# Отчёт по НИР

## Статически оптимальное дерево отрезков

Артем Комендантян, 4 курс МФТИ, кафедра анализа данных

**Научный руководитель:** Михаил Тихомиров, к.ф.-м.н.

# Введение

Дерево отрезков - это мощная структура данных, которая позволяет решать многие задачи. Например такие как нахождение наименьшего общего предка двух вершин в дереве и нахождение площади объединения прямоугольников со сторонами, параллельными осям координат.

Результаты, полученные в этом дипломе, потенциально могут привести к ускорению на реальных задачах.

Также стоит заметить, что аналогичная задача для некоторых других структур данных хорошо изучена.

# Определения

Пусть дан массив из n элементов

**Дерево отрезков** - это двоичное дерево, в котором есть

- $n$ листьев, соответствующих отрезкам единичной длины
- 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.

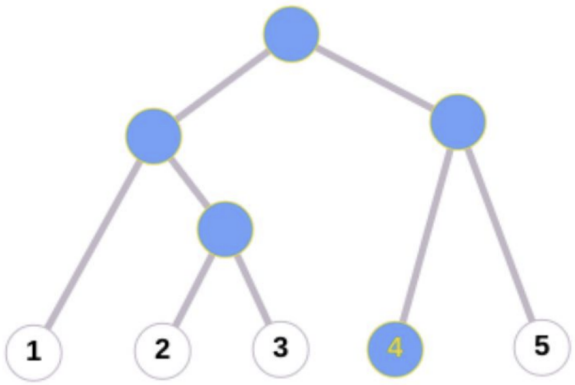
Если сейчас рассматривается вершина, отрезок которой не лежит полностью в отрезке $[L, R]$, то запрос рекурсивно вызывается от тех сыновей, отрезки которых пересекаются с $[L, R]$. Иначе рекурсивных вызовов от сыновей не происходит.

В обоих случаях вершина считается посещенной и в ней выполняются какие-то действия, специфичные для запроса.

### Пример

Дерево отрезков на пяти элементах. Запрос на отрезке $[2; 4]$.

Будет посещено пять выделенных вершин.




# Постановка задачи

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

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

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

# Обзор литературы

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:

```
def find(node, x):
    if node is None:
```

```
        return

    if node.value < x:
        find(node.left, x)
    elif node.value > x:
        find(node.right, x)
```

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

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

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

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.

Было показано, что $0.63H \le P_{opt} \le P_{approx} \le 2 + 1.44H$, 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 обобщение того, когда точные решения за $O(n^3)$ времени могут быть соптимизированы до $O(n^2)$

В статье [4] было приведено точное решение за $O(n \log n)$ времени в случае, когда вероятности $A_i$ равны нулю, я не нашел информации про точные решения оптимальнее чем $O(n^2)$ для других случаев.

# Решение задачи

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

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

- $exact_{L,R}$ - суммарный вес запросов с границами $[L, R]$
- $include_{L,R}$ - 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

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

$dp_{L,R}$ 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 какой $m$ 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 $O(n^4)$. Приведем теперь способ их подсчёта за $O(n^2 + S)$.

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

$$include_{L,R} = exact_{L,R} + include_{L-1,R} + include_{L,R+1} - include_{L-1,R+1}$$

В первых трёх пунктах просто суммируются веса всех запросов, которые содержат соответствующий отрезок. В последнем пункте любой запрос, содержащий $[L, R]$ либо совпадает с этим отрезком, либо содержит отрезок с длиной увеличенной на 1. При этом два раза считаются отрезки, содержащие $[L-1, R+1]$, их надо вычесть.

Здесь любой $include_{L,R}$ зависит только от $exact_{L,R}$ (подсчитывается тривиально за $O(n^2 + S)$), либо от $include$ с большей длиной отрезка. Так что, считая по убыванию длины отрезка можно найти $include_{L,R}$ за $O(n^2 + S)$

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

Пересечение с отрезком длины 1 - это тоже самое, что его содержание внутри себя. Если запрос пересекается с $[L, R]$, то он либо содержит внутри себя $[L, L]$, либо пересекается с $[L+1, R]$. При этом два раза учитываются отрезки, содержащие $[L, L+1]$, поэтому их надо вычесть. Если перебирать отрезки по возрастанию длины, то $intersect_{L,R}$ можно подсчитать за $O(n^2)$.

В итоге получается точное решение за $O(n^3 + S)$ времени и $O(n^2)$ памяти.

### Дальнейшие идеи для более оптимального точного решения

Так как для нахождения оптимального дерева поиска есть оптимизация решения за куб до $O(n^2)$, то хочется найти решение быстрее чем за $O(n^3 + S)$ и для моей задачи. В [3] было доказано, что точно можно оптимизировать до $O(n^2)$, если

- $dp_{L,L} = cost_{L,L}$
- $dp_{L,R} = cost_{L,R} + \min\limits_{L \le m < R} dp_{L,m} + dp_{m+1,R}$
- $\forall a \le b \le c \le d: \ cost_{a,d} \ge cost_{b,c}$ (монотонность)
- $\forall a \le b \le c \le d: \ cost_{a,\mathrm{c}} + cost_{b,d} \le cost_{a,d} + cost_{b,c}$ (quadrangle неравенство)

Если в точном решении для нашей задачи заменить $cost_{L,L}$ с $intersect_{L,L}$ до $intersect_{L,L} - 2 \cdot include_{L,L}$, то оптимальное решение не поменяется (сумма $cost_{L,L}$ в листьях учитывается одинаково для любого решения), но при этом можно доказать, что выполняются следующие свойства:

- $\forall a \le b \le c \le d: \ cost_{a,d} \ge cost_{b,c}$
- $\forall a \le b \le c \le d: \ cost_{a,\mathrm{c}} + cost_{b,d} \ge cost_{a,d} + cost_{b,c}$

Я пока что не выяснил, можно ли что-то получить из комбинации этих двух свойств.

Еще довольно много вариантов оптимизаций с ссылками на научные статьи есть в `https://codeforces.com/blog/entry/8219` , но актуальных для моей задачи я не нашёл.

## Асимптотически оптимальное приближенное решение

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.

## Доказательство ухудшения не более чем в константу раз при приближении

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 в 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 $[L, n]$ равно $d_{lca} + d_L$, запроса $[1, R]$ равно $d_{lca} + d_R$. 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 в более чем в два раза превосходить количество посещенных вершин и так же 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.

## 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 не учитывание 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)
- $dp_{L,R} = \sum\limits_{i=L+1}^{R} lw_i + \sum\limits_{i=L}^{R-1} rw_i + \min\limits_{L \le m < R} dp_{L,m} + dp_{m+1,R}$

  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 $w_i = rw_i + I\{i > 0\} \cdot lw_{i-1}$ для $i \in \{0, \dots, n-1\}$

Тогда $dp_{L,R} = \sum\limits_{i=L}^{R-1} w_i + \min\limits_{L \le m < R} dp_{L,m} + dp_{m+1,R}$

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 $\approx 4 \cdot 2.29 = 9.16$

### Дальнейшие идеи по приближенному решению

Данное решение идеально с точки зрения времени нахождения, но при этом константа кажется слишком большой для реальных применений. Тем не менее я решил написать код точного решения за $O(n^3)$ и код приближенного решения и сравнить насколько приближенное решение хуже.

Я генерировал много тестов с различными $n < 100$ и различным количеством отрезков (от меньше десятка до примерно $n^2$). Координаты отрезков и их веса я брал случайно.

В результате на всех таких тестах:

- Решение, которое берет первоначальное приближение, а затем находит точное оптимальное дерево поиска, находит дерево отрезков с весом хуже не более чем в $\approx 1.42$ раз чем оптимальное
- Решение, которое находит оптимальное дерево поиска тоже приближенно, находит дерево отрезков с весом хуже не более чем в $\approx 2.3$ раз чем оптимальное

Получается, что на практике для небольших случаев приближенное решение для дерева отрезков сравнимо по константе ухудшения с приближенным решением для дерева поиска. Если это правда и в теории, то я считаю это очень хорошим результатом.

Дальнейший план - попытаться доказать константу лучше, чем доказанная сейчас.

# Список литературы


1. Knuth, Donald E. (1971), "Optimum binary search trees", Acta Informatica. URL: `https://doi.org/10.1007%2FBF00264289`
2. Mehlhorn, Kurt (1975), "Nearly optimal binary search trees", Acta Informatica. URL: `https://people.mpi-inf.mpg.de/~mehlhorn/ftp/mehlhorn3.pdf`
3. F. Frances Yao (1980), "Efficient Dynamic Programming Using Quadrangle Inequalities". URL: `https://dl.acm.org/doi/10.1145/800141.804691`

4. Garsia, Adriano M.; Wachs, Michelle L. (1977), "A new algorithm for minimum cost binary trees", SIAM Journal on Computing, URL: https://doi.org/10.1137/0206045