

Оглавление

Предисловие	11
Эта книга для меня?.....	12
Но разве computer science не только для ученых?	13
Глава 1. Основы	14
1.1. Идеи.....	15
Блок-схемы	15
Псевдокод	17
Математические модели.....	18
1.2. Логика.....	20
Операторы	21
Булева алгебра	23
Таблицы истинности.....	25
Логика в вычислениях.....	29
1.3. Комбинаторика.....	31
Правило умножения	31
Перестановки	32
Перестановки без повторений.....	34
Комбинации	35
Правило суммирования	36
1.4. Вероятность	38
Подсчет количества возможных вариантов	38
Независимые (совместные) события.....	39
Несовместные события.....	40
Взаимодополняющие события	40
«Заблуждение игрока»	41
Более сложные вероятности.....	42
Подведем итоги	42
Полезные материалы	43

Глава 2. Вычислительная сложность	44
Надейтесь на лучшее, но готовьтесь к худшему	45
2.1. Оценка затрат времени	47
Понимание роста затрат	48
2.2. Нотация « <i>O</i> большое»	50
2.3. Экспоненциальное время	52
2.4. Оценка затрат памяти	54
Подведем итоги	55
Полезные материалы	56
Глава 3. Стратегия	57
3.1. Итерация	58
Вложенные циклы и степенные множества	59
3.2. Рекурсия	62
Рекурсия против итераций	63
3.3. Полный перебор	64
3.4. Поиск (перебор) с возвратом	67
3.5. Эвристические алгоритмы	71
«Жадные» алгоритмы	71
Когда жадность побеждает силу	73
3.6. Разделяй и властвуй	75
Разделить и отсортировать	75
Разделить и заключить сделку	80
Разделить и упаковать	82
3.7. Динамическое программирование	84
Мемоизация Фибоначчи	84
Мемоизация предметов в рюкзаке	85
Лучшая сделка снизу вверх	86
3.8. Ветви и границы	88
Верхние и нижние границы	88
Ветви и границы в задаче о рюкзаке	89
Подведем итоги	92
Полезные материалы	93
Глава 4. Данные	94
Абстракции	95
Тип данных	96

4.1. Абстрактные типы данных.....	96
Преимущества использования АТД.....	97
4.2. Общие абстракции	98
Примитивные типы данных	98
Стек	99
Очередь	100
Очередь с приоритетом	100
Список	101
Сортированный список	102
Множество	103
4.3. Структуры	104
Массив	104
Связный список	105
Двусвязный список.....	107
Массивы против связных списков	108
Дерево	109
Двоичное дерево поиска	112
Двоичная куча.....	115
Граф	117
Хеш-таблица	117
Подведем итоги	118
Полезные материалы	119
Глава 5. Алгоритмы	120
5.1. Сортировка.....	121
5.2. Поиск	124
5.3. Графы	125
Поиск в графах.....	126
Раскраска графов	129
Поиск путей в графе	130
PageRank.....	133
5.4. Исследование операций	133
Задачи линейной оптимизации.....	134
Задачи о максимальном потоке в Сети	137
Подведем итоги	138
Полезные материалы	139

Глава 6. Базы данных.....	140
6.1. Реляционная модель	142
Отношения.....	142
Миграция схемы	145
SQL	146
Индексация	148
Транзакции	151
6.2. Нереляционная модель	152
Документные хранилища.....	152
Хранилища «ключ — значение»	154
Графовые базы данных	155
Большие данные	156
SQL против NoSQL.....	157
6.3. Распределенная модель	158
Репликация с одним ведущим.....	159
Репликация с многочисленными ведущими	159
Фрагментирование	160
Непротиворечивость данных	162
6.4. Географическая модель.....	163
6.5. Форматы сериализации	165
Подведем итоги	166
Полезные материалы	166
Глава 7. Компьютеры	167
7.1. Архитектура.....	168
Память	168
Процессор	171
7.2. Компиляторы.....	177
Операционные системы.....	181
Оптимизация при компиляции.....	182
Языки сценариев	183
Дизассемблирование и обратный инженерный анализ	184
Программное обеспечение с открытым исходным кодом	185
7.3. Иерархия памяти	186
Разрыв между памятью и процессором.....	187
Временная и пространственная локальность	188

Кэш L1.....	189
Кэш L2.....	189
Первичная память против вторичной	191
Внешняя и третичная память	193
Тенденции в технологии памяти.....	194
Подведем итоги	195
Полезные материалы	196
Глава 8. Программирование	197
8.1. Лингвистика	198
Значения.....	198
Выражения.....	198
Инструкции	200
8.2. Переменные	201
Типизация переменных	202
Область видимости переменных.....	202
8.3. Парадигмы	204
Императивное программирование	204
Декларативное программирование.....	207
Логическое программирование.....	213
Подведем итоги	214
Полезные материалы	214
Заключение.....	215
Приложения	217
I. Системы счисления.....	217
II. Метод Гаусса	219
III. Множества	220
IV. Алгоритм Кэдайна	222

Поиск в графах

Как найти узел в графе? Если структура графа не предоставляет никакой помощи в навигации, вам придется посетить каждую вершину, пока не обнаружится нужная. Есть два способа сделать это: выполнить обход графа в глубину и в ширину (рис. 5.2).

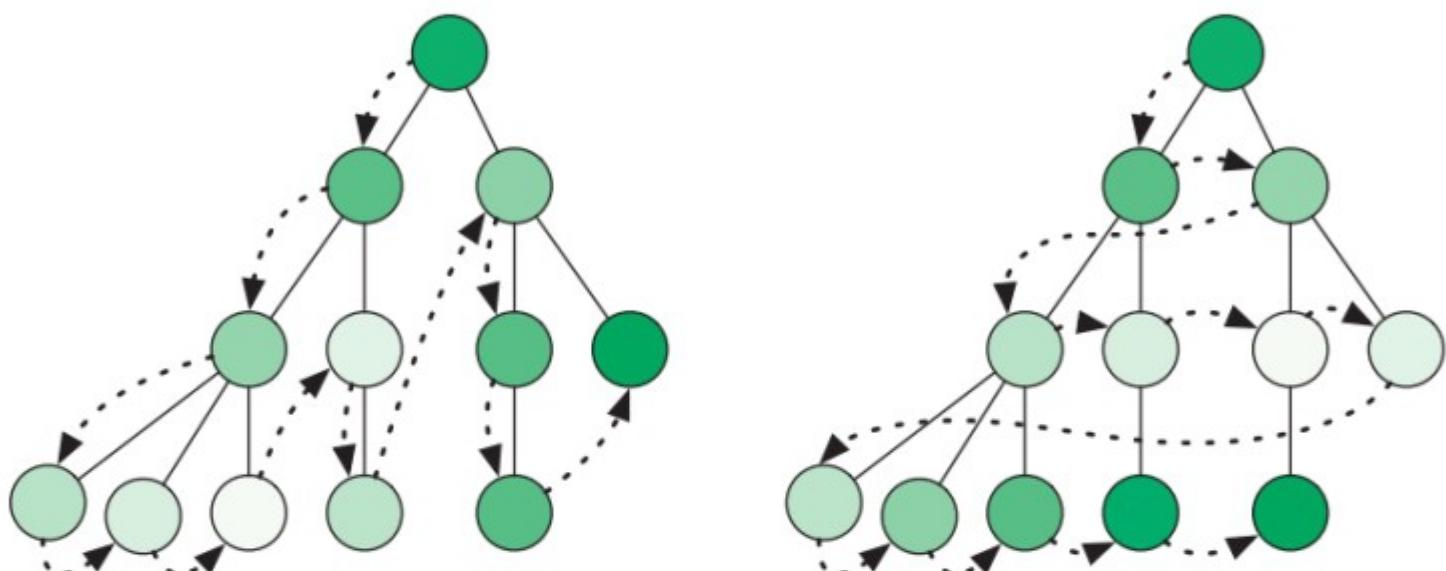


Рис. 5.2. Обход графа в глубину против обхода в ширину

Выполняя поиск в графе в глубину (DFS, depth first search), мы продвигаемся вдоль ребер, уходя все глубже и глубже в граф. Достигнув вершины без ребер, ведущих к каким-либо новым вершинам, мы возвращаемся к предыдущей и продолжаем процесс. Мы используем стек, чтобы запомнить путь обхода графа, помещая туда вершину на время ее исследования и удаляя ее, когда нужно вернуться. Стратегия поиска с возвратом (см. соответствующий раздел главы 3) выполняет обход решений точно так же.

```
function DFS(start_node, key)
    next_nodes ← Stack.new()
    seen_nodes ← Set.new()

    next_nodes.push(start_node)
    seen_nodes.add(start_node)
```

```
while not next_nodes.empty
    node ← next_nodes.pop()
    if node.key = key
        return node
    for n in node.connected_nodes
        if not n in seen_nodes
            next_nodes.push(n)
            seen_nodes.add(n)
return NULL
```

Если обход графа вглубь не кажется приемлемым решением, можно попробовать обход в ширину (BFS, breadth first search). В этом случае обход графа выполняется по уровням: сначала соседей начальной вершины, затем соседей его соседей и т. д. Вершины для посещения запоминаются в очереди. Исследуя вершину, мы ставим в очередь ее дочерние вершины, затем определяем следующую исследуемую вершину, извлекая ее из очереди.

```
function BFS(start_node, key)
    next_nodes ← Queue.new()
    seen_nodes ← Set.new()

    next_nodes.enqueue(start_node)
    seen_nodes.add(start_node)

    while not next_nodes.empty
        node ← next_nodes.dequeue()
        if node.key = key
            return node
        for n in node.connected_nodes
            if not n in seen_nodes
                next_nodes.enqueue(n)
                seen_nodes.add(n)
    return NULL
```

Обратите внимание, что алгоритмы DFS и BFS отличаются только способом хранения следующих исследуемых вершин: в одном случае это очередь, в другом — стек.

Итак, какой подход нам следует использовать? Алгоритм DFS более прост в реализации и использует меньше памяти: достаточно хра-

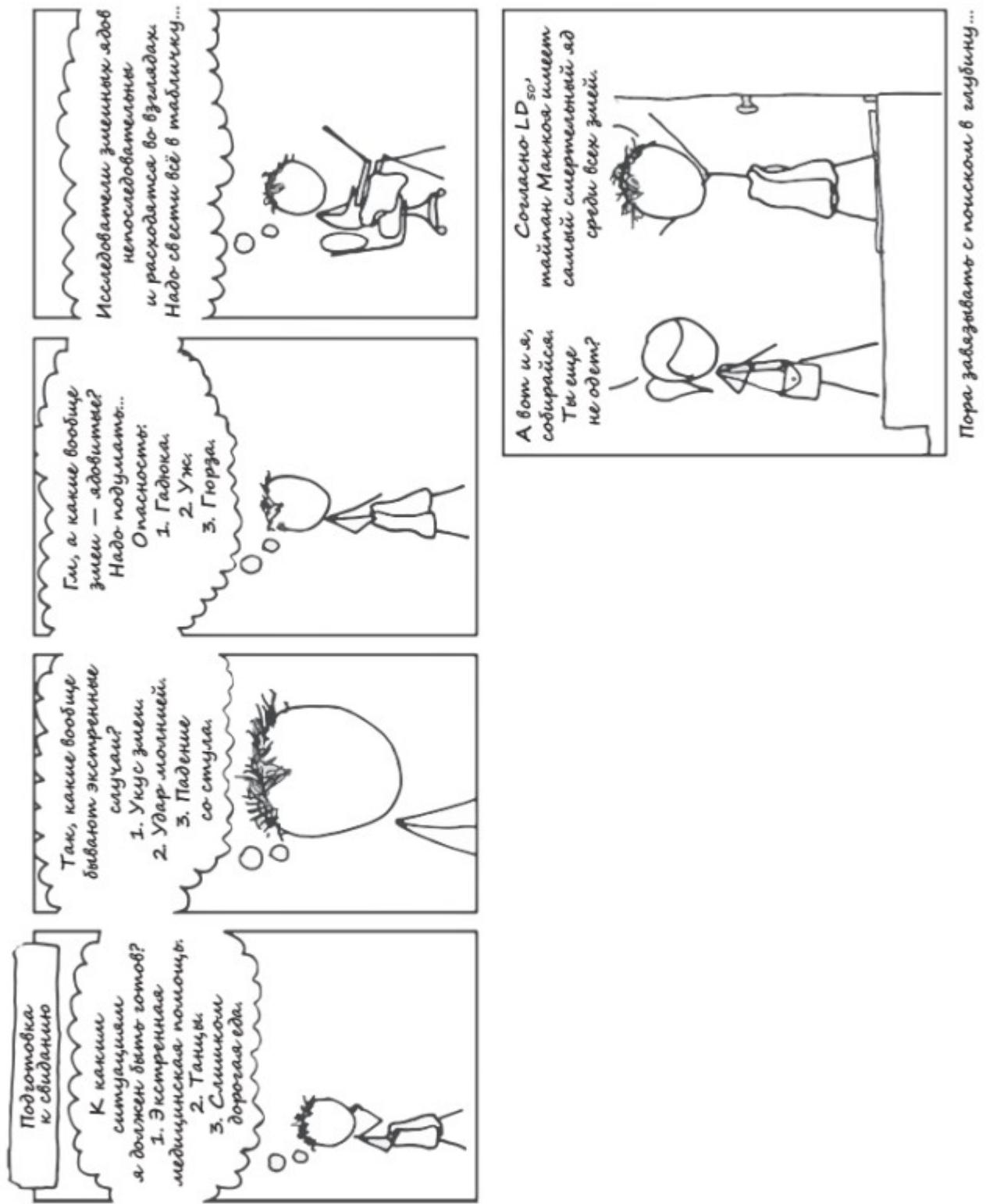


Рис. 5.3. Поиск в графе в глубину¹

¹ Любезно предоставлено <http://xkcd.com>.

нить родительские вершины, ведущие к текущей исследуемой вершине. В BFS придется хранить всю границу процесса поиска. Если граф состоит из миллиона вершин, это может оказаться непрактичным.

Когда есть основания предполагать, что искомая вершина не находится многими уровнями ниже начальной, обычно имеет смысл заплатить более высокую стоимость BFS, потому что так вы, скорее всего, закончите поиск быстрее. Если нужно исследовать абсолютно все вершины графа, лучше придерживаться алгоритма DFS из-за его простой реализации и меньшего объема потребляемой памяти.

Рис. 5.3 показывает, что выбор неправильного метода обхода может иметь страшные последствия.

Раскраска графов

Задачи раскраски графов возникают, когда есть фиксированное число «красок» (либо любой другой набор меток) и вы должны назначить «цвет» каждой вершине в графе. Вершины, которые соединены ребром, не могут иметь одинаковый «цвет». В качестве примера давайте рассмотрим следующую задачу.

Помехи  Дано карта вышек сотовой связи и районов обслуживания. Вышки в смежных районах должны работать на разных частотах для предотвращения помех. Имеется четыре частоты на выбор. Какую частоту вы назначите каждой вышке?

Первый шаг состоит в моделировании задачи при помощи графа. Вышки являются вершинами в графе. Если две из них расположены настолько близко, что вызывают помехи, соединяем их ребром. Каждая частота имеет свой цвет.

Как назначить частоты приемлемым способом? Можно ли найти решение, которое использует всего три цвета? Или два? Определение

минимально возможного количества цветов на самом деле является NP-полной задачей — для этого подходят только экспоненциальные алгоритмы.

Мы не покажем алгоритм для решения данной задачи. Используйте то, чему вы научились к настоящему моменту, и попробуйте решить задачу самостоятельно. Это можно сделать на сайте UVA¹ с онлайн-экспертом, который протестирует предложенное вами решение, выполнит ваш программный код и сообщит, работоспособен ли он. Если с кодом окажется все в порядке, эксперт также оценит время выполнения вашего кода в сравнении с тем, что написали другие люди. Дерзайте! Продумайте алгоритмы и стратегии решения данной задачи и испытайте их. Чтение книги может лишь подвести вас к решению. Взаимодействие с онлайновым экспертом даст вам практический опыт, необходимый для того, чтобы стать отличным программистом.

Поиск путей в графе

Поиск кратчайшего пути между узлами является самой известной графовой задачей. Системы навигации GPS проводят поиск в графе улиц и перекрестков для вычисления маршрута. Некоторые из них даже используют данные дорожного движения с целью увеличения веса ребер, представляющих улицы, где образовался затор.

Для поиска кратчайшего пути вполне можно использовать стратегии BFS и DFS, но это плохая идея. Одним из хорошо известных и очень эффективных способов поиска кратчайшего пути является *алгоритм Дейкстры*. В отличие от BFS, для запоминания просматриваемых вершин алгоритм Дейкстры использует очередь с приоритетом. Когда исследуются новые вершины, их связи добавляются в эту очередь. Приоритетом вершины является вес ребер, которые приводят ее в стартовую вершину. Благодаря этому следующая ис-

¹ Задача раскраски графа на сайте онлайновых экспертов UVA: <https://code.energy/uva-graph-coloring>.