Lecture 13:
2-4 Trees

Pure Consciousness is the field of perfect order, balance, and efficiency

1

1

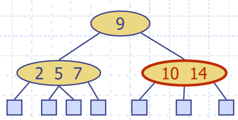
Wholeness Statement

A Map data structure allows users to assign keys to elements then to access or remove those elements by key. An ordered Map maintains an order relation among keys allowing access to adjacent keys in sorted order while supporting efficient implementation. *Science of Consciousness:* Each of us has access to the source of thought which is a field of perfect order, balance, and efficiency; contact and experience of this field brings those qualities into our mind and physiology for benefit in daily life.

2

2

(2,4) Trees



3

3

Outline and Reading

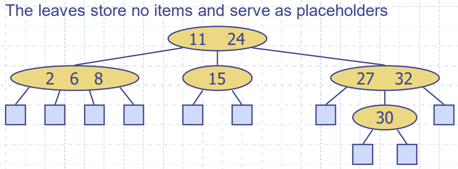
- Multi-way search tree
  - Definition
  - Search
- (2,4) tree
  - Definition
  - Search
  - Insertion
  - Deletion
- Comparison of dictionary implementations

4

4

Multi-Way Search Tree

- A multi-way search tree is an ordered tree such that
  - Each internal node has at least two children and stores $d-1$ key-element items (k_i, o_i), where d is the number of children
  - For a node with children $v_1\ v_2 \ldots v_d$ storing keys $k_1\ k_2 \ldots k_{d-1}$
    - keys in the subtree of v_1 are less than k_1
    - keys in the subtree of v_i are between k_{i-1} and k_i $(i = 2, \ldots, d-1)$
    - keys in the subtree of v_d are greater than k_{d-1}
  - The leaves store no items and serve as placeholders

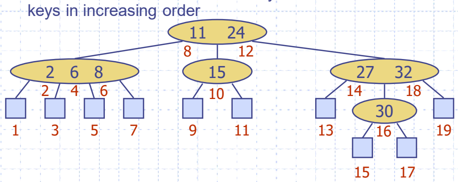


5

5

Multi-Way Inorder Traversal

- We can extend the notion of inorder traversal from binary trees to multi-way search trees
- Namely, we visit item (k_i, o_i) of node v between the recursive traversals of the subtrees of v rooted at children v_i and v_{i+1}
- An inorder traversal of a multi-way search tree visits the keys in increasing order

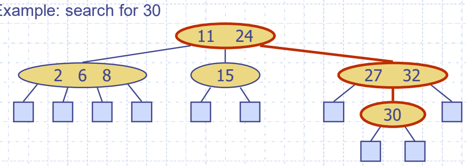


6

6

Multi-Way Searching

- Similar to search in a binary search tree
- A each internal node with children $v_1\ v_2\ \ldots\ v_d$ and keys $k_1\ k_2\ \ldots\ k_{d-1}$
  - $k = k_i\ (i = 1, \ldots, d-1)$: the search terminates successfully
  - $k < k_1$: we continue the search in child v_1
  - $k_{i-1} < k < k_i\ (i = 2, \ldots, d-1)$: we continue the search in child v_i
  - $k > k_{d-1}$: we continue the search in child v_d
- Reaching an external node terminates the search unsuccessfully
- Example: search for 30



7

7

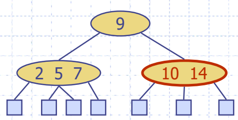
B-Trees

- A B-Tree is a balanced multi-way search tree, i.e., all leaves are at the same depth
- B-Trees are used to implement a file structure that allows random access by key as well as sequential access of keys in sorted order
- The size of a node in a B-Tree file structure is the size of a sector of a track of a disk file (also called a physical block)

8

8

(2,4) Tree



9

9

Why (2-4) Trees?

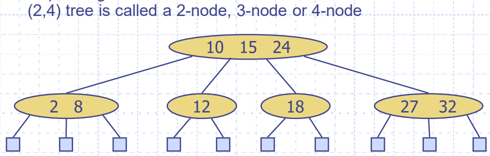
- A Red-Black tree is an implementation of a (2-4) Tree in a binary tree data structure
- If you understand the (2-4) Tree implementation, you will more easily understand what is done and why in a Red-Black Tree to keep it balanced
- You will appreciate that it's easier and more efficient in space and time to implement a Red-Black Tree than a (2-4) Tree

10

10

(2,4) Tree

- A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search tree with the following properties
  - Node-Size Property: every internal node has at most four children
  - Depth Property: all the external nodes have the same depth
- Depending on the number of children, an internal node of a (2,4) tree is called a 2-node, 3-node or 4-node



11

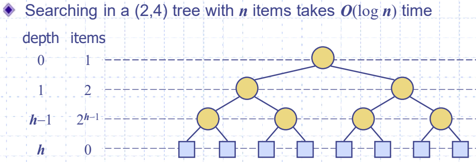
11

Height of a (2,4) Tree

- Theorem: A (2,4) tree storing n items has height $O(\log n)$

  Proof:
  - Let h be the height of a (2,4) tree with n items
  - Since there are at least 2^i items at depth $i = 0, \ldots, h-1$ and no items at depth h, we have

    $$n \geq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$$
  - Thus, $h \leq \log(n+1)$
- Searching in a (2,4) tree with n items takes $O(\log n)$ time

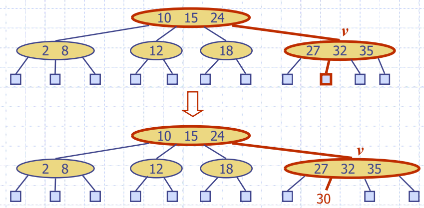


12

12

Insertion

- We insert a new item (k, o) at the parent v of the leaf reached by searching for k
  - We preserve the depth property but
  - We may cause an overflow (i.e., node v may become a 5-node)
- Example: inserting key 30 causes an overflow

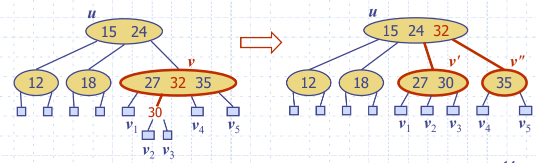


13

13

Overflow and Split

- We handle an overflow at a 4-node v with a split operation:
  - let $v_1 \ldots v_5$ be the children of v and $k_1 \ldots k_4$ be the keys of v
  - node v is replaced by nodes v' and v''
    - v' is a 3-node with keys k_1 k_2 and children v_1 v_2 v_3
    - v'' is a 2-node with key k_4 and children v_4 v_5
  - the middle key 32 of node v is inserted into the parent u of v (a new root may be created); the new key 30 is inserted into either v' or v''
- The overflow may propagate to the parent node u



14

14

Analysis of Insertion

Algorithm *insertItem*(k, o)

1. We search for key k to locate the insertion node v
2. We add the new item (k, o) at node v
3. **while** *overflow*(v)
   if *isRoot*(v)
   create a new empty root above v
   $v \leftarrow$ *split*(v)

- Let T be a (2,4) tree with n items
  - Tree T has $O(\log n)$ height
  - Step 1 takes $O(\log n)$ time because we visit $O(\log n)$ nodes
  - Step 2 takes $O(1)$ time
  - Step 3 takes $O(\log n)$ time because each split takes $O(1)$ time and we perform $O(\log n)$ splits
- Thus, an insertion in a (2,4) tree takes $O(\log n)$ time

15

15

Example:

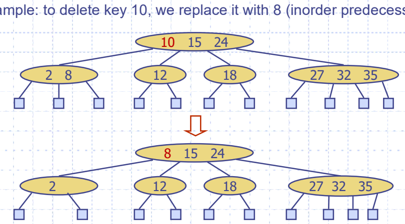
- Insert the following into an initially empty 2-4 tree in this order:
  (16, 5, 22, 45, 2, 10, 18, 30, 50, 12, 1)

16

16

Deletion

- We reduce deletion of an item to the case where the item is at the node with leaf children
- Otherwise, we replace the item with its inorder predecessor (or, equivalently, with its inorder successor) and delete the latter item
- Example: to delete key 10, we replace it with 8 (inorder predecessor)

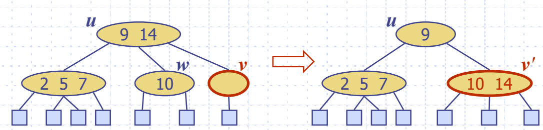


17

17

Underflow and Fusion

- Deleting an item from a node v may cause an underflow, where node v becomes a 1-node with one child and no keys
- To handle an underflow at node v with parent u, we consider two cases
- Case 1: the adjacent siblings of v are 2-nodes
  - Fusion operation: we merge v with an adjacent sibling w and move an item from u to the merged node v'
  - After a fusion, the underflow may propagate to the parent u

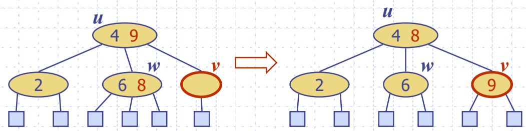


18

18

Underflow and Transfer

- To handle an underflow at node v with parent u, we also consider a second case
- Case 2: an adjacent sibling w of v is a 3-node or a 4-node
  - Transfer operation:
    1. we move an item from u to v
    2. we move an item from w to u
  - After a transfer, no underflow occurs



19

19

Analysis of Deletion

Algorithm *deleteItem*(k)

1. We search for key k and locate the deletion node v

2. **while** *underflow*(v) **do**
   - **if** *isRoot*(v)
     - change the root to child of v; return
   - **if** *a sibling*(v) = u is a 3- or 4-node
     - transfer(u, v); return
   - **else** {both siblings are 2-nodes}
     - *fusion*(u, v) {*merge* v *with sibling* u}
     - $v \leftarrow$ *parent*(v)

- Let T be a (2,4) tree with n items
  - Tree T has $O(\log n)$ height
  - Step 1 takes $O(\log n)$ time because we visit $O(\log n)$ nodes
  - Step 2 takes $O(\log n)$ time because each fusion takes $O(1)$ time and we perform $O(\log n)$ fusions
- Thus, a deletion in a (2,4) tree takes $O(\log n)$ time

20

20

Analysis of Deletion

- Let T be a (2,4) tree with n items
  - Tree T has $O(\log n)$ height
- In a deletion operation
  - We visit $O(\log n)$ nodes to locate the node from which to delete the item
  - We handle an underflow with a series of $O(\log n)$ fusions, followed by at most one transfer
  - Each fusion and transfer takes $O(1)$ time
- Thus, deleting an item from a (2,4) tree takes $O(\log n)$ time

21

21

Main Point

1. By introducing some flexibility in the data content of each node, all leaf nodes of a (2,4) Tree can be kept at the same depth, i.e., flexible content is the basis of stable leaf depth, tree height, and balance.
   Science of Consciousness: Stability and adaptability are fundamentals of progress and evolution in nature. These qualities grow in our lives through regular contact with pure consciousness.

22

22

Selection

Prune-and-Search
or
Decrease-and-Conquer

23

23

The Selection Problem

- Given an integer k and n elements $x_1, x_2, \ldots, x_n$, taken from a total order, find the k-th smallest element in this set.
- Of course, we can sort the set in O(n log n) time and then index the k-th element.

k=3   7 4 9 6 2 → 2 4 6 7 9

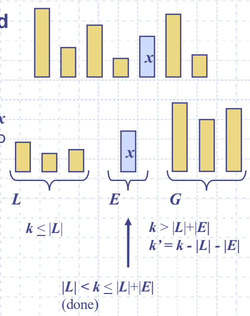
- Can we solve the selection problem faster?

24

24

Quick-Select

- Quick-select is a **randomized** selection algorithm based on the prune-and-search paradigm:
  - Prune: pick a random element x (called pivot) and partition S into
    - L elements less than x
    - E elements equal x
    - G elements greater than x
  - Search: depending on k, either answer is in E, or we need to recurse in either L or G

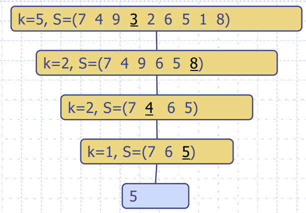


25

25

Quick-Select Visualization

- An execution of quick-select can be visualized by a recursion path
  - Each node represents a recursive call of quick-select, and stores k and the remaining sequence



26

26

Quick Select

```
Algorithm QuickSelect(S, lo, hi , k )
    Input Unsorted ArrayList S and k
    Output the k-th smallest element in S

    p ← inPlacePartition(S, lo, hi)
    j ← p – lo + 1
    if j = k then
          return S.get(p)
    else if j > k then
          return QuickSelect(S, lo, p-1, k)
    else
          return QuickSelect(S, p+1, hi, k-j)
```

27

27

In Place Version of Partition

```
Algorithm inPlacePartition(S, lo, hi)
    Input ArrayList S and ranks lo and hi, 0 ≤ lo ≤ hi < S.size()
    Output the pivot is now stored at its sorted rank

    p ← a random integer between lo and hi
    swapElements(S, lo, p)
    pivot ← S.get(lo)
    j ← lo + 1
    k ← hi
    while j ≤ k do
        while k ≥ j ∧ S.get( k ) ≥ pivot do
            k ← k – 1
        while j ≤ k ∧ S.get( j ) < pivot do
            j ← j + 1

        if j < k then
            swapElements(S, j, k)
            j ← j + 1
            k ← k – 1

    swapElements(S, lo, k)   {move pivot to sorted rank}
    return k
```

Quick Select 28

28

- What is the time complexity of Partition?
- What is the loop invariant of the outermost loop?

29

29

The loop invariant of the outer-most loop of inPlacePartition

forall i; lo+1 ≤ i < j; S.get(i) ≤ pivot
- The values in S at ranks between lo+1 and j are less than the pivot

/\

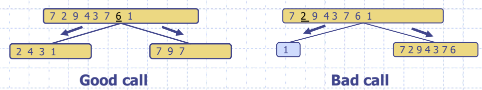
forall i; k < i ≤ hi; S.get(i) ≥ pivot
- The values in S at ranks between k and hi are greater or equal to the pivot

30

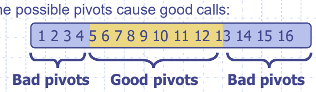
30

Expected Running Time

- Consider a recursive call of quick-select on a sequence of size s
  - **Good call:** the sizes of L and G are each less than $3s/4$
  - **Bad call:** one of L and G has size greater than $3s/4$



- A call is good with probability $1/2$
  - 1/2 of the possible pivots cause good calls:



31

31

Expected Running Time, Part 2

- Probabilistic Fact #1: The expected number of coin tosses required in order to get one head is two
- Probabilistic Fact #2: Expectation is a linear function:
  - $E(X + Y) = E(X) + E(Y)$
  - $E(cX) = cE(X)$
- Let T(n) denote the expected running time of quick-select.
- By Fact #2,
  - $T(n) \leq T(3n/4) + (\text{expected \# of calls before a good call}) * bn$
- By Fact #1,
  - $T(n) \leq T(3n/4) + 2bn$
- So T(n) is O(?).

32

32

Conclusion

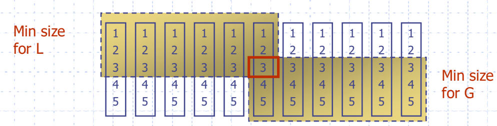
- We can solve the selection problem in O(n) expected time

33

33

Deterministic Selection

- We can do selection in O(n) worst-case time.
- Main idea: recursively use the selection algorithm itself to find a good pivot for quick-select:
  - Divide S into n/5 sets of 5 each
  - Find a median in each set
  - Recursively find the median of the "baby" medians.



34

34

Main Point

2. Prune-and-Search algorithms reduce the search space by some fraction at each step, then the smaller problem is recursively solved.
   Science of Consciousness: The problem of world peace can be reduced to the smaller problem of peace and happiness of the individual. The problem can be further reduced to the much smaller problem of forming a small group (square root of 1%) practicing the TM and TM-Sidhi program together.

35

35

Connecting the Parts of Knowledge with the Wholeness of Knowledge

1. In a (2,4) tree, each node has 2, 3, or 4 children and all leaf nodes are at the same depth so search, insertion, and deletion are efficient, O(log n).
2. The insert and delete operations in a (2,4) tree are carefully structured so that the activity at each node promotes balance in the tree as a whole. Each node contributes to the dynamic balance by giving and receiving keys during the splitting and fusion of nodes.

36

36

3. **Transcendental Consciousness** is the state of perfect balance, the foundation for wholeness of life, the basis for balance in activity.
4. **Impulses within Transcendental Consciousness**: The dynamic natural laws within this unbounded field create and maintain the order and balance in creation.
5. **Wholeness moving within itself** : In Unity Consciousness, one experiences the dynamics of pure consciousness that gives rise to the laws of nature, the order and balance in creation, as nothing other than one's own Self.