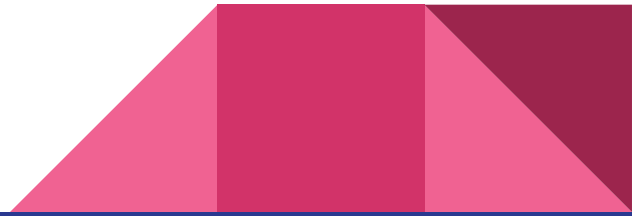
Policy Exploration of JITDs (C)

Team Twinkle

What we have available:

- Current implementation of cracking policy
  - crack
  - crack_one
  - pushdown_concats
  - crack_scan
- Current implementation of adaptive merge policy
  - gather_partitions
  - amerge
  - merge_partitions
  - extract_partitions
- Basic BTree test cases



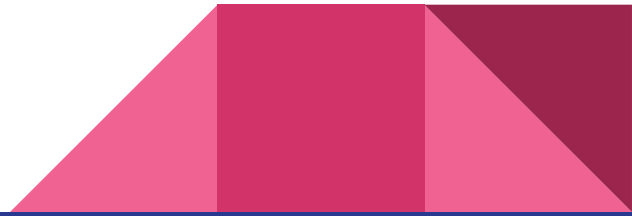
What we have implemented: (so far…)

- JITD - printing (mostly for debugging)

```
/**
 * Prints the internal representation of the JITD providing a detailed layout
 * of the current cogs and data present within.
 * @param cog - the root cog
 * @param depth - depth of the current cog in the tree - set to 0 for root
 */
void printJITD(struct cog *c, int depth);
```

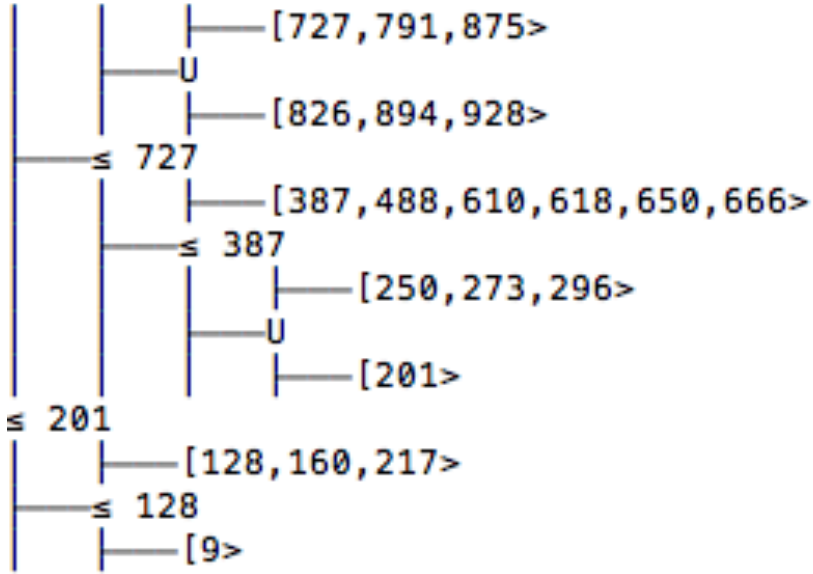
- Splaying

```
/**
 * The splay operation moves a given node to the root.
 * @param root - current root of the tree
 * @param node - node to be moved to the root
 * @return the new root of the rearranged tree
 */
struct cog *splay(struct cog *root, struct cog *node);
```



JITD - printing

- Great for debugging
- Great help for implementing splaying
- Shows cog type and data
- Reverse in-order

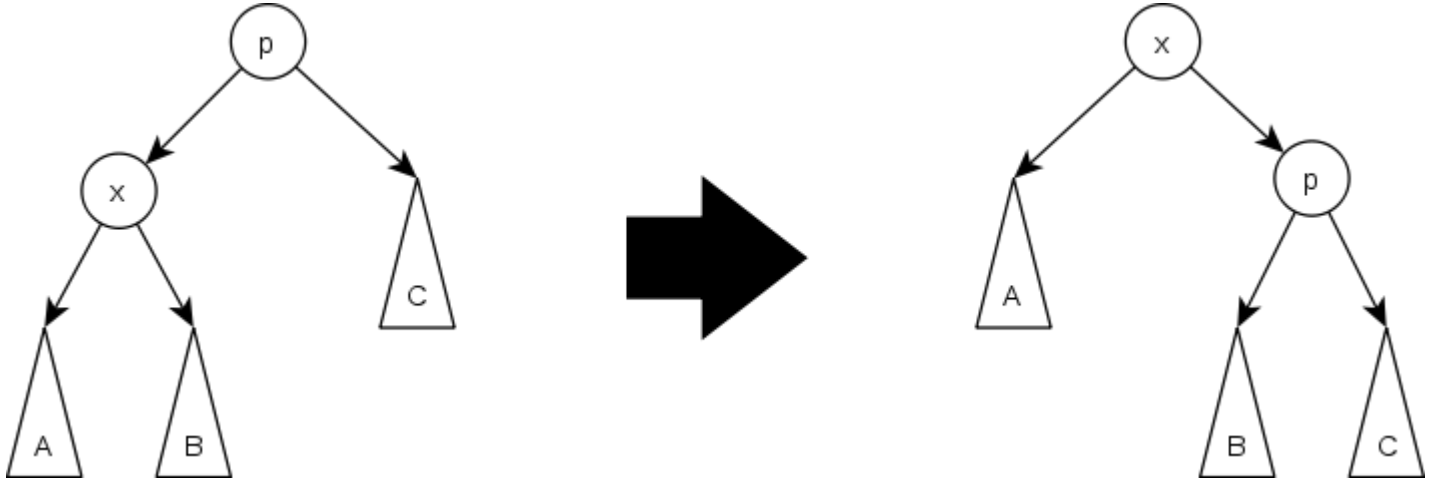


Splay Tree

- A splay tree is a self-adjusting **binary search tree**
- Additional property that recently accessed elements are quick to access again.
- Performs basic operations in O(log n) amortized time.
- For many sequences of non-random operations, splay trees perform better than other search trees, even when the specific pattern of the sequence is unknown.

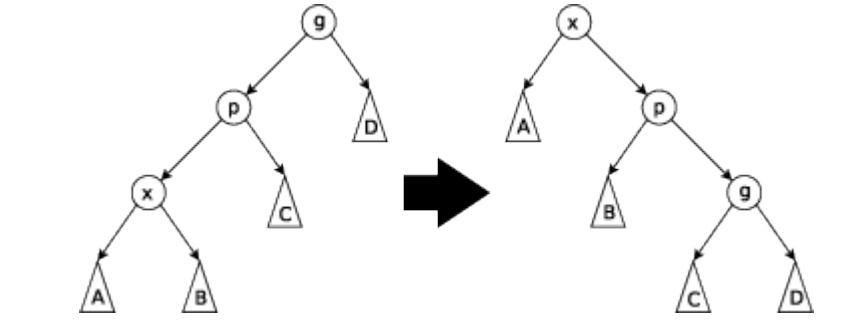
Splaying

- Zig - NOTE: Only done when the node we are moving is at an odd depth

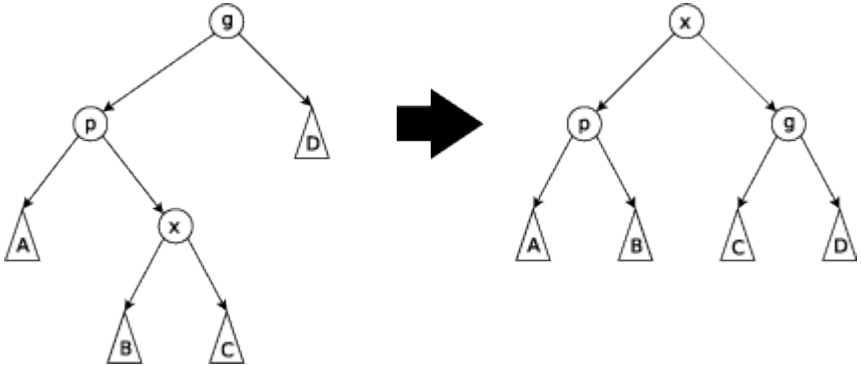


Splaying

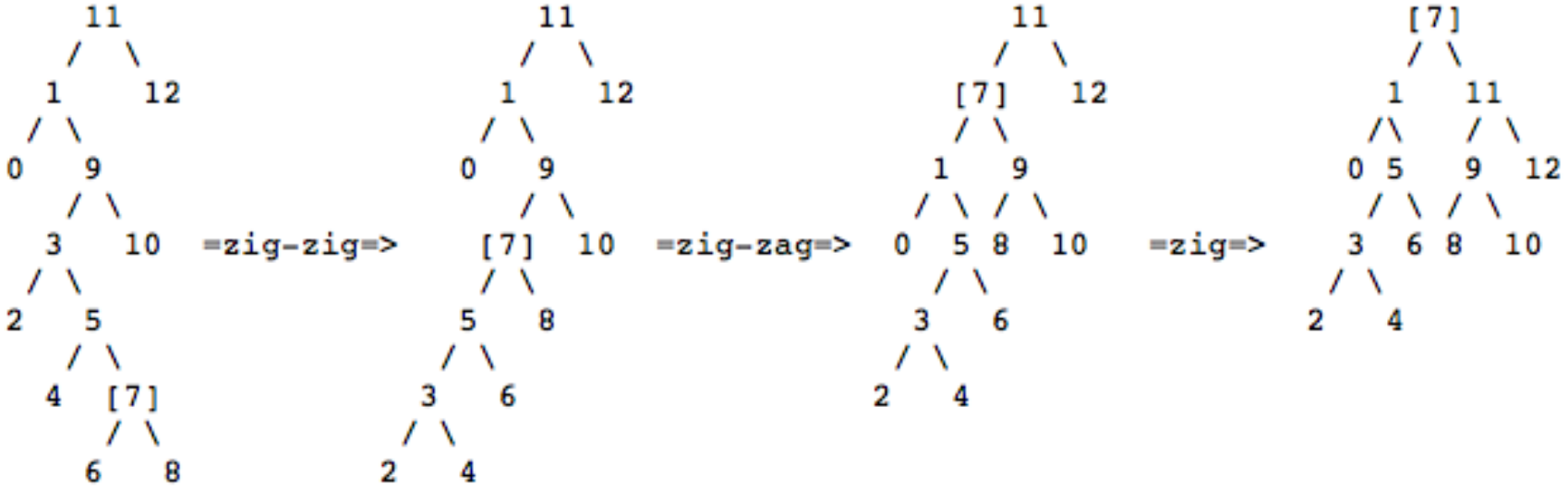
- Zig-Zig



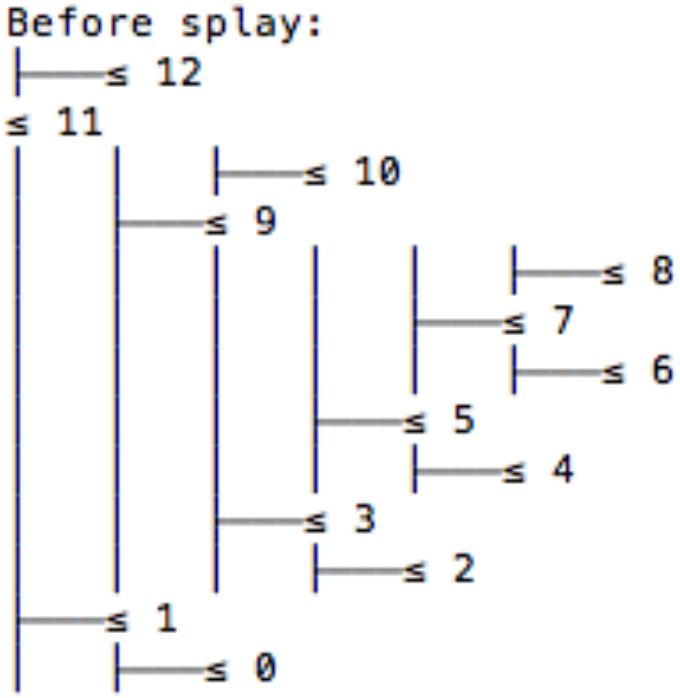
- Zig-Zag

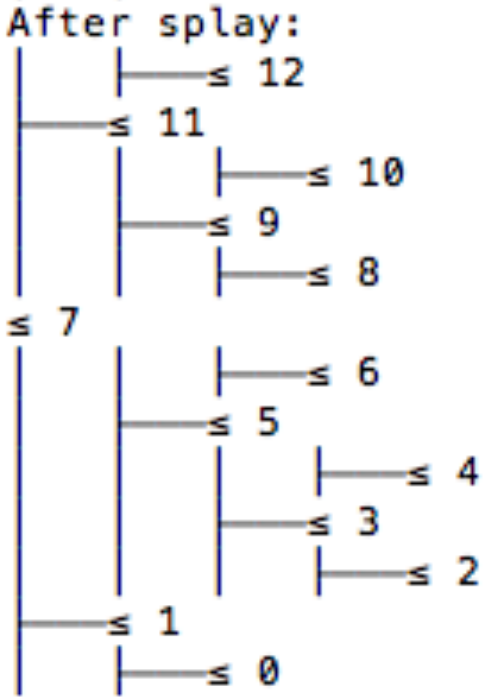


Splaying @ 7 - Example (Concept)



Splaying @ 7 - Example (Our flavor)

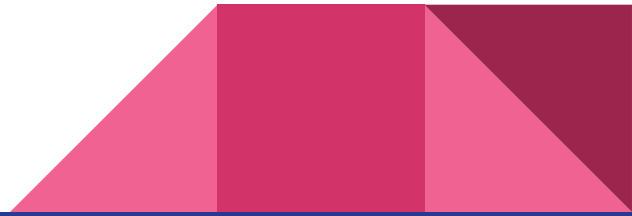




Cool stuff - seems to work!

Okay… what now?

- No real performance tests (or testing framework)
- Make some performance tests for the standard approach
- Make some performance tests for splaying
- Observe results and ponder! :D
- Figure out a sweet spot for splaying
- Implement a neat policy for splaying
- And on to other policies and interesting data structures (LSM tree, Prefix trie, HashTable, etc.)



Questions ???

Policy Exploration for JITDs (Java)

by

Team Datum

What we have done till now..

- Looked into the Java implementation of JITD policies.
- Trying to replicate the existing experimental results.

Next steps..

- To analyze the current implementation extensively using other benchmarking workloads like YCSB.
- Will look into the behavior of Splay Trees so as to check where it can fit into the current implementation

Splay Tree

Working:

https://www.cs.usfca.edu/~galles/visualization/SplayTree.html

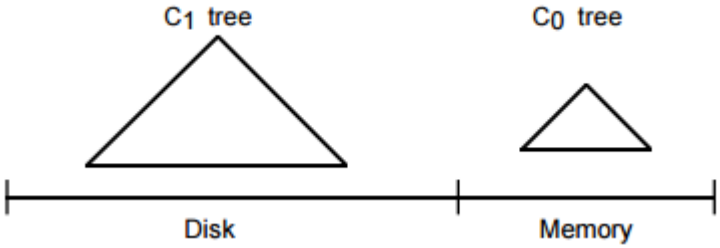
Splay Tree over Binary search Tree and Sorted Arrays

Time Complexity	Sorted Arrays	Splay Tree
Insertion	n	log n
Deletion	n	log n
Search or Scanning	log n	log n

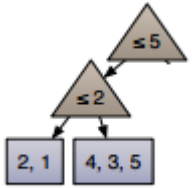
JITDs on Disk

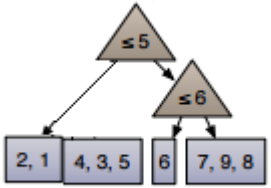
Team Warp

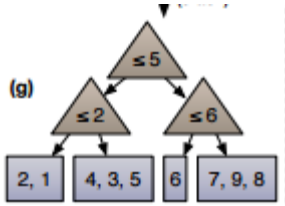
Animesh, Archit, Rishabh, Rohit



Data,5,Null	Data,6,Data



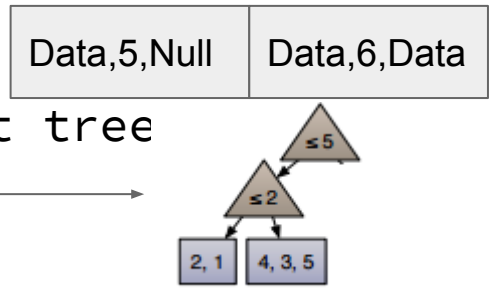




Range lookups

- Fetch data if not in memory. Reconstruct required part of the tree. Problems?
- Index it and flush when needed. What to flush? What to keep?
- Merge it later. When to merge?

Ex- for where condition value<=2
Assume memory is empty and index on file is



- Fetch Data,5,Null from file and construct tree
- Index it in memory to produce
- Flush it to new file Data,2,Data | Null,5,Null
- Merge it with the old file accordingly

Different FILE formats

Data,2,Data	Null,5,Null	Data,6,Data

File,2,File	Null,5,Null	File,6,File

Offset,2,Offset	Null,5,Null	Offset,6,Offset

Data,2,Data	Null,5,Null	Data,6,Data

- Ideal for sequential access

BUT

- Size of data unknown
- Binary search is difficult
- Store offset data for random access during binary search

File,2,File	Null,5,Null	File,6,File

- Binary search is easy

BUT

- As index grows file size becomes smaller
- Will cause more disk seeks even for range scan queries

Offset,2,Offset	Null,5,Null	Offset,6,Offset

- Binary search is easy

BUT

- Maintenance overheads
- Every crack will trigger entire file rewrite

Hybrid Approach !!