CS 340 - Fall 2003
Assignment 4
Due : Thu November 27 (written part), Tue Dec 02 (programming part)

Exercise 1 : B-Trees [1+1+4+3+8=17pts]

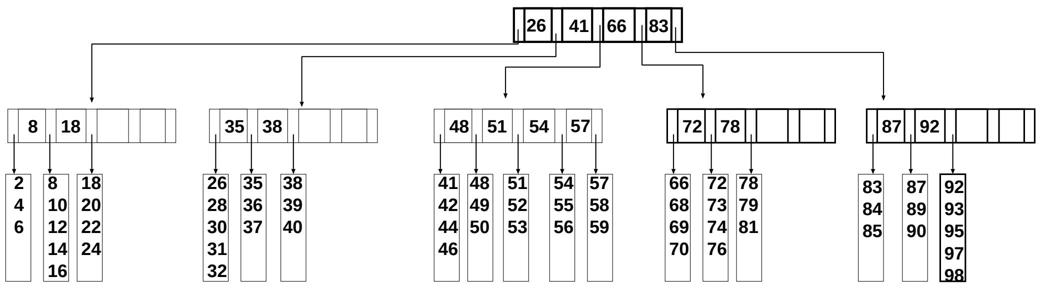

Figure 1: B-Tree.

Consider the B-Tree of figure 1.

1. What are the values of M and L. **solution : M=L=5**
2. What is the time cost (in terms of number of disk access) needed to access any information (data items). **Solution: 3 disk access**
3. Update the B-Tree after performing each of the following operations :
   (a) Inserting 1. **Solution: see figure 2**.
   (b) Inserting 33. **Solution: see figure 3**.
   (c) inserting 17. **Solution: see figure 4**.
   (d) deleting 81. **Solution: see figure 5**.
4. Write the search algorithm that, given a key, looks for the corresponding record in a B-Tree. **Solution: similar to the binary search algorithm but we have to deal with more than one key/node in the case of a B-Tree**.
5. Using a B-Tree, we want to store and process information concerning the driving records for citizens in the state of Florida. We assume the following :
   - we have 300,000,000 items,
   - each key is 8 bytes (ID number),
   - a record is 256 bytes,
   - one disk block holds 8,192 bytes,
   - and a pointer to each block is of size 4 bytes.

   (a) What are the values of M and L in this case ? **Solution: M=684, L=32.**
   (b) What is the number of leaf nodes in the best and worst cases ? **Solution: 9,375,000 leaves in the best case and 18,750,00 in the worst case**
   (c) What is the height of the B-Tree in the worst case ? **Solution: Height = 3 (or 4 if we count the root).**

(d) What is the number of disk access needed to get any information? **Solution: 4 disk access.**

(e) In case we want to store the above items (300,000,000) using an AVL Tree. What is the worst case number of disk access required to access any information. **Solution:** $\log 300,000,000$ **(we can add 1 to be more precise).**

(f) Assuming that records are stored on leaves, write the search algorithm that, given a key, looks for the corresponding record in an AVL-Tree. **Solution: similar to the binary search algorithm**

Exercise 2: *exercise 5.1 page 204* [2x4=8pts]

Exercise 3: *exercise 4.6 page 171* [4pts]

Solution

This can be shown by induction. Alternatively, let N = number of nodes, F = number of full nodes, L = number of leaves, and H = number of half nodes (nodes with one child). Clearly, N = F + H + L. Further, 2F + H = N - 1. Subtracting yields L - F = 1.

Exercise 4: *exercise 4.45 page 175* [6pts]

The function below is O(N) since in the worst case it does a traversal on both t1 and t2.

```
bool similar (Node *t1, Node *t2)
{

    if (t1 == NULL || t2 == NULL)

         return t1 == NULL && t2 == NULL;

    return similar (t1->left, t2->left) && similar (t1->right, t2->right);

}
```

Exercise 5: *exercise 6.13 page 247, questions a and b* [8pts]

Solution

a) If the heap is organized as a (min) heap, then starting at the hole at the root, find a path down to a leaf by taking the minimum child. This requires roughly logN comparisons. To find the correct place where to move the hole, perform a binary search on the logN elements. This takes O(log log N) comparisons.

b) Find a path of minimum children, stopping after log N - log log N levels. At this point, it is easy to determine if the hole should be placed above or below the stopping point. If it goes below, then continue finding the path, but perform the binary search on only the last loglogN elements on the path, for a total of logN + log log log N comparisons. Otherwise, perform a binary search on the first logN - log log N elements. The binary search takes at most log log N comparisons, and the path finding took only log N - log log N, so the total in this case is log N. So the worst case is the first case.

Exercise 6: *exercise 7.41 page 299* [6pts]

Solution

Proof: $log4! = log24 < log32 = log2^5 = 5$ Give an algorithm to sort 4 elements in 5 comparisons.

Method : Compare and exchange (if necessary) a1 and a2 so that a1>=a2, and repeat with a3 and a4. Compare and exchange a1 and a3. Compare and exchange a2 and a4. Finally, compare and exchange a2 and a3.

Exercise 7 : *exercise 7.42 page 299* [6pts]

Solution

- a) $log5! = log120 < log128 = log2^7 = 7$
- b) Compare and exchange (if necessary) a1 and a2 so that a1 >= a2, and repeat with a3 and a4 so that a3 4>= a4. Compare and exchange (if necessary) the two winners, a1 and a3. Assume without loss of generality that we now have a1 >= a3 >= a4, and a1 >= a2 (the other case is obviously identical). Insert a5 by binary search in the appropriate place among a1,a3,a4. this can be done in two comparisons. finally, insert a2 among a3,a4,a5. If it is the largest among those three, then it goes directly after a1 since it is already known to be larger than a1. This takes two more comparisons by a binary searh. The total is thus seven comparisons.

Exercise 8 [3pts]

Determine the running time of mergesort for :

1. sorted input
2. reverse-ordered input
3. random input

Solution: NlogN for the 3 cases

Exercise 9 [6pts]

Suppose that the splits at every level of quicksort are in the proportion 1- a to a, where $0 < a < 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately - log(n) / log(a) and the maximum depth is approximately -log(n)/log(1-a) (don't worry about integer round-off).

Solution

- The shortest branch corresponds to log1/a(N) = (log(N)) / (log(1/a)) = - log(N) / log(a)
- The longest branch corresponds to log1/1-a (N) = (log N) / (log(1/1-a)) = - log(N) / log(1-a)

Exercise 10 [6pts]

The running time of quicksort can be improved in practice by taking advantage of the fast running time of insertion sort when its input is "nearly" sorted. When quicksort is called on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process. Argue that this sorting algorithm runs in O(nk + n log(n/k)) expected time. How should k be picked, both in theory and in practice ?

Solution

- T(N) = running time of quicksort + running time of insertion sort (for the remaing N/K arrays of size K(or less than K))
- $T(N) = N(logN - logK) + N/K(K^2) = Nlog(N/K) + NK$
- In theory, K has to be chosen such that : N K + N log (N/K) = N log N ==¿ K = log N (K can't be more than log N otherwise the total running time will be more than N log N).
- In practice, K should be the largest list length on which insertion sort is faster than quicksort.

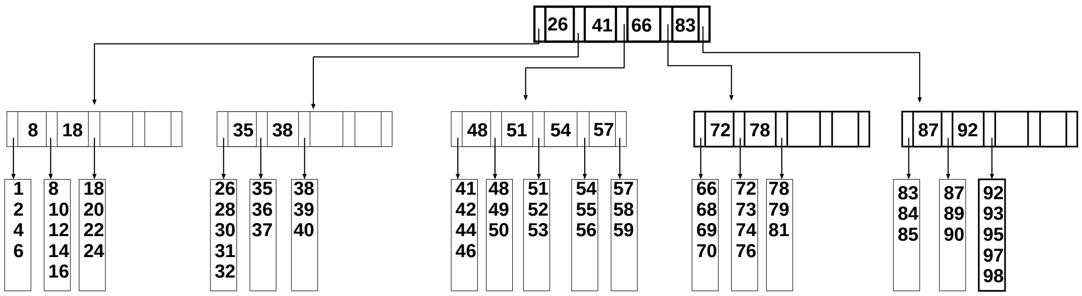

Figure 2: Inserting 1.

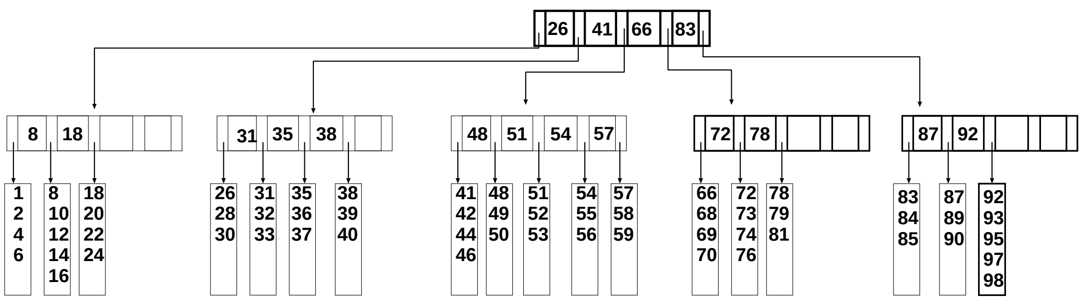

Figure 3: Inserting 33.

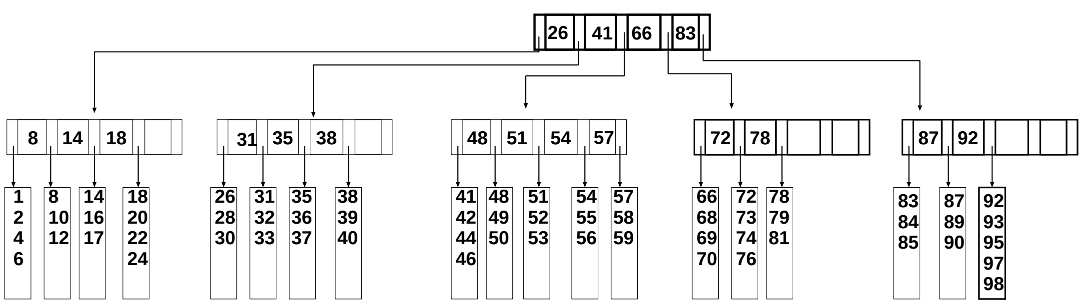

Figure 4: Inserting 17.

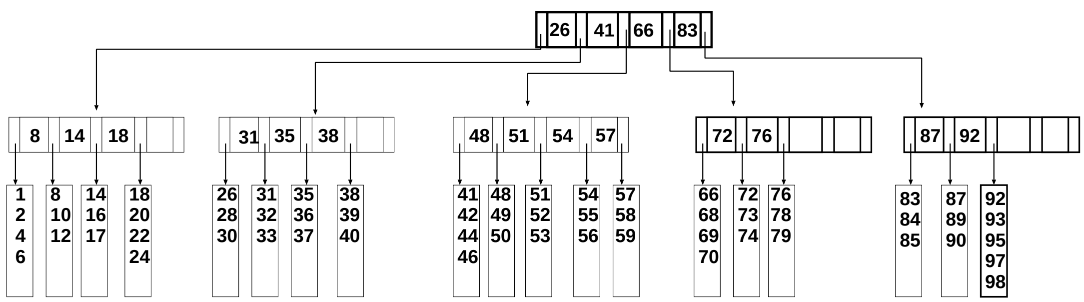

Figure 5: Deleting 81.