HW #5 B+ Tree

- You will implement a B+ tree index in C / C++ / Java / Python, or **other** programming language if you prefer.
- At the end of the project, you will have a specific interface, used by various **command** tools. The tools will let you create and manipulate persistent B+ tree indexes stored in virtual disks and accessed through manipulating disk blocks or pages.
- You can assume that requests to the B+ tree are serialized, meaning that you can finish a request before starting the next one. In a real database system, however, locking and logging are used to allow multiple requests to simultaneously execute on the tree.

HW #5 (2)

- You can assume that keys in the B+ tree are of fixed size and given when the B+ tree is initialized. In a real database system, however, keys can be of variable size.

- You will be implementing a "**pure**" B+ tree. Usually in a B+ tree, leaf nodes hold keys and values, however, in this project, leaf nodes hold **keys** only, while interior nodes hold **keys** and **block pointers**. You can also optionally chain the B+ tree leaf nodes together into a linked list, making range queries much faster.

HW #5 (3)

- You will be evaluated using a test that will evaluate its correctness. The test will generate a stream of requests, run them through your implementation, compare the outputs.
- At a high-level of abstraction, a B+ tree is a mapping from keys to values. B+ trees can require that all keys be unique, but it is not necessary – there is a distinction between a key in a B+ tree and a key in relational database terminology. In B+ tree, it is perfectly OK to create an index on some attribute or set of attributes that form **neither** a key **or** superkey. Your implementation can assume that all keys are unique though.

HW #5 (4)

- A B+ tree implementation must perform the following operations:
- **Initialize:** create a new B+ tree structure on the virtual disk.
- **Attach:** open an existing B+ tree for use. You can ask for the B+ tree to be attached **without** initialization. B+ trees are inputted using a way of semicolon-delimited, inorder-like traversal. (see example for details)
- **Bulkload:** bulk load the B+ tree with space-delimited keys.
- **Lookup (key):** returns true if such key exists; otherwise, returns false.

HW #5 (5)

- **Insert (key):** your insert routine must be capable of dealing with overflows (at any level of the tree) by splitting pages; you will **not** consider re-distribution. If you really feel ambitious, you can add support for this for **extra credit**.

- **Delete (key):** you can deal with deletes by simply marking the corresponding leaf entry as `deleted'; you do **not** have to implement merging. (Do **extra credit** if you have time!)

HW #5 (6)

- In addition, your B+ Tree implementation will also support the following operation:

- **Display:** you can graphically display trees, or do an inorder-like traversal of the B+ tree, printing out the keys in (approximately) ascending order of the keys.

HW #5 (7)

- Moreover, you should provide **sanity**理智 **check** whenever possible, i.e., do a self-check of the tree operation for problems, for example,

1. when deleting keys in an empty tree, or
2. when inserting keys without initialization, or
3. after an insert or a delete, the resultant tree is no longer a **legal** B+ tree.

- A sample test-run request stream is provided. Note this is not a comprehensive set of tests. It is just to get you started, illustrating some sample operations.

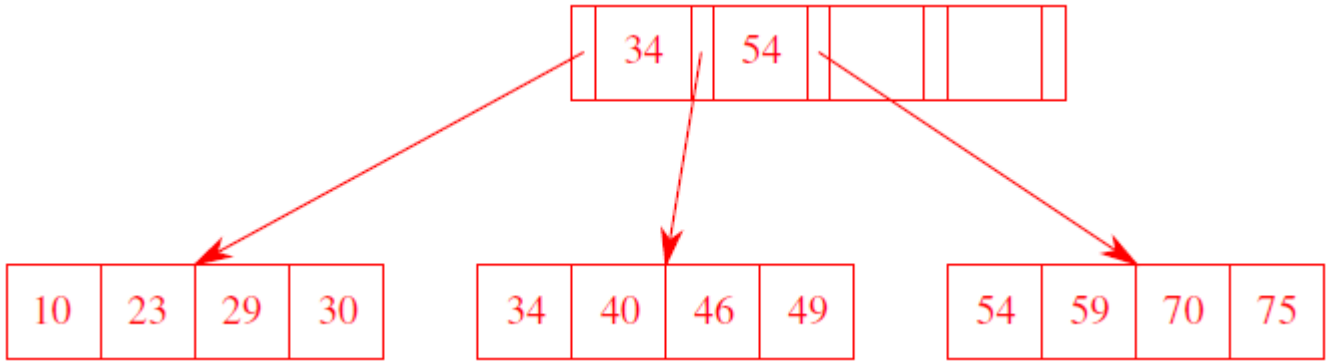
HW #5 (8)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 1 Enter
Initializing… order = 2 Enter // nodes contain up to 4 keys (5 pointers)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 3 Enter
Bulkloading… key sequence = 46 10 70 49 23 40 59 29 34 54 75 30 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

HW #5 (9)



也可以利用inorder-like的方式印出keys

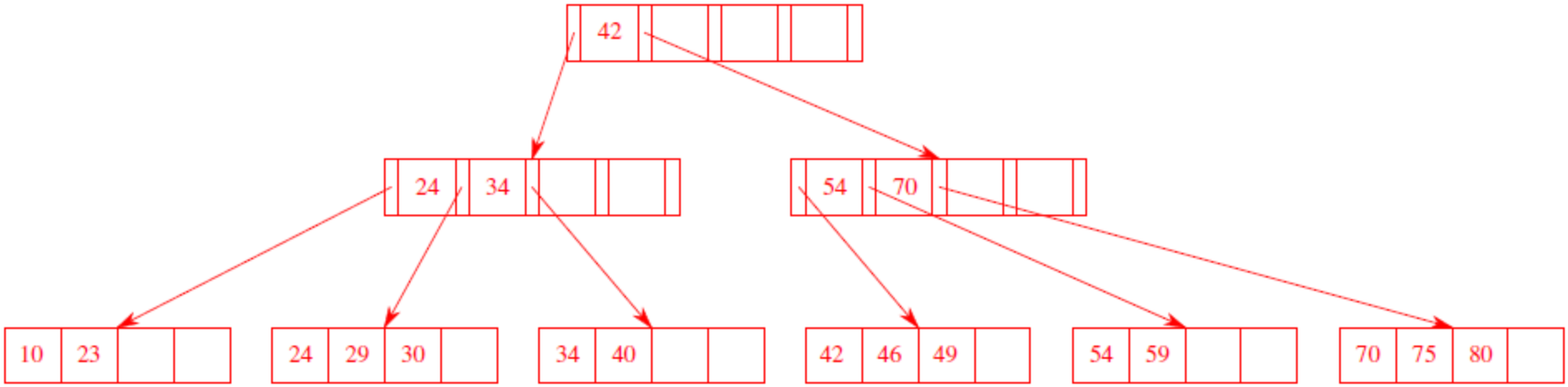
1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting… key = 80 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting:… key = 24 Enter

HW #5 (10)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting… key = 42 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter



10 23 ; 24 ; 24 29 30 ; 34 ; 34 40 ; 42 ; 42 46 49 ; 54 ; 54 59 ; 70 ; 70 75 80

HW #5 (11)

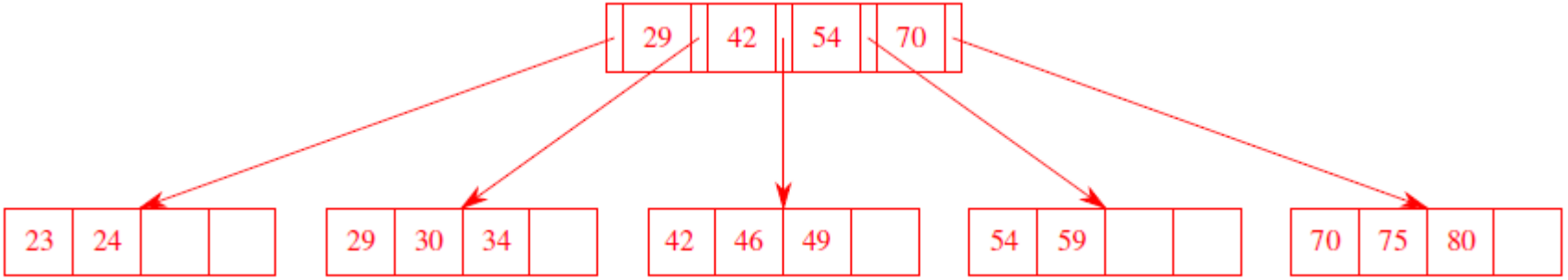
1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 6 Enter
Deleting… key = 10 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 6 Enter
Deleting… key = 40 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

HW #5 (12)

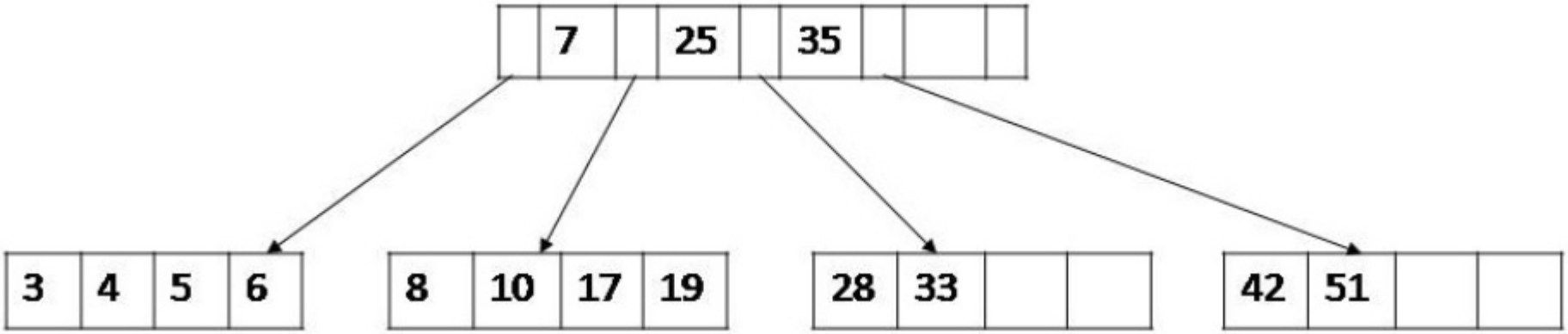
這裡**所有的**示意圖, **不一定**就是答案
課堂上所學的演算法, 通常都無法產生這麼精簡的B+ trees



1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 2 Enter
Attaching… order = 2 Enter
Nodes in inorder-like traversal = 3 4 5 6 ; 7 ; 8 10 17 19 ; 25 ; 28 33 ; 35 ; 42 51 Enter

HW #5 (13)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter



HW #5 (14)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting… key = 34 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting… key = 2 Enter

HW #5 (15)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 5 Enter
Inserting… key = 15 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

HW #5 (16)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 6 Enter
Deleting… key = 28 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 6 Enter
Deleting… key = 8 Enter

HW #5 (17)

1) Initialize 2) Attach 3) Bulkload 4) Lookup
5) Insert 6) Delete 7) Display 8) Quit
> Select an operation: 7 Enter

- Extra credit: The main motivation for trying these additional challenges should be the opportunity to write more complete software and understand some of the finer points.

1. Support **duplicate** records by allowing records with the same key to exist on more than one page. (You should **not** use any overflow pages!)
2. Implement node redistribution and merging during **deletion**.