

The Probabilistic method

1. If a random object in a set satisfies some property with positive probability then there is an object in that set that satisfies that property.

2. Any random variable assumes at least one value that is no smaller than its expectation and at least one value that is no greater than its expectation.

The Geometric Distribution

Assume that an experiment has probability p for success $1 - p$ for failure. How many trials we need till the first success.

$$Pr(X = i) = (1 - p)^{i-1}p.$$

X has a Geometric distribution with parameter p
 $X \sim G(p)$.

Assume that X get values in \mathcal{N} . We have the following useful identity:

$$E[X] = \sum_{i \geq 0} i Pr(X = i) = \sum_{i \geq 1} Pr(X \geq i)$$

Let $X \sim G(p)$.

$$\begin{aligned} E[X] &= \sum_{i \geq 1} Prob(X \geq i) = \sum_{i \geq 1} (1 - p)^{i-1} \\ &= \frac{1}{1 - (1 - p)} = \frac{1}{p} \end{aligned}$$

Theorem 1. *In the two-level hashing scheme the expected amount of storage required for all the secondary hash tables is:*

$$E \left[\sum_{j=0}^{m-1} n_j^2 \right] < 2n$$

Proof. $E \left[\sum_{j=0}^{m-1} n_j^2 \right] = E \left[\sum_{j=0}^{m-1} (n_j + 2 \binom{n_j}{2}) \right]$

$$= E \left[\sum_{j=0}^{m-1} n_j \right] + 2E \left[\sum_{j=0}^{m-1} \binom{n_j}{2} \right]$$
$$\leq n + 2 \binom{n}{2} 1/m = n + 2n(n-1)/2n < 2n \quad \square$$

Corollary 1. *In the two-level hashing scheme the probability that the total storage used for secondary hash tables exceeds $4n$ is $< 1/2$.*

Splay Trees

Binary search trees with $O(\log n)$ **amortized** cost of searching, insertion, deletion and more.

Unlike many other balanced binary tree schemes such as 2-3 trees, red-black trees, or AVL trees it is not necessary to rebalance the structure after every operation. Rather it happens automatically !

Also no additional information needed to be stored in the nodes for balancing.

Data (keys) is represented at all nodes of the splay tree in an **inorder** fashion. We will assume that the keys are all distinct.

Operations

Operations on a splay tree: SEARCH, INSERT, DELETE, JOIN, SPLIT.

JOIN(S, S') takes two search trees S and S' , where all keys in S are less than S' , and joins them in to a single splay tree.

SPLIT(i, S) splits the splay tree S into two new splay trees S' and S'' such that $x \leq i \leq y$ for all $x \in S'$ and $y \in S''$.

All the above operations is done using a basic operation called SPLAY.

Splay Operation

$\text{SPLAY}(i, S)$ reorganizes the splay tree S so that key i is at the root if $i \in S$ and otherwise the new root is either $\max\{k \in S \mid k < i\}$ or $\min\{k \in S \mid k > i\}$.

All other operations can be performed with a constant number of splay operations in addition to a constant number of other low-level operations such as pointer manipulations and comparisons.

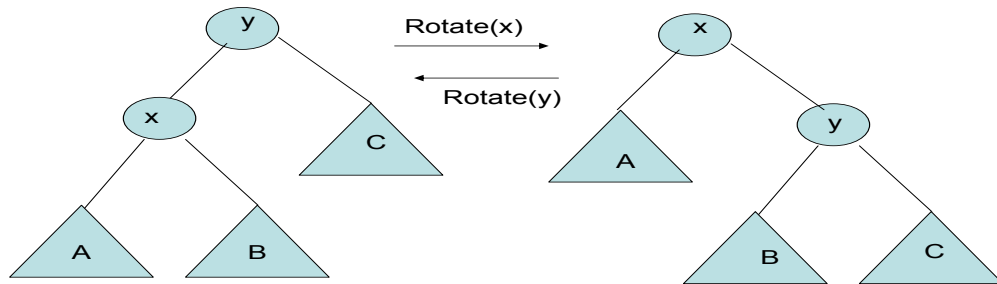
For example, to do $\text{JOIN}(S, S')$, first call $\text{SPLAY}(+\infty, S)$ and then make S' the right subtree.

To do $\text{DELETE}(i, S)$, call $\text{SPLAY}(i, S)$ to bring i to the root if it is there, then remove i and call JOIN to merge the left and right subtrees.

Implementation of Splaying

Splay operation consists of one or more **rotation** steps.

Given a binary tree S and a node x with parent y , the operation $\text{ROTATE}(x)$ is:



Rotation preserves the binary search tree property.

3 Cases

To implement $\text{SPLAY}(x, S)$ we rotate x up until it becomes the root. Depending on the relationship between x to its parent and grandparent, we distinguish 3 different cases:

Case 1: If x has a parent but no grandparent we just $\text{ROTATE}(x)$.

Case 2: If x has a parent $p(x)$ and a grandparent, and if x and $p(x)$ are either both left children or both right children, we first $\text{ROTATE}(p(x))$ and then $\text{ROTATE}(x)$.

Case 3: If x has a parent $p(x)$ and a grandparent, and if one of x , $p(x)$ is a left child and the other is a right child, we first $\text{ROTATE}(x)$ and then $\text{ROTATE}(x)$ again.