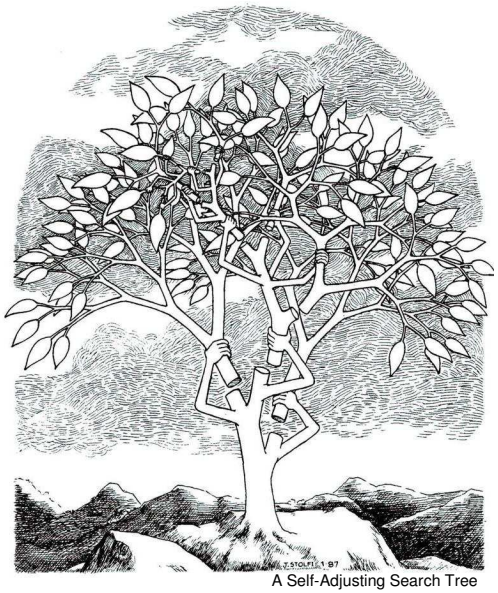
Splay Tree

by HUNG Lau Yung (Simon)

COMP670K On-line Algorithm, 4th November, 2003

Outline

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



A Self-Adjusting Search Tree
By Jorge Stolfi

Introduction

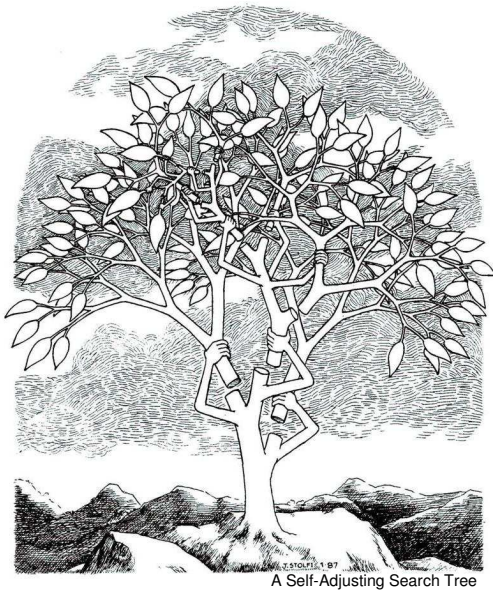
- Balanced binary search tree (BST)
  - O(log n) per operation
  - Unbalanced after some insertions and deletions
    - Skinny tree
- Self-organized trees
  - Restructuring the tree according to the operations
  - State-based algorithm and memoryless algorithm

Introduction (Cont'd)

- Splay trees
  - Invented by Sleator and Tarjan
  - Memoryless algorithm
  - Different from state-based algorithm
    - Dynamic monotone trees, WPL trees and D-trees
    - Store information about the accesses to the nodes

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

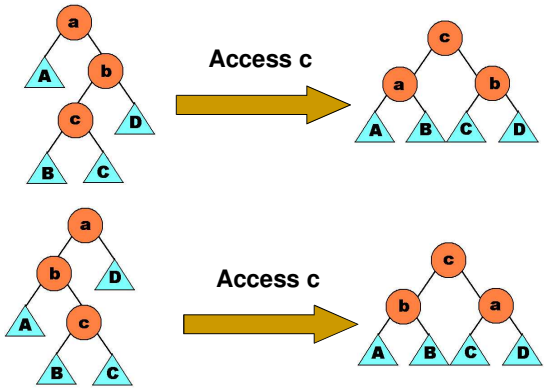
- Splaying rules
  - Each rule consists of a template and a result tree
  - Both template and result are binary trees containing the same number of nodes
  - Each rule is either a terminal rule or non-terminal rule
  - Bottom-up splaying
  - 3 different splaying rules

Introduction (Cont'd)

- Splay tree
  - All operations are splaying
    - Still keep symmetric order
  - Reduce the height of a BST
  - Good performance for skewed data accesses
  - On-line algorithm
    - Restructuring after each access

Splaying (Cont'd)

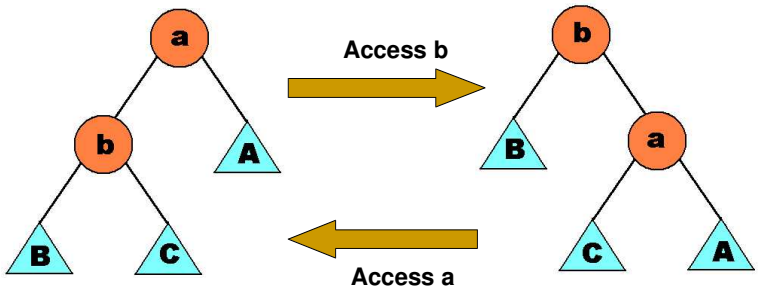
- Rule 3 (ZIG-ZAG rule) – non-terminal rule



Splaying (Cont'd)

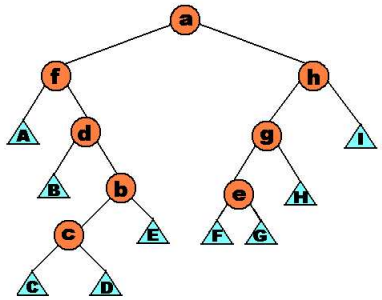
- Rule 1 (ZIG rule) – terminal rule

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



Splaying (Cont'd)

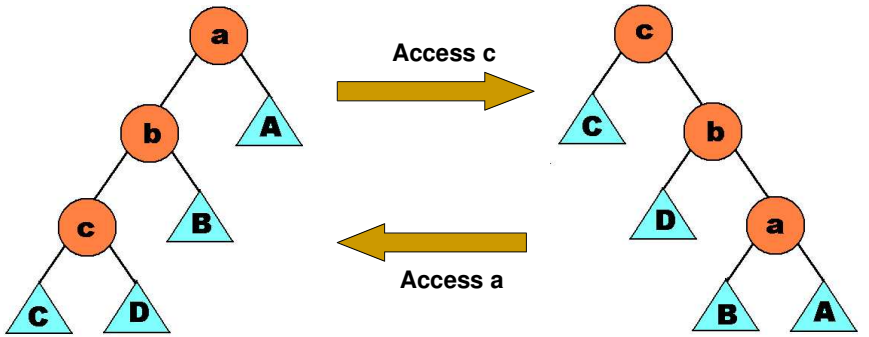
- Example of splaying (access a)



Splay tree demo

Splaying (Cont'd)

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

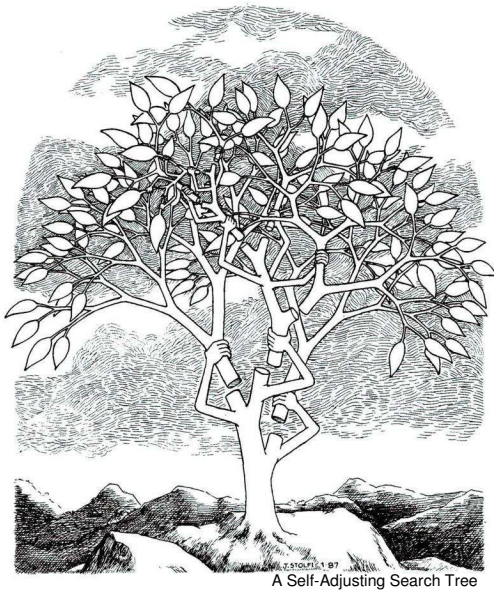


Cost of Splaying (Cont'd)

- 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 ∈ Tx : means y is a node in the tree Tx
- size(x) = $\sum_{y \in T_x} w(y)$
- Rank(x)=log(size(x))
- Potential function (Φ) for splay tree = $\sum_{x \in T} rank(x)$

Outline

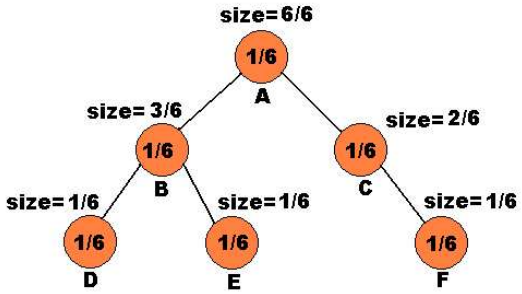
- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion



A Self-Adjusting Search Tree
By Jorge Stolfi

Cost of Splaying (Cont'd)

- Example of potential of a tree, and all nodes have the same weights which are 1/6
  - e.g. the rank(C) = log(2/6)=-1.5850
  - Potential (Φ)
    = 3*log(1/6)+log(2/6)
    +log(3/6)+log(6/6)
    = -10.33985

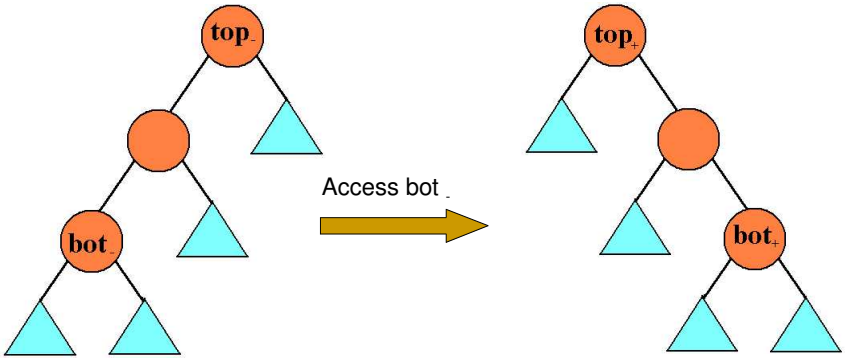


Cost of Splaying

- Not interested in the cost of each operation
- Interested in the cost of a sequence of accesses, we use 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



Cost of Splaying (Cont'd)

- Amortized cost is defined by

  actual operation cost + change of potential
- Let A be the amortized cost of applying a splaying rule (here we only consider the non-terminal rules first)

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

Potential of the tree after applying the splaying rule

Potential of the tree before applying the splaying rule

Cost for replacing the template with the result tree

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

Cost of Splaying (Cont'd)

- h+1 nodes in the template, because the template must be a path
- Because $rank_+(top_+) = rank_-(top_-)$, we only consider h nodes

$$\ddot{A}\ddot{O} \leq h(rank_{\text{☞}}(top_{\text{☞}}) - rank_{\text{✍}}(bot_{\text{✍}}))$$

$Rank_+(top_+)$: the maximum rank after applying the rule

$Rank_-(bot_-)$: the minimum rank before applying the rule

Cost of Splaying (Cont'd)

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

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

y : nodes in the template

T : the template tree

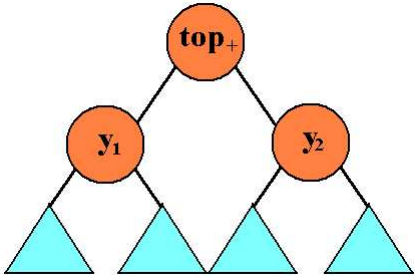
R : the result tree

$rank_+(y)$: the rank of node y after applying the rule

$rank_-(y)$: the rank of node y before applying the rule

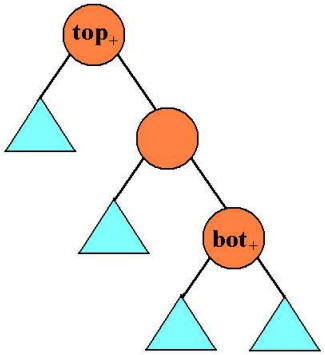
Cost of Splaying (Cont'd)

- To derive a better upper bound, consider two cases
- Case 1: the result of the rule is not a path



Cost of Splaying (Cont'd)

- Case 2: the result of the rule is a path



Cost of Splaying (Cont'd)

- Consider the size of top_+, bot_+ and bot_- (note that bot_+ and bot_- are disjointed in the result tree)

$$size_{\square}(top_{\square}) \otimes size_{\square}(bot_{\square}) \leq size_{\square}(bot_{\square})$$

by inequality $(a \leq b)^2 \otimes 4ab$

$$\Rightarrow (size_{\square}(top_{\square}))^2 \otimes 4size_{\square}(bot_{\square})size_{\square}(bot_{\square})$$

After taking log

$$\Rightarrow 2rank_{\square}(top_{\square}) \otimes rank_{\square}(bot_{\square}) \leq rank_{\square}(bot_{\square}) \leq 2$$

Cost of Splaying (Cont'd)

- Because y_1 and y_2 are the children of top_+, implies

$$size_{\square}(top_{\square}) \otimes size_{\square}(y_1) \leq size_{\square}(y_2)$$

by inequality $(a \leq b)^2 \otimes 4ab$

$$\Rightarrow (size_{\square}(top_{\square}))^2 \otimes (size_{\square}(y_1) \leq size_{\square}(y_2))^2$$

After taking log

$$\Rightarrow (size_{\square}(top_{\square}))^2 \otimes 4size_{\square}(y_1)size_{\square}(y_2)$$

$$\Rightarrow 2rank_{\square}(top_{\square}) \otimes rank_{\square}(y_1) \leq rank_{\square}(y_2) \leq 2$$

$$\Rightarrow h(rank_{\square}(top_{\square}) - rank_{\square}(bot_{\square})) \leq 2 \otimes \clubsuit\spadesuit$$

The last step is because the result tree contains y_1 and y_2, therefore they are overestimate by the $rank_+(top_+)$

Cost of Splaying (Cont'd)

$$(\log(\text{size}(r)) - \log(\text{size}(\text{bot}_+(i)))) \geq \Delta\Phi$$
$$(\log(\text{rank}_+(\text{top}_+)) - \text{rank}_+(\text{bot}_+))$$

- 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)

- In ÄÖ h(rank(top) rank(bot)) , it overestimates rank(bot) by rank(top), if we add rank(top)-rank(bot) to the right hand side and combine with 2rank(top) ®rank(bot) rank(bot) 2 We get

$$(\log(\text{size}(r)) - \log(\text{size}(\text{bot}_+(i)))) - 2 \geq \Delta\Phi$$
$$(\log(\text{rank}_+(\text{top}_+)) - \text{rank}_+(\text{bot}_+))$$

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

Because the height of the splaying template is at most 2

Rank of the initial access node

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

$$A_{access\_node} \leq 3(\text{rank}(r) - \text{rank}(i)) + 1$$

Rank of the root

† $3\log(\text{size}(r)/\text{size}(i))$ 1

† $O(\log(\text{size}(r)/\text{size}(i)))$

Cost of Splaying (Cont'd)

- Combine case 1 and case 2, we have

$$(\log(\text{size}(r)) - \log(\text{size}(\text{bot}_+(i)))) \geq \Delta\Phi + 2$$
$$(\log(\text{rank}_+(\text{top}_+)) - \text{rank}_+(\text{bot}_+))$$

- Since amortized cost (A) after applying one splaying rule = ΔΦ+1, implies

$$(\log(\text{size}(r)) - \log(\text{size}(\text{bot}_+(i)))) \geq \Delta\Phi$$
$$(\log(\text{rank}_+(\text{top}_+)) - \text{rank}_+(\text{bot}_+))$$

[ASHOK SUBRAMANIAN,1996]

Cost of Splaying (Cont'd)

- The largest potential drop (Φ_0-Φ_m) is $\sum_{i=1}^{n} \log(W/w(i))$ where w(i) is the weight of node i and $W = \sum_{i=1}^{n} w(i)$ So we have

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

size(r)=W because r is the root

Cost of Splaying (Cont'd)

- The total operation cost (t_{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 jth access

    t_i :the actual operation cost for the jth access

    Φ_{j-1}: the potential before the jth access

    Φ_j : the potential after the jth access

Cost of Splaying (Cont'd)

- Balance Theorem: [SLEATOR and TARJAN,1985] the total access cost (t_{total}) 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)

- So total operation cost (t_{total}) over m operations is:

$$t_{total} = \sum_{j=1}^{m} t_j = \sum_{j=1}^{m}(A_j + \Phi_{j-1} - \Phi_j)$$
$$\Rightarrow t_{total} = \sum_{j=1}^{m} A_j + \Phi_0 - \Phi_m$$
$$\Rightarrow t_{total} \le \sum_{j=1}^{m}(3\log(size(r)/size(i_j))+1) + \Phi_0 - \Phi_m$$
$$\Rightarrow t_{total} \le \sum_{j=1}^{m}(3\log(size(r)/w(i_j))+1) + \Phi_0 - \Phi_m$$

The initial size of the jth accessed node

Because $size(i_j)$ is at least $w(i_j)$

Cost of Splaying (Cont'd)

Proof of balance theorem:

Assign a weight of 1/n to each node in the tree, then

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

Because $w(i_j)=w(i)=1/n$, and $W=1$

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

$$= m*3\log(n)+m+n*\log(n)$$

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

Cost of Splaying (Cont'd)

Proof of Static Optimality Theorem:

Assign a weight of q(i)/m to node i, then

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

W=1,w(i)=q(i)/m

$$= \sum_{j=1}^{m}(3\log(1/(q(i_j)/m))+1)+\sum_{i=1}^{n}\log(1/(q(i)/m))$$

$$= \sum_{j=1}^{m}(3\log(m/q(i_j)))+m+\sum_{i=1}^{n}\log(m/q(i))$$

$$= \sum_{i=1}^{n}q(i)(3\log(m/q(i)))+m+\sum_{i=1}^{n}\log(m/q(i))$$

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

$$= O(\sum_{i=1}^{n}q(i)(\log(m/q(i)))+m)$$

Cost of Splaying (Cont'd)

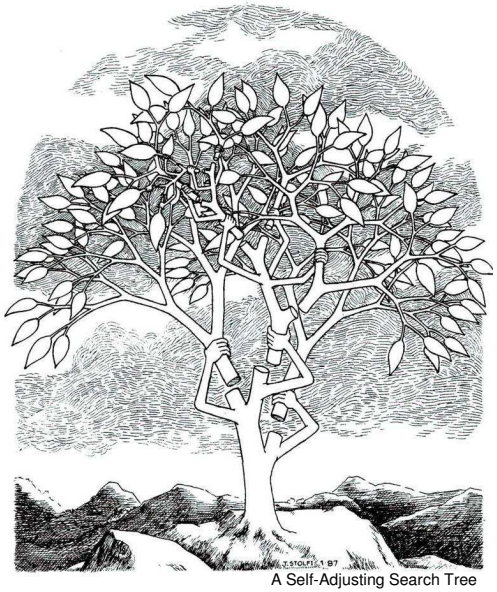
- Static Optimality Theorem:
  [SLEATOR and TARJAN,1985]
  If every item is accessed at least once, then the total access cost (t_{total}) is

$$O(\sum_{i=1}^{n} q(i)\log(m/q(i))+m)$$

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

Outline

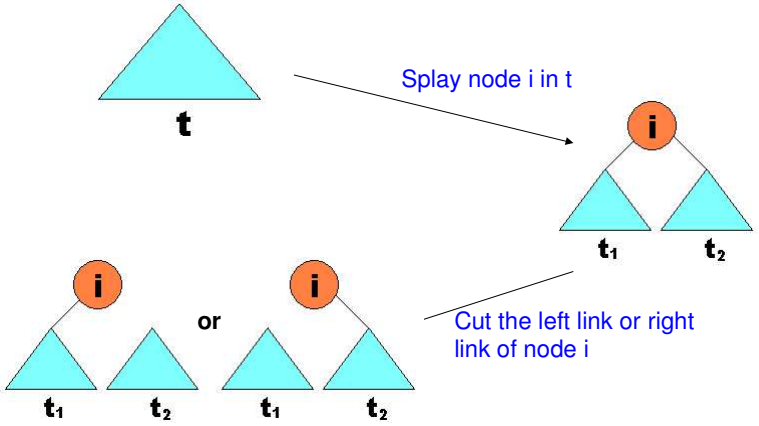
- Introduction
- Splaying
- Cost of splaying
- Splay tree operations
- Application of splay tree
- Conclusion



A Self-Adjusting Search Tree
By Jorge Stolfi

Splay Tree Operations

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



Splay Tree Operations

- In the this section, we use W_t to represent the total weights of a tree t.
- 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): $A = 3\log(\frac{size(r)}{size(i)}) + 1$

      $\leq 3\log(\frac{W_t}{w(i)}) + 1$
  - 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):

      $A \leq 3\log(\frac{W_t}{\min\{w(i-), w(i+)\}}) + 1$

Splay Tree Operations

- Amortized Cost (A) of split operation:
  - If i is in the tree t, then

    $A \leq 3\log(\frac{Wt}{w(i)}) + O(1)$

    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

    (Recall that i- is the node preceding i, i+ is the node following i)

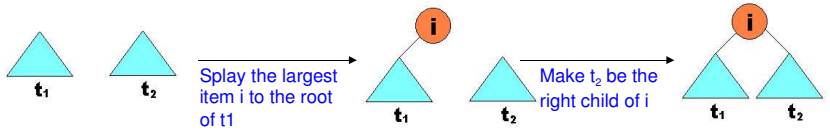
    $A \leq 3\log(\frac{W_t}{\min\{w(i-), w(i+)\}}) + O(1)$

    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+

Splay Tree Operations

- Join(t_1,t_2): join a tree t_1 with another tree t_2, all the elements in t_2 are greater than t_1



$A \leq 3\log(\frac{W_{t1}}{w(i)}) + O(1)$

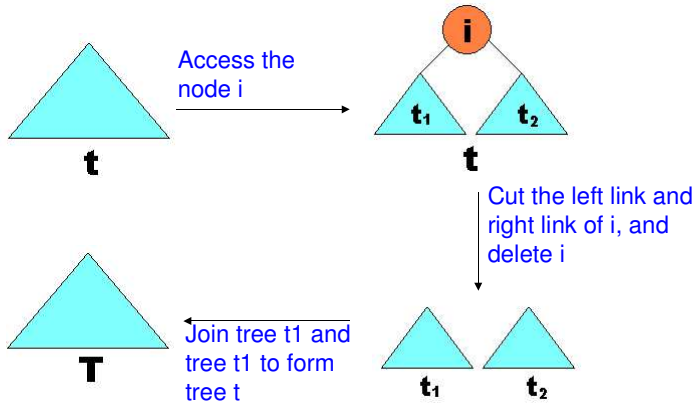
- Amortized Cost of join operation:

  Cost for splaying the node i in t_1

  Cost for making t2 be the right child of i

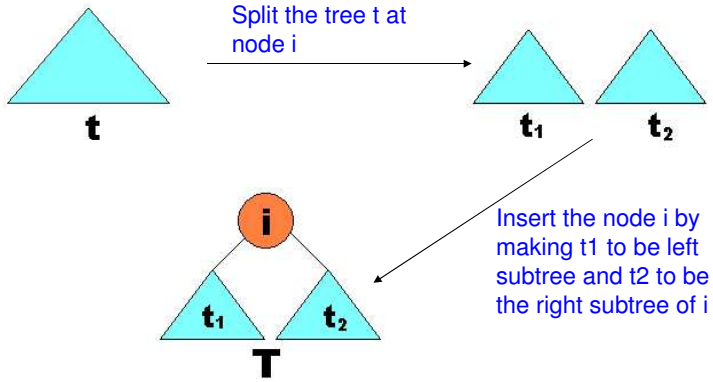
Splay Tree Operations

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



Splay Tree Operations

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



Splay Tree Operations

- Amortized cost (A) of delete operation:

$$A \le 3\log\left(\frac{W_t}{w(i)}\right) + 3\log\left(\frac{W_t - w(i)}{w(i-)}\right) + O(1)$$

Cost for access node i in tree t

Cost for join the two subree t_1 and t_2

Cost for cut the links in the intermediate tree

[SLEATOR and TARJAN, 1985]

Splay Tree Operations

- Amortized Cost (A) of insert operation:

$$A \le 3\log\left(\frac{(W_T - w(i))}{\min\{w(i-), w(i+)\}}\right) + \log\left(\frac{W_T}{w(i)}\right) + O(1)$$

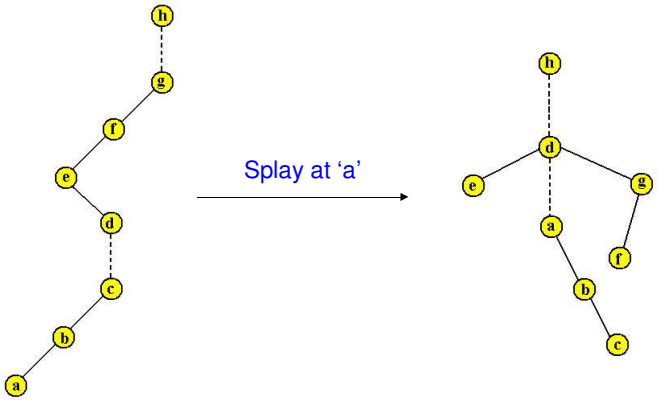
The cost for splitting the tree t, because tree t does not contain i, so the total weight of t =(W_T-w(i))

The change in potential for the tree before the insert of i (but after the split step) and the after the insert of i

Some constant cost for linking the two trees in the final step

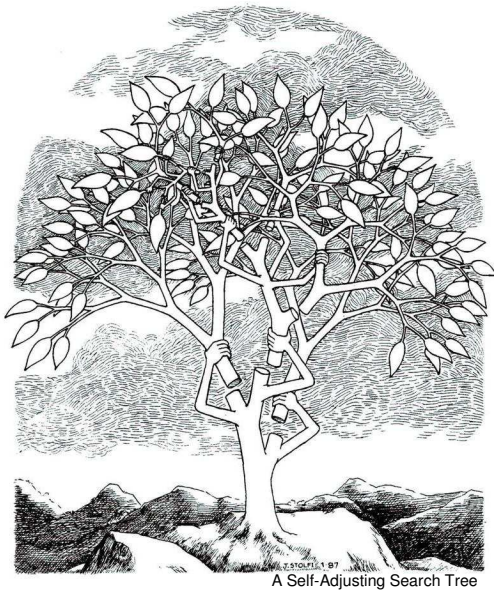
Application of Splay Tree

- Example of spalying in a lexicographic splay tree



Outline

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

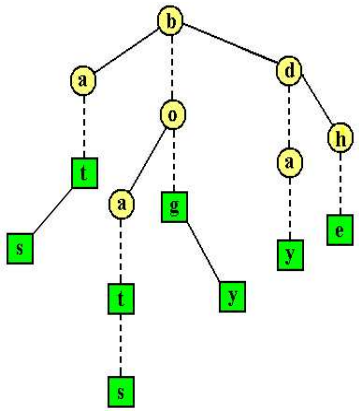


Application of Splay Tree

- 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

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

Application of Splay Tree

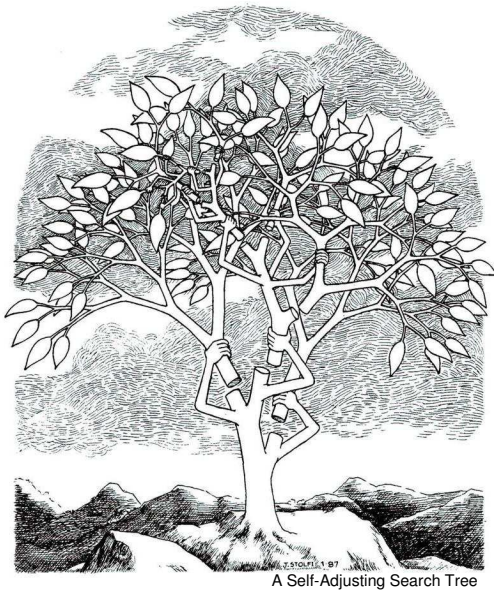
- Other application – data compression
  - Dynamic Huffman coding [D.W. JONES,1988]
    - Digital search tree
    - Good compression scheme for image data

Conclusion

- Splay tree operation
  - Access
  - Join
  - Split
  - Insert
  - Delete
- Application of splay tree
  - Lexicographic splay tree

- End -

Outline



A Self-Adjusting Search Tree
By Jorge Stolfi