Sets and maps

Lecture outline

Pierre Senellart

8 November 2018

Call the roll. Preliminary questions?

1 Introduction

- Sets, maps
- Corresponding data structures in Python, C++, Java
- Simple implementation: array; size and space complexity

2 Hash tables (11)

- Hashing space of size m, hash function
- Hash table: buckets, chaining for collisions
- Get, Insert, Delete
- Complexity in the worst case; complexity under uniformity assumption in terms of load factor $\alpha = \frac{n}{m}$
- Designing hash functions
  - Using modulo arithmetics (careful of patterns)
  - Universal hashing (the number of functions in H s.t. $h(k) = h(l)$ for any distinct k, l is $\leqslant \frac{|H|}{m}$); complexity
  - Choosing a set of universal functions: $h_{a,b} = ((ak + b) \bmod p) \bmod m$ for $p > m$ prime number, $a \in \mathbb{F}_p \neq 0$, $b \in \mathbb{F}_p$
- Open addressing:
  - linear probing $h(k,i) = (h'(k) + i) \bmod m$; disadvantages
  - double hashing $h(k,i) = (h_1(k) + i \times h_2(k)) \bmod m$ with $(h_2(k), m)$ relatively prime; number of distinct hashing sequences
- Dynamic hash tables:
  - Size doubling
  - Linear hashing, only mentioned

3 Red–black binary trees (13)

- Reminder on (balanced) binary search trees
- Red–black trees. Properties of red–black trees:
  - The root is black.
  - If a node is red, its children are black.
  - Same number of black nodes on every path from one node to its descendants with 0 or 1 child.
- Example 13.4 (without the z node)
- Bound: $n \geqslant 2^{\mathrm{bh}(x)}$ and $\mathrm{bh}(x) \geqslant \frac{h-1}{2}$
- Left rotation, right rotation
- Insertion in a red–black tree:
  - Insertion of z at its natural position, red colored
  - Correction of red–black violation between z and its parent:
    * If z is the root, we color it in black.
    * If z's parent is black, nothing to do.
    * If z's parent is red (and thus its grandparent is black):
      · Deal with the case where z's parent is a left child, the other case is symmetric
      · If the uncle of z is red, it is colored in black together with z's parent, and the grandparent is colored in red. Recursively process the grandparent.
      · Else, if z is a right child, we left-rotate z and its parent and consider the new left child of z and move to the following case.
      · z is a left child, we color its parent in black and its grandparent in red, then right-rotate z's parent and z's grandparent. Done.
  - Complexity
- A word on deletion, no detail
- Balanced binary search trees vs hash tables