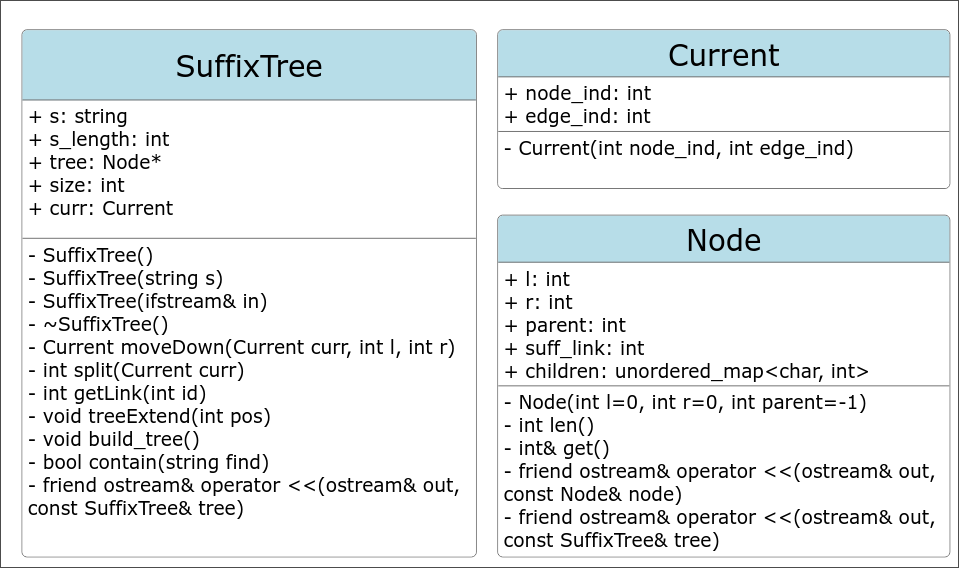
**Пояснительная записка**

**Файловая структура**

**Проект состоит из следующих основных файлов:**

* **main.cpp**:   
  Точка входа программы. Обрабатывает аргументы командной строки, считывает исходную строку из файла или с консоли, строит суффиксное дерево и в интерактивном режиме запрашивает подстроки для поиска.
* **current.h**:   
  Класс Current для текущего активного состояния (пара «индекс вершины / смещение по ребру») во время обхода или расширения дерева. Служит удобной реализацией active point из алгоритма Укконена.
* **node.h**:   
  Определяет класс Node — вершину суффиксного дерева. Хранит границу ребра [l,r), указатель на родителя, суффиксную ссылку и хеш-таблицу children <char,int> для переходов по символам. Реализует вспомогательный метод len(), get(char c).
* **suffixTree.h/suffixTree.cpp:**Объявление и реализация класса SuffixTree — инкапсулирует весь алгоритм построения (онлайн алгоритм Укконена) и поиска подстрок. Содержит методы build\_tree, treeExtend, moveDown, split, getLink, contain, а также структуру данных – суффиксное дерево.
* **Makefile:**Скрипт для сборки проекта.

**UML-диаграмма**

****

**Ключевые Алгоритмы**

1. **Представление строки и дерева**
   1. Строка хранится в поле string s, к ней однократно добавляется терминальный символ '$', гарантирующий, что все суффиксы различны.
   2. Дерево — динамический массив Node\* tree длиной 2·|s|. Доступ к вершинам осуществляется по целочисленным индексам; корень имеет индекс 0.
   3. Ребро кодируется парой индексов [l,r) в исходной строке — тем самым память для меток рёбер не дублируется.
2. **Алгоритм построения (онлайн, Укконен O(n))**
   1. Расширение (treeExtend(pos)): Выполняет фазу i алгоритма Укконена. Повторяет: «перемещение вниз» → split → добавление листа → создание суффиксной ссылки, пока активный узел не станет корнем и не будет создан открытый суффикс.
   2. Перемещение (moveDown(curr, l, r)): Опускается от активного состояния по отрезку [l,r); может вернуть невалидное состояние (−1) при несоответствии символов.
   3. Разбиение рёбер (split(curr)): При необходимости расщепляет ребро и создаёт внутренний узел, возвращая его индекс.
   4. Суффиксные ссылки (getLink(id)): Рекурсивно строит или возвращает суффиксную ссылку для вершины id, используя правило спуска (функция moveDown) и последующие расщепления.
   5. Основной цикл (build\_tree()): Итерируется по символам строки и расширяет дерево с помощью treeExtend(i).
3. **Поиск подстроки:** Метод contain(pattern) реализует детерминированный переход по суффиксному дереву:
   1. Сбрасывает curr в корень.
   2. Последовательно сопоставляет символы pattern, перемещаясь вдоль рёбер, контролируя смещение edge\_ind.
   3. При несовпадении или отсутствии перехода — немедленный отказ (false).
   4. Если весь шаблон исчерпан, возвращает true.
4. У**правление памятью**
   1. Все вершины размещаются в едином динамическом массиве Node\* tree, уничтожаемом в деструкторе SuffixTree::~SuffixTree().
   2. Суффиксные ссылки и дочерние переходы — обычные целочисленные индексы, что минимизирует накладные расходы.