::ref(StatisticsMazeSearch3),   NStart,   NEnd,   NStep,   MRatio,
NumberOfRun,        std::ref(generator3),        std::ref(PrHandler),
PlotHandler.addandreturn("LeeOut.txt"), Lee);
    thread3.join();
    std::thread
thread4(&StatisticsMazeSearch::StatisticsMazeSearchFunc<maze>,
std::ref(StatisticsMazeSearch4),   NStart,   NEnd,   NStep,   MRatio,
```

```
NumberOfRun,         std::ref(generator4),         std::ref(PrHandler),
PlotHandler.addandreturn("Lee2WavesOut.txt"), Lee2Waves);
        thread4.join();

     //   thread1.join();
     //   thread2.join();
     //   thread3.join();
     //   thread4.join();

        PlotHandler.tofile("Plotfilelist.txt");
        system("./PlotScript.bash");
    }
```