Data Structures and Algorithms (12)

Instructor: Ming Zhang
Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)
https://courses.edx.org/courses/PekingX/04830050x/2T2014/
Chapter 12 Advanced data structure

• 12.1 Multidimensional Array
• 12.2 Generalized Lists
• 12.3 Storage management
• 12.4 Trie
• 12.5 Improved binary search tree
12.3 Trie

- Ideal situation: The average time of insertion, deletion, and search is $O(\log N)$
- Input 9, 4, 2, 6, 7, 15, 12, 21
- Output 2, 4, 6, 7, 9, 12, 15, 21
Structure of Trie

- Space division of key
- “trie” comes from “retrieval”
- Application
  - Information retrieval
  - Large scale of English dictionary
- 26-branch Trie
- Binary Trie
  - Letters (numbers) represented as binary coding
  - Coding includes just 0 and 1
12.4 Trie

Tree of English words: 26-branch Trie

Store words ‘and, ant, bad, bee’

Subtree ‘an’ contains set {and, ant} that every word from the set has the same prefix ‘an’.

A subtree contains the words with the same prefix.
12.4 Trie

Store words an, and, ant, bad, bee

Diagram of a Trie tree storing the words an, and, ant, bad, bee.
12.4 Trie

Compact the Single Paths close to the leaf

Store words an, and, ant, bad, bee
12.4 Trie

Binary Trie

Elements are 2, 5, 9, 17, 41, 45, 63
Chapter 12
Advanced Data Structure

12.4 Trie

PATRICIA Structure

Compression

Code: 2: 000010 5: 000101 9: 001001
17: 010001 41: 101001 45: 101101 63: 111111
Characteristics of PATRICIA Tree

• The compressed PATRICIA tree is a full binary tree
  • Every internal node represents a 1-bit comparison
  • Always at least two children are generated

• The number of comparisons will not exceed the length of the key
12.4 Trie

Suffix Trees

$\sum \varepsilon$

$T = ababc$

Explicit states

Implicit states

Ascending Order

Signs:

ab

b

c

ababc

babc

ababc

babc

ababc

babc

ababc

babc

ababc

babc

ababc

babc

ababc

babc

ababc

babc

ababc

babc
### Suffix Array

**Suffix Array**

<table>
<thead>
<tr>
<th></th>
<th>ALAM$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALAYALAM$</td>
</tr>
<tr>
<td>7</td>
<td>AM$</td>
</tr>
<tr>
<td>3</td>
<td>AYALAM$</td>
</tr>
<tr>
<td>6</td>
<td>LAM$</td>
</tr>
<tr>
<td>2</td>
<td>LAYALAM$</td>
</tr>
<tr>
<td>0</td>
<td>MALAYALAM$</td>
</tr>
<tr>
<td>8</td>
<td>M$</td>
</tr>
<tr>
<td>4</td>
<td>YALAM$</td>
</tr>
<tr>
<td>9</td>
<td>$</td>
</tr>
</tbody>
</table>

**MALAYALAM$**

0 1 2 3 4 5 6 7 8 9

**Suffix Array**

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>1</th>
<th>7</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>0</th>
<th>8</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
</table>

**The longest common prefix array**

Suffix 5 and Suffix 1 share “ALA”
Suffix 1 and Suffix 7 share “A”
LCP always adjacent
12.4 Trie

<table>
<thead>
<tr>
<th></th>
<th>ALAM$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALAYALAM$</td>
</tr>
<tr>
<td>7</td>
<td>AM$</td>
</tr>
<tr>
<td>3</td>
<td>AYALAM$</td>
</tr>
<tr>
<td>6</td>
<td>LAM$</td>
</tr>
<tr>
<td>2</td>
<td>LAYALAM$</td>
</tr>
<tr>
<td>0</td>
<td>MALAYALAM$</td>
</tr>
<tr>
<td>8</td>
<td>M$</td>
</tr>
<tr>
<td>4</td>
<td>YALAM$</td>
</tr>
<tr>
<td>9</td>
<td>$</td>
</tr>
</tbody>
</table>

SA

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>7</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>0</th>
<th>8</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>D = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D = 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$lcp$
Discussions

• Can Trie handle Chinese characters? What about PATRICIA Trie structure?
• Learn related document about Suffix Array and Suffix Tree. And think about their applications.
Chapter 12 Advanced data structure

• 12.1 Multidimensional Array
• 12.2 Generalized Lists
• 12.3 Storage management
• 12.4 Trie

• 12.5 Improved binary search tree
  • 12.5.1 Balanced binary search tree
    • Concept and inserting operation of AVL tree
    • Deleting operation and efficiency analysis of AVL tree
  • 12.5.2 Splay Tree
12.5 Improved binary search tree

12.5.1 AVL

- The performance of BST operations are affected by the input sequence
  - Best $O(\log n)$; Worst $O(n)$
- Adelson-Velskii and Landis
  - AVL tree, a balanced binary search tree
  - Always $O(\log n)$
12.5 Improved binary search tree

12.5.1 AVL

• Single Rotation
  - Swap the node with its father, while keeping the property of BST

![Diagram of AVL tree rotations](attachment:image.png)
12.5 Improved binary search tree

12.5.1 AVL

- Single Rotation and Double Rotation: Keep the BST property.
12.5 Improved binary search tree

12.5.1 AVL

• Equivalent rotation: Keep the BST property
12.5 Improved binary search tree

AVL

• Empty tree is allowed
• The height of AVL tree with n nodes is $O(\log n)$
• If T is an AVL tree
  • Then the left and right subtree of T: $T_L, T_R$ are also AVL trees
  • And $|h_L - h_R| \leq 1$
    • $h_L, h_R$ are the heights of its left and right subtree.
Examples of AVL tree

12.5 Improved binary search tree
12.5 Improved binary search tree

Balance Factor

- Balance Factor, \( bf(x) \):
  - \( bf(x) = \text{height}(x_{\text{rc}hild}) - \text{height}(x_{\text{l}c}hild) \)
- Balance Factor might be 0, 1 and -1
12.5 Improved binary search tree

Insertion in an AVL tree

• Just like BST: insert the current node as a leaf node
• Situations during adjustment
  • The current node was balanced. Now its left or right subtree becomes heavier.
    • Modify the balance factor of the current node
  • The current node had a balance factor of ±1. Now the current node becomes balanced.
    • Height stays the same. Do not modify.
  • The current node had a balance factor of ±1. Now the heavier side becomes heavier
    • Unbalanced
    • “dangerous node”
12.5 Improved binary search tree

**Rebalance**

Become unbalanced after inserting 17

Adjustment
• Unbalanced situation occurs after insertion
• Insert the current node as leaf node in BST
• Assume a is the most close node to the current node. And its absolute value of balance factor is not zero.
• The current node s with key is in its left subtree or its right subtree.
• Assume that it is inserted into the right subtree. The original balance factor:
  • (1) \( bf(a) = -1 \)
  • (2) \( bf(a) = 0 \)
  • (3) \( bf(a) = +1 \)
• Assume a is the most close node to the current node s. And its absolute value of balance factor is not zero.
  • S is in a’s left subtree or right subtree.
• Assume S is in the right subtree. Because balance factors of nodes in paths from s to a change from 0 to +1. So as for node a:
  1. $bf(a) = -1$, then $bf(a) = 0$, and the height of node a’s subtree stays the same.
  2. $bf(a) = 0$, then $bf(a) = +1$, and the height of node a’s subtree stays the same.
      • Because of the definition of a ($bf(a) \neq 0$), we can know that node a is the root.
  3. $bf(a) = +1$, then $bf(a) = +2$, and adjustment is needed.
12.5 Improved binary search tree

Unbalanced Cases

• The balance factors of any nodes must be 0, 1, -1
• a’s left subtree was heavier, $bf(a) = -1$, and $bf(a)$ become -2 after insertion.
  • LL: insert into the left subtree of a’s left child.
    • Left heavier + left heavier, $bf(a)$ become -2
  • LR: insert into the right subtree of a’s left child.
    • Left heavier + right heavier, $bf(a)$ become -2
• Likewise, $bf(a) = 1$, and $bf(a)$ become 2 after insertion
  • RR: the node that causes unbalanced is in the right subtree of a’s right child.
  • RL: the node that causes unbalanced is in the left subtree of a’s right child.
12.5 Improved binary search tree

Unbalanced Cases

LL

RR

$h+1$

$h$

$h$

$h+1$

$h$

$h$

$h$

$a$ -2

$b$ -1

$a$ 2

$b$ 1

$h$

$h$

$h$

$h$

Ming Zhang  “Data Structures and Algorithms"
Summary of unbalanced cases

• LL is symmetry with RR, and LR is symmetry with RL.
• Unbalanced nodes happen on the path from inserted node to the root.
• Its balance factor must be 2 or −2
  • If 2, the balance factor before insertion is 1
  • If −2, the balance factor before insertion is −1
12.5 Improved binary search tree

**LL single rotation**

Diagram illustrating the LL single rotation in a binary search tree.
Insight of Rotations

• Take RR for instance, there are 7 parts
  • Three nodes: a, b, c
  • Four subtrees $T_0, T_1, T_2, T_3$
    • The structure will not change after making c’s subtree heavier.
    • $T_2, c, T_3$ could be regarded as b’s right subtree.

• Goal: construct a new AVL structure
  • Balanced
  • Keep the BST property
    • $T_0 \ a \ T_1 \ b \ T_2 \ c \ T_3$
Double Rotation

• RL or LR needs double rotation.
  • They are symmetry with each other
• We discuss about RL only
  • LR is the same.
First step of RL double rotation

Height of a’s subtree is \( h+2 \) before inserting
Height of a’s subtree is \( h+3 \) after inserting
Second step of RL double rotation

- Balance factor is meaningless in the middle status
- Balance factor of a is -1 or 0
- Balance factor of b is 0 or 1
Insight of Rotations

- Doing any rotations (RR, RL, LL, LR)
- New tree keeps the BST property
- Few pointers need to be modified during rotations.
- Height of the new subtree is $h+2$, and heights of subtrees before insertion stay same
- Rest parts upon node a (if not empty) are always balanced
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia

Unbalanced after inserting copy
LR double rotation
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia

RL double rotation
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia
12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia
Chapter 12
Advanced Data Structure

12.5 Improved binary search tree

AVL tree after inserting word: cup, cop, copy, hit, hi, his and hia

LL single rotation
Discussions

• Can we modify the definition of balance factor of AVL tree? For example, allow the height difference as big as 2.

• Insert 1, 2, 3, ..., $2^k - 1$ into an empty AVL tree consecutively. Try to prove the result is a complete binary tree with height k.
Chapter 12 Advanced data structure

• 12.1 Multidimensional Array
• 12.2 Generalized Lists
• 12.3 Storage management
• 12.4 Trie
• 12.5 Improved binary search tree
  • 12.5.1 Balanced binary search tree
    • Concept and insertion operation of AVL tree
    • Deletion operation and efficiency analysis of AVL tree
  • 12.5.2 Splay Tree
Deletion in an AVL Tree

• Deletion is the reverse operation of insertion. Deletion in an AVL tree is almost the same with the deletion of BST.
• Deletion in an AVL tree is a little bit complicated.
Adjustment after Deletion

- Conditions of adjustment
  - Height and balance factor has changed.
  - Adjust these changes from the current node to the root
- Balance factor needs to be modified
  - Modify it
- If modification is not needed then stop the adjustment
  - Use bool variable modified to mark, initialize it with TRUE
  - when modified = FALSE, stop the adjustment.
• Balance factor of the current node $a$ is 0
  • Its left or right subtree is cut, balance factor is 1 or -1
  • modified = FALSE
  • Modification will not influence the nodes above it. So the adjustment is over.

```
delete
```

```
0   1
\h / \h
  h   h
```

```
\h\rightarrow\h-1\rightarrow\h
```
Case 2

- Balance factor of the current node is not 0, but the taller subtree is cut.
  - Modify its balance factor to 0
  - modified = TRUE
  - Keep modifying upward

![Diagram showing height reduction by 1](image-url)
12.5 Improved binary search tree

**Case 3.1**

- Balance factor of the current node a is not zero. And the shorter subtree is cut. So node a isn’t balanced.
- Assume the root of the taller subtree is b. There are three cases.
- case 3.1: balance factor of b is zero
  - Single rotation
  - modified = FALSE

![Diagram of Case 3.1](image-url)
Case 3.2

• Case 3.2: balance factor of b is equal to that of a.
  - Single rotation
  - Balance factors of a and b change to zero
  - modified = TRUE

![Diagram of improved binary search tree](image_url)
Case 3.3

- Case 3.3: balance factor of b is opposite to that of a
  - Double rotation, rotate b and c, then rotate c and a
  - Because of balance factor of the new root is 0
  - Other nodes need adjustment
  - Besides modified = TRUE
12.5 Improved binary search tree

Series of adjustments after deletion

• Series of adjustments
  • Ancestor nodes may be unbalanced after the adjustment
  • Keep adjusting. These adjustments might pass to the root.
• Find the grandfather of the node deleted just now.
  • Start single rotation or double rotation
  • Rotation times is $O(\log n)$
Examples of deletion

(a) Delete node m, replace m with node I (case 1)
Chapter 12
Advanced Data Structure

12.5 Improved binary search tree

Examples of deletion

(b) delete n (Case 3.2)

(c) Regard l as the root, doing LL single rotation (Case 3.2)
Examples of deletion

(d) After LL single rotation, then adjust its father node I, doing LR double rotation (Case 3.3)

(e) Adjustment is done, AVL tree become balanced
12.5 Improved binary search tree

Height of AVL tree

• Height of AVL tree with n nodes must be $O(\log n)$
• The maximum height of AVL tree with n nodes is no more than $K \log_2 n$
  • $K$ is a small factor
  • AVL tree that is the most close to unbalanced
  • Construct a series of AVL tree $T_1, T_2, T_3, ...$
12.5 Improved binary search tree

Ti 's height is i

Any other AVL trees with height I have more nodes than Ti

Ti is the most unbalanced AVL tree. Deleting any node will cause it to be unbalanced.
Proof of height

• We can see that:
  \[ t(1) = 2 \]
  \[ t(2) = 4 \]
  \[ t(i) = t(i-1) + t(i-2) + 1 \]
  \[ (t(i)+1) = (t(i-1)+1) + (t(i-2)+1) \]

• As for i>2 these relations are very similar to Fibonacci numbers:
  \[ F(0) = 0 \]
  \[ F(1) = 1 \]
  \[ F(i) = F(i-1) + F(i-2) \]
Proof of height

• As for $i > 1$, we have
  \[ t(i) = F(i+3) - 1 \]

• Fibonacci number has this formula
  \[ F(i) = \frac{1}{\sqrt{5}} \phi^i, \text{ here } \phi = \frac{1 + \sqrt{5}}{2} \]

• So the approximate formula is
  \[ t(i) \approx \frac{1}{\sqrt{5}} \phi^{i+3} - 1 \]
Proof of height (result)

• Relation between height $i$ and the number of nodes $t(i)$

$$\phi^{i+3} \approx \sqrt{5} (t(i) + 1)$$

• Use formula $\log_{\phi} X = \log_2 X / \log_2 \phi$ and $\log_2 \phi \approx 0.694$, calculate the approximate upper bound.
  • $t(i) = n$

$$i + 3 \approx \log_{\phi} \sqrt{5} + \log_{\phi} (t(i) + 1)$$

• So height of AVL tree with $n$ nodes must be $O(\log n)$

$$i < \frac{3}{2} \log_2 (n+1) - 1$$
Efficiency of AVL tree

- Time cost of searching, inserting and deleting is $O(\log_2 n)$
  - Height of AVL tree with $n$ nodes must be $O(\log n)$
- AVL tree is fit to data of small scale in memory
- For large scale data stored in external memory
  - B tree or B+ tree
12.5 Improved binary search tree

Discussion

• Compare AVL tree with black-red tree, which is better?
  • Height of tree in the worst case
  • Efficiency of operation in statistics
  • Which one is easier to implement?
Chapter 12 Advanced data structure

• 12.1 Multidimensional Array
• 12.2 Generalized Lists
• 12.3 Storage management
• 12.4 Trie
• 12.5 Improved binary search tree
  • 12.5.1 Balanced binary search tree
  • 12.5.2 Splay tree
12.5.2 Splay Tree

- A self-organized data structure
  - Adjust data during search
  - Dictionary of Chinese input method
- Splay tree is not a new data structure. It just adds some rules to BST to improve efficiency.
  - Make sure the total cost of search is not high and achieve a satisfactory performance.
- Heights of tree are not balanced.
12.5 Improved binary search tree

12.5.2 Splay Tree

- Single Rotation
  - Swap the current node and its father. Keep the BST property.
12.5 Improved binary search tree

12.5.2 Splay Tree

- Double Rotation: LL and RR. Keep the BST property.
12.5 Improved binary search tree

12.5.2 Splay Tree

- Double Rotation: LL and RR. Keep the BST property.

\[
\begin{array}{c}
\text{zig-zag} \\
\text{zag-zig}
\end{array}
\]
Splaying

• Visit a node (e.g., node x). Then do an operation called splaying.
  • After inserting or visiting node x, splay (move) node x to the root.
  • After deleting node x, splay (move) the father of node x to the root.

• Like AVL tree. The operation of splaying x consists of a series of rotations.
  • Adjust x, father of x and grandfather of x.
  • Move x toward the root.
### Single rotation

- x is the child of the root
  - Swap x and the father of x
  - Keep the BST property

X, y is marks of inner node not value.
A, B, C represent subtree.
Double rotation

• Double rotation involves:
  • Node x
  • The father of node x (called y)
    • The grandfather of node x (called z)
• Rise node x by two (height).
• zigzig rotation
  • Also called homogeneous configuration
• zigzag rotation
  • Also called heterogeneous configuration
ZigZig

Node x is the left child of y
Node y is the left child of z
12.5 Improved binary search tree

ZigZig

Diagram of a binary search tree with nodes X, Y, Z, A, B, C, D.
12.5 Improved binary search tree

ZigZag

Node x is the right child of y
Node y is the left child of z
12.5 Improved binary search tree

ZigZag
12.5 Improved binary search tree

ZigZag

Diagram of a binary tree with nodes labeled X, Y, Z, and A, B, C, D.
Different results of ZigZig and ZigZag

• ZigZag
  • Move the current node toward the root
  • The height of its subtree decreased by one
  • Make the tree more balanced

• ZigZig
  • The height of the subtree will not change usually
  • Move the current node toward the root
12.5 Improved Binary Search Tree

Adjustment of Splay Tree

• A series of rotations
  • Until node x become root or the child of root.
• If the node x is the child of root
  • Single rotation will make x become root
• This process makes tree more balanced
  • Make nodes, which are frequently visited, closer to the root
  • Reduce the cost of search
Adjustment of Splay Tree

(a-b-c) ZagZag

(a, c)Zag

(a, b)Zag
12.5 Improved binary search tree

Adjustment of Splay Tree

a-d-e
ZagZig
Adjustment of Splay Tree

(a, g) Zig

(a-f-g) ZagZig

(a, f) Zag
12.5 Improved binary search tree

Adjustment of Splay Tree

(a, h) Zag

Single rotation

Diagram of a splay tree with nodes labeled from a to h, illustrating the (a, h) Zag operation.
Adjustment of Splay Tree
Chapter 12
Advanced Data Structure

12.5 Improved binary search tree

Splay Tree

```
struct TreeNode
{
    int key;
    ELEM value;
    TreeNode *father, *left, *right;
};
```

```
Splay(TreeNode *x, TreeNode *f); // Splay x to the child of f
Splay(x, NULL); // Splay x to the root
Find(int k);
Insert(int k);
Delete(TreeNode *x); // delete node x
DeleteTree(TreeNode *x); // delete subtree x
```
void Splay (TreeNode *x, TreeNode *f) {
    while (x->parent != f) {
        TreeNode *y = x->parent, *z = y->parent;
        if (z != NULL) {  // grandfather is null
            if (z->lchild == y) {
                if (y->lchild == x) {
                    Zig(y); Zig(x); }  // ZigZig
                else { Zag(x); Zig(x); }  // Zag x then Zig x
            } else {
                if (y->lchild == x) {
                    Zig(x); Zag(x); }  // Zig x then Zag x
                else { Zag(y); Zag(x); }  // ZagZag
            }
        } else {
            if (y->lchild == x) Zig(x);  // Zig
        } else Zag(x); }
    if (x->parent == NULL) Root = x;
}
Delete all nodes range from u to v

- Splay u to the root
- Splay v to the right child of u
- Delete the left subtree of v

```c
void DeleteUV(TreeNode* rt, TreeNode* u, TreeNode* v) {
    Splay(u, NULL);
    Splay(v, u);
    DeleteTree(v->lchild);
    v->lchild = NULL;
}
```
Efficiency of Splay tree

- The number of nodes is $N$
- $m$ operations (insertion, deletion and search). when $m \geq n$ the total time cost is $O(m \log n)$
  - Single operation may not be efficient
  - The average time cost is $O(\log n)$
12.5 Improved binary search tree

Semi-Splay Tree

ZigZig

Semi-ZigZig

Next rotation start from y not x
12.5 Improved binary search tree

Discussion

• What are the applications of splay tree?
• Comparisons among red-black tree, AVL tree and Splay tree
  • Visiting frequency?
  • Relationship between tree structure and input data?
  • Which is better from the statistics?
  • Which is easier to implement?
Data Structures
and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpck.pku.edu.cn/pkujpk/course/sjjg/
Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)