A B-tree of order $d$ is a multiway search tree with the following properties:

- The root has at least 2 children and at most $d$.
- All internal nodes other than the root have at least $\left\lceil \frac{d}{2} \right\rceil$ and at most $d$ children.
- All the leaves are at the same level.
B-Trees

B-Tree of order 6
B-Trees

B-Tree of order 6

```
41
/     |
23 35  |
   /    |
  25 29 |
 / |   |
36 39 40
```

```
45 56 68
/    |
49 55 |
 /    |
59 66 67
```

```
72 85
```

```
2 6
```

```
25 29
```

```
36 39 40
```

```
43 44
```

```
49 55
```

```
59 66 67
```

```
72 85
```
B-Trees

B-Tree of order 6

Not a B-tree of order 6
What is the Maximum Height of a B-Tree?

Height is $O(\log_d n)$
B-Trees

B-Tree of order 6

Put 38
B-Trees

B-Tree of order 6

Put 38
B-Trees

B-Tree of order 6

Put 38
B-Trees

B-Tree of order 6

Put 38
B-Trees

B-Tree of order 6

Put 38

split
B-Trees

B-Tree of order 6

Put 38

ew root

25 29
36 38
40 43 44
49 55
59 66 67
72 85

2 5

39

23 35

45 56 68
Algorithm *put* \((r, k, o)\)

**In:** Root \(r\) of a B-tree, data item \((k, o)\)

**Out:** \{Insert data item \((k, o)\) in the B-tree

- Search for \(k\) to find the **lowest** insertion internal node \(v\)
- Add the new data item \((k, o)\) at node \(v\)

**while** node \(v\) **overflows** **do** {
    **if** \(v\) is the root **then**
    
    Create a new empty root and set as parent of \(v\)
    
    Split \(v\) around the **middle** key \(k’\), move \(k’\) to parent, and update parent’s children
    
    \(v \leftarrow\) parent of \(v\)
}
Algorithm *put* \((r, k, o)\)

**In:** Root \(r\) of a B-tree, data item \((k, o)\)

**Out:** \{Insert data item \((k, o)\) in the B-tree \}
   - Search for \(k\) to find the lowest insertion internal node \(v\)
   - Add the new data item \((k, o)\) at node \(v\)

while node \(v\) overflows do {
   if \(v\) is the root then 
   Create a new empty root and set as parent of \(v\)
   Split \(v\) around the middle key \(k'\), move \(k'\) to parent, and update parent’s children
   \(v \leftarrow\) parent of \(v\)
}

Time complexity of put is \(O(d \log_d n)\)
B-Trees

B-Tree of order 6

Remove 35
B-Trees

B-Tree of order 6

Remove 35
B-Trees

B-Tree of order 6

Remove 35
B-Trees

B-Tree of order 6

Remove 35

```
2  6
25 29
39
```

```
  36
```

```
  23
```

```
41
```

```
  45  56  68
```

```
  43  44
```

```
  49  55
```

```
  59  66  67
```

```
  72  85
```
B-Trees

B-Tree of order 6

Remove 35

underflow
B-Trees

B-Tree of order 6
Remove 35

fusion
B-Trees

B-Tree of order 6

Remove 35
B-Trees

B-Tree of order 6

Remove 35

transfer
B-Trees

B-Tree of order 6

Remove 35
Algorithm \textit{remove}(r, k)

\textbf{In:} Root $r$ of a B-tree, key $k$

\textbf{Out:} \{remove data item with key $k$ from the tree\}

Find the node $v$ storing key $k$

Remove $(k, o)$ from $v$ replacing it with successor if needed

\textbf{while} node $v$ underflows \textbf{do} {
\textbf{if} $v$ is the root then
\hspace{1em} make the first child of $v$ the new root
\textbf{else if} a sibling has more than \lfloor \frac{d}{2} \rfloor \text{ keys} \textbf{then}
\hspace{1em} perform a transfer operation
\textbf{else} {
\hspace{1em} perform a fusion operation
$v \leftarrow \text{parent of } v$
\}
\}

23
Algorithm \textit{remove}(r, k) \quad \text{Time complexity } O(d \log_d n)

\textbf{In:} Root \( r \) of a B-tree, key \( k \)

\textbf{Out:} \{remove data item with key \( k \) from the tree\}

- Find the node \( v \) storing key \( k \)
- Remove \((k, o)\) from \( v \) replacing it with successor if needed

\textbf{while} node \( v \) \textit{underflows} \textbf{do} \{

\hspace{1em} \textbf{if} \( v \) is the root then
  \hspace{2em} make the first child of \( v \) the new root

\hspace{1em} \textbf{else if} a sibling has at least \( \lceil d/2 \rceil \) keys \textbf{then}
  \hspace{2em} perform a transfer operation

\hspace{1em} \textbf{else} \{
  \hspace{2em} perform a fusion operation
  \hspace{3em} \( v \leftarrow \) parent of \( v \)
\hspace{1em} \}
\}

\textbf{O}(d \log_d n)
Disk Blocks

• Consider the problem of maintaining a large collection of items that does not fit in main memory, such as a typical database.
• In this context, we refer to the external memory is divided into blocks, which we call **disk blocks**.
• The transfer of a block between external memory and primary memory is a **disk transfer** or **I/O**.
• There is a great time difference that exists between main memory accesses and disk accesses
• Thus, we want to minimize the number of disk transfers needed to perform a query or update. We refer to this count as the **I/O complexity** of the algorithm involved.
Memory Hierarchies

- Computers have a hierarchy of different kinds of memories, which vary in terms of their size and distance from the CPU.
- Closet to the CPU are the internal **registers**. Access to such locations is very fast, but there are relatively few such locations.
- At the second level in the hierarchy are the memory **caches**.
- At the third level in the hierarchy is the **internal memory**, which is also known as main memory or core memory.
- Another level in the hierarchy is the **external memory**, which usually consists of disks.