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 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 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 exist 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 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 in a Fibonacci heap we simply attach that element to the root list and update the min pointer. The following figure illustrates an example; insert the sequence $25, -50, 20, 2, 9, 8, 7, 6, 5, -1$ into the Fibonacci heap.

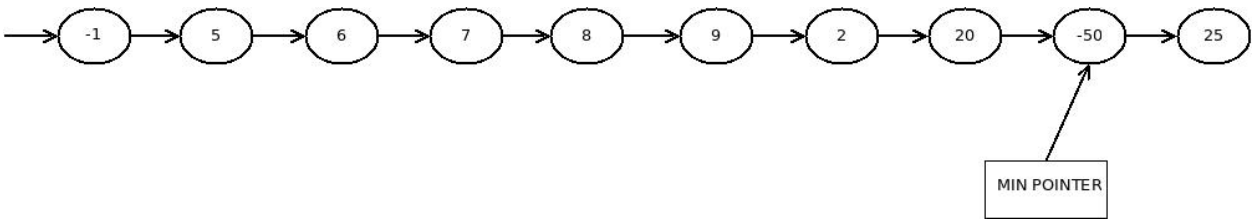

Figure 1: Insertion in a Fibonacci Heap

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 inserted node, and update the minimum pointer if necessary.

Hence insertion 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 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.
- 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.

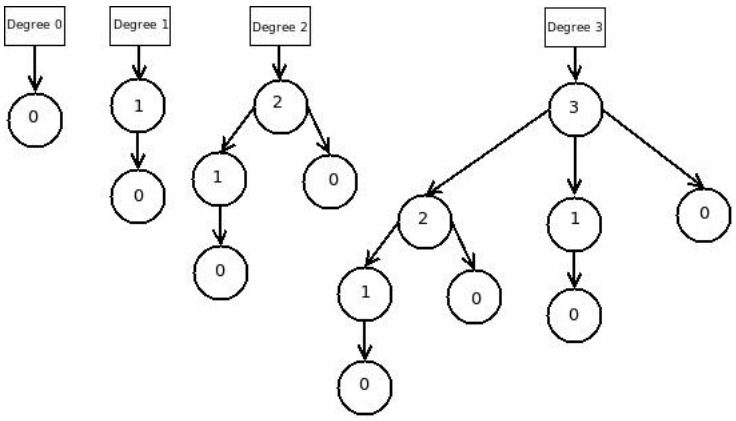

Figure 2: Illustration: Degree of root nodes

Consolidate Fibonacci heap

Degree of a node in the root list is the number of children attached to it. Figure 2 illustrates degrees of nodes in the root list. 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 restructured. That is, after consolidate procedure, the nodes in the root list has distinct degree values. There can not be two nodes in 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. Figure 3 illustrates an extract min operation performed on Figure 1.

Note: *Consolidate* operation is performed only during an extract minimum operation.

Amortized analysis

For a Fibonacci heap with N nodes in the root list and D be the the maximum degree of a node in the root list. We claim that the actual cost for extracting min operation is $O(N + D)$.

Proof: Let the maximum degree of a Fibonacci heap before performing extract minimum operation be D' and the maximum degree after performing extract minimum operation be D. After extract minimum, we perform *consolidate* Fibonacci heap, wherein we merge two nodes having the same degree k to form a single node having degree $k + 1$, and continue until there exist no two nodes having the same degree. So we have

$$D \geq D'$$

Let the degree of the minimum node be D'' (before the operation),

$$D'' \leq D' \Rightarrow D'' \leq D$$

After extracting minimum we have $N+D''-1$ nodes on the root list. (`Already` N `nodes present, we add all the child nodes of the minimum node to the root list, i.e., we add` D'' `nodes to the root list and remove the minimum node`). Then, we merge the nodes, merging two nodes is $O(1)$ and we merge $N+D''-1$ nodes. The actual cost is $O(N+D''-1)$ which is equal to $O(N+D-1)$ as $N+D''-1 \leq N+D-1$. So the actual cost for extracting minimum from a Fibonacci heap is $O(N + D)$. Hence, the claim.

We shall now analyze the total cost involved in the extract min operation. Let the number of nodes in

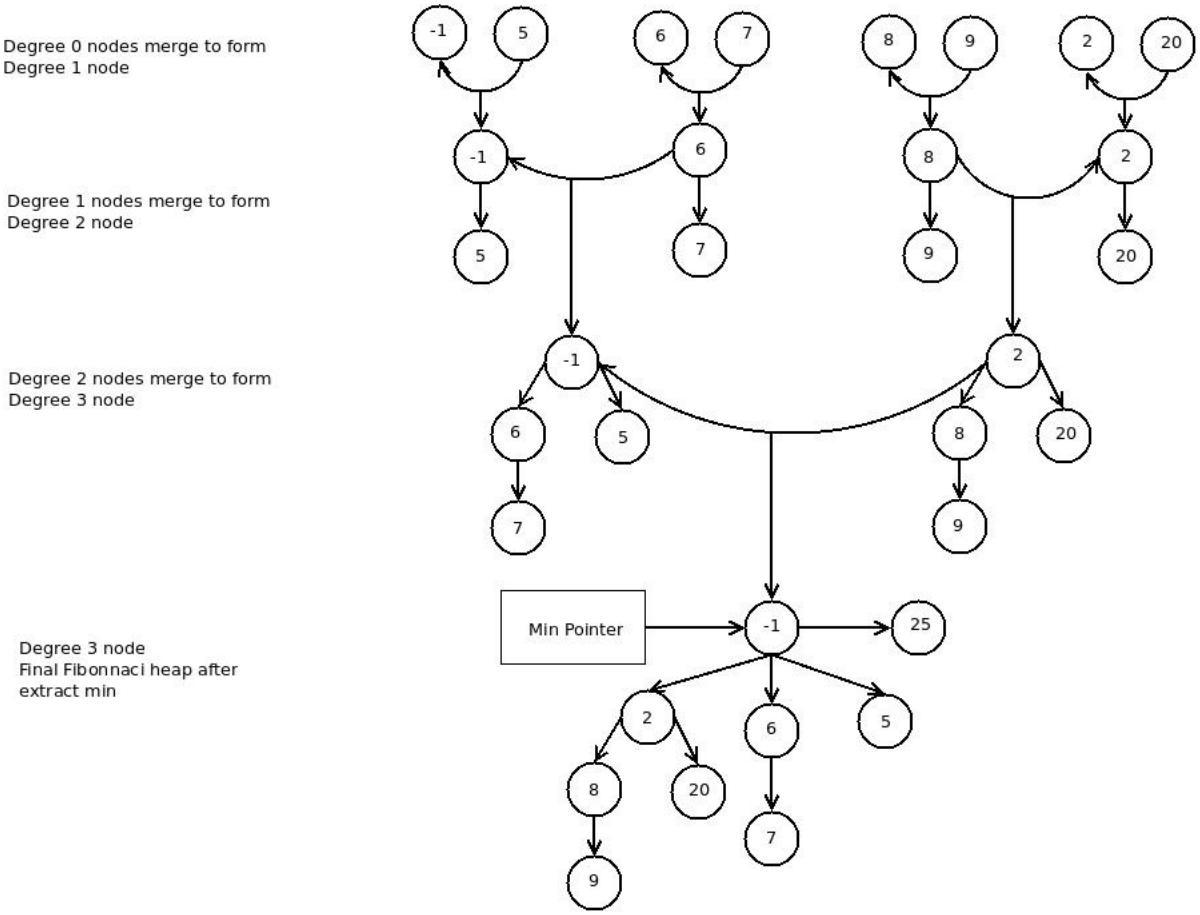

Figure 3: An illustration: Extract min

the root list before extract min be N and the number of marked nodes be M. After extracting minimum, the number of nodes in the root list is $D+1$, where D is the degree of the maximum degree node of the Fibonacci heap after the consolidation procedure. Since the extract min operation do not alter any 'marking' of nodes, the number of marked nodes remain the same after extract min operation. Recall from equation (1)

$$\begin{aligned} \phi_i &= D+1+2M \\ \phi_{i-1} &= N+2M \\ \phi_i - \phi_{i-1} &= D-N+1 \end{aligned}$$

Total cost $= O(N+D) + D - N + 1$
Here notice that the there exist a positive constant c involved in the order notation, and we circumvent the cost by appropriately scaling the potential function. That is, we scale the potential function as $\phi_i = c(D+1+2M)$ and $\phi_{i-1} = c(N+2M)$. This implies that,
Total cost $= c(N+D) + c(D-N+1) = 2cD + c = O(D)$.
By increasing ϕ to $c\phi$ we just change our potential function but the actual cost remains same.
Note: We will calculate D in terms of n, where n is the number of nodes present in the Fibonacci heap. Calculation for D will be discussed in the `Relation of Fibonacci heap with Fibonacci series` section 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.

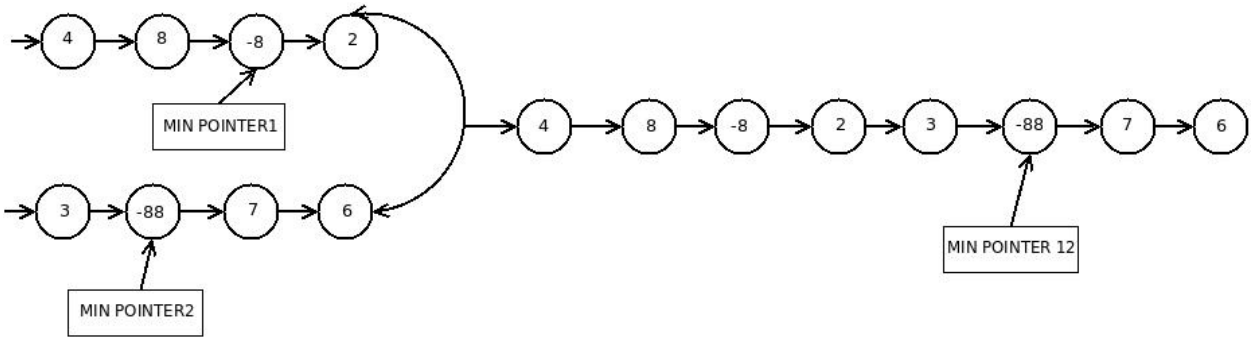

Figure 4: Merging of Fibonacci heaps

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 total cost for merge operation. Before merge let the number of nodes in the root list of Fibonacci heap 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 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 ($\mathbf{x}$) in a Fibonacci heap:

- Decrease the value at node $'\mathbf{x}'$ to the defined value, and cut $\mathbf{x}$ along with it subtrees and place it on the root list. Unmark $\mathbf{x}$ if it is marked.
- **If**(parent($\mathbf{x}$) is unmarked)
  Mark parent($\mathbf{x}$).
  Else
  Recursively cut parent($\mathbf{x}$) which is marked and place parent($\mathbf{x}$) on the root list
  After placing on the root list, *unmark parent*($\mathbf{x}$)
  Continue till parent($\mathbf{x}$) is unmarked or the parent(x) is a root node. When you encounter non-root unmarked parent(x), *mark parent*($\mathbf{x}$) and stop the recursion. Other stopping criterion is when the parent(x) is the root node and in this case do not change the marking scheme.
- Update the min pointer.

Amortized Analysis

Actual cost for Decreasing a key of a node $'\mathbf{x}'$ in a Fibonacci heap is $O(c) + O(1) = O(c)$, where c are the number of recursions we need to make until we find a parent($\mathbf{x}$) which is unmarked or we reach the

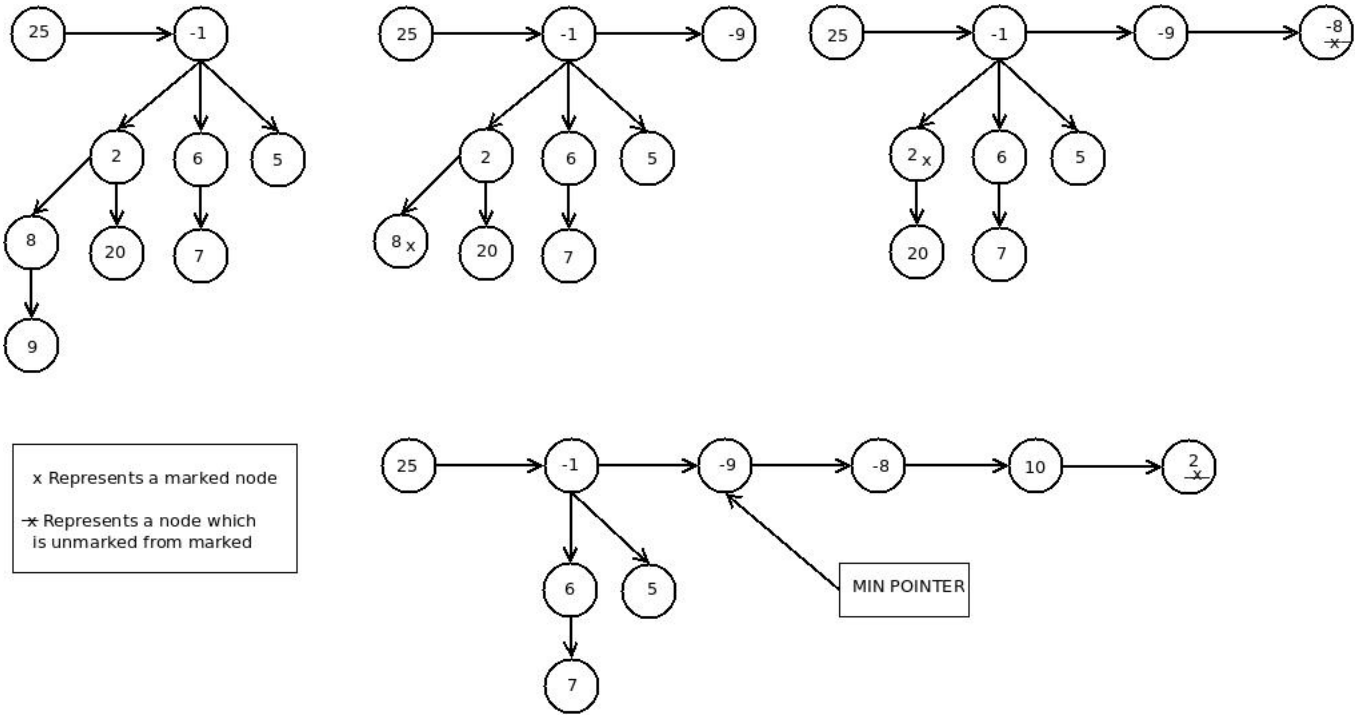

Figure 5: This figure shows decrease keys 9 to -9, then 8 to -8, and then 20 to 10 of Figure 3

root. Cost for updating the min pointer is $O(1)$.(*Compare the current min pointer value with the value of the decreased node, it takes one comparison* $O(1)$.)

Note: We have assumed that there is a pointer to the node for which decrease key has to be performed.

The total cost for decreasing a key of Fibonacci heap is: let, the number of nodes in the root list before decrease key be N and number of marked nodes be M. After decrease key, the number of nodes in the root list will be $N + c$. (c `nodes are added to the root list)` and the number of marked nodes will be at most $M - c + 2$, whose justification is given below.

```
if (x is marked)
 {
   if(Recursion stops at root)
    {
       No of unmarked nodes = c - 1 + 1;
       ( c - 1 nodes are changed from marked to unmarked.
         Even x is changed from marked to unmarked. )
    }
   else
    {
      No of unmarked nodes = c - 1 + 1 - 1;
      ( c - 1 nodes are changed from marked to unmarked.
        Even x is changed from marked to unmarked.
        Also there will be a parent node which is unmarked
        and it will be changed to marked. )
    }
```

```
 }
else
 {
  if(Recursion stops at root)
    {
       No of unmarked nodes = c - 1;
       ( c - 1 nodes are changed from marked to unmarked. )
    }
   else
    {
      No of unmarked nodes = c - 1 - 1;
      ( c - 1 nodes are changed from marked to unmarked.
        Also there will be a parent node which is unmarked
        and will be changed to marked. )
    }
 }
```

The number of marked nodes will be M - *no of unmarked nodes*. The number of marked nodes can be $M - c$ or $M - c + 1$ or $M - c + 2$. So *max(no of marked nodes)* $= M - c + 2$. From equation (1)

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

The total 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 $'\mathbf{x}'$ of a Fibonacci heap is *O(1)* Amortized.

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 $-\infty$ 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 can not 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 a 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.

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
$s_k \geq 2 + \sum_{i=2}^{k} F_k$.

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

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

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 is the relationship of Fibonacci heap with the Fibonacci series, and $O(D) = O(\log_\phi n)$.

3 Summary

- Fibonacci heap was discovered by Robert Tarzan, an American computer scientist.
- Using Fibonacci heap we can find the minimum element or maximum element or sort a given sequence efficiently.
- Potential function for Amortized analysis of a Fibonacci heap is $\phi = N + 2M$,where N is the total number of root nodes present in the root list of all the Fibonacci heaps present in the analysis and M is the total number of marked nodes present in all the Fibonacci heaps present in the analysis.
- A node is marked or unmarked only when we perform *Decrease key* operation on the Fibonacci heap.
- Cost for insert is $O(1)$ *amortized*.
- Cost for extracting minimum is $O(log_\phi n)$ *amortized*, where ϕ is the golden ratio.
- Cost for merge is $O(1)$ *amortized*.
- Cost for decrease key is $O(1)$ *amortized*.

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 thanks 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.