Московский физико - технический институт
(государственный университет)
Факультет радиотехники и кибернетики
Кафедра информатики и вычислительной техники

Выпускная квалификационная работа бакалавра

Оптимизация конструкции switch в промышленном компиляторе

Выполнил: Крупник Е.С., 313 группа

Научный руководитель: к.ф.-м.н., Нейман-Заде М.И.

Конструкция switch

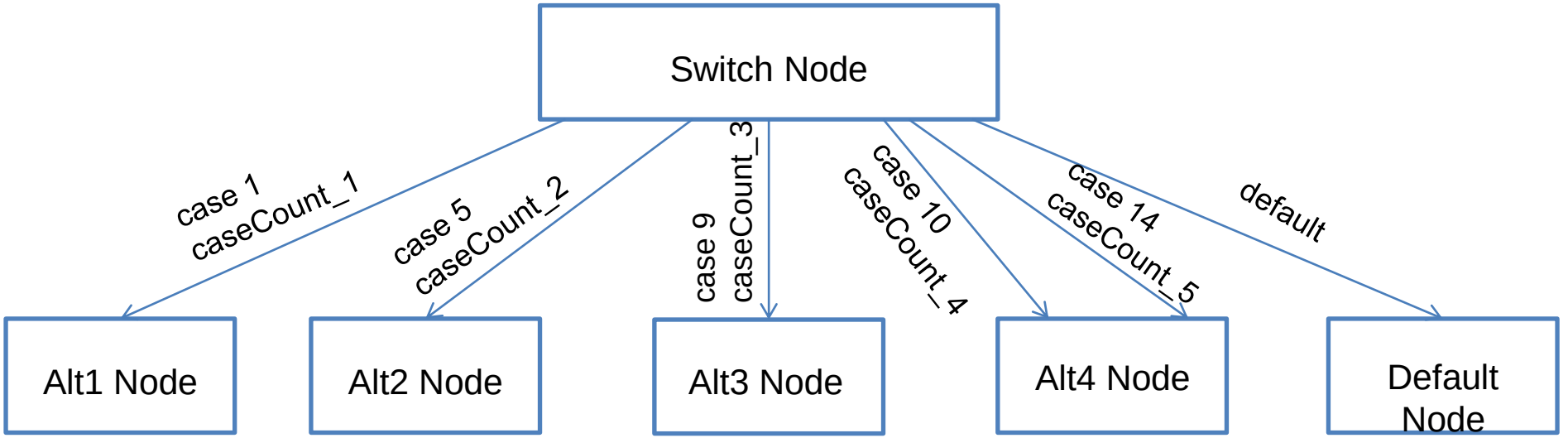
Пример конструкции switch в языке С

```
switch (a)// выражение a называется селектором
{
        case 1:// последовательность операторов 1
                break;
        case 5: // последовательность операторов 2
                break;
        case 9: // последовательность операторов 3
                break;
        case 10:
        case 14:// последовательность операторов 4
                break;
        default : // последовательность операторов default
}
```

Альтернативы

числа 1, 5, 9, 10, 14 называются case-константами

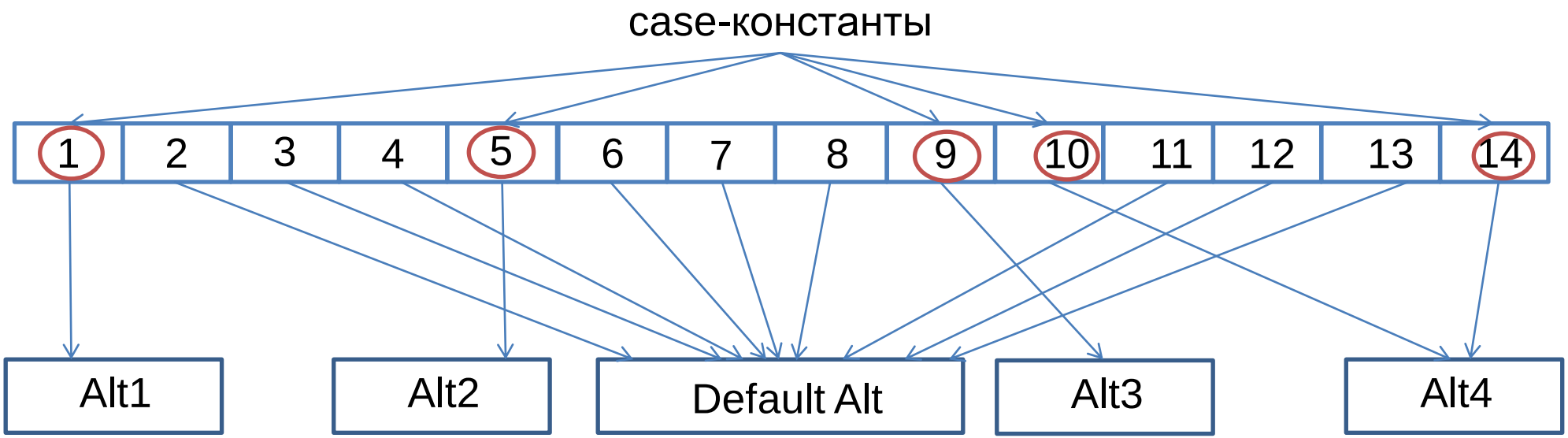
Схема switch в CFG(Control Flow Graph)



Реализации switch в компиляторе для микропроцессоров «Эльбрус»

Таблица поиска

Таблица поиска – массив, где индекс элемента соответствует значению case-константы, а его значение это адрес перехода

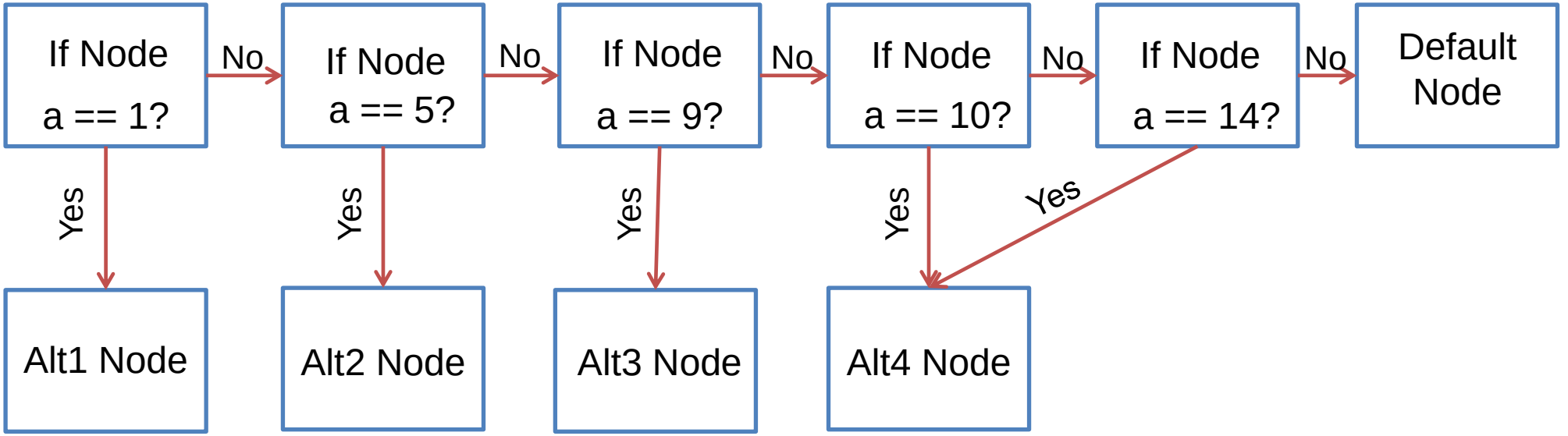


Недостатки

Занимает место в памяти соответственно разнице между максимальной и минимальной case-константами и может оказаться большим

Реализации switch в компиляторе для микропроцессоров «Эльбрус»

Цепочка конструкций if-else



Недостатки

- При большом количестве case-констант длина пути до узла-альтернативы из начального узла может достигать больших значений
- Длина пути до default-узла всегда равна числу case-констант

Цель

Создать новую оптимизацию, уменьшающую время исполнения программ с конструкцией switch.

Задачи

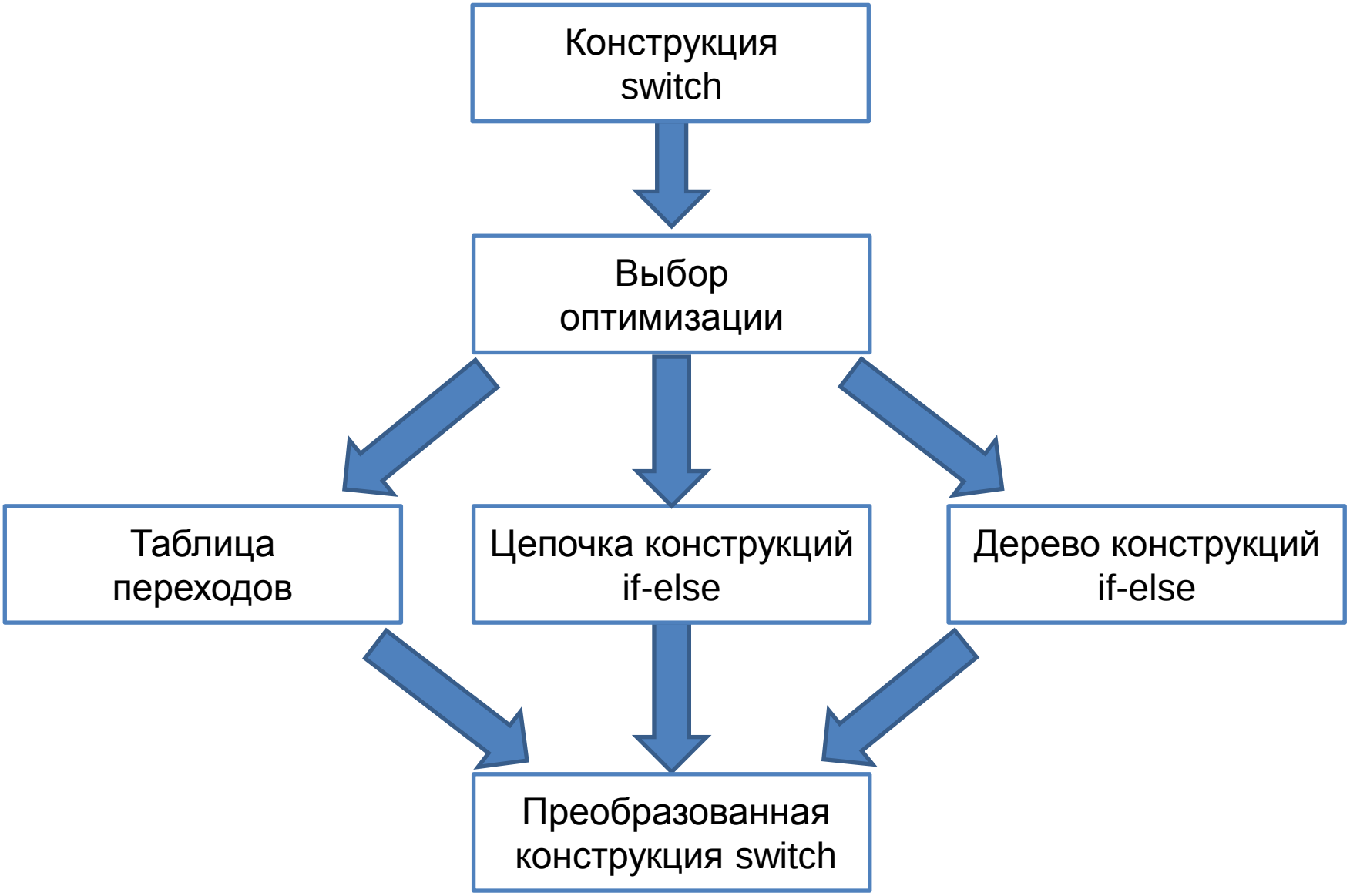
- Выбрать новую оптимизацию конструкции switch
- Провести анализ контекста применимости выбранной оптимизации
- Реализовать в компиляторе новую оптимизацию
- Произвести анализ влияния данной оптимизации на целевые программы

Возможные оптимизации конструкции switch

Реализация	Достоинства	Недостатки
Хэш-таблица	1. Время поиска в хэш-таблице составляет O(1) 2. Размер таблицы зависит от количества case-констант	1. Сложность реализации 2. Может потребоваться слишком много времени для вычисления хэш-функции
Сбалансированное дерево If-else конструкций	1. Время поиска в бинарном дереве поиска составляет $O(\log_2 N)$ 2. Простота реализации	1. Время поиска при балансировке по числу case-констант одинаково для всех альтернатив

Решено реализовать сбалансированное дерево конструкций If-else, как более простую и универсальную реализацию.

Схема преобразования конструкции switch



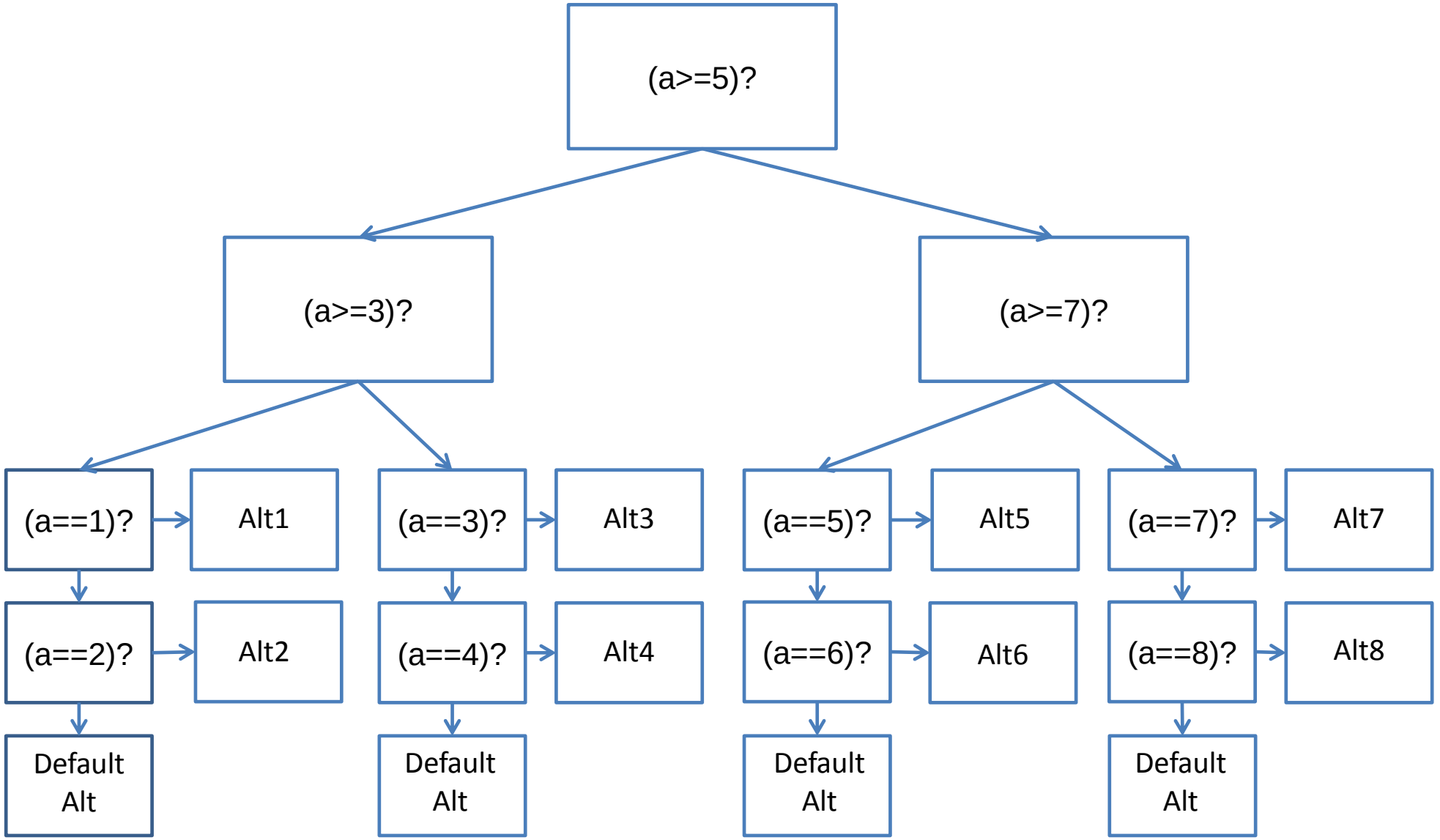
Анализ целесообразности применения дерева if-else конструкций

Если отношение количества case-констант к разности между максимальной и минимальной case-константами меньше 0,1, то:
строим таблицу поиска
иначе - цепочку или дерево конструкций if-else

	Дерево if-else конструкций		**Цепочка if-else конструкций**	
Количество case-констант	Среднее число переходов на узел-альтернативу	Среднее число переходов на default-узел	Среднее число переходов на узел-альтернативу	Число переходов на default-узел
1	1	1	1	1
2	2	2	1,5	3
3	$2\frac{1}{3}$	2,5	2	4
4	2,5	3	2,5	5
5	2,8	3,5	3	6

При количестве case-констант большем трех сбалансированное дерево поиска целесообразнее, чем цепочка if-else конструкций

Пример сбалансированного дерева



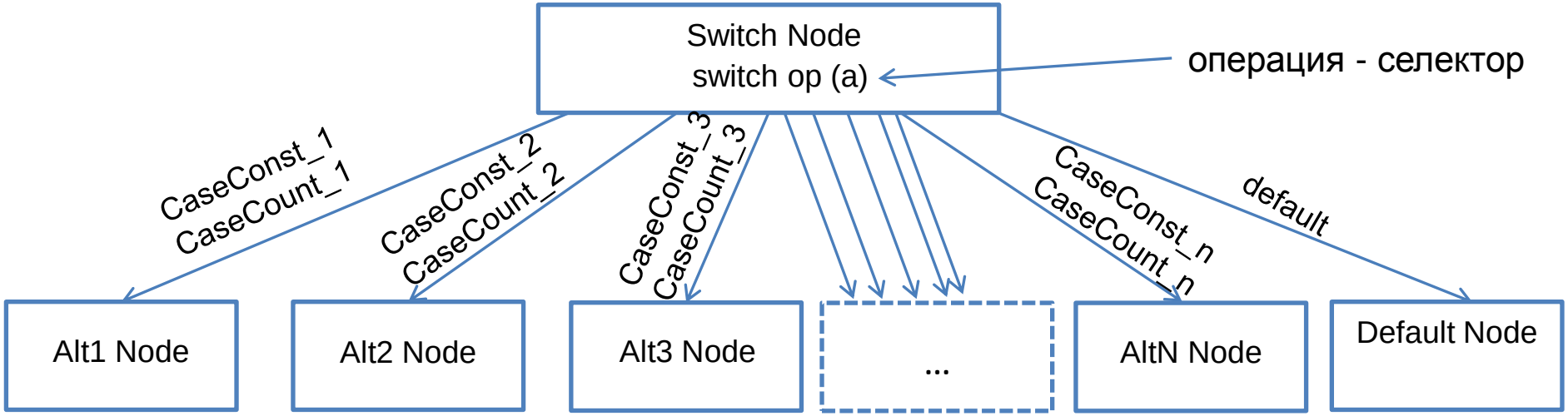
Балансировка дерева

Тип балансировки	**Основной принцип**	**Преимущества**	**Недостатки**
По числу case-констант	Высоты левого и правого поддеревьев отличаются не более, чем на 1	Для построения дерева не нужна профильная информация	Путь до всех альтернатив одинаковый и может оказаться большим для часто встречающихся case-констант
По счетчикам case-дуг	Разность сумм счетчиков case-констант в левом и правом поддеревьях минимальная	Путь до альтернативы тем короче, чем выше вероятность перехода на нее по данной case-константе	Если предсказанная профильная информация неверна, путь до альтернатив может оказаться больше, чем в дереве сбалансированном по числу case-констант

Оба метода реализованы

Подготовка к построению дерева

Сбор информации



Вспомогательный массив

Упорядоченные по возрастанию case-константы

Суммарный счетчик всех case-констант до текущего элемента таблицы или NO_PROFILE, если профильная информация отсутствует

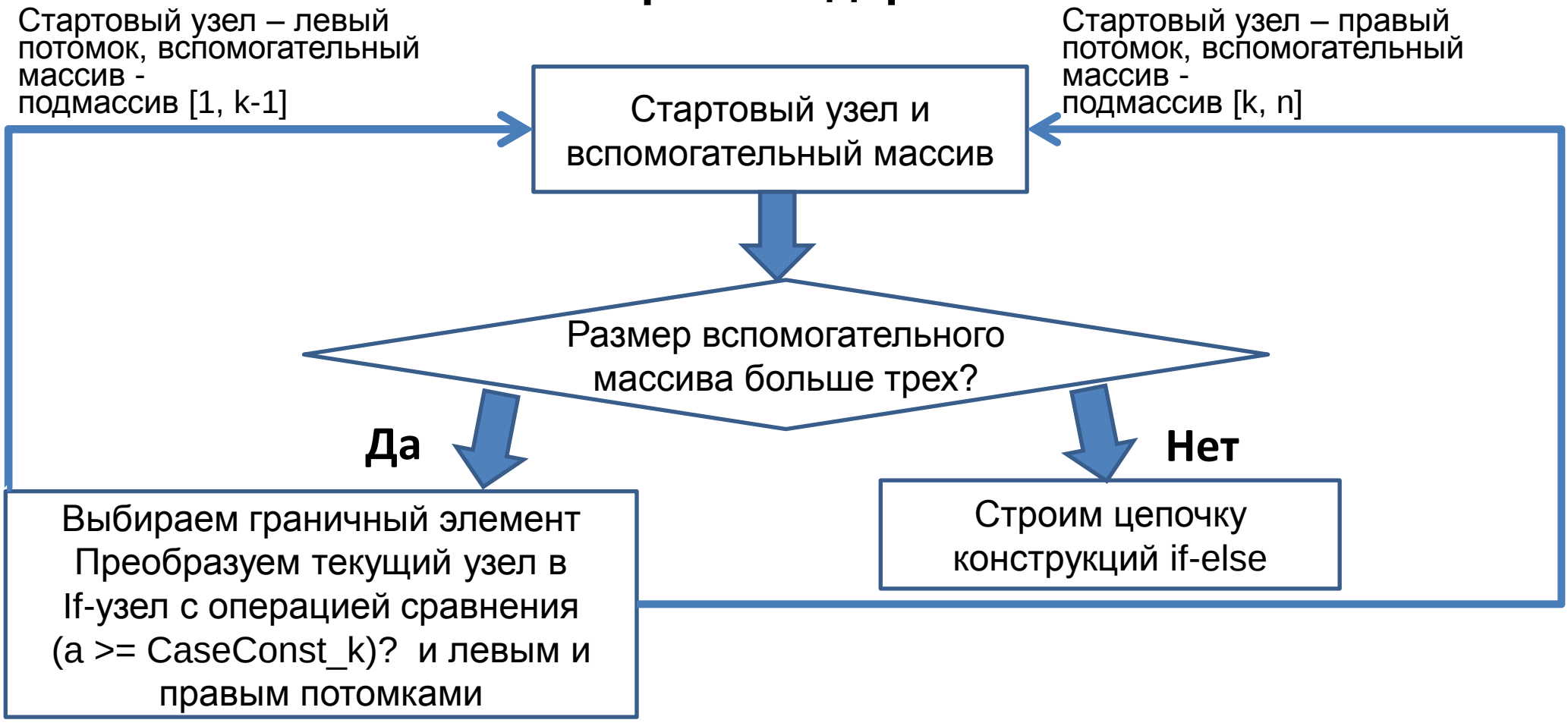
1	2	3	…	n
CaseConst_1	CaseConst_2	CaseConst_3	…	CaseConst_n
CaseCount_1	CaseCount_1 + CaseCount_2	CaseCount_1 + CaseCount_2 + CaseCount_3	…	$\sum_{i}^{n}$ CaseCount_i

Построение дерева

Нахождение граничного элемента

При балансировке по числу case-констант выбираем элемент вспомогательного массива с индексом k=[n/2]
При балансировке по счетчикам case-дуг выбираем элемент вспомогательного массива с индексом k таким, что сумма $\sum_i^k$ CaseCount_i максимально близка к $\frac{1}{2}\sum_i^n$ CaseCount_i

Построение дерева



Измерения

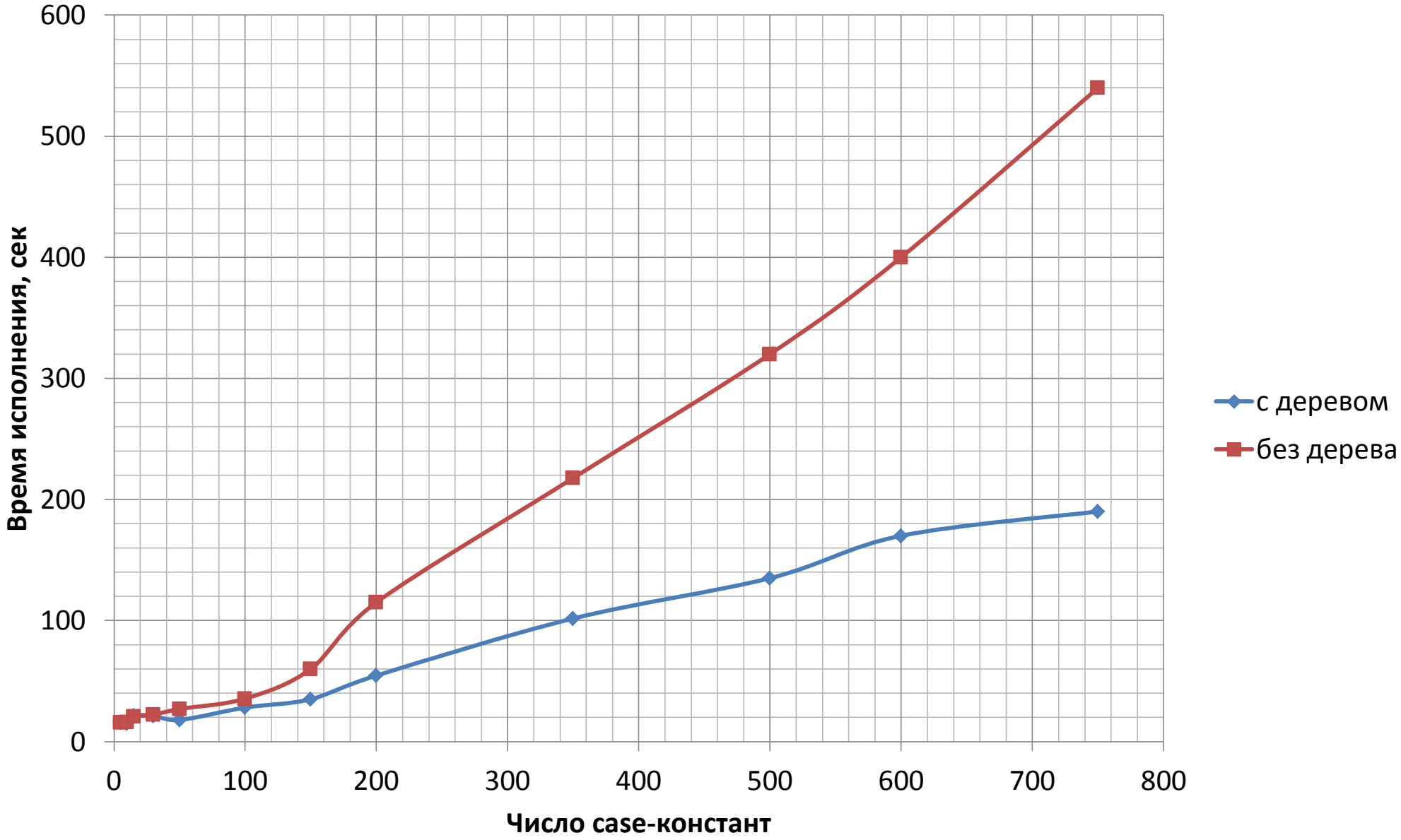
Описание тестовой среды

Измерения эффективности проходили на сгенерированных тестах. Каждый тест представляет из себя программу на языке C, состоящую из функции, с циклом, содержащим в себе конструкцию switch и имеющим большое число итераций. Каждый тест содержит разное число case-констант в конструкции switch.

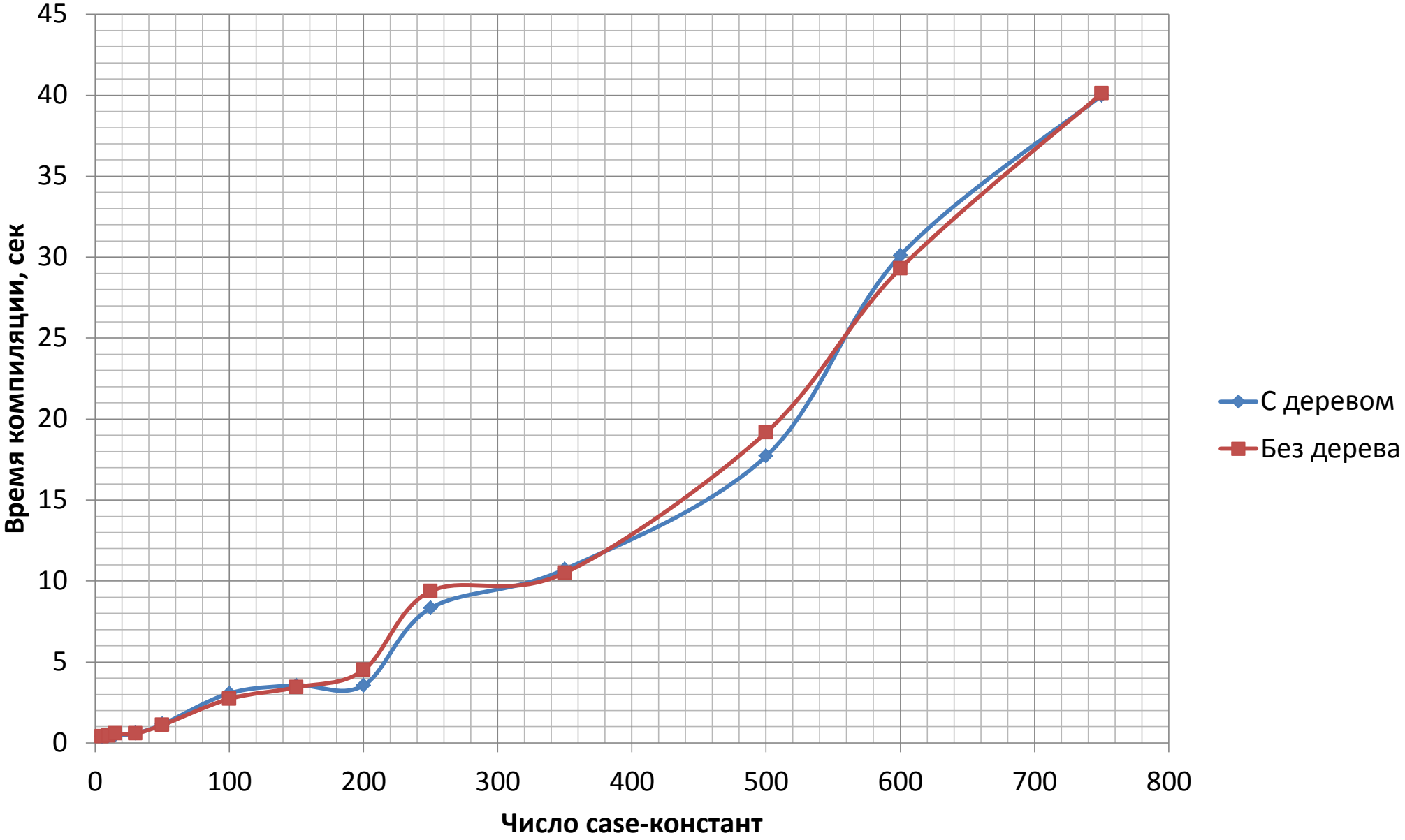
Цикл тестов

```
for (i = 0; i <= 100000000; i++)
{
        switch(i)
        {
                case 1: //Alt1
                case 27: //Alt2
                …
                case c://AltN
                default://Default Alt
        }
}
```

Зависимость времени исполнения программ от числа case-констант



Зависимость времени компиляции программ от числа case-констант



Результаты

- Проведен анализ контекста применимости оптимизации «преобразование конструкции switch в сбалансированное дерево if-else конструкций»
- В компиляторе для микропроцессоров «Эльбрус» реализовано преобразование конструкции switch в дерево сбалансированное по числу case-констант или по счетчикам case-дуг
- Произведены измерения эффективности оптимизации. Они показали уменьшение времени исполнения конструкции switch до трех раз.