Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибирнетики
Кафедра автоматизации систем вычислительных комплексов

Курсовая работа

Анализ и исследование структур данных для поиска в таблицах классификации в архитектуре сетевого процессора без выделенного ассоциативного устройства

Никифоров Никита Игоревич, 321 группа
Научные руководители: Волканов Д. Ю., Скобцова Ю. А

Москва
2019

Аннтотация

В данной курсовой работе рассматривается проблема классификации пакетов в рамках архитектуры СПУ. В качестве критериев выбора структур данных были минимизация используемой памяти и минимизация времени классификации. Был сделан обзор существующих структур данных, на его основе была выбрана `такая-то` структура данных. Выбранная структура данных была реализована на эмуляторе архитектуры СПУ.

Оглавление

Введение

В настоящее время ведётся активная разработка ПКС сетей, в которых требутся высокопроизводительных коммутаторов. Возникает задача разработки СПУ, являющегося основным элементом коммутаторов. СПУ - сетевое процессорное устройство представляет из себя интегральную микросхему, выполняющую следующие функции:

1. Получение пакетов с физического порта
2. Выделение заголовка
3. Классификация пакета по его заголовку
4. Принятие решение о дальнейшем пути следования пакета
5. Управление очередями
6. Отправка пакета на физический порт

Заголовок пакета представляет из себя битовую строку, содержащую в себе признаки пакета, которые также являются битовыми строками. Для классификации пакетов используются таблицы классификации – набор правил содержащих в себе признаки и действия над пакетом. Определения соответствия пакета правилам производится по выделенным из заголовка признакам, данный процесс называется классификацией. Разрабатываются соответствующие архитектуры СПУ, в которых в основновном используется ассоциативное устройство памяти для хранения таблиц классификации, его использование оправданно скоростью, с которой оно позволяет выполнять классификацию пакетов. Так же ведутся разработки архитектуры СПУ без использования ассоциативного устройства памяти. В данной работе рассматривалась архитектура без выделенного ассоциативного устройства. Существуют различные виды и реализации структур данных для выполнения классификации пакетов[1], но так как в работе рассматривается конкретная архитектура, невозможно применить существующие реализации напрямую. Поэтому в данной работе стояла задача разработать структуру данных непосредственно для данной архитектуры СПУ. Для выполнения этой задачи был проведён обзор структур данных и....

Глава 1

Цели и задачи работы

Целью данной работы является исследование и разработка структур данных и алгоритм для поиска в рамках архитектуры СПУ без выделенного ассоциативного устройства. Для достижения поставленной цели необходимо было выполнить следующие задачи:

1. Провести обзор структур данных для оптимальному поиску в таблицах классификации пакетов с целью выбора для применения в рассматриваемой архитектуре процессора.
2. Реализовать выбранные структуры данных и алгоритмы поиска в эмуляторе сетевого процессора.
3. Провести экспериментальное исследование реализованных структур данных.

Глава 2

Рассматриваемая архитектура СПУ

Из особенностей рассматриваемой архитектуры можно выделеть отсутствие ассоциативного устройства памяти и конвейерную архитектуру. Они непосредственно влияют на ограничения предъявляемые к реализуемой структуре данных. В СПУ используется конвейерная архитектура, каждый конвейер состоит из 10 вычислительных блоков. Вычислительный блок - это набор более низкоуровневых устройств, которые в данной работе не рассматриваются. Каждый вычеслительный блок имеет доступ к участку памяти в котором располагаются микрокод и данные. Из-за особенностей микроархитектуры, отсутствует отдельная область памяти, в которой хранятся данные. Поэтому микрокод содержит в себе все данные, необходимые для классификации пакетов. Для достяжения требуемой производительности в архитектуре

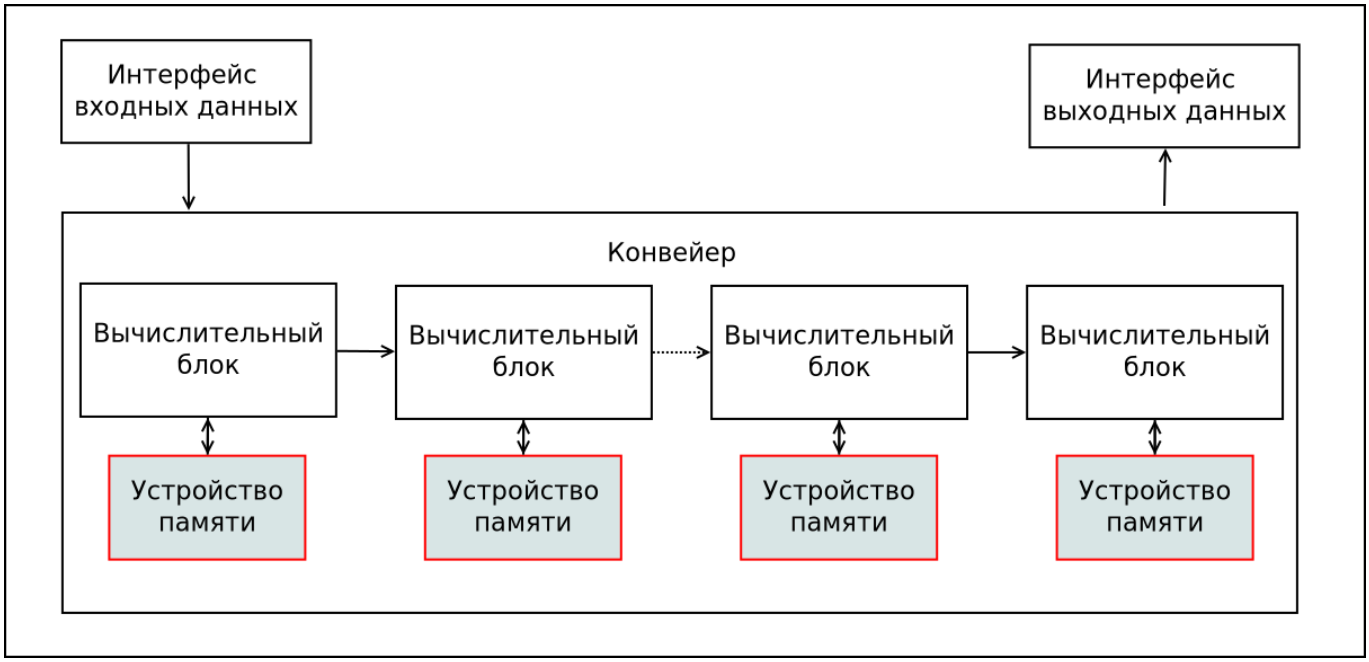

Рис. 2.1: Архитектура рассматриваемого СПУ

используется ограничение на количество тактов обработки пакета на одном вычислительном блоке. Каждый пакет может обрабатываться на одном вычислительном блоке не более чем 25 тактов.

Глава 3

Постановка задачи

Разработать структуру данных для классификации пакетов в рамках рассматриваемой архитектуры СПУ без выделенного ассоциативного устройства. Из-за ограничений связанных с рассматриваемой архитектурой СПУ к реализуемой структуре данных предъявляются следующие требования:

- Объём памяти занимаемый структурой данной данных не должен превышать 2 мегабайт, это обусловлено ограничением количества памяти внутри одного вычислительного блока
- Количество тактов затраченных на поиск не должно превышать 250 тактов, данное требование обусловлено ограничением на количество тактов исполняемой программы внутри вычислительного блока.
- Реализуемая структура данных должна быть универсальна.

Глава 4

Обзор структур данных для классификация пакетов в рамках рассматриваемой архитектуре СПУ

Целью данного обзора является сравнение и выбор структур данных для поиска в таблицах классификации. Для достижения поставленной цели, необходимо было выполнить следующие задачи:

В силу особенностей архитектуры СПУ, а именно отсутствие адресуемой памяти, которая требуется для не древовидных структур данных будут рассматриваться только древовидные структуры данных. Под памятью занимаемой структурой данных понимается `N * 128 = memmory` , где `N` - количество инструкций в конкретной реализации рассматриваемой структуры данных, 128 - бит на одну инструкцию.

В настоящем обзоре использовались следующие критерии:

1. Асимтотическая сложность поиска - позволяет оценить использование ресурсов СПУ, для поиска в структуре данных.
2. Универсальность структуры данных - используемая структура данных должна поддерживать поиск произвольных битовых строк не превосходящих по длине 128 бит.
3. Дополнительные данные необходимые для построения структуры данных.
4. Необходимость использования адресуемой памяти - некоторым рассматриваемым структурам данных требуется адресуемая память для их реализации, соответствующей их асимптотической сложности.
5. Количество вершин, которое необходимо посетить для поиска в худшем случае.
6. Возможность оптимизации алгоритма поиска с помощью инструкций микроархитектуры ядерконвейера СПУ.
7. Необходимость изменения микроархитектуры ядер конвейера СПУ.
8. Оценка памяти занимаемой структурой данных - рассматривается структура данных для 10000 вождений префиксов IPv4.
9. Сложность добавления и удаления элементов из рассматриваемой структуры данных.

Сравнение структур данных будет провоодиться по критериям 1, 2, 4, 5, 8.

4.1 Структуры данных для классификации пакетов в рамках рассматриваемой архитектуры СПУ

4.1.1 Двоичное однобитное дерево

Наиболее распространенная структура данных для LPM. Строится дерево по заданным префиксам, так что каждому биту префикса соответствует своя вершина в дереве. Поиск осуществляется спуском в глубину по битам элемента, для которого выполняется поиск. Поиск заканчивается тогда, когда достигнута пустая вершина, результатом поиска считается последний встретившийся префикс. На Рис. 1 изображён пример данного дерева для набора префиксов.

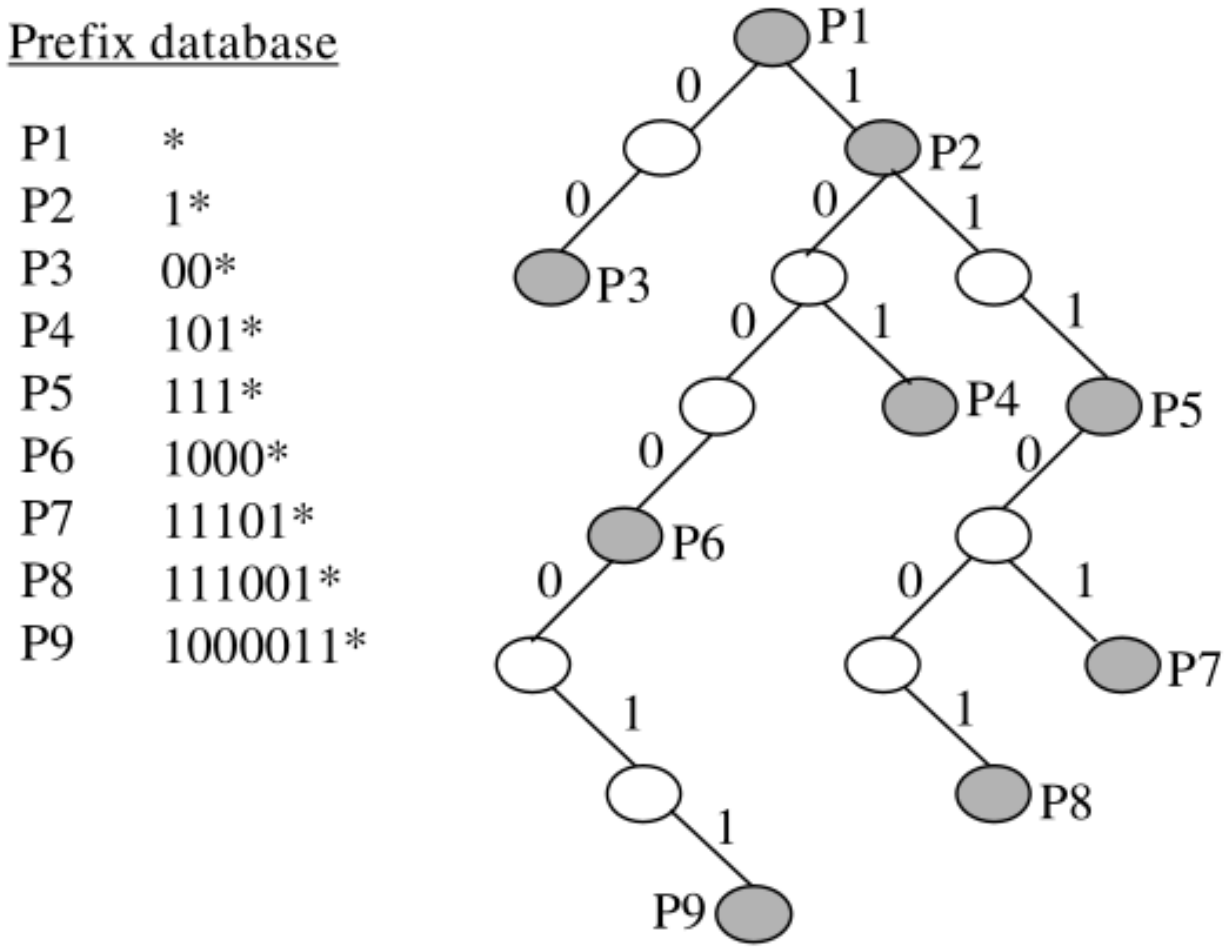

Рис. 4.1: Пример построения двоичного однобитного дерева

Асимптотическая сложность поиска для этой структуры данных соответствует $O(W)$, где `W` - максимальная длина префикса.
Универсальность структуры данных - рассматриваемое дерево может быть использовано для любых данных представимых в битовом виде.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 128. `(Коментарий: тут вообще имеется в виду худший случай, т.е. кода мы ищем префикс IPv6 длиной 128 бит. При этом он есть в дереве и есть все префиксы до него.)`
Оптимизацию агоритма поиска проводить не требуется.
Изменение микроархитектуры ядер конвейера СПУ не требуется для реализации данного алгоритма.

Оценка памяти занимаемой структурой данных - Будем считать, что на каждую вершину используется не более 5 инструкций, тогда рассматриваемая структура данных занимает не более чем 15000 Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных имеет асимптотическую сложность $O(\log_2 W)$, где W - максимальная длина префикса и не требует перестройки всей структуры данных.

4.1.2 Сжатое двоичное дерево

Рассматриваемая структура данных – пришедшая со временем оптимизация двоичного однобитного дерево. Сначала строится двоичное однобитное дерево. далее оно сжимается, то есть все вершины у которых только один лист сокращаются, и в следующую вершину заносятся данные о количестве пропущенных вершин. Таким образом построенное дерево не имеет вершин с одним листом. Благодаря описанной оптимизации данное дерево занимает гораздо меньше памяти. Это обусловлено отсутствием проходных вершин. Однако, уменьшает количество затраченных инструкций на поиск лишь для префиксов, перед которыми были однолистные вершины. Для худшего случая, когда для префикса есть все его более короткие версии, количество вершин, которые нужно посетить для поиска анологично двоичному однобитному дереву.

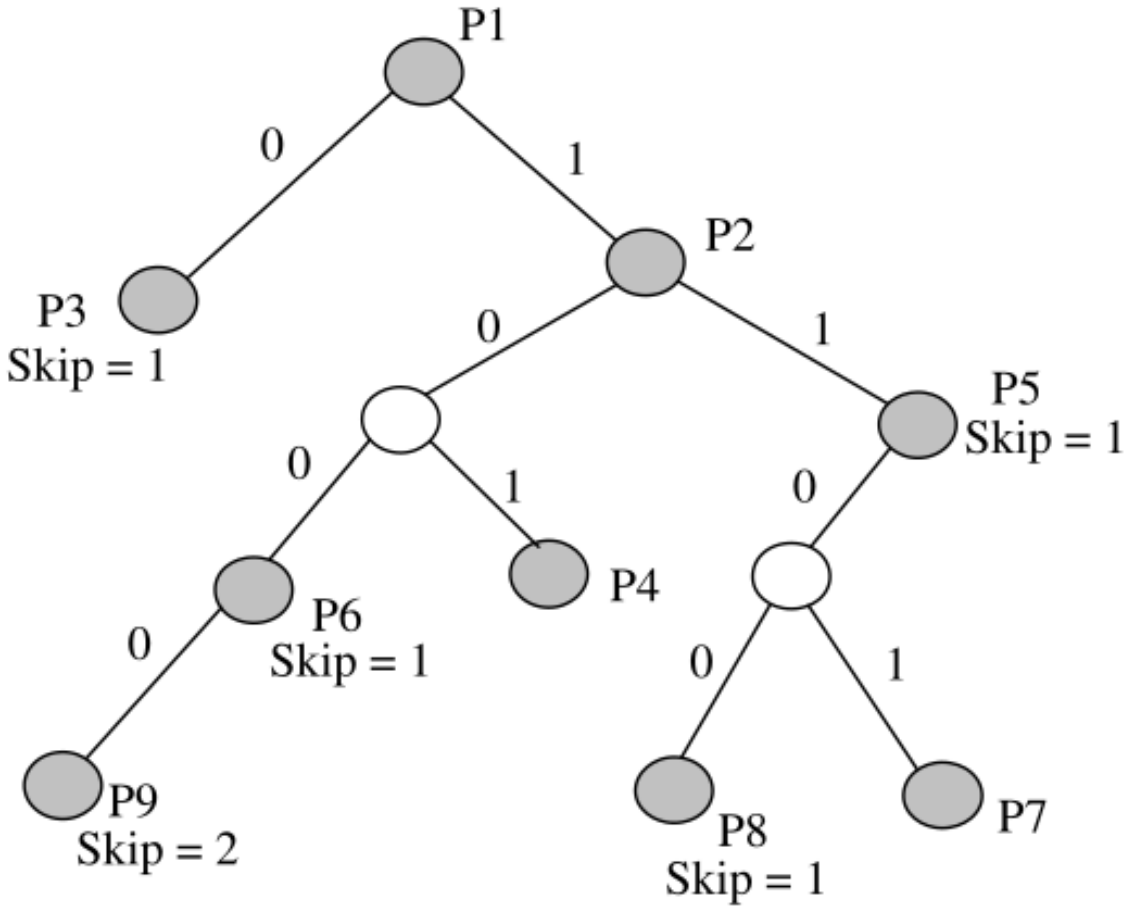

Рис. 4.2: Пример построения сжатого двоичного дерева

Асимптотическая сложность поиска для этой структуры данных соответствует $O(W)$, где W - максимальная длина префикса.
Универсальность структуры данных - рассматриваемое дерево может быть использовано для

любых данных представимых в битовом виде.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 128.
Оптимизацию агоритма поиска проводить не требуется.
Изменение микроархитектуры ядер конвейера СПУ не требуется для реализации данного алгоритма.
Оценка памяти занимаемой структурой данных - рассматриваемая структура данных занимает не более чем 8000Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных имеет асимптотическую сложность $O(\log_2 W)$, где `W` - максимальная длина префикса и не требует перестройки всей структуры данных.

4.1.3 Мультибитное сжатое дерево

Данное дерево – оптимизация двоичного сжатого дерева. Используется другая стуктура деревьев, когда в каждой вершине может быть максимум не два листа, а 2^h, где `h` – это максимальная глубина поддерева данной вершины. При использовании рассматриваемой структуры данных в рамках архитектуры процессора общего назначение, количество операций для поиска ограничевается глубиной дерева, которая равна $\frac{W}{K}$, где `W` – длина максимального префикса, а `K` – количество уровней в нашем дереве. В рамках архитектуры СПУ будет рассматриваться реализация данного дерева, в которой используется линейный поиск в каждой вершине по дочерним вершинам.

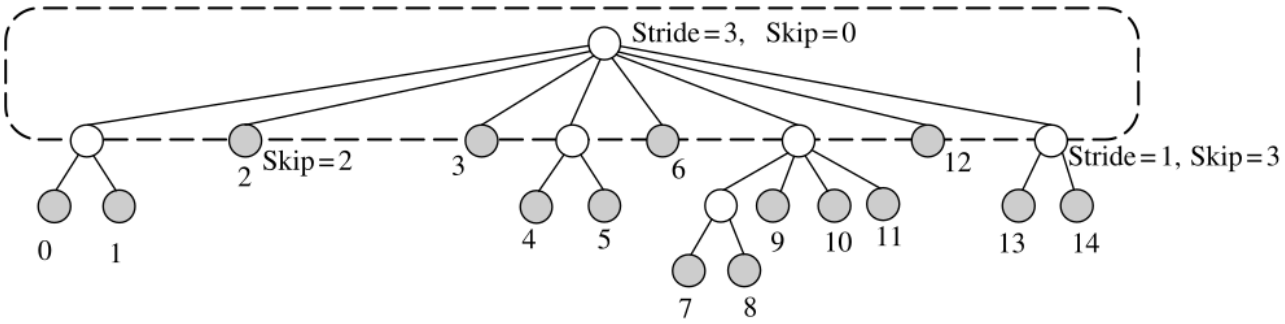

Рис. 4.3: Пример построения мультибитного сжатого дерева

Асимптотическая сложность поиска для этой структуры данных соответствует $O(\frac{W}{K} * N)$, где `K` - количество префиксов в рассматриваемой структуре данных, а `W` - максимальная длина префикса и `N` – количество листьев в вершине.
Универсальность структуры данных - рассматриваемое дерево может быть использовано для любых данных представимых в битовом виде.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 30. `Коментарий: Тут надо как-то сказать, что вершин то мы посещяем мало, а вот инструкций на каждую вершину много`
Оптимизацию агоритма поиска нужна, для организации бастрого доступа к листьям внутри одной вершины.
Изменение микроархитектуры ядер конвейера СПУ для данной реализации не требуется.
Оценка памяти занимаемой структурой данных - Будем считать, что каждая вершина занимает 15 инструкций в среднем, из-за линейного поиска по листьям. Тогда рассматриваемая структура данных занимает не более чем 18000 Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных имеет асимптотическую сложность $O(\log_2 W)$, где `W` - максимальная длина префикса и не требует перестройки

всей структуры данных.

При реализации микроинструкции, которая по заданной метке перехода и маске переходит на инструкцию находящуюся на метке + значение регистра с применённой маской, есть возможность реализовать структуру данных, в которой алгоритм поиска будет быстрее. Рассмотрим эту структуру данных в рамках архитектурвы с добавлением указанной микроинструкции.

Асимптотическая сложность поиска для этой структуры данных соответствует $O(\frac{W}{K})$,
где K - количество префиксов в рассматриваемой структуре данных, а W - максимальная длина префикса. Универсальность структуры данных - рассматриваемое дерево может быть использовано для любых данных представимых в битовом виде.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 30.
Оптимизацию агоритма поиска нужна, для организации бастрого доступа к листьям внутри одной вершины.
Изменение микроархитектуры ядер конвейера СПУ требуется аппаратная реализации быстрого перехода внутри одной вершины. А именно необходима инструкция, которая по заданной метке перехода и маске переходит на инструкцию находящуюся на метке + значение регистра с применённой маской.
Оценка памяти занимаемой структурой данных - Будем считать, что каждая вершина занимает 5 инструкций, благодаря новой микроинструкции. Тогда рассматриваемая структура данных занимает не более чем 7000 Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных имеет асимптотическую сложность $O(\log_2 W)$, где W - максимальная длина префикса и не требует перестройки всей структуры данных.

4.1.4 Бинарный поиск по длинам префиксов

Структура данных основана на построении специальных таблиц для префиксов определённой длины. Пусть максимальная длина префикса W, тогда строятся таблицы $h_1 ... h_w$. В каждой из них хранятся префиксы длины соответствующие номеру этой таблицы. Предполагается, что в каждой такой таблице реализована своя хеш-функция, которая быстро позволяет найти вхождение префикса в данную таблицу. Таким образом мы можем выполнить бинарный поиск по длине префиксов. В рамках рассматриваемой архитектуры СПУ, реализация таких таблиц возможна только с использованием древовидных структур данных.

Асимптотическая сложность поиска для этой структуры данных соответствует $O(K * \log_2 N)$,
где K - количество префиксов в рассматриваемой структуре данных, а W - максимальная длина префикса. и N – количество префиксов в таблице.
Универсальность структуры данных - рассматриваемая структура данных может быть использовано только для поиска префиксов.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 128.
Оптимизацию агоритма поиска требуется, а именно необходима для реализации поиска внутри таблицы.
Изменение микроархитектуры ядер конвейера СПУ требуется аппаратная реализация хеш-функций.
Оценка памяти занимаемой структурой данных - рассматриваемая структура данных занимает не более чем `TODO: рассчитать` Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных требует перестройки всей структуры данных.

Example prefixes
90/8
90.1/16
90.1.10/24
90.1.20/24
90.2.30/24

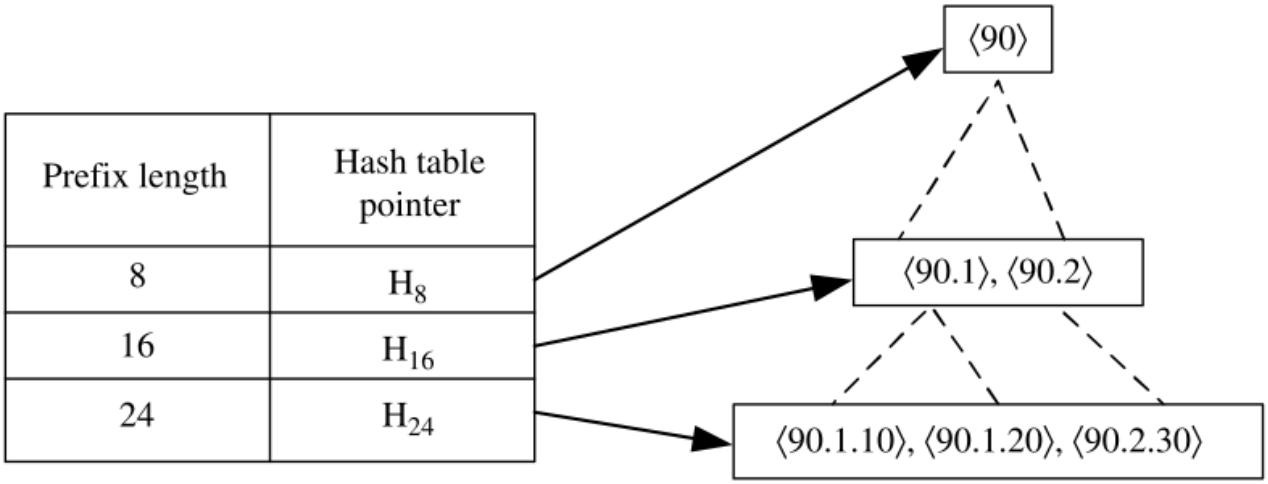

Рис. 4.4: Пример бинарного поиска по длинам префиксов

4.1.5 АВЛ дерево

Представление префиксов как скалярных префиксов позволяет использовать больший набор структур данных. В качестве примера рассмотрим AVL дерево, основной особенностью которого является правило его построения: у каждой вершины разность глубины левого и правого поддерева не превосходит 1, что даёт ассимптотическую сложность поиска $O(1 + \log_2 N)$, где `N` – количество префиксов в нашей структуре данных. Из этого следует, что время поиска не зависит от длины искомых данных, а значит с помощью данной структуры данных очень эффективно выполнять поиск префиксов IPv6.

Асимптотическая сложность поиска для этой структуры данных соответствует $O(1 + \log_2 N)$, где `N` – количество префиксов в таблице.
Универсальность структуры данных - рассматриваемое дерево может быть использовано для любых данных представимых в битовом виде.
Дополнительные данные для построения структуры данных не требуются.
Количество вершин которое необходимо посетить для поиска не превосходит 20.
Оптимизацию агоритма поиска проводить не требуется.
Изменение микроархитектуры ядер конвейера СПУ не требуется.
Оценка памяти занимаемой структурой данных - рассматриваемая структура данных занимает не более чем `TODO: рассчитать` Кбайт.
Добавление и удаление элементов из рассматриваемой структуры данных имеет логарифмическую сложность, и не требует перестройки всей структуры данных.

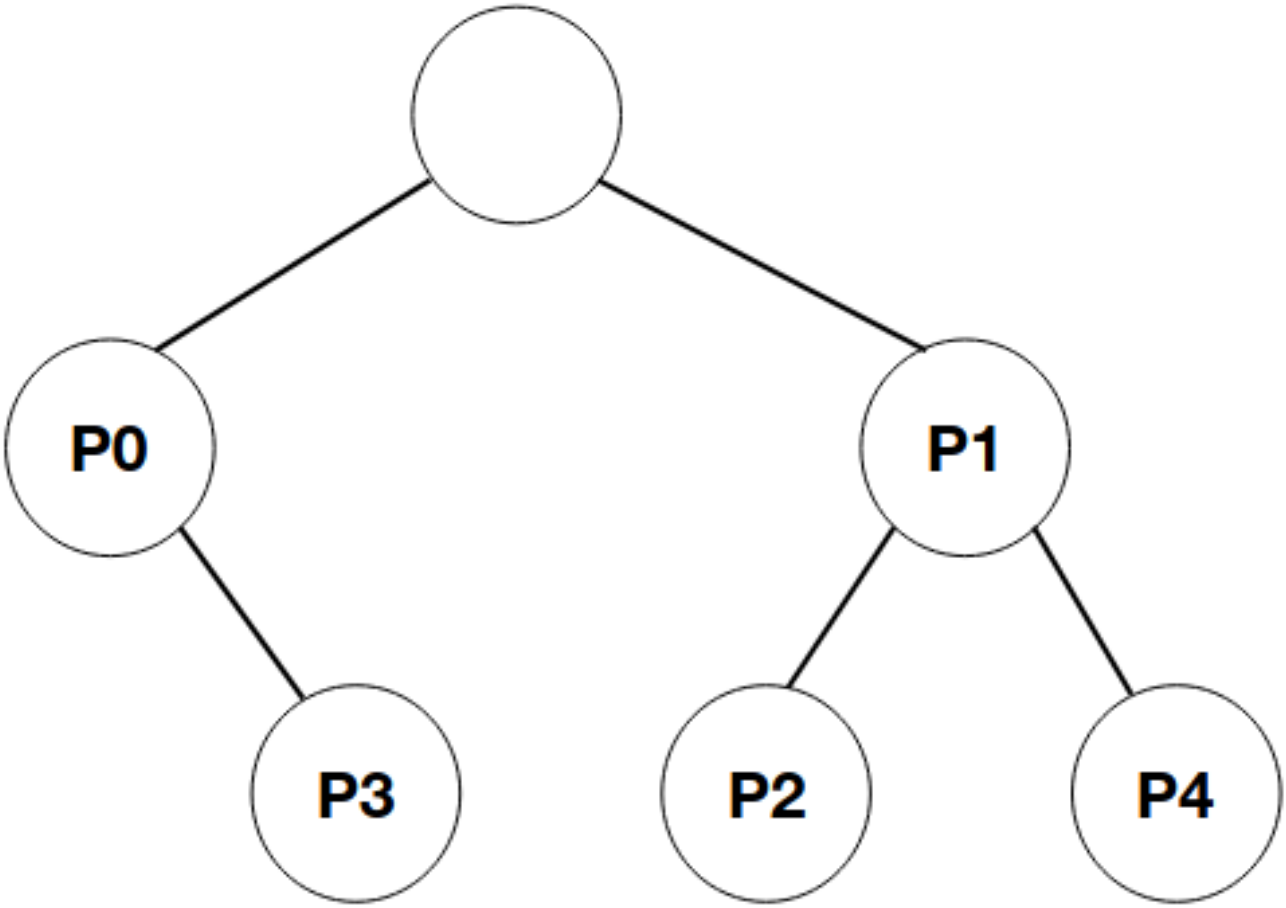

Рис. 4.5: Пример построения АВЛ дерева

4.2 Сравнение структур данных

Для сравнения структур данных рассмотрим Таблицу 2.1 сравнения по критериям: ассимптотическая сложность, универсальность структуры данных, количество вершин, которое необходимо поситить для поиска и оценка памяти занимаемой структурой данных. У каждой рассмотренной структуры данных есть свои достоинства и недостатки, рассмотрим их:

1. Binary one bit tree данная структура проста в реализации, но занимает много памяти и поиск требует прохождения 128 вершин.

2. Path Compressed trie занимает меньше памяти, чем Binary one bit tree, но поиск требует прохождения 128 вершин. Соответственно использование данного дерева предпочтительнее, чем Binary one bit tree.

3. Level Compressed trie занимает много памяти, но поиск требует прохождения сильно меньшего количества вершин. Из-за проблем с реализацией в рамках рассматриваемой

Сравнение структур данных				
Название структуры данных	**Сложность поиска**	**Универсальность**	**Количество вершин необходимых для поиска**	**Память, Кбайт**
Двоичное однобитное дерево	$O(W)$	да	128	15000
Двоичное сжатое дерево	$O(W)$	да	128	8000
Мультибитно сжатое дерево	$O(\frac{W}{K} * L)$	да	32	18000
Бинарный поиск по длинам префиксов	$O(K * \log_2 N)$	нет	128	TODO
АВЛ дерево	$O(1 + \log_2 N)$	да	20	TODO

Таблица 4.1: Сравнение структур данных
W - максимальная длина префикса, N - количество префиксов в структуре данных, L - количество дочерних вершин, K - глубина дерева, T - количество таблиц

архитектуры, эта структура данных не может быть реализованна.

4. Binary search on prefix length занимает много памяти, и может быть использовано только для LPM. Также из-за проблем с реализацией на рассматриваемой архитектуре, данная структура не подходит для решения проблемы.

5. AVL tree занимает столько же памяти, как и Path Compressed trie, но при этом, поиск требует прохождения малого количества вершин, которое зависит от количества вхождений в структуру данных, а не от конкретного префикса.

4.3 Выводы

TODO: написать выводы, сделаю это позже, потому что пока не могу полностью оценить рассматриваемые структуры данных.

Список литературы

[1] H Jonathan Chao и Bin Liu. *High performance switches and routers*. John Wiley & Sons, 2007.