



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана (национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## К КУРСОВОЙ РАБОТЕ

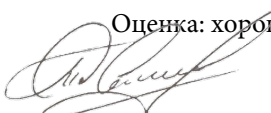
по дисциплине: «Технологии интернет»

Студент Ершов Виталий Алексеевич

Группа РК6-82Б

Тема Обзор алгоритмов DFS и BFS

|               |                                    |                                      |
|---------------|------------------------------------|--------------------------------------|
| Студент       | <u>Ершов В.А.</u><br>подпись, дата | <u>Ершов В.А.</u><br>фамилия, и.о.   |
| Преподаватель | <u>17.05.2022</u><br>подпись, дата | <u>Соколов А.П.</u><br>фамилия, и.о. |

Оценка: хорошо.  
  
А.П. Соколов

Москва, 2022 г.

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой РК-6  
(Индекс)

\_\_\_\_\_ А.П. Карпенко \_\_\_\_\_  
(И.О.Фамилия)

« \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

**З А Д А Н И Е**  
**на выполнение курсовой работы**

по дисциплине Технологии интернет

Студент группы РК6-82Б

Ершов Виталий Алексеевич

\_\_\_\_\_  
(Фамилия, имя, отчество)

Тема курсовой работы

Обзор алгоритмов DFS и BFS

Направленность **КР** (учебная, исследовательская, практическая, производственная, др.)  
учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения КР: 25% к 3 нед., 50% к 10 нед., 75% к 15 нед., 100% к 17 нед.

**Техническое задание** Провести обзор алгоритмов обхода графа DFS и BFS, проанализировать эффективность алгоритмов, представить типовые задачи для которых применяются данные алгоритмы

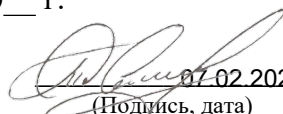
**Оформление курсовой работы:**

Расчетно-пояснительная записка на 20 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

Руководитель курсовой работы

  
(Подпись, дата)

\_\_\_\_\_ Соколов А.П.

(И.О.Фамилия)

Студент

\_\_\_\_\_ (Подпись, дата)

\_\_\_\_\_ Ершов В.А.

(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

## **АННОТАЦИЯ**

Данная работа посвящена особенностям использования алгоритмов обхода графа: поиск в глубину (Bread-First Search, BFS) и поиск в ширину (Depth-First Search, DFS). Оба алгоритма являются основой для множества других, более сложных, алгоритмов из теории графов поскольку позволяют эффективно осуществлять обход графовой модели, однако, эффективность работы алгоритмов очень сильно зависит от свойств, которыми обладает графовая модель. В работе представлен подробный обзор алгоритмов BFS и DFS.

Тип работы: курсовой проект.

Тема работы (проект темы): Аналитический обзор алгоритмов обхода графа DFS и BFS.

Объект исследований: Алгоритмы обхода графа BFS и DFS

# СОДЕРЖАНИЕ

|   |    |
|---|----|
| ВВЕДЕНИЕ .....  | 5  |
| 1. ПОСТАНОВКА ЗАДАЧИ .....                                  | 7  |
| 1.1. Концептуальная постановка задачи.....                  | 7  |
| 2. ОБЗОР АЛГОРИТМОВ ОБХОДА ГРАФА .....                      | 8  |
| 2.1. Описание алгоритма DFS .....                           | 8  |
| 2.2. Описание алгоритма BFS.....                            | 11 |
| 2.3. Анализ и сравнение алгоритмов .....                    | 12 |
| 3. ЗАДАЧИ НА ГРАФАХ.....                                    | 14 |
| 3.1. Программная реализация алгоритма поиска в глубину..... | 14 |
| 3.2. Программная реализация алгоритма поиска в ширину ..... | 15 |
| 3.3. Задача проверки графа на ацикличность.....             | 16 |
| 3.4. Задача нахождения кратчайшего пути в графе.....        | 17 |
| ЗАКЛЮЧЕНИЕ .....  | 19 |
| СПИСОК ЛИТЕРАТУРЫ.....                                      | 20 |

## ВВЕДЕНИЕ

Графовые модели применялись и активно использовались людьми во всех сферах жизни и науки еще задолго до того как графы стали изучать как отдельное направление – теорию графов в дискретной математике. Использование графовых моделей позволяет наглядно представить данные любого типа: маршрут сетевого запроса, иерархию подчинения в крупной компании, информацию о движении денежных средств. В современной науке с помощью графовых моделей обучают нейронные сети [1], строят оптимальные маршруты в системах навигации [2]. При проектировании печатных схем графовые модели используются для решения задач оптимального размещения компонент на плате.

Впервые идеи теории графов были применены математиком Леонардом Эйлером в 1736 году при доказательстве некоторых утверждений в задаче о кенигсберских мостах. Однако, в работе Эйлер не выделял какие-либо формулировки и термины теории графов. В 1869 году математик Камиль Жордан с математической точки зрения описал и исследовал дерево, а само слово «граф» в смысле теории графов было предложено позже – в 1879 году английским математиком Джеймсом Сильвестром.

В настоящее время теория графов интенсивно развивается, актуальность исследований в этой области подтверждается тем, что графовые модели используются во всех сферах нашей жизни. Каждый год возникает множество задач, решение которых было бы попросту невозможным без существования теории графов. Большую часть задач можно решить применив уже существующие алгоритмы из теории графов: алгоритмы обхода графа, поиск кратчайшего пути в графе, построение оптимального  $k$ -связного графа с  $n$  вершинами и так далее.

Абсолютно во всех алгоритмах из теории графов в том или ином виде осуществляется обход графа. Обход графа – это переход от одной вершины графа к другой в поисках свойств связей этих вершин. Существуют два алгоритма для обхода графов: поиск в глубину (Depth-First Search, DFS) и поиск в ширину (Breadth-First Search, BFS). Алгоритмы обхода графа являются фунда-

ментом для множества других алгоритмов из теории графов. Таким образом, изучение теории графов необходимо начинать с алгоритмов DFS и BFS.

# **1. ПОСТАНОВКА ЗАДАЧИ**

## **1.1. Концептуальная постановка задачи**

Для обхода графовых моделей существуют два алгоритма: поиск в глубину (DFS) и поиск в ширину (BFS). Несмотря на то, что оба алгоритма используются для решения одной задачи – эффективный обход графовой модели, они имеют ряд отличительных особенностей и применяются при решении разных задач.

Таким образом, необходимо провести анализ и сравнение алгоритмов DFS и BFS, а также рассмотреть ряд задач для решения которых используется тот или иной алгоритм.

## 2. ОБЗОР АЛГОРИТМОВ ОБХОДА ГРАФА

### 2.1. Описание алгоритма DFS

Суть алгоритма поиска в глубину (Depth-First Search, DFS) - погружаться вглубь графа следуя в определенном направлении. Движение начинается от некоторой начальной вершины по определенному пути до тех пор, пока не будет достигнута искомая вершина или конец пути. Если искомая вершина не была найден и был достигнут конец пути, то необходимо вернуться в начальную вершину и пойти по другому маршруту.

Рассмотрим работу алгоритма на примере простого ориентированного графа представленного на рисунке 1.

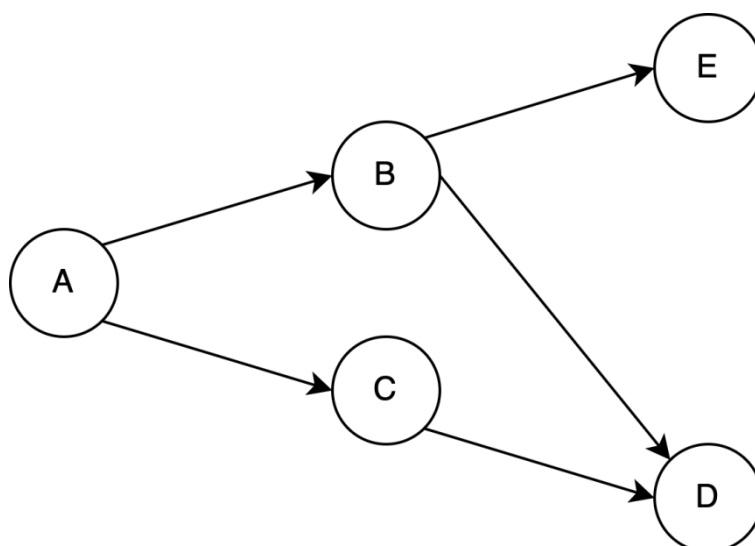


Рисунок 1 – Ориентированный граф

Мы находимся в точке «А» и хотим найти вершину «Е». Согласно принципу алгоритма DFS, необходимо исследовать один из возможных путей из точки «А». На рисунке 2 представлено изображение графа с отмеченным направлением движения из вершины «А» в вершину «С».



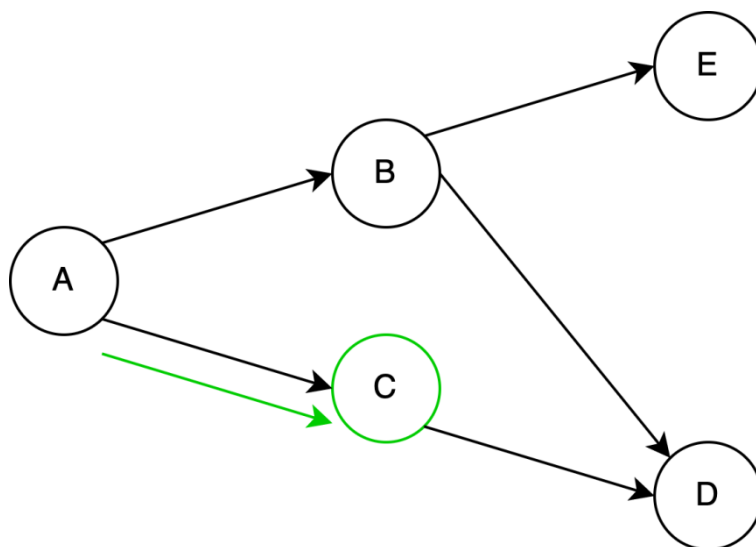


Рисунок 2 – Переход из вершины «А» в вершину «С»

Достигнув вершины «С» мы понимаем, что это не конец пути, поскольку из вершины «С» можно перейти в еще нерассмотренную ранее вершину «D». На рисунке 3 представлено изображение графа с отмеченным направлением движения из вершины «С» в вершину «D».

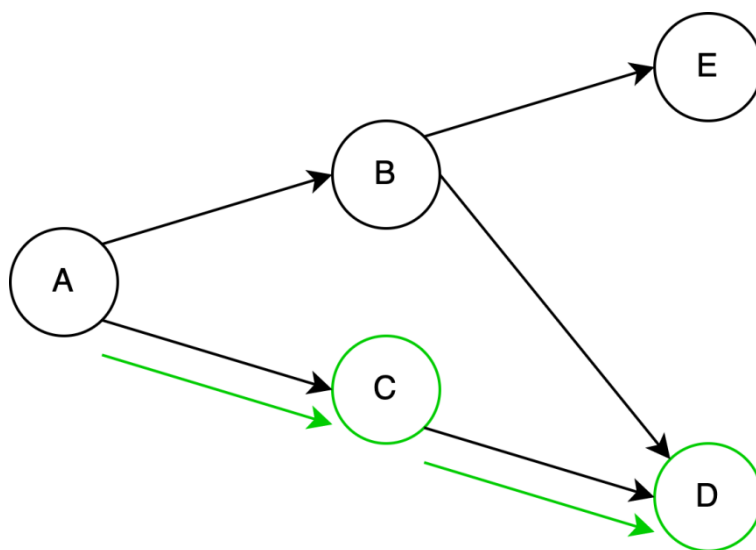


Рисунок 3 – Переход из вершины «С» в вершину «D»

Мы достигли конца пути, так как из вершины «D» не выходит ни одного ребра. Поскольку искомая вершина «E» не была найдена, возвращаемся в начальную вершину «А» и двигаемся по другому пути – из вершины «А» в вершину «B». На рисунке 4 представлено изображение графа с отмеченным направлением движения из вершины «А» в вершину «B».

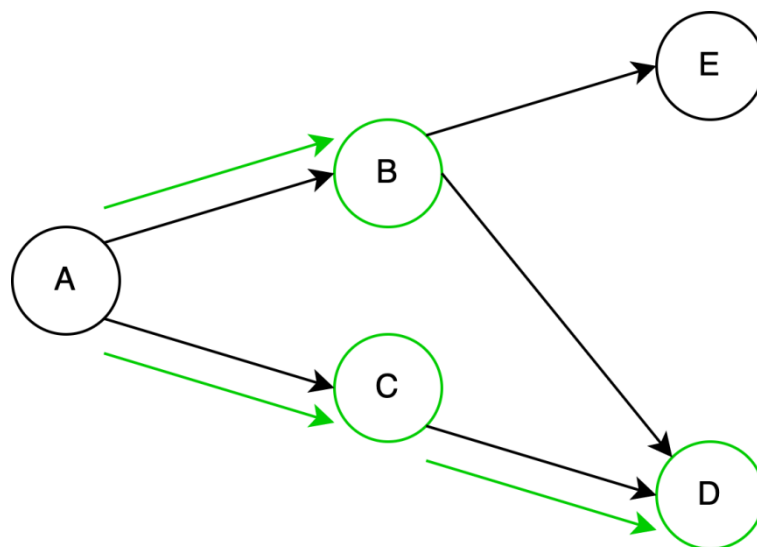


Рисунок 4 – Движение по другому пути из вершины «A» в вершину «B»

Из вершины «B» можно перейти в вершину «D», но вершина «D» уже была посещена, поэтому остается только один возможный путь из вершины «B» в вершину «E». На рисунке 5 представлено изображение графа с отмеченным направлением движения из вершины «B» в вершину «E».

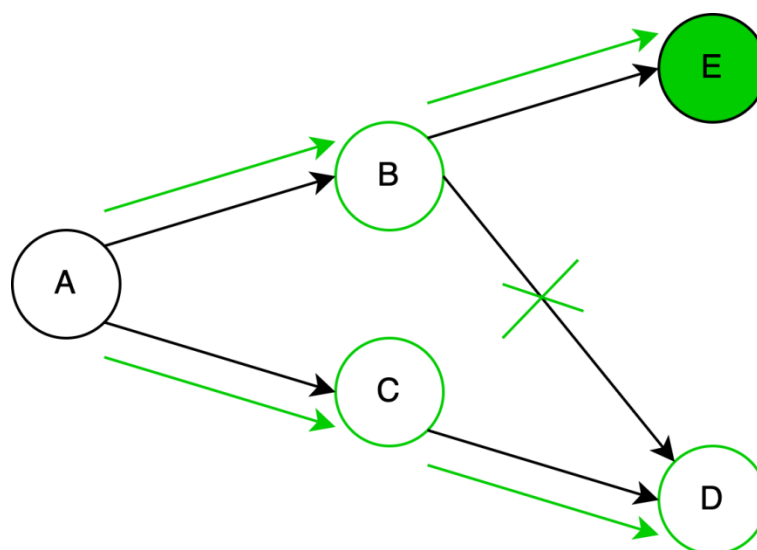


Рисунок 5 – Переход из вершины «B» в вершину «E»

Таким образом, мы достигли искомой вершины «E», следовательно можно завершать выполнение алгоритма.

Перейдем к рассмотрению алгоритма поиска в ширину BFS.

## 2.2. Описание алгоритма BFS

В отличие от алгоритма DFS, суть алгоритма BFS состоит в том, чтобы на каждом шаге продвигаться вперед по одному соседу вперед. Рассмотрим работу алгоритма на примере графа представленного на рисунке 6.

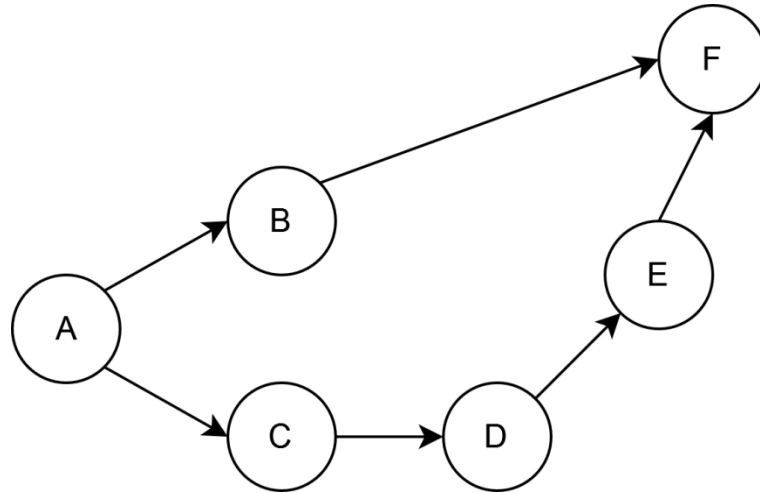


Рисунок 6 – Ориентированный граф

Допустим, что мы находимся в вершине «A» и хотим найти вершину «F». Согласно принципу алгоритма BFS на каждом шаге необходимо исследовать всех соседей, следовательно необходимо рассмотреть вершины «B» и «C». На рисунке 7 представлено изображение графа, на котором отмечены рассматриваемые вершины.

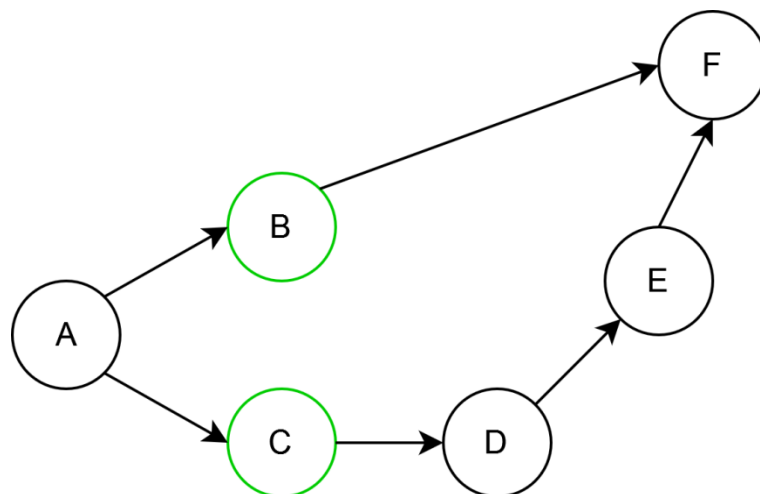


Рисунок 7 – Граф с отмеченными соседями вершины «A»

Аналогично предыдущему шагу, посещаем соседей вершин «B» и «C». Из вершины «B» можно перейти в вершину «F», а из вершины «C» в вершину «D». На рисунке 8 представлено изображение графа, на котором отмечены вершины, в которые будет совершен переход на этом шаге.

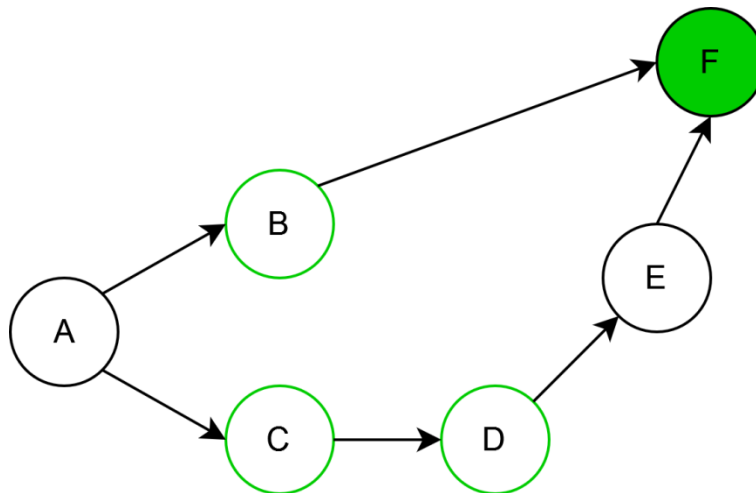


Рисунок 8 – Граф с отмеченными соседями вершин «B» и «C»

Мы достигли искомой вершины «F», следовательно обход графа можно завершать.

Перейдем к анализу и сравнению алгоритмов обхода графа DFS и BFS.

### 2.3. Анализ и сравнение алгоритмов

Для анализа алгоритмов рассмотрим асимптотическую сложность каждого из них. Введем следующие обозначения:

$V$  – количество вершин в графе

$E$  – количество ребер (граней) в графе

Начнем с алгоритма поиска в глубину (DFS). Из описания алгоритма и представленного примера можно сделать вывод, что в худшем случае необходимо обойти все вершины (асимптотическая сложность  $O(V)$ ) и исследовать все ребра (асимптотическая сложность  $O(E)$ ). Таким образом, асимптотическая сложность алгоритма составит  $O(V + E)$ .

Рассмотрим алгоритм поиска в ширину (BFS). Нетрудно заметить, что в худшем случае придется посетить все вершины графа, а следовательно, асимптотическая сложность алгоритма составит  $O(V + E)$ .

Оба алгоритма имеют одинаковую асимптотическую сложность, следовательно каждый из них одинаково эффективен. Однако, из описания алгоритмов, нетрудно заметить, что в одних задачах более рационально будет применить поиск в глубину, а в других поиск в ширину.

С точки зрения программной реализации, алгоритм поиска в глубину (DFS) использует рекурсию для обхода всех возможных путей. В то время как, алгоритм поиска в ширину (BFS) использует очередь для хранения соседних вершин и итеративно продвигается по графу шаг за шагом. Таким образом, при работе с большими графовыми моделями поиск в глубину будет крайне неэффективен, а в худшем случае будет вызывать переполнение стека и аварийное завершение программы. Однако и поиск в ширину обладает рядом недостатков, которые могут очень серьезно снизить эффективность работы алгоритма. При работе с достаточно широкими графами (у вершин очень много соседей) все соседние вершины необходимо сохранить в очереди, что несомненно потребует большой объем памяти. Также, нередко возникает ситуация, когда решения расположены очень глубоко в графе и применение поиска в ширину для их нахождения будет просто непрактично.

Таким образом, оба алгоритма достаточно эффективны, но выбирать какой из них использовать необходимо только после подробного изучения задачи и проверки всех граничных условий.

### 3. ЗАДАЧИ НА ГРАФАХ

При решении любой задачи на графах, в первую очередь, необходимо определить какой алгоритм будет использоваться для обхода графовой модели, DFS или BFS. Таким образом, обзор популярных задач на графах следует начинать с рассмотрения программной реализации алгоритмов обхода DFS и BFS.

Для общности во всех, рассматриваемых в этом разделе, задачах будем считать, что граф представлен в виде списка смежности. Это значит, что для каждой вершины мы можем за время  $O(1)$  получить все связанные с ней вершины. Также, вместо привязки к какому-либо конкретному языку программирования, будем использовать псевдо-код.

Начнем с рассмотрения программной реализации алгоритма поиска в глубину DFS.

#### 3.1. Программная реализация алгоритма поиска в глубину

Основная идея поиска в глубину это движение по определенному маршруту вплоть до его конца. Таким образом, необходимо рассмотреть все вершины в графе и для каждой из них исследовать все возможные пути. Для реализации обхода всех путей из вершины необходимо использовать рекурсию. На каждом шаге рекурсии нужно проверить, что рассматриваемая вершина не была посещена ранее. Если вершина отмечена как посещенная, то необходимо выйти из рекурсии и перейти к рассмотрению следующей непосещенной вершины графа.

Перейдем к программной реализации поиска в глубину:

```
Func: DFS
Data: Graph

visited <- [False]
for vertex in Graph.vertices do
  if visited[vertex] == False then
    visited[vertex] = True
    ProcessDFS(Graph, visited, vertex)
  end
end
end
```

```

Func: ProcessDFS
Data: Graph, visited, v

for vertex in Graph[vertex] do
  if visited[vetex] == False then
    visited[vertex] = True
    ProcessDFS(Graph, visited, vertex)
  end
end

```

Перейдем к рассмотрению программной реализации алгоритма поиска в ширину BFS.

### 3.2. Программная реализация алгоритма поиска в ширину

Суть алгоритма BFS состоит в том, чтобы на каждой итерации просматривать соседние вершины. Таким образом, выполнение алгоритма необходимо начинать с некоторой начальной вершины, а затем на каждой итерации просматривать соседние вершины. Аналогично поиску в глубину, нам необходим массив или хэш-таблица, чтобы запоминать посещенные вершины. Для этого необходима дополнительная структура данных, которая позволяет за линейное время  $O(1)$  получить, а затем удалить элемент. Структуры данных стек и очередь удовлетворяют этому требованию. Разницы в скорости работы при использовании стека и при использовании очереди нет, однако на практике принято использовать очередь.

Перейдем к программной реализации поиск в ширину:

```

Func: BFS
Data: Graph, start_vertex

visited <- [False]
queue.push(start_vertex)

while queue.empty() != true do
  current_vertex = queue.front()
  queue.pop()

  for vertex in Graph[current_vertex] do
    if visited[vertex] == False then
      queue.push(vertex)
      visited[vertex] = True
    end
  end
end
end

```

Перейдем к рассмотрению популярных задач из теории графов. Начнем с задачи проверки графа на ацикличность.

### 3.3. Задача проверки графа на ацикличность

Для решения данной задачи необходимо рассмотреть все вершины графа и для каждой из вершин исследовать все возможные маршруты из нее. Если в процессе движения по маршруту мы пришли в стартовую вершину, то это значит, что мы нашли цикл – можно завершать выполнение программы с сообщением о том, что в графе есть циклы. Из этого следует, что для обхода графа необходимо использовать поиск в глубину. Заметим, что у каждой вершины графа может быть 3 состояния:

- вершина еще не рассмотрена (обозначим как *not\_visited*)
- вершина уже была рассмотрена ранее (обозначим как *visited*)
- вершина находится на рассматриваемом в данный момент маршруте (обозначим как *in\_current\_route*)

Теперь можно перейти к программной реализации задачи проверки графа на ацикличность.

Func: CheckIsAcyclic

Data: Graph

Return: Bool

```
visited <- [not_visited]
for vertex in Graph.vertices do
  if visited[vertex] == not_visited then
    visited[vertex] = in_current_route
    if ProcessDFS(Graph, visited, vertex) == True then
      print('Graph has cycles')
      return False
    end
  end
end
```

Func: ProcessDFS

Data: Graph, visited, v

Return: Bool

```
for vertex in Graph[v] do
  if visited[vertex] == in_current_route then
    return True
  else if visited[vertex] == not_visite then
    visited[vertex] = in_current_route
    return ProcessDFS(Graph, visited, vertex)
  end
end
end
```



```
visited[v] = visited
```

Если потребуется восстановить цикл, то необходимо дополнительно использовать структуру данных стек и в процессе следования по конкретному маршруту добавлять вершины в стек. Таким образом, когда будет найден цикл, в стеке будут находиться все пройденные вершины.

### 3.4. Задача нахождения кратчайшего пути в графе

Рассмотрим задачу нахождения кратчайших путей от заданной вершины до всех остальных вершин графа. Для решения данной задачи можно использовать как поиск в ширину, так и поиск в глубину. Заметим, что нахождение кратчайшего пути между двумя точками — это достаточно распространенная задача, которая применяется при построении маршрута в навигационных системах, используется аналитиками данных. Также поиск кратчайшего пути лежит в основе нескольких протоколов маршрутизации. Обычно в подобных ситуациях необходимо работать с графовыми моделями очень большого размера, поэтому, как ранее было отмечено, поиск в глубину будет работать крайне неэффективно или не будет работать вовсе. Следовательно, для решения задачи нахождения кратчайшего пути в графе необходимо использовать поиск в ширину BFS.

С точки зрения программной реализации задача нахождения кратчайшего пути не сильно отличается от простой реализации поиска в ширину. Необходимо добавить дополнительную структуру данных: массив или хэш-таблицу, в которой будут храниться минимальные расстояния между начальной вершиной и остальными вершинами графа. Перейдем к программной реализации нахождения кратчайшего пути в графе.

```
Func: ShortestPaths
Data: Graph, start_vertex
Return: dist

dist <- [Graph.size]
dist[start_vertex] = 0
queue.push(start_vertex)

while queue.empty() != true do
```

```

current_vertex = queue.front()
queue.pop()

for vertex in Graph[current_vertex] do
    if dist[vertex] > dist[current_vertex] + 1 then
        dist[vertex] = dist[current_vertex] + 1
        queue.push(vertex)
    end
end
end

return dist

```

В рамках данной работы не получится рассмотреть все популярные задачи из теории графов, поэтому далее представлена таблица, в которой для BFS и DFS перечислены решаемые с их помощью задачи.

| Поиск в глубину (DFS)   | Поиск в ширину (BFS)  |
|---|---|
| 1) Поиск циклов в графе<br>2) Топологическая сортировка<br>3) Проверка что граф сильно связан | 1) Определение того, имеет ли граф связанные компоненты или нет.<br>2) Обнаружение двудольного графа<br>3) Поиск кратчайшего пути |

## **ЗАКЛЮЧЕНИЕ**

В данной работе проведено подробное исследование алгоритмов обхода графа DFS и BFS. Оба алгоритма имеют одинаковую асимптотическую сложность, но фактически эффективность зависит от свойств графовой модели. В глубоких графах поиск в ширину будет работать быстрее нежели поиск в глубину, но в широких графовых моделях для поиска в ширину может потребоваться очень большой объем памяти, в то время как, поиск в глубину потребляет память линейно. Таким образом, при выборе между DFS и BFS необходимо очень тщательно исследовать графовую модель с которой предстоит работать.

## **СПИСОК ЛИТЕРАТУРЫ**

1. K. Kutzkov Machine Learning on Graphs 2021 URL:  
<https://towardsdatascience.com/machine-learning-on-graphs-part-1-9ec3b0bd6abc>
2. Yang, Liping & Worboys, Michael. (2015). Generation of navigation graphs for indoor space. International Journal of Geographical Information Science. 10.1080/13658816.2015.1041141.