МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль)/ специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Исследование алгоритмов поиска пути в графе»

Студент М.А. Степанов _____
(И.О. Фамилия) (личная подпись)

Руководитель Д.М.Ахмедханлы
(ученая степень, звание, И.О. Фамилия)

Консультант М.А. Четаева)
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Тема:Исследование алгоритмов поиска пути в графе.

Как показывает практика, исследование и программная реализация алгоритмов поиска пути в графе представляют актуальностьи научно-практический интерес.

Объект исследования бакалаврской работы - граф.

Предмет исследования - алгоритмы поиска пути в графе.

Цель выпускной квалификационной работы - исследование и программная реализация алгоритмов поиска кратчайшего пути в графе.

Методы исследования: математический аппарат графов, методы и алгоритмы поиска путей в графе, объектно-ориентированный подход к разработке программного обеспечения.

Описаны задачи поиска кратчайшего и оптимального пути в графе, проанализированы известные алгоритмы поиска кратчайшего пути в графе.

Выполнена программная реализация алгоритмов на платформе 1С: Предприятие 8.3. Проведено экспериментальное тестирование алгоритмов.

Результаты бакалаврской работы могут быть рекомендованы для практического решения задач поиска кратчайшего и оптимального пути в графе.

Структура бакалаврской работы: страниц 48 с приложением, рисунков 14, таблиц 2, источников 24.

Abstract

The topic of the given graduation work is Study of pathfinding algorithms in a graph.

Practice shows that study and software implementation of pathfinding algorithms in a graph are of relevance scientific and practical interest.

The objects of the graduation workare graph.

The subjects of study of the graduation work are the pathfindingalgorithmsin graph.

The aim of the graduation work is study and software implementationofpathfinding algorithms in graph.

Research methods: mathematical theory of graphs, methods and algorithms of pathfinding in graph, object-oriented approach to software development.

Problems of finding the shortest and optimal path in a graphare described. The known algorithms of pathfinding in a graph are analyzed.

Software implementation of algorithms on platform 1C: Enterprise 8.3 is executed. Experimental testing of algorithms is carried out.

The results of the graduation work can be recommended for practical solution of problems of finding the shortest and optimalpath in a graph.

The graduation workconsists of an explanatory note on 48 pages including 14 figures, 2 tables, the list of 24 references.

Оглавление

Введение

Большое разнообразие проблем в области науки и техники может быть сведено к проблемам поиска пути в графе, таких как задача кратчайшего пути или задача поиска оптимальных путей относительно более общих определенных целевых функций др. [20].

Стоить отметить, что поиск кратчайшего и оптимального пути в графе имеет широкое практическое применение, например, при решении задач маршрутизации телефонного или интернет-трафика, компоновки печатных плат, GPS-маршрутизации, принятии решений в области искусственного интеллекта и робототехникии др.

Для решения таких задач используются алгоритмы поиска пути в графе.

Исследования и программная реализация алгоритмов поиска пути в графеявляются **актуальными** и представляют научно-практический интерес.

Объектом исследованиябакалаврской работы является граф.

Предметом исследованиябакалаврской работы являются алгоритмы поиска пути в графе.

Цельвыпускной квалификационной работы – исследование и программная реализация алгоритмов поиска кратчайшего пути в графе на платформе 1С: Предприятие 8.

Для достижения данной цели необходимо выполнить следующие задачи:

– проанализировать проблему поиска кратчайшего пути в графе;

– произвести анализ известных алгоритмов поиска кратчайшего пути в графе;

– выполнить программную реализацию и тестирование известных алгоритмов кратчайшего поиска пути в графе на платформе 1С: Предприятие 8.

Методы исследования– математический аппарат графов, методы и алгоритмы поиска путей в графе, объектно-ориентированный подход к

проектированию программного обеспечения.

Практическая значимостьбакалаврской работызаключается в разработке комплекса программ, реализующих популярные алгоритмы поиска кратчайшего пути в графе.

Данная работа состоит из введения, трех глав, заключения, списка используемой литературы и приложений.

Первая глава посвящена анализу и определению проблемы поиска кратчайшего пути в графе, описаны задачи поиска кратчайшего и оптимального пути в графе.

Во второй главе проанализированы известные алгоритмы поиска кратчайшего пути в графе: алгоритм поиска в ширину; жадный алгоритм; алгоритм Дейкстры; алгоритм A*.

Третья главапосвящена программной реализации известных алгоритмов поиска кратчайшего пути в графе. Разработаны логическая архитектура и конфигурация программы. Выполнено экспериментальное тестирование алгоритмов.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Приложение содержит фрагменты программного кода.

Структура бакалаврской работы: страниц 49с приложением, рисунков 14, таблиц 2, источников 24.

Глава 1 Анализ и определение проблемы поиска кратчайшего пути в графе

Граф представляет собой важную сложную сетевую модель, описывающую отношения между различными объектами в реальных приложениях, включая граф знаний, социальную сеть и сеть трафика.

Проблемы поиска пути в графе являются важными и хорошо изученными проблемами.

1.1 Постановка задачи поиска кратчайшего пути в графе

Одной из классических задач теории графов является задача поиска кратчайшего пути в графе.

Данная задача связана с понятием взвешенного графа [22].

Взвешенный граф или граф с ребрами - это тройка вида:

$$G = (E, V, w), \quad (1)$$

w: $E \rightarrow L$ - функция, отображающая ребра или дуги (для ориентированного графа) в их значения;

$L \subseteq \mathbf{R}$- множество возможных действительных значений.

На практике веса или другие значения на ребрах могут представлять такие параметры, как расстояние, емкость или сила отношений.

Рассмотрим взвешенный граф $G = (V, E, w)$, w: $E \rightarrow \mathbf{R}$.

Граф может быть ориентированным или неориентированным.

Для удобства определим $w(u, v) = \infty$, если $(u, v) \notin E$.

Вес пути определяется как сумма весов ребер вдоль этого пути.

Определение: для взвешенного графа $G(V,E,w)$ кратчайшим путь от вершины u к вершине v - это путь от u до v с минимальным весом.

Если существует несколько путей с одинаковым весом, то все они являются кратчайшими взвешенными путями от u до v.

Используем обозначение $\sigma_G(u, v)$ для указания веса кратчайшего пути от u до v.

Таким образом, в задачах с кратчайшим путем необходимо найти кратчайший взвешенный путь между вершинами или, возможно, только вес таких путей σ_G.

На рисунке 1 изображен пример взвешенного графа.

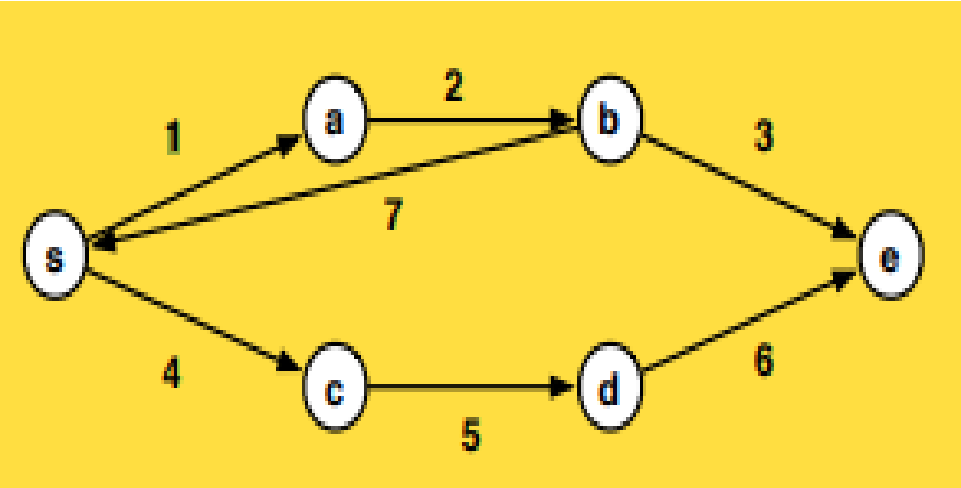

Рисунок 1 – Пример взвешенного графа

Для данного графа кратчайший путь:

$$\sigma_G(s, e) = \sigma_G(s, a) + \sigma_G(a, b) + \sigma_G(b, e) = 1+2+3=6.$$

Существуют различные постановки задачи о кратчайшем пути [8]:

Вычисление кратчайших путей важно во многих практических приложениях. Фактически, существует несколько вариантов этой проблемы, таких как версии с «одним источником» и «с несколькими источниками», как с отрицательными весами, так и без них.

Следует отметить, отрицательные веса ребер затрудняют поиск кратчайших путей.

Поэтому в настоящей бакалаврской работе мы рассматриваем проблему нахождения кратчайшего пути ориентированных графов с неотрицательными весами ребер.

1.2 Задача поиска кратчайшего пути в графе с одним источником

Дан взвешенный граф и исходная вершина [13].

Задача кратчайшего пути с одним источником (SingleSourceShortestPath, SSSP) состоит в том, чтобы найти кратчайший взвешенный путь от исходной вершины до любой другой вершины графа.

Хотя между двумя вершинами может быть много кратчайших путей равного веса, задача требует только один.

Рассмотрим математическую постановку задачи.

Пусть задан граф $G=(V,E)$с весами ребер $f(e)$и выделенной вершиной - источником u.

Последовательность $P(u,v)$ ребер $e_1=(u,w_1)$, $e_2=(w_1,w_2),\ldots$, $e_k=(w_k,v)$ называется путем, идущим от вершины u к вершине v.

Суммарный вес этого пути равен:

$$f(P(u,v)) = \sum_{k=1}^{n} f(e_i) \qquad (2)$$

Требуется для каждой вершины v, достижимой из вершины-источника u, указать путь $P^*(u,v)$, имеющий наименьший возможный суммарный вес:

$$f^*(v) = f(P^*(u,v)) = \min f(P(u,v)) \qquad (3)$$

Опишем свойства задачи:

− решение задачи удовлетворяет принципу оптимальности;

− если путь $P^*(u,w)$ является частью кратчайшего пути $P^*(u,v)$, то $P^*(u,w)$ является кратчайшим путём от источника u до вершины w.

Принцип оптимальности означает, что для каждой вершины vвместо всего кратчайшего пути $P^*(u,v)$ достаточно хранить его последнее ребро e^*_u.

Таким образом, определяется оценка памяти $O(m)$.

Опишем входные и выходные данные.

Входные данные: взвешенный граф (V,E,W) (n вершин v_iи m ребер

$e_j=(v_j^{(1)}, v_j^{(2)})$ с весами f_j), вершина-источник u.

Объём входных данных: $O(m+n)$.

Выходные данные (возможные варианты):

1) для каждой вершины v исходного графа – последнее ребро $e^*_v=(w,v)$, лежащее на кратчайшем пути от вершины uк v, или соответствующая вершина w:

2) для каждой вершины v исходного графа - суммарный вес $f^*(v)$ кратчайшего пути от вершины u к v.

Объём выходных данных: $O(n)$.

Преобразование выходных данных и поиск кратчайшего пути:

Кратчайший путь от u к v может быть восстановлен за время $O(|P^*(u,v)|)$, если, начиная с вершины v, проходить ребра e^*_w в обратном направлении до тех пор, пока не будет посещена вершина u.

Ребра e^*_v могут быть восстановлены за время $O(m)$ перебором ребер для каждой вершины:

$$e_v^* \in \{(w,v) \in E \mid f^*(v) = f^*(w) + f((v,w))\} \qquad (4)$$

Перебор может осуществляться параллельно.

Для поиска кратчайших расстояний $f^*(v)$ по набору ребер e^*_v необходимо построить из них дерево, на котором выполнить поиск в ширину за время $O(m)$ (с возможностью параллелизации).

1.3 Задача поиска оптимального пути в графе

Концепция поиска оптимального пути в графе, которая может быть определена как процесс идентификации пути с определенными критериями оптимальности, такими как расстояние, затраты и т. д.

Эта проблема была тщательно изучена, в результате чего появилось большое количество задач поиска оптимальных путей в графе.

Рассмотрим пример задачи поиска оптимального пути в графе на

примере поиска оптимального пути в сетях [23].

Многие практические задачи оптимизации можно сформулировать как задачу маршрутизации трафика, например, направить поток товаров через сеть по оптимальному маршруту [24].

Пусть $G = (V,A)$ - ориентированный граф,

где:

$|V| = n<\infty$- вершины графа;

$|A| = m<\infty$- ребра графа;

$a_{u,v} := (u,v) \in A$ - направленное ребро от $u \in V$ до $v \in V$.

Рассматриваемый граф не устой, связный, не имеющий ребра с одинаковым направлением между двумя узлами.

Задаем множество исходных узлов, $S \subseteq V$, и множество целевых узлов, $T \subseteq V$.

Пары $(s_1, t_1),\ldots, (s_k,t_k)$, для $s_i \in S$ и $t_i \in$ T, называются парами «источник-назначение» (пары ИН).

Множество ИН-пар не обязательно содержит все возможные пары s-t с $s \in S$ и $t \in T$.

Каждая ИН-пара (s_i, t_i) для $i \in \{1,..,k\}$ имеет множество возможных путей от s_i до t_i, обозначаемое как ρ_i.

Допущение 1. Рассматриваются только ИН-пары, для которых $P_i \neq 0$.

Другими словами, всегда существует, по крайней мере, один путь, который соединяет s_i с t_i, для каждой ИН-пары (s_i, t_i), с $i \in \{1,..,k\}$.

Кроме того, определяем:

$$P := \bigcup_{i=1}^{k} \rho_i \qquad (5)$$

Каждому пути $P \in \rho_i$ сопоставляется поток пути $f_P \in [0; \infty)$.

Поток f на графе является совокупностью всех путевых потоков f_P для $P \in \rho$, то есть:

$$f = \bigcup_{P \in \rho} f_p \quad (6)$$

Поток на одном ребре $a \in A$ определяется как:

$$f_a = \sum_{P \in \rho} \delta_P^a f_P, \qquad (7)$$

где:

$$\delta_P^a := \begin{cases} 1, & \text{если } a \in P \\ 0, & \text{в противном случае} \end{cases}$$

Напомним, что f_a - это количество потока между всеми ИН-парами, который проходит через ребро a, где f_P - поток только на одном пути для одной ИН-пары.

Также отметим, что поток f не обязательно удовлетворяет обычному свойству сохранения потока (рисунки 2 и 3).

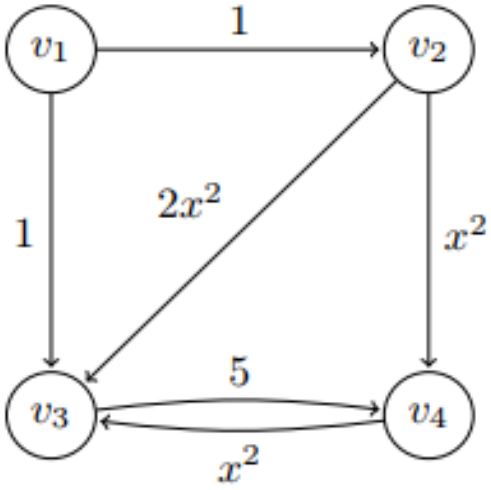

Рисунок 2 - Граф с заданными весовыми функциями ребер

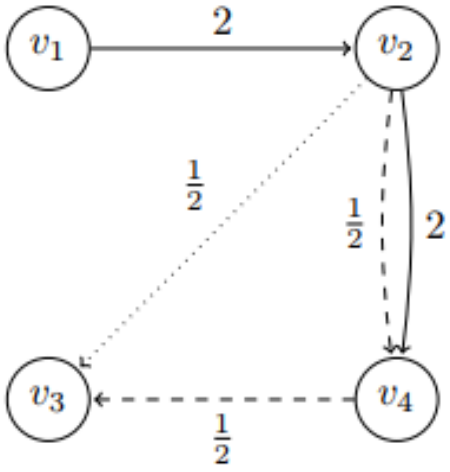

Рисунок 3 – Выполнимый поток графа, представленного на рисунке 2

Между каждой (s_i, t_i) - парой должно быть передано предписанное количество потока, обозначенное как скорость потока (или потребность) r_i.

Тройка C_i: = (s_i,t_i,r_i) называется товаром, где C: = $\{C_1,\ldots, C_k\}$ – множество всех товаров.

Определение 1. Поток f называется выполнимым, если для каждого товара $C_i \in C$ для $i \in \{1,..,k)$ имеют место следующие свойства:

- $\Sigma_{P\in\rho i} f_P = r_i$;
- $f_P \geq 0$ для всех $P \in \rho_i$.

Кроме того, каждому ребру (u, v)=a $\in$A присваивается неотрицательная, непрерывная и неубывающая весовая функция:

$$c_a: \mathbf{R}^+ \rightarrow \mathbf{R}^+,$$

а также обозначенное через $c_{u,v}$ количество потока на ребре, соответствующее его весу.

Эта весовая функция является абстрактной величиной, которая может представлять многие особенности ребра. Данная функция не уменьшается для отображения заторов на ребрах с более высокой пропускной способностью.

Для выполняемого потока f общий вес пути P определяется как:

$$c_P(f) := \sum_{a\in P} c_a(f_a), \qquad (8)$$

Общий вес потока f определяется как:

$$z(f) := \sum_{P\in\rho} c_P(f) f_P = \sum_{a\in A} c_a(f_a) f_a, \qquad (9)$$

Граф G вместе с набором товаров C и весовой функцией c, как описано выше, определяют экземпляр $G_{C,c}$.

Следует отметить, что такой экземпляр может иметь много разных возможных потоков.

Определение 2. Пусть есть экземпляр $G_{C,c}$. Тогда поток f называется

оптимальным, если он позволяет решить задачу оптимизации вида:

$$z(f) \rightarrow \min \qquad (10)$$

для ограничений:

$$\sum_{P \in \rho_i} f_P = r_i, \quad i = \{1, ..., k\}$$

$$f_P \geq 0, \qquad P \in \rho \qquad (11)$$

Нахождение такого потока является задачей маршрутизации трафика.

Примером такой задачи является задача поиска оптимального пути в дорожном графе [11].

Выводы к главе 1

1. В задачах с кратчайшим путем необходимо найти кратчайший взвешенный путь между вершинами.

2. Задача кратчайшего пути с одним источником состоит в том, чтобы найти кратчайший взвешенный путь от исходной вершины до любой другой вершины графа.

3. Концепция поиска оптимального пути в графе может быть определена как процесс идентификации пути с определенными критериями оптимальности, такими как расстояние, затраты и т. д. Эта проблема была тщательно изучена, в результате чего появилось большое количество задач поиска оптимальных путей в графе.

Глава 2 Анализ алгоритмов поиска пути в графе

Для решения задач поиска пути в графе разработано большое количество эффективных алгоритмов.

Рассмотрим наиболее известные из них.

2.1Алгоритм поиска в ширину

Поиск в ширину (Breadth-FirstSearch, BFS) позволяет вычислить кратчайшие расстояния (в терминах количества ребер) от выделенной вершины ориентированного графа до всех остальных вершин, и/или построить корневое направленное дерево, расстояния в котором совпадают с расстояниями в исходном графе. Кроме того, поиск в ширину позволяет решать задачу проверки достижимости (существуют ли пути между вершиной источником и остальными вершинами графа) [12].

Впервые алгоритм поиска в ширину описан в работах Мура и Ли.

Алгоритм основан на обходе вершин графа «по слоям». На каждом шаге есть множество «передовых» вершин, для смежных к которым производится проверка, относятся ли они к еще не посещенным. Все еще не посещенные вершины добавляются в новое множество «передовых» вершин, обрабатываемых на следующем шаге. Изначально в множество «передовых» вершин входит только вершина-источник, от которой и начинается обход.

Рассмотрим математическое описание алгоритма.

Пусть задан граф $G=(V, E)$ без весов, и с выделенной вершиной-источником u.

Путем $P(u,v)$ между вершинами u и v считается множество ребер $(u,v_1),(v_1,v_2),\ldots, (v_{n-1},v)$.

Длиной пути $d(u,v)$ обозначим число ребер в данном пути между

вершинами u и v. Поиск в ширину находит кратчайшие пути $d(u,v)$ от вершины u до всех остальных вершин графа описанным далее образом.

В начале работы алгоритма расстояние до вершины-источника $d(u)=0$, до остальных вершин $d(v)=\infty, \forall v \neq u$.

Также в начале работы алгоритма инициализируется множество $F=\{u\}$.

Далее на каждом шаге алгоритма строится множество вершин $P=w$, таких, что для $\forall v \in F \exists (v,w) \in E |\ d(w)=\infty$, при этом обновляются расстояния $d(w)=d(v)+1$для $\forall w \in P$. Затем производится переход на следующий шаг до тех пор, пока $P \neq 0$.

При этом в начале каждого шага множество F заменяется на P.

Вычислительным ядром алгоритма является обход вершин, смежных с выбранной вершиной v, с последующим добавлением еще не посещенных вершин в множество P.

Данная операция выполняется на каждом шаге для каждой вершины $v \in F$.

Алгоритм последовательно уточняет значения функции $d(v)$.

Макроструктура алгоритм поиска в ширину приведена ниже.

Алгоритм начинается с проверки узла A и всех его соседей. На следующем шаге исследуются соседи ближайшего узла A, и процесс продолжается на следующих этапах. Алгоритм исследует всех соседей каждого узлоа и гарантирует, что каждый узел посещается ровно один раз, и ни один узел не посещается дважды [1].

Шаг 1: Установить СОСТОЯНИЕ = 1 (состояние готовности)
для каждого узла в G;

Шаг 2: Поставить в очередь начальный узел A и установить его СОСТОЯНИЕ = 2 (состояние ожидания);

Шаг 3: повторяем шаги 4 и 5 пока не очистится очередь;

Шаг 4: Исключаем узел N. Обрабатываем его и устанавливаем его

СОСТОЯНИЕ = 3 (обработанное состояние);

Шаг 5: Поставить в очередь всех «соседей» N, которые находятся в состоянии готовности (СОСТОЯНИЕ = 1) и установить их СОСТОЯНИЕ = 2 (состояние ожидания);

[КОНЕЦ ЦИКЛА].

Шаг 6: ВЫХОД.

Опишем входные и выходные данные.

Входные данные: граф $G(V,E)$, $|V|$ вершин v_iи |E| ребер: $e_j=(v_j^{(1)},v_j^{(2)})$, вершина-источник u.

Объём входных данных: $O(|V|+|E|)$.

Выходные данные (возможные варианты):

– для каждой вершины v исходного графа расстояние $d(v)$, определенное как число ребер, лежащих на кратчайшем пути от вершины uк v;

– для каждой вершины v исходного графа значение достижимости (достижима или нет) от вершины-источника u.

Объём выходных данных: $O(|V|)$.

Алгоритм имеет временную сложность $O(|V|+|E|)$,

где:

|V| и |E| - число вершин и ребер графа соответственно: алгоритм инициализирует начальный массив расстояний - $O(|V|)$ операций, а затем обходит каждую вершину один единственный раз - $O(|E|)$ операций.

Данная оценка верна в случае, если формат хранения графа позволяет обходить вершины, смежные к выбранной (к примеру форматы списка смежности, сжатого списка смежности).

При использовании других форматов оценка сложности может быть большей.

2.2 Жадный алгоритм поиска кратчайшего пути

Жадный алгоритм - это простой, интуитивно понятный алгоритм, который используется в задачах оптимизации. Алгоритм делает оптимальный выбор на каждом этапе, так как он пытается найти общий оптимальный способ решения всей проблемы.

Жадный алгоритм - это алгоритм, который делает выбор на основе образованных эвристик (догадок) на каждом этапе.

Затем будет исследован узел с кратчайшим эвристическим расстоянием от целевого узла [19].

Иными словами, жадные алгоритмы берут все данные в конкретной задаче, а затем устанавливают правило, для которых элементы добавляются в решение на каждом этапе алгоритма.

Чтобы создать жадный алгоритм, необходимо определить оптимальную подструктуру или подзадачу в задаче. Затем определить, что будет включать решение (например, наибольшая сумма, кратчайший путь и т. д.).

Необходимо создать некий итеративный способ прохождения всех подзадач и построения решения.

Макроструктура жадного алгоритма имеет вид:

Шаг 1. Определяем dis[v] для всех узлов = INT_MAX (расстояние от каждого узла до целевого узла).

Шаг 2. Определяем dis[root] = эвристика (корень, цель) (расстояние от корневого узла до цели).

Шаг 3. Добавляем корневой узел в приоритетную очередь.

Шаг 4. Выполняем цикл в очереди, пока она не пуста.

- В каждом цикле выбрать узел с минимальным эвристическим расстоянием от целевого узла в очереди (корневой узел будет выбран первым).
- Удалить текущий выбранный узел из очереди (vis [current] = true).
- Если текущий выбранный узел является целевым, восстановить его.
- Для каждого дочернего элемента текущего узла сделать следующее:

– если дочерний узел уже посещен (ранее удален из очереди), пропустить эту итерацию;
– назначитьdist[current] = эвристика (current, цель);
– добавить дочерний узел в очередь.

Шаг 5. Если очередь пуста, то целевой узел не найден.

Шаг 6. Выход.

Оценим временную сложность жадного алгоритма.

На первом этапе применяется алгоритм Флойда-Уоршелла, имеющий сложность $O(n^3)$.

Затем на каждом шаге для каждой вершины перебираем путь во все подключенные: $O(n^3)$.

Остальные шаги алгоритма не добавляют сложность.

В результате получаем итоговую сложность алгоритма равную $O(n^3)$.

Жадные алгоритмы весьма успешны в задачах нахождения пути в графе.

Однако во многих задачах жадная стратегия не дает оптимального решения в задачах нахождения кратчайшего пути в графе [5].

Если оба из приведенных ниже свойства верны, для решения проблемы можно использовать жадный алгоритм:

– свойство жадного выбора: глобальное (общее) оптимальное решение может быть достигнуто путем выбора оптимального выбора на каждом этапе;

– оптимальная подструктура. Задача имеет оптимальную подструктуру, если оптимальное решение всей задачи содержит оптимальные решения подзадач.

Другими словами, жадные алгоритмы работают над задачами, для которых верно, что на каждом шаге есть выбор, который является оптимальным для задачи до этого шага, и после последнего шага алгоритм выдает оптимальное решение полного проблем.

2.3 Алгоритм Дейкстры

Алгоритм Дейкстры предназначен для решения задачи поиска кратчайшего пути на графе. Для заданного ориентированного взвешенного графа с неотрицательными весами алгоритм находит кратчайшие расстояния от выделенной вершины - источника до всех остальных вершин графа [18].

Алгоритм Дейкстры является асимптотически быстрейшим из известных последовательных алгоритмов для данного класса задач.

Рассмотрим математическое описание алгоритма Дейкстры [0%D1%8B].

Пусть задан граф $G=(V,E)$ с весами ребер f(e) и выделенной вершиной-источником u.

Обозначим через $d(v)$ кратчайшее расстояние от источника u до вершины v.

Пусть уже вычислены все расстояния, не превосходящие некоторого числа r, то есть расстояния до вершин из множества $V_r=\{v\in V|d(v)\leq r\}$.

Пусть$(v,w)\in\text{argmin}\{d(v)+f(e)|v\in V,e=(v,w)\in E\}$.

Тогда $d(w)=d(v)+f(e)d(w)$, и v лежит на кратчайшем пути от u к w.

Величины

$$d+(w)=d(v)+f(e), \qquad (11)$$

где:

$v\in V_r$, $e=(v,w)\in E$ называются предполагаемыми расстояниями и являются оценкой сверху для настоящих расстояний: $d(w)\leq d^+(w)$.

Алгоритм Дейкстры на каждом шаге находит вершину с наименьшим предполагаемым расстоянием, помечает её как посещённую и обновляет предполагаемые расстояния для всех концов рёбер, исходящих из неё.

Входные данные: взвешенный граф (V, E, W) (n вершин v_i и m ребер $e_j=(v_j^{(1)},v_j^{(2)})$ с весами fj), вершина-источник u.

Объём входных данных: $O(m+n)$.

Выходные данные (возможные варианты):

– для каждой вершины v исходного графа – последнее ребро $e^*_v = (w,v)$, лежащее на кратчайшем пути от вершины u к v, или соответствующая вершина w;

– для каждой вершины v исходного графа – суммарный вес $f^*(v)$ кратчайшего пути от вершины u к v.

Объём выходных данных: $O(n)$.

Макроструктура алгоритма Дейкстры имеет вид:

```
Q := новая приоритетная очередь
Для каждого v ∈ V:
   Если v = u то d(v) := 0 else d(v) := ∞
Q.вставить(v, d(v))
Пока Q ≠ ∅:
v := Q.удалить_min()
   Для каждого e = (v, w) ∈ E:
      Если d(w) > d(v) + f(e):
         d(w) := d(v) + f(e)
Q.уменьшить_ключ(w, d(w))
```

Блок-схема алгоритма Дейкстры представлена на рисунке 4 [10].

Рисунок 4 – Блок-схема алгоритма Дейкстры

Алгоритм Дейкстры использует структуру данных для хранения и запроса частичных решений, отсортированных по расстоянию от начала.

Исходный алгоритм использует очередь с минимальным приоритетом и выполняется за время$O(|V|^2)$.

При использовании фибоначчиевой кучи время вычисления минимума сокращается до $O(|E|+|V|\log|V|)$, что является асимптотически наилучшим известным результатом для данного класса задач.

2.4 Алгоритм A*

Алгоритм A* представляет собой комбинацию алгоритмов Дейкстры и жадного алгоритма (в некоторых работах называется усовершенствованным алгоритмом Дейкстры).

Здесь используется расстояние от корневого узла плюс эвристическое расстояние до цели. Алгоритм завершается, когда мы находим целевой узел.

Рассмотрим математическое описание алгоритма[1].

В процессе работы алгоритма для вершин рассчитывается функция:

$$f(v)=g(v)+h(v) \quad (12)$$

где:

– g(v) — наименьшая стоимость пути вv из стартовой вершины;

– $h(v)$ — эвристическое приближение стоимости пути от vv до конечной цели.

Фактически, функция $f(v)$ — длина пути до цели, которая складывается из пройденного расстояния $g(v)$ и оставшегося расстояния $h(v)$.

Исходя из этого, чем меньше значение $f(v)$, тем раньше мы откроем вершину v, так как через неё мы предположительно достигнем расстояние до цели быстрее всего.

Открытые алгоритмом вершины можно хранить в очереди с приоритетом по значению $f(v)$.

Таким образом, алгоритм A* действует подобно алгоритму Дейкстры и просматривает среди всех маршрутов ведущих к цели сначала те, которые благодаря имеющейся информации (эвристическая функция) в данный момент являются наилучшими.

Макроструктура алгоритма A* представлена ниже.

Шаг 1. Определяем dis[v] для всех узлов = INT_MAX (расстояние от корневого узла + эвристика каждого узла).

Шаг2. Определяем dis[root] = 0 + эвристика (корень, цель) (расстояние от корневого узла до самого себя + эвристика).

Шаг 3. Добавим корневой узел в приоритетную очередь.

Шаг 4. Выполняем цикл в очереди, пока она не пуста.

- В каждом цикле выбрать узел с минимальным расстоянием от корневого узла в очереди + эвристика (корневой узел будет выбран первым).
- Удалить текущий выбранный узел из очереди (vis[current] = true).
- Если текущий узел является целевым, восстановить его.

- Для каждого дочернего элемента текущего узла делаем следующее:
  - определяем temp = расстояние (корень, текущий узел) + расстояние (текущий узел, дочерний узел) + эвристика (дочерний узел, цель).
  - если temp <dis [child], тогдаприсвоитьdist[child] = temp. Это означает, что был найден более короткий путь к дочернему узлу.
  - добавить дочерний узел в очередь, если его еще нет в очереди (таким образом, он теперь помечен как не посещенный снова).

Шаг 5. Если очередь пуста, то целевой узел не найден.

Шаг 6. Выход.

Блок-схема алгоритма A* изображена на рисунке 5.

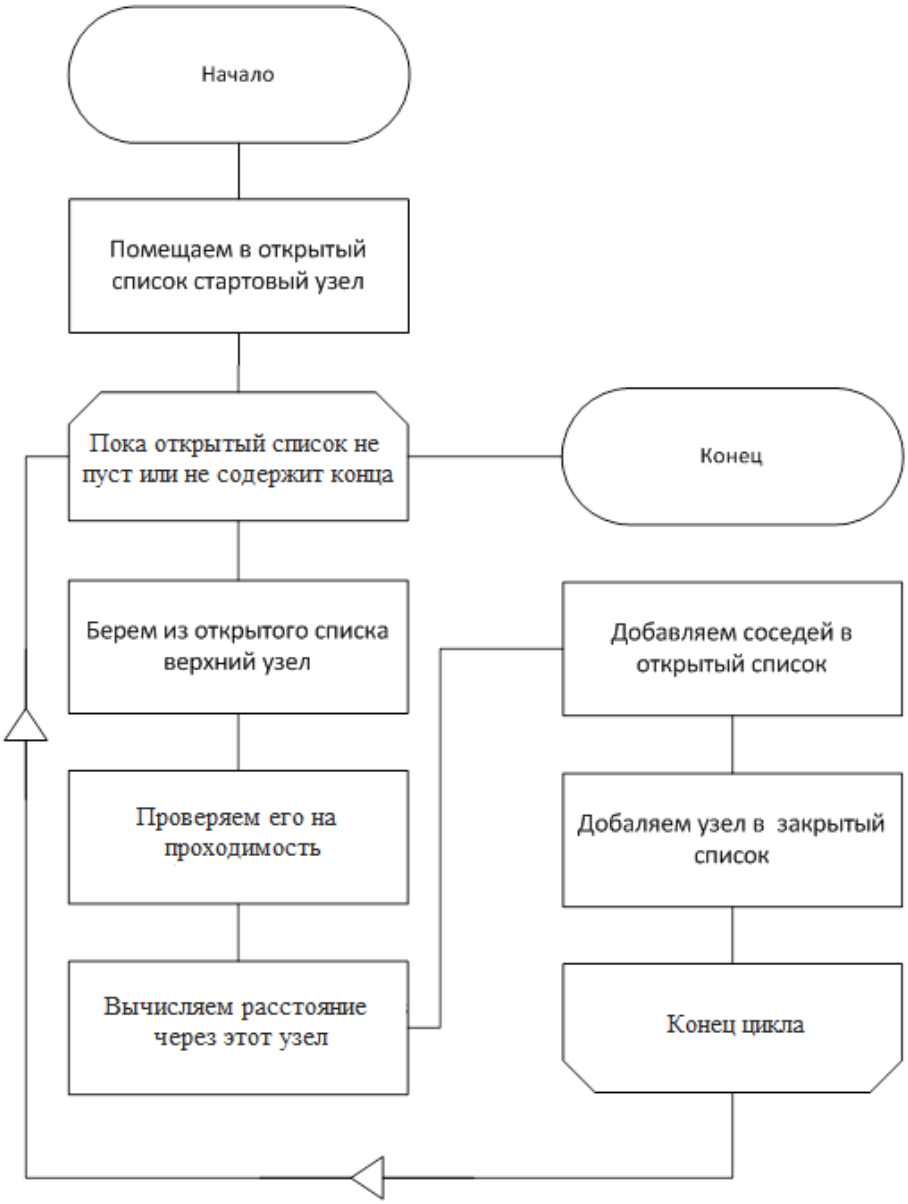

Рисунок 5 – Блок-схема алгоритма A*

Алгоритм A* обладает двумя ключевыми характеристиками алгоритмов такого рода: оптимальность и полнота.

Временная сложность алгоритма A* зависит от эвристики.

В худшем случае неограниченного пространства поиска число развернутых узлов экспоненциально по глубине решения (кратчайший путь) d: $O(b^d)$, где b - коэффициент ветвления (среднее число наследников на состояние).

Это предполагает, что целевое состояние существует вообще и достижимо из начального состояния; если это не так, а пространство

состояний бесконечно, алгоритм не завершится.

2.4 Сравнительный анализ алгоритмов поиска пути в графе

Для сравнения характеристик вышеописанных алгоритмов составим таблицу 1[15, 21].

Таблица 1 – Сравнительный анализ алгоритмов поиска пути в графе (*n*,*m*- количество вершин и ребер графа, соответственно)

Алгоритм	Временная сложность	Преимущества	Недостатки
Алгоритм поиска в ширину	$O(n+m)$	– сравнительная простота реализации; – полнота и оптимальность	– при увеличении пространства поиска производительность поиска снижается по сравнению с другими эвристическими алгоритмами; – относительно высокая емкостная и временная сложность
Жадный алгоритм поиска кратчайшего пути	$O(n^3)$.	– простота поиска оптимального решения; – простота оценки временной сложности	– сложность оценки правильности алгоритма

Продолжение таблицы 1

Алгоритм Дейкстры	$O(m+n\log n\|)$	– сравнительная простота реализации; – невысокая полиномиальная временная сложность;	– отыскивает кратчайшие пути и их величины только от одной вершины; – используется для

			поиска в графе, имеющих неотрицательные веса ребер
Алгоритм A*	$O(m) = O(b^d)$	– полнота и оптимальность; – считается лучшим для решения очень сложных задач	– алгоритм завершается, если коэффициент ветвления конечен и каждое действие имеет фиксированную стоимость; – скорость выполнения поиска сильно зависит от точности эвристического алгоритма, который используется для вычисления фактической стоимости от текущего состояния к целевому состоянию

Как следует из таблицы 1, все представленные алгоритмы позволяют решить задачу поиска кратчайшего пути в графе.

При этом выбор алгоритма для решения конкретной задачи зависит от многих факторов, в том числе от требования простоты программной реализации алгоритма.

Выводы к главе 2

1. Для решения задач поиска пути в графе разработано большое количество алгоритмов. Как показал сравнительный анализ, все известные

алгоритмы позволяют эффективнорешить задачу поиска кратчайшего пути в графе.

2. Выбор конкретного алгоритма поиска пути в графе зависит от многих факторов, в том числе от требования простоты программной реализации алгоритма.

Глава 3 Программная реализация поиска пути в графе

3.1 Разработка логической архитектуры программы

Логическая архитектура описывает программу с точки зрения ее концептуальной организации в слоях, пакетах, основных платформах, классах объектов, интерфейсах и подсистемах и их взаимодействия[9].

Иными словами, логическая архитектура – это объектно-ориентированное представление разрабатываемой программы.

Для разработки логической архитектуры программы реализации алгоритмов поиска пути в графе используем базовые диаграммы языка визуального моделирования UML: диаграмму вариантов использования и диаграмму классовUML [16].

Диаграмма вариантов использования UML - это тип диаграмм поведения, который отображает зависимости между участниками и вариантами использования.

Диаграмма вариантов использования UML отображает систему на концептуальном уровне.

Целью диаграммы вариантов использования UML является демонстрация различных типов пользователей системы и различных способов их взаимодействия с этой системой.

Диаграммы вариантов использования часто используются вместе с текстовыми вариантами использования и диаграммами других типов.

Обозначение диаграммы вариантов использования включает в себя следующие типы символов:

− варианты использования представлены в виде горизонтальных овалов и отображают различные варианты использования;

− акторы - это люди, которые используют варианты использования и представлены на диаграмме в виде фигур людей. Акторы не могут быть связаны друг с другом (кроме отношений обобщения / наследования);

− ассоциации показаны в виде линий между субъектами и вариантами использования;

− граница системы - поле с именем и овалами (прецедентами) внутри, которые устанавливают область системы для прецедентов.

Диаграмма вариантов использования разрабатываемой программы представлена на рисунке 6.

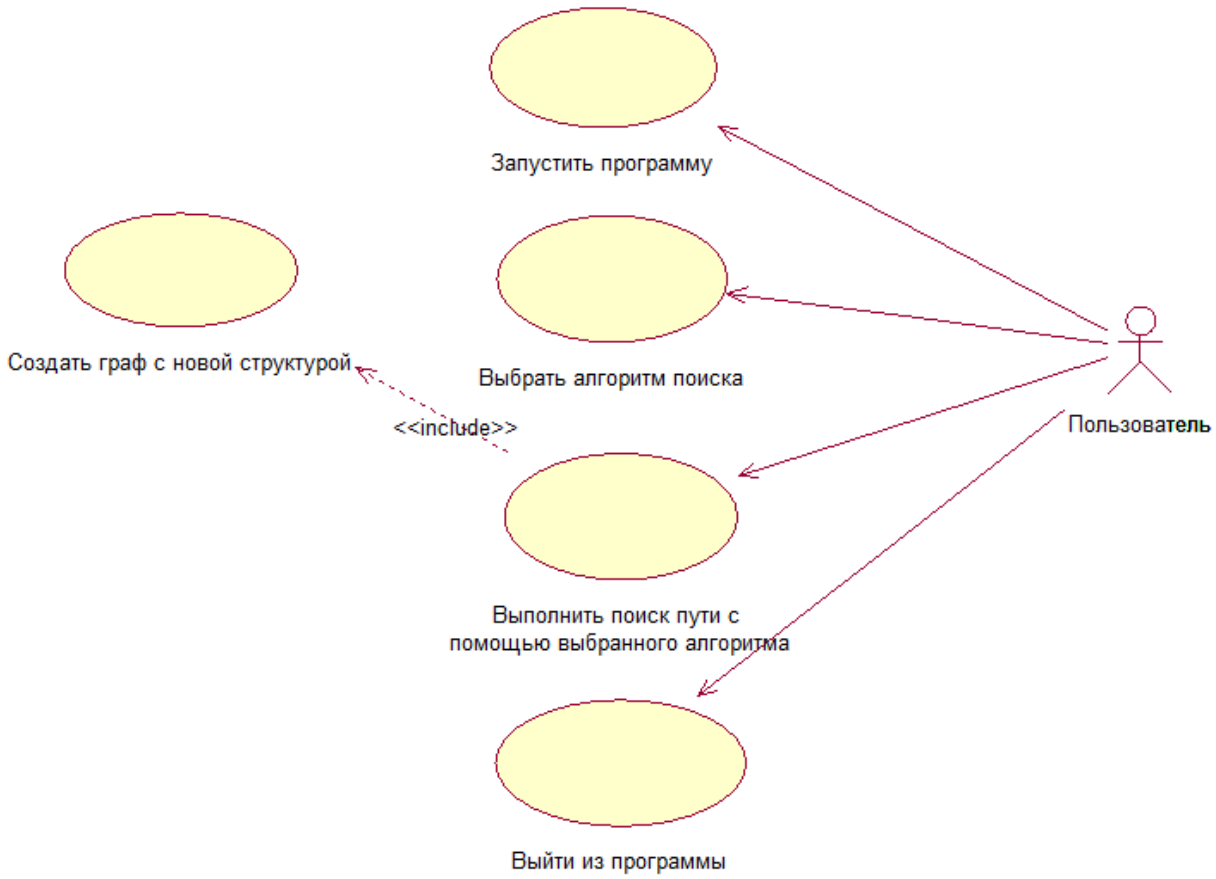

Рисунок 6 - Диаграмма вариантов использования программы

Диаграмма вариантов использования программы отражает ее функциональный аспект.

Диаграммы классов являются одним из наиболее полезных типов диаграмм в UML, так как они четко отображают структуру конкретной системы путем моделирования ее классов, атрибутов, операций и отношений между объектами.

На рисунке 7 представлена диаграмма классов программы.

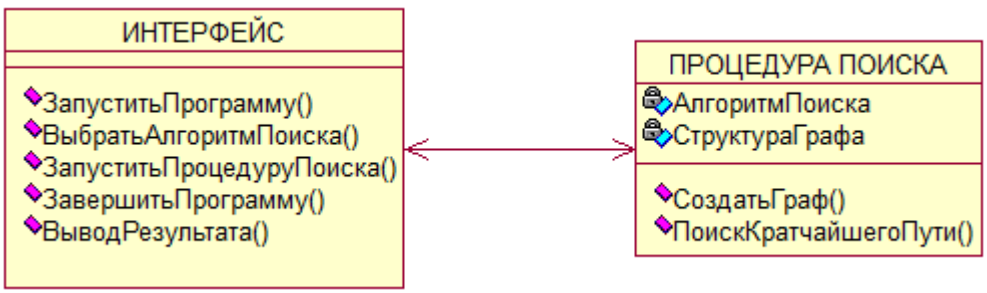

Рисунок 7 - Диаграмма классов программы

Диаграмма классов отражает статический аспект программы.

Представленные диаграммы положены в основу программы, реализующей алгоритмы поиска кратчайшего пути в графе.

3.2 Разработка программного обеспечения для реализации алгоритмов поиска

В качестве среды для реализации алгоритмов поиска кратчайшего пути в графе используем платформу 1С: Предприятие 8.3[4].

Для разработки программных кодов алгоритмов используется объектно-ориентированный язык 1С8, позволяющий реализовать алгоритмы с необходимой полнотой и наглядностью.

Для реализации алгоритмов используется абстрактная структура данных «Очередь» или «Приоритетная очередь»[3].

В концепции 1С: Предприятие 8.3:

«Очередь» - это абстрактный тип данных, представляющий собой список элементов, организованных по принципу ФИФО («первый вошел - первый вышел»).

Основные операции с очередью (рисунок 8):

- Добавить элемент в конец очереди;
- Получить элемент из начала очереди (без удаления из очереди);
- Получить элемент из начала очереди (с удалением из очереди);
- Проверка пустая ли очередь.

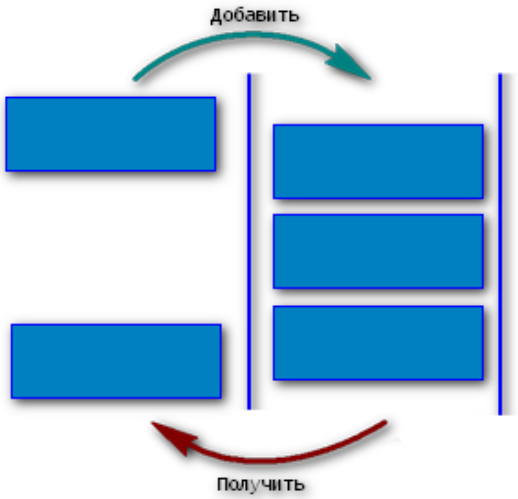

Рисунок 8 – Структурная схема очереди

«Приоритетная очередь»— это абстрактная структура данных, где у каждого элемента есть приоритет. Элемент с более высоким приоритетом находится перед элементом с более низким приоритетом. Если у элементов одинаковые приоритеты, они располагаются в зависимости от своей позиции в очереди (рисунок 9).

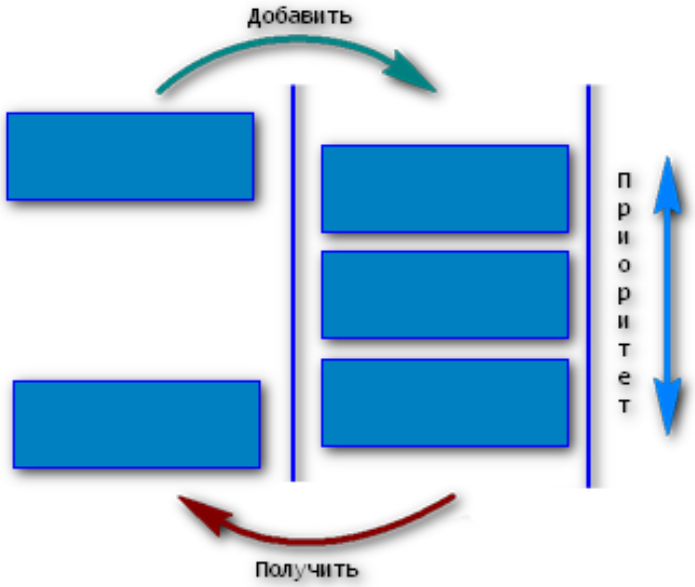

Рисунок 9 – Структурная схема приоритетной очереди

Очередь с приоритетом использует такие же операции что и обычная очередь.

Приоритетную очередь можно реализовать следующими методами:

1) Соответствие - при добавлении элемента сортирует данные по ключу, но ключ должен быть уникальным. Нам не подходит, потому как могут быть элементы с одинаковым приоритетом. На самом деле можно сделать ключ уникальным и "правильным" для нас образом сортируемым. Но сейчас я не буду рассматривать его как претендента для реализации приоритетной очереди.

2) Таблица Значений - есть метод Сортировать(), а также можно добавлять сколь угодно колонок. Подходит, но есть ограничение - не работает в тонком клиенте.

3) Список Значений - есть методы СортироватьПоПредставлению() и СортироватьПоЗначению(). Подходит, работает как в тонком, так и в других клиентах.

Для представления архитектуры программного обеспечения используем диаграмму компонентов UML.

Диаграмма компонентов UML описывает организацию и разводку физических компонентов в системе. Диаграммы компонентов часто составляются для того, чтобы помочь моделировать детали реализации и перепроверить, что каждый аспект требуемых функций системы охватывается запланированной разработкой.

Диаграмма компонентов архитектуры программного обеспечения на платформе 1С: Предприятие 8.3, реализующей алгоритмы поиска пути в графе, изображена на рисунке10.

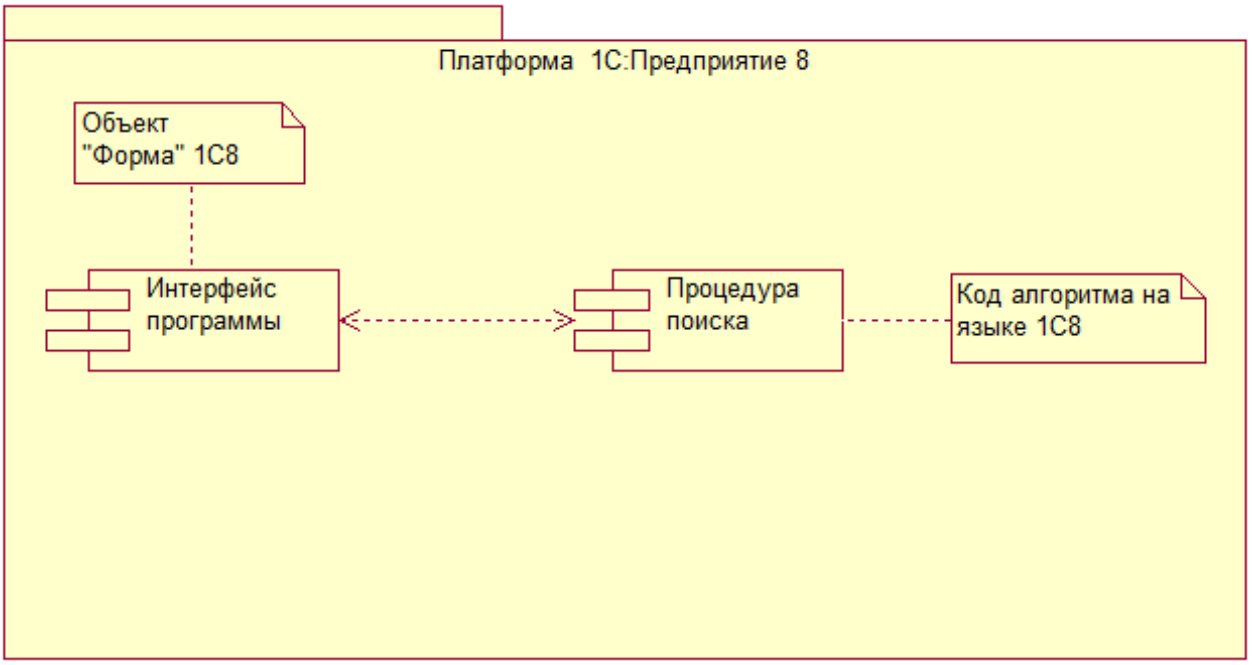

Рисунок 10 – Архитектура программного обеспечения

Примеры кодов алгоритмов поиска пути в графе, реализованные в виду функций на языке 1С8, представлены ниже.

Алгоритм поиска в ширину:

```
ФункцияГраф_ПоискВШирину(ВершинаКуда)
    Очередь =Очередь_Новый();
    Очередь_Добавить(Очередь,ВершинаКуда);
    ПосещенныеВершины=Новый Структура;
    ПосещенныеВершины.Вставить(ВершинаКуда,Новый
Структура("Вершина, Стрелка","А"));
        ПокаНеОчередь_Пустой(Очередь)Цикл
        Вершина =Очередь_Получить(Очередь);
            Соседи =Граф_ПолучитьСоседейВершины(Вершина);
        Длякаждого Сосед Из Соседи Цикл
            ЕслиНеПосещенныеВершины.Свойство(Сосед.Ключ)Тогда
                Очередь_Добавить(Очередь,Сосед.Ключ);
                ПосещенныеВершины.Вставить(Сосед.Ключ,Новый
Структура("Вершина,
```

```
Стрелка",Вершина,Карта_ПолучитьСтрелкуРеверс(Сосед.Значение)));
                КонецЕсли;
            КонецЦикла;
        КонецЦикла;
        ВозвратПосещенныеВершины;
КонецФункции
```

Алгоритм A*:

```
ФункцияГраф_АлгоритмА(ВершинаКуда,ВершинаОткуда)
        ПриоритетнаяОчередь=ПриоритетнаяОчередь_Новый();
        ПриоритетнаяОчередь_Добавить(ПриоритетнаяОчередь,
ВершинаКуда,0);
        ВершиныОткуда=НовыйСтруктура(ВершинаКуда,Новый
Структура("Вершина, Стрелка","А"));
        СтоимостьДвижения=Новый Структура(ВершинаКуда,0);
        ВершинаЦель= Граф[ВершинаОткуда];

        ПокаНеПриоритетнаяОчередь_Пустой(ПриоритетнаяОчередь)Цикл
            Вершина
=ПриоритетнаяОчередь_Получить(ПриоритетнаяОчередь);

            Если Вершина =ВершинаОткудаТогда
                Прервать;
            КонецЕсли;
            Соседи =Граф_ПолучитьСоседейВершины(Вершина);
            Длякаждого Сосед Из Соседи Цикл
                НоваяСтоимость=СтоимостьДвижения[Вершина]
+Граф_ПолучитьСтоимость(Сосед.Ключ);
                ПрежняяСтоимость=Неопределено;
```

```
        ЕслиНеСтоимостьДвижения.Свойство(Сосед.Ключ,ПрежняяСтоимость
)
                        ИлиНоваяСтоимость<ПрежняяСтоимость
                        Тогда

        СтоимостьДвижения.Вставить(Сосед.Ключ,НоваяСтоимость);

                        Приоритет
=НоваяСтоимость+Граф_ПолучитьРасстояние(Граф[Сосед.Ключ],ВершинаЦ
ель);

        ПриоритетнаяОчередь_Добавить(ПриоритетнаяОчередь,Сосед.Ключ,
Приоритет);
                        ВершиныОткуда.Вставить(Сосед.Ключ,
НовыйСтруктура("Вершина,
Стрелка",Вершина,Карта_ПолучитьСтрелкуРеверс(Сосед.Значение)));
                    КонецЕсли;
                КонецЦикла;
        КонецЦикла;
                ВозвратВершиныОткуда;
КонецФункции
```

В программе средствами 1С8 реализована анимация процесса поиска.

На рисунках11-14 представлены результаты тестирования алгоритмов поиска пути в графе.

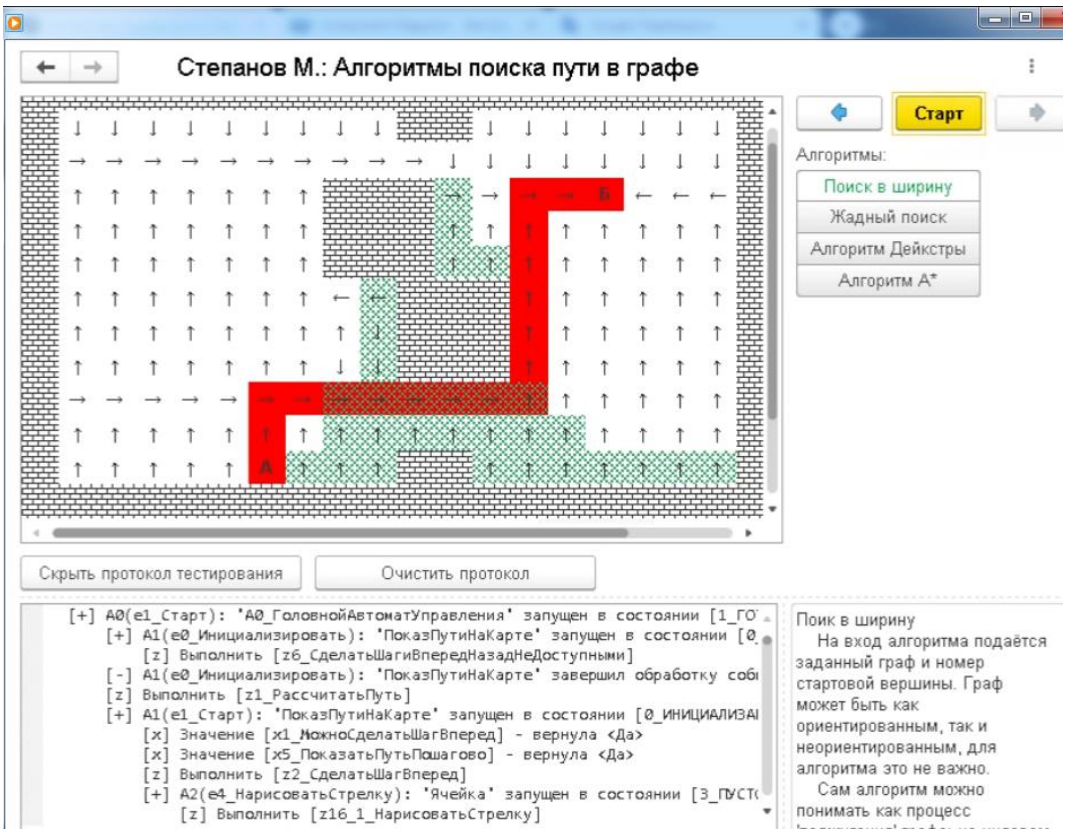

Рисунок 11 - Результаты тестирования алгоритма поиска в ширину

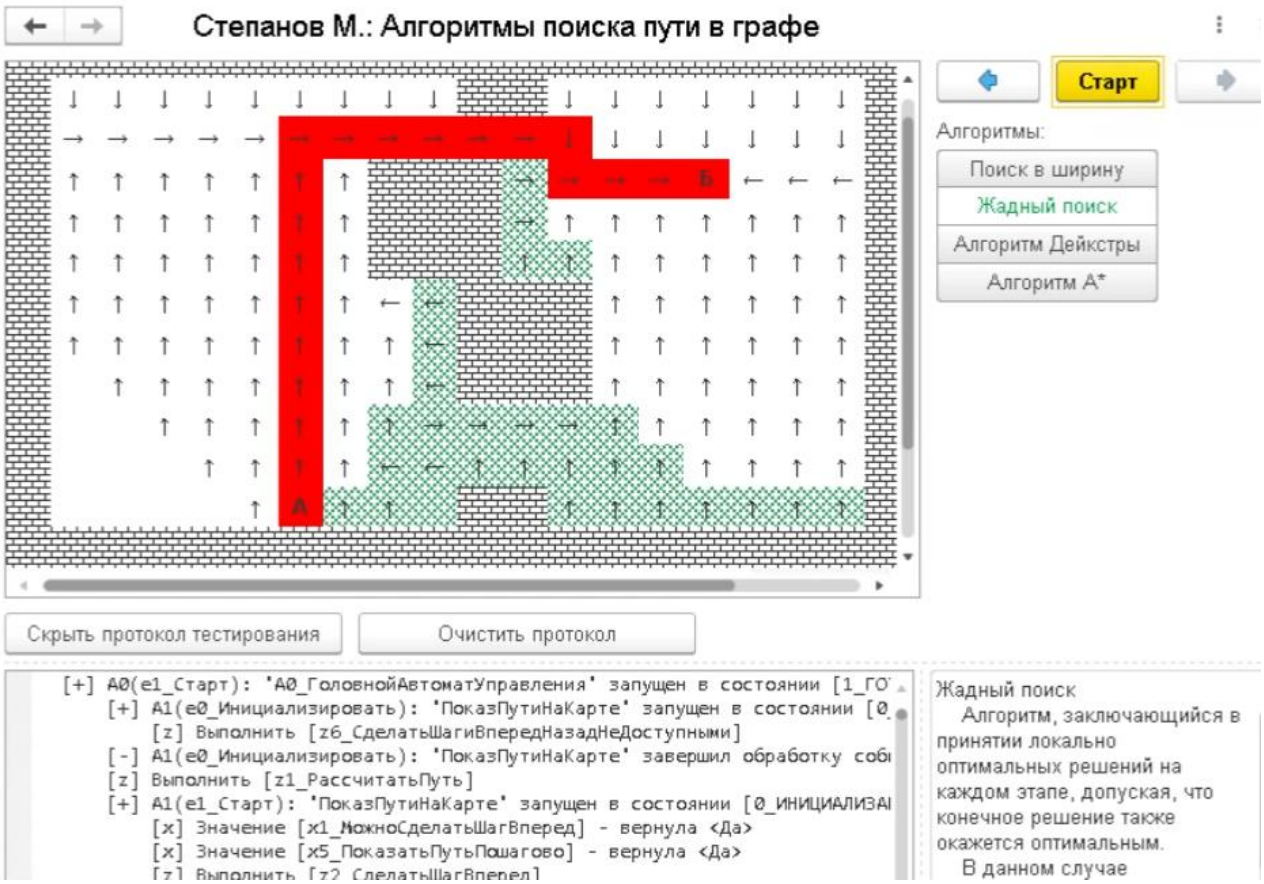

Рисунок 12 - Результаты тестирования жадного алгоритма поиска

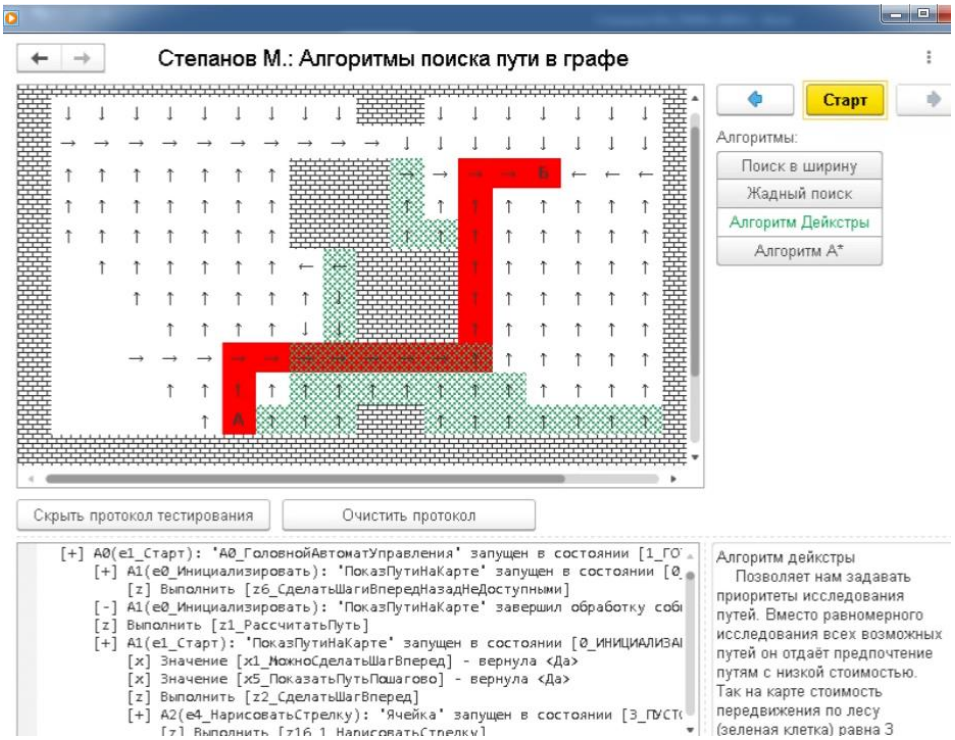

Рисунок 13 - Результаты тестирования алгоритма Дейкстры

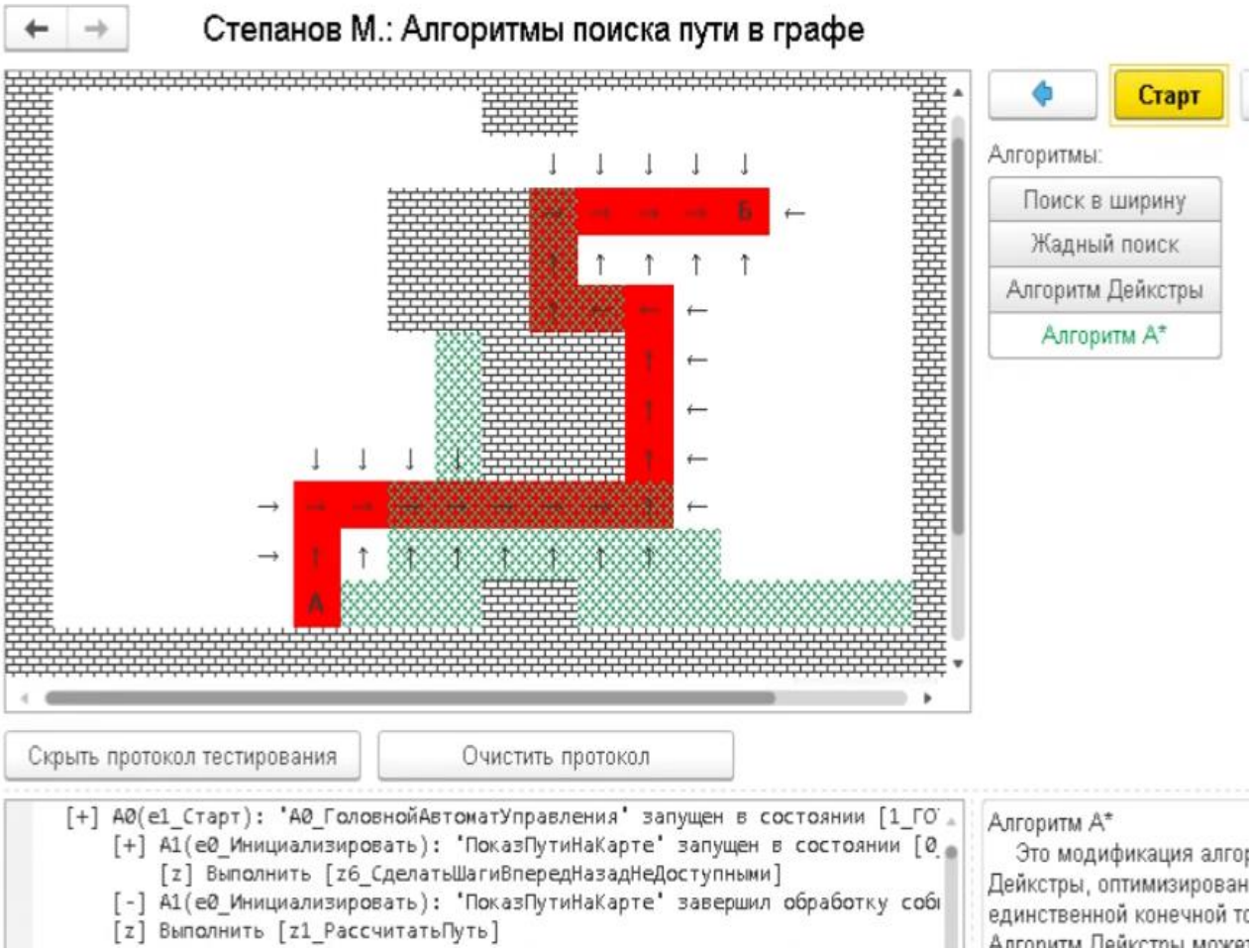

Рисунок 14 - Результаты тестирования алгоритма А*

Программа представляет собой конфигурацию 1С: Предприятие 8.3 с расширением *.EPF.

Фрагмент кода 1С8представленной выше формы интерфейса программы приведен в Приложении А.

В программе имеется возможность экспорта протокола тестирования в текстовый файл (.TXT).

Соответственнопроизводительность алгоритма определяется количеством шагов в протоколе.

Было проведено экспериментальное тестирование алгоритмов на одном и том же случайном графе, результаты которого представлены в таблице 2.

Таблица 2. Результаты тестирования алгоритмов поиска кратчайшего пути в графе

Наименование алгоритма	Производительность в шагах
Алгоритм A*	592
Алгоритм Дейкстры	1451
Жадный алгоритм	1493
Алгоритм поиска в ширину	1574

Как следует из таблицы, лучшую производительность показал алгоритм A*.

Помимо известных положительных свойств данного алгоритма полученный результат достигнут за счет применения при его реализации приоритетной очереди.

Таким образом, экспериментальное тестирование известных алгоритмы поиска пути в графе с помощью разработанной программы подтвердило их основные свойства.

Заключение

Бакалаврская работа посвящена актуальной проблеме анализа и реализации алгоритмов поиска пути в графе.

В ходе выполнения бакалаврской работы достигнуты следующие результаты:

1. Проанализирована проблема поиска пути в графе. Описаны общая задача поиска кратчайшего пути в графе, задача поиска пути с одним источником и задача поиска оптимального пути в графе.

2. Приведен анализ известных алгоритмов поиска кратчайшего пути в графе: алгоритма поиска в ширину, жадного алгоритма, алгоритма Дейкстры и алгоритма A*. Как показал сравнительный анализ, все известные алгоритмы позволяют решить задачу поиска кратчайшего пути в графе. Выбор конкретного алгоритма поиска пути в графе зависит от многих факторов, в том числе от требования простоты программной реализации алгоритма.

3. Выполнена программная реализацияизвестных алгоритмов поиска кратчайшего пути в графе. Разработаны логическая архитектура программы и конфигурация на платформе 1С: Предприятие 8.3.

4. Проведено экспериментальное тестирование алгоритмов на одном и том же случайном графе. Тестирование известных алгоритмы поиска пути в графе с помощью разработанной программы подтвердило их описанные свойства.

Результаты бакалаврской работы могут быть рекомендованы для практического решения задач поиска кратчайшего и оптимального пути в графе на платформе 1С: Предприятие 8.

Список используемой литературы и используемых источников

1. Алгоритм A* [Электронный ресурс]. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_A*#.D0.A0.D0.B5.D0.B0.D0.BB.D0.B8.D0.B7.D0.B0.D1.86.D0.B8.D1.8F(дата обращения: 12.02.2020).

2. Алгоритм Дейкстры[Электронный ресурс]. URL:https://algowiki-project.org/ru/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B(дата обращения: 12.02.2020).

3. Алгоритмы поиска пути в графе[Электронный ресурс]. URL: https://infostart.ru/public/1088569/ (дата обращения: 12.02.2020).

4. Архитектура платформы 1С:Предприятие(версия 8.3.17)[Электронный ресурс]. URL:https://v8.1c.ru/platforma/(дата обращения: 12.02.2020).

5. БойковВ. А. О применении жадных алгоритмов в некоторых задачах дискретной математики // Программные продукты и системы. 2019. №1. С.55-62.

6. ГОСТ 19.402–78. Единая система программной документации. Описание программы.

7. ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

8. Задача о кратчайшем пути [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%BC_%D0%BF%D1%83%D1%82%D0%B8 (дата обращения: 12.02.2020).

9. Ларман К. Применение UML и шаблонов проектирования. М. :

Издательский дом «Вильямс», 2004. 624 с.

10. Лекция «Алгоритм Дейкстры» [Электронный ресурс]. URL:https://www.intuit.ru/studies/professional_retraining/943/courses/2/lecture/42?page=6(дата обращения: 12.02.2020).

11. Плотников, Подвальный Е. С. Решение задачи поиска оптимального пути между двумя точками на графе снерегулярным весом ребер // Вестник ВГТУ. 2012. №6. С. 22-26.

12. Поиск в ширину (BFS)[Электронный ресурс]. URL: https://algowiki-project.org/ru/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%B2_%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D1%83_(BFS)(дата обращения: 12.02.2020).

13. Поиск кратчайшего пути от одной вершины (SSSP)[Электронный ресурс]. URL: https://algowiki-project.org/ru/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B5%D0%B3%D0%BE_%D0%BF%D1%83%D1%82%D0%B8_%D0%BE%D1%82_%D0%BE%D0%B4%D0%BD%D0%BE%D0%B9_%D0%B2%D0%B5%D1%80%D1%88%D0%B8%D0%BD%D1%8B_(SSSP)(дата обращения: 12.02.2020).

14. Реализуем Стек, Очередь и Приоритетную очередь в 1С[Электронный ресурс]. URL:https://infostart.ru/public/1079893/ (дата обращения: 12.02.2020).

15. Сайт Braincart.com[Электронный ресурс]. URL: https://www.brainkart.com/(дата обращения: 12.02.2020).

16. СамуйловС.В. Объектно-ориентированное моделирование на основе UML [Электронный ресурс]: учебное пособие. Саратов: Вузовское образование, 2016. 37 с., URL:http://www.iprbookshop.ru/47277.html(дата обращения: 12.02.2020).

17. BreadthFirstSearch (BFS) Algorithm [Электронныйресурс]. URL: https://www.javatpoint.com/breadth-first-search-algorithm (дата обращения: 12.02.2020).

18. Dijkstra E. W. A note on two problems in connexion with graphs // Numer. Math / F. Brezzi — Springer Science+Business Media, 1959. Vol. 1, Iss. 1. P. 269–271.

19. PathFindingAlgorithms [Электронный ресурс]. URL: https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40(дата обращения: 12.02.2020).

20. Rote G. (1990) Path Problems in Graphs. In: Tinhofer G., Mayr E., Noltemeier H., Syslo M.M. (eds) Computational Graph Theory. ComputingSupplementum, Vol. 7, Springer, Vienna.

21. Shabina Banu Mansuri1, Shiv kumar. Comparative Analysis of Path Finding Algorithms, Journal of Computer Engineering (IOSR-JCE), Volume 20, Issue 5, Ver. I (Sep - Oct 2018), PP 38-45.

22. ShortestPaths[Электронный ресурс]. URL:http://www.cs.cmu.edu/afs/cs/academic/class/15210-s14/www/lectures/shortest-path.pdf (дата обращения: 12.02.2020).

23. The Global Optimal Algorithm of Reliable Path Finding Problem Based on Backtracking Method / [Liang Shen, Hu Shao, Long Zhang et al.] // Mathematical Problems in Engineering. 2017. Vol. 2017. P. 1–10.

24. Van der LaanJ.S. (2017) Optimal Routing Algorithms, MathematischInstituut, Universiteit Leiden.

Приложение А

Фрагмент кода формы интерфейса программы

```
//Степанов М.А.
&НаКлиенте
Перем Карта, Граф, Состояние_у0, Состояние_у1, ОписаниеАлгоритмов;

//////////////// СОБЫТИЯ ФОРМЫ

&НаКлиенте
Процедура ПриОткрытии(Отказ)
        Анимация = Истина;
    Скорость = 4;
    АлгоритмыПриИзменении(Неопределено); // Инициализируем описание алгоритмов
    Протокол(Неопределено); // Инициализируем протокол
    Протокол(Неопределено);
    А0_ГоловнойАвтоматУправления(е0_Инициализировать());
КонецПроцедуры

//////////////// СОБЫТИЯ ЭЛЕМЕНТОВ ФОРМЫ
//////////////// КОМАНДЫ
&НаКлиенте
Процедура Протокол(Команда)
    ВыводитьЛог = Не ВыводитьЛог;
    Элементы.Лог.Видимость = ВыводитьЛог;
    Если ВыводитьЛог Тогда
        Элементы.Протокол.Заголовок = "Скрыть протокол тестирования";
    Иначе
```

```
            Элементы.Протокол.Заголовок = "Показать протокол
тестирования";
        КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура ОчиститьПротокол(Команда)
        Лог.Очистить();
КонецПроцедуры

&НаКлиенте
Процедура АлгоритмыПриИзменении(Элемент)
        Если Алгоритмы >ОписаниеАлгоритмов.ВГраница() Тогда
            Описание = "";
        Иначе
            Описание = ОписаниеАлгоритмов[Алгоритмы];
        КонецЕсли;
КонецПроцедуры

//////////////// СЛУЖЕБНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ
&НаКлиенте
Функция МодульЧисла(Число)
        Возврат Макс(Число, -Число);
КонецФункции
#Область Лог
&НаКлиенте
Процедура ЛогНачалоА(НомерА, ИмяА, у, е)
        Если ВыводитьЛог Тогда
            Если Лог.КоличествоСтрок() > 1500 Тогда Лог.Очистить()
КонецЕсли;
```

```
            Лог.ДобавитьСтроку(СтрШаблон(Отступ + "[+] А%1(%4): '%2' запущен в состоянии [%3] с событием <%4>", НомерА, ИмяА, у, е));
            Отступ = Отступ + Символы.Таб;
        КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура ЛогПереходА(НомерА, ИмяА, уПрошлое, уТекущее)
        Если ВыводитьЛог Тогда
            Лог.ДобавитьСтроку(СтрШаблон(Отступ + "[>] А%1: '%2' перешел из состояния [%3] в ==> [%4]", НомерА, ИмяА, уПрошлое, уТекущее));
        КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура ЛогКонецА(НомерА, ИмяА, е, у)
        Если ВыводитьЛог Тогда
            Отступ = Лев(Отступ, СтрДлина(Отступ) - 1);
            Лог.ДобавитьСтроку(СтрШаблон(Отступ + "[-] А%1(%3): '%2' завершил обработку события <%3> в состоянии [%4]", НомерА, ИмяА, е, у));
        КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура ЛогВходногоВоздействия(ТекстВходногоВоздействия, Результат)
        Если ВыводитьЛог Тогда
            Лог.ДобавитьСтроку(СтрШаблон(Отступ + "[x] Значение [%1] - вернула <%2>", ТекстВходногоВоздействия, Результат));
```

```
        КонецЕсли;
КонецПроцедуры

&НаКлиенте
Процедура ЛогВыходногоВоздействия(ТекстВыходногоВоздействия)
        Если ВыводитьЛог Тогда
                Лог.ДобавитьСтроку(Отступ + "[z] Выполнить [" +
ТекстВыходногоВоздействия + "]");
        КонецЕсли;
КонецПроцедуры

#КонецОбласти

/////////////// ОБЪЕКТЫ УПРАВЛЕНИЯ
#Область Карта
&НаКлиенте
Процедура Карта_Новый(ШаблонКарты)
        Экран = Новый ТабличныйДокумент; // Очищаем карту на экране
        Карта = Новый Структура;
        Карта.Вставить("Стрелки",           Новый Структура("Право, Лево,
Верх, Низ", "→", "←", "↑", "↓"));
        Карта.Вставить("СтрелкиРеверс",   Новый Структура("Лево, Право,
Низ, Верх", "→", "←", "↑", "↓"));
        Карта.Вставить("РазмерКлетки", 6);
        Карта.Вставить("КоличествоКолонок",
ШаблонКарты.ШиринаТаблицы);
        Карта.Вставить("КоличествоСтрок", ШаблонКарты.ВысотаТаблицы);
        Карта.Вставить("ШаблонСтены", Спрайты.Рисунки.Стена);
        Карта.Вставить("ШаблонПусто", Спрайты.Рисунки.Пусто);
        Карта.Вставить("ШаблонОткуда", Спрайты.Рисунки.Откуда);
```

```
        Карта.Вставить("ШаблонКуда", Спрайты.Рисунки.Куда);
        Карта.Вставить("ШаблонЛеса", Спрайты.Рисунки.Лес);
        Карта.Вставить("ШаблонКарты", ШаблонКарты);
        Карта.Вставить("КлеткиНаЭкране", Новый
Массив(Карта.КоличествоКолонок + 1, Карта.КоличествоСтрок + 1));
        Карта.Вставить("ОткудаИД", "");
        Карта.Вставить("КудаИД", "");
        Карта.Вставить("Темп", "");
        Карта.Вставить("ПосещенныеВершины", Неопределено);
        Карта.Вставить("ОчередьСообщений", Очередь_Новый());
        Карта.Вставить("ЦветВыделения", Новый Цвет(255, 0, 0));
        // Подгоним размеры шаблонных картинок к установленным
        Для каждого Рис Из Спрайты.Рисунки Цикл
            Рис.Ширина = Карта.РазмерКлетки;
            Рис.Высота = Карта.РазмерКлетки;
        КонецЦикла;
КонецПроцедуры

&НаКлиенте
Функция Карта_ДобавитьРисунокПоШаблону(ШаблонРисунка, х, у)

        Рис = Экран.Рисунки.Добавить(ШаблонРисунка.ТипРисунка);
        ЗаполнитьЗначенияСвойств(Рис, ШаблонРисунка, ,"Имя, Лево, Верх");
        Рис.Текст = ШаблонРисунка.Текст;
        Рис.Лево = х;
        Рис.Верх = у;
        Возврат Рис;
КонецФункции
&НаКлиенте
Процедура Карта_ИзменитьРисунок(ТекущийРисунок, НовыйРисунок)
```

```
        ЗаполнитьЗначенияСвойств(ТекущийРисунок, НовыйРисунок, , "Имя,
Лево, Верх");
        ТекущийРисунок.Текст = НовыйРисунок.Текст;
КонецПроцедуры

&НаКлиенте
Функция Карта_ПолучитьСтрелкуРеверс(Направление)
        Возврат Карта.СтрелкиРеверс[Направление];
КонецФункции
&НаКлиенте
Функция Карта_ПолучитьСтрелку(Направление)
        Перем Стрелка;
        Возврат ?(Карта.Стрелки.Свойство(Направление, Стрелка), Стрелка,
Направление);
КонецФункции
&НаКлиенте
Функция Карта_ПолучитьИмяВершины(Знач Колонка, Знач Строка)
        Если Колонка < 1 Тогда
                Колонка = Карта.КоличествоКолонок;
        ИначеЕслиКолонка>Карта.КоличествоКолонок Тогда
                Колонка = 1;
        КонецЕсли;
        Если Строка < 1 Тогда
                Строка = Карта.КоличествоСтрок;
        ИначеЕслиСтрока>Карта.КоличествоСтрок Тогда
                Строка = 1;
        КонецЕсли;
                Возврат Карта.КлеткиНаЭкране[Колонка][Строка];
КонецФункции
```