Advanced Data Structures

Lecture 2

Lecture by Dr. Shay Mozes
Typeset by Steven Karas

2016-11-17
Last edited 20:17:26 2016-12-03

1 Overview of Heaps

Operation	Linked List	Binary	Binomial	Fibonacci[1]	Relaxed
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N[1]	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

1.1 Binary Heaps

Binary heaps are a type of balanced binary tree that efficiently implements a total ordering. Heaps are always perfectly balanced trees. As such, the height of the tree is $\Theta(\log n)$

1.1.1 Max-Heap Property

For each node n in a binary heap, the value at n is at least as large as both its children. This is true for all nodes in the heap.

1.1.2 Heap-Maximum

The root of the heap is the largest value in the heap. $\Theta(1)$

1.1.3 Max-Heap-Insert, Heap-Increase-Key

$\Theta(\log n)$. Insert at next open leaf, and swap with parents until no longer larger.

1.1.4 Heap-Extract-Max, Max-Heapify

Removing the last leaf is $\Theta(1)$, and is trivially easy.

To remove the root, swap it with the last leaf, and remove the last leaf.

[1] amortized

Max-Heapify To rebalance a heap, compare the root to each of its children, and swap with the largest child. Recurse until root is not swapped, or is a leaf node.

$\Theta(\log n)$

1.1.5 Build-Max-Heap

Naive heap creation takes at worst $\Theta(nlogn)$, even though for random permutations it's $\Theta(n)$.

By building the heap from the bottom up, we can run Max-Heapify on each leaf node and recurse upwards through the heap.

Number of nodes with height i: $\frac{n}{2^{i+1}}$

1.1.6 Decrease-Key

Changing the priority of a specific element in the heap.

1.1.7 Union

Combine two heaps. If the heap is stored using pointers, this can be done in $O(\log n) * O(\log m)$.

Correctness Each node at height h can move down h levels:

$$\sum_{h=1}^{\log n} \frac{n}{2^h} \leq n \sum_{h=1}^{\infty} 2^h = n \cdot (\frac{1}{2} + \frac{1}{4} + ...) = n$$

1.2 Application: Dijkstra

Better explained elsewhere. CLRS does it wonderfully.

Complexity

$$n \cdot T(\text{Delete-Min}) + m \cdot T(\text{Decrease-Key})$$

Heap	Delete-Min	Decrease-Key	Overall
Binary	TODO	TODO	$O((n+m)\log n)$
Fibonacci	TODO	TODO	$O(n \log n + m)$

1.3 Application: Prim MST

Given a weighted directed graph, we want a minimal set of edges for which the entire graph is connected.

```
for all vertices in G:
  let v.key = +Inf, v.parent = NULL
Let s be a random vertex in G
s.key = 0
Q = priority queue over V wrt key
While Q is not empty:
  u = Q.DeleteMin
  for each neighbor v of u:
```

```
if v.key > w(uv)
  Q.DecreaseKey(v, w(uv))
  v.parent = u
```

Complexity

$$n \cdot T(\text{Delete-Min}) + m \cdot T(\text{Decrease-Key})$$

Note that this is the same as Dijkstra's Algorithm.

1.4 Binomial Trees

A binomial tree is defined recursively as a tree of binary trees.

- Number of nodes = 2^k
- Height = k
- Degree of root = k
- Deleting root yields k binomial trees $B_0, ..., B_{k-1}$

1.5 Binomial Heaps

List of binomial trees that satisfy heap property. 0 or 1

1.5.1 Extract-Min-Key

Minimum key is guaranteed to be in one of the roots. At most $\lfloor \log_2 n \rfloor + 1$ binomial trees.

1.5.2 Union

Set smaller tree root as root of new combined tree, set other as other child. Takes $\log n$ because merging tree of order 0 into a fully packed heap means merging each tree.

1.5.3 Delete-Min

Find min in root list. Remove the root. Merge the children of the deleted root as if they were another heap.

Complexity $O(\log n)$ because $O(1)$ to remove the root, and $O(\log n)$ to merge.

1.5.4 Decrease-Key

Bubble the key up the tree if it's too small.

Complexity $O(\log n)$, because maximum node depth is $\log_2 n$

1.5.5 Insert

Merge in the element as if it were a binomial heap.

Complexity $O(\log n)$

Amortized Complexity $O(1)$ as per a binary counter from last week.

1.6 Fibonacci Heaps

Guidelines Be lazy. Force the user to make us do work. If we already have to work, then do the least amount possible to simplify the data structure.

Design Doubly linked ring of heaps[2]. Keep a reference to the smallest heap.

1.6.1 Union

Combine the two lists of heaps, update the root. $O(1)$

Amortized Complexity Define a potential function $\phi(H) = t(H) =$ the number of roots in the heap.

$$amort(\text{Union}) = O(1) + t(H_1) + t(H_2) - (t(H_1) + t(H_2)) = O(1)$$

1.6.2 Insert

Treat the singular value as a heap, and merge. $O(1)$

Amortized Complexity

$$amort(\text{Insert}) = O(1) + t(H) + 1 - t(H) = O(1)$$

1.6.3 Delete-Min

Delete the minimal root. Put the children of the deleted root into the list.

While finding a new minimum, convert the fibonnacci heap into a binomial heap.

Complexity Worse, and yet...

Amortized Complexity

$$amort(\text{Delete-Min}) = O(1) + t(H) + \begin{array}{c}\#\text{ children of}\\\text{minimal heap}\end{array} + O(\log n) - t(H) = O(\log n)$$

[2]This isn't necessary, but whatever

1.6.4 Decrease-Key

Described in terms of splicing subtrees to become roots. First subtree is ok, second is not.

If the heap property in v is not kept, detach it to become a root. However, if we do this, we may remove too many nodes from a tree (needs to be exponential to the order k). Our solution to this is to mark nodes that have already lost a child, and if we splice off the child of a marked node, we splice off the node itself.

Problem: What happens if we remove the bottom child in a tree full of marked nodes? Solution: ensure that any node loses at most 2 children.

Amortized Complexity Due to this marking, we need to redefine the potential function, and repeat our analysis for all other actions:

$\phi'(H) = t(H) + 2m(H)$ where $m(H) =$ number of marked nodes and $t(H) =$ number of roots

$$amort(\text{Decrease-Key}) = c + c + 2(-c + 1) = O(1)$$

where c is the number of splices performed.

Proof Let v be a node in a binomial tree of degree k with children $y_1, ..., y_k$, ordered by insertion order. We want to show that for any $i \geq 2$, the degree of y_i is at least 2.

Before y_i was attached as a child of v, they have the same degree, meaning that y_i has $i - 1$ children at that moment. Since then, y_i has lost at most 1 child.

Let S_k be the minimal size of a tree of degree k. Note that $S_0 = 1$, and $S_1 \geq 2$.

$$S_k = 2 + S_0 + S_1 + ... + S_{k-2}$$

Note that:

$$S_k \geq F_k = F_{k-1} + F_{k-2} \geq \phi^k$$

where $\phi = \frac{1+\sqrt{5}}{2} =$ is the root of $x^2 - x - 1 = 0$, which because of its connection to Fibonacci is where this heap gets it name from.

The remainder of the proof is in the video, and Shay does a better job of it than I do.