Аннотация

Дерево отрезков - это мощная структура данных, которая позволяет эффективно решать множество задач различных тематик. Данный диплом посвящен построению оптимального дерева в случае известного распределения вероятностей запросов. Было предложено как точное решение, так и асимптотически оптимальное приближенное решение. На наших тестах приближенное решение показало достаточно небольшое ухудшение по сравнению с оптимальным.

Содержание

1 Введение

1.1 Определения

Пусть дан массив из n элементов.

Дерево отрезков - это двоичное дерево, в котором:

- Есть n листьев, соответствующих отрезкам единичной длины
- Есть вершины с двумя сыновьями. Правый сын соответствует отрезку, следующему сразу за отрезком левого сына. Вершина соответствует объединению отрезков сыновей
- Корень дерева соответствует всему массиву (отрезку $[1; n]$).

Запрос сверху на отрезке $[L, R]$ начинается в корне. Если сейчас рассматривается вершина, отрезок которой не лежит полностью в отрезке $[L, R]$, то запрос рекурсивно вызывается от тех сыновей, отрезки которых пересекаются с $[L, R]$. Иначе рекурсивных вызовов от сыновей не происходит. В обоих случаях вершина считается посещенной и в ней выполняются какие-то действия, специфичные для запроса.

Пример запроса сверху указан на рисунке 1. Здесь дерево отрезков на пяти элементах и запрос на отрезке $[2; 4]$. Будет посещено пять выделенных вершин.

1.2 Постановка задачи

Дано: Распределение вероятностей на запросах-отрезках границами из $[1; n]$

Необходимо: построить дерево отрезков, для которого минимально среднее количество посещенных вершин при запросах сверху.

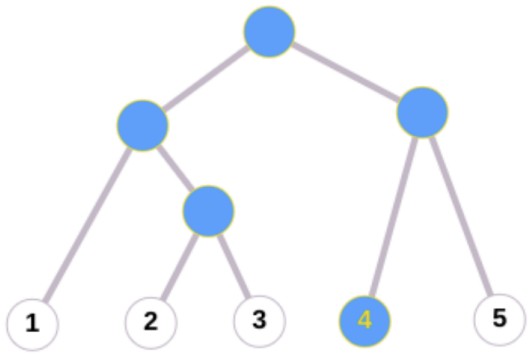

Рисунок 1 – Пример запроса к дереву отрезков

Интересует как точное решение за как можно более лучшую асимптотику, так и приближенное за сложность нахождения $O(n+S)$ или $O(n+S\log S)$, где S - количество отрезков с ненулевой вероятностью

1.3 Мотивация

Дерево отрезков - это мощная структура данных, которая позволяет решать большое количество задач. Если дан массив a из n элементов, то она позволяет эффективно:

- Отвечать на запросы $f(a_L, f(\dots, a_R)\dots)$, где f - произвольная ассоциативная функция с двумя аргументами
- Изменять элементы $a_L, \dots, a_R$ для некоторых типов изменений

Пример: запросы суммы/минимума на подотрезке массива и прибавления числа ко всем элементам подотрезка массива.

Обычно для этого в каждой вершине дерева отрезков хранится значение функции на соответствующем подотрезке массива и информация о том, нужно ли применить какие-то отложенные изменения к сыновьям данной вершины.

Запросы на расчёт функции находятся с помощью запроса сверху и объединения результатов для посещенных вершин. Запросы на изменение тоже

совершаются с помощью запросов сверху, для посещенных вершин происходит изменение информации в них нужным образом.

Если для каждой вершины дерева с двумя сыновьями её отрезок разбивается примерно пополам и каждой половине соответствуют её сыновья, то можно показать, что при запросе сверху посещается $O(\log n)$ вершин.

При этом большое количество задач различных тематик можно свести к запросам на подотрезках массива. Примеры таких задач:

- Для дерева из n вершин после препроцессинга за $O(n)$ можно отвечать на запросы минимального общего предка за $O(\log n)$
- Для строки из n символов после построения суффиксного массива и дополнительного препроцессинга за $O(n)$ можно находить длину наибольшего общего префикса двух любых её подстрок
- Если нам дано n прямоугольников на плоскости со сторонами, параллельными осям координат, то можно найти площадь их объединения за $O(n \log n)$

Как уже было сказано, если для каждой вершины разбивать её подотрезок на две равные части, то сложность запроса к дереву отрезков $O(\log n)$. Но нам может быть известно распределение вероятностей запросов, и в таком случае можно построить более оптимальное дерево. Таким образом решение задачи, поставленной в дипломе, может привести к ускорению на большом классе задач и может представлять не только теоретический, но и практический интерес.

Кроме того надо заметить, что данный диплом далеко не первый, посвященный вопросу построения оптимальных структур данных при известном

распределении вероятностей запросов. В частности, смежная задача построения статически оптимального дерева поиска хорошо изучена, её результаты будут использоваться в дальнейшем.

2 Обзор смежной задачи

2.1 Определения и постановка задачи

Хорошо изучена похожая задача нахождения статически оптимального дерева поиска.

Постановка этой задачи такова: есть множество из n различных упорядоченных элементов и вероятности:

- $A_1, \ldots, A_n$, что запрос будет с значением, равным соответствующему элементу множества
- $B_0, \ldots, B_n$, что запрос будет лежать до первого элемента, либо между двумя элементами, либо после последнего

Необходимо найти дерево поиска на данном множестве такое, что среднее количество посещенных вершин при запросах будет минимально.

Дерево поиска - это двоичное дерево такое, что

- Есть n вершин, в которых стоят различные значения из данного множества
- Если в вершине стоит число x, то у всех вершин в её левом поддереве значения меньше чем x, в правом поддереве больше чем x

Запрос к дереву поиска выглядит таким образом. Он начинается в корне. Если число из запроса меньше, чем число в текущей вершине, то переходим в её левого сына. Если больше, то в правого. Иначе, а также если мы попытались пойти в несуществующую вершину, запрос завершается.

2.2 Решения

В статье [1] было получено точное решение за $O(n^2)$ времени и памяти.

Идея такова - можно найти $dp_{L,R}$ - оптимальное среднее количество посещенных вершин, если дерево построено на значениях с L по $R-1$. Например, для случая нулевых B_i верно, что

- $dp_{L,L} = 0$
- $dp_{L,R} = A_L + \cdots + A_{R-1} + \min\limits_{L \leqslant m < R} dp_{L,m} + dp_{m+1,R}$

Это верно, так как на каждом шаге выбирается элемент в корне поддерева, а затем задачу можно свести к задачам для левого и правого поддерева. При этом каждый запрос пройдёт через корень поддерева и его вероятность нужно прибавить.

Так как $dp_{L,R}$ зависит только от значений dp для меньших по длине отрезков, то если перебирать отрезки по неубыванию, $dp_{L,R}$ можно найти за $O(n^3)$ времени. Дальше в статье доказывалось, что в данном случае можно оптимизировать решение до $O(n^2)$ времени.

В статье [2] было получено приближенное решение за $O(n)$ времени. Идея такова - на каждом шаге выбирать такое значение в корне поддерева, что суммы вероятностей запросов в подзадачах для левого и правого поддерева как можно более близки друг к другу, после этого рекурсивно запустить решение для поддеревьев.

Было показано, что $0.63H \leqslant P_{opt} \leqslant P_{approx} \leqslant 2 + 1.44H$, где H - энтропия распределения вероятностей, P_{opt} - среднее количество посещенных вершин в оптимальном решении, P_{approx} - в приближенном. Таким образом, приближенное решение не хуже оптимального более чем в константу раз, где константу можно оценить сверху как 2.29

В статье [3] было приведено точное решение за $O(n \log n)$ времени для важного случая, когда вероятности A_i равны нулю.

В статье [4] было приведено обобщение того, когда точные решения за $O(n^3)$ времени могут быть соптимизированы до $O(n^2)$

3 Точное решение за $O(n^3 + S)$ времени и $O(n^2)$ памяти

3.1 Решение за $O(n^3 + n^2 S)$ времени

Пусть вес запроса - это то же самое, что и его вероятность.

Пусть вес дерева отрезков - это сумма по всем данным запросам произведения их веса на количество посещенных вершин в дереве отрезков.

Введём понятие дерева отрезков, построенного на $[L, R]$ и его веса. Корень такого дерева отрезков соответствует отрезку $[L, R]$. Запросы, не пересекающиеся с $[L, R]$ убираются из рассмотрения, а для остальных берется их пересечение с отрезком $[L, R]$.

Дальше несколько фактов:

- Для отрезка $[L, R]$ к весу дерева отрезков прибавляется сумма весов запросов, которые пересекаются с $[L, R]$
- Если сыновьям корня соответствуют отрезки $[L, m]$ и $[m + 1, R]$, то в весе дерева отрезков на $[L, R]$ надо учесть вес дерева отрезков на $[L, m]$ и на $[m + 1, R]$.
- При этом запросы целиком содержащие $[L, R]$ не посещают никаких вершин кроме корня, поэтому их нужно вычесть из весов деревьев отрезков на $[L, m]$ и на $[m + 1, R]$.

Введем несколько обозначений:

- $include_{L,R}$ - суммарный вес запросов, которые содержат отрезок $[L, R]$
- $intersect_{L,R}$ - суммарный вес запросов, которые пересекаются с отрезком $[L, R]$
- $dp_{L,R}$ - минимальный вес дерева отрезков, построенного на $[L, R]$.

Тогда из предыдущих фактов следует, что верно

- $dp_{L,L} = intersect_{L,L}$
- $L \neq R \Rightarrow dp_{L,R} = intersect_{L,R} - 2 \cdot include_{L,R} + \min\limits_{L \leqslant m < R} dp_{L,m} + dp_{m+1,R}$

$dp_{L,R}$ зависит только от меньших по длине отрезков, поэтому можно найти минимальный вес всего дерева отрезков, считая $dp_{L,R}$ по возрастанию длины отрезков.

Если запоминать для каждого отрезка, какое m дало минимальную сумму значений у сыновей, то можно и восстановить само дерево.

Без подсчёта $intersect_{L,R}$ и $include_{L,R}$ это решение будет работать за $O(n^3)$ времени и $O(n^2)$ памяти.

Самый простой способ подсчёта этих значений - перебор для каждого отрезка всех запросов. Это работает за $O(n^2 S)$, что при $S = O(n^2)$ превращается в $O(n^4)$.

3.2 Оптимизация до $O(n^3 + S)$ времени

Для того чтобы получить желаемую асимптотику, мы научимся находить все $intersect_{L,R}$ и $include_{L,R}$ за $O(n^2 + S)$ времени суммарно.

Введем новую величину $exact_{L,R}$ - суммарный вес запросов с границами $[L, R]$. Все её значения очевидно ищутся за $O(n^2 + S)$.

Тогда утверждаются следующие факты про $include_{L,R}$:

- $include_{1,n} = exact_{1,n}$
- $R \neq n \Rightarrow include_{1,R} = include_{1,R+1} + exact_{1,R}$
- $L \neq 1 \Rightarrow include_{L,n} = include_{L-1,n} + exact_{L,n}$
- $L \neq 1 \wedge R \neq n \Rightarrow$

Первый пункт очевиден. Второй пункт верен, так как любой запрос, содержащий префикс массива, является этим префиксом или большим. Третий пункт верен, так как любой запрос, содержащий суффикс массива, является этим суффиксом или большим.

В последнем пункте верно, что любой запрос, содержащий $[L, R]$, либо совпадает с этим отрезком, либо содержит отрезок с длиной увеличенной на один. При этом два раза считаются отрезки, содержащие $[L-1, R+1]$, их надо вычесть.

Таким образом, зная $exact_{L,R}$ можно пересчитывать $include_{L,R}$ по убыванию длины отрезка $[L, R]$, и все значения ищутся за $O(n^2)$ времени.

Теперь утверждаются следующие факты про $intersect_{L,R}$:

- $intersect_{L,L} = include_{L,L}$
- $L \neq R \Rightarrow intersect_{L,R} = include_{L,L} + intersect_{L+1,R} - include_{L,L+1}$

Первый факт верен, так как пересечение с отрезком длины один, это то же самое, что и содержание его внутри себя.

Второй факт верен, так как сложив $include_{L,L}$ и $intersect_{L+1,R}$ мы учтём все пересечения, но отрезки, содержащие отрезок $[L, L+1]$ посчитается и там, и там.

Зная $include_{L,R}$ можно пересчитывать $intersect_{L,R}$ по возрастанию длины отрезка $[L, R]$, и все значения ищутся за $O(n^2)$ времени.

Таким образом, мы находим все вспомогательные величины для подсчёта $dp_{L,R}$ за $O(n^2+S)$, что и приводит к решению за $O(n^3+S)$ времени и $O(n^2)$ памяти.

3.3 Предпосылки для существования более оптимального решения

Сейчас наше решение выглядит как подсчёт значений $dp_{L,L}$, где

- $dp_{L,L} = cost_{L,L}$
- $dp_{L,R} = cost_{L,R} + \min\limits_{L \leqslant m < R} dp_{L,m} + dp_{m+1,R}$

Заметим, что суммарная стоимость листьев $cost_{L,L}$ не зависит от формы дерева, поэтому мы можем её поменять произвольным образом, и это не повлияет на оптимальность решения. Поэтому будем считать, что для всех L, R верно $cost_{L,R} = intersect_{L,R} - 2 \cdot include_{L,R}$.

В [4] было показано, что все $dp_{L,R}$ можно найти за $O(n^2)$ времени и памяти, если выполняются два неравенства:

- $\forall a \leqslant b \leqslant c \leqslant d: \ cost_{a,d} \geqslant cost_{b,c}$ (монотонность)
- $\forall a \leqslant b \leqslant c \leqslant d: \ cost_{a,c} + cost_{b,d} \leqslant cost_{a,d} + cost_{b,c}$ (quadrangle неравенство)

Я собираюсь доказать, что в нашем случае первое неравенство верно, а второе верно с противоположным знаком.

Если увеличить отрезок, то количество пересечений с ним не может уменьшиться, а количество запросов, содержащих его, не может увеличиться. Отсюда сразу следует неравенства о монотонности для $intersect_{L,R}$ и $-2 \cdot include_{L,R}$, а значит сложив эти два неравенства, мы получим неравенство о монотонности для $cost_{L,R}$.

Пусть $a \leqslant b \leqslant c \leqslant d$. Рассмотрим как запрос $[L, R]$ может повлиять на левую и правую часть второго неравенства для пересечений, а именно $intersect_{a,c} + intersect_{b,d}$ и $intersect_{a,d} + intersect_{b,c}$.

- $R < a \vee L > d \Rightarrow$ не влияет на обе части
- $a \leqslant R < b \Rightarrow$ запрос пересекается с $[a, c]$ и $[a, d]$, но не с другими двумя отрезками, поэтому учитывается одинаково в левой и правой части
- $c < L \leqslant d \Rightarrow$ запрос пересекается с $[b, d]$ и $[a, d]$, но не с другими двумя отрезками, поэтому учитывается одинаково в левой и правой части
- иначе запрос пересекается с $[b, d]$, а значит и со всеми остальными отрезками, и опять же учитывается одинаково.

Отсюда верно $intersect_{a,c} + intersect_{b,d} = intersect_{a,d} + intersect_{b,c}$.

Пусть снова $a \leqslant b \leqslant c \leqslant d$. Рассмотрим как запрос $[L, R]$ может повлиять на левую и правую часть второго неравенства для включений, а именно $include_{a,c} + include_{b,d}$ и $include_{a,d} + include_{b,c}$.

- Содержит ровно один из отрезков $[a, c]$ и $[b, d] \Rightarrow$ пересекается с $[b, c]$ и не пересекается с $[a, d] \Rightarrow$ учитывается и слева, и справа по одному разу
- Содержит отрезки $[a, c]$ и $[b, d] \Rightarrow$ пересекается со всеми отрезками и учитывается с обеих сторон одинаково
- Не содержит отрезки $[a, c]$ и $[b, d] \Rightarrow$ либо не пересекается ни с одним из отрезков, либо пересекается только с $[b, c]$, что увеличивает только правую часть

Отсюда $include_{a,c} + include_{b,d} \leqslant include_{a,d} + include_{b,c}$. Домножив это неравенство на минус два и прибавив равенство для пересечений, получаем неравенство $cost_{a,c} + cost_{b,d} \geqslant cost_{a,d} + cost_{b,c}$

Итак, мы доказали оба неравенства, которые хотели доказать. Я считаю, что они могут оказаться важными для дальнейшего поиска более оптимального точного решения.

4 Асимптотически оптимальное приближенное решение

4.1 Идея приближения

Сейчас у нас запросы на отрезках и при подсчете их веса берется количество посещенных вершин. Давайте решать похожую задачу. Заменим запрос $[L, R]$ на два запроса $[1, R]$ и $[L, n]$ с весами, равными весу исходного запроса

Будем в такой задаче считать весом дерева отрезков сумму по всем новым запросам веса запроса умножить на максимальную глубину посещенной им вершины. Здесь глубину корня считаем равной единице, а глубина любой другой вершины выражается как глубина её непосредственного предка плюс один.

4.2 Доказательство ухудшения не более чем в константу раз

Посмотрим, какие вершины запрос $[1, R]$ посещает, начиная с корня. Если сейчас он находится в вершине, соответствующей отрезку $[x, R]$, то дальше он не спускается. Пусть иначе правый сын не посещается, тогда запрос спускается в левого сына. Если правый сын посещается, то посещается и левый, но отрезок левого сына обязан лежать целиком в $[1, R]$, поэтому дальше левого сына спуск не идёт, и запрос спускается в правого сына.

В итоге, запрос приходит только в наименее глубокую вершину с правой границей R. Так как на каждом шаге либо запрос посещает одного сына, либо спускается в другого сына максимум на одну вершину, то более глубоких вершин запрос не посетит.

Отсюда следуют два утверждения:

- **Утверждение:** Самая глубокая вершина, которую посещает запрос $[1, R]$ соответствует отрезку $[x, R]$ для некоторого x, причём если таких вершин несколько, то соответствует наименее глубокой из них.

- **Утверждение:** Запрос $[1, R]$ на каждой глубине посещает не более двух вершин.

  Аналогично доказываются симметричные утверждения:

- **Утверждение:** Самая глубокая вершина, которую посещает запрос $[L, n]$ соответствует отрезку $[L, x]$ для некоторого x, причём если таких вершин несколько, то соответствует наименее глубокой из них.

- **Утверждение:** Запрос $[L, n]$ на каждой глубине посещает не более двух вершин.

Давайте смотреть, как соотносится количество посещенных вершин исходного запроса $[L, R]$ и сумма глубин двух новых запросов $[1, R]$ и $[L, n]$.

- **Первый случай** - исходный запрос начинает в корне и затем спускается ровно в одного сына текущей вершины, либо заканчивает работу.

  Заметим, что закончить работу он может только в вершине, которая соответствует отрезку $[L, R]$ Непосредственный предок этой вершины имеет ровно одну отличающуюся границу отрезка. Следовательно, один из запросов $[1, R]$ и $[L, n]$ заканчивается там же, где и исходный запрос, а другой заканчивается раньше. Следовательно, сумма глубин новых запросов больше или равна и не более чем в два раза больше чем количество посещенных вершин исходного запроса

- **Второй случай** - исходный запрос сначала спускается ровно в одного сына текущей вершины, потом в вершине, у которой либо левая, либо правая граница совпадает с соответствующими границами $[L, R]$ переходит в обоих сыновей.

Не теряя общности, что запрос разветвляется в вершине - отрезке $[L, x]$, где $x > R$. Тогда в правом сыне будет отрезок $[m+1, x]$, где $m+1 \leq R$, так как иначе правого сына запрос бы не посетил. Но тогда в левом сыне будет отрезок $[L, m]$, целиком лежащий в $[L, R]$, поэтому в левом сыне будет посещена ровно одна вершина.

В правом сыне $[m+1, x]$ запрос $[L, R]$ будет действовать аналогично запросу $[1, R]$ на всём дереве отрезков. То есть на каждой глубине будет посещено не более двух вершин и самая глубокая вершина совпадает с самой глубокой вершиной для $[1, R]$

Обозначим за d_{lca} глубину вершины, где происходит разветвление, за d_R разность между глубинами самой глубокой вершины $[1, R]$ и d_{lca}.

Глубина запроса $[L, n]$ не превосходит d_{lca}. Глубина запроса $[1, R]$ равна $d_{lca} + d_R$. Сумма этих глубин лежит в отрезке $[d_{lca} + 1 + d_R; 2d_{lca} + d_R]$. Количество посещенных вершин запросом $[L, R]$ лежит в отрезке $[d_{lca} + 1 + d_R; d_{lca} + 1 + 2d_R]$

Следовательно, сумма глубин запросов может не более чем в два раза превосходить количество посещенных вершин и так же может не более чем в два раза уступать количеству посещенных вершин.

- **Третий случай** - исходный запрос сначала спускается ровно в одного сына текущей вершины, потом переходит в обоих сыновей и в этой вершины и левая, и правая граница отличается от соответствующей границы $[L, R]$.

  Пусть в вершине с разветвлением отрезок $[a, b]$. Тогда в левом сыне отрезок $[a, m]$, в правом $[m+1, b]$ для некоторого m.

  Так как $a \neq L, b \neq R$, то дальше запрос на $[a, m]$ ведёт себя так же как

и запрос $[L, n]$, на $[m + 1, R]$ как запрос $[1, R]$

Обозначим за d_{lca} глубину вершины, где происходит разветвление, за d_L разность между глубинами самой глубокой вершины $[L, n]$ и d_{lca}, за d_R разность между глубинами самой глубокой вершины $[1, R]$ и d_{lca}.

Тогда глубина запроса $[L, n]$ равна $d_{lca} + d_L$, запроса $[1, R]$ равна $d_{lca} + d_R$. Сумма этих двух значений $2d_{lca} + d_L + d_R$. Количество посещенных вершин аналогично предыдущему случаю лежит на отрезке $[d_{lca} + (d_L + d_R); d_{lca} + 2(d_L + d_R)]$

Следовательно, сумма глубин запросов может не более чем в два раза превосходить количество посещенных вершин и также может не более чем в два раза уступать количеству посещенных вершин.

После разбора случаев становится ясно, что вес всего дерева отрезков на новых запросах может отличаться не более чем в два раза в обе стороны от веса дерева отрезков на старых запросах. Тогда в худшем случае оптимальное на новых запросах дерево отрезков может быть хуже не более чем в четыре раза на старых запросах, чем соответствующее оптимальное дерево.

4.3 Алгоритм нахождения приближенного решения

Повторим доказанные ранее утверждения

Утверждение: Самая глубокая вершина, которую посещает запрос $[1, R]$ соответствует отрезку $[x, R]$ для некоторого x, причём если таких вершин несколько, то соответствует наименее глубокой из них.

Утверждение: Самая глубокая вершина, которую посещает запрос $[L, n]$ соответствует отрезку $[L, x]$ для некоторого x, причём если таких вершин несколько, то соответствует наименее глубокой из них.

Если мы можем определить, как выбирать самую глубокую вершину, то можно так же и определить, как выбирать путь до этой вершины (не включающий эту вершину).

Пусть в дереве отрезков есть вершина $v_{[L,R]}$, соответствующая отрезку $[L, R]$. Скажем, что стоимость отрезка $[L, R]$ - это сумма по всем новым запросам их веса помножить на количество вершин из пути до самой глубокой вершины, которые лежат в поддереве $v_{[L,R]}$. Тогда $dp_{L,R}$ - это минимальная стоимость отрезка $[L, R]$ среди всех возможных разбиений этого отрезка на поддерево.

Заметим, что итоговая стоимость стоимость отрезка $[1, n]$ отличается от введенного ранее веса дерева отрезков на сумму весов всех запросов (так как не учитывается самая глубокая вершина). Это константа, не зависящая от вида дерева, поэтому неучитывание самой глубокой вершины запроса не влияет на оптимальность.

Введём вспомогательные величины

- lw_i - сумма весов всех запросов $[i, n]$
- rw_i - сумма весов всех запросов $[1, i]$

Теперь можно описать как считать $dp_{L,R}$

- $dp_{L,L} = 0$ (так как только самая глубокая вершина может попасть в лист дерева отрезков, а её мы не учитываем)
- $dp_{L,R} = \sum\limits_{i=L+1}^{R} lw_i + \sum\limits_{i=L}^{R-1} rw_i + \min\limits_{L \leqslant m < R} dp_{L,m} + dp_{m+1,R}$

  Для запросов $[x, n]$, где x не лежит в $[L, R]$ самая глубокая вершина просто не лежит в поддереве $v_{[L,R]}$. Если $x = L$, то лежать в поддереве может только самая глубокая вершина, но не вершины из пути до неё.

Для всех остальных запросов добраться до вершины с левой границей x можно только пройдя через $v_{[L,R]}$, поэтому их веса мы прибавляем. Аналогичны рассуждения про то, какие запросы $[1, x]$ нужно учитывать.

Введём $w_i = rw_i + lw_{i+1}$ для $i \in \{1, \ldots, n-1\}$

Тогда $dp_{L,R} = \sum_{i=L}^{R-1} w_i + \min_{L \leqslant m < R} dp_{L,m} + dp_{m+1,R}$

В чём смысл данного выражения? Все w_i от L до $R-1$ учитываются в $dp_{L,R}$. После этого выбирается оптимальное m такое, что все w_i с $i < m$ идут влево, все с $i > m$ идут вправо, w_m больше нигде не учитывается.

Но тогда $dp_{L,R}$ соответствует весу статически оптимального дерева поиска, где вероятности запроса элемента пропорциональны $w_L, \ldots, w_{R-1}$, а вероятности запросов между двух элементов дерева равна нулю.

Эта задача уже хорошо изучена. Как упомянуто в обзоре литературы, её можно решать точно за $O(n^2)$ и приближенно за $O(n)$.

В итоге получаем решение, которое работает за $O(n+S)$ времени. Ухудшение может вносить как первоначальное приближение, так и приближение нахождения дерева поиска. В итоге решение хуже оптимального не более в чем константу раз, где константу сверху можно оценить как 4 умножить на константу приближения дерева поиска $\approx 4 \cdot 2.29 = 9.16$

4.4 Изучение поведения на реальных тестах

Решение выше асимптотически оптимально, поэтому уже представляет теоретический интерес. Тем не менее, доказанная константа слишком велика для практического применения.

Я написал приближенное решения за $O(n + S)$ (использующее приближенное решения для статически оптимального дерева поиска) и за $O(n^2 + S)$ (использующее точное решение) и сравнивал с оптимальным результатом, по-

лученным с помощью точного решения за $O(n^3 + S)$.

Я рассматривал несколько вариантов с n от единицы до сотни и количеством отрезков S от меньше десятка до n^2. Для фиксированных n и S я прогонял большое количество тестов, генерируя границы запросов и их веса равномерно случайно.

$n > 100$ не рассматривались, так как

- На каждый тест запускалось точное решение за $O(n^3 + S)$, при этом для большей достоверности хотелось бы запустить как можно больше тестов.

- Есть подозрение, что среди больших случайных тестов вероятность плохого теста для приближенного решения крайне мала.

На всех таких случайных тестах я получил такие результаты:

- Решение за $O(n^2 + S)$ находит дерево отрезков с весом хуже не более чем в ≈ 1.42 раз чем оптимальное

- Решение за $O(n+S)$ находит дерево отрезков с весом хуже не более чем в ≈ 2.3 раз чем оптимальное

Это гораздо лучше чем доказанные константы и выглядит уже как что-то, что можно применить на практике. В частности константа для решения за $O(n+S)$ близка к константе у приближенного алгоритма для нахождения статически оптимального дерева поиска.

5 Заключение

Было найдено точное решение задачи за $O(n^3 + S)$ и асимптотически оптимальное приближенное решение. Также были доказаны два неравенства, почти совпадающие с теми, которые используются для доказательства существования решения за $O(n^2)$ для класса задач.

Все найденные решения были написаны и проверены на достаточно небольших случайных тестах, на которых у приближенного решения константа ухудшения ответа по сравнению с оптимальным оказалась достаточно невелика, чтобы данное решение могло иметь не только теоретический, но и практический интерес.

Возможные улучшения - доказательство лучшей константы у приближенного решения и нахождение еще более оптимального точного решения.

Список литературы

[1] Knuth, Donald E, *"Optimum binary search trees"*. Acta Informatica, 1971.

[2] Mehlhorn, Kurt, *"Nearly optimal binary search trees"*. Acta Informatica, 1975.

[3] Garsia, Adriano M, Wachs, Michelle L, *new algorithm for minimum cost binary trees"*. SIAM Journal on Computing, 1977.

[4] F. Frances Yao, *"Efficient Dynamic Programming Using Quadrangle Inequalities"*. STOC, 1980.