По статье "Alphabetic minimax trees in linear time"от Pawel Gawrychowski.

Когда мы здесь будем говорим, что какое-то одно число делится на другое (оба числа при этом могут быть нецелыми), мы имеем в виду, что их отношение целое.

1 Введение

Речь будет идти об *оптимальных деревьях*. С ними не подразумевается никаких операций слияния, добавления элемента и т.д., а задача поставлена следующим образом: даны веса $w_1, w_2, ..., w_n$, и по ним нужно построить в некотором смысле оптимальное двоичное дерево с n листьями, в которых расположены наши веса. Существуют разные способы конкретизировать это понятие. Давайте обозначим за $L_1, L_2, ..., L_n$ пути от корня до всех n листов искомого дерева, упорядоченные слева направо, а за $l_1, l_2, ..., l_n$ их длины. Здесь уже появляются различные вариации: мы можем не накладывать никаких ограничений на порядок w_i, а можно рассмотреть *алфавитную* версию задачи, где мы требуем, чтобы w_i были в данном порядке расположены по листьям, т.е. на конце L_i расположено w_i. Мы будем рассматривать именно алфавитную версию. Далее, каждому L_i мы присваиваем вес. Можно, например, положить его равным произведению $l_i \cdot w_i$, но мы будем рассматривать веса $l_i + w_i$. Наконец, по весам путей можно определить вес всего дерева: можно взять сумму всех весов путей L_i (что в нашем случае не имеет смысла, но в случае $l_i \cdot w_i$ является известной задачей), а можно взять максимум, что мы и будем делать. Итого, мы хотим найти дерево, которое минимизирует $\max\limits_i(l_i + w_i)$, отсюда и в название статьи alphabetic minimax trees. Сам минимум назовем $M(w_1, ..., w_n)$.

2 Правильные формы

Назовем формой дерева последовательность введенных нами длин путей $< l_1, l_2, ..., l_n >$. Последовательность натуральных чисел, соответственно, назовем правильной формой, если она является формой какого-то дерева. Как по последовательности натуральных чисел понять, является ли она правильной формой или нет? Если бы мы не упорядочивали эти длины, то равносильное условие мы знаем из базового курса по дискретной математике:

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1$$

.

Но в нашем случае все несколько сложнее. Давайте введем функцию $f_a(x) = \frac{\lceil 2^a x \rceil + 1}{2^a}$ для целых a.

Теорема 1. *(Неравенство Янга) Последовательность натуральных чисел $< l_1, ..., l_n >$ является правильной формой тогда и только тогда, когда*

$$f_{l_n}(f_{l_{n-1}}(...(f_{l_1}(0)...) \leq 1$$

Доказательство. Давайте переходу влево вниз сопоставим 0, а переходу вправо вниз сопоставим 1. Таким образом, каждой вершине двоичного дерева соответствует уникальная строка из 0 и 1, если мы рассмотрим путь от корня до нее. Поставим эту строку после 0 с запятой, и получим какое-то число в $[0, 1)$ в двоичной записи (из-за возможных нулей на конце это отображение уже не обязательно инъективно). Индукцией по i докажем, что если у дерева форма $< l_1, ..., l_n >$, то i-ому слева листу соответствует число m_i, которое не меньше $p_i = f_{l_{i-1}}(...(f_{l_1}(0)...)$.

База $i = 1$ очевидна: у нас все числа, соответствующие вершинам, не меньше 0. Переход $i \to i+1$: по предположению мы знаем, что i-ому листу соответствует число хотя бы p_i. Кроме того, i-ый лист находится на глубине l_i, а значит m_i делится на 2^{-l_i}, поэтому $m_i \geq \frac{\lceil 2^{l_i} p_i \rceil}{2^{l_i}}$, т.к. это наименьшее число, которое хотя бы p_i и кратно 2^{-l_i}. Поскольку $i+1$-ый лист находится строго правее i-ого, это как раз означает, что если оставить только первые l_i цифр после запятой, то получится строго большее число, а значит $m_{i+1} \geq f_{l_i}(p_i) = p_{i+1}$, что и требовалось.

Повторяя начало этого рассуждения для $i = n$, получаем, что $m_n \geq \frac{\lceil 2^{l_n} p_i \rceil}{2^{l_n}}$, но также $m_n < 1$, поэтому $1 > \frac{\lceil 2^{l_n} p_i \rceil}{2^{l_n}}$. Обе части этого нер-ва делятся на 2^{-l_n}, а значит можно прибавить к правой части 2^{-l_n} и заменить строгое неравенство на нестрогое, получится $1 \geq f_{l_n}(p_n)$, что и требовалось.

В другую сторону: если неравенство Янга выполняется для последовательности $< l_1, ..., l_n >$, то для построения дерева просто действуем жадным алгоритмом как в нашей индукции. А именно, в полном бесконечном двоичном дереве в качестве i-ого листа нашего искомого дерева выберем вершину на глубине l_i, которой соответствует число $\frac{\lceil 2^{l_i} p_i \rceil}{2^{l_i}}$, что можно сделать, т.к. это число делится на 2^{-l_i}.

□

Нам будет удобнее работать с этим неравенством в немного другом виде: давайте все наши промежуточные p_i домножим на 2^n, тогда нетрудно видеть, что вместо f_{l_i} переходной функцией будет g_{n-l_i}, где $g_a(y) = \lceil \frac{y}{2^a} \rceil 2^a + 2^a$, т.е. неравенство Янга превращается в

$$g_{n-l_n}(...(g_{n-l_1}(0)...) \leq 2^n$$

Заметим еще, что если среди целых l_i есть хоть одно отрицательное, то это неравенство не может выполняться, поэтому условие равносильно не только для натуральных, но и для любых целых чисел.

3 Линейный алгоритм для целых w_i

Мы будем работать в так называемой word RAM модели, где w_i – целые числа в промежутке $[0, n-1]$, а операции с числами в этом промежутке по типу сложения, сравнения и т.д. выполняются за константное время (из этого следует, что аналогичные операции проводятся за константное время с числами из $O(\log n)$ бит). На самом деле, нетрудно понять, что ограничение на размер w_i не ограничивает общность задачи, т.к. к такому промежутку все сводится. Обозначим за $g_{a_1,a_2,...,a_k}$ для краткости композицию $g_{a_1} \circ g_{a_2} \circ ... \circ g_{a_k}$.

Понятно, что раз w_i целые, то и $M(w_1, ..., w_n)$ тоже будет целым. Мы хотим найти минимальное целое M, что существуют целые l_i с условиями $l_i + w_i \leq M$, что $g_{n-l_n,...,n-l_1}(0) \leq 2^n$. Из определения функции g_a видно, что она монотонна по a: $g_{a+1} > g_a$. Поэтому для ответа M выполняется и неравенство $g_{n-M+w_n,...,n-M+w_1}(0) \leq 2^n$. И наоборот, если такое неравенство выполняется для какого-то целого M, то положив $l_i = M - w_i$, будет выполняться и условие на правильность формы искомого дерева, т.е. это M достигается.

Итого, мы ищем наименьшее целое M, что $g_{n-M+w_n,...,n-M+w_1}(0) \leq 2^n$. Заметим, что $g_{a+1}(2y) = 2g_a(y)$, значит при прибавлении к M целого b, значение $g_{n-M+w_n,...,n-M+w_1}(0)$ делится на 2^b. Это значит, что достаточно посчитать $g_{n-m+w_n,...,n-m+w_1}(0)$ для какого-то конкретного целого m, найти наименьшее целое c, что $g_{n-m+w_n,...,n-m+w_1}(0) \leq 2^c$, и ответом будет $m + c - n$.

Осталось эффективно посчитать эту композицию для какого-то m. Выберем $m = \max\limits_j w_j$. Благодаря такому выбору, имеем $0 \le n - m + w_i \le n$. Из этого следует, что при последовательном применении функций g_{n-m+w_i} будут получаться только целые числа, а также индукцией по k получаем $g_{n-m+w_k,\dots,n-m+w_1}(0) \le k \cdot 2^n$: действительно, в переходе индукции если применить к этому еще $g_{n-m+w_{k+1}}$, то получится не более $\lceil \frac{k \cdot 2^n}{2^{n-m+w_{k+1}}} \rceil 2^{n-m+w_{k+1}} + 2^{n-m+w_{k+1}} = k \cdot 2^n + 2^{n-m+w_{k+1}} \le (k+1)2^n$. Итого, все числа в процессе будут не более $n \cdot 2^n$. Но нам нужно сделать линейное число операций, а у $n \cdot 2^n$ совсем не $O(\log n)$ бит, а $O(n)$. Для того, чтобы обойти эту проблему, мы будем представлять числа не в простой двоичной записи, а строкой из номеров битов, где есть единичка. Заметим, что при применении очередного g в таком виде легко посчитать $\lceil \frac{y}{2^a} \rceil 2^a$, а также затем прибавить 2^a. Количество единичек в двоичной записи при этом увеличивается максимум на 1, и даже уменьшается, если есть переносы через разряд при прибавлении 2^a. Благодаря этому и тому, что номера битов с единичками, как мы доказали, записываются $O(\log n)$ битами, применение g занимает амортизированную константу, что мы и хотели.

4 Алгоритмы для вещественных w_i

Мы по-прежнему будем считать, что операции с нашими числами по типу сложения занимают константу, а также что целые части наших w_i попадают в $[0, n-1]$ (это ограничение опять не нарушает общности). Начнем с двух лемм, которые математически полностью сводят вещественный случай к целому.

Лемма 2. $[M(w_1, ..., w_n)] = M([w_1], ..., [w_n])$

Доказательство. Пользуясь в конце тем, что l_i целые, получаем цепочку равенств

$$[M(w_1, ..., w_n)] = [\min_T \max_i (l_i + w_i)] = \min_T [\max_i (l_i + w_i)] = \min_T \max_i [l_i + w_i] = \min_T \max_i ([w_i] + l_i) =$$

$$= M([w_1], ..., [w_n])$$

□

Лемма 3. *Пусть X – вещественное. Тогда $M(w_1, ..., w_n) \le X \Leftrightarrow M(w'_1, ..., w'_n) \le [X]$, где $w'_i = [w_i] + \mathbb{1}_{\{w_i\} > \{X\}}$*

Доказательство. $M(w_1, ..., w_n) \le X$ тогда и только тогда, когда для некоторой правильной формы $< l_1, ..., l_n >$ для всех i выполняется $w_i + l_i \le X$. Дробная часть левой части совпадает с $\{w_i\}$. Поэтому, если $\{w_i\} > \{X\}$, то неравенство равносильно $[w_i] + l_i + 1 \le [X]$, а в противном случае неравенство равносильно $[w_i] + l_i \le [X]$. Т.е. как раз получается $w'_i + l_i \le [X]$ для всех i. Существование такой правильной формы равносильно $M(w'_1, ..., w'_n) \le [X]$. □

Таким образом, $[M(w_1, ..., w_n)]$ мы можем найти за линейное время по предыдущему алгоритму. Заметим также, что $\{M(w_1, ..., w_n)\}$ обязательно совпадает с $\{w_i\}$ для какого-то i. Поэтому, например, с помощью последней леммы можно бинпоиском перебрать дробные части всех w_i в качестве $\{X\}$, каждый раз пользуясь алгоритмом для целых w'_i, таким образом получив алгоритм за $O(n \log n)$. Но на самом деле есть алгоритм за линейное время. Времени его разобрать его не хватило, но мы посмотрим на основные идеи.

4.1 Алгоритм за $O(nd)$

Лемма 4. *Для целых $a_1, ..., a_k$ функция $g_{a_1,...,a_k}$ имеет вид $g_{a_1,...,a_k}(x) = \lceil \frac{x+r}{2^a} \rceil 2^a + c$, где $a = \max\limits_i a_i$, а числа r и c имеют вид $\sum_{i=1}^{k} \alpha_i 2^{a_i}$, где все $\alpha_i \in \{0, 1, 2\}$.*

На доказательство леммы у меня времени во время доклада не хватило, поэтому и тут этого не будет, но доказывается она простой индукцией по k, желающие могут посмотреть на страницу 11 в оригинальной статье.

В нашем случае в качестве a_i нас интересуют $n-m+[w_j]$ или $n-m+[w_j]+1$, а подставить в качестве x мы хотим 0. Тогда из этой леммы следует, что если отмерить $O(\log n)$ битов влево от бита на позиции $n-m+[w_i]$ для всех i, то такие отрезки покроют все ненулевые биты числа $g_{n-m+w'_k, n-m+w'_{k-1}, ..., n-m+w'_1}(0)$ вне зависимости от конкретных значений w'_i.

Обозначим за d мощность множества всех $[w_i]$ (без учета кратности, конечно). Так, можно ввести *лаконичное представление* наших чисел: мы храним d чисел из $O(\log n)$ бит, и указатели на места, где они должны стоять. Аналогично вводится *лаконичное представление функции*: это просто совокупность из числа a и лаконичных представлений чисел r, c из последней леммы.

Лемма 5. *По лаконичным представлениям функций g_1, g_2 можно за $O(d)$ посчитать лаконичное представление их композиции.*

Доказательства опять не будет (его можно увидеть на 12 странице оригинальной статьи), но лаконичная форма композиции чисто математически находится в явном виде, и получаются операции по типу сложения в каждом из d блоков по $O(\log n)$ бит.

В начале алгоритма за $O(nd)$ мы так же находим целую часть нашего ответа, и затем перебираем бинпоиском дробную часть X, но при этом на каждом шаге мы будем действовать по-другому. Мы поддерживаем растущие множества L, R, где L – индексы i, для которых мы знаем, что истинное значение w'_i равно $[w_i]$, а R – для которых равно $[w_i]+1$ (истинное значение – это при $\{X\} = \{M(w_1, ..., w_n)\}$), за C обозначим все остальные индексы. Соответственно, когда мы в качестве очередной дробной части в бинпоиске рассматриваем $\{w_i\}$, то если оказывается, что реальная дробная часть меньше, то все w_j-ые с такой или большей дробной частью добавляются в R, а в противном случае все w_i с меньшей дробной частью добавляются в L. Суть в том, что когда какой-то индекс i попадает в L или R, для него с этого момента значение w'_i не будет меняться для дальнейших шагов, и мы можем использовать эту информацию. А именно, множество $L \cup R$ мы разобъем на максимальные отрезки подряд идущих индексов в нем, и для каждого отрезка мы будем хранить лаконичную форму композиции соответствующих $g_{n-m+w'_i}$. После каждого шага, когда у нас расширяется L или R, мы склеиваем все рядом стоящие отрезки индексов в $L \cup R$ пока не получим максимальные отрезки подряд идущих индексов для нового $L \cup R$. За все шаги между двумя соседними индексами произойдет ровно одно склеивание, а на одно склеивание уходит $O(d)$, поэтому за все шаги на это уйдет $O(nd)$ времени. Кроме того, на каждом шаге мы считаем композицию из $O(|C|)$ функций g (пользуясь тем, что для максимальных отрезков $L \cup R$ все посчитано, и между двумя максимальными отрезками должен быть индекс из C), чтобы применить лемму 3. Каждый раз $|C|$ падает в 2 раза, поэтому у нас уйдет $O(nd + \frac{nd}{2} + \frac{nd}{4} + ...) = O(nd)$ времени, что и требовалось.

4.2 Идеи для линейного алгоритма

Для линейного алгоритма мы поделим индексы от 1 до n на блоки подряд идущих индексов по $\log n$ в каждом. Тогда из леммы 4 следует, что в выражении r, c композиции какого-то

набора g в пределах одного блока есть только $O(\log n)$ бит, поэтому для каждого блока можно ввести свое лаконичное представление функций, которое будет занимать только $O(\log n)$ бит. Далее мы действуем как в алгоритме за $O(nd)$, только максимальные отрезки индексов в $L \cup R$ мы рассматриваем только в пределах каждого блока. На их постепенное склеивание при расширении $L \cup R$, уйдет $O(n)$ (поскольку аналог леммы 5 для лаконичных представлений из $O(\log n)$ битов – это подсчет композиции за константное время). Аналогично на подсчет композиции g для каждого блока за все шаги уйдет $O(n + \frac{n}{2} + ...) = O(n)$ времени. Осталась основная и самая сложная проблема: на каждом шаге посчитать композицию $\frac{n}{\log n}$ функций, соответствующих блокам, от нуля, что для нужного нам времени хочется посчитать за $O(\frac{n}{\log n})$. Они имеют лаконичные представления на разных битах, поэтому аналогом леммы 5 это сделать не получится. Для этого вводится *гибридное представление* числа в двоичной записи, которое состоит из данных вида:

1) Кусок подряд идущих $\leq \log n$ бит в числе.

2) Разреженный кусок: не более $\log n$ битов, между которыми стоят только нули, причем эти биты в разреженном куске стоят в точности на позициях, соответствующих лаконичным представлениям одного из блоков.

Гибридное представление числа – это упорядоченная по положению в числе последовательность данных такого вида, которые не пересекаются и покрывают все ненулевые биты в числе. Оказывается, такое представление как раз позволяет считать композиции лаконичных (хоть и на разных битах) представлений функций от нуля за амортизированное константное время, что мы и хотим. На доказательство, конечно, времени не хватило.