Yash Avinash Patole

Course: CS600

Homework 8

CWID: 10460520

Chapter 20

Exercise 20.6.3

For what values of d is the tree T of the previous exercise an order-d a-tree?

Answer:

When d = 3 the tree T an order-d a-tree.

Exercise 20.6.21

Suppose you are processing a large number of operations in a consumer producer process, such as a buffer for a large media stream. Describe an external-memory data structure to implement a queue so that the total number of disk transfers needed to process a sequence of n enqueue and dequeue operations is O(n/B).

Answer:

Consider several inefficient external-memory dictionary implementations based on sequences. If the dictionary sequence is implemented as an unsorted, doubly linked list, then insert and remove can be performed with O(1) transfers each, where each block holds an item to be removed. And, in the O(n) worst-case scenario, searching necessitates transfers because each link hop we make may reach a new block. This search time can be reduced to O(n/B) transfers, where n represents the number of Enqueue and Dequeue operations and B is the number of list nodes that can fit inside a block. We could also use a sorted array to implement the sequence. In this scenario, a binary search performs O(log2 n) transfers. However, in the worst-case scenario, we will need (n/B) transfers to accomplish an insert or removal operation because we may need to access all blocks to shift pieces up or down. As a result, implementations of sequence dictionaries are inefficient for external memory.

We can conduct dictionary searches and updates with only O(logB n) = O(log n/ log B) transfers.

To prevent a single disk transfer, we should conduct up to O(B) internal-memory accesses, where B specifies the size of a block. This many internal-memory accesses are performed by the disk only to move a block into internal memory, and this is only a minor part of the cost of a disk transfer. Thus, O(B) high-speed internal-memory accesses are a tiny price to pay to prevent a time-consuming disk transfer.

To lessen the impact of the performance difference between internal and external memory accesses for searching, we can describe our dictionary using a multi-way search tree, which is a generalization of the (2, 4) tree data structure to a structure known as a (a, b) tree.

Thus, a Buffered Repository Tree is a structure whose insertion cost is less than its lookup cost.

Exercise 20.6.22

Imagine that you are trying to construct a minimum spanning tree for a large network, such as is defined by a popular social networking website. Based on using Kruskal's algorithm, the bottleneck is the maintenance of a union-find data structure. Describe how to use a a-tree to implement a union-find data structure (from Section 70) so that union and find operations each use at most O(log n/ log B) disk transfers each.

Answer:

UNION FIND IMPLEMENTATION USING BTREE - using cross - n tuple lists.

The union find problem is one in which all of the union operations are specified and the find queries are in-line. It can be solved by combining the batched union search algorithm and the persistent B - tree. As you can see in the batched union find algorithm, the information of find queries is not employed until the very end when a ray is shot downwards from the query point and then shot to the left. Ray shooting queries can be handled by a persistent Btree in worst-case O(logB N) IO per query.

As a result, the batched union search method will be executed till we have created the persistent B - Trees on the respective segments. Because a persistent B - tree occupies linear space and may be built in O(SORT(N)) IO. So, using O(SORT(N))

IO, a linear size data structure for the semi-on-line union find issue may be developed, allowing a find query at any moment to be replied in O. (logB N). input: a list of union and find operations; For each element x involved in, the output is a set R of (x,%(x)) pairs. if Σ can be processed in main memory then Call an internal memory algorithm; else Split Σ into two halves Σ1 and Σ2; R1 = UNION-FIND(Σ1); (a) For ∀(x, %(x)) ∈ R1, replace all occurrences of x in Σ2 with %(x); R2 = UNION-FIND(Σ2); (b) For ∀(x, %(x)) ∈ R1, if ∃(y, %(y)) ∈ R2 s.t. y = %(x), replace (x, %(x)) with (x, %(y)) in R1; return R1 ∪ R2.

Chapter 23

Exercise 23.7.11

What is the longest prefix of the string "cgtacgttcgtacg" that is also a suffix of this string?

Answer:

The longest prefix that is also the suffix of this string is "cgtacg".

This can be determined by characters in the first and last strings in the array. Since the array is sorted, common characters among the first and last element will be common among all the elements of the array.

```
class Solution {
   int lps(String s) {

   int start = 0;
   int end = 1;
   int[] store = new int[s.length()];

   while( end < s.length() ){

      if( s.charAt(start) == s.charAt(end) ){
         store[end] = start+1;
         start+=1;
         end+=1;
      }
      else{
            if( start == 0){
                store[end] = 0;
                end=end+1;
            }else{
                start = store[start-1];
                }
            }
        }

        return store[s.length()-1];
        }
}
```

Exercise 23.7.15

Give an example of a text T of length n and a pattern P of length m that force the brute-force pattern matching algorithm to have a running time that is Ω(nm).

Answer:

Following example can be determined:

```
public static int brute(String text,String pattern) {
 int n = text.length();    // n is length of text.
 int m = pattern.length(); // m is length of pattern
 int j;
 for(int i=0; i <= (n-m); i++) {
    j = 0;
    while ((j < m) && (text.charAt(i+j) == pattern.charAt(j)) ) {
       j++;
    }
    if (j == m)
     return i;   // match at i
  }
  return -1; // no match
} // end of brute()
```

Eventually you've checked all of m which is O(m), but the partial matches mean that you have either checked all of n which is O(n)(found a complete match), or at least enough characters to equal the amount of characters in n (partial matches only).

Brute force pattern matching runs in time O(mn) in the worst case.

Exercise 23.7.32

Answer:

Explanation for the Algorithm:

Let M be a subsequence of C, and the first character of M be matched to a position i. However, the initial character of M also appears in C at position j i. It is still legitimate to view the character at j as part of M, and M is a subsequence of C beginning at position j as well.

As a result, for each M character, just the first match starting from the previous character's match is considered. Thus, the algorithm is explained in the following.

Keep two indices, a and b, to iterate over M and C, respectively. Set both to 0.

If the characters M[a] and C[b] match, then both indices are increased by one. This signifies that a M character has been matched. Otherwise, increment b by one. If the end of M is reached before the end of C, then M is a subsequence. As a result, return true. Otherwise, output false. In the worst scenario, b is increased as many times as the length of C, hence the procedure takes O(n) time.

Code for the algorithm:

```
a🡐0, b🡐 0, found 🡐 false
while true:
  if M[a] == C[b]
    a🡐 a + 1
    b 🡐 b + 1
  else
    b 🡐 b+ 1
  if b == len(M)
```

```
    break
  if b == len(C)
    break

return found
```