Постановка задачи по оптимизации библиотеки libSBT

Цель работы:

> *Ускорение работы библиотеки libSBT (выполняющей операции над size-balanced tree), в частности - за счёт использования 64-битных инструкций платформы IA-64. Предполагаемое ускорение — порядка десятикратного.*

Содержание работы: Написать код на Си с ассемблерными вставками, либо на чистом ассемблере в AT&T-синтаксисе.

Результат работы:

1) Оптимизированный вариант библиотеки — программный код с ассемблерными вставками (или код на ассемблере).
2) Набор тестов, иллюстрирующих ускорение вставки/удаления/поиска/перебора по сравнению с оригиналом.
3) Подробная документация по применённой оптимизации: в чём отличия от оригинального кода на Си, как выполнена оптимизация: PDF-файл с иллюстрациями, диаграммами — поясняющими работу оптимизированного варианта, результаты тестов.

Требования к работе:

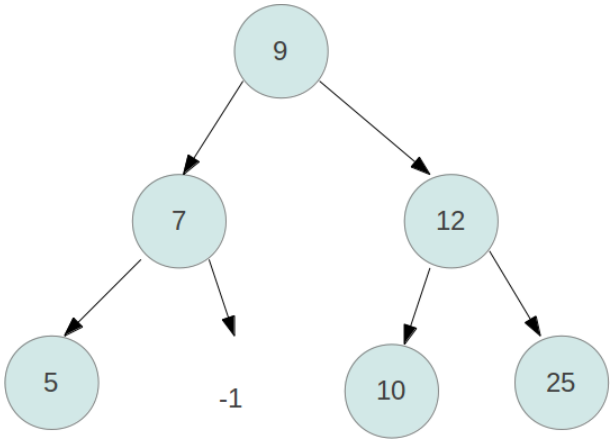
Результаты работы должны быть опубликованы на ресурсе GitHub:
- исходный код - под лицензией GNU Affero GPLv3,
- изображения - под лицензией Creative Commons BY-SA 3.0,
- тексты документации - под лицензией Creative Commons BY-SA 3.0.

Описание оригинального кода

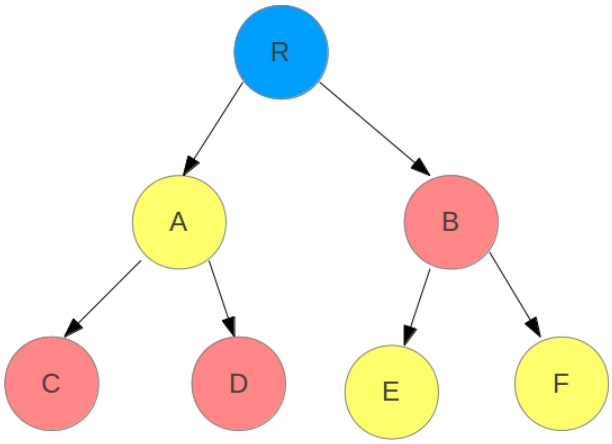
1. Основной тип (структура данных):

```
typedef struct TNode {
    TNumber value; // значение, привязанное к ноде
    TNodeIndex parent; // ссылка на уровень выше
    TNodeIndex left;  // ссылка на левое поддерево, = -1, если нет дочерних вершин
    TNodeIndex right; // ссылка на правое поддерево
    TNodeSize size; // size в понимании SBT
    int unused; // «удалённая»; это поле можно использовать и для других флагов
} TNode;
```

2. Пример дерева



3. Условие сбалансированности

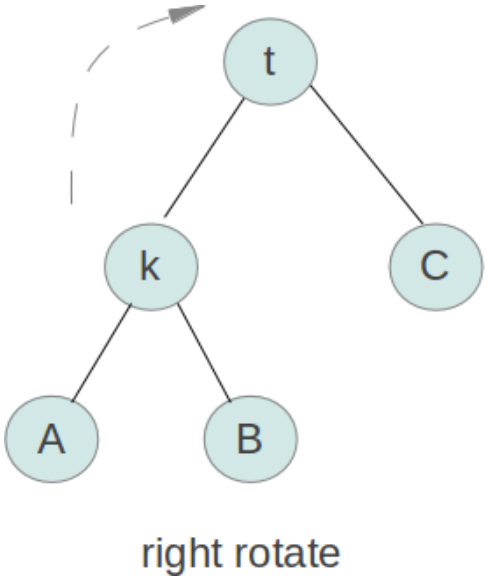


условие сбалансированности задаётся неравенствами:
E.size <= A.size C.size <= B.size
F.size <= A.size D.size <= B.size

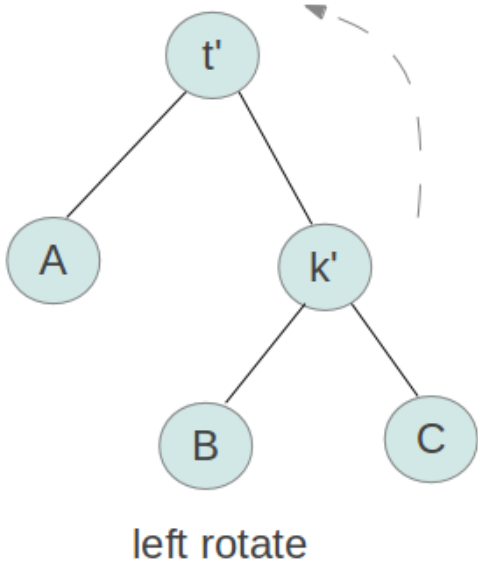
размеры (size) вершин определяются правилом:
R.size = A.size + B.size + 1
(то есть, размер вершины — это число вершин в соответствующем поддереве)

4. Вращения для балансировки
Для балансировки наших деревьев (SBT) используются «вращения»:

«правое вращение»



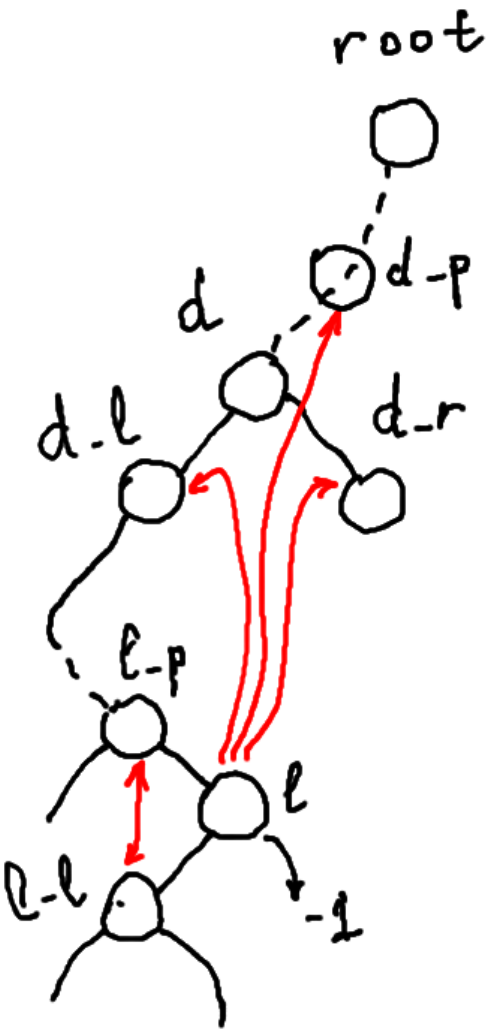
«левое вращение»



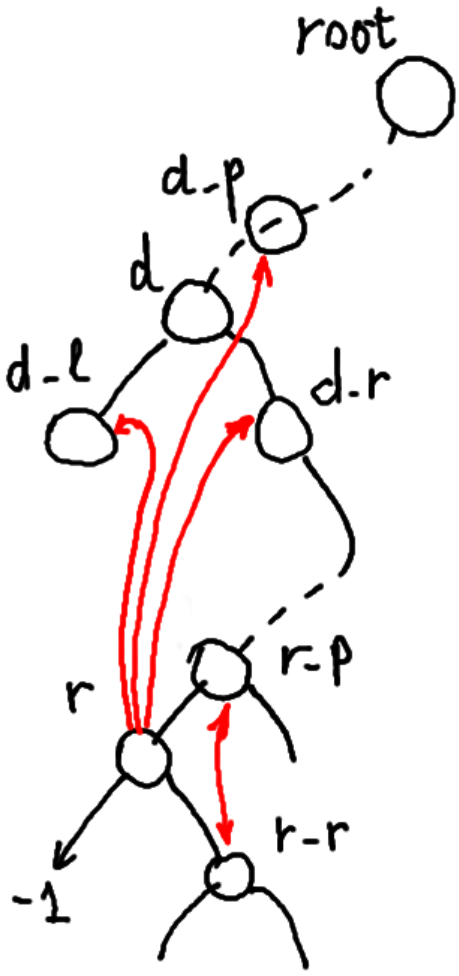
Основной момент здесь - это критерий «вращений» (rotates). Рассматривается четыре случая взаимного отношения «размеров» (size) вершин, близких к "корню" балансировки (t). В зависимости от соотношения «размеров», выполняется то или иное вращение.

Балансировка выполняется после добавления вершины (функция AddNode), а также — после удаления вершины из дерева (функция DeleteNode).

5. Удаление вершины



NearestAndLesser



NearestAndGreater

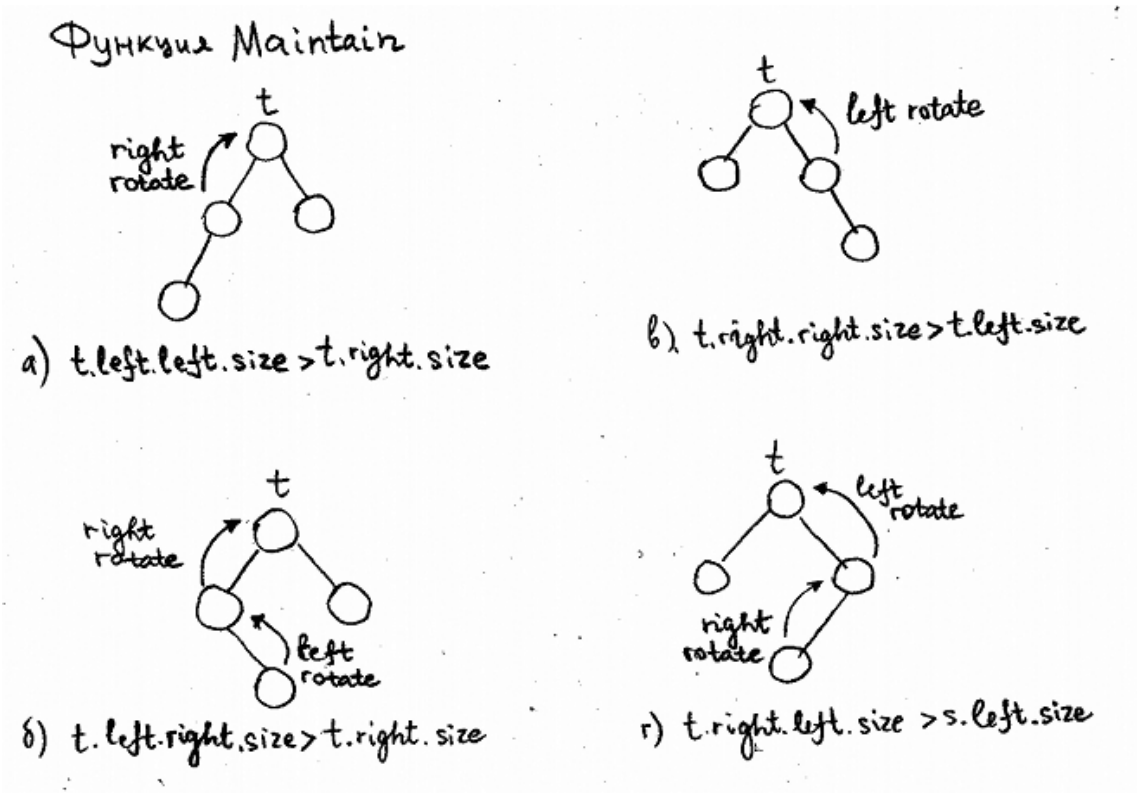
DeleteNode, удаление вершины

Удаление вершины — это:
1) собственно вырезание вершины из дерева,
2) освобождение памяти из-под этой ячейки,
3) перевешивание на место удалённой вершины — другой, наиболее близкой по значению (NearestAndLesser, NearestAndGreater),
4) балансировка поддерева, которое изменили.

Вырезание вершины — это ключевая часть функции DeleteNode.

Балансировка выполняется после вырезания и перевешивания вершины. Выполняется от непустой вершины: L_L или L_P (R_R или R_P) вверх, до корня.

6. Добавление вершины



Добавление вершины происходит в три шага:
1) Выделение памяти под ячейку (node),
2) Добавление вершины (node) в дерево,
3) Упорядочивание — балансировка вершин (вызов Maintain для соответствующего поддерева и «соседних» поддеревьев).