Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
http://www.iiitdm.ac.in
COM 501 Advanced Data Structures and Algorithms

Instructor
N.Sadagopan
Scribe:
Roopesh Reddy
Renjith.P

Fibonacci Heap

Objective: In this lecture we discuss Fibonacci heap, basic operations on a Fibonacci heap such as insert, delete, extract-min, merge and decrease key followed by their *Amortized analysis*, and also the relation of Fibonacci heap with *Fibonacci series*.

Motivation: Is there a data structure that supports operations such as insert, delete, extract-min, merge and decrease key efficiently. Classical min-heap incurs $O(n)$ for merge and $O(\log n)$ for the rest of the operations, whereas, binomial heap incurs $O(\log n)$ for all of the above operations. In asymptotic sense, is it possible to perform these operations in $o(\log n)$? If not, can we think of a data structure that performs maximum number of these operations in $O(1)$ amortized and the rest in $O(\log n)$ in amortized time.

1 Introduction

Fibonacci heap is an unordered collection of rooted trees that obey *min-heap property*. Min-heap property ensures that the key of every node is greater than or equal to that of its parent. The roots of the rooted trees are linked to form a linked list, termed as *Root list*. Also there exists a min pointer that keeps track of the minimum element, so that the minimum can be retrieved in constant time. Elements in each level is maintained using a doubly linked list termed as *child list* such that the insertion, and deletion in an arbitrary position in the list can be performed in constant time. Each node is having a pointer to its left node, right node, parent, and child. Also there are variables in each node for recording the degree (the number of children of the node), marking of the node, and the key (data) of the node. We shall now see various Fibonacci heap operations. We shall work with min Fibonacci heap throughout this lecture.

Fibonacci Heap operations

We shall discuss each operation and its associated amortized analysis. For asymptotic analysis, we introduce a potential function, and we analyze the change in potential for each operation to obtain the amortized cost.

Insertion

To insert an element into a Fibonacci heap, we simply attach that element to the root list and update the min pointer. It is a constant time task. The following figure illustrates an example for the sequence $5, -1, 3, 5, 7, 8, 9, 20, -50, 25$ into the initially empty Fibonacci heap. This operation is as good as creating a simple linked list on the root nodes, assuming each node is a root node to begin with. No other computational tasks such as *marking*, *consolidation* are done as part of insert operation.

$$5 \to -1 \to 3 \to 5 \to 7 \to 8 \to 9 \to 20 \to \underbrace{-50}_{MIN} \to 25$$

Potential Function

The potential function ϕ for the amortized analysis of an operation on Fibonacci heap at time (iteration) i is given by the following equation:

$$\phi_i = N + 2M \tag{1}$$

Here, N is the number of nodes present in the root list and M is the number of marked nodes present in the Fibonacci heap.

- **Note 1:** If there are n Fibonacci heaps $H_1, H_2, \ldots, H_n$ in the analysis with $N_1, N_2, \ldots, N_n$ be the number of the nodes in the root list and $M_1, M_2, \ldots, M_n$ be the number of marked nodes, respectively, then we use $N = N_1 + N_2 + \cdots + N_n$ and $M = M_1 + M_2 + \cdots + M_n$.
- **Note 2:** A node is marked or unmarked only when we perform *Decrease key* operation. We will discuss marked nodes in detail in the **Decrease key** section of this lecture.
- **Note 3:** The total cost of amortized analysis is given by the equation

Amortized cost = Actual Cost + Change in potential
Amortized cost = Actual Cost + $(\phi_i - \phi_{i-1})$

Amortized Analysis of Insert operation:

Actual cost for insert into a Fibonacci heap is $O(1)$.

- To insert, we append the new node to the root list.
- To update the minimum pointer, one comparison is involved. That is, compare the minimum pointer's value with the value of the inserted node, and update the minimum pointer, if necessary.

Hence the actual cost of insert takes two comparisons, which is $O(1)$. Now, we shall see the total amortized cost for the insertion operation. Let the number of nodes in the root list before insertion be N and the number of marked nodes be M. After insertion, the number of nodes in the root list is $N + 1$, and the number of marked nodes remain M. From equation (1),
we get $\phi_i = N + 1 + 2M$, $\phi_{i-1} = N + 2M$, and $\phi_i - \phi_{i-1} = 1$.
Therefore, the total amortized cost is $O(1) + 1 = O(1)$. Thus, the cost for insertion into a Fibonacci heap is *O(1)* amortized.

Extract Minimum

Extract minimum operation removes the minimum element from the Fibonacci heap. Four steps involved in this operation are as follows.

- Remove the minimum element from the root list.
- Append the child list of the removed node onto the root list.
- *Consolidate* the Fibonacci heap. During this step, the structure of the heap is modified, and it takes a shape which is close to a binomial heap. We start merging two Fibonacci trees of degree d to obtain a Fibonacci tree of degree $d + 1$ iteratively, so that at the end of consolidate for each $d \geq 0$, there exists at most one Fibonacci tree of degree d.
- Update the min pointer.

Since the implementation of Fibonacci heap is based on a doubly linked list, the first two steps can be performed in constant time.

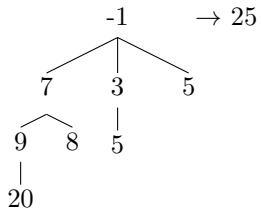
Consolidate in a Fibonacci heap

Note that after removing the minimum node, and placing the child list onto the root list, consolidation is done in the resultant structure. It is during this procedure that the structure of the Fibonacci heap is re-structured. That is, after consolidate procedure, the nodes in the root list has distinct degree values. There cannot be two nodes in the root list having the same degree. So, whenever an extract minimum occurs, we need to check whether there are nodes in the root list with the same degree; if there are nodes with the same degree then we should merge their min heaps to form a single min heap, and this process continues until there are no nodes on the root list having the same degree.

If roots of two min heaps with the same degree d exist, then the min heaps should be merged to form a single min heap with the degree of the root node being $d+1$. During merge the min heap whose root node contains the min value becomes the root of the merged tree. The figure below illustrates an extract minimum operation.
Fibonacci Heap before extract min:

$$5 \to -1 \to 3 \to 5 \to 7 \to 8 \to 9 \to 20 \to \underbrace{-50}_{MIN} \to 25$$

Fibonacci Heap after extract min:



Note: *Consolidate* operation is performed only during an extract minimum operation. There is no change in the *marking* scheme associated with nodes. For the input sequence $(I, \ldots, I, E)$, where I denotes insert and E denotes extract-min, the behavior of Fibonacci heap and Binomial heap are same. The subsequent inserts do not guarantee binomial heap property, for example, $(I, \ldots, I, E, I, I)$.

Amortized Analysis

For a Fibonacci heap with N nodes in the root list and let D be the maximum degree of a node in the root list, we claim that the actual cost for extract minimum operation is $O(N+D)$ and the amortized cost is $O(D)$.

Proof: Let H be a Fibonacci heap and r be a node in the root list of H pointed by the minimum pointer. After extracting the node pointed by r, we place its child list onto the root and invoke consolidate() subroutine. During this process, two nodes of the same degree in the root list are merged and as a result, the resultant Fibonacci heap H' has distinct degrees in the root list. Let D' be the degree of r in H and D be the maximum degree of a node in H'. Clearly, for each node in the root list of H', the degree is in the range $[0..D]$ and there can be at most $D+1$ nodes in the root list of H'.

- The transition from H to H': The number of nodes in the root list of H is N and in H', it is at most $D+1$.
- The actual cost for consolidate: We first place the children of r onto the root list of H which has already N nodes. Since the degree of r is D', the number of nodes in the root list will be $N-1+D'$ (the minimum node is removed and its D' children are placed onto the root list)
- We then start merging two nodes of identical degrees from the root list to obtain H'; this process incurs $(N-1+D') \cdot O(1) = O(N+D'-1)$. Thus, the actual cost is $O(N+D'-1)$.
- Finally, the number of nodes in the root list of H' is $D+1$

Amortized cost = Actual cost + change in potential
$= O(N+D'-1) + (1+D+2M) - (N+2M)$
$= O(N+D'-1) + 1 + D - N$
To simplify the above expression, we shall analyze the relation between D and D'.

1. Three possible scenarios are (i) $D' = D$, (ii) $D' < D$ and (iii) $D' > D$

2. If $D' = D$, then the expression $N + D' - 1 = N + D - 1$, which is at most $N + D$.

3. If $D' < D$, then $D' = D - r, r \geq 1$. The expression, $N + D' - 1 = N + D - r - 1 \leq N + D$. It is possible that there can be a large difference between D' and D. For example, a sequence of 256 insert into an initially empty Fibonacci heap followed by extract-min results in a Fibonacci heap which has $D' = 1$ and $D = 8$.

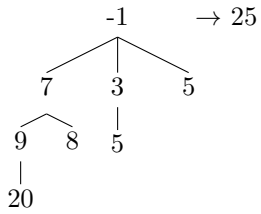
4. If $D' > D$, then observe that $D' = D + 1$. The expression, $N + D' - 1 = N + D + 1 - 1 = N + D$. Since D' is more than D, it is clear that there exists exactly one node x of degree D' in H. Clearly, the degrees of the children of x are in the range $[0..D' - 1]$ by the construction of Fibonacci heap, and therefore, $D = D' - 1$.

From the above discussion, it is clear that, the actual cost $O(N + D' - 1)$ is $O(N + D)$. Thus, the amortized cost is $O(N + D) + 1 + D - N$. If we the hidden constant c in $O(N + D)$ is 1, then the amortized cost is $O(D)$. Otherwise, increase the potential based on the hidden constant. That is,
notice that the there exists a positive constant c involved in the order notation, and we circumvent the cost by appropriately scaling the potential function. We scale the potential function as $\phi = c(N + 2M)$. This implies that,
the amortized cost $= c(N + D) + c(1 + D - N) = 2cD + c = O(D)$.
By increasing ϕ to $c\phi$ we just change our potential function but the actual cost remains same.
Note: We shall calculate D in terms of n, where n is the number of nodes in the Fibonacci heap towards the end of this lecture. Showing $D = O(\log_\phi n)$ will be discussed in the section `Relation of Fibonacci heap with Fibonacci series` of this lecture.

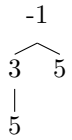
Merge

Merging two Fibonacci heaps is simply appending the root list of a Fibonacci heap with the other. After appending the root the min pointer is also updated. That is, compare the min nodes in both the Fibonacci heaps, and update the min pointer appropriately.

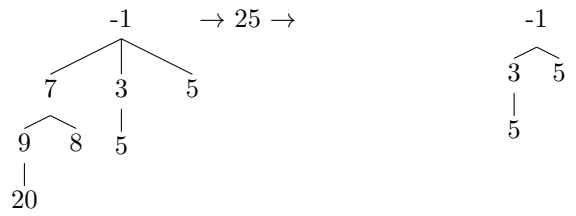
Fibonacci Heap: H_1



Fibonacci Heap: H_2



After Merge;



Amortized analysis

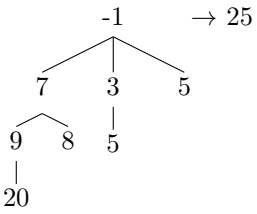
Actual cost for merging two Fibonacci heaps is $O(1)$. In particular, merge incurs a comparison for updating the min pointer and constant effort for reassigning pointers in the root list. Now we shall analyze the amortized cost for merge operation. Before merge let the number of nodes in the root list of Fibonacci heap H_1 be N_1 and the number of marked nodes be M_1. Similarly, the number of nodes in the root list of Fibonacci heap H_2 be N_2 and the number of marked nodes be M_2. In the merged Fibonacci heap the number of nodes in the root list is $N_1 + N_2$, and the number of marked nodes is $M_1 + M_2$.
From equation (1) $\phi_i = N_1 + N_2 + 2(M_1 + M_2)$,
$\phi_{i-1} = N_1 + 2M_1 + N_2 + 2M_2$, and $\phi_i - \phi_{i-1} = O(1)$
the total cost $= O(1) + O(1) = O(1)$
Therefore, the cost for merging two Fibonacci heaps is $O(1)$ amortized.

Decrease Key

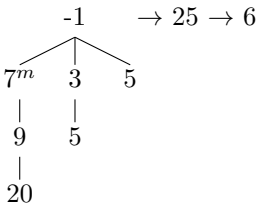
The following steps are involved in decreasing the value of a key stored at a node pointed by x in a Fibonacci heap H.

- Case 1: x is a node in the root list
  1. Decrease the value at node x to the defined value, say y.
  2. Do not change the marking scheme of x.
  3. Update min pointer, if required.
- Case 2: x is a non-root node (a node not in the root list)
  1. Decrease the value at node x to the defined value, and cut x along with its subtrees (children) and place it on the root list. Unmark the node x, if it is already marked. Proceed to the next step.
  2. **IF** (parent(x) is unmarked and parent(x) is a node in the root list), **THEN** stop. Do not mark parent(x).
  3. **IF** (parent(x) is unmarked and parent(x) is a non-root node), **THEN** Mark parent(x) and stop.
  4. **IF** (parent(x) is marked and parent(x) is a non-root node), **THEN** Recursively cut parent(x) and place parent(x) along with its children on the root list
     After placing the parent on the root list, unmark parent(x)
     Continue until parent(x) is unmarked or the parent(x) is a node in the root list. When you encounter the non-root unmarked parent(x) during the last recursive call, then mark parent(x) and stop the recursion. The other stopping criterion is when the last recursive call encounters parent(x) which is a node in the root list, and in this case do not change the marking scheme of parent(x).
  5. Update the min pointer, if required.

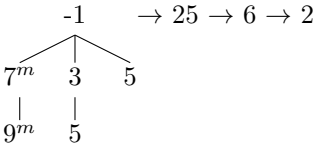
Illustration:
Fibonacci Heap before Decrease key:



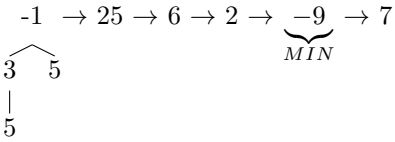
Decrease 8 to 6; as per the procedure, 6 is removed and placed at the root list and its parent 7 is marked. The procedure terminates as 7 was a unmarked parent.



Decrease 20 to 2. Remove that node and place at the root list. Since 9 is unmarked, mark 9 and stop.



Decrease 9 to -9; since 9 is marked, unmark 9 and place it at the root list. Also, cut its parent '7' as it is marked and place it at the root list. Since, the parent of 7 is a node in the root list, the recursive cut terminates.



Amortized Analysis

Actual cost for Decreasing a key of a node x in a Fibonacci heap H is $O(1)$ if we fall into Case 1 or Case 2.2 or Case 2.3. Note that, we incur 1 credit for removing x with its children and another credit for placing them onto the root. Thus, simple cut costs 2 credits, which is $O(1)$. For Case 2.4, suppose there are c recursive calls, then each recursive call incurs 2 credits and thus, we incur a cost of $O(c)$. To analyze the amortized cost, we need to understand the change in the number of marked nodes (M) as it appears in the potential function.

The number of Marked nodes after decrease key operation:
Note that there are M marked nodes before the decrease key operation.

1. If Case 1 or 2.1, M remains the same
2. If Case 2.2, M is changed to $M-1$ (if x is marked) or M (if x is unmarked)
3. If Case 2.3, and there are c recursive calls;
   (a) If x is unmarked and stopping criterion is 'non-root node', then M becomes $M-(c-1)+1 = M-c+2$. Except the last recursive call, every other call (there are $c-1$ of them) unmarks the marked parent, and the last recursive call marks the parent as it is a non-root node.

(b) If x is unmarked and stopping criterion is 'root node', then M becomes $M-(c-1) = M-c+1$. Each of $(c-1)$ recursive calls unmark the marked parent and no marking/unmarking for the c^{th} recursive call.

(c) If x is marked and stopping criterion is 'non-root node', then M becomes $M-1-(c-1)+1 = M-C+1$. First x is unmarked and $(c-1)$ recursive calls unmark the marked parent followed by the marking of the parent node encountered in c^{th} recursive call.

(d) If x is marked and stopping criterion is 'root node', then M becomes $M-1-(c-1) = M-c$. x is unmarked followed by unmarking of parent(x) encountered during $(c-1)$ recursive calls and no marking for the last recursive call.

4. From the above discussion, we can conclude that there are at most $M-c+2$ marked nodes at the end of decrease key operation. For Cases 1, 2.1, 2.2, assume $c = 0$.

The change in potential is calculated as follows. The number of nodes in the root list after decrease key operation is $N+1+(c-1) = N+c$; x is placed on the root list followed by one new node during each of $(c-1)$ recursive calls.

$$\begin{aligned} \phi_i &= N + c + 2(M - c + 2) \\ \phi_{i-1} &= N + 2M \\ \phi_i - \phi_{i-1} &= 4 - c \end{aligned}$$

The total amortized cost $= O(c) + 4 - c = kc + 4 - c$.
Where k is a constant.

If we increase ϕ suitably to $k\phi$, then we have, Total cost $= O(c) + 4k - kc = 4k$ (constant). So the cost for Decreasing the key of node x of a Fibonacci heap is $O(1)$ amortized.
If it is Cases 2.1 and 2.2, then the number of nodes in the root list is $N+1$. Accordingly, the change in potential is $3 = O(1)$ and the total amortized cost is $O(1) + O(1) = O(1)$. Therefore, the decrease key can be performed in constant amortized time.

Deletion in a Fibonacci Heap

Deleting a node x from the Fibonacci heap is equivalent to performing Decrease key operation on node x and decreasing it's value to $-\infty$ (or some small number) and then performing extract minimum operation. This will remove node x from the Fibonacci heap. Therefore, the cost for deletion is $O(D)$ amortized.

2 Relation of Fibonacci heap with Fibonacci series

The objective of this section is two fold. Firstly, we establish the relation between Fibonacci heap and Fibonacci series through a series of structural observations. Secondly, we show that the value of 'D' is $O(\log n)$.

2.1 Claim 1:

Let x be any node in a Fibonacci heap having $degree(x) = k$. Let $y_1, y_2, \ldots, y_k$ be the children of x in the order in which they were linked to x from the earliest to the latest.

Then $degree(y_1) \geq 0$ and $degree(y_i) \geq i-2$, for $i = 2, 3, \ldots, k$.

Proof:
At the time when y_i was linked to x, $degree(y_i) = degree(x)$. This is true, because as part of consolidate(), we merge two trees whose root nodes have same degree. In particular x contains $y_1, y_2, \ldots, y_{i-1}$ as its children. So, $degree(y_i) = degree(x) = i - 1$.
After merging x and y_i nodes, assuming x becomes the root and y_i is a child node, at most one child of y_i can be cut as part of decrease key operation. When one child of y_i is cut, node y_i is marked, the moment we try to cut another child, we would also remove y_i and place it on the root. If y_i is also cut, then the degree of x becomes $k - 1$. So, y_i cannot be cut and it can lose at most one child. Thus, $degree(y_i) \geq i - 2$.

2.2 Claim 2:

$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$, where F_i is the i^{th} term of the Fibonacci series.
Proof:
We will prove this by mathematical induction on k.
Base case : $k = 0$, $F_2 = 1 + F_0 = 1$, it is true for the base case.

Induction hypothesis : We assume that its true for smaller values of $k \geq 1$, i.e., $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

Anchor step : Consider, F_k. From the definition of Fibonacci series, we get $F_k = F_{k-1} + F_{k-2}$.
From the hypothesis, we get
$F_k = 1 + \sum_{i=0}^{k-3} F_i + F_{k-2}$
$F_k = 1 + \sum_{i=0}^{k-2} F_i$
Therefore, the claim follows.

2.3 Claim 3:

Let x be any node in the root list having degree k and s_k be the size of the tree (the number of nodes) rooted at x. $s_k \geq F_{k+2} \geq \phi_k$, where F_i is the i^{th} term of the Fibonacci series and ϕ is golden ratio. $\phi = \frac{1+\sqrt{5}}{2}$.
Proof:
We will prove this by mathematical induction on k. If multiple nodes are there with degree k, then choose a node with the least number of nodes for our discussion.

Base case : $k = 0$, $s_0 = 1 = F_2$, its true for the base case.
In general $s_k = 1 + \sum_{i=1}^{k} s_{degree(y_i)}$. The number of nodes rooted at x is the size of its children and also the node itself.
Induction hypothesis : We assume that its true for smaller values of $k \geq 1$, $s_k \geq F_{k+2}$

Anchor step : Consider, s_k. We know that $s_k = 1 + \sum_{i=1}^{k} s_{degree(y_i)}$. From Claim 1, we know that $degree(y_i) \geq i - 2$.
$s_k = 1 + s_{degree(y_1)} + \sum_{i=2}^{k} s_{degree(y_i)}$.
Note that $degree(y_1) = 0$ and $s_0 = 1$.
$s_k \geq 1 + 1 + \sum_{i=2}^{k} s_{i-2}$.
$s_k = 2 + \sum_{i=2}^{k} s_{i-2}$.
From the induction hypothesis, we get
$s_k \geq 2 + \sum_{i=2}^{k} F_k$.

$s_k \geq 1 + F_0 + F_1 + \sum_{i=2}^{k} F_i$.

$s_k = 1 + \sum_{i=0}^{k} F_i$.

From Claim 2, we thus get, $s_k \geq F_{k+2}$.

2.4 Claim 4:

The max degree D of a n node in a Fibonacci heap is $O(log_\phi n)$.
Proof:

Let k be the degree of a node x.

$$\begin{aligned} n &\geq size(x) \\ size(x) &\geq s_k \geq F_{k+2} \\ F_{k+2} &= \phi^{k+2} \\ \phi^{k+2} &\geq \phi^k \\ \phi^k &\leq n \\ k &\leq \log_\phi n \end{aligned}$$

So, for any node x, degree$(x) \leq \log_\phi n$, so the maximum degree $D \leq \log_\phi n$.

This establishes the relationship between Fibonacci heap and the Fibonacci series.

Remarks:
1. What is the significance of '2' in $N + 2M$. When a node is marked as part of decrease key, the potential associated with the data structure increases by 2 and stored with the data structure. When the node is cut as part of recursive cut of decrease key, the stored credit supplies for that operation and hence the cost of recursive cut is free. Out of 2 credits, one credit is used for removing the node along with its children and the other credit is used for placing the node on the root list.

2. Potential function $N + kM$, $k \geq 2$ works fine.

3. As part of accounting method, we charge 2 credits for insert and 1 credit for merge. The delete is free as excess credit given during insert will pay for this operation. $O(\log n)$ credits for extract-min as consolidate procedure incurs $O(D) = O(\log n)$ effort. We charge 4 credits for decrease key operation: out of 4, 2 credits are used for removing and placing the node on the root list and the other two credits are stored with the node's parent which will be used later as part of recursive cut. This means, the recursive cut is free.

4. Aggregate Analysis: The actual costs in worst case for (i) insert is $O(1)$ (ii) merge is $O(1)$ (iii) extract-min is $O(n)$ for the first extract min and $O(\log n)$ for subsequent extract min (iv) $O(\log n)$ for decrease key. The worst case scenario occurs when the input sequence on y operations has $O(y)$ extract min/decrease key operations. Thus, the amortized cost is $O(\log n)$ as per this analysis.

Acknowledgements: Lecture contents presented in this module and subsequent modules are based on the text books mentioned at the reference and most importantly, author has greatly learnt from lectures by algorithm exponents affiliated to IIT Madras/IMSc; Prof C. Pandu Rangan, Prof N.S.Narayanaswamy, Prof Venkatesh Raman, and Prof Anurag Mittal. Author sincerely acknowledges all of them. Special thanks to Teaching Assistants Mr.Renjith.P and Ms.Dhanalakshmi.S for their sincere and dedicated effort and making this scribe possible. Author has benefited a lot by teaching this course to senior undergraduate students

and junior undergraduate students who have also contributed to this scribe in many ways. Author sincerely thank all of them.

References:

1. E.Horowitz, S.Sahni, S.Rajasekaran, Fundamentals of Computer Algorithms, Galgotia Publications.
2. T.H. Cormen, C.E. Leiserson, R.L.Rivest, C.Stein, Introduction to Algorithms, PHI.
3. Sara Baase, A.V.Gelder, Computer Algorithms, Pearson.