**Full Trees**

**Definition:** A *full binary tree* of $n$ vertices:

- All non-leaf vertices have exactly 2 children.
- All levels are full except possibly for some rightmost portion of the bottom level (if a leaf at the bottom level is missing, then so are all of the leaves to its right).

The *rightmost bottom leaf (RB leaf)* is the rightmost leaf of the bottom level.

**Observations:**

- When traversed in level order, full trees are like complete trees except that the traversal stops sooner.
- Can be shown that the height of a full binary tree of $n$ vertices is $\leq \lceil \log_2(n) \rceil$. 
Idea:
Pack a full tree in level order into an array starting at 0.

Example:
Consider the following full binary tree:

- The left child of a vertex with index $i$ has index $2i+1$.
- The right child of a vertex with index $i$ has index $2i+2$. 
Basic $O(1)$ time operations:
(Assume the $n$ items are stored in positions 0 through $n-1$.)

PARENT($i$):
   \textbf{if} $i>0$ \textbf{then} ⌊($i-1)/2$⌋ \textbf{else} nil

LCHILD($i$):
   \textbf{if} (($2i)+1) < n \textbf{then} (2i)+1 \textbf{else} nil

RCHILD($i$):
   \textbf{if} (($2i)+2) < n \textbf{then} (2i)+2 \textbf{else} nil

RSIB($i$):
   \textbf{if} $i<(n-1)$ \textbf{and} $i$ not a multiple of 2 \textbf{then} $i+1$ \textbf{else} nil
**Full K-ary Trees:**

For \( k \geq 2 \), the concept of a full binary tree can easily be generalized to full k-ary trees where each internal vertex has exactly \( k \) children:

(Again, assume the items are stored in positions 0 through \( n-1 \).)

\text{PARENT}(i):
\[
\text{if } i > 0 \text{ then } \left\lfloor \frac{(i-1)}{k} \right\rfloor \text{ else nil}
\]

\text{LCHILD}(i):
\[
\text{if } ((ki)+1) \leq n \text{ then } (ki)+1 \text{ else nil}
\]

\text{mCHILD}(i):
\[
\text{if } ((ki)+m) \leq n \text{ then } (ki)+m \text{ else nil}
\]

\text{RSIB}(i):
\[
\text{if } i < (n-1) \text{ and } i \text{ not a multiple of } k \text{ then } i+1 \text{ else nil}
\]

*** We will only need the case \( k=2 \) to present heaps.