**Документация по использованию Wikipedia Walker**

Приложение предназначенное для поиска количества путей между статьями, вывода кратчайшего пути между двумя статьями, а также вывода на экран графа всех путей. Предусмотрена возможность ограничить количество выводимых путей, возможность вывода путей 1,2,3 и максимальное число отображенных путей. А так же возможность отображения графа путей с возможностью перемещать вершины при необходимости. Приложение может сохранить данные в файл формата JSON.

**Графический интерфейс приложения:**

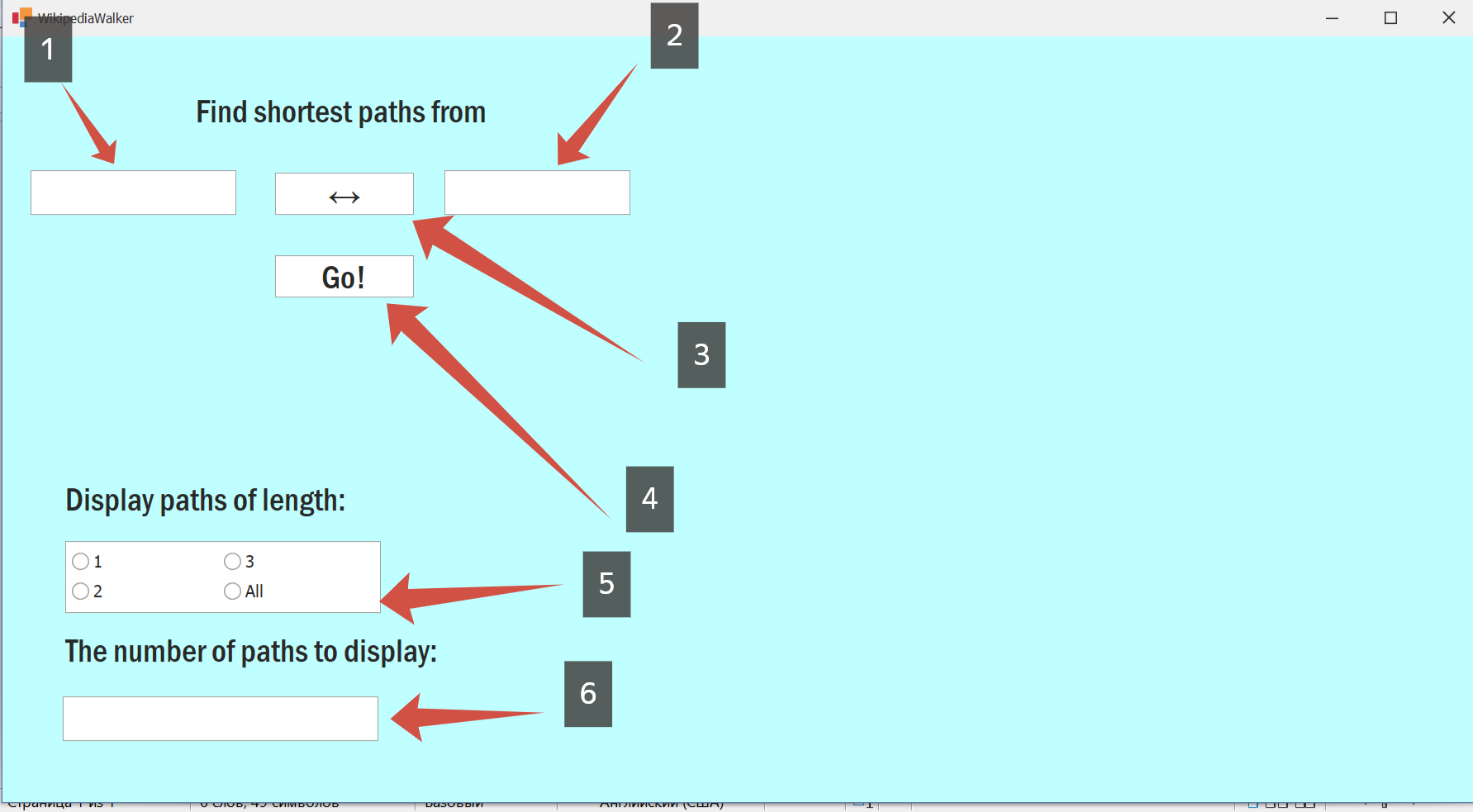


Рис 1. Окно приложения при запуске

1. Поле для ввода начальной статьи

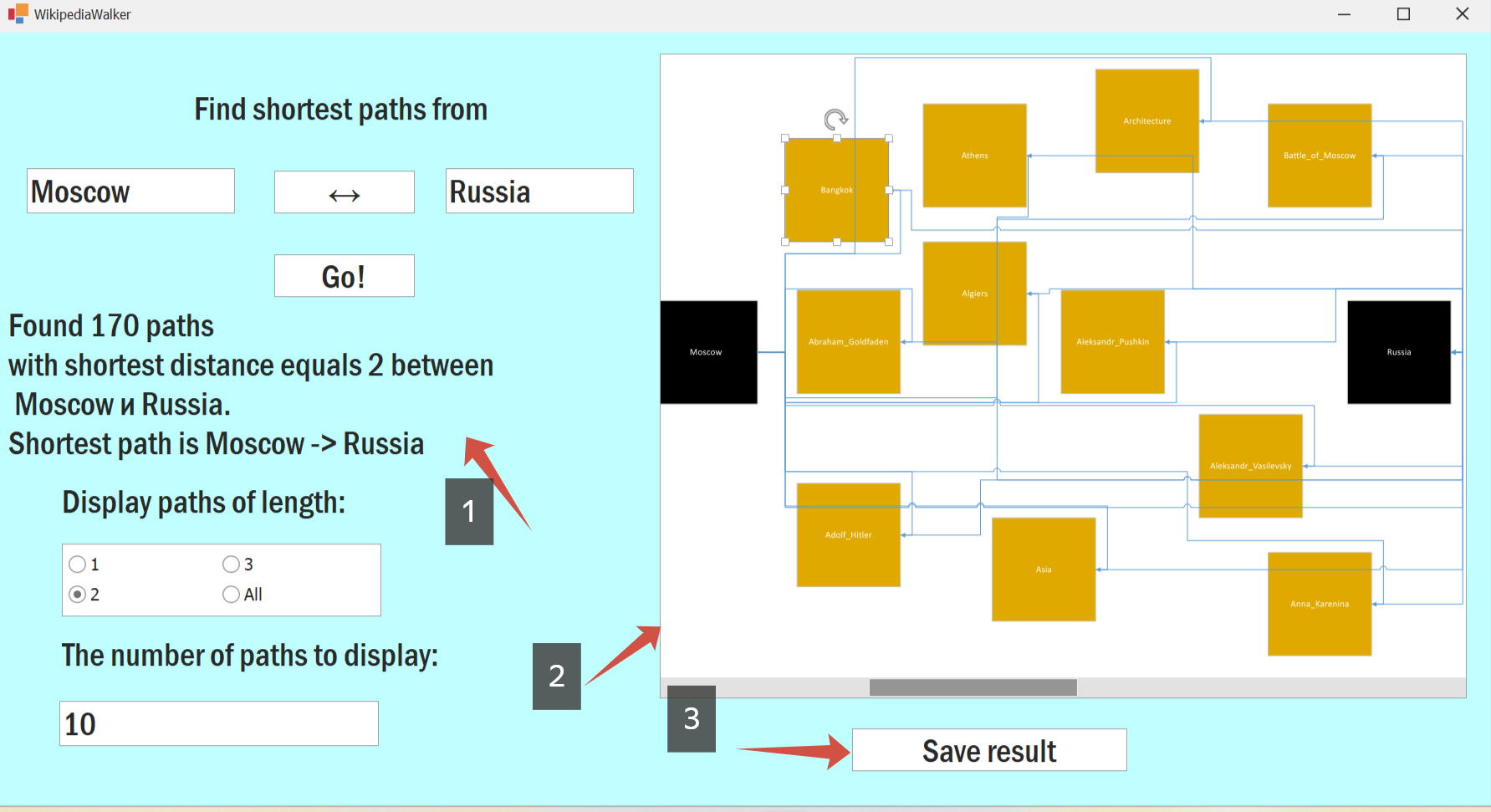
2. Поле для ввода конечной статьи

3. Кнопка, при нажатии на которую поменяются местами начальная и конечная статьи

4. Кнопка, при нажатии на которую запустится программа, то есть отобразится информация с количеством найденных путей, кратчайший путь и его длина

5. Поле с возможностью выбора максимальной длины пути для отображаемого графа

6. Поле с возможностью ввода количества отображаемых путей

Рис 2. Окно после того как пользователь ввел данные

1. Здесь выводятся результаты

2. Здесь выводится граф

3. Кнопка, при нажатии на которую откроется диалоговое окно сохранения файла.

**Использованные алгоритмы**

**Граф**

Представим некоторую точку, она может представлять собой условный город, некоторый узел или любой другой объект. Пусть на плоскости есть несколько таких точек. Теперь возьмем и соединим некоторые из них отрезками, отрезки будут символизировать дороги между городами или связи какого-либо другого рода.

Такие точки называются ****вершинами****, а отрезки – ****ребрами****

****Граф**** — это абстрактная математическая модель, состоящая из множества вершин и набора соединяющих их рёбер.

**Алгоритм Дейкстры**

Для нахождения кртачайшего пути в неориентированном графе воспользуемся алгоритмом Дейкстры

**Алгоритм Дейкстры** — это алгоритм позволяющий найти кратчайшие пути из заданной вершины до всех остальных вершин. Необходимым условием является то, что веса всех рёбер должны быть неотрицательными.

Идея алгоритма следующая:

1. Мы поддерживаем множество вершин distance, до которых мы уже знаем расстояние из начальной вершины. Полагаем, что до всех вершин, кроме начальной, путь равен бесконечности.
2. Далее мы выбираем из известного нам множества непосещенную вершину v, расстояние до которой distance[v] минимально.
3. Перебираем все вершины, в которые можно попасть из выбранной вершины v и обновляем расстояния до них, если они меньше, чем нам известные. То есть если из v можно попасть в вершину u по ребру длиной s, то distance[u] = min(distance[u], distance[v] + s).

**Поиск в глубину**

****Обход в глубину**** — стратегия обхода графа, которая заключается в том, что мы пытаемся продвигаться вглубь графа, насколько это возможно. В англоязычной литературе этот обход называется ****depth first search (dfs)****. На основе dfs работают алгоритмы поиска кратчайшего пути в графе, поиска компонент связности и многие другие. Об этих алгоритмах мы поговорим далее в нашем курсе, а сейчас мы изучим dfs.

Пошагово алгоритм представляет собой следующее:

1. Выбираем любую вершину из еще не посещенных, обозначим ее как u.
2. Запускаем обход в глубину (функцию dfs) от вершины u
   1. Помечаем вершину u как посещенную
   2. Для каждой не посещенной смежной с u вершиной vзапускаем dfs(v)
3. Повторяем шаги 1 и 2, пока все вершины не окажутся посещенными.

**Двунаправленный поиск**

Для построения графа был использован двунаправленный поиск

Двунаправленный поиск – это алгоритм поиска, который находит расстояние от исходной точки до конечной точки в ориентированном графе. Алгоритм запускает 2 одновременных поиск: от начальной точки к конечной и от конечной точки к начальной. Алгоритм заканчивает свою работу, когда оба поиска приходят в одну точку. Т.к. прямой и обратный поиск могут быть выполнены параллельно друг с другом в разных потоках приложения, чисто теоретически, это позволяет получить двукратное уменьшение времени затрачиваемого на поиск. Однако на практике существует ряд дополнительных проблем, которые связаны с таким ускорением алгоритма. Алгоритм двунаправленный поиска должен содержать дополнительную логику, которая будет принимать решение по какой из дуг перейти на следующем шаге. От этого выбора сильно зависит успешность алгоритма. Конечная точка должна быть жёстко задана, но на практике часто возникает необходимость задать конечную точку какими либо параметрами, в этом случае алгоритм двунаправленного поиска в исходном виде не применим. Алгоритм двунаправленного поиска так же должен включать логику, которая будет эффективно определять пересечение двух деревьев поиска для нахождения общей точки, что сильно осложняет реализацию, особенно учитывая тот факт, что коэффициент ветвления обратного поиска может отличаться от коэффициента ветвления прямого поиска.

**Хеш-таблица**

Для хранения статей связанных с текущей используем хеш-таблицу

**Хеш-таблица** — это структура данных, в которой все элементы хранятся в виде пары ключ-значение, где:

ключ — уникальное число, которое используется для индексации значений

значение — данные, которые с этим ключом связаны

Используют, если хотят быстро выполнять операции вставки/удаления/нахождения.

**Методы использованные в приложении**

**public static bool CheckNumber(string data,out int number) —** метод проверки чисел на корректность

public **string ConvertArticleToReadbleForm(string article)** — небольшие преобразования названия статьи

public **bool IsArticleExists(string article)** — проверка статьи на существование

public **Dictionary<string, List<string>> ReadArticleLinks()** - чтение статей из файла и добавление их в хеш таблицу

public **List<string> FindArticleLinks(string articleToFind)** — возвращение списка связей для статьи

**public string ConvertToJson(T obj)** — преобразование в Json формат объекта

public **string GetArticleInfo\_Python(string articleTitle)** — получение информации с помощью python-скрипта информации о статье

public **void VisualizeGraph()** - построение графа в Diagram Connector

private **void UpdateData()** - обновление инофрмации на графе

**private void saveToFileButton\_Click(object sender, EventArgs e)** — при нажатии на кнопку откроется диалоговое окно в котором мы можем сохранить файл

private **void findPathButton\_Click(object sender, EventArgs e)** — ключевая функция для отображения информации в программе

private **void reverseButton\_Click(object sender, EventArgs e)** — смена местами начальной и конечной статьи

def **get\_wikipedia\_info(page\_title):** - в Python Функция для сбора информации о статье в графе

def **is\_page\_exists(page\_title):** - в python – проверка статьи на существование

def **get\_api\_page(page\_title):** - в Python получение страницы из Api в wikipedia