Date: Aug 21, 2025

Fibonacci Heap

- Fibonacci heaps are similar to binomial heaps but less rigid in structure.
- Here **consolidation** is deferred until the next **delete-min** operation.
- Unlike binomial heaps, trees of the fibonacci heaps are unordered.
- A fibonacci heap has the following pointers:
    1. A pointer to its parent.
    2. A pointer to one of its children.
    3. Circular doubly linked list to link the children of every node.
- Each node stores two fields:
    1. The number of children in the child-list of a node is stored as **degree**.
    2. A boolean value **mark** indicates whether a node has lost a child since the last time it was made the child of another node.

Finding the minimum

- Finding the minimum is a trivial operation because we keep the pointer to the node containing it.
- So this can be done in $O(1)$ time.

Insertion

- Works by creating a new heap with one element and doing merge.
- This also takes constant time.

Delete-min

This operates in three phases:

1. We take the root containing the minimum element and remove it, make its children the roots of new trees, update the pointer to the root with minimum key.
2. We decrease the number of roots by successively liniking together roots of same degree – when two roots u and v have the same degree, make one of

them the child of the other so that the one with the smaller key remains at the root. Its degree will be increased by one. This is repeated until every root has a different degree. To find trees of same degree efficiently, we use an array of length of $O(\log n)$ in which we keep a pointer to one root of each degree. When a second root is found of the same degree, the two are linked and the array is updated.

3. We check each of the remaining roots and find the min.

Decrease key operation

- This operation will decrease the key and if the heap property is violated, the node cuts from the parent if the parent is not a root, it is marked.
- If the parent was already marked, then it is cut as well and its parent is marked.
- We continue upwards until we reach either the root or an unmarked parent.

Delete operation

- Delete operation can be implemented by decreasing the key of the element to be deleted to $-\infty$. Thus turning it into the minimum of the whole heap.
- Then we call **delete-min** subroutine to remove it.