Performance Boosts of Using a B-Tree II: Takeaways

Syntax

- Creating a simple B-Tree:

```
class Node:
    def __init__(self, keys=None, children=None):
        self.keys = keys or []
        self.children = children or []
    def is_leaf(self):
        return len(self.children) == 0
def __repr__(self):
    # Helpful method to keep track of Node keys.
    return "".format(self.keys)
class BTree:
    def __init__(self, t):
        self.t = t
        self.root = None
    def insert_multiple(self, keys):
        for key in keys:
            self.insert(key)
```

- Loading the contents of a csv into a B-Tree:

```
with open('amounts.csv', 'r') as f:
    reader = csv.reader(f)
    next(reader)
values = [float(l[0]) for l in reader]
btree = BTree(5)
btree.insert_multiple(values)
```

- Using the pickle module to save and load a B-tree:

```
import pickle
 # Save the model.
with open('btree_example.pickle', 'wb') as f:
    pickle.dump(btree, f)
 # Load the model.
with open('btree_example.pickle', 'rb') as f:
    new_btree = pickle.load(f)
```

Concepts

- Time complexity for loading in data into a B-Tree is .
- The pickle module allows us to save a B-Tree model and then load it every time we want to run a range query.
- The pickle module serializes Python objects into a series of bytes and then writes it out into a Python readable format (but not human readable).

Resources

- Pickle module