# BTree Analysis

**October 25, 2017**

**BTree Properties**

For a BTree of order $m$:

1. All keys within a node are ordered.
2. All nodes contain no more than $m - 1$ keys.
3. All internal nodes have exactly one more child than key.
4. Root nodes can be a leaf or have $[2, m]$ children.
5. All non-root, internal nodes have $[\text{ceil}(m/2), m]$ children.
6. All leaves are on the same level.

**BTree Analysis**

The height of the BTree determines maximum number of ________ possible in search data.

...and the height of our structure:

Therefore, the number of seeks is no more than: ________.

...suppose we want to prove this!

Remember from our AVL analysis:

- Finding an upper bound on the height (given $n$) is the same as finding a lower bound on the nodes (given $h$).
- Goal: Find a relationship for BTrees between the number of keys ($n$) and the height ($h$).

**BTree Strategy:**

1. Count the number of nodes, level by level.
2. Add the minimum number of keys per node.
3. Proving a minimum number of nodes provides us with an upper-bound for the maximum possible height.

---

1a. The minimum number of nodes for a BTree of order $m$ at each level is as follows:

   root:
   
   level 1:
   
   level 2:
   
   level 3:
   
   ...
   
   level $h$:

1b. The total number of nodes is the sum of all levels:

2. The total number of keys:

3. Finally, we show an upper-bound on height:
So, how good are BTrees?
Given a BTree of order 101, how much can we store in a tree of height=4?

Minimum:

Maximum:

Hashing

<table>
<thead>
<tr>
<th>Locker Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td></td>
</tr>
</tbody>
</table>

...how might we create this today?

Dictionary ADT (Part 2)

Dictionary.h

```cpp
#ifndef DICTIONARY_H
#define DICTIONARY_H

template <class K, class V>
class Dictionary {
    public:
        void insert(K & k, V & v);
        void remove(const K & k);
        V & find(const K & k) const;
    private:
        ...
};
#endif
```

Goals for Understanding Hashing:
1. We will define a **keyspace**, a (mathematical) description of the keys for a set of data.

2. We will define a function used to map the **keyspace** into a small set of integers.

CS 225 – Things To Be Doing:

1. Exam #7 (theory exam) is live!
2. MP5 is available now; extra credit +7 deadline is Monday
3. lab_btree starts today
4. Daily POTDs