Лекция 1.

Информационно-графовая модель данных.
Понятие задачи информационного поиска.
Понятие информационного графа. Примеры.

1 Информационно-графовая модель данных

База данных (БД) — формализованное представление информации, удобное для хранения и поиска данных в нем.

Одно из основных направлений теории БД связано с вопросами сложности алгоритмов обработки данных.

Это направление связано с проблематикой физической организации баз данных. При физической организации баз данных мы имеем дело не с представлением данных в прикладных программах, а с их размещением на запоминающих устройствах.

Критерии, определяющие выбор физической организации, отличаются от тех, которые определяют выбор логической организации данных. При выборе физической организации решающим фактором является эффективность, причем согласно Дж. Мартину на первом месте стоит обеспечение эффективности поиска, далее идут эффективность операций занесения и удаления и затем обеспечение компактности данных.

На первом шаге нам необходимо ввести понятие задачи информационного поиска.

В понятие задачи поиска исследователи вкладывают по крайней мере 3 различных смысла.

Специалисты в теории исследования операций понимают задачи поиска как задачи управления сближением одной системы (поисковой) с другой (искомым объектом) по неполной априорной информации. Понимается, что цель поиска — это обнаружение искомого объекта, определяемое как выполнение определенных терминальных условий. Интенсивно проблемой поиска подвижных объектов начали заниматься в период Второй мировой войны. Интерес к этой проблеме в тот период был вызван

необходимостью разработки тактики борьбы против подводных лодок. Среди работ, посвященных этой тематике, можно выделить, например, работы О. Хеллмана и Д. П. Кима.

Другое понимание задач поиска можно найти в книге Р.Альсведе, И.Вегенера "Задачи поиска". Приведем поясняющий пример из этой книги.

Во время Второй мировой войны все призываемые в армию США подвергались проверке на реакцию Вассермана. При этом проверялось, есть ли в крови обследуемого определенные антитела, имеющиеся только у больных. Во время этого массового обследования было замечено, что разумнее анализировать пробу крови целой группы людей. Если такая объединенная проба крови не содержит антител, то, значит, ни один из обследуемых не болен. В противном случае среди них есть хотя бы один больной. Хороший алгоритм поиска для этой задачи — это тот, который для типичной выборки людей позволяет "наискорейшим образом" выявлять множество всех больных.

Другой пример мы находим в статье Д.Ли, Ф.Препараты "Вычислительная геометрия". Дано множество горизонтальных и вертикальных отрезков. Надо найти все точки пересечения отрезков.

Эти два примера объединяет то, что в обоих случаях поиск производится однократно. Это порождает свои особенности таких задач поиска. Как правило, данные в этих задачах не имеют сложной организации. Фиксация множества, в котором производится поиск, однозначно определяет множество найденных объектов. "Хорошесть" алгоритма поиска определяется при варьировании множества, в котором производится поиск.

В третьем понимании задачи поиска предполагается многократное обращение к одним и тем же данным, но возможно каждый раз с разными требованиями к искомым объектам, то есть с разными запросами на поиск. Такие задачи поиска обычно возникают в системах, использующих базы данных. Многократное использование порождает особую проблему — проблему специальной организации данных, направленной на последующее ускорение поиска. Процесс такой специальной организации данных, проводимый до того, как осуществляется поиск, называется предобработкой и часто может занимать очень большое время, которое затем окупается сторицей в результате многократности поиска. Простейшим примером предобработки является сортировка. Построение "хорошего" алгоритма поиска в этом случае сводится к нахождению хо-

роших структур данных, то есть к осуществлению такой хорошей предобработки данных, которая обеспечила бы хорошую скорость поиска. "Хорошесть" алгоритма поиска в этом случае определяется варьированием запроса на поиск, например, как среднее время поиска на запросе.

В зависимости от того, является ли база данных фиксированной или изменяется на протяжении времени работы с ней, мы имеем два типа организации баз данных: *статическую* и *динамическую* соответственно.

Задачи поиска, возникающие в статических базах данных и предполагающие многократное обращение к одним и тем же данным, и являются объектом исследования в данной работе.

Приведем примеры таких задач поиска.

Простейшим и самым распространенным примером задачи поиска, встречающейся в любой базе данных, является задача поиска по ключу. Суть ее состоит в том, что любой объект в базе данных имеет свой уникальный ключ. Это может быть порядковый номер, уникальное имя, или уникальное значение некоторого поля, например, номер паспорта. Задача состоит в том, чтобы по заданному в запросе ключу найти в базе данных объект с этим ключом (если такой объект в базе имеется). Более формально эту задачу можно поставить следующим образом. Имеется некоторое конечное множество ключей. Имеется некоторое более широкое множество запросов. Требуется по произвольному запросу из множества запросов найти в множестве ключей ключ идентичный (равный) ключу-запросу. В такой постановке эта задача называется задачей поиска идентичных объектов.

Другой пример взят из систем машинной графики, обработки изображений и систем автоматизированного проектирования Дано конечное множество точек из отрезка $[0, 1]$ вещественной прямой. Множество запросов есть множество всех отрезков, содержащихся в $[0, 1]$. Надо для произвольного запроса $[u, v]$ из множества запросов перечислить все точки из нашего множества точек, которые попадают в отрезок $[u, v]$. Эта задача носит название одномерной задачи интервального поиска.

Если мы хотим всерьез изучать сложность алгоритмов поиска, то без введения математической модели объекта изучения нам не обойтись.

Поэтому начнем с формализации понятия задачи информационного поиска (ЗИП). В работах Ахо А., Хопкрофт Дж., Ульман Дж. "Построение и анализ вычислительных алгоритмов", Решетников В. Н. "Алгебраическая теория информационного поиска", Селтон Г. "Автоматическая обработка, хранение и поиск информации" и Bentley J. L., Saxe

J. B. "Decomposable searching problems. I. Static-to-dynamic transformation" вводились различные формализации ЗИП. Так в последней работе вводилась формализация наиболее близкая к той, которую мы будем использовать в данной работе. Бентли и Сакс вводят формализацию ЗИП следующим образом: вопрос к базе данных представляет некоторый запрос, имеющий тип $T1$, сама база данных состоит из элементов типа $T2$, а ответ на вопрос — значение типа $T3$, например, $T3$ может иметь логический тип, если предполагается, что ответ на запрос должен быть "да" или "нет", $T3$ может совпадать с $T2$, если ответом на запрос является элемент базы данных, и наконец $T3$ может быть множеством элементов типа $T2$, если в ответ на запрос надо перечислять некоторые элементы из базы данных. Вопрос Q рассматривается как отображение из $T1$ и множества подмножеств $T2$ в $T3$, то есть $Q: \ T1 \times 2^{T2} \to T3$.

Мы будем рассматривать только такие задачи поиска, в которых в ответ на запрос надо перечислить элементы базы данных, удовлетворяющие запросу. ЗИП такого типа называются задачами на перечисление. С учетом этого факта мы и дадим собственную формализацию ЗИП, более удобную для использования в дальнейшем.

Из приведенных примеров видно, что в задачах поиска имеется некий универсум, из которого берутся объекты поиска (элементы базы данных). В первом примере таким универсумом является множество всевозможных ключей, а во втором — отрезок $[0, 1]$ вещественной прямой. Этот универсум обозначим через Y и будем называть множеством объектов поиска или *множеством записей*, а элементы множества Y, будем называть, *записями*.

Далее в задачах поиска всегда имеется *множество запросов*. В первом примере множество запросов совпадает с множеством объектов поиска и является множеством всевозможных ключей, а во втором примере множество запросов есть множество всех отрезков, содержащихся в $[0, 1]$, то есть множество $\{(u, v) : 0 \leq u \leq v \leq 1\}$. Множество запросов будем обозначать через X.

На декартовом произведении $X \times Y$ имеется бинарное отношение, которое позволяет устанавливать, когда запись из Y удовлетворяет запросу из X. Это отношение будем называть *отношением поиска*. В первом примере отношение поиска есть отношение идентичности (равенства), то есть запись удовлетворяет запросу, если они идентичны. Во втором примере отношение поиска, которое обозначим через ρ_{int}, определяется

соотношением

$$(u, v)\rho_{int}y \Longleftrightarrow u \leq y \leq v, \tag{1}$$

где $(u, v) \in X,\ y \in Y$.

Тройку $S = \langle X, Y, \rho\rangle$, где X — множество запросов, Y — множество записей, ρ — отношение поиска, заданное на $X \times Y$, будем называть *типом*, или иногда более развернуто *типом задач информационного поиска*.

Тройку $I = \langle X, V, \rho\rangle$, где X — множество запросов; V — некоторое конечное подмножество множества Y, в дальнейшем называемое *библиотекой*; ρ — отношение поиска, заданное на $X \times Y$, будем называть *задачей информационного поиска* (ЗИП) типа $S = \langle X, Y, \rho\rangle$. Содержательно будем считать, что задача $I = \langle X, V, \rho\rangle$ состоит в перечислении для произвольно взятого запроса $x \in X$ всех тех и только тех записей из V , которые находятся в отношении ρ с запросом x, то есть удовлетворяют запросу x.

Тем самым мы формализовали понятие ЗИП.

Для полной определенности отметим, что поскольку библиотека V есть множество, то в ней все элементы различные, то есть в ней нет повторяющихся элементов.

Следующий шаг, который мы обязаны сделать — это выбрать математическую модель для алгоритмов поиска.

Отметим, что здесь и всюду далее, используя термин "алгоритм", мы часто будем подменять им понятие "условного алгоритма" (или "относительного алгоритма", см. книгу Мальцева А. И. "Алгоритмы и рекурсивные функции", с. 44–45), то есть будем рассматривать алгоритмы, которые выполняются при условии, что мы умеем выполнять некоторые операции из описанного заранее множества. В качестве таких операций могут выступать, например, некоторые операции над вещественными числами, но так или иначе мы всегда будем явно оговаривать операции, относительно которых рассматривается каждый описываемый условный алгоритм.

Переберем возможных кандидатов на роль математической модели для алгоритмов поиска.

1) Алгоритмы поиска информации можно описывать на языке программирования, например на Си++. Положительным фактором такого подхода является то, что любое описание алгоритма одновременно будет и его реализацией. Отрицательным — то, что помимо описания алгорит-

ма нам понадобится по алгоритму определять его сложность, а в этом случае это сделать весьма трудно, более того неясно, как сделать этот переход формально. Как следствие, мы не можем использовать этот язык для получения нижних оценок сложности алгоритмов.

2) Алгоритмы поиска можно описывать с помощью машин Тьюринга. Если под сложностью алгоритма понимать время поиска, этот подход обладает тем же недостатком, что и первый, но плюс к этому добавляется трудность программирования для машин Тьюринга.

3) Для описания алгоритмов поиска можно использовать схемы алгоритмов Янова или стандартные схемы программ Котова. Неудобство использования этих схем определяется тем, что они предназначены для других целей, а именно для исследования проблемы эквивалентности алгоритмов. Поэтому если мы даже сможем адаптировать эти понятия для наших нужд, то пропадает главное преимущество, которое дает использование известных моделей — это наработанный аппарат.

4) В качестве модели вычислений можно использовать машину с произвольным доступом к памяти, аналогичную описанной в книге Ахо А., Хопкрофт Дж., Ульман Дж. "Построение и анализ вычислительных алгоритмов" с добавлением возможности выполнения арифметических операций над действительными числами. Это значит, что в такой машине каждая ячейка памяти может содержать действительное число, а каждая арифметическая операция, такая, как сложение, умножение и деление, может быть выполнена за единицу времени. В зависимости от решаемой задачи машина может иметь некоторые другие примитивные операции, о которых предполагается, что они выполняются за постоянное время.

Известной моделью, используемой для исследования сложности алгоритмов, является алгебраическое дерево вычислений (см., например, Ben-Or M. "Lower bounds for algebraic computation trees").

Пусть $\mathbf{R}$ — множество действительных чисел. *Алгебраическое дерево вычислений* (АДВ) на множестве переменных $W = \{x_1, x_2, \ldots, x_n\}$, где $x_i \in \mathbf{R}$, — бинарное дерево D, размеченное следующим образом.

1. Каждой вершине v, имеющей в точности одного сына (простая вершина), приписывается операция вида $f_v := f_{v_1} \# f_{v_2}$, или $f_v := f_{v_1} \# c$, или $f_v := \sqrt{f_{v_1}}$, где v_i $(i = 1, 2)$ — предок вершины v в дереве D, или $f_{v_i} \in W$, $\# \in \{+, -, \times, /\}$, а $c \in \mathbf{R}$ является константой.

2. Каждой вершине v, имеющей в точности двух сыновей (вершина ветвления), приписывается операция сравнения вида $f_{v_1} > 0$, или $f_{v_1} < 0$, или $f_{v_1} = 0$, где v_1 — предок вершины v в дереве D, или $f_{v_1} \in W$.

3. Каждому листу дерева приписывается одно из значений ДА или НЕТ.

Приведем пример использования АДВ, например, для задачи поиска идентичных объектов $I = \langle X, V, = \rangle$ в случае, когда множества запросов $X = [0,1] \subset \mathbf{R}$ и $V \subset [0,1]$. Для каждого заданного запроса $x \in X$ программа прокладывает в дереве D путь $P(x)$, начинающийся в корне. При прохождении каждой простой вершины выполняется арифметическая операция, приписанная этой вершине, а в каждой вершине ветвления происходит ветвление в соответствии с результатом сравнения, приписанного вершине. При достижении листа дерева возвращается ответ ДА или НЕТ. Считается, что АДВ D решает задачу I, если возвращаемый ответ верен для любого запроса $x \in X$. Сложность дерева D, обозначаемая $C(D)$, определяется как максимум величины $cost(x, D)$ по всем значениям x, где $cost(x, D)$ — число вершин, проходимых путем $P(x)$ в дереве D. Сложность задачи I, обозначаемая $C(I)$, есть минимум $C(D)$ по всем алгебраическим деревьям вычислений D, решающих задачу I.

Разновидностью алгебраического дерева вычислений является так называемое (алгебраическое) *дерево решений порядка d* . Дерево решений порядка d — это дерево, каждой вершине которого соответствует сравнение вида $f(x)?0$, где f имеет полиномиальную сложность относительно входных данных с показателем степени не более d и $? \in \{>, <, =\}$. В случае, когда d равно 1, получается *линейное дерево решений*, с использованием которого получены доказательства ряда нижних оценок сложности. Серьезные исследования временной сложности алгебраических деревьев решений проводились М. Ю. Мошковым.

В чем заключаются недостатки использования алгебраического дерева вычислений Бен-Ора или алгебраического дерева решений порядка d для моделирования алгоритмов поиска? Во-первых, хотелось бы более полно учитывать специфику алгоритмов поиска. Во-вторых, использование алгебраического дерева вычислений Бен-Ора или алгебраического дерева решений порядка d приводит к некоторым заблуждениям. Так в рамках АДВ (так же как и в рамках линейного дерева решений) за-

дача поиска идентичных объектов $I = \langle X, V, = \rangle$ в случае, когда множества запросов $X = [0,1] \subset \mathbf{R}$ и $V \subset [0,1]$, имеет нижнюю оценку сложности $\underline{O}(\log_2 n)$, где n — число точек в библиотеке V. Это так называемая *теоретико-информационная оценка*. Понятно откуда получается эта оценка — бинарное дерево с n листьями должно иметь высоту как минимум $\log_2 n$. Сила влияния этой оценки настолько велика и заблуждение о неизбежности такого времени для решения настолько глубоко, что часто алгоритмы с оценкой $\underline{O}(\log_2 n)$ называют оптимальными. Тогда как эта оценка всего лишь следствие бинарности дерева решений, и если отказаться от бинарности, то не будет и оценки. С помощью этой оценки получается нижняя оценка сложности порядка $\underline{O}(n \log_2 n)$ для задачи сортировки множества из n элементов. На основе оценки сложности сортировки с помощью метода сведения одной задачи к другой можно показать, что большое количество задач требуют для своего решения время $\underline{O}(n \log_2 n)$ в рамках АДВ-модели вычислений в том числе задача построения выпуклой оболочки n точек на плоскости, задача построения евклидова минимального остовного дерева и т. д.

Но если отказаться от АДВ-модели вычислений, то для нашей задачи поиска идентичных объектов $I = \langle X, V, = \rangle$ легко предложить алгоритм, который решает эту задачу за константное время, при условии, что можно осуществлять предобработку и использовать дополнительную память. Пусть $V = \{y_1, \ldots, y_n\}$ — упорядоченное по возрастанию множество точек. Найдем расстояние между двумя ближайшими точками. Пусть оно равно d. Поделим отрезок $[0,1]$ на $m =]1/d[$ равных частей, так что i-ая часть ($i = \overline{0, m-1}$) имеет вид $(i/m, (i+1)/m]$. Тогда в каждую часть попадет не более одной точки из V. Заведем целочисленный массив A длины m, i-ый элемент которого ($i = \overline{0, m-1}$) содержит 0, если в i-ой части нет точек из V, и номер j, если в i-ую часть попала точка y_j из V. После такой предобработки поиск по произвольному запросу $x \in [0,1]$ будем осуществлять следующим образом. Вычислим $j = [x \cdot m]$. Номер j дает номер части, в которую попала точка x. Если $A_j = 0$, то точки x в библиотеке V нет, иначе сравниваем y_{A_j} с x и если они равны, то точка x найдена.

Мы откажемся от использования АДВ-модели, в частности, чтобы не быть зажатыми теоретико-информационной оценкой.

5) Прежде чем предложить еще одну модель алгоритмов поиска взглянем на алгоритм, решающий задачу поиска с функциональной точки зрения.

Рассмотрим произвольную ЗИП $I = \langle X, V, \rho \rangle$. Алгоритм поиска решает ЗИП, если на любой запрос из множества запросов X он выдает все те и только те записи из V, которые удовлетворяют запросу. Возьмем произвольную запись $y \in Y$. Для нее можно ввести *характеристическую функцию*

$$\chi_{y,\rho}(x) = \begin{cases} 1, & \text{если } x\rho y \\ 0 & \text{в противном случае} \end{cases},$$

то есть она равна 1 на тех запросах, которым удовлетворяет запись y. Тогда можно сказать, что алгоритм, решающий ЗИП $I = \langle X, V, \rho \rangle$, где $V = \{y_1, \ldots, y_k\}$, — ни что иное как алгоритм, реализующий систему функций $\{\chi_{y_1,\rho}, \ldots, \chi_{y_k,\rho}\}$. Следовательно, управляющая система, моделирующая алгоритм, решающий ЗИП $I = \langle X, V, \rho \rangle$, должна представлять собой многополюсник, реализующий множество характеристических функций $\{\chi_{y_1,\rho}, \ldots, \chi_{y_k,\rho}\}$.

Рассмотрим случай когда множество запросов $X = B^n$ — n-мерный единичный куб. Тогда каждая из функций $\chi_{y_i,\rho}$ будет функцией алгебры логики и, значит, в классе контактных схем можно построить многополюсник, реализующий множество функций $\{\chi_{y_1,\rho}, \ldots, \chi_{y_k,\rho}\}$ как функций проводимости.

Если множество запросов не является n-мерным единичным кубом, то можно вместо множества $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ использовать для нагрузки ребер некоторое множество F предикатов, определенных на множестве запросов X. Тогда при удачном выборе множества F и правильной нагрузке ребер многополюсника предикатами из F мы можем в качестве функций проводимости между полюсами получить характеристические функции $\chi_{y_i,\rho}$ $(i = \overline{1,k})$. Но если вводить сложность полученной многополюсной сети так же как в контактных схемах (то есть как количество ребер сети), то эта сложность скорее будет характеризовать объем памяти, соответствующего алгоритма, но не время поиска. Чтобы сложность сети характеризовала время поиска, ее надо вводить на подобии того, как это делалось в АДВ-модели. Для этого будем считать, что сеть у нас ориентированная, что в сети есть такой полюс, который называется *корнем*, и каждая из характеристических функций записей $\chi_{y_i,\rho}$ $(i = \overline{1,k})$ реализуется как функция проводимости между корнем и своим полюсом, который отметим приписыванием ему записи y_i. Теперь если взять произвольный запрос $x \in X$, то алгоритм функционирования сети на запросе x можно описать аналогично алгоритму разметки гра-

фа: считаем, что в начальный момент все вершины сети, кроме корня, неотмеченные, а некоторое упорядоченное множество вершин сети, которое назовем *множеством активных вершин*, содержит только корень сети. На каждом очередном шаге делаем следующее. Если множество активных вершин не пусто, то выбираем первую активную вершину и удаляем ее из множества активных вершин. Если выбранная вершина является полюсом, то соответствующую ей запись включаем в ответ на запрос x. Просматриваем в некотором порядке все ребра, исходящие из выбранной вершины, и если предикат, приписанный просматриваемому ребру, на запросе x принимает значение 1 и конец просматриваемого ребра неотмеченная вершина, то отмечаем конец просматриваемого ребра и включаем его в множество активных вершин (если мы включим его в начало множества активных вершин, то получим алгоритм обхода "сначала вглубь", а если в конец — то "сначала вширь"). Алгоритм завершает работу в тот момент, когда множество активных вершин окажется пустым.

Легко видеть, что если между корнем сети и некоторой вершиной на запросе x есть проводимость, то есть из корня в эту вершину ведет некоторая цепочка ребер, каждому из которых приписан предикат, принимающий значение 1 на запросе x, то обязательно в результате описанного алгоритма обхода сети данная вершина окажется отмеченной, то есть алгоритм обхода посетит ее.

Описанный алгоритм обхода сети можно считать соответствующим сети алгоритмом поиска на запросе x, а сложность сети на запросе x можно считать равной, как и в АДВ-модели, числу просмотренных ребер, или, что то же самое числу вычисленных алгоритмом обхода предикатов. Такая сложность будет характеризовать время работы соответствующего алгоритма поиска на запросе x.

Описанную сеть с описанным функционированием будем называть *предикатным информационным графом.*

Предикатный информационный граф можно рассматривать как обобщение контактных схем и АДВ-модели, причем от контактных схем заимствована структура в виде сети с нагрузкой ребер функциями и использование функций проводимости, а от АДВ-модели — способ введения сложности сети. Кроме того вводится алгоритм обхода сети, похожий на алгоритм разметки графа, который можно рассматривать как соответствующий сети алгоритм поиска. Причем в каждой вершине сети, до которой на некотором запросе x дошел алгоритм поиска, каждое из ре-

бер, исходящее из этой вершины, задает возможное направление поиска, и если на этом запросе x значение предиката, соответствующего ребру, равно 1, то в направлении данного ребра поиск продолжается, а если равно 0, то по этому направлению поиск прекращается. Таким образом, из одной вершины сети может возникнуть сразу несколько направлений, по которым поиск продолжается. Но в задачах поиска часто встречается ситуация, когда из нескольких направлений поиска выбирается точно одно, например, так как это происходит в вершине ветвления алгебраического дерева вычислений, когда в зависимости от результата сравнения выбирается одно из двух направлений движения. В таких случаях гораздо удобнее приписать вершине функцию-переключатель и в зависимости от ее значения на запросе выбрать то направление, на которое указывает значение переключателя. Поэтому слегка усложним понятие предикатного информационного графа. Для этого введем множество G переключателей, то есть функций, определенных на множестве запросов X и принимающих значения из конечного подмножества натурального ряда. Теперь если дана ненагруженная многополюсная ориентированная сеть, то нагрузку сети функциями будем осуществлять следующим образом. Сначала выберем в сети некоторое количество вершин, которые назовем *точками переключения*, и каждой точке переключения припишем некоторый переключатель из G и затем занумеруем подряд, начиная с 1, ребра, исходящие из нее. Такие ребра, исходящие из точек переключения, назовем *переключательными*. Остальные ребра назовем *предикатными* и нагрузим их предикатами из множества F. Нагрузим все полюса сети, кроме корня, записями, причем можем считать, что одна и та же запись может быть приписана нескольким полюсам. Алгоритм обхода сети на запросе в этом случае полностью аналогичен описанному ранее за тем исключением, что если активная выбранная вершина является точкой переключения, то вычисляем значение переключателя, соответствующего вершине, на запросе x, и если среди ребер, исходящих из этой вершины, есть ребро с номером, равным вычисленному значению, то включаем конец этого ребра в множество активных вершин, если только этот конец не был отмеченной вершиной. Полученную нагруженную сеть с описанным функционированием будем называть *информационным графом* (ИГ).

Прежде чем дать строгое формальное определение ИГ, введем некоторые вспомогательные понятия и обозначения.

Пусть M — некоторое конечное множество. Через $|M|$ обозначим число элементов во множестве M, называемое *мощностью множества* M.

Через $\{\overline{1,m}\}$ договоримся обозначать множество $\{1,2,\ldots,m\}$.

Некоторые оценки мы будем приводить с точностью до главного члена, поэтому введем обозначения, обычно принятые при описании асимптотических оценок.

Будем писать $\alpha(n)=\bar{o}(1)$, если $\lim\limits_{n\to\infty}\alpha(n)=0$; $A(n)=\bar{o}(B(n))$, если $A(n)=B(n)\cdot\bar{o}(1)$. Скажем, что $A(n)$ *асимптотически не превосходит* $B(n)$ при $n\to\infty$ и обозначим $A\lesssim B$, если существует $\alpha(n)=\bar{o}(1)$ такое, что начиная с некоторого номера n_0, $A(n)\le(1+\alpha(n))\cdot B(n)$. Если $A\lesssim B$ и $B\lesssim A$, то будем говорить, что A и B *асимптотически равны* при $n\to\infty$ и обозначать $A\sim B$. Будем писать $A\precsim B$, если существует такая положительная константа c, что, начиная с некоторого номера n_0, $A(n)\le c\cdot B(n)$. Если $A\precsim B$ и $B\precsim A$, то будем говорить, что A и B *равны по порядку* при $n\to\infty$ и обозначать $A\asymp B$ или $A=\underline{O}(B)$.

Через $\binom{n}{k}$ будем обозначать *число сочетаний из n элементов по k*. Если r — действительное число, то через $[r]$ будем обозначать максимальное целое, не превышающее r, а через $]r[$ — минимальное целое, не меньшее, чем r. Значок $+$ будем понимать как "по определению равно". Математическое ожидание будем обозначать значком $\mathbf{M}$, а значок $\mathbf{M}_x$ будем понимать как среднее значение при вариации переменной x.

Договоримся также о теоретико-графовой терминологии.

Пусть нам дан ориентированный граф. В ориентированном ребре (α,β) вершину α будем называть *началом ребра*, а β — *концом*. Скажем, что ориентированное ребро графа *исходит из вершины* β (*входит в вершину* β), если β — начало (конец) данного ребра. Скажем, что ребро *инцидентно* вершине, если эта вершина является одним из концов данного ребра. *Полустепенью исхода* (*захода*) вершины графа назовем число ребер, исходящих из данной вершины (входящих в данную вершину). *Степенью инцидентности вершины* назовем число инцидентных ей ребер. Вершину графа назовем *концевой*, если полустепень ее исхода равна 0. Остальные вершины графа назовем *внутренними*.

Последовательность ориентированных ребер графа

$$(\alpha_1,\alpha_2),(\alpha_2,\alpha_3),\ldots,(\alpha_{m-1},\alpha_m)$$

назовем *ориентированной цепью* от вершины α_1 к вершине α_m.

Если f — одноместный предикат, определенный на X , то есть $f:X\to\{0,1\}$, то множество $N_f=\{x\in X:f(x)=1\}$ назовем *характеристическим множеством предиката* f.

Множество $O(y,\rho) = \{x \in X : x\rho y\}$ назовем *тенью записи* $y \in Y$.

Функцию $\chi_{y,\rho} : X \to \{0,1\}$ такую, что $N_{\chi_{y,\rho}} = O(y,\rho)$, назовем *характеристической функцией записи* y.

В формальном определении понятия ИГ используются 4 множества:

- множество запросов X;
- множество записей Y;
- *множество* F *одноместных предикатов*, заданных на множестве X;
- *множество* G *одноместных переключателей*, заданных на множестве X (*переключатели* — это функции, область значений которых является начальным отрезком натурального ряда).

Пару $\mathcal{F} = \langle F, G\rangle$ будем называть *базовым множеством*.

Определение понятия ИГ разбивается на два шага. На первом шаге раскрывается структурная (схемная) часть этого понятия, на втором — функциональная.

Определение ИГ с точки зрения его структуры.

Пусть нам дана ориентированная многополюсная сеть.

Выделим в ней один полюс и назовем его *корнем*, а остальные полюса назовем *листьями*.

Выделим в сети некоторые вершины и назовем их *точками переключения* (полюса могут быть точками переключения).

Если β — вершина сети, то через ψ_β обозначим *полустепень исхода* вершины β.

Каждой точке переключения β сопоставим некий символ из G. Это соответствие назовем *нагрузкой точек переключения*.

Для каждой точки переключения β ребрам, из нее исходящим, поставим во взаимно однозначное соответствие числа из множества $\{\overline{1,\psi_\beta}\}$. Эти ребра назовем *переключательными*, а это соответствие — *нагрузкой переключательных ребер*.

Ребра, не являющиеся переключательными, назовем *предикатными*.

Каждому предикатному ребру сети сопоставим некоторый символ из множества F. Это соответствие назовем *нагрузкой предикатных ребер*.

Сопоставим каждому листу сети некоторую запись из множества Y. Это соответствие назовем *нагрузкой листьев*.

Полученную нагруженную сеть назовем *информационным графом* над базовым множеством $\mathcal{F} = \langle F, G \rangle$.

Определение функционирования ИГ.

Скажем, что предикатное ребро проводит запрос $x \in X$, если предикат, приписанный этому ребру, принимает значение 1 на запросе x; переключательное ребро, которому приписан номер n, проводит запрос $x \in X$, если переключатель, приписанный началу этого ребра, принимает значение n на запросе x; ориентированная цепочка ребер проводит запрос $x \in X$, если каждое ребро цепочки проводит запрос x; запрос $x \in X$ проходит в вершину β ИГ, если существует ориентированная цепочка, ведущая из корня в вершину β, которая проводит запрос x; запись y, приписанная листу α, попадает в ответ ИГ на запрос $x \in X$, если запрос x проходит в лист α. *Ответом ИГ U на запрос x* назовем множество записей, попавших в ответ ИГ на запрос x, и обозначим его $\mathcal{J}_U(x)$. Эту функцию $\mathcal{J}_U(x) : X \to 2^Y$ будем считать результатом функционирования ИГ U и называть *функцией ответа ИГ U*.

Понятие ИГ полностью определено.

Проиллюстрируем приведенное определение на примере одномерной задачи интервального поиска. В этом случае 4 множества, определяющие ИГ, имеют вид:

- множество запросов $X_{int} = \{(u, v) : 0 \le u \le v \le 1\}$;
- множество записей $Y_{int} = [0, 1]$;
- множество предикатов
  $F = F_1 \bigcup F_2$, где $F_1 = \{f^1_{\le,a} : a \in [0, 1]\}$, $F_2 = \{f^2_{\ge,a} : a \in [0, 1]\}$,

$$f^1_{\le,a}(u, v) = \begin{cases} 1, & \text{если } u \le a \\ 0, & \text{если } u > a \end{cases}, \tag{2}$$

$$f^2_{\ge,a}(u, v) = \begin{cases} 1, & \text{если } v \ge a \\ 0, & \text{если } v < a \end{cases}, \tag{3}$$

- множество переключателей
  $G = G_1 \bigcup G_2 \bigcup G_3$, где $G_1 = \{g_{.,m} : m \in \mathbf{N}\}$, $G_2 = \{g_{-,m} : m \in \mathbf{N}\}$, $G_3 = \{g_{\le,a} : a \in (0, 1]\}$,

$$g_{.,m}(u, v) = \max(1,]u \cdot m[), \tag{4}$$

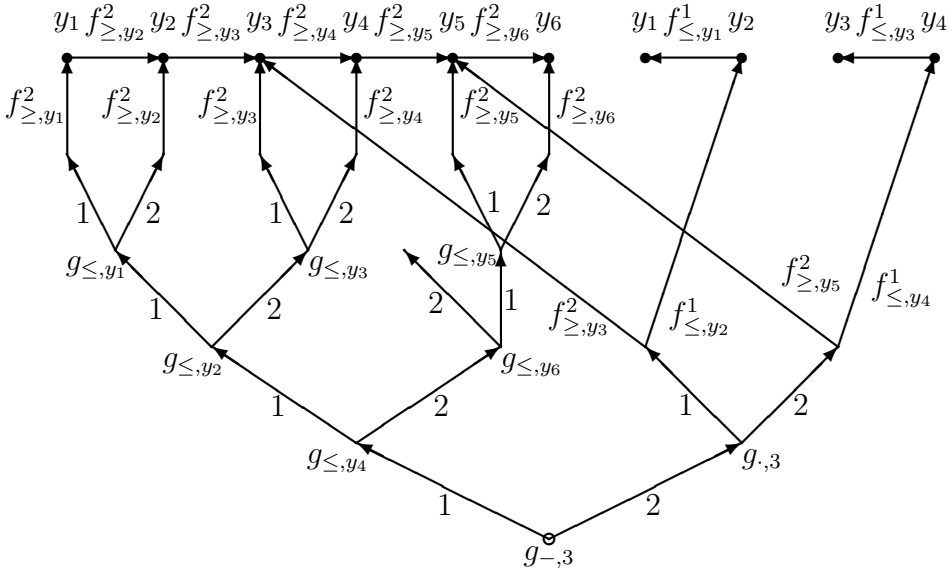

Рис. 1: Решение одномерной задачи интервального поиска

$$g_{-,m}(u,v) = \begin{cases} 1, & \text{если } v-u < 1/m \\ 2, & \text{если } v-u \geq 1/m \end{cases}, \tag{5}$$

$$g_{\leq,a}(u,v) = \begin{cases} 1, & \text{если } u \leq a \\ 2, & \text{если } u > a \end{cases}. \tag{6}$$

В информационном графе, приведенном на рисунке 1, корень изображен полым кружком. Листья изображены жирными точками, а записи, приписанные листьям, — это символы y с индексами. На рисунке имеется 8 переключательных вершин (им приписаны символы g с индексами) и 17 предикатных ребра (им приписаны символы f с индексами).

Если n — натуральное число, а $g(x)$ — некий переключатель, то через $\xi^n_g(x)$ обозначим предикат, определенный на X , такой, что

$$N_{\xi^n_g} = \{x \in X : g(x) = n\}.$$

Обозначим

$$\widehat{G} = \{\xi^n_g : g \in G, n \in \mathbf{N}\},$$

где $\mathbf{N}$ — множество натуральных чисел.

Если c — ребро ИГ, то через $[c]$ обозначим его нагрузку.

В соответствии с приведенными выше определениями введем функции проводимости.

Проводимостью ребра (α, β) назовем предикат, равный $[(\alpha, \beta)]$, если ребро предикатное, и $\xi_g^{[(\alpha,\beta)]}$, если ребро переключательное, где g — переключатель, соответствующий вершине α.

Проводимостью ориентированной цепи назовем конъюнкцию проводимостей ребер цепи.

Если зафиксировать запрос x, то цепь, проводимость которой на запросе x равна 1, назовем *проводящей цепью на запросе* x.

В ИГ по аналогии с контактными схемами введем для каждой пары вершин α и β *функцию проводимости* $f_{\alpha\beta}$ от вершины α к вершине β следующим образом:

- если $\alpha = \beta$, то $f_{\alpha\beta}(x) \equiv 1$ $(x \in X)$;
- если $\alpha \neq \beta$ и в ИГ не существует ориентированных цепей от α к β, то $f_{\alpha\beta}(x) \equiv 0$;
- если $\alpha \neq \beta$ и множество ориентированных цепей от α к β не пусто, то $f_{\alpha\beta}(x)$ равно дизъюнкции проводимостей всех ориентированных цепей от α к β.

Функцию проводимости от корня ИГ к некоторой вершине β ИГ назовем *функцией фильтра вершины* β и обозначим $\varphi_\beta(x)$.

Через $\mathcal{R}(U), \mathcal{P}(U), \mathcal{L}(U)$ (или просто $\mathcal{R}, \mathcal{P}, \mathcal{L}$) обозначим множества вершин, точек переключения и листьев ИГ U соответственно.

Пусть $\mathcal{N}$ — некоторая подсеть (то есть произвольное подмножество вершин и ребер) ИГ U. Через $\langle\mathcal{N}\rangle$ обозначим множество записей, соответствующих листьям этой подсети (если α — некоторый лист ИГ U, то под $\langle\alpha\rangle$ будем понимать запись, соответствующую листу α).

Легко видеть, что функция ответа ИГ U определяется соотношением

$$\mathcal{J}_U(x) = \langle\{\alpha \in \mathcal{L}(U) : \varphi_\alpha(x) = 1\}\rangle.$$

Из определения функционирования ИГ видно, что ИГ как управляющая система может рассматриваться в качестве модели алгоритма поиска, работающего над данными, организованными в структуру, определяемую структурой ИГ.

В случае, когда базовое множество переключателей G пусто, то есть в графах нет переключателей, то ИГ называются *предикатными информационными графами* (ПИГ).

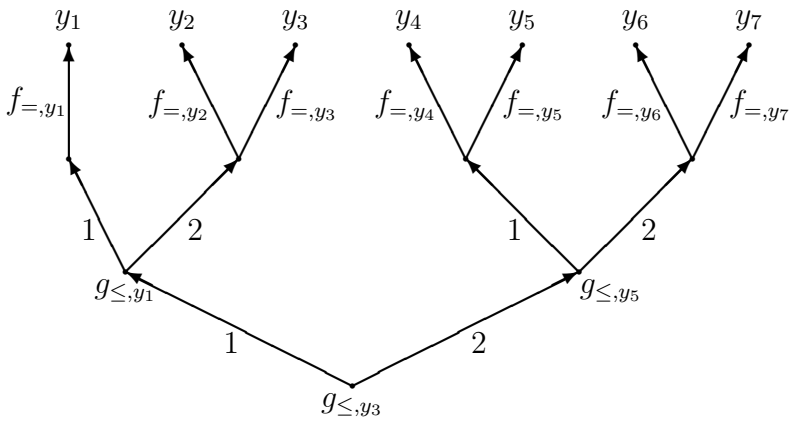

Рис. 2: Информационный граф бинарного поиска

ПИГ, различным листьям которого соответствуют различные записи, называется *однозначным информационным графом* (ОИГ).

ОИГ, имеющий вид дерева, листья которого совпадают с концевыми вершинами дерева, назовем *информационным деревом* (ИД).

ИД удобны и интересны тем, что структуры данных, им соответствующие, практичны и их гораздо проще реализовать на ЭВМ.

Приведем пример еще одного ИГ. Пусть $I = \langle X, V, = \rangle$ — задача поиска идентичных объектов, где на множестве $V = \{y_1, \ldots, y_7\}$ задан линейный порядок и записи упорядочены в порядке возрастания, то есть $y_1 \leq y_1 \leq \cdots \leq y_7$. Пусть множество предикатов имеет вид

$$F = \{f_{=,a}(x) = \begin{cases} 0, & \text{если } x \neq a \\ 1, & \text{если } x = a \end{cases} : a \in X\}, \tag{7}$$

а множество переключателей — вид

$$G = \{g_{\leq,a}(x) = \begin{cases} 1, & \text{если } x \leq a \\ 2, & \text{если } x > a \end{cases} : a \in X\}. \tag{8}$$

Бинарным поиском или *методом деления пополам* называется алгоритм поиска в упорядоченном массиве, при котором массив делится пополам, запрос сравнивается со средней точкой и в зависимости от результата сравнения поиск рекурсивно повторяется в одной из половин.

На рисунке 2 приведен ИГ над базовым множеством $\mathcal{F} = \langle F, G \rangle$, решающий ЗИП I, соответствующий бинарному поиску в версии Боттенбрука, в которой вопрос о равенстве записи и запроса откладывается до самого последнего момента.

Введение управляющей системы, то есть схемы с функционированием, для моделирования алгоритмов поиска позволяет использовать аппарат теории сложности управляющих систем для исследования сложности алгоритмов поиска.

Упражнения

1. По аналогии с одномерной задачей интервального поиска приведите тип, описывающий n-мерные задачи интервального поиска.

2. Опишите тип, задающий задачи интервального поиска на n-мерном булевом кубе, которые состоят в поиске в конечном подмножестве n-мерного булевого куба всех тех точек, которые попадают в подкуб, задаваемый запросом. Какова мощность множества запросов у данного типа?

3. Задача о метрической близости состоит в том, чтобы по произвольно взятой точке-запросу единичного n-мерного куба n-мерного евклидового пространства найти в конечном подмножестве этого куба (библиотеке) все точки, находящиеся на расстоянии не более, чем R от точки запроса. Опишите тип, задающий задачи о метрической близости.

4. Опишите тип, задающий задачи включающего поиска. Напомним, что в задаче включающего поиска имеется некоторое конечное множество свойств, и каждый элемент библиотеки (множества данных) обладает или не обладает каждым из этих свойств. Запрос задает некоторое подмножество множества свойств, и необходимо найти все элементы библиотеки, которые обладают всеми свойствами из запроса.

5. Рассмотрим следующую задачу поиска, которая может возникнуть, например, при разгадывании кроссвордов. Элементы библиотеки (записи) есть слова фиксированной длины n в алфавите $\{0, 1\}$. Запрос задает некоторый набор позиций и значения букв в этих позициях. Необходимо найти в библиотеке все записи, у которых в позициях, задаваемых запросом, стоят буквы, совпадающие с соответствующими значениями позиций запроса. Опишите тип, задающий эти задачи поиска. Сравните полученный тип с типом задач интервального поиска на булевом кубе (см. упражнение 2).