COMP 3170 - Analysis of Algorithms & Data Structures

Shahin Kamali

Lectures 15 - Feb. 5, 2018
CLRS 6.1, 6.2, 6.3
University of Manitoba
A **priority queue** is an abstract data type formed by a set $S$ of key-value pairs

**Basic operations** include:

- **insert** $(k)$ inserts a new element with key $k$ into $S$
- **get-Max** which returns the element of $S$ with the largest key
- **extract-Max** which returns the element of $S$ with the largest key and delete it from $S$

We are often given the whole data and need to **build** the data structure based on it.

- Any data structure for a priority queue should be **constructed** efficiently.
Priority queue implementation

- What is a good implementation (data structure) for priority queues?
  
- You have seen **binary heaps** before: get-Max runs in $O(1)$ and extract-Max and insert both take $\Theta(\log n)$ for $n$ keys.
  
- Is a balanced binary search tree a good implementation of a priority queue?
  
  - with a little augmentation, get-Max runs in $O(1)$ and extract-Max and insert both can run in $\Theta(\log n)$.
  
- The problem with BSTs: it is costly to build them
  
  - How long does it take to form a BST from a given set of items?
  
  - It takes $\Omega(n \log n)$; otherwise you can sort them in $o(n \log n)$ by building the BST and doing an inoder traverse in $O(n)$.
  
  - We know we cannot comparison-sort in $o(n \log n)$ and hence cannot build the tree in such time.
Binary heaps

- A **heap** is a **tree** data structure.
- For every node $i$ other than the root, we have $key[parent[i]] \geq key[i]$.
- A **binary** heap is a complete binary tree which can be stored using an array.
  - build-heap takes $\Theta(n)$ time
  - insert, extract-Max take $\Theta(\log n)$
  - get-Max takes $O(1)$
Binary heaps

- Suppose multiple priority queues on different servers.
- Occasionally a server must be rebooted, requiring two priority queues to be merged.
- With a typical binary heap, merging requires concatenating arrays and re-running build-heap; this takes $\Theta(n)$.
- When implementing an abstract data type always consider if you need it to be mergable or not.

![Diagram of binary heaps and their merging process]
Rethinking about Data Structure

- We would like to build a data structure for priority queues that:
  - supports insert, extract-Max, get-Max, and build efficiently (as in binary heaps)
  - merging two priority queues takes $o(n)$

- Solution: **bionomial heaps** which are mergable heaps that efficiently support:
  - `insert(H, x)`
  - `extract-Max(H)`
  - `get-Max(H)`
  - `build(A)`

  - `union(H_1, H_2)` (merge)
  - `increase-key(H, x, k)`
  - `delete(H, x)`
Bionominal Trees

- A **bionomial tree** is an ordered tree defined recursively.
  - children of each node have a specific ordering (similar to ‘left’ and ‘right’ child in binary trees).
- The base case for a bionomial tree $B_0$ is a single node.
- To build $B_k$, we take two copies of $B_{k-1}$ and let the first child of the root of the second copy be the root of the first copy.
Fun with Binomial Trees

Fun 1: The children of the root of the binomial tree $B_k$ are the binomial trees $B_{k-1}, \ldots B_0$.

- Induction: assume it is true for all binomial trees $B_i$ with $i \leq k - 1$ (base easily holds).
- The tree $B_k$ has its first child as $B_{k-1}$ (recursive construction).
- With respect to other children, it is a binomial tree $B_{k-1}$ and hence has children $B_{k-2}, \ldots, B_0$ by induction hypothesis.
Fun with Bionomial Trees

- Fun 2: $B_k$ has $2^k$ nodes:
  - The recursion is $N(B_k) = 2N(B_{k-1}), N(B_0) = 1$

- $B_k$ has height $k$:
  - The recursion is $h(B_k) = h(B_{k-1}) + 1$:

- Within $B_k$ there are $\binom{k}{i}$ nodes at depth $i$.
  - The recursion is $ch(k, i) = ch(k - 1, i - 1) + ch(k - 1, i)$
  - Solving this recursion gives $ch(k, i) = \binom{k}{i}$. To get an idea of the proof, note that $\binom{k}{i} = \binom{k-1}{i-1} + \binom{k-1}{i}$