(a, b)-tree experiment

Test program

The test program evaluates the implementation of (2, 3)-tree and (2, 4)-tree.
It performs three types of tests:

1. Insert test
2. Min test
3. Random test

Performance

Performance is measured in terms of the average number of structural changes.
Structural change is either a node split (in insert) or merging of two nodes (in delete).

- #SPLIT the number of performed split operations
- #INSERT the number of performed insert operations

- #NODES the number of nodes
- HEIGHT the height of the tree

Plots

Each plot shows the dependence of the average number of structural changes ASC on the set size #KEYS.

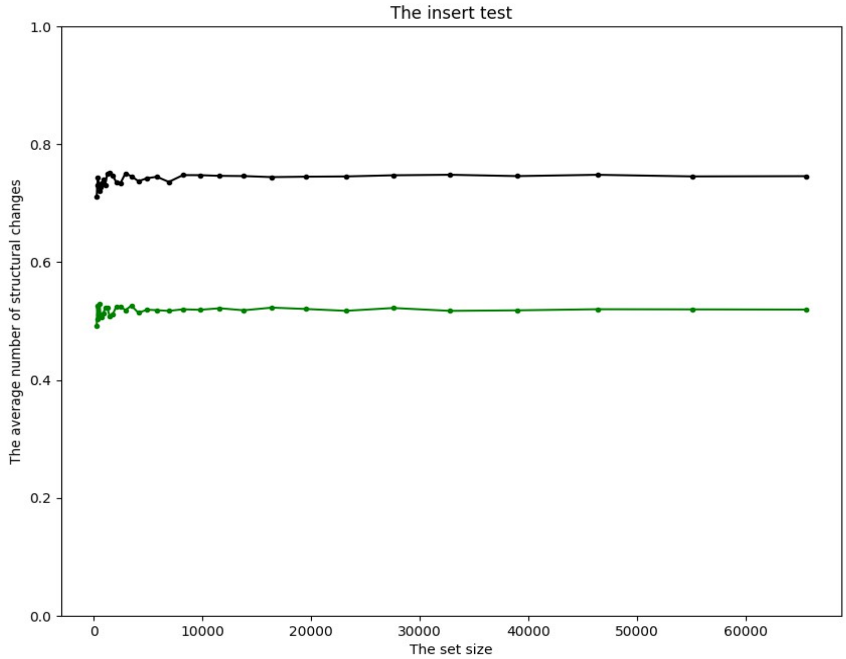
- (2, 3)-tree the black curve
- (2, 4)-tree the green curve

Theoretical facts

- The height of (a, b)-tree is at least $\log_b$ (#INSERT + 1).

1. Insert test

Insert n elements in random order.



Estimation of bounds for the average number of structural changes:

- $1 / (b - 1) - \log_b(\#INSERT + 1) / \#INSERT \quad <= \#SPLIT / \#INSERT \quad < 1.$
  - $1 / (b - 1) - \log_b(\#INSERT + 1) / \#INSERT \quad <= \#SPLIT / \#INSERT$:
    - #INSERT <= (b - 1) * #NODES.
      - Each node has at most (b – 1) keys.
    - #NODES - #HEIGHT <= #SPLIT.
      - Each split creates at least one node.
      - When we split the root, another new node is created and added as a new root.
    - $$\begin{aligned}\#SPLIT / \#INSERT &>= (\#NODES - \#HEIGHT) / \#INSERT \\ &= \#NODES / \#INSERT - \#HEIGHT / \#INSERT \\ &>= \#NODES / ((b - 1) * \#NODES) - \#HEIGHT / \#INSERT \\ &= 1 / (b - 1) - \#HEIGHT / \#INSERT \\ &>= 1 / (b - 1) - \log_b(\#INSERT + 1) / \#INSERT.\end{aligned}$$

- #SPLIT / #INSERT < 1:

  - Consider the worst case:
    - (2, 3)-tree: insert sequence in ascending or descending order
    - (2, 4)-tree: insert sequence in ascending order

  - There is only one branch, to that the keys are inserted.
    Let's say it has internal nodes $n_1, n_{2, \ldots,} n_h$, where n_1 is the lowest internal node and n_k is root.
  - For each such node holds:
    - except n_1 (initial root), each node was created with (b - 2) key.
    - it has (b - 2) keys after each split.
    - after each insertion to this node without a split, the next insertion is performed with a split, because the node can't contain more than (b - 1) keys.

  - First insertion creates n_1 with one key.
  - Next (b -2) insertions adds keys to n_1.
  - Next insertion splits n_1 and creates n_2 (as a new root) with one key.

  - After creating n_i, all nodes in the tree have exactly one key and n_i is the new root.
    We need to perform next 2^i insertions { $\text{insert}_1, \text{insert}_2, \ldots, \text{insert}_{2^i}$ } until n_{i+1} is created and added as a new root:

    - For j: $1 <= j <= 2^i$ each insert_j starts in n_1.
    - For j: $1 <= j <= 2^{i-1}$ each insert_{2j} is performed with splitting of n_1.
      After this, some key is inserted to n_2.
    - For j: $1 <= j <= 2^{i-2}$ each $\text{insert}_{2^2 j}$ is performed with splitting of n_1 and n_2.
      After this, some key is inserted to n_3.
    - Last insertion, insert_{2^i}, is performed with splitting of $n_1, \ldots, n_i$.
      After this, n_{i+1} is created and added as a new root.

    We perform exactly 2^{i-1} splits on node n_i during these 2^i insertions.

    - Sum of the number of splitting over $n_1, \ldots, n_i$ is:
      $2^{i-1} + 2^{i-2} + 2^{i-3} + \ldots + 2^0 = 2^i - 1$.

  - #SPLIT / #INSERT < 1.
    - We perform exactly 2^i insertions until new root n_{i+1} is created, the average number of structural changes for each level i is $(2^i - 1) / 2^i < 1$.

TODO:

Min test:
Insert n elements sequentially and then n times repeat: remove the minimal element in the tree and then insert it back.

Random test:
Insert n elements sequentially and then n times repeat: remove random element from the tree and then insert random element into the tree.
Removed element is always present in the tree and inserted element is always not present in the tree.