Министерство науки и высшего образования РФ

ФГАОУ ВПО

Национальный исследовательский технологический университет «МИСИС»

Институт компьютерных наук (ИКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

Отчет по теме «Метод поиска в глубину (DFS) и построение компонент связности в графе»

по дисциплине «Комбинаторика и теория графов»

Выполнила:

Студентка БИВТ-23-6

Мушкарина В.А.

Москва 2024

Введение

Метод поиска в глубину (DFS, от англ. Depth-First Search) — один из основных алгоритмов для обхода и поиска в графах. Он применяется для нахождения пути между вершинами, построения дерева обхода, определения связности графа и в решении многих задач в компьютерных науках, таких как поиск в лабиринте, анализ сетей и др. В этом докладе рассмотрим принцип работы поиска в глубину, его особенности и область применения.

Основные понятия

Прежде чем углубляться в детали алгоритма, разберем, что такое граф и как он представлен.

1. Граф — это структура данных, состоящая из множества вершин (узлов) и соединяющих их ребер (дорожек). Графы бывают:

- Ориентированные — когда движение между вершинами возможно только в определенном направлении;
- Неориентированные — когда движение возможно в обоих направлениях.

2. Поиск в графе — процесс обхода всех вершин и ребер графа с определенной целью. В зависимости от порядка обхода выделяют разные алгоритмы поиска: поиск в глубину и поиск в ширину.

Принцип работы поиска в глубину

Алгоритм поиска в глубину начинает обход с определенной стартовой вершины и направляется по одному из ребер к следующей вершине, после чего рекурсивно повторяет процесс. При этом алгоритм «погружается» в граф до тех пор, пока не достигнет конечной вершины или вершины, из которой нет непосещенных соседей. После этого он возвращается назад и проверяет, есть ли непосещенные

вершины на других уровнях, поднимаясь таким образом «вверх» по пути, пока не найдет такие вершины или не обойдет весь граф.

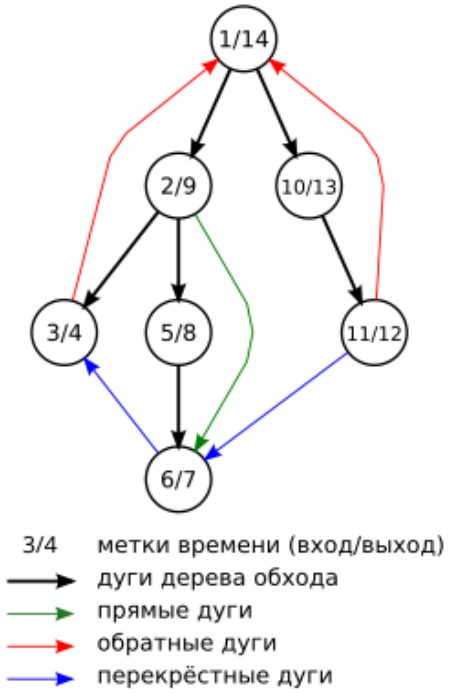
Для графа с V вершинами и E рёбрами временная сложность DFS составляет:

- **O(V + E)** — поскольку каждая вершина и каждое ребро обрабатываются не более одного раза.

Алгоритм

1. Посещение вершины — при входе в новую вершину алгоритм помечает ее как посещенную, чтобы избежать повторного посещения и возможного зацикливания.
2. Переход к соседям — далее выбирается непосещенный сосед и запускается процесс обхода уже для этой вершины.
3. Возврат и проверка других соседей — если в текущей вершине все соседи посещены, алгоритм возвращается на предыдущую вершину, откуда продолжает поиск среди других соседей.

Этот процесс продолжается до тех пор, пока не будут посещены все вершины, доступные из начальной.



Пример кода

```
visited = [False] * (n + 1)
prev = [None] * (n + 1)

def dfs(vertex, visited, prev, graph):
    visited[vertex] = True
    for neighbor in graph[vertex]:
        if not visited[neighbor]:
            prev[neighbor] = vertex
            dfs(neighbor, visited, prev, graph)
dfs(start, visited, prev, graph)
```

1. **Инициализация массивов visited и prev:**

   - visited — это массив булевых значений, где visited[i] = True, если вершина i была посещена. Инициализируется значением False для всех вершин.
   - prev — массив для хранения предшественников каждой вершины. Изначально все элементы None, так как предшественники не назначены.

2. **Функция dfs**:

Параметры:

   - vertex — текущая вершина, с которой начинается или продолжается обход.
   - visited — массив для отслеживания посещенных вершин, чтобы избежать повторных посещений и зацикливания.
   - prev — массив для хранения предшественников, что помогает восстанавливать путь.
   - graph — представление графа, передается в виде списка смежности (где graph[i] — список вершин, смежных с вершиной i).

3. **Обработка текущей вершины**:

   - Помечаем текущую вершину vertex как посещенную, устанавливая visited[vertex] = True.

4. **Проход по соседним вершинам**:

   - Для каждой вершины neighbor, смежной с текущей вершиной vertex, проверяем, посещена ли она.
   - Если neighbor не была посещена (visited[neighbor] == False), устанавливаем текущую вершину vertex как предшественника для neighbor (prev[neighbor] = vertex) и рекурсивно вызываем dfs для neighbor.

5. **Запуск DFS**:

   - После определения функции мы вызываем dfs(start, visited, prev, graph), где start — начальная вершина для обхода графа.

Особенности алгоритма DFS

Рекурсивность. Поиск в глубину часто реализуют с помощью рекурсии, поскольку алгоритм естественно возвращается назад при достижении «конца пути». В некотрых случаях, особенно когда граф очень велик, может возникать проблема переполнения стека вызовов.

Избежание зацикливания. Поскольку граф может содержать циклы, необходимо помечать посещенные вершины, чтобы избежать бесконечной рекурсии.

Построение дерева обхода. Одной из задач поиска в глубину является построение дерева обхода, где каждая вершина имеет своего «родителя» — предшественника, из которого был произведен переход.

Применение поиска в глубину

Поиск в глубину — универсальный инструмент, и его применение охватывает множество областей:

- Анализ связности графа. DFS используется для проверки, являются ли вершины графа связными, то есть можно ли добраться из одной вершины в любую другую.
- Проверка на наличие циклов. Поиск в глубину помогает найти циклы в графе, что полезно, например, в анализе зависимостей в программировании.
- Поиск наибольшей компоненты связности. Можно найти все компоненты связности и выделить наибольшую.
- Топологическая сортировка. DFS используется для упорядочивания вершин ориентированного ациклического графа.

Сравнение с другими алгоритмами

Пример задачи:

Дан граф, который представлен в виде списка смежности. Каждая вершина графа связана с другими вершинами. Необходимо определить, существует ли путь между двумя заданными вершинами, и вывести этот путь, если он существует.

Граф представлен в виде словаря, где ключи — это вершины, а значения — это списки соседних вершин.

```python
def dfs(graph, start, end, path, visited):
    path.append(start)       # Добавляем текущую вершину в путь
    visited.add(start)        # Отмечаем вершину как посещенную

    # Если мы достигли конечной вершины, возвращаем True
    if start == end:
        return True

    # Проходим по всем соседним вершинам текущей вершины
    for neighbor in graph[start]:
        if neighbor not in visited:  # Если соседняя вершина не посещена
            if dfs(graph, neighbor, end, path, visited):  # Рекурсивно вызываем dfs для соседа
                return True  # Если путь найден, возвращаемся

    # Если все соседи уже проверены и путь не найден, удаляем вершину из пути
    path.pop()
    return False

def find_path(graph, start, end):
    path = []            # Список для хранения пути
    visited = set()      # Множество для отслеживания посещенных вершин

    if dfs(graph, start, end, path, visited):
        return path
    else:
        return "Путь не существует"
```

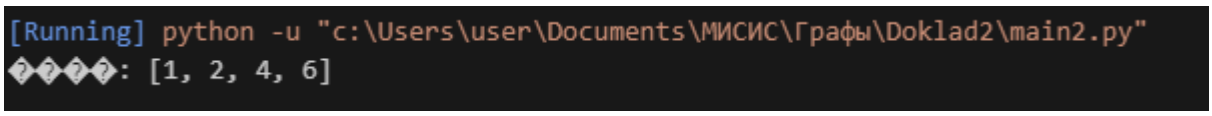
Входные данные:

```python
graph = {
    1: [2, 3],
    2: [4],
    3: [5],
    4: [6],
    5: [],
    6: []
}

# Стартовая и конечная вершины
start = 1
end = 6

# Поиск пути
path = find_path(graph, start, end)
print("Путь:", path)
```

Выходные данные:



Вывод:

В заключение, DFS — это универсальный и мощный инструмент для работы с графами, особенно полезный в задачах, где важен полный обход или исследование всех возможных путей. Простота реализации и умеренная сложность делают его одним из базовых алгоритмов, изучаемых в информатике и используемых на практике.