Finder

CS 293 Final Project Demo

Dhanesh Kumar,110050021
Himanshu Roy,110050019

Outline <for a total of 30 mins>

- Aim of project (1 mins)
- Demo (5 mins)
- Teamwork Details (0.5 min)
- Design Details –Algorithm (5 mins)
- Design Details – Implementation (8 mins)
- Viva (9 mins)
- Transition time to next team (2 mins)

Aim of the project

- The project is a sort of desktop-search with features like auto-completion, based on history of search, and normal prefix-search.
- 10 files will be shown at a time and more files can be viewed using the next feature.
- Also the files and applications can be launched, in case of files appropriate apps are used.

Demo

Teamwork Details

- The main components of work were
  - Developing the Basic Class Structure
  - Building the GUI
  - Implementing Data-structures for indexing of files.
  - Integrating the GUI and Algorithmic part.

Dhanesh Kumar	Himanshu Roy
Developing the Basic Class Structure(60%)	Developing the Basic Class Structure(40%)
Implementing the GUI in QT(25%)	Implementing the GUI in QT(75%)
Implementing B-tree and Tries structure(70%)	Implementing B-tree and Tries structure(30%)
Roughly 60% to the complete Project	**Roughly 40%** to the complete Project

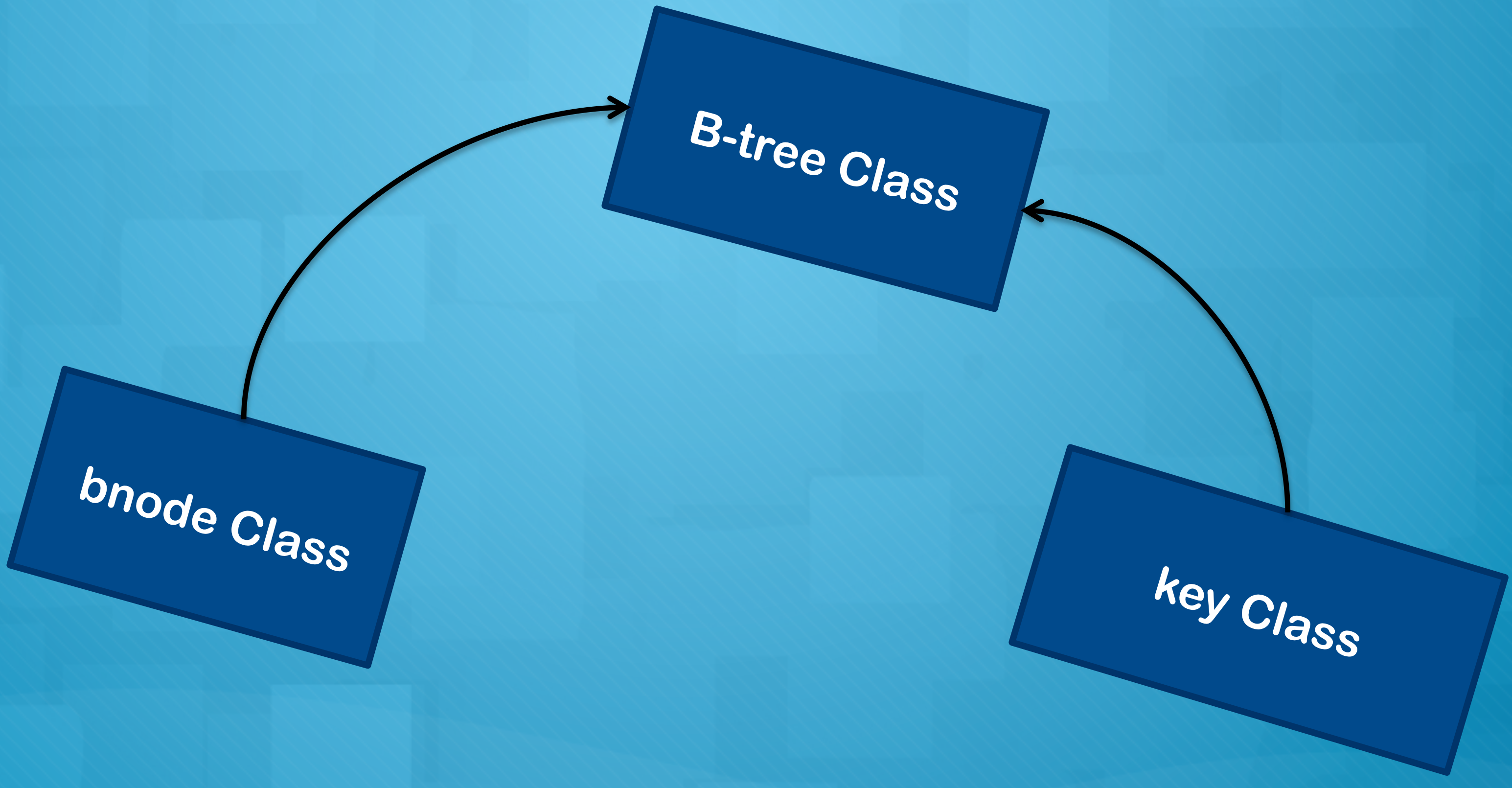
Design Details

- Algorithm
  - B-Tree and Trie as the main indexing data-structure for the database.
  - Prefix-Trie is used as the history maintaining data-structure.
- Implementation
  - Graphics Library : QtGui, dirent.h, unistd.h
  - Language : C++

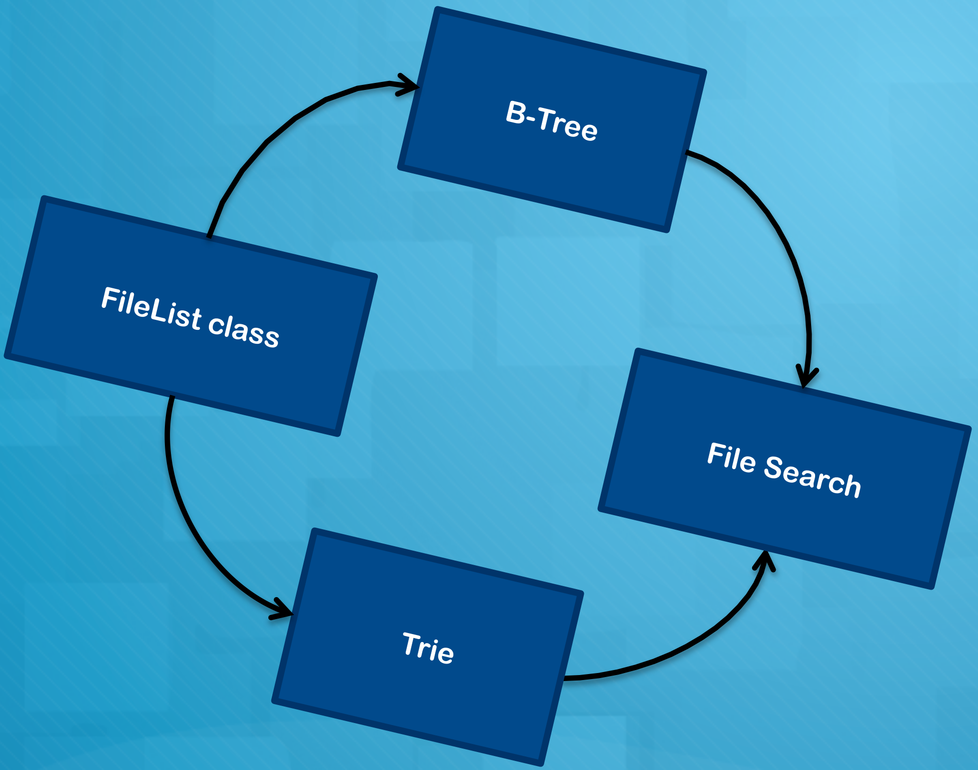
Algorithm Design

- Implementing the main indexing Data-strucutre with insertion(O($\log_k$n)) where ‘k’ is the maximum number children of the B-Tree.
- Algorithm used : B-Tree
- History maintaining by Data-structure with insertion{O(d*m)} and deletion{O(d*m)} where d:no. of childs, m:depth
- Algorithm used : Trie
- Other Algorithms used : Insertion Sort , Breadth first Search , Binary Search

Class Design



Class Design – High level



We have the Filelist class which collects the list of all the files on the system. This FileList is then used for indexing and maintained in data-structures such as tries and b-tree, which in turn are used in file-search.

Class Design – High level

- *Also in GUI part we are providing the file as a button with its name and address.*
- *In addition any file can be opened by clicking on it.*
- *Apart from that 10-files are shown at a time, more can be viewed using next and back buttons.*

Class Design - Details

Class Name	Brief Description
fileList	This class extracts all the files from system and puts in a single (input)file.
Tries	This class implements basic functions like insert, delete search for trie data structure.
Btree	**This class implements basic functions like insert , search for Btree data structure.**
Key	**This class is supporting class of B-tree. It keeps keylist of a node.**
bnode	**This is also supporting class of B-tree. It keeps all the nodes of b-tree.**

Data Structures Used

`Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Prefix Search	**Trie**	**Own**
Full String Search	**B-tree**	**Own**
Auto completion	**Trie**	**Own**
For implementation of trie and B-tree	**list**	**STL**
For implementation of Trie and B-tree	**vector**	**STL**

File Extraction

Accessing all files of system and make indexes :

Using dirent.h and unistd.h library, we make index of all files, installed apps and store them in index.txt in (name, address) format.

Full Key-Word Search :

Complete file-name is given :

- Using Btree and Trie data-structure, we are maintaining indexes of all the files .
- Using it's functions like insert, delete, search we can maintain the database.

Prefix Search :

- Predict/AutoComplete the file name
- Using Trie data structure to maintain history and file index
- Maintaining counter of searched files in history and predict filename according to that

Source Code Information

File Name	Brief Description	Author (Team Member)
fileList.h	**Using libraries dirent.h and unistd.h access all files of system and write it to index.txt**	**Dhanesh Kumar**
FileList.cpp	**Defines member functions of fileList class**	**Dhanesh Kumar**
Tries.h	**Implementation of tries class and it's member function**	**Dhanesh Kumar**

Source Code Information

File Name	Brief Description	Author (Team Member)
Tries.cpp	**Defines member functions of tries class**	**Dhanesh Kumar**
B-tree.h	**Implements B-tree class and define it's member functions**	**Both**
B-tree.cpp	**Define member functions and declare member variables of class B-tree**	**Both**

Source Code Information

File Name	Brief Description	Author (Team Member)
Bnode.h & *Bnode.cpp*	**Helper class of B-tree Keep all nodes of that tree**	**Both**
Key.h *&* *Key.cpp*	**Supporting class of B-tree , keep key List of B-tree.**	**Both**
Random.h *&* *Random.cpp*	**Extra helping functions used in integration of all files**	**Both**

Source Code Information

File Name	Brief Description	Author (Team Member)
System.h	**The main simulation system class , the heart of the simulation manages most of the algorithmic part.**	**Both**
System.cpp	**Implementation of the System class**	**Both**
main.cpp	**Main file contains some GUI and main function**	**Both**

Source Code Information

File Name	Brief Description	Author (Team Member)
piechartqt.h	**Main GUI header file**	**Himanshu Roy**
piechartqt.cpp	**Main GUI cpp file**	**Himanshu Roy**
mybutton.h	**Custom button class header**	**Both**
mybutton.cpp	**Custom button class cpp file**	**Both**

Bugs

- We are not able to open some of the apps and also some files do not open with their corresponding applicaions.

Brief Conclusion

- We would like to conclude that we successfully completed what we started on , file accessing, making indexes ,file search(prefix search, full-string-match search, auto-completion)

Brief Conclusion

- We also spent a great deal of time in the substring search algorithms.
- We also implementedcertain algorithms that we learned during the course such as quick sort for efficiently detecting collisions , depth first search for efficiently calculating the force on each particle.

References :

- http://www.dreamincode.net/forums/topic/59943-accessing-directories-in-cc-part-i/
- www.wikipedia.org
- www.google.com
- http://qt-project.org/doc/qt-4.8/

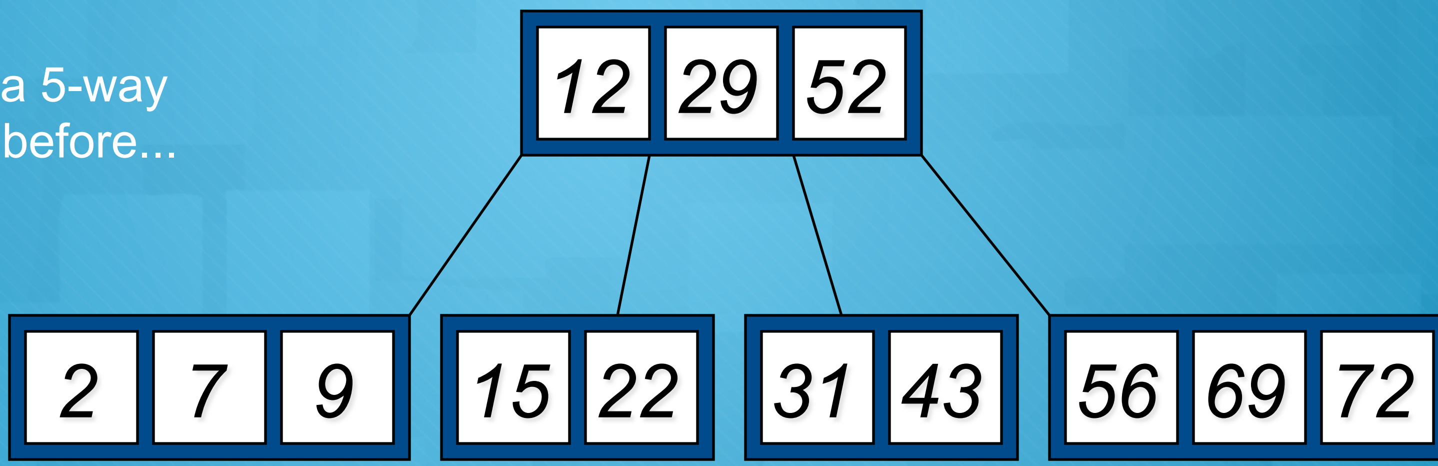
Thank You – Questions?

Backup Slides

- Sample implementation of B-Tree and Trie data-strucutre.
- Reference B-tree: www.cs.uga.edu/~eileen/2720/Notes/Btrees.ppt

Type #1: Simple leaf deletion

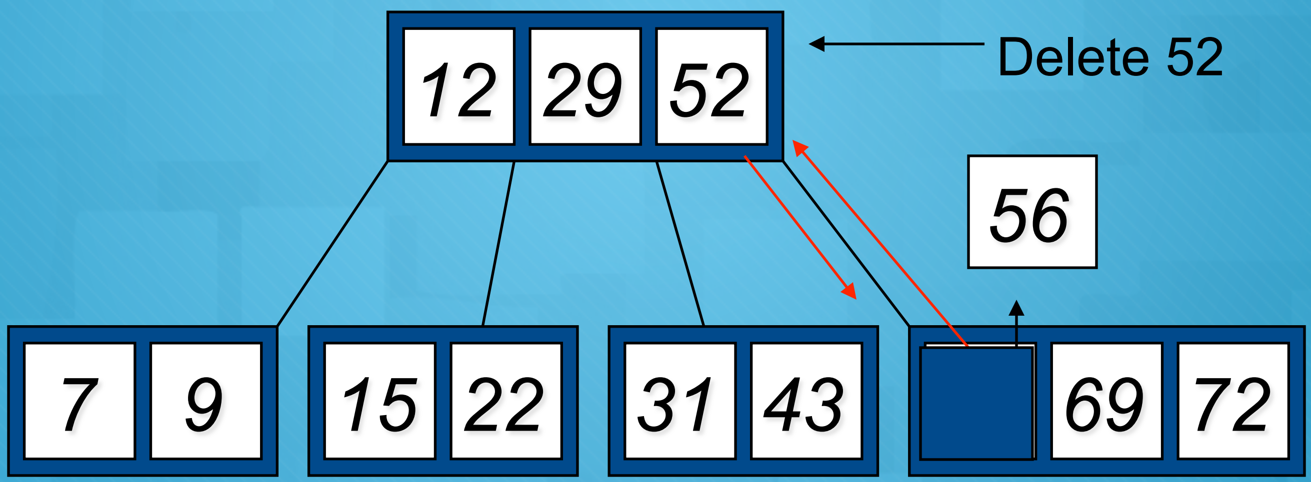
Assuming a 5-way B-Tree, as before...



Delete 2: Since there are enough keys in the node, just delete it

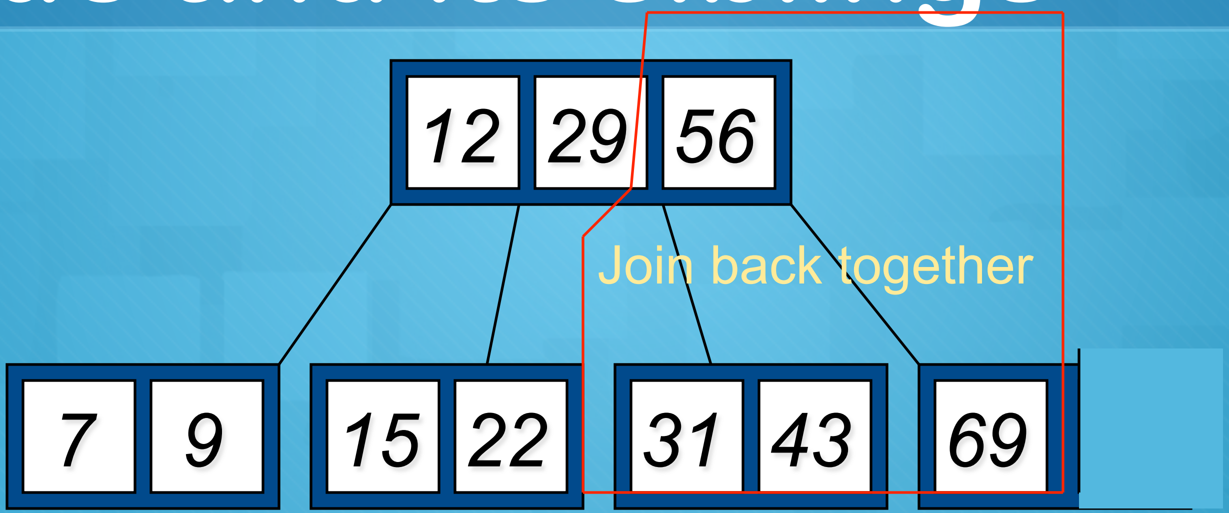
Note when printed: this slide is animated

Type #2: Simple non-leaf deletion



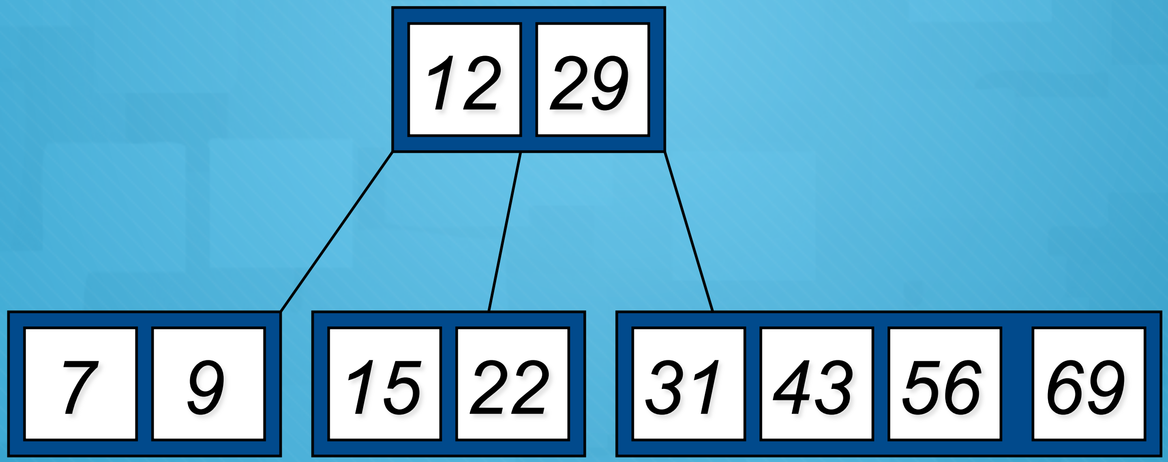
Borrow the predecessor
or (in this case) successor

Type #4: Too few keys in node and its siblings

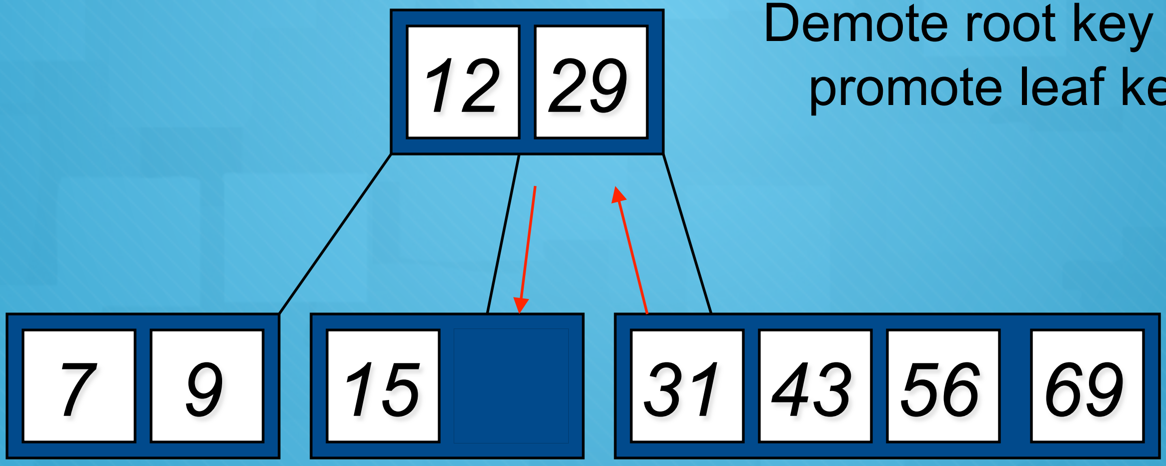


Delete 72

Type #4: Too few keys in node and its siblings



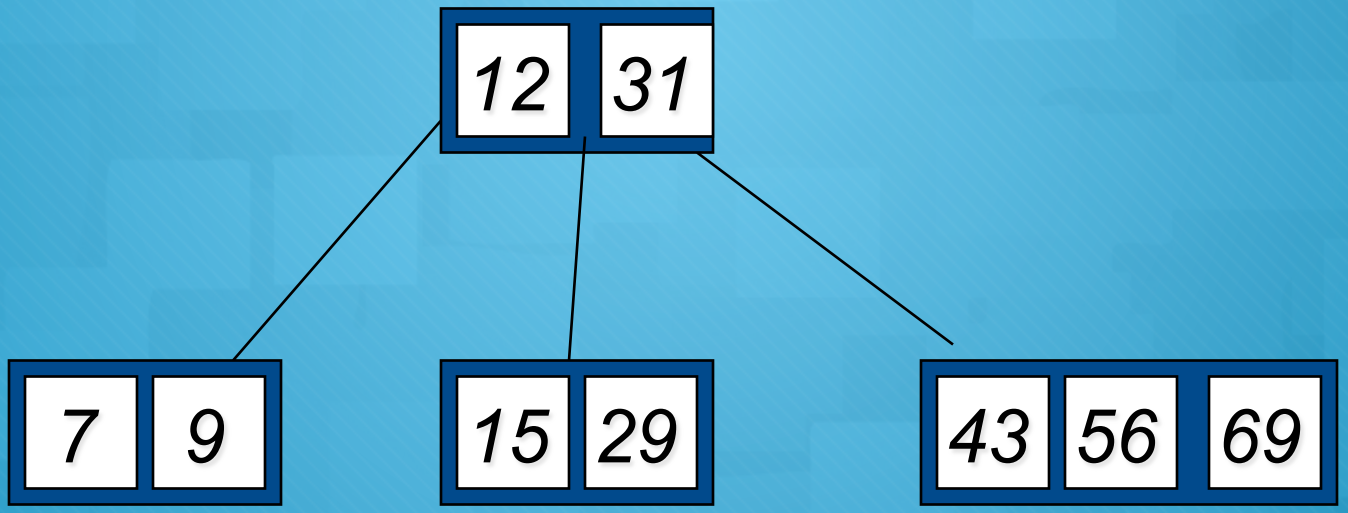
Type #3: Enough siblings



Demote root key and promote leaf key

Delete 22

Type #3: Enough siblings



Note when printed: this slide is animated

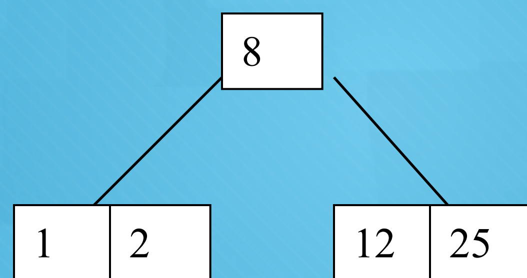
Constructing a B-tree

- Suppose we start with an empty B-tree and keys arrive in the following order: 1 12 8 2 25 5 14 28 17 7 52 16 48 68 3 26 29 53 55 45
- We want to construct a B-tree of order 5
- The first four items go into the root:

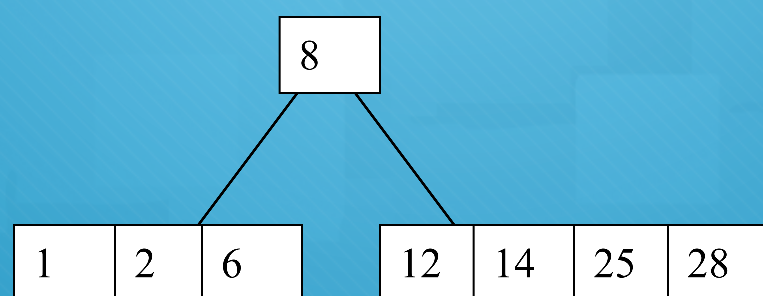
1	2	8	12

- To put the fifth item in the root would violate condition 5
- Therefore, when 25 arrives, pick the middle key to make a new root

Constructing a B-tree (contd.)

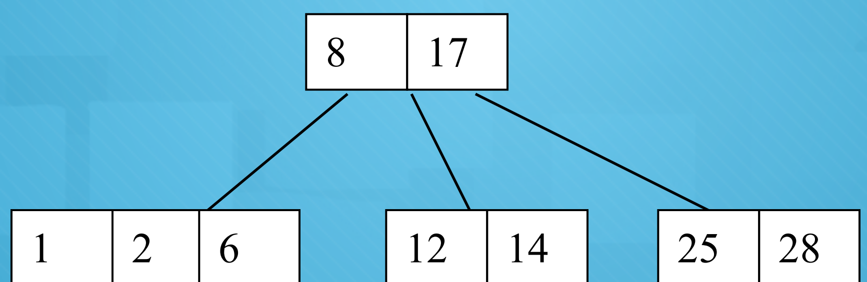


6, 14, 28 get added to the leaf nodes:

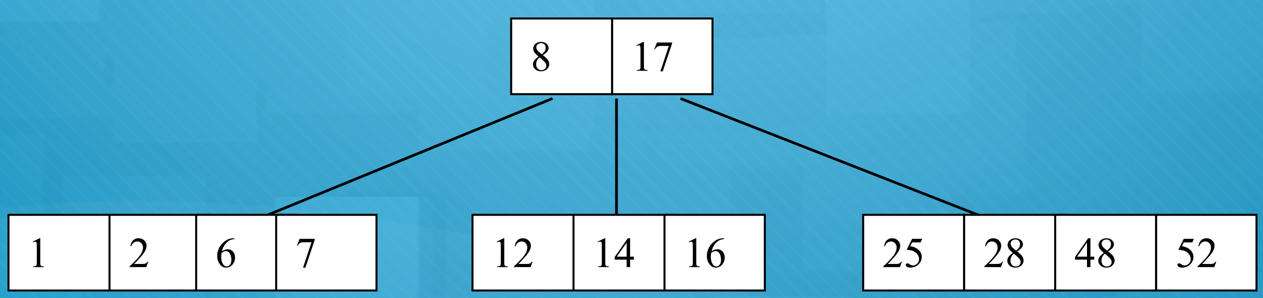


Constructing a B-tree (contd.)

Adding 17 to the right leaf node would over-fill it, so we take the middle key, promote it (to the root) and split the leaf

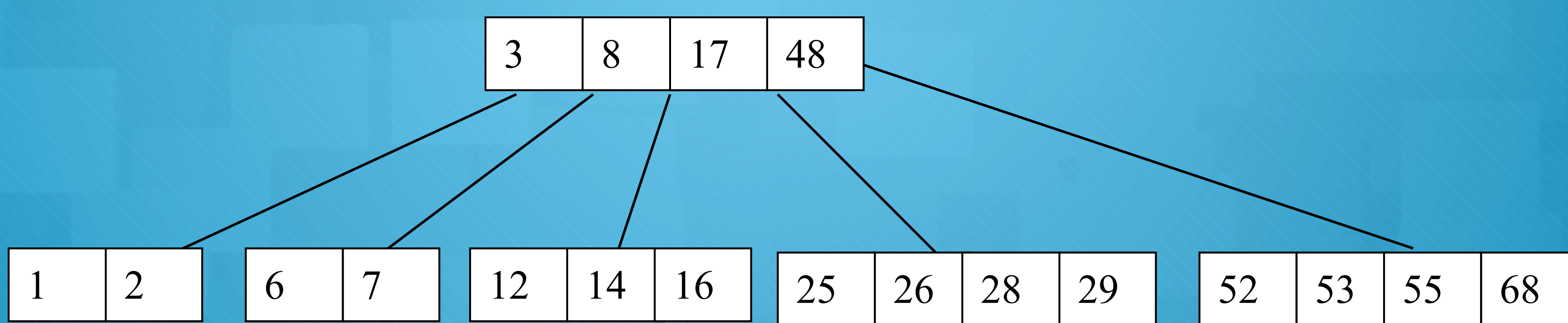


7, 52, 16, 48 get added to the leaf nodes



Constructing a B-tree (contd.)

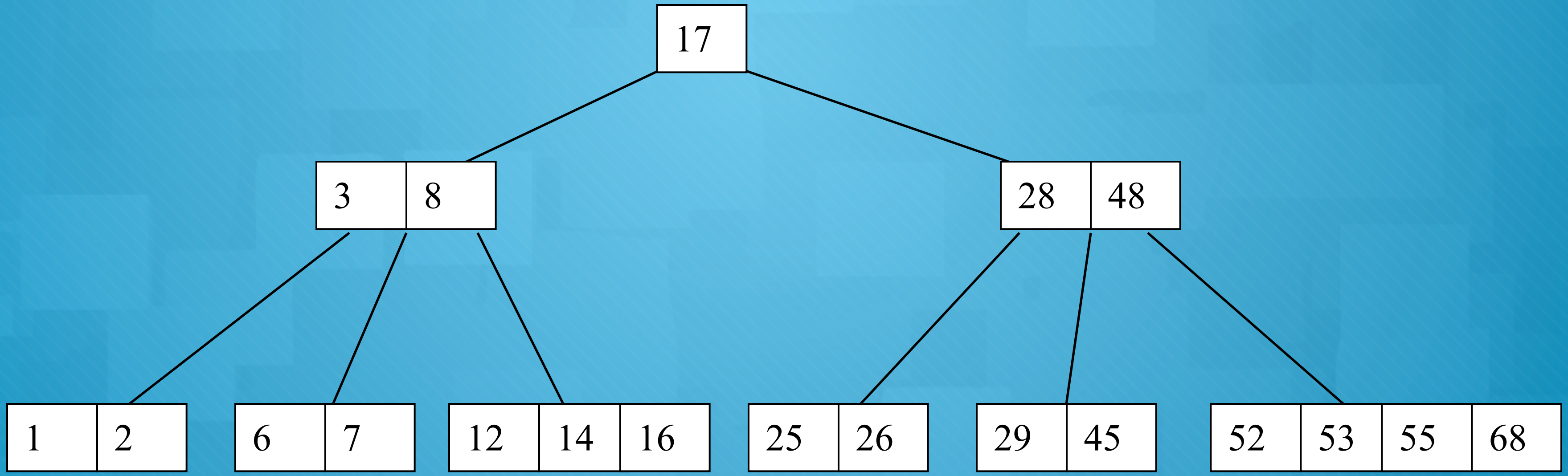
Adding 68 causes us to split the right most leaf, promoting 48 to the root, and adding 3 causes us to split the left most leaf, promoting 3 to the root; 26, 29, 53, 55 then go into the leaves



Adding 45 causes a split of | 25 | 26 | 28 | 29 |

and promoting 28 to the root then causes the root to split

Constructing a B-tree (contd.)



Inserting into a B-Tree

- Attempt to insert the new key into a leaf
- If this would result in that leaf becoming too big, split the leaf into two, promoting the middle key to the leaf's parent
- If this would result in the parent becoming too big, split the parent into two, promoting the middle key
- This strategy might have to be repeated all the way to the top
- If necessary, the root is split in two and the middle key is promoted to a new root, making the tree one level higher

Tries: Example