По статье "The Geometry of Binary Search Trees"от Demaine, Harmon, Iacono, Kane, Pătraşcu (SODA, 2010).

1 Введение

Буквы n и m всегда обозначают размер BST и общее количество выполненных операций поиска соответственно.

1.1 BST

В нашей модели вычислений за одно действие можно взять какое-то поддерево, которое затрагивает корень и произвольно его преобразовать (сохраняя свойство BST).

Определение 1 (Переконфигурация)**.** Пусть дано бинарное дерево поиска (BST) T_1, поддерево τ дерева T_1, содержащее корень, и дерево τ' на тех же узлах, что и τ. Скажем, что T_1 может быть преобразовано с помощью операции $\tau \to \tau'$ в другое BST T_2, если T_2 идентично T_1, за исключением замены τ на τ'. Стоимость такой переконфигурации определяется как $|\tau| = |\tau'|$.

Определение 2 (Исполнение BST)**.** Пусть задана поисковая последовательность $S = \langle s_1, s_2, \ldots, s_m \rangle$. Скажем, что алгоритм BST выполняет S через исполнение $E = \langle T_0, \tau_1 \to \tau'_1, \ldots, \tau_m \to \tau'_m \rangle$, если все переконфигурации допустимы и $s_i \in \tau_i$ для всех i (т.е. поиск таки нашел запрашиваемый элемент s_i).

Для $i = 1, 2, \ldots, m$ определим T_i как T_{i-1} с переконфигурацией $\tau_i \to \tau'_i$. Стоимость выполнения E задается как $\sum_{i=1}^{m} |\tau_i|$.

Данная модель эквивалентна другим моделям BST с точностью до постоянного множителя. Например, другой способ моделирования дерева поиска заключается в представлении каждой операции поиска как начинающейся с указателя на корень, с использованием следующих элементарных операций с фиксированной стоимостью: перемещение указателя к левому потомку, перемещение указателя к правому потомку, перемещение указателя вверх, правый поворот в точке указателя, левый поворот в точке указателя. Эта эквивалентность сохраняется, поскольку любое BST может быть преобразовано в любое другое BST с теми же узлами за линейное время.

1.2 Арбореально удовлетворенные множества

Переходя к геометрической интерпретации, точка p обозначает точку на плоскости с целочисленными координатами $(p.x, p.y)$, удовлетворяющими условиям $1 \le p.x \le n$ и $1 \le p.y \le m$. Обозначим за $\square ab$ оси-ориентированного прямоугольника с вершинами a и b (включая границы и внутренность).

Определение 3. Пара точек (a, b) (или индуцированный ими прямоугольник $\square ab$) является *арбореально*[1] *удовлетворенной* относительно множества точек P, если выполняется одно из следующих условий: (1) точки a и b ортогонально коллинеарны (горизонтально или вертикально выровнены); (2) в $\square ab$ содержится хотя бы одна точка из $P \setminus \{a, b\}$.

[1]Термин происходит от латинского слова arbor, что означает «дерево». Он используется в различных контекстах для описания чего-то, связанного с деревьями или имеющего древовидную структуру.

Определение 4. Множество точек P называется *арбореально удовлетворенным*, если каждая пара точек в P является арбореально удовлетворенной относительно P.

Утверждение 1. *В арбореально удовлетворенном множестве точек P для любых $a, b \in P$, которые не являются ортогонально коллинеарными, существует хотя бы одна точка из $P \setminus \{a, b\}$ на сторонах $\square ab$, инцидентных a, и хотя бы одна точка на сторонах, инцидентных b. (Эти две точки могут совпадать.)*

Доказательство. Рассмотрим любые две точки $a, b \in P$, которые не являются ортогонально коллинеарными. Так как $\square ab$ является удовлетворенным, он содержит некоторую точку $c \in P$. Если c не находится на стороне $\square ab$, инцидентной a, тогда мы можем рекурсивно перейти к $\square ac$, пока не найдем такую точку. Аналогично, если c не находится на стороне $\square ab$, инцидентной b, то мы можем рекурсивно перейти к $\square cb$, пока не найдем такую точку. □

Теперь мы визуализируем выполнение алгоритма BST интуитивным способом: на момент времени i (в строке i) мы отображаем все узлы, затронутые в τ_i. Модель BST была выбрана так, чтобы игнорировать лишь несущественные детали (например, точные повороты и перемещения указателя), что упрощает геометрическую интерпретацию.

Определение 5. *Геометрическое представление выполнения* BST E задается как множество точек $P(E) = \{(x, y) \mid x \in \tau_y\}$.

Лемма 1. *Множество точек $P(E)$ для любого выполнения BST является арбореально удовлетворенным.*

Доказательство. Предположим противное, а именно, что существуют $a \in \tau_i$ и $b \in \tau_j$ при $i < j$ и $a \neq b$, и при этом никакие другие узлы в отрезке $[a, b]$ не были затронуты во временном интервале $[i, j]$. Обозначим через c наименьшего общего предка a и b в дереве T_i. Рассмотрим два случая:

- Если $c \neq a$, то c должно быть затронуто в момент времени i, чтобы добраться до a ($c \in \tau_i$), и $c \in (a, b]$. Получаем противоречие.
- Если $c = a$, то в момент времени i узел a является предком b. По предположению, что $\square ab$ не удовлетворено, узел b не затрагивается в промежутке времени $[i, j)$. Следовательно, a остается на пути к b, то есть a должен быть предком b в T_j и будет затронут, что противоречит условию ($a \in \tau_j$).

□

2 Оффлайн-эквивалентность

Геометрическая интерпретация представляется как весьма упрощённое представление выполнения BST, так как она отражает только множество узлов в τ_i и не указывает, каким образом эти узлы должны быть перестроены с помощью поворотов. Довольно неожиданно оказывается, что точная форма дерева не является существенной информацией и может быть восстановлена лишь на основе множеств затронутых узлов! Другими словами, можно реконструировать последовательность выполнения по любому геометрическому представлению, удовлетворяющему необходимому условию арбореальной удовлетворённости.

Лемма 2. *Для любого арбореально удовлетворенного множества точек X существует выполнение BST E такое, что $P(E) = X$. Назовём E арбореальным представлением X и обозначим $P^{-1}(X) = E$.*

Доказательство. Опишем алгоритм для обратного преобразования $P^{-1}(\cdot)$, смотреть Картинку 1. Определим *время следующего доступа* $N(x, i)$ для x в момент времени i как минимальную координату y среди всех точек в X на луче от (x, i) до (x, ∞). Если такой точки нет, положим $N(x, i) = \infty$.

Пусть T_i — это Декартово дерево, построенное на всех точках $(x, N(x, i))$. Напомним, что Декартово дерево представляет собой BST по первой координате и кучу по второй, где совпадающие значения разрешаются произвольно. Таким образом, T_i является корректным BST на n значениях, удовлетворяющим свойству кучи согласно времени следующего доступа (с минимумом в корне).

Пусть τ_i — это точки в X с $y = i$. По свойству Декартового дерева T_i, множество τ_i должно образовывать связное поддерево T_i, включающее корень (так как i является минимальным возможным временем доступа $N(*, i)$). Далее, формируем T_{i+1}, переставляя узлы в τ_i так, чтобы они образовали Декартово дерево, основанное на времени следующего доступа $(x, N(x, i + 1))$.

Покажем, что T_{i+1} является Декартовым деревом на $(x, N(x, i + 1))$. Свойство BST выполняется по построению, поэтому рассмотрим свойство кучи. Достаточно показать, что оно выполняется для каждой пары родитель/потомок (q, r) в T_{i+1}. Если оба узла находятся в τ_i, то свойство кучи сохраняется по построению. Если оба узла находятся вне τ_i, то их время следующего доступа и их отношение родитель/потомок не изменились при переходе от i к $i + 1$, а значит, свойство кучи также выполняется. Остаётся случай, когда $q \in \tau_i$, а $r \notin \tau_i$. Однако если свойство кучи нарушается в T_{i+1}, то прямоугольник от (q, i) до $(r, N(r, i))$ противоречит Утверждению 1: вертикальная сторона при $x = q$ пуста, поскольку $N(q, i + 1) > N(r, i)$ (в силу предположения о нарушении свойства кучи). Горизонтальная сторона при $y = i$ также пуста, так как в противном случае найдется некий $x \neq q$ на этой стороне и тогда r, q были бы в разных ветвях относительно x в дереве T_{i+1}, противоречие. □

Пусть *геометрическое представление* последовательности доступа S определяется как множество точек $P(S) = \{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$. Леммы 1 и 2 показывают, что арбореальное утверждение «E выполняет S» эквивалентно геометрическому утверждению «$P(S) \subseteq P(E)$». Обозначим через $\mathrm{minASS}(S)$ размер наименьшего арбореально удовлетворённого над-набора множества $P(S)$. Тогда имеем $\mathrm{OPT}(S) = \mathrm{minASS}(P(S))$. Таким образом, вопрос того чтобы находить/аппроксимировать $\mathrm{OPT}(S)$ эквивалентен разработке алгоритмов для поиска минимального арбореально удовлетворённого над-набора.

3 Онлайн-эквивалентность

Выше мы установили комбинаторную эквивалентность между деревьями поиска и арбореально удовлетворёнными множествами, тем самым охарактеризовав офлайн-алгоритмы BST. Теперь мы стремимся усилить эту характеристику для *онлайн*-алгоритмов BST, которые должны выполнять преобразование $\tau_i \to \tau_i'$ после каждого s_i без доступа к последующим запросам.

Определение 6. Задача *онлайн-арбореально удовлетворённого над-набора* (online ASS) заключается в разработке алгоритма, который получает множество точек $\{(s_1, 1), (s_2, 2), \ldots, (s_m, m)\}$ поступательно.

После получения точки i алгоритм должен вывести множество P_i точек на линии $y = i$ так, чтобы

$$\{(s_1, 1), (s_2, 2), \ldots, (s_i, i)\} \subseteq P_1 \cup P_2 \cup \cdots \cup P_i$$

было арбореально удовлетворённым. Стоимость алгоритма определяется как $\sum_{i=1}^{n} |P_i|$.

Онлайн-алгоритм BST естественным образом определяет онлайн-алгоритм ASS через стандартное геометрическое представление (Лемма 1). Обратное утверждение не столь очевидно, так как Лемма 2 требует знания будущих доступов: она реконструирует форму дерева, накладывая порядок кучи, основанный на будущих временах доступа. Однако, используя следующую концепцию, мы сможем «угадывать» форму дерева динамически, теряя лишь постоянный множитель во времени работы.

Определение 7. *Split-дерево* — это абстрактный тип данных, реализующий две операции в модели BST:

- `MakeTree`$(x_1, x_2, \ldots, x_n)$ — создаёт BST из n узлов на основе заданных значений.
- `Split`(x) — перемещает x в корень дерева, а затем удаляет его, оставляя левые и правые поддеревья корректными split-деревьями.

Утверждение 2. Split-деревья можно реализовать с худшим случаем стоимости $O(n)$ для `MakeTree` и произвольной последовательности из n операций `Split` (т.е. амортизировано $O(1)$).

Доказательство. Опущен. □

Лемма 3. *Для любого онлайн-алгоритма ASS A существует онлайн-алгоритм BST A', такой что для любой последовательности доступа стоимость A' ограничена сверху константным множителем от стоимости A.*

Доказательство. Основная идея заключается в том, что когда нам нужно сохранить множество элементов в некотором неизвестном будущем порядке, мы избегаем принятия решений и храним их в split-дереве. Получаемая конструкция может рассматриваться как инверсия доказательства Леммы 1, поскольку split-деревья хранятся в порядке кучи по времени предыдущего доступа (вместо времени следующего доступа).

Формально, пусть $\rho(x, i)$ — это время последнего доступа к x перед i, то есть y-координата наивысшей точки на луче от (x, i) до $(x, -\infty)$. Если такой точки нет, полагаем $\rho(x, i) = -\infty$. Обозначим через G_i *общее Декартово дерево*, определённое на всех точках $(x, \rho(x, i))$, то есть BST по координатам x и *общий heap* по $\rho(x, i)$ (в этот раз на максимум). В общем heap'е совпадающие ключи объединяются в *суперузел* с несколькими значениями. Т.е. каждая вершина Декартового дерева это множество элементов с одинаковым $\rho(x, i)$ ключи которых хранятся в split-дереве.

Рассмотрим, как эта структура изменяется при переходе от момента времени i к $i+1$. Все точки на строке $y = i+1$ будут иметь $\rho(x, i+1) = i+1$ и будут перемещены из G_i в корень G_{i+1}.

Ключевое свойство, вытекающее из арбореальной удовлетворённости (и которым мы уже пользовались в Лемме 1) это то, что вершины с $\rho(x, i+1) = i+1$ будут образовывать связный подграф включающий корень в G_{i+1}. Действительно, пусть есть такая тройка вершин $a < b < c$, что $\rho(a, i+1) = \rho(c, i+1) = i+1$ и $\rho(b, i+1) < i+1$ и при этом b лежит между a, c. Значит, c находится внутри

Это означает, что мы можем переместить все соответствующие значения в корень одним обходом дерева от корня к листьям. На каждом суперузле мы вызываем `Split` для выделения запрашиваемых узлов, формируя новый скелетный узел и два дочерних суперузла. Затем все собранные скелетные узлы объединяются в корневой суперузел с помощью `MakeTree`.

Поскольку каждая операция `Split` имеет амортизированную константную стоимость, а `MakeTree` выполняется за линейное время, общая стоимость моделирования доступа на строке $y = i$ пропорциональна числу точек на этой строке. □

4 Greedy Future

Данный кусок не был затронут на лекции.

Алгоритм 1 (GreedyFuture (GF) Algorithm)**. Input:** Последовательность запросов $X \in [n]^m$ и начальное BST T_0. Мы перестраиваем T_{t-1} в T_t после обработки запроса x_t с T_{t-1} для $t = 1, \ldots, m$.

Function Restructure(*запрос* v, *дерево* T_{t-1}, *будущие запросы* X'):

1: Пусть $v_1 < v_2 < \cdots < v_k$ — узлы на пути от корня T_{t-1} до запрашиваемого значения v (включая v и корень).
2: Определим $v_0 = -\infty$ и $v_{k+1} = +\infty$.
3: Обозначим поддеревья, отходящие от этого пути, как $R_0, \ldots, R_k$.
4: Положим $\tau(v_i)$ — индекс первого появления запроса значения $x \in (v_{i-1}, v_{i+1})$ в X' для каждого $i \in [k]$.
5: Переструктурируем узлы $v_1, \ldots, v_k$ в Декартово дерево:
- Дерево поддерживает порядок BST.
- Приоритеты в heap'е определяются значениями τ, где корень имеет наименьшее τ.
- При равных значениях предпочтение отдаётся, например, меньшим ключам или узлам с меньшей глубиной до реструктуризации.

6: Подвесим поддеревья $R_0, \ldots, R_k$ на их соответствующие позиции.
7: Возвращаем полученное дерево T_t.

Данный жадный алгоритм предполагался как такой, что работает лучше любого онлайн алгоритма. Оказывается, что его можно сделать онлайн с потерей в константный множитель.

Алгоритм 2. Проводим сканирующую горизонтальную линию по множеству точек в порядке увеличения координаты y. В момент времени i, GreedyASS добавляет минимальное количество точек на $y = i$, чтобы множество точек вплоть до $y \le i$ стало арбореально удовлетворённым.

Этот минимальный набор точек определяется однозначно: для любого неудовлетворённого прямоугольника, содержащего (s_i, i) в одном из углов, добавляется противоположный угол при $y = i$.

Теорема 4. *При применении Леммы 2 к Алгоритму 2 получается Алгоритм 1.*

Доказательство. Доказательство по индукции, на очередном шаге (s_i, i) достаточно показать, что Алгоритм 2 достроит ровно те точки, которые в дереве Алгоритма 1 будут на пути от корня до s_i. Далее, по Лемме 2 на этом пути построится Декартово дерево, также как и в обычном GF.

Рассмотрим вершину v на пути от корня до s_i в дереве T_{i-1} и пусть j – момент когда последний раз к ней был доступ. Пусть $\square(s_i, i)(v, j)$ арбореально удовлетворённый, тогда найдется некая точка (c, k) на границе этого прямоугольника. Тогда $k = j$, ведь иначе бы мы взяли точку v повыше. Но тогда c лежит строго между s_i, v. А следовательно, путь прошел бы по нему. В обратную сторону, аналогично. □

Заметим, что Алгоритм 2 не смотрит в будущее. Этот онлинифицированный вариант «максимально оффлайнного» алгоритма, по-видимому, указывает на то, что оффлайн-алгоритмы не могут асимптотически превосходить лучшие онлайн-алгоритмы в модели BST, то есть динамическая оптимальность возможна.

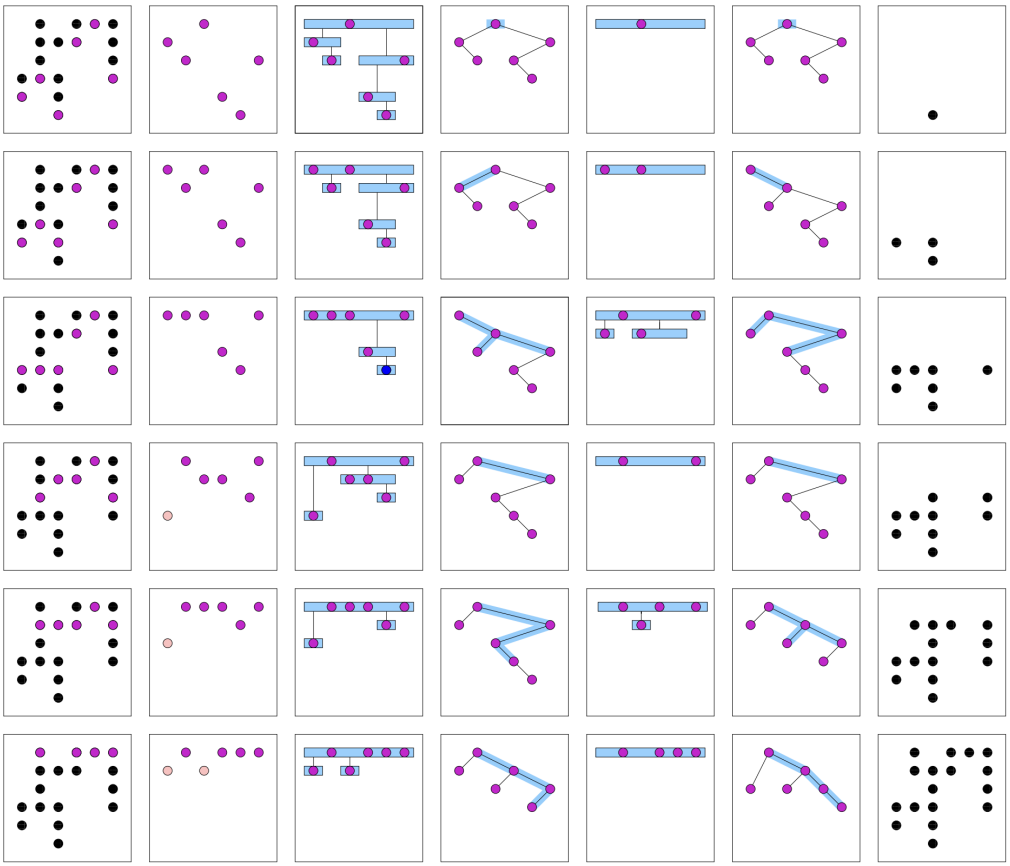

Рис. 1: Иллюстрация преобразований между деревом и геометрическим представлением. Колонки слева направо: (1) Множество точек X; в строке i светлые узлы представляют появления, определяющие $N(x, i)$. (2) Светлые узлы из колонки 1 перерисованы и отражены по вертикали для упрощённого преобразования в дерево; ещё более светлые узлы, для которых $N(x, i) = \infty$, располагаются внизу. (3) Общее Декартово дерево. (4) Дерево T_{i-1}. (5) Декартово Дерево, сформированное по времени следующего доступа узлов τ_i в момент $i + 1$ (произвольная бинаризацию). (6) Дерево T_i. (7) затенённые узлы τ из колонок 4 или 6, восстанавливая исходное множество точек из колонки 1.