Introduction

- Balanced binary search tree (BST)
  - $O(\log n)$ per operation
  - Unbalanced after some insertions and deletions
    - Skinny tree
- Self-organized trees
  - Restructuring the tree according to the operations
  - State-based algorithm and memoryless algorithm

Outline

- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion
Introduction (Cont’d)

Splay tree
- Similar to Move-to-Root algorithm
  - [BRAIN ALIEN and IAN MUNRO, 1978]
    - Both of them move the accessed node to the root
  - Different from simple exchange algorithm
    - Rotate the accessed node with its parent

Splaying rules
- Each rule consists of a template and a result tree
- Both template and result are binary trees containing the same number of nodes
- Each rule is either a terminal rule or non-terminal rule
- Bottom-up splaying
- 3 different splaying rules
Splaying (Cont’d)

- Rule 3 (ZIG-ZAG rule) – non-terminal rule

  ![Diagram showing ZIG-ZAG rule with access paths for a, b, c, and d]

- Rule 1 (ZIG rule) – terminal rule

  (This rule is applied at most once depending the access path is odd or even)

  ![Diagram showing ZIG rule with access paths for a, b, and c]

Splaying (Cont’d)

- Example of splaying (access a)

  ![Diagram showing an example of splaying with access a]

- Rule 2 (ZIG-ZIG rule) – non-terminal rule

  ![Diagram showing ZIG-ZIG rule with access paths for a, b, c, and d]
Cost of Splaying (Cont’d)

- \( w(x) \) : weight of a node \( x \) in the tree, a fixed but arbitrary value
- \( T \) : denotes the whole splay tree
- \( T_x \) : denotes the tree rooted at \( x \)
- \( y \in T_x \) : means \( y \) is a node in the tree \( T_x \)
- \( \text{size}(x) = \sum_{y \in T_x} w(y) \)
- \( \text{Rank}(x) = \log(\text{size}(x)) \)
- Potential function (\( \phi \)) for splay tree = \( \sum_{x \in T} \text{rank}(x) \)

Cost of Splaying

- Not interested in the cost of each operation
- Interested in the cost of a sequence of accesses, we use \textit{amortized cost}
  - Theoretical cost of a given sequence of operations
  - Consider the change of potential of the tree
- Potential function
  - A measure of a data structure configuration or state
Cost of Splaying (Cont’d)

- Let $h$ be the height of the template, e.g. $h=2$

Amortized cost is defined by

$$A = \Phi_{\text{end}} - \Phi_0 + 1$$

Cost of replacing the template with the result tree

Potential of the tree after applying the splaying rule

Potential of the tree before applying the splaying rule

- Let $\Delta \Phi = \Phi_{\text{end}} - \Phi_0 \Rightarrow A = \Delta \Phi + 1$

Cost of Splaying (Cont’d)

- $h+1$ nodes in the template, because the template must be a path
- Because $\text{rank}_+(\text{top}_+) = \text{rank}(\text{top})$, we only consider $h$ nodes

$$\Delta \Phi = \sum_{y \in R} \text{rank}_+(y) - \sum_{y \in T} \text{rank}_-(y)$$

$y$: nodes in the template
$T$: the template tree
$R$: the result tree
$\text{rank}_+(y)$: the rank of node $y$ after applying the rule
$\text{rank}_-(y)$: the rank of node $y$ before applying the rule

Only the ranks of the nodes in the template will be changed, implies

Cost of Splaying (Cont’d)

- Access bot
- Rank$_{(\text{top})}$: the maximum rank after applying the rule
- Rank$_{(\text{bot})}$: the minimum rank before applying the rule
To derive a better upper bound, consider two cases

Case 1: the result of the rule is not a path

Case 2: the result of the rule is a path

Consider the size of top⁺, bot⁺ and bot⁻ (note that bot⁺ and bot⁻ are disjointed in the result tree)

- $\text{size}_w(\text{top}_w) \otimes \text{size}_w(\text{bot}_w) \leq \text{size}_w(\text{bot}_w)$ by inequality $(a \otimes b)^2 \leq 4ab$
- $(\text{size}_w(\text{top}_w))^2 \otimes 4\text{size}_w(\text{bot}_w) \geq \text{size}_w(\text{bot}_w)$
- $2\text{rank}_w(\text{top}_w) \otimes \text{rank}_w(\text{bot}_w) \leq \text{rank}_w(\text{bot}_w) \leq 2$

After taking log

Because $y_1$ and $y_2$ are the children of top⁺, implies

- $\text{size}_w(\text{top}_w) \otimes \text{size}_w(y_1) \geq \text{size}_w(y_2)$ by inequality $(a \otimes b)^2 \leq 4ab$
- $(\text{size}_w(\text{top}_w))^2 \otimes 4\text{size}_w(y_1) \geq \text{size}_w(y_2)^2$
- $2\text{rank}_w(\text{top}_w) \otimes \text{rank}_w(y_1) \geq \text{rank}_w(y_2) \geq 2$
- $h(\text{rank}_w(\text{top}_w) - \text{rank}_w(\text{bot}_w)) \leq 2$

The last step is because the result tree contains $y_1$ and $y_2$, therefore they are overestimate by the rank(\text{top}_w)
Cost of Splaying (Cont’d)

- The above inequality is only for applying one splaying rule
- If we repeat to apply the splaying rule until the accessed node becomes the root
- Then we just sum up the above equation for applying different splaying rules to find the upper bound for amortized cost per access
- All terms in the intermediate trees will be cancelled out

Cost of Splaying (Cont’d)

- So we only concern the change of potential for the initial tree and the result tree
- Now let \( A_{access\ node} \) be the amortized cost for splaying a node \( i \) to the root

\[
A_{access\ node} \leq 3 \log(\text{size}(r)/\text{size}(i)) + 1
\]

Because the height of the splaying template is at most 2

\[
\text{Rank of the initial access node} \quad \text{Rank of the root}
\]

If the access path is odd, then rule 1 need to be applied

\[
3 \log(\text{size}(r)/\text{size}(i)) + 1
\]

\[
O(\log(\text{size}(r)/\text{size}(i)))
\]

Cost of Splaying (Cont’d)

- In \( h(\text{rank}_a(\text{top}_a) - \text{rank}_a(\text{bot}_a)) \), it overestimates \( \text{rank}_a(\text{bot}_a) \) by \( \text{rank}_a(\text{top}_a) \), if we add \( \text{rank}_a(\text{top}_a) - \text{rank}_a(\text{bot}_a) \) to the right hand side and combine with \( 2 \text{rank}_a(\text{top}_a) @ \text{rank}_a(\text{bot}_a) \). We get

\[
( h + 1)(\text{rank}_a(\text{top}_a) - \text{rank}_a(\text{bot}_a)) \leq \Delta \Phi + 3
\]

Combine case 1 and case 2, we have

\[
( h + 1)(\text{rank}_a(\text{top}_a) - \text{rank}_a(\text{bot}_a)) \leq \Delta \Phi + 3
\]

Since amortized cost \( (A) \) after applying one splaying rule \( = \Delta \Phi + 1 \), implies

\[
( h + 1)(\text{rank}_a(\text{top}_a) - \text{rank}_a(\text{bot}_a)) \leq \Delta \Phi + 3
\]

[ASHOK SUBRAMANIAN, 1996]
The total operation cost ($t_{\text{total}}$) over $m$ accesses

Recall that

amortized cost = actual operation cost + change of potential

so we have

$$A_j = t_j + \Phi_j - \Phi_{j-1}$$

$A_j$: the amortized cost for the $j$th access
$t_j$: the actual operation cost for the $j$th access
$\Phi_j$: the potential before the $j$th access
$\Phi_j$: the potential after the $j$th access

The largest potential drop ($\Phi_0 - \Phi_m$) is

where $w(i)$ is the weight of node $i$ and

So we have

$$t_{\text{total}} \leq \sum_{j=1}^{m} (3\log(size(r)/w(i_j)) + 1) + \sum_{i=1}^{n} \log(W/w(i))$$

size($r$) = $W$ because $r$ is the root

So total operation cost ($t_{\text{total}}$) over $m$ operations is:

$$O((m+n)\log(n) + m)$$

$n$: the number of nodes in the tree
$m$: the number of accesses
Cost of Splaying (Cont’d)

Proof of Static Optimality Theorem:
Assign a weight of \( q(i)/m \) to node \( i \), then

\[
\text{cost} \leq \sum_{i=1}^{n} (3 \log(W/w(i)) + 1) + \sum_{i=1}^{n} \log(W/w(i))
\]

\[ W = 1, w(i) = q(i)/m \]

Because each node is accessed at least once, so the last term can be dropped

\[ t_{\text{total}} = \sum_{j=1}^{m} (3 \log(W/w(j)) + 1) + \sum_{i=1}^{n} \log(1/(1/n)) \]

\[ = \sum_{j=1}^{m} (3 \log(1/(1/n)) + 1) + \sum_{i=1}^{n} \log(1/(1/n)) \]

\[ = m * 3 \log(n) + m + n \log(n) \]

\[ = O((m+n) \log(n) + m) \]

Outline

- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion

Cost of Splaying (Cont’d)

Proof of balance theorem:
Assign a weight of \( 1/n \) to each node in the tree, then

\[
\text{cost} \leq \sum_{i=1}^{m} (3 \log(W/w(i)) + 1) + \sum_{i=1}^{n} \log(1/(1/n)) \]

\[ \text{Because } w(i) = 1/n, \text{ and } W = 1 \]

Static Optimality Theorem:
[SLEATOR and TARJAN, 1985]
If every item is accessed at least once, then the total access cost \( (t_{\text{total}}) \) is

\[
O \left( \sum_{i=1}^{n} q(i) \log \left( m/q(i) \right) + m \right)
\]

\( n \): the number of nodes in the tree
\( m \): the number of accesses
\( q(i) \): the number of accesses to node \( i \), and \( \sum_{i=1}^{n} q(i) = m \)
Splay Tree Operations

- **Split(i,t):** split the tree t at the node i

![Diagram](image)

- **Join(t₁,t₂):** join a tree t₁ with another tree t₂, all the elements in t₂ are greater than t₁

![Diagram](image)

Splay Tree Operations

- **Access (i,t):** access the node i in the tree t
  - If node i is in the tree t,
    - First go down the tree, and locate the node
    - Splay the node i to the root
    - Amortized cost (A): \( A = 3 \log \left( \frac{\text{size}(i)}{w(i)} \right) + 1 \)
  - If node i not in the tree t,
    - First go down the tree, and locate a node i- or i+, where i- is the node preceding node i, i+ is the node following i
    - Splay i- or i+ to the root
    - Amortized cost (A):
      \[ A \leq 3 \log \left( \frac{W_i}{\min\{w(i-), w(i+)) \} \right) + 1 \]

- **Split(i,t):** split the tree t at the node i
  - Splay node i in t
  - Cut the left link or right link of node i

![Diagram](image)

- **Amortized Cost (A) of split operation:**
  - If i is in the tree t, then
    \[ A \leq 3 \log \left( \frac{w_i}{w(i)} \right) + O(1) \]
    
    Cost for splaying node i to the root of t
    Cost for cut the left or right link of node i
  - If i is not in the tree t, then
    \[ A \leq 3 \log \left( \frac{w_i}{\min\{w(i-), w(i+)) \} \right) + O(1) \]
    
    Cost for splaying node i- or i+ to the root of t
    Cost for cut the left or right link of node i- or i+

- **Amortized Cost of join operation:**
  \[ A \leq 3 \log \left( \frac{W_{t1}}{w(i)} \right) + O(1) \]
  
  Cost for splaying the node i in t₁
  Cost for making t₂ be the right child of i
### Splay Tree Operations

- **Delete**\((i, t)\): delete node \(i\) from the tree \(t\)

- **Insert**\((i, t)\): insert a node \(i\) into the tree \(t\)

### Amortized cost (A) of delete operation:

\[
A \leq 3\log \left( \frac{W_i}{w(i)} \right) + 3\log \left( \frac{W_i - w(i)}{w(i)} \right) + O(1)
\]

- **Cost for access node** \(i\) in tree \(t\)
- **Cost for join the two subtree** \(t_1\) and \(t_2\)
- **Cost for cut the links in the intermediate tree**

[SLEATOR and TARJAN, 1985]

### Amortized Cost (A) of insert operation:

\[
A \leq 3\log \left( \frac{W_T - w(i)}{w(i)} \right) + \log \left( \frac{W_T}{w(i)} \right) + O(1)
\]

- **Cost for splitting the tree** \(t\), because tree \(t\) does not contain \(i\), so the total weight of \(t = (W_T - w(i))\)
- **Change in potential for the tree before the insert of** \(i\) (but after the split step)
- **Some constant cost for linking the two trees in the final step**

[SLEATOR and TARJAN, 1985]
Application of Splay Tree

- Example of splaying in a lexicographic splay tree

When a string is accessed, we just splay the string character by character.
Most frequent accessed string will be near the root.
The tree store the prefix for the string, so this reduce the redundancy.

Outline

- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion

Application of Splay Tree

Lexicographic search tree
- Circle and square are nodes, square is terminal node of a string
- The solid lines form the binary trees and they represent different strings
- The dashed lines represent a continuous string
- The graph consists the words: “at”, “as”, “bat”, “bats”, “bog”, “boy”, “day”, “he”
Conclusion

- Splaying
  - 3 different rules
  - Reduce the tree height
  - Move the frequently accessed node near the root

- The amortized and actual cost of splaying
  - Use of potential function
  - Balance and static optimality theorem

Outline

- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion

Application of Splay Tree

- Other application – data compression
  - Dynamic Huffman coding [D.W. JONES, 1988]
    - Digital search tree
    - Good compression scheme for image data