



Алгоритмы обхода связного графа

Разберём связность, цель обхода и разницу DFS и BFS. Поймём, как память и данные влияют на сложность.

Что значит «обойти» связный граф

Связный граф позволяет добраться между вершинами по цепочке рёбер. Обход нужен, чтобы системно посетить вершины, проверить связность, найти пути, разобрать структуру сети и построить дерево обхода.



Как граф хранится в памяти компьютера

01

Тремя основными вариантами выступают список смежности, матрица смежности и список рёбер. У каждого своя «логика доступа» к соседям, а значит — разный вклад в скорость обхода.

02

Список смежности хранит только существующие связи и занимает $O(V+E)$ памяти. Он особенно эффективен для разреженных графов, где рёбер значительно меньше, чем всех возможных пар вершин.

03

Матрица смежности требует $O(V^2)$ памяти, зато быстро отвечает на вопрос «есть ли ребро?». При обходе чаще приходится просматривать строки матрицы, что влияет на практическую эффективность.

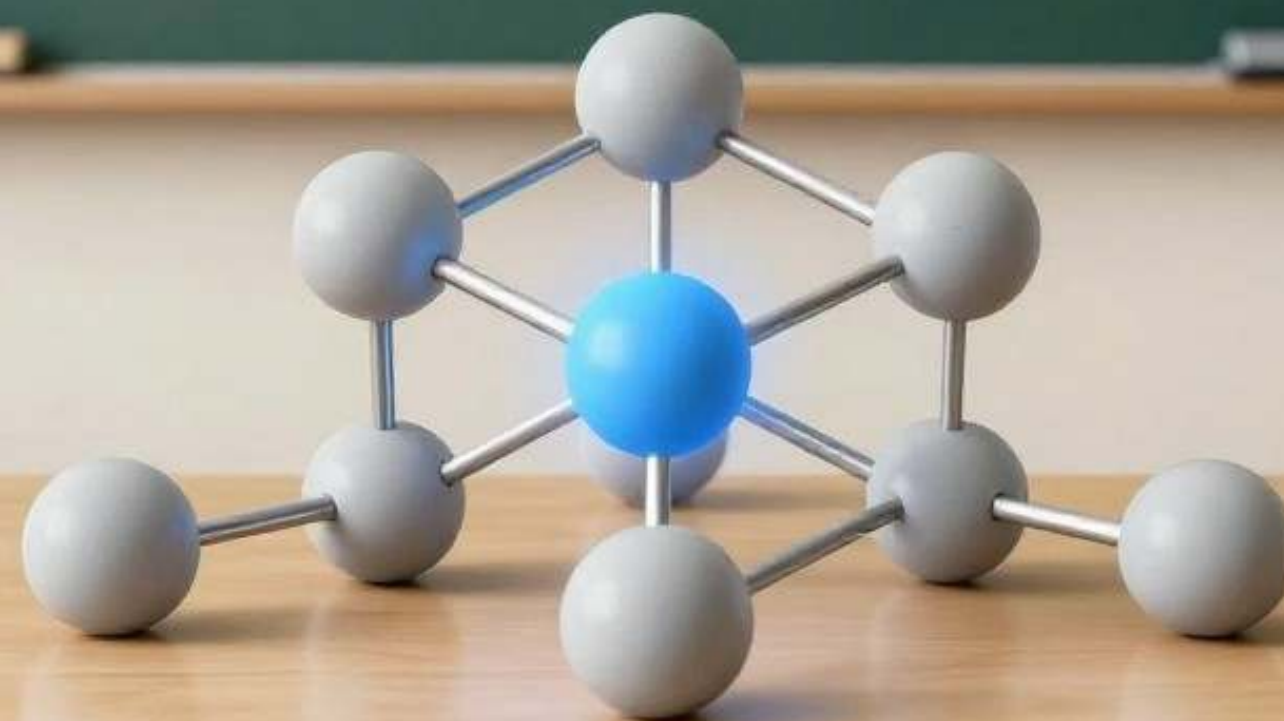
Обход графа как последовательность посещений

Старт и отметки

Любой обход начинается с заранее выбранной стартовой вершины. Чтобы не возвращаться к уже обработанным данным, каждую посещённую вершину помечают. Это защищает от повторов и циклов в логике выполнения.

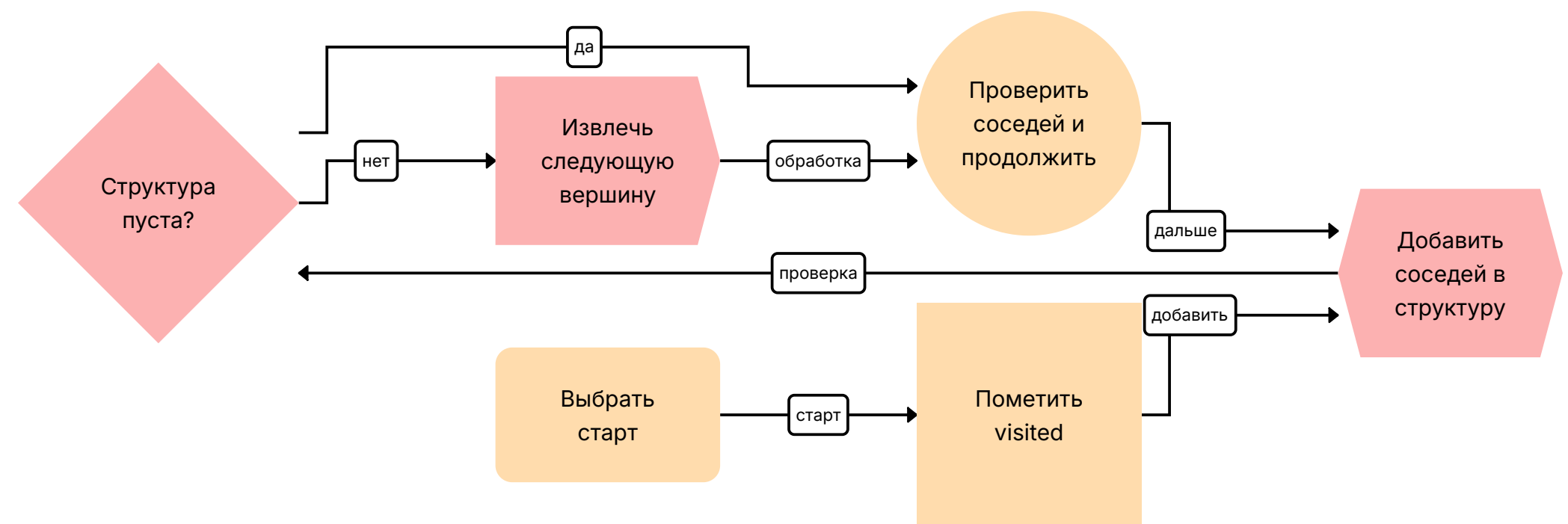
Переходы к соседям

Дальше алгоритм следует правилу перехода: из текущей вершины рассматриваются соседи, которые ещё не посещены. Для визуального объяснения используют маршруты с выделением ребра, отметками «посмотрено» и показом структуры данных.



Обход связного графа: от старта до всех достижимых вершин

Структура данных (очередь или стек) определяет стратегию обхода и порядок извлечения вершин.



Поиск в глубину: принцип работы

1

DFS, или поиск в глубину, продвигается по одному пути максимально далеко. Если дальше идти нельзя, алгоритм возвращается назад к последней точке разветвления и пробует другие варианты.

2

Обычно DFS реализуют рекурсией или явным стеком: вершины «в ожидании» лежат в структуре и извлекаются в обратном порядке. Это естественно отражает идею отката.

3

На практике DFS помогает находить компоненты связности, обнаруживать циклы и строить дерево обхода. Его удобно применять, когда важна именно структура связей между вершинами, а не уровни расстояния.

Поиск в ширину: принцип работы

01

BFS, или поиск в ширину, рассматривает вершины по уровням расстояния от старта. Сначала обрабатываются соседи, затем вершины на два ребра, и так далее, пока не исчерпаются достижимые.

Алгоритм использует очередь FIFO: новые вершины ставятся в конец, а обработка идёт по порядку поступления. Благодаря этому не нарушается очередность «от близкого к дальнему».

02

03

BFS особенно полезен для невзвешенных графов: он находит кратчайшие пути по числу рёбер. Также он хорошо подходит для анализа распространения информации по сети, где важны уровни доступа.

Параметр	DFS	BFS	Смысл
Структура	стек/рекурсия	очередь FIFO	куда кладём кандидаты
Порядок	глубже → назад	уровни от старта	как формируется обход
Кратчайший путь	не гарантирован	гарантирован	в невзвешенном графе
Доп. память	$O(V)$	$O(V)$	visited и контейнер
Типичные задачи	циклы, структура	расстояния, уровни	что легче решать

DFS и BFS: сравнение основных характеристик

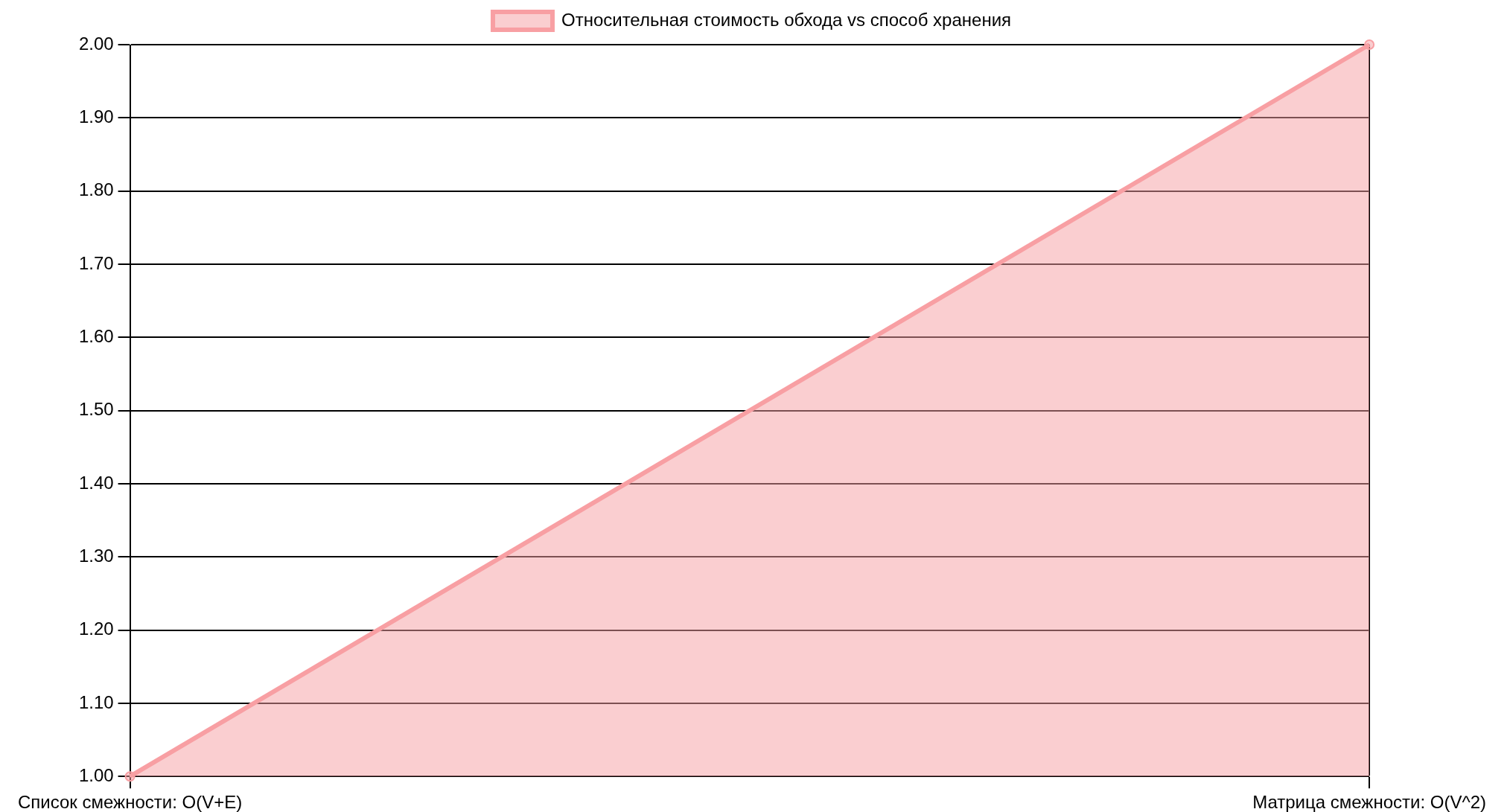
Сравним, чем эти два обхода отличаются на уровне данных и результата: порядок посещения, память и типичные задачи.

DFS чаще «раскрывает» структуру и циклы, потому что уходит глубже; BFS закономерно даёт кратчайшие пути, так как расширяет фронт по уровням.

Сложность обхода при разных способах хранения графа

Когда граф разреженный, E заметно меньше V^2 , поэтому список смежности чаще выигрывает по скорости и экономит память.

Форма представления графа меняет практическую эффективность обхода: теоретические оценки напрямую отражаются на том, сколько соседей реально перебирается.



Учебные сведения по сложности обхода графов; опора на асимптотику $O(V+E)$ и $O(V^2)$.

Временная и пространственная сложность DFS и BFS

Если хранить граф списком смежности, и DFS, и BFS обходят все вершины и рёбра за $O(V+E)$, где V — число вершин, E — число рёбер. Это базовая оценка «по знакомым связям».

01

При матрице смежности проверка соседей усложняется до $O(V^2)$: для каждой вершины приходится просматривать всю строку матрицы, даже если ребро фактически отсутствует.

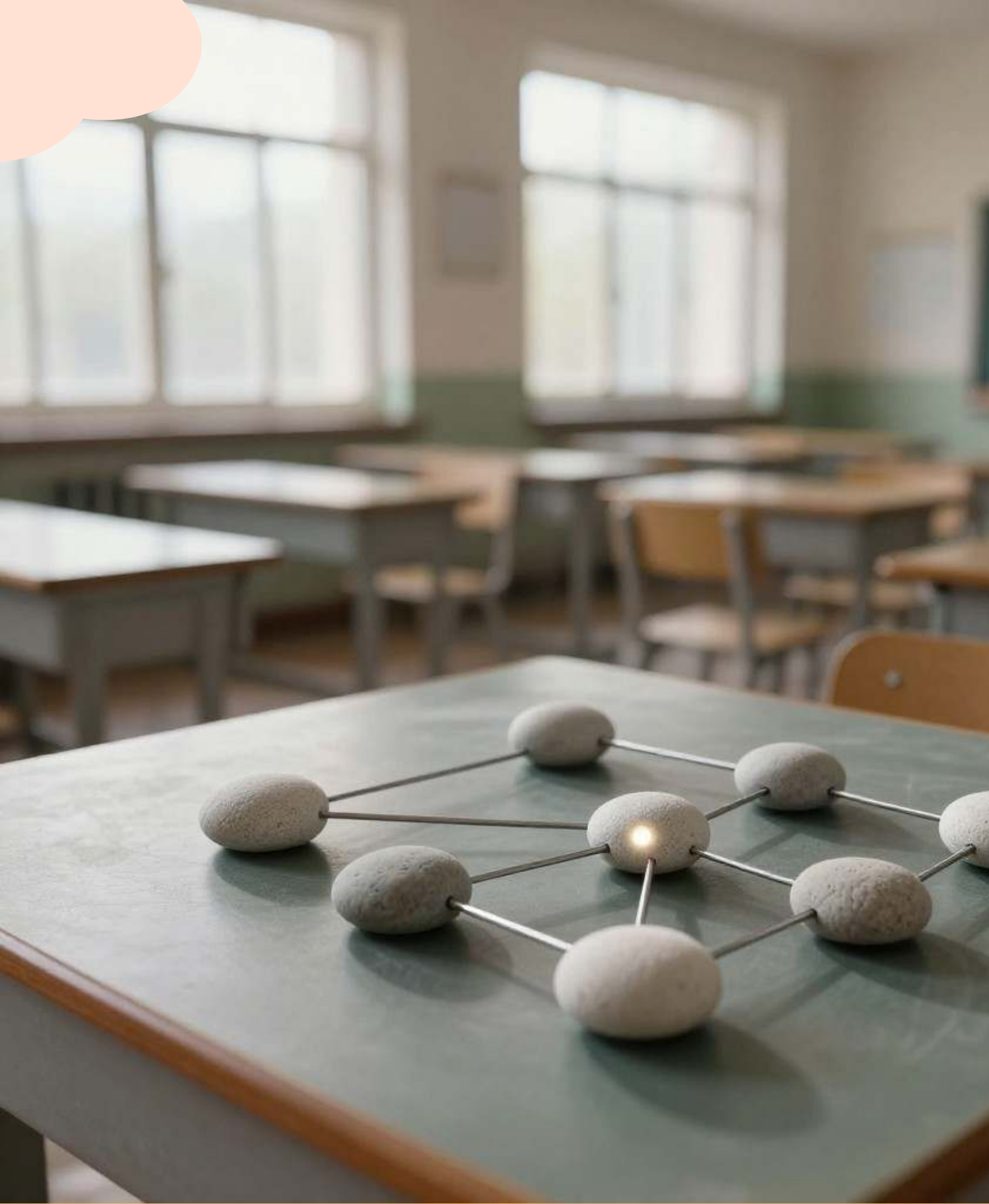
03

02

Дополнительная память обычно равна $O(V)$: массив `visited` фиксирует факт посещения, а стек (для DFS) или очередь (для BFS) хранит вершины-кандидаты. При этом обход не требует хранения всех путей целиком.

04

Итог: теоретическая сложность одинакова для DFS и BFS при одинаковом хранении графа, но разная форма представления резко влияет на реальное время за счёт того, как ищутся соседи.



Пример обхода на небольшом графе

Возьмём граф из 7 вершин: у него две «ветви», которые расходятся от старта, и есть цикл. На рисунке отмечают нумерацию вершин и рёбра, чтобы одинаковый граф был одинаково понятен.

Для DFS вершины продвигаются «вглубь», затем откатываются, как только дальше по одному пути нельзя. Для BFS посещение идёт слоями по расстоянию от старта, поэтому порядок оказывается принципиально другим.

Проверка связности и дерево обхода

Выбор стартовой вершины

Начинаем обход из одной фиксированной вершины, помечаем её как посещённую и последовательно обрабатываем достижимых соседей по выбранному правилу (DFS или BFS).

Формирование дерева обхода

При первом обнаружении каждой новой вершины сохраняют ребро «кто откуда» — так возникает дерево обхода. Оно показывает путь открытия вершин без повторных переходов.

Проверка достижимости

После завершения обхода сравниваем, сколько вершин отметили посещёнными. Если достигнуты все вершины, связность подтверждена для неориентированного графа.

Поиск кратчайшего пути в невзвешенном графе

01

DFS и BFS по-разному «раскрывают» граф, но именно BFS гарантирует кратчайший путь в невзвешенном графе по числу рёбер. Идея опирается на обработку вершин по слоям от старта.

02

Чтобы восстановить конкретный маршрут, для каждой вершины хранят расстояние и предка — вершину, из которой она была впервые обнаружена. Тогда путь собирают в обратном порядке от цели.

03

Такой подход применяют в задачах навигации по сетям, где все переходы равноправны: минимизируют количество шагов между объектами и анализируют структуру связей в графовых моделях.

Типичные ошибки при реализации обхода

1

Самая частая ошибка — забыть массив `visited` или аналогичный признак. Тогда вершины многократно попадают в очередь/стек, обход начинает повторяться и может даже привести к существенному росту времени.

2

Ещё одна проблема — некорректная обработка петель и кратных рёбер. Даже если само ребро ведёт в ту же вершину или соединяет пару вершин несколько раз, повторные добавления нужно блокировать правилами посещения.

3

Для DFS на больших графах риск связан с переполнением стека рекурсии: при глубокой цепочке рекурсивная реализация может завершиться ошибкой. Надёжнее контролировать глубину или использовать явный стек.

4

Для BFS критично не нарушить работу очереди FIFO. Если случайно поменять порядок извлечения вершин или неправильно обновлять расстояния, слоевой характер исчезает, и кратчайшие пути перестают быть корректными.

Какой алгоритм выбрать для конкретной задачи

Выбор алгоритма обхода зависит от того, что именно нужно получить: кратчайшие пути по числу рёбер, проверка связности, поиск циклов или разбиение на компоненты. Ориентируются на цель и структуру графа.

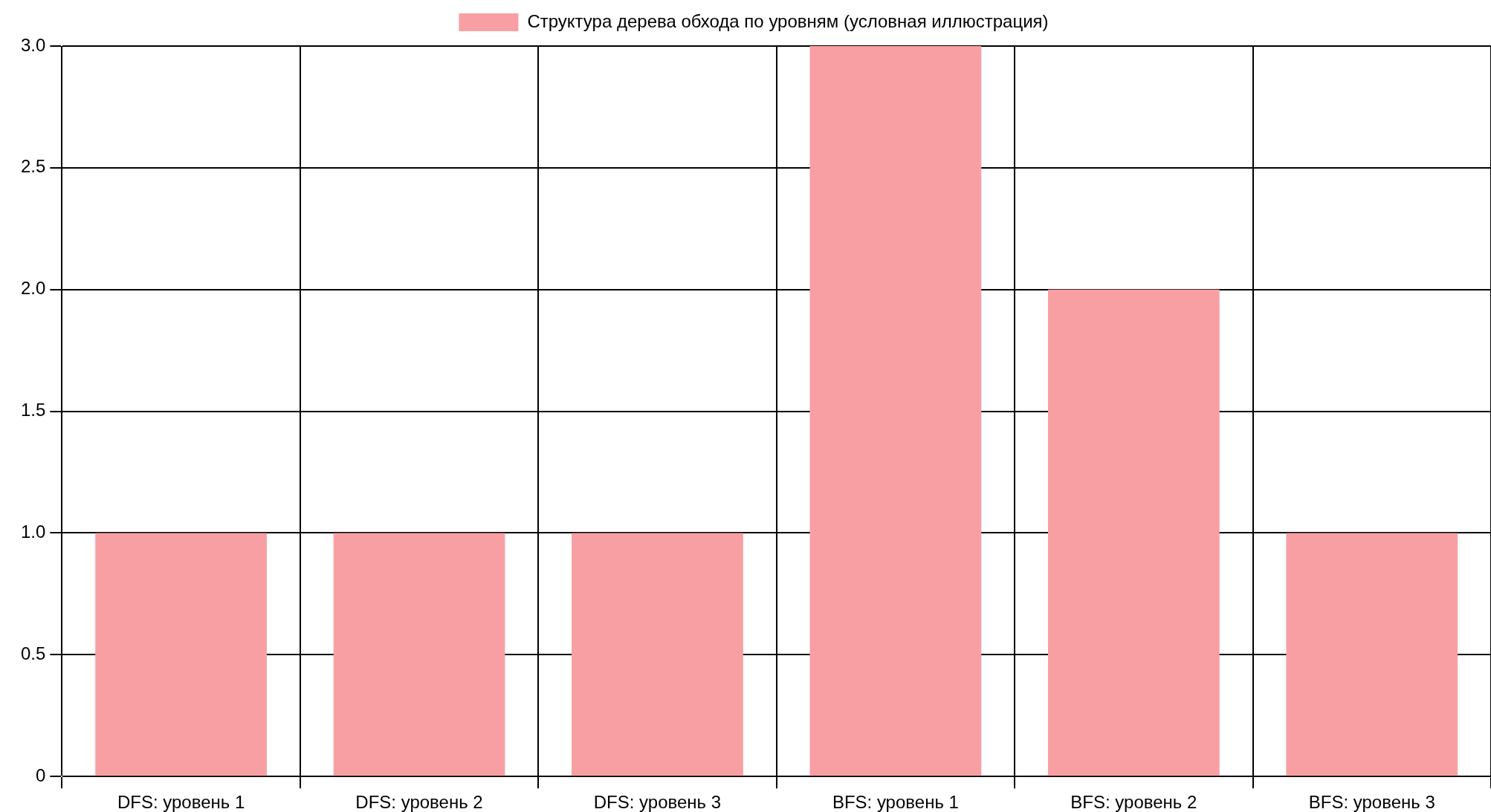
Практический вывод: алгоритм выбирают не «по привычке», а по ожидаемому свойству результата. BFS отвечает за расстояния и минимальное число рёбер, DFS — за структуру и разбиение на части.

Задача	Подходящий алгоритм	Причина выбора
Кратчайший путь без весов	BFS	Рассматривает вершины по уровням
Поиск циклов	DFS	Удобно следить за структурой обхода
Проверка связности	DFS или BFS	Если из старта достигаем всех
Компоненты связности	DFS	Глубокий обход естественно собирает компонент
Уровневый анализ сети	BFS	Достаёт расстояния и слои от старта

Различие дерева обхода в глубину и в ширину

У DFS дерево обычно «вытягивается» — новые вершины появляются на больших глубинах по одной цепочке. У BFS вершины группируются по слоям, поэтому на ранних уровнях их больше.

Диаграмма подсказывает, почему BFS удобно использовать для расстояний и кратчайших путей, а DFS — для анализа связности и циклов: формы деревьев заранее отражают нужные свойства.



Обобщение поведения DFS и BFS на примерах деревьев обхода

Применения обхода графов в реальных системах



Социальные сети

Чтобы понять, насколько пользователи связаны друг с другом, строят граф взаимодействий и обходят его от выбранного человека. Так выявляют компоненты, группы и потенциал распространения контента.



Навигационные системы

В картах и маршрутизаторах рассматривают сеть дорог как граф. BFS помогает находить маршрут с минимальным числом переходов там, где каждый шаг одинаково «дорог».



Компиляторы и зависимости

При анализе зависимостей между модулями обход помогает собрать, что нужно для сборки и в каком порядке. DFS удобен, когда важно обнаружить циклические зависимости или структуру связей.



Поиск и обход страниц

Поисковые роботы исследуют гиперссылки как граф. Обход связанных страниц позволяет быстро наращивать покрытие, а разные стратегии помогают контролировать глубину и приоритеты.

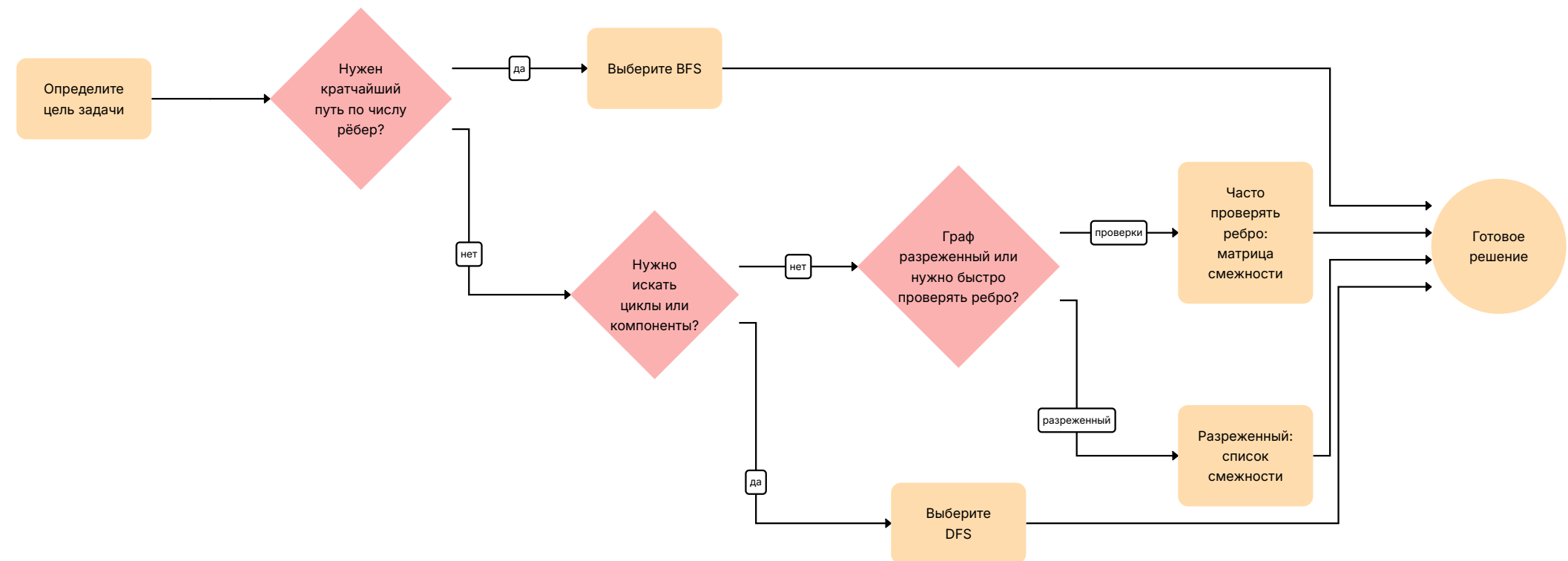


Компьютерные игры

В игровых мирах граф соответствует навигационной сетке. Обход используется для построения маршрутов перемещения, планирования действий и оценки доступности зон, с учётом ограничений карты.

Выбор обхода в зависимости от задачи

Логика выбора учитывает цель (расстояния, кратчайшие пути, циклы, компоненты) и ограничения по хранению графа (разреженность, матрица/список смежности).





Один граф — два разных маршрута обхода

01

На одном и том же графе при одинаковой стартовой вершине DFS и BFS дают разные последовательности посещения. DFS уходит глубже по одной ветви, а затем возвращается, когда упирается в конец пути.

02

BFS же расширяется слоями: сначала посещает все вершины на расстоянии 1 ребра, затем на расстоянии 2 и так далее. Поэтому порядок открытия вершин отличается даже без изменений в рёбрах графа.

Итоги: что важно знать об обходе связного графа

Обход связного графа лежит в основе анализа сетей: DFS помогает изучать структуру, находить циклы и компоненты связности. BFS удобен для уровней, расстояний и кратчайших путей без весов. Выбирайте алгоритм по цели, размеру графа и способу хранения.