# Examples on the (2,3)-Tree, BFS and DFS

Dan (Doris) He

ITEE
University of Queensland

## (2,3)-Tree

A (2,3)-tree on a set $S$ of $n$ integers is a good 3-ary tree $T$ satisfying:

- Every leaf node — if not the root — stores either 2 or 3 data elements, each of which is an integer in $S$.
- Every integer in $S$ is stored as a data element exactly once.
- For every internal node $u$, if its child nodes are $v_1, ..., v_f$ $(f = 2$ or $3)$, then
  - For any $i, j \in [1, f]$ such that $i < j$, all the data elements in the subtree of $v_i$ are smaller than those in the subtree of $v_j$.
  - For each $i \in [i, f]$, $u$ stores a routing element, which is an integer that equals the smallest data element in the subtree of $v_i$.

## (2,3)-Tree

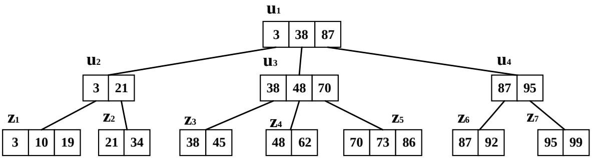
From the lecture, we know that a (2,3)-tree on a set of $n$ integers has the following guarantees:

- Space consumption $O(n)$
- Predecessor/successor query $O(\log n)$
- Insertion $O(\log n)$ time
- Deletion $O(\log n)$ time

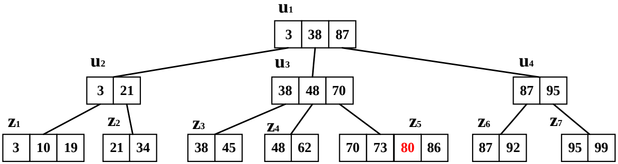
In the following, let's go through the algorithms of insertion and deletion with examples.

## Example on Insertion

Suppose that we want to insert 80 into the following tree, which should go into Leaf $z_5$.

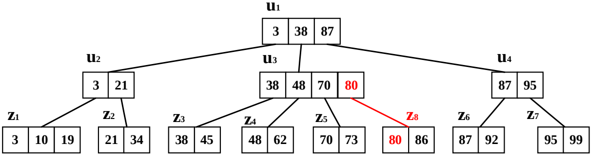



Now $z_5$ overflows, and needs to be split.

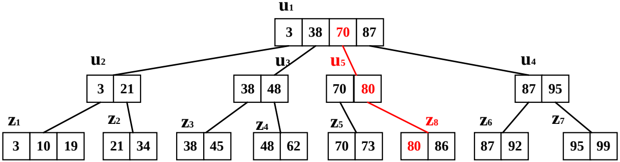


## Example on Insertion

Sliptting $z_5$ makes parent node $u_3$ overflow, which also needs to be split.

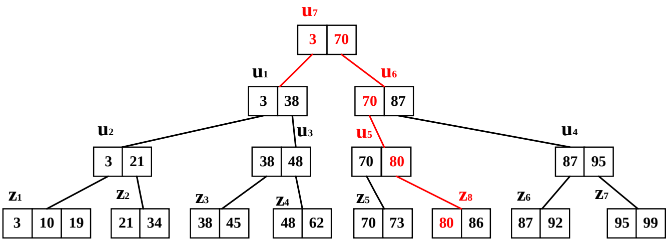



Sliptting $u_3$ makes the root node $u_1$ overflow, then split it. (create a new root)



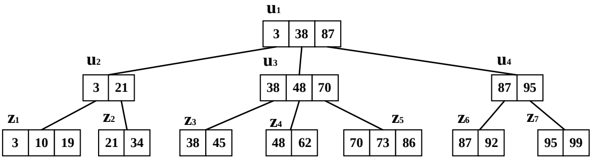
## Example on Insertion

Now the insertion completes.

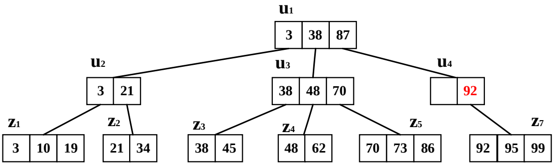


## Example on Deletion

Suppose that we want to delete 87 from the following tree. Remove it from Leaf $z_6$, which then underflows.

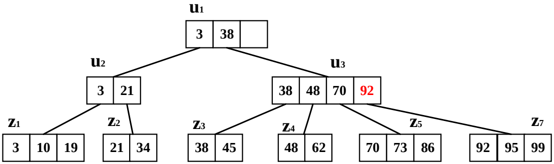



Merging $z_6$ with its right sibling $z_7$ causes their parent $u_4$ to underflow.

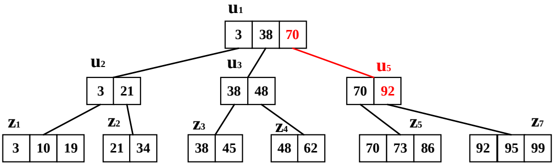


## Example on Deletion

Merging $u_4$ with its left sibling $u_3$ causes overflowing on $u_3$, then split it.




Now the deletion completes.



## Construction of a (2,3)-Tree

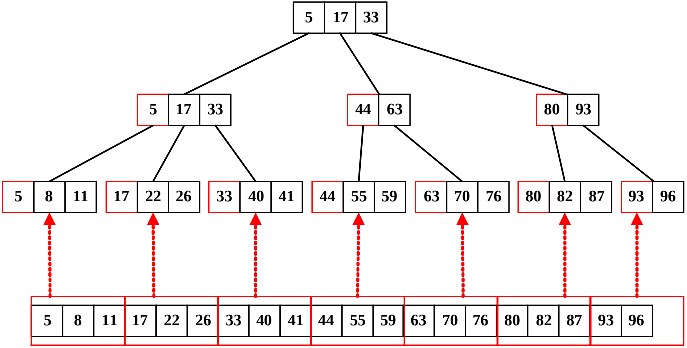
In the following, we will discuss how to construct a (2, 3)-tree on a given set $S$ of $n$ integers.

The most trivial way is to initialize an empty (2,3)-tree and insert the elements of $S$ one by one to this tree. The time complexity is $O(n \log n)$.

What's next is a more efficient algorithm to construct a (2, 3)-tree provided that $S$ is already sorted. The time complexity of this algorithm is $O(n)$.

The basic idea is to construct the tree level by level in a bottom-up manner.

## Example


5 17 33
5 17 33
44 63
80 93
5 8 11
17 22 26
33 40 41
44 55 59
63 70 76
80 82 87
93 96
5 8 11 17 22 26 33 40 41 44 55 59 63 70 76 80 82 87 93 96

## Construction of a (2,3)-Tree

Assume that the sorted set $S$ of $n$ integers is stored in an array. A (2,3)-tree on $S$ can be constructed as follows:

- Scan the sorted array and create a leaf node for every three consecutive integers except possibly for the last two leaf nodes which may have only two integers. All these leaf nodes are at level-0 and in ascending order.
- Given all the nodes at level-$i$ in ascending order, all the nodes at level-$(i+1)$ can be constructed in ascending order as follows:
  - Scan the level-$i$ nodes and create a parent node for every three consecutive nodes (except possibly for the last two parent nodes which may have only two child nodes).
  - Each parent node stores the smallest routing element in each of its child nodes as its routing elements.
- If there is only one parent node created at level-$(i+1)$, stop and return this node as the root of the (2,3)-tree.

## Construction of a (2,3)-Tree

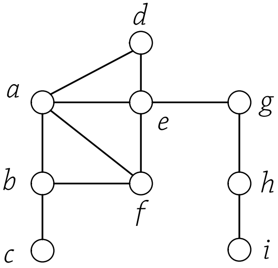
- The construction cost of level-0 is $O(n)$.
- The construction cost of level-$(i+1)$ $(i \geq 0)$ is bounded by the cost of scanning all level-$i$ nodes. Since there are at most $\frac{n}{2^{i+1}}$ nodes at level-$i$, the scanning cost is $O(\frac{n}{2^{i+1}})$.

The total construction cost is bounded by:

$$O(n + \frac{n}{2} + \frac{n}{2^2} + \cdots + 1) = O(n)$$

In the following, we will go over the algorithm of BFS and DFS on undirected graph.

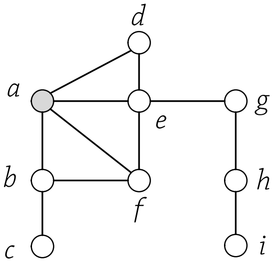
Input




Suppose that we start from $a$, namely $a$ is the root of both BFS tree and DFS tree.

BFS

Firstly, create a queue $Q$, insert the starting vertex $a$ into $Q$ and color it gray. Create a BFS Tree with $a$ as the root.

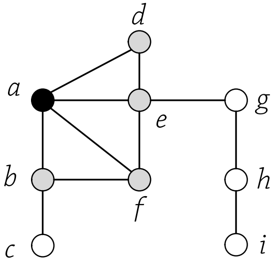


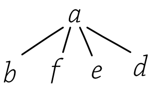

BFS Tree


$Q = (a)$

BFS

After de-queueing $a$.

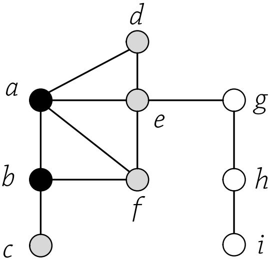


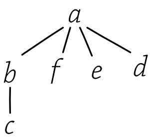




$$Q = (b, f, e, d)$$

BFS

After de-queueing $b$.

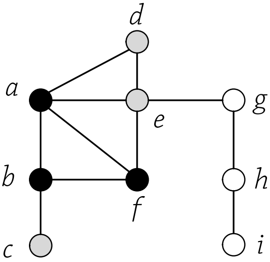


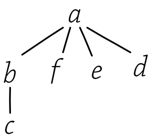




$Q = (f, e, d, c)$

## BFS

After de-queueing $f$.

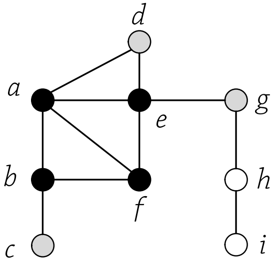


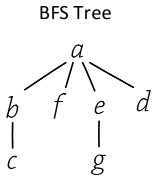




$Q = (e, d, c)$

## BFS

After de-queueing $e$.

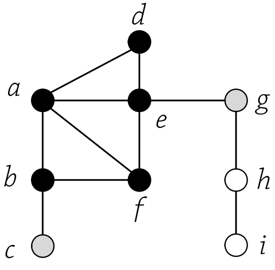


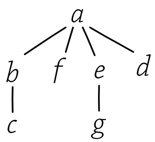

$Q = (d, c, g)$

## BFS

After de-queueing *d*.



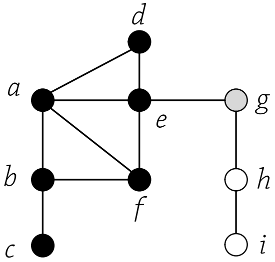

BFS Tree

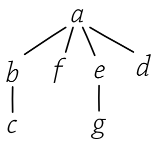



$Q = (c, g)$

## BFS

After de-queueing $c$.

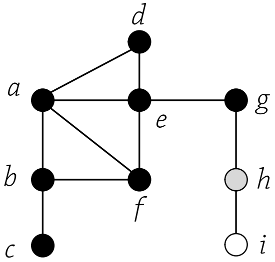


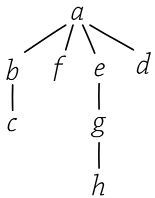




$Q = (g)$

## BFS

After de-queueing $g$.

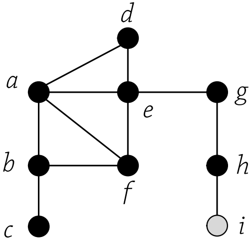


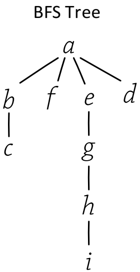




$Q = (h)$

## BFS

After de-queueing $h$.

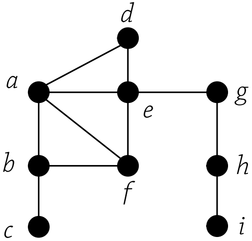


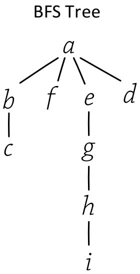




$Q = (i)$

## BFS

After de-queueing $i$.



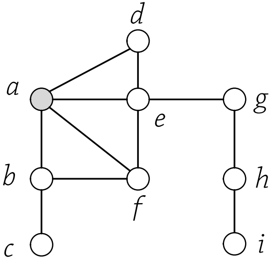




$Q = ()$

Done.

DFS

Firstly, create a stack $S$, push the starting vertex $a$ into $S$ and color it gray. Create a DFS Tree with $a$ as the root.



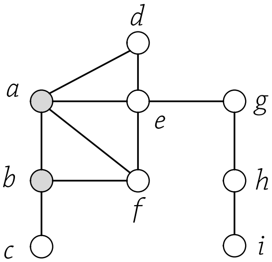

DFS Tree

$a$

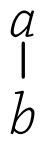
$S = (a)$

## DFS

Top of stack: $a$, which has white neighbors $b$, $f$, $e$ and $d$. Suppose that we access $b$ first. Push $b$ into $S$.



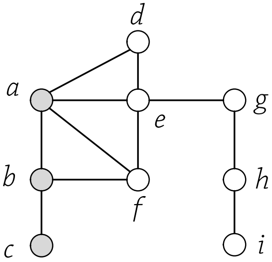

DFS Tree




$S = (a, b)$

## DFS

After pushing $c$.



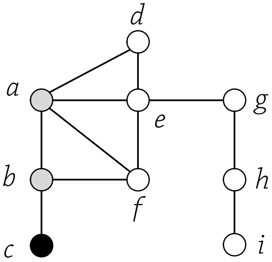

DFS Tree

$$\begin{array}{c} a \\ | \\ b \\ | \\ c \end{array}$$

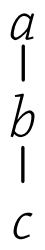
$S = (a, b, c)$

## DFS

Since $c$ has no white neighbors, pop it from $S$, and color it black.



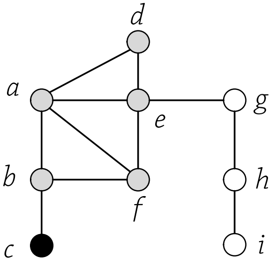

DFS Tree



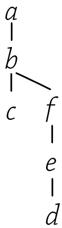

$S = (a, b)$

## DFS

After pushing $f$, $e$, $d$.



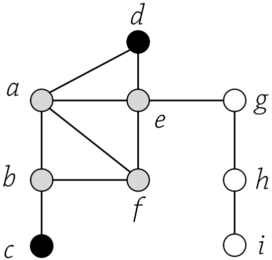

DFS Tree



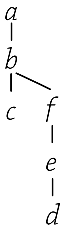

$$S = (a, b, f, e, d)$$

## DFS

Since $d$ has no white neighbors, pop it from $S$, and color it black.



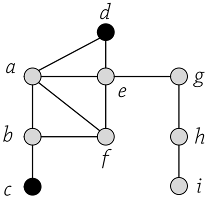

DFS Tree

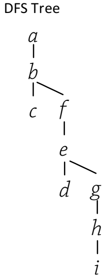



$S = (a, b, f, e)$

## DFS

Consecutively push $g$, $h$, $i$.

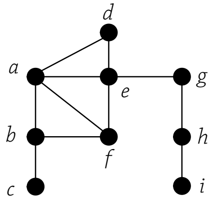


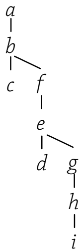

$S = (a, b, f, e, g, h, i)$

## DFS

After popping $i$, $h$, $g$, $e$, $f$, $b$, $a$.




DFS Tree




$S = ()$

Done.

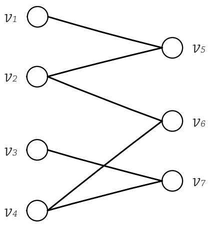
In the following, let's talk about an application on BFS.

## Bipartite Graph

A Bipartite Graph is an undirected graph $G(V, E)$ where:

- $V$ can be partitioned into two parts $V_1$ and $V_2$ such that:
  - $V_1 \cap V_2 = \emptyset$,
  - $V_1 \cup V_2 = V$.
- $\nexists (u, v) \in E$ such that $u, v \in V_i$ for $i = 1, 2$.

**Example**



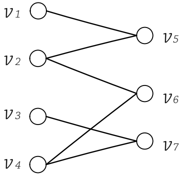

The left one is a bipartite graph, while the right one is not.

## Bipartite Graph Dectection

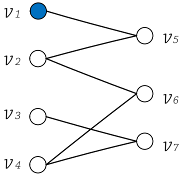
Given a graph $G = (V, E)$, how to determine whether it is bipartite?

Example

Create a queue Q, color all vertices white:



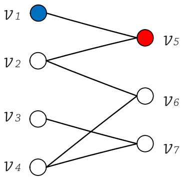

Arbitrarily pick a white vertex, e.g. $v_1$, color it blue and insert it into $Q$:




$$Q = (v_1)$$

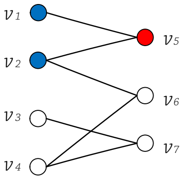
Example

De-queue from $Q$ the first vertex $v_1$, color its white neighbor $v_5$ red and insert it into $Q$.




$$Q = (v_5)$$

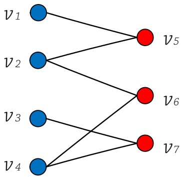
De-queue from $Q$ the first vertex $v_5$ and color its white neighbor $v_2$ blue, insert it into $Q$:




$$Q = (v_2)$$

## Example

While de-queueing a vertex $v$, if one of its neighbors $u$ has been colored with the same color as $v$, then terminate and report it is not bipartite. Otherwise, terminate untill all vertices are colored either blue or red.




Note that the vertices from the same subset have the same color.

## Bipartite Graph Dectection

In other to determine whether a graph $G = (V, E)$ is bipartite, we perform a BFS on it with a little modification. The algorithm is as follow:

1. Create a queue $Q$ and color all vertices white.
2. Arbitrarily pick a white vertex from the graph, color it blue and insert it into $Q$.
3. De-queue from $Q$ the first vertex $v$ and check its color.
4. For every neighbor $u$ of $v$,
    - if $u$ is colored, then check whether the color of $u$ is the same as $v$. If so, terminate and output "No". Otherwise, continue.
    - if $u$ is white and $v$ is blue, then color $u$ red; if $v$ is red, then color $u$ blue; insert it into $Q$.
5. Repeat from 3 untill $Q$ is emtpy, then repeat from 2 untill all vertices are colored and output "Yes".

## Bipartite Graph Dectection – Correctness

3 milestone steps to prove:

1. To perform BFS on an undirected graph, all nodes with shortest path distance $\ell$ from the source are at level $\ell$ of the BFS tree (root at level 0).
2. For an undirected graph, every edge in the BFS tree connects 2 nodes that are
   - Either at the same level
   - Or at adjacent levels.
3. The graph is bipartitie if and only if we do not have edges connecting nodes at adjacent levels.