COSI 127B
Introduction to Databases
Programming Assignment 3

Help Session

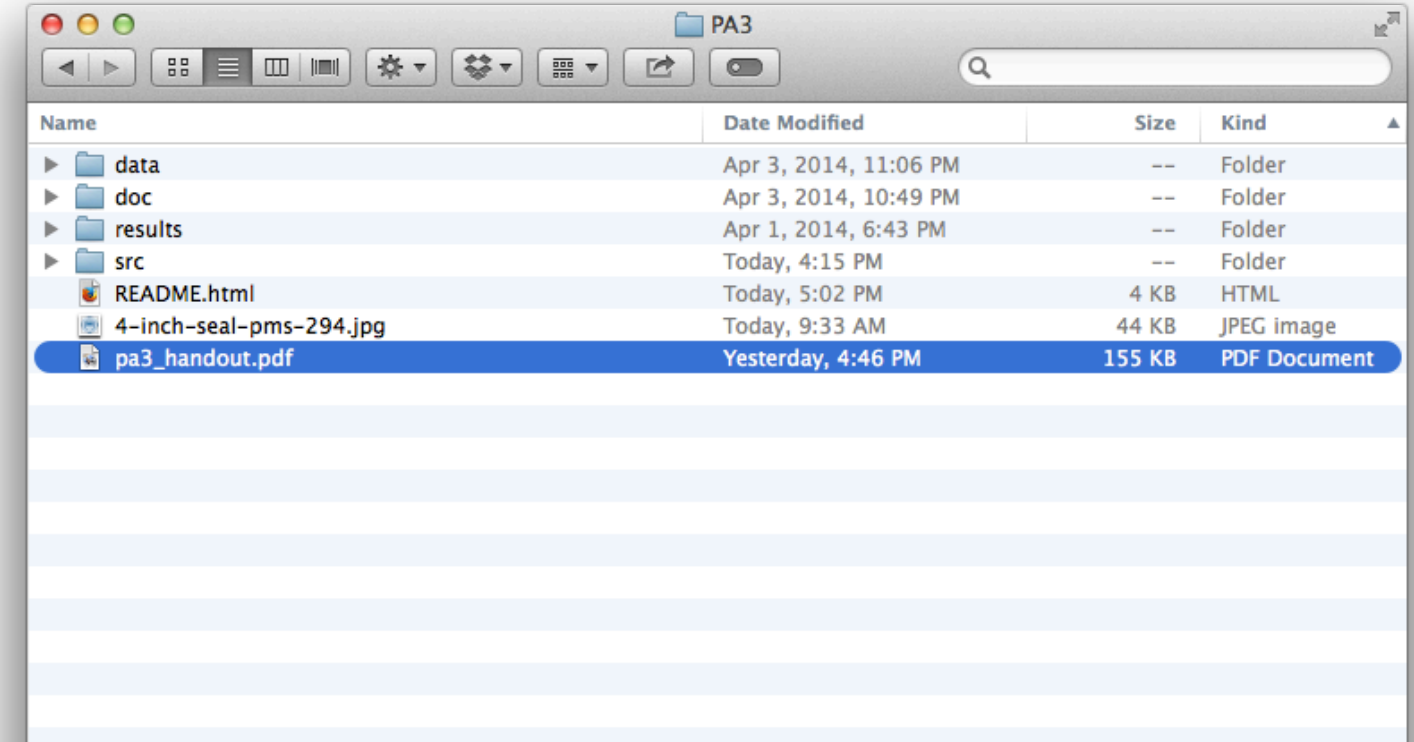
April 7, 2014

Assignment Overview

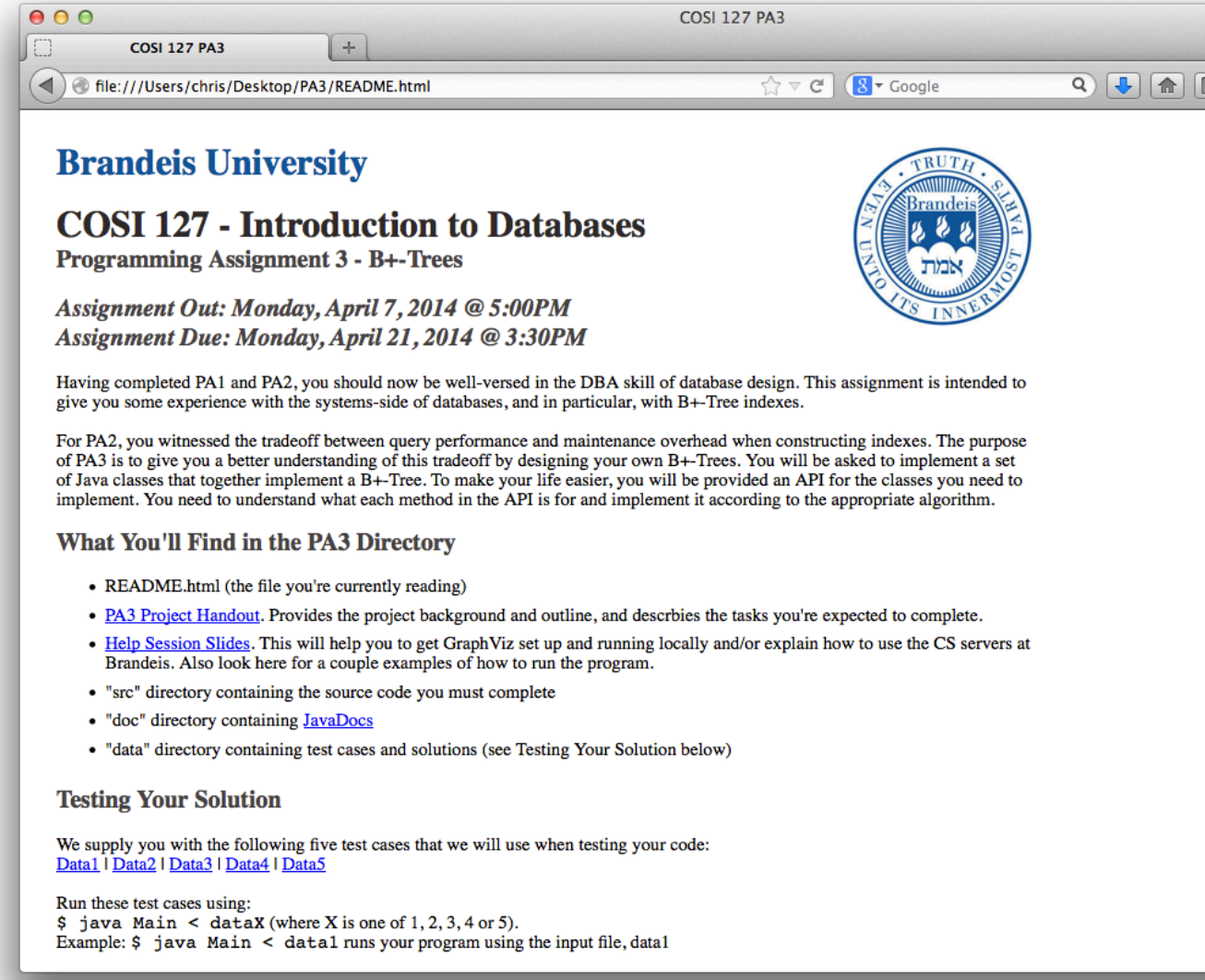
- A Java implementation of B+-Trees
- Requires you to finish off some classes that are partially complete
- Handout and javadocs can be used as a reference
- You will have 2 weeks to complete this assignment

Getting the Code

- This part is pretty simple
  - Download from Latte and unzip



README.html


COSI 127 PA3
COSI 127 PA3
file:///Users/chris/Desktop/PA3/README.html
Google
Brandeis University
COSI 127 - Introduction to Databases
Programming Assignment 3 - B+-Trees
Assignment Out: Monday, April 7, 2014 @ 5:00PM
Assignment Due: Monday, April 21, 2014 @ 3:30PM
Having completed PA1 and PA2, you should now be well-versed in the DBA skill of database design. This assignment is intended to give you some experience with the systems-side of databases, and in particular, with B+-Tree indexes.
For PA2, you witnessed the tradeoff between query performance and maintenance overhead when constructing indexes. The purpose of PA3 is to give you a better understanding of this tradeoff by designing your own B+-Trees. You will be asked to implement a set of Java classes that together implement a B+-Tree. To make your life easier, you will be provided an API for the classes you need to implement. You need to understand what each method in the API is for and implement it according to the appropriate algorithm.
What You'll Find in the PA3 Directory
README.html (the file you're currently reading)
PA3 Project Handout. Provides the project background and outline, and descrbies the tasks you're expected to complete.
Help Session Slides. This will help you to get GraphViz set up and running locally and/or explain how to use the CS servers at Brandeis. Also look here for a couple examples of how to run the program.
"src" directory containing the source code you must complete
"doc" directory containing JavaDocs
"data" directory containing test cases and solutions (see Testing Your Solution below)
Testing Your Solution
We supply you with the following five test cases that we will use when testing your code:
Data1 | Data2 | Data3 | Data4 | Data5
Run these test cases using:
$ java Main < dataX (where X is one of 1, 2, 3, 4 or 5).
Example: $ java Main < data1 runs your program using the input file, data1

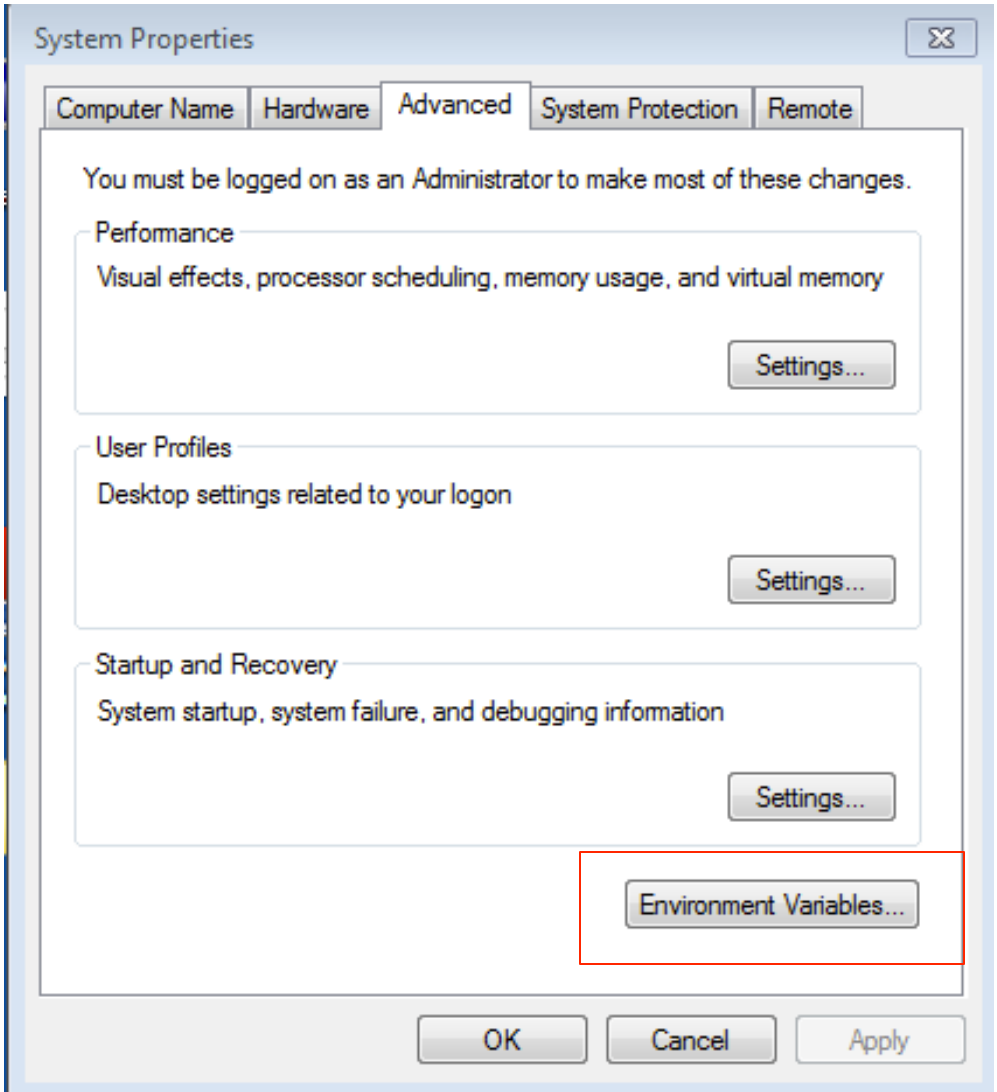
Dependencies

- Java
  - Found on all departmental Mac/Linux machines if you don't already have it
- GraphViz
  - Open source visualization software
  - Straight-forward installations available from:
    http:www.//graphviz.org
  - Will need to update the Path variable if using Windows…

Setting Windows Path for GraphViz

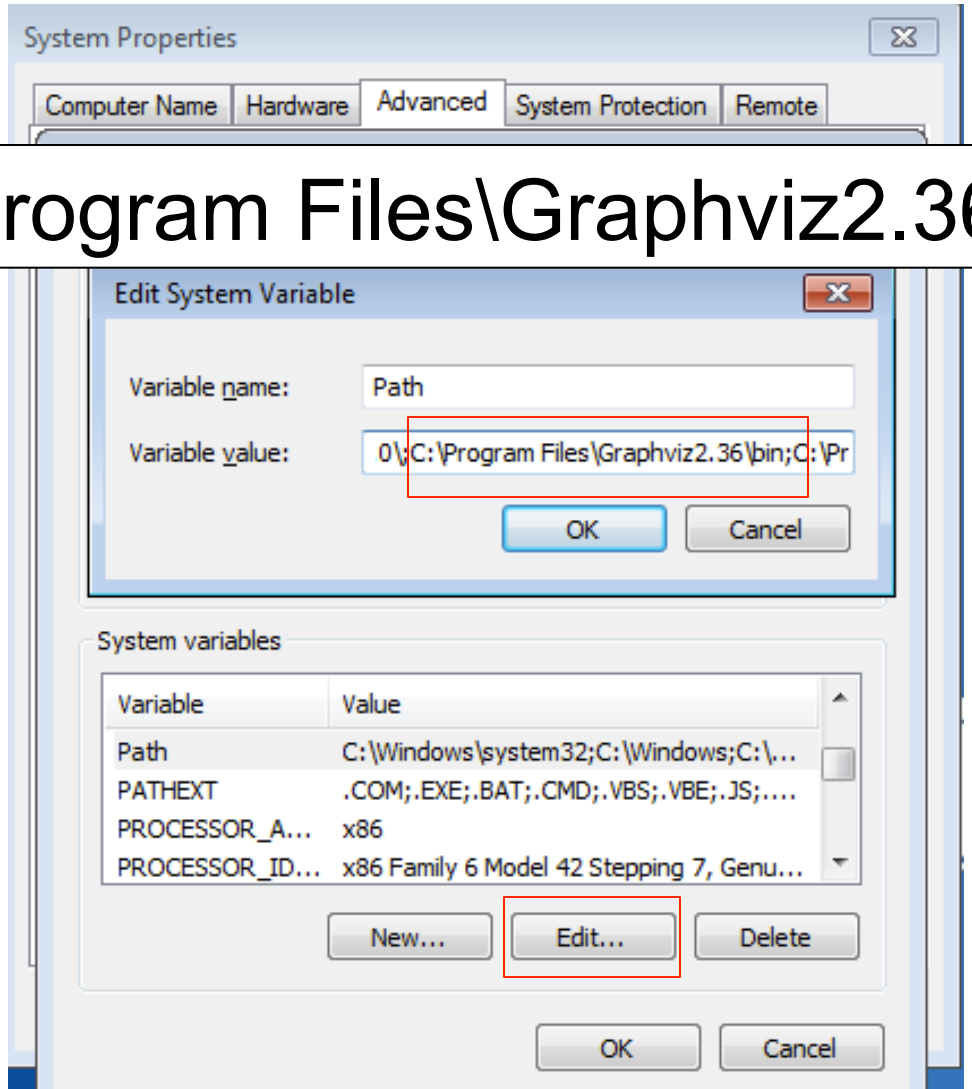
- Right-Click ‘My Computer’, and select ‘Preferences’
- Select ‘Advanced’ tab and choose ‘Environment Variables’
- Find ‘Path’ and click ‘Edit’
- Insert the path to the bin directory of your GraphViz install

Setting Windows Path for GraphViz



Setting Windows Path for GraphViz

C:\Program Files\Graphviz2.36\bin;



Testing GraphViz

- You can test GraphViz with the following command at command line, using the file graphvizTest.dot from the PA3/data directory

$ dot –Tpdf –o output.pdf graphvizTest.dot

- If working properly, a PDF will be created that resembles data1_result.pdf in the results directory

Main.java

- Interprets user input during tree creation and acts accordingly
- ‘o’ option outputs the resulting tree to file and displays it

```
…
// Prepares Runtime

Runtime r = Runtime.getRuntime();
// CREATE THE GRAPH
// To create a PDF
Process dot = r.exec("dot -Tpdf -o out.pdf tree.dot");

// To create PostScript
Process dot = r.exec("dot -Tps -o out.ps tree.dot");

…
```

```
…
// DISPLAY THE GRAPH
// Use this on Windows
//r.exec("rundll32 url.dll,FileProtocolHandler out.pdf");

// Use this on Mac
r.exec("open out.pdf");

// Use this on Linux (requires gv to be installed)
// r.exec("gv out.ps")
```

Summary

- You should be able to configure your desktop or laptop to work on this without needing the Brandeis servers
  - Install GraphViz, make sure it's working from command line
  - Determine which output format you would like to use (I used PDF. See GraphViz docs for more)
  - Pick your display command in Main.java
- Still one more option: CS servers

Using Brandeis CS Servers via SSH

- All RedHat servers are equipped with GraphViz
  - diadem, tiara, helios, etc.
- You will need to use X Windows system and X11 forwarding to display graphs remotely, but creating the output .pdf or .ps files will work fine

Using Brandeis CS Servers via SSH

(transfer files to Brandeis servers first. See next slides)

To Create AND Display Remotely:

```
$ ssh -X johndoe@diadem.cs.brandeis.edu
$ cd pa3Directory
$ javac *.java
$ java Main < ../data/data1
$ gv out.ps &
```

<~~ This may be slow from off campus

(This will require an X11 client. I used Xquartz on OSX Mavericks.
It's also possible using Putty on Windows. See Google and links below for more information.)

Mac - https://xquartz.macosforge.org/landing/
Windows - http://www.math.umn.edu/systems_guide/putty_xwin32.html

Using gentree

- gentree is a script within your PA3 folder that can be used to generate random trees
- log in to any linux-based Brandeis server
- ./gentree X Y Z
  - X = degree of the tree
  - Y = ratio of insertions to deletions (0.75 is default)
  - Z = number of operations

Helpful Linux Commands

Creating a compressed tar file: tar cvzf pa3files.tar.gz src data

Transferring to Brandeis, *Remote* pa3 folder

scp pa3files.tar.gz johndoe@diadem.cs.brandeis.edu:/home/m/johndoe/pa3

Connecting to Brandeis: ssh johndoe@diadem.cs.brandeis.edu

Transferring Back from Brandeis

scp johndoe@diadem.cs.brandeis.edu:/home/m/johndoe/pa3 .

(Note that this will be useful for transferring sets generated using gentree back to your local workspace.)

Unpacking a compressed tar file: tar xvf myResults.tar.gz

Piping to output file: gentree 5 0.75 500 > testTree1

Insertion/Deletion in B+-Trees

- Doing some examples on paper will help you understand the algorithm before you start coding.

- Let's go through one of the tests, data2, step by step…

data2

- All data files (found in the data directory) define a set of insertions and deletions used to create a tree
- data2 instruction set:

n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

data2

- All data files (found in the data directory) define a set of insertions and deletions used to create a tree
- data2 instruction set:

n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

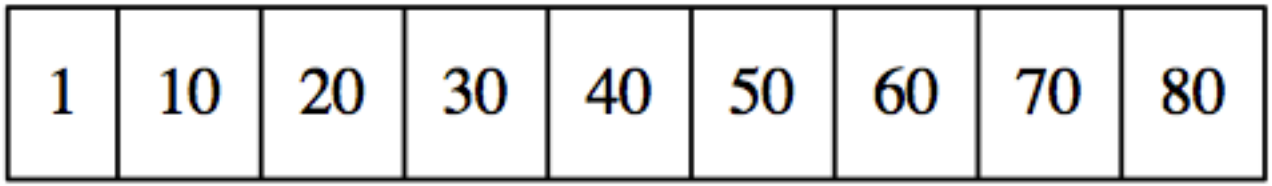
show the tree

degree = 10

so **internal nodes** must have a **minimum of** ceil(n/2) = **5 pointers**
and **leaf nodes** must have a **minimum of** ceil((n-1)/2) = **5 keys**

data2

n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o



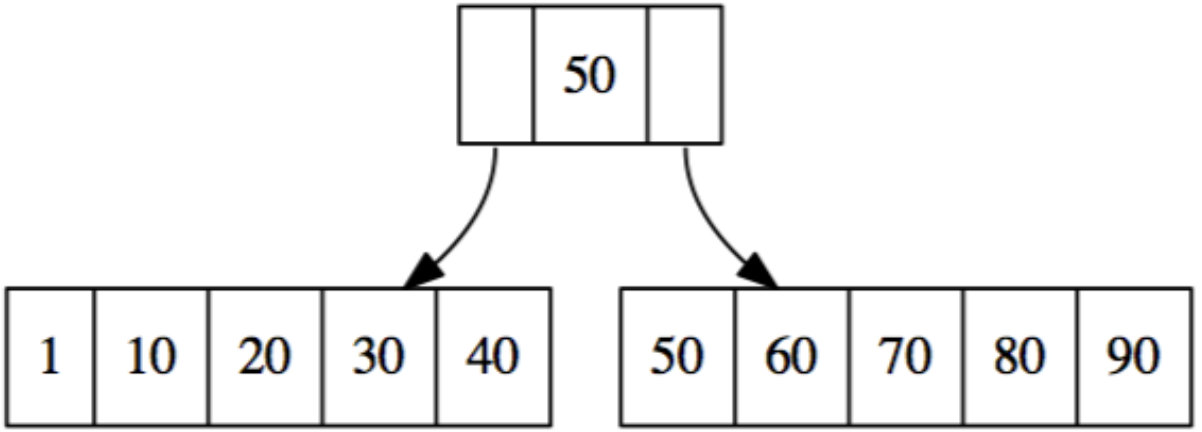
After inserting the first 9 values, our node is full.
To insert 90 we will split it.

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80~~**, i90**, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously

1	10	20	30	40	50	60	70	80

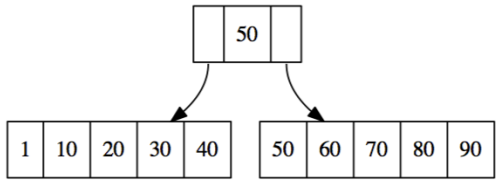


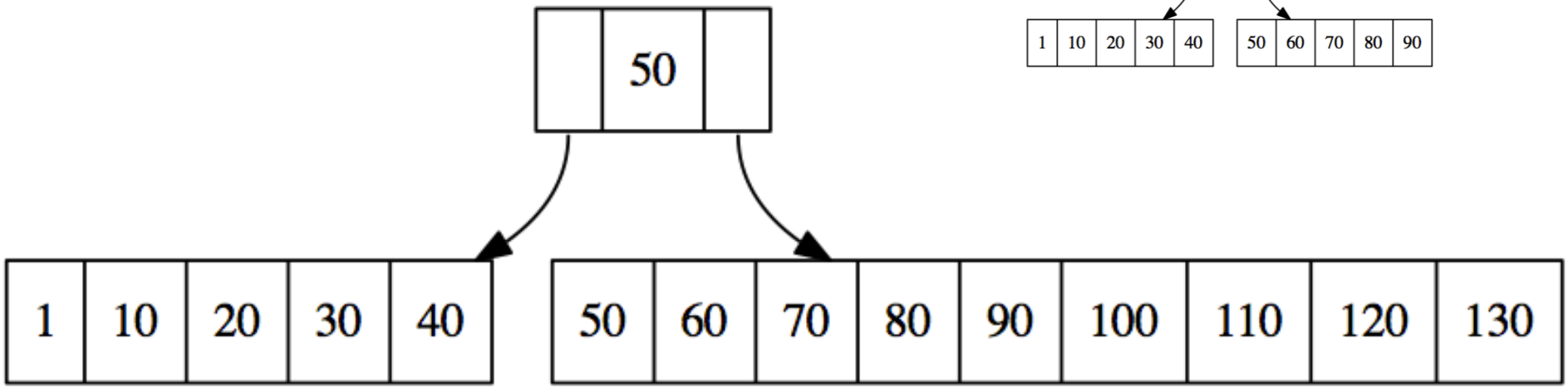
After splitting the nodes and adding the root. Notice the next few insertions will easily fit into our right leaf…

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90~~, **i100, i110, i120, i130**, i140, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



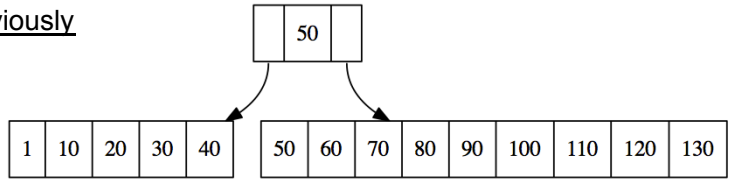


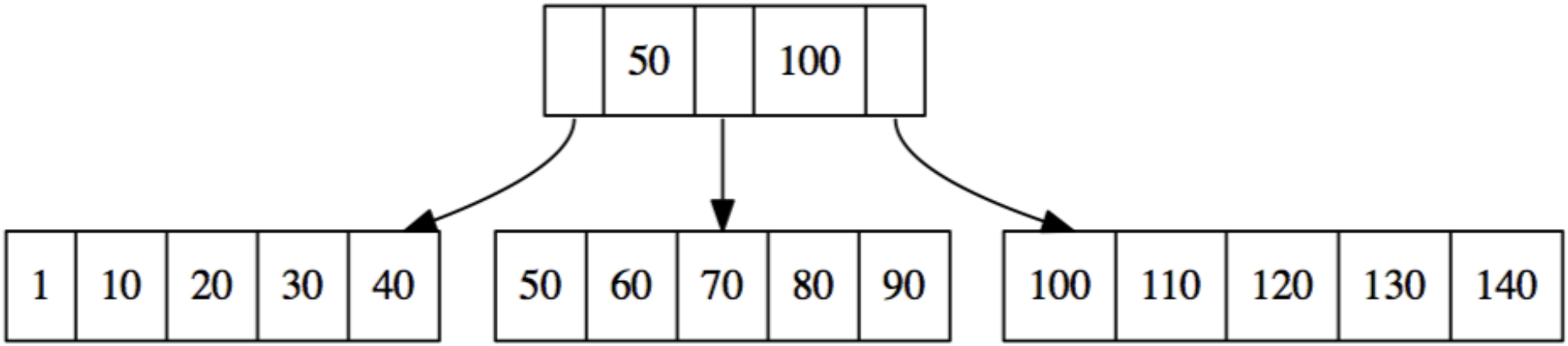
That was easy, but now we're full again.
For the next insertion, we'll have to split.

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130,~~ **i140**, i150, i160, i170, d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



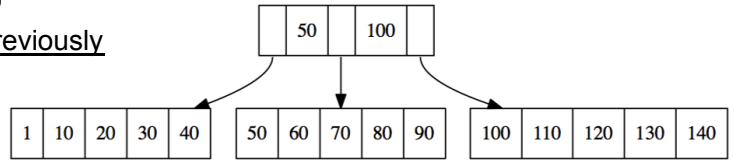


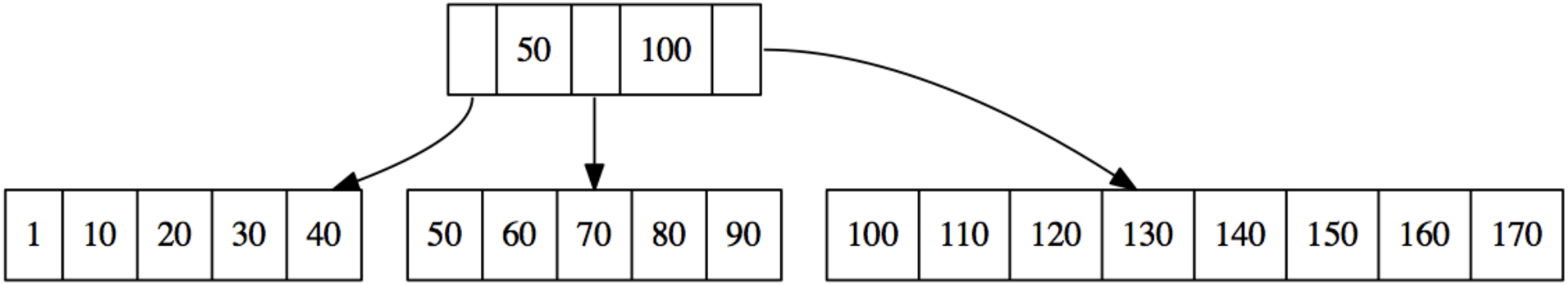
After the split it looks like we'll have room for 150, 160, 170…

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80,~~ ~~i90, i100, i110, i120, i130, i140~~, **i150, i160, i170,** d50, d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



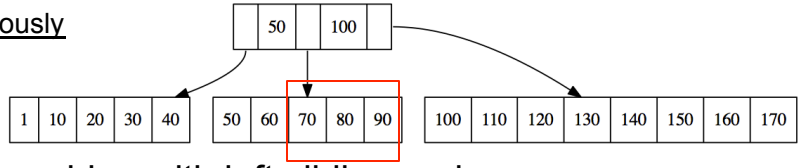


Those insertions were simple. Now let's delete 50… notice that the node after deleting 50 will be underfull.

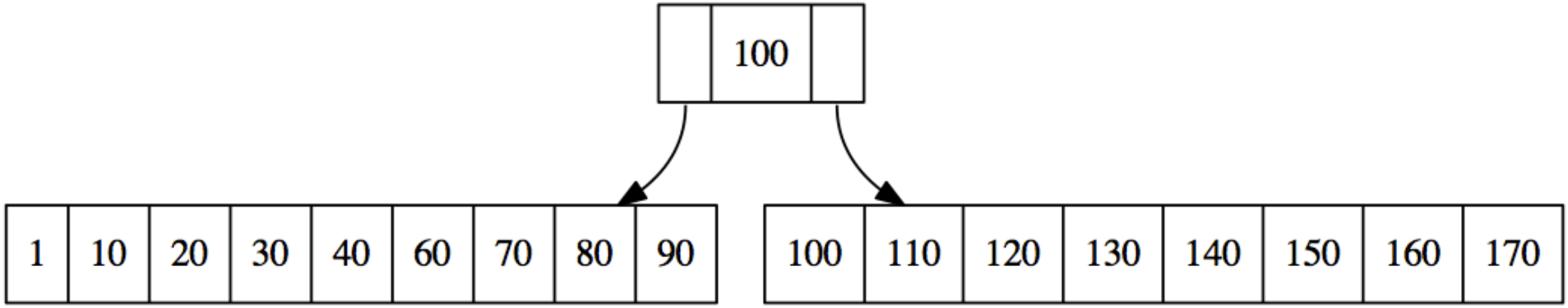
data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80,~~ ~~i90, i100, i110, i120, i130, i140, i150, i160, i170~~, **d50,** d100, d170, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



(Cannot combine right because it would be overfull. Instead we combine with left sibling and remove the root entry. Recursive deletions may be required with multiple levels)

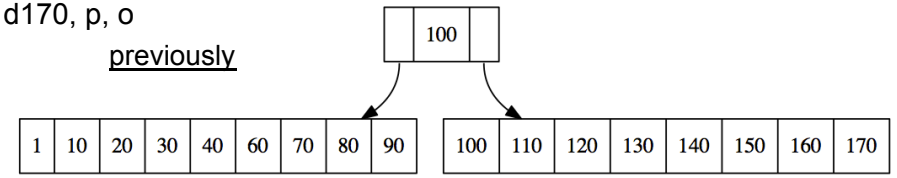


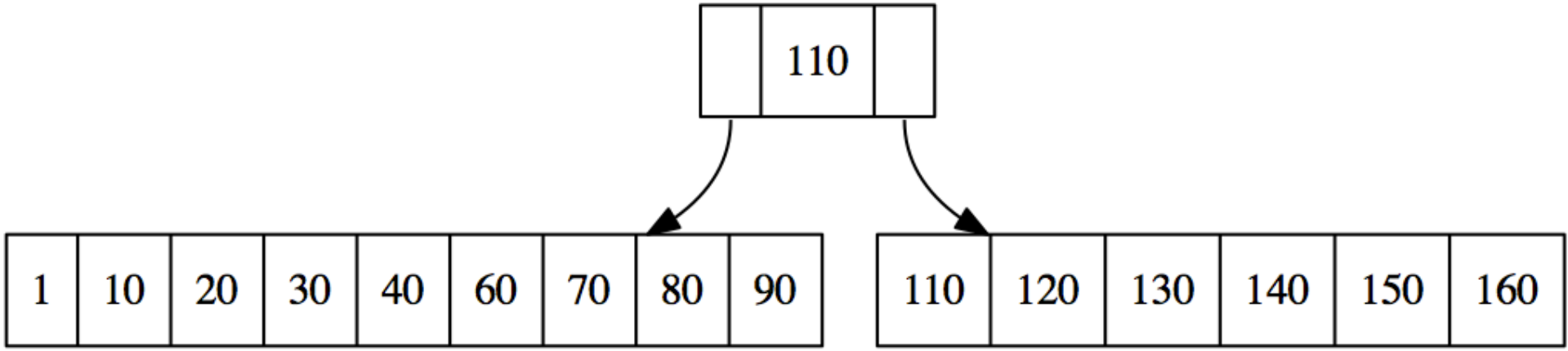
The next couple deletions will be easier since the node will not be underfull…

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80,~~ ~~i90, i100, i110, i120, i130, i140, i150, i160, i170, d50~~**, d100, d170**, i100, i170, d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously





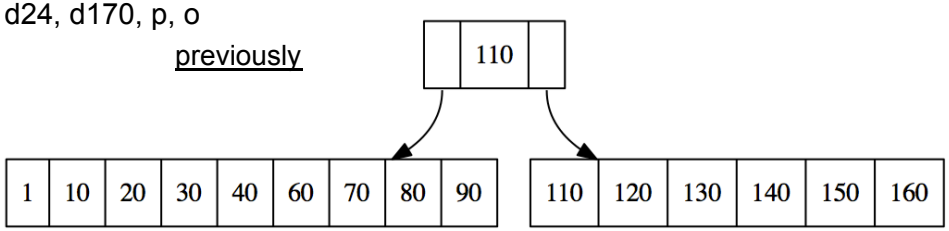
Note that we updated the root key value (see assignment requirements in the pa3 handout).

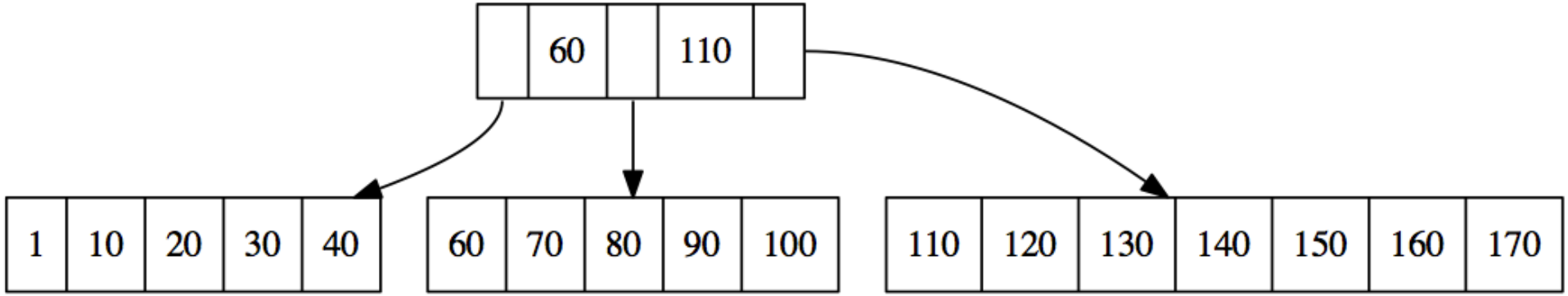
Next up: another split to insert 100..

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170,~~ **i100, i170,** d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



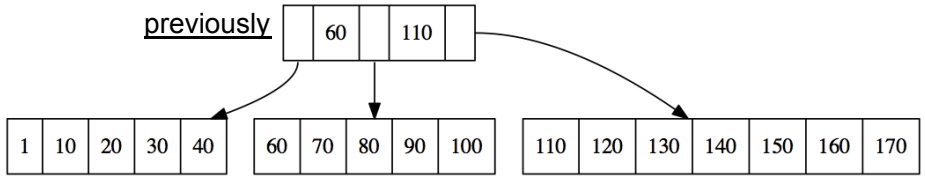


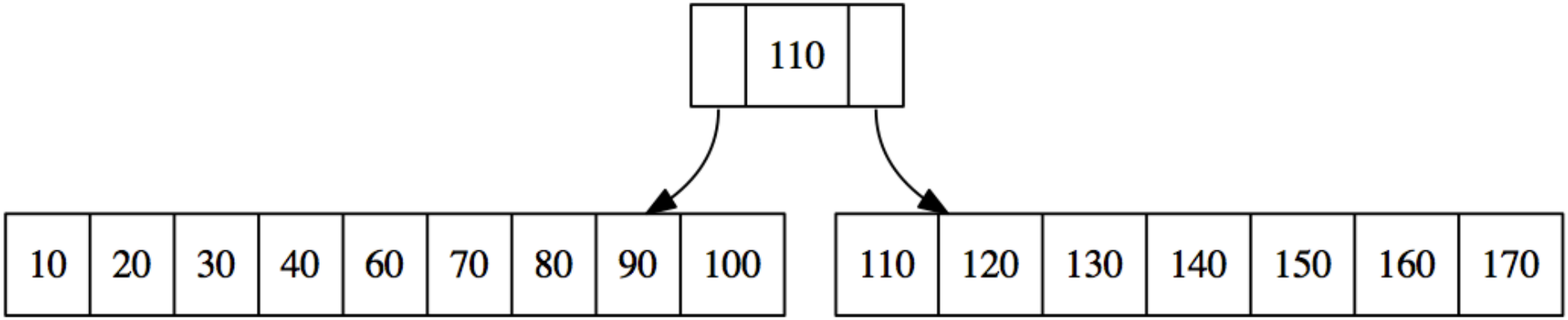
What will happen with our next instruction (delete 1)?

The leaf will be underfull, so we merge with its sibling to the right.

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170~~,**d1**, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o



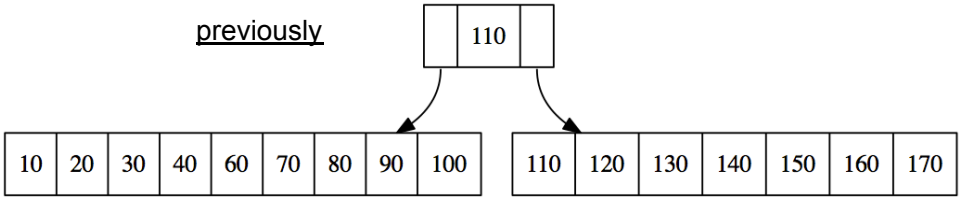


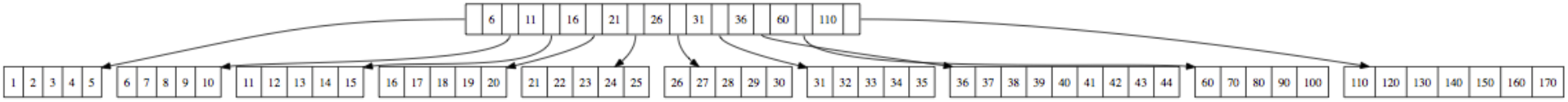
Coming up are a series of insertions and splits like we have been doing. Let's jump ahead a bit…

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170,d1~~, **i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44**, i45, i46, i47, d110, d21, d22, d23, d41, d24, d170, p, o

previously



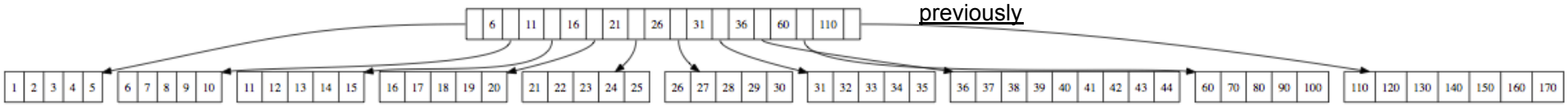


Why did I pause here?

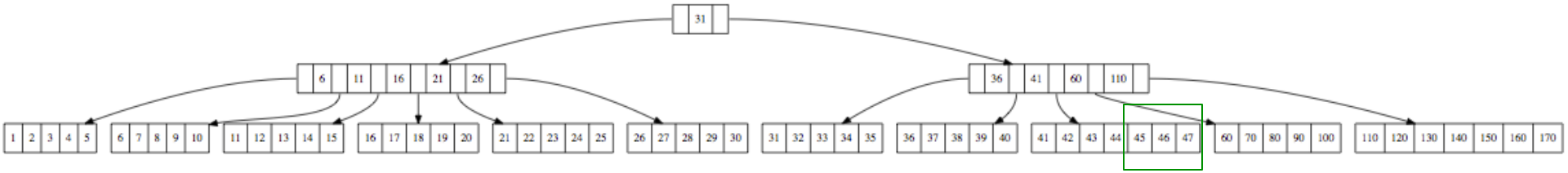
We have another split coming up, but the root is also full…

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170,d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44~~, **i45, i46, i47,** d110, d21, d22, d23, d41, d24, d170, p, o



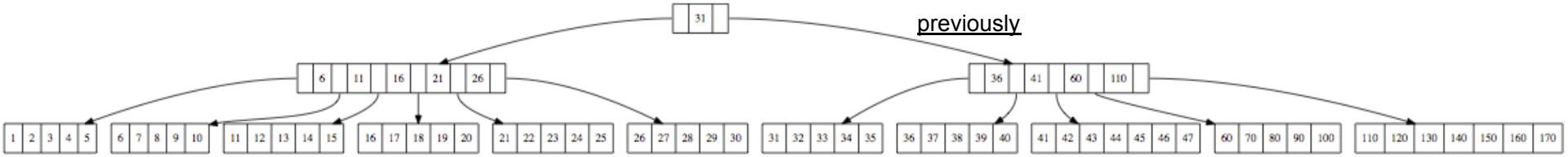
i45, i46, i47



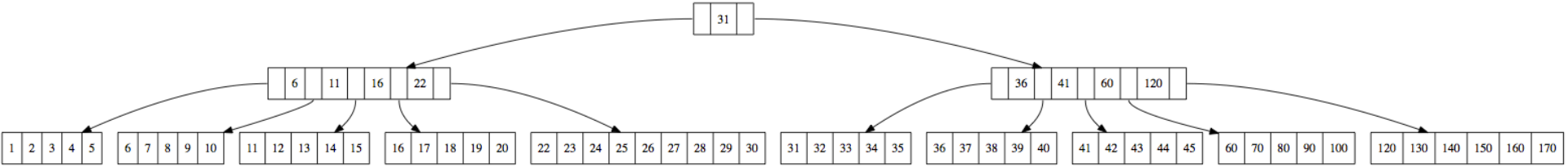
We split the internal node and redistributed values around the newly formed root. Note that the left internal node has more elements than the new one we created at that level (see requirements in handout).

data2

~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170,d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47~~, **d110, d21**, d22, d23, d41, d24, d170, p, o



d110, d21

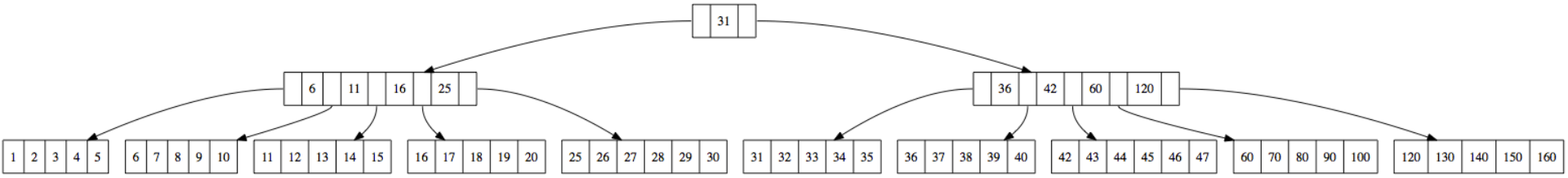


Note that deleting 21 resulted in an underfull Leaf node which was merged with it's right sibling, but the parent internal node still contained ceil(n/2) pointers.

data2

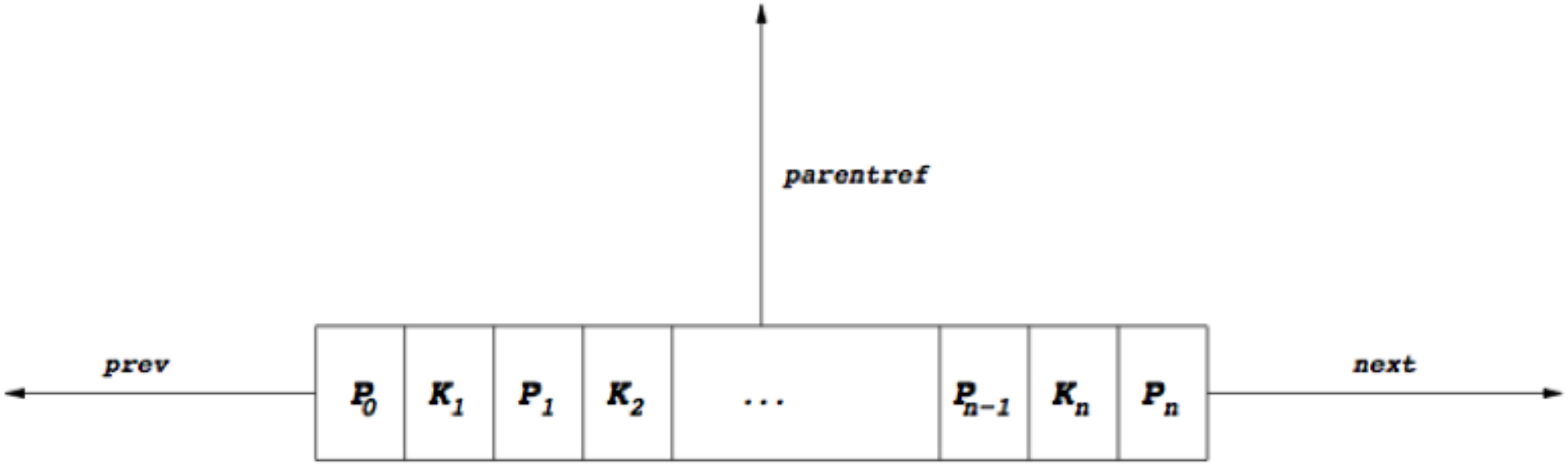
~~n10, i1, i10, i20, i30, i40, i50, i60, i70, i80, i90, i100, i110, i120, i130, i140, i150, i160, i170, d50, d100, d170, i100, i170,d1, i10, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14, i15, i16, i17, i18, i19, i20, i21, i22, i23, i24, i25, i26, i27, i28, i29, i30, i31, i32, i33, i34, i35, i36, i37, i38, i39, i40, i41, i42, i43, i44, i45, i46, i47, d110, d21,~~ **d22, d23, d41, d24, d170**, p, o

The rest were just simple deletions (no underfull leaf nodes)

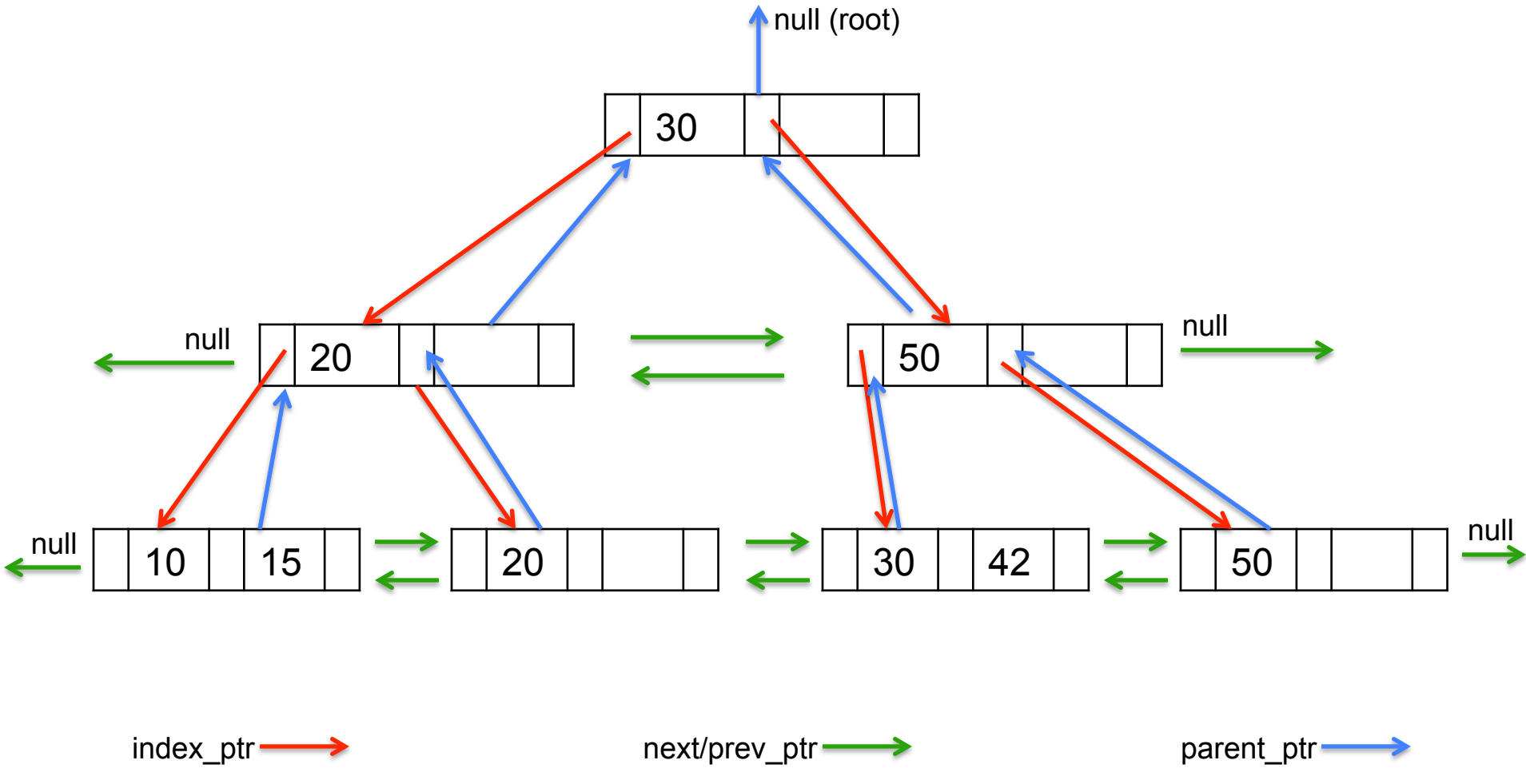


The Final Tree (data2_result.pdf)

The Node



Some nodes with Pointers (degree 3)



Code Overview & Demo