Exam Conflicts are due this week:

1. Put all course meetings, quizzes, and exams in your calendar
2. Report any conflicts with cs400 exams by Friday of this week
3. Report complete information via the Exam Conflicts Form.

GitHub (before class this week)

1. Go to GitHub.com
2. Create a GitHub.com account using your wisc.edu
3. Record your GitHub user name (and remember your password)
4. install Student Pack (unlimited free private repositories)

Week 5

ASSIGNMENTS
h3 available due before 10pm on Monday 2/18 (week 5)
p2 available due before 10pm on Thursday 2/21 (week 5)
h4 available soon and due before 10pm on Monday 2/25 (week 6)
x2 available due before 10pm on Thursday 2/28 (week 6)

Peer Mentors: will help student teams with git and GitHub , 2-3, B- Tree

Module: Week 5 (start on week 6 before next week)

THIS WEEK

- Version Control
  - **git**
  - **GitHub**
- B-Trees
  - **2-3 Tree**
  - **2-3-4 Tree**
  - **B+ Trees**

NEXT WEEK

- **x2 due next week (don't wait to meet -- get it done early)**
- **Hashing and Hash Tables**

Version Control

In the beginning...

backups
- good for one user one file → no crash.
- must save someone else

hello.java
hello1.java
hello.java.bak

cloud
Internet
CS server

Is there better way?
" A program

What do we need?

save multiple copy (versions)
organize and name for us
restore / return to earlier version – crash, bug
a way to share with teammates.

Why would we want to return to an earlier version?

crash
new bug was introduced
change mind about feature or direction.
want a way to experiment

Version Control Software

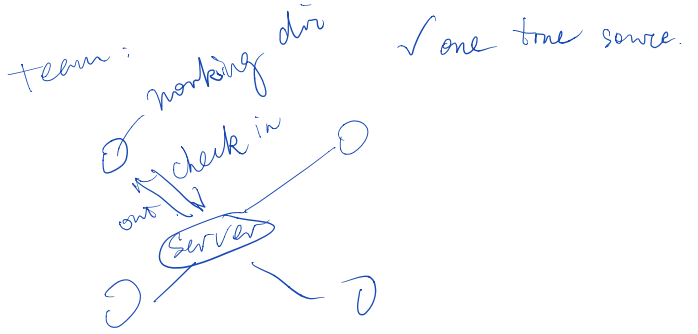
(system)
a program to track changes
allow return to earlier.
allow teams to work together.

Popular options

Git.
Mercurial
SVN
CVS

Centralized version control

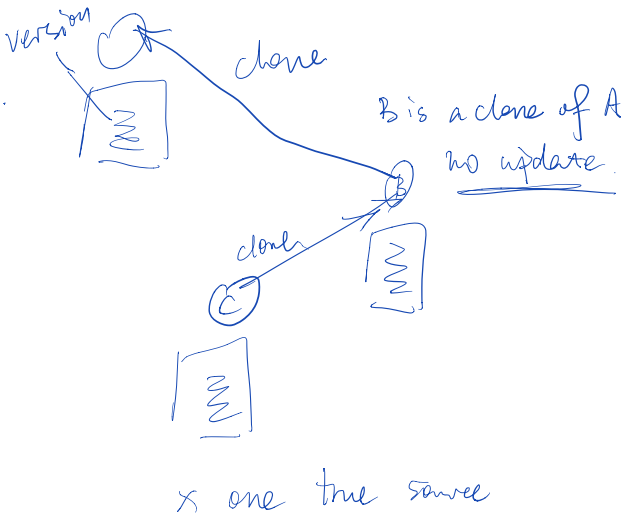
- a single source repository
- client - server
- source is on server.
  clients check-in, check-out, edit



Distributed version control

each copy is a clone of file, track changes.

changes are all local, do not change origin.



Version Control Terminology

version (a.ka. revision) - name (or number) for a given copy

repository - files, tracking data, configurations, etc. that is being tracked

checking in - put new and changed files into a repository. (git commit)

checking out - get latest version of files from your repository (git checkout abdf)

local repository - a repository that is on your file system

remote repository - a repository on a different computer or network

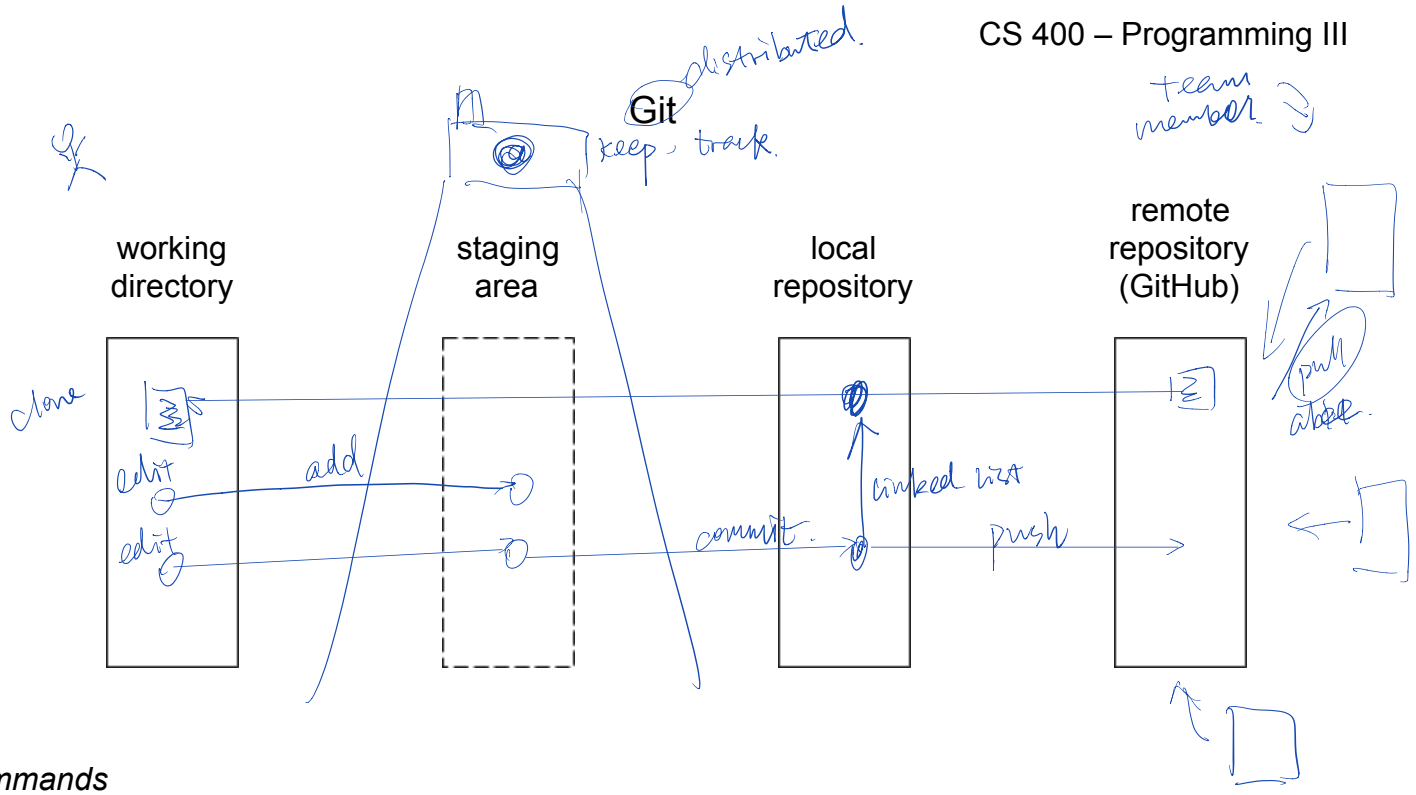
commit - saves and names changes as a new "version" in the repository

working copy - the particular version or workspace that you are using and editing

merge - add your changes to a repository modified by other users

conflict - when merge fails because your changes overlap changes by another user

resolution - choose which content will prevail between merge conflicts. -- mostly solved by a person not program.



git commands

clone
status
(log)
add
commit
push
pull
(config)

Public vs Private

public – anyone can clone, only collaborate can push. +pull

private – nobody – pull, pull, clone

git and GitHub demo

GitHub	Teammate: Bucky	Teammate: Becky
create repo		
	git clone cd repo-name [create todo] git status git add todo.txt [edit todo] git status git add todo git status git commit git status git log	
		git clone ls git status git log
	git push (must login)	
		git pull git log
	[edit todo.txt] [add test empty]	[edit todo] [add test insert]
	git add . git commit -m "add test empty"	git add todo.txt git commit -m "add test one" git push
	git push [merge conflict] gti pull [edit todo] git add . git commit git push	
		git pull

git config --global user.name "user email"

B-Trees

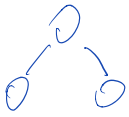
maintain balance by modifying node structure

Properties

- H o(logN) $\log_m N$ × #nodes
  - branching factors.
- general tree
- insert all leaf nodes.
- non-leaf (interior) 2 or more children

0 leaf node, no children

Node types

2-node  one key, 2 children

3-node  2 key, 3 children

4-node  3 key, 4 children

n-node n-1 key. n children

k-ary k child (max number).

B-Tree Structure

```
public class BTree<K extends Comparable<K>,V> implements DataStructureADT<K,V>{
	private int branchingFactor;
	private BTreeNode<K,V> root;
	private class BTreeNode<K,V> {
		private List<K> keyList;
		private List<BTreeNode<K,V>> childList;
		private BTreeNode() {
			keyList = new ArrayList<K>();
			childList = new ArrayList<BTreeNode<K,V>>();
		}
	}

	public BTree() {
		this(3);
	}

	public BTree(int branchingFactor) {
		this.branchingFactor = branchingFactor;
		root = null;  // empty tree at start
	}
```

inner class

+ private List<V> value list.

⇐ default no-arg constructor.

(max 3 children)

↑ bf = 3

⇐ user chose branching factor.

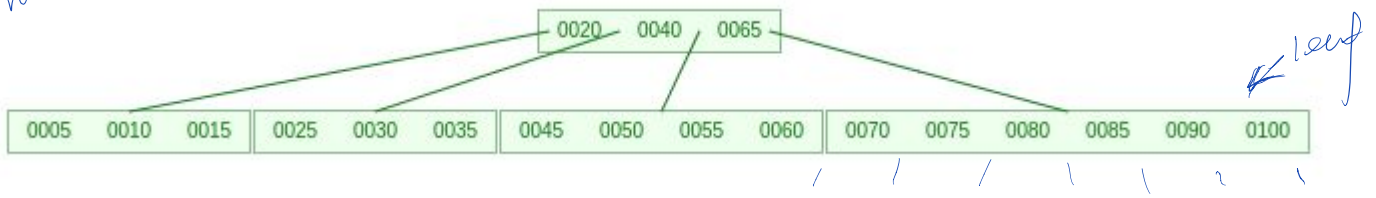
Branching Factor and Databases

- database: span multiple external storage devices ⇒ slow
- choose a branching factor that minimizes disk access and maximize # of keys that can be processes

Branching Factor and height

$m = 7$ nodes have ϕ, 2-7 children

has keys in leaf node and interior.



leaf

6 keys - 7 children

keys are always inserted in leaf

It is $O(\log N)$

$O(\log_m N)$ ← N: # of nodes
m: branching factor.

B-Tree Operations

m=3, left, mid, right.

print_inorder() print_inorder (root)

private void print_inorder (BTNode<K, V> node){
if (node == null)
return;
if node is a leaf, print key inorder.
else if node has 3 children.
print_inorder (node. left). // print left subtree
print (Kleft) // print left key. of this node.
(node. midchild)
(key)
(node. rightchild).
(Rkey)
else print_inorder (node. left)
print (Kleft)
(node. midchild)
}

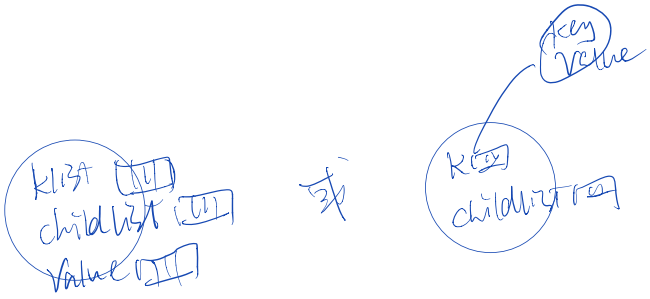
lookup(key)

private V lookUp (BTNode <K,V> node , K key) {
if root is null, return null;
if node is a leaf , search through k list, return matching value.
else search through key list. for loop.
if nkey in key list, match, return nkey.
> key
return lookup (n. childlist . get(i), key);
else, return lookup (n. childlist. get (i+1), key);

Inserting into a 2-3 Tree

If T is Empty

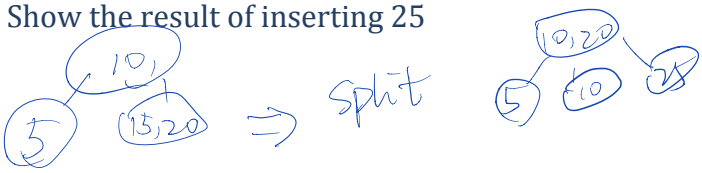
If T is non-empty



2-3 Tree Insert Practice (grows "up")

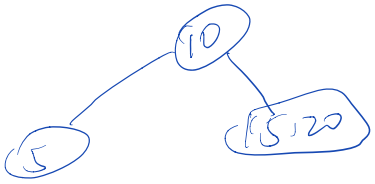
m = 3
most 2 key, 3 children

4) Show the result of inserting 25



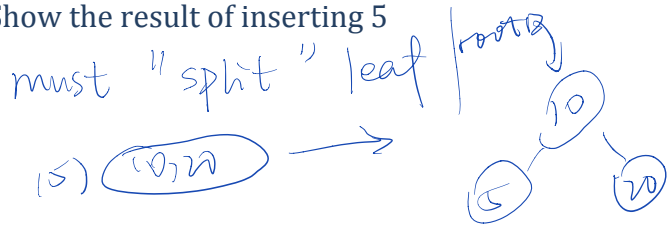
3) Show the result of inserting 15

Switching



2) Show the result of inserting 5

must "split" leaf



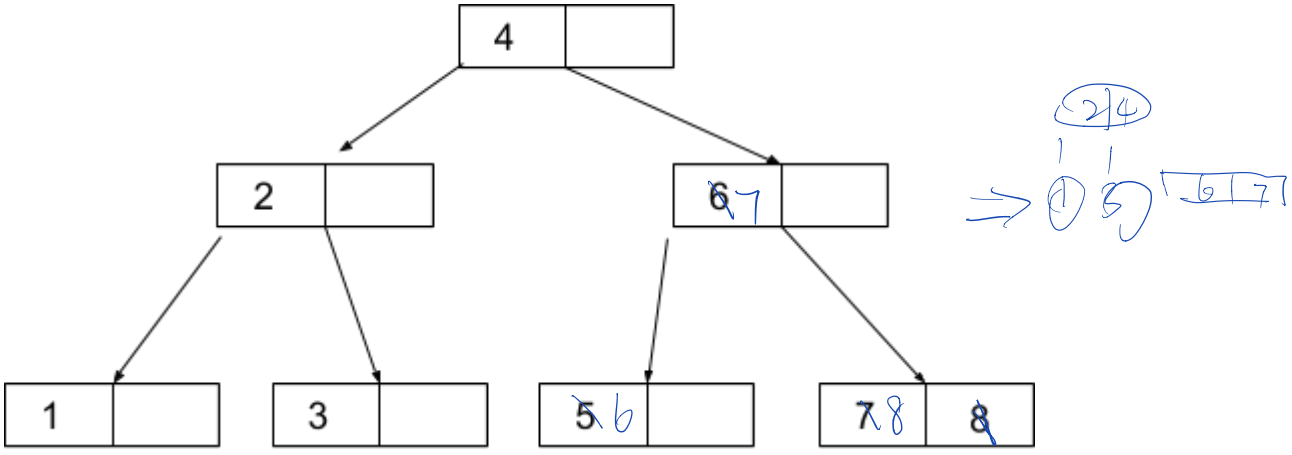
1) Start with empty tree, show tree that results from inserting 10 and 20.

2-3 Delete

find parent of leaf node with key to delete.

if leaf has more than 1 key, delete key

only 1 key, - borrow (merge) from a sibling, parent.



delete 5

5 borrows 6

6 borrows 7.

delete 8

merge 6 with 7.

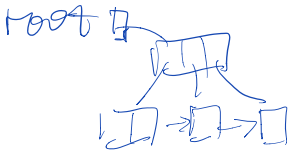
(same efficiency)
equivalent to RBT

2-3-4 Tree

- B tree with 2, 3, and 4 nodes, max 4 children.
- let node have 4 children
- split occurs the next time you reach the node
- pre emptive split. before adding (key, value) pair

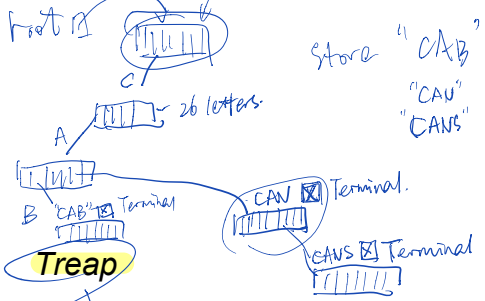
B+ Tree

- great for range search.
- allow duplicates
- internal node has "keys" directional values and children.
- leaf nodes contain (key, value) pair.
- leaf nodes are linked list



Other Search Trees

Trie tree – B tree with 26 node (array) easy for search.

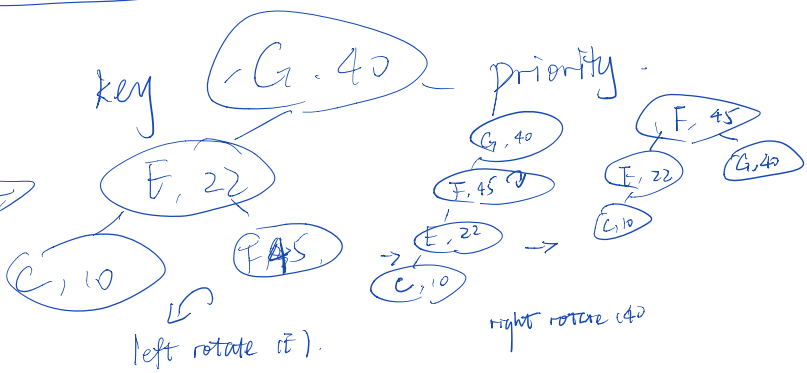


Treap

Tree + Heap

wikipedia

Index



- To search for a given key value, apply binary search algorithm in a binary search tree, ignoring the priorities.
- To insert a new key *x* into the treap, generate a random priority *y* for *x*. Binary search for *x* in the tree, and create a new node at the leaf position where the binary search determines a node for *x*should exist. Then, as long as *x* is not the root of the tree and has a larger priority number than its parent *z*, perform a tree rotation that reverses the parent-child relation between *x* and *z*.
- To delete a node *x* from the treap, if *x* is a leaf of the tree, simply remove it. If *x* has a single child *z*, remove *x* from the tree and make *z* be the child of the parent of *x* (or make *z* the root of the tree if *x* had no parent). Finally, if *x* has two children, swap its position in the tree with the position of its immediate successor *z* in the sorted order, resulting in one of the previous cases. In this final case, the swap may violate the heap-ordering property for *z*, so additional rotations may need to be performed to restore this property.

BST + random rotation
to achieve balance
sometimes.

skiplist