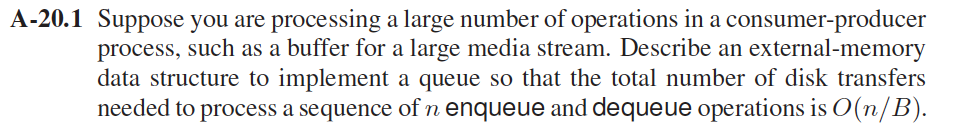
Chapter 20 Exercises: R-20.3,

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

Answer:

In the previous exercise , it is a (4,8) tree. According to the definition of the order-d B-tree. since a=d/2 and b=d ,therefore d is 8.

A-20.1,



Answer:

Considering some external-memory inefficient dictionary implementations  
based on sequences. If the sequence representing a dictionary is implemented  
as an unsorted, doubly linked list, then insert and remove can be performed with  
O(1) transfers each, such that each block holds an item to be removed.  
And, searching requires Θ(n) transfers in the worst case, since each  
link hop we perform could access a different block. This search time can be improved to O(n/B) transfers, where n denotes the n Enqueue and Dequeue operations, B denotes the number of nodes of the list that can fit into a block. We  
could alternately implement the sequence using a sorted array. In this case, a search performs O(log2 n) transfers, via binary search.  
But this requires Θ(n/B) transfers to implement an insert or remove operation  
in the worst case, for we may have to access all blocks to move elements up  
or down. Thus, sequence dictionary implementations are not efficient for external  
memory.

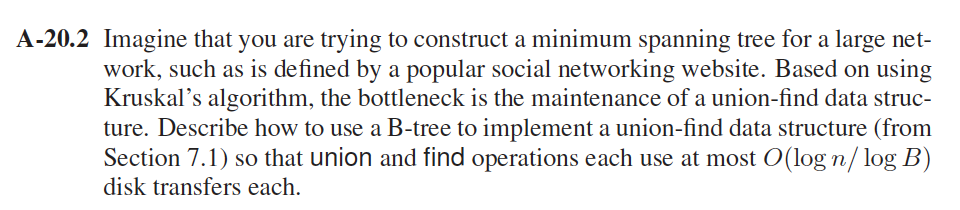
We can perform dictionary queries and updates using only O(logB n) = O(log n/ log B) transfers.

The main idea for improving the external-memory performance of the dictionary  
implementations is that we should perform up to O(B) internal-memory accesses to avoid a single disk transfer, where B denotes the size of a block. The disk performs this many internal-memory accesses just to bring a block into internal memory, and this is only a small part of the cost of a disk transfer. Thus, O(B) high-speed, internal-memory accesses are a small price to pay to avoid a time-consuming disk transfer.

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

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

A-20.2



Answer:

For the union operation, we combine a new tree to the larger tree, do as follow steps:

1. Add the new key to the suitable leaf, Where the search of key brought us

2. If overflow (i.e. leaf increased to b keys)

2.1 Split node in two

2.2 Promote the median key ,i.e. move median key to parent

2.3 If parent has an overflow then repeat

For the find operation, by traversing the muti-way trees, it has smaller heights compared to binary tree, the running time is O(logB n) = O(log n /log B)

Chapter 23 Exercises: C-23.11,

Give an efficient algorithm for deleting a string from a compressed trie and analyze its running time.

Answer:

This algorithm essentially uses tail recursion to remove the word one character at a time. Because the relation is tail-recursive and has many different cases, it is more convenient to use an explicit stack and a for loop as opposed to recursion per words.

Algorithm:

0. initialize an empty stack to hold visited nodes

1. set auxiliary node n to root and add n to the stack

2. for each character c in the given word

2.1 set n to the child node of n that's holding c

2.2 if n is null return false that means word not found

2.3 add n to stack

3. at this point n represents the last char in word so if n is not marked, return false. ELSE

4. unmark n

5. if n is a just a prefix then return true. ELSE

6. pop the stack to remove the last char in word

7. since the word is found and it's not a prefix:

7.1 for each node n on the stack, if n only has one child，delete n's child

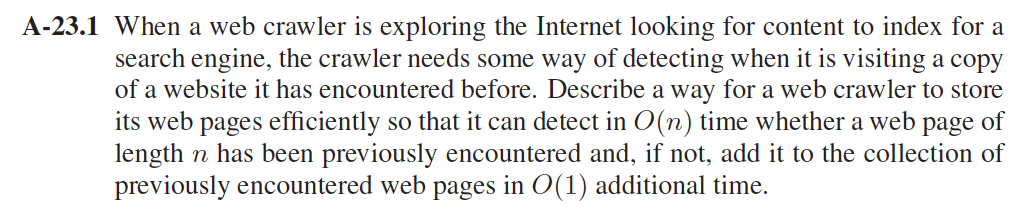
  7.2 if n is marked, return true.

8 return true.

Time Complexity：

1. for the first for loop , it scan the word and if the length of the word is n,and the time is O(n)
2. the second loop is executed not more that n times
3. So the total running time is O(n).

A-23.1,



Answer:

Use the Trie with suffix links. A suffix link in a trie is a pointer from a node for string w to the node corresponding to the longest proper suffix of w.

We treated the web pages as a string with length n. so when we compare the pages, if the pages have been compared before, there is no need to compare them again by using the suffix algorithm:

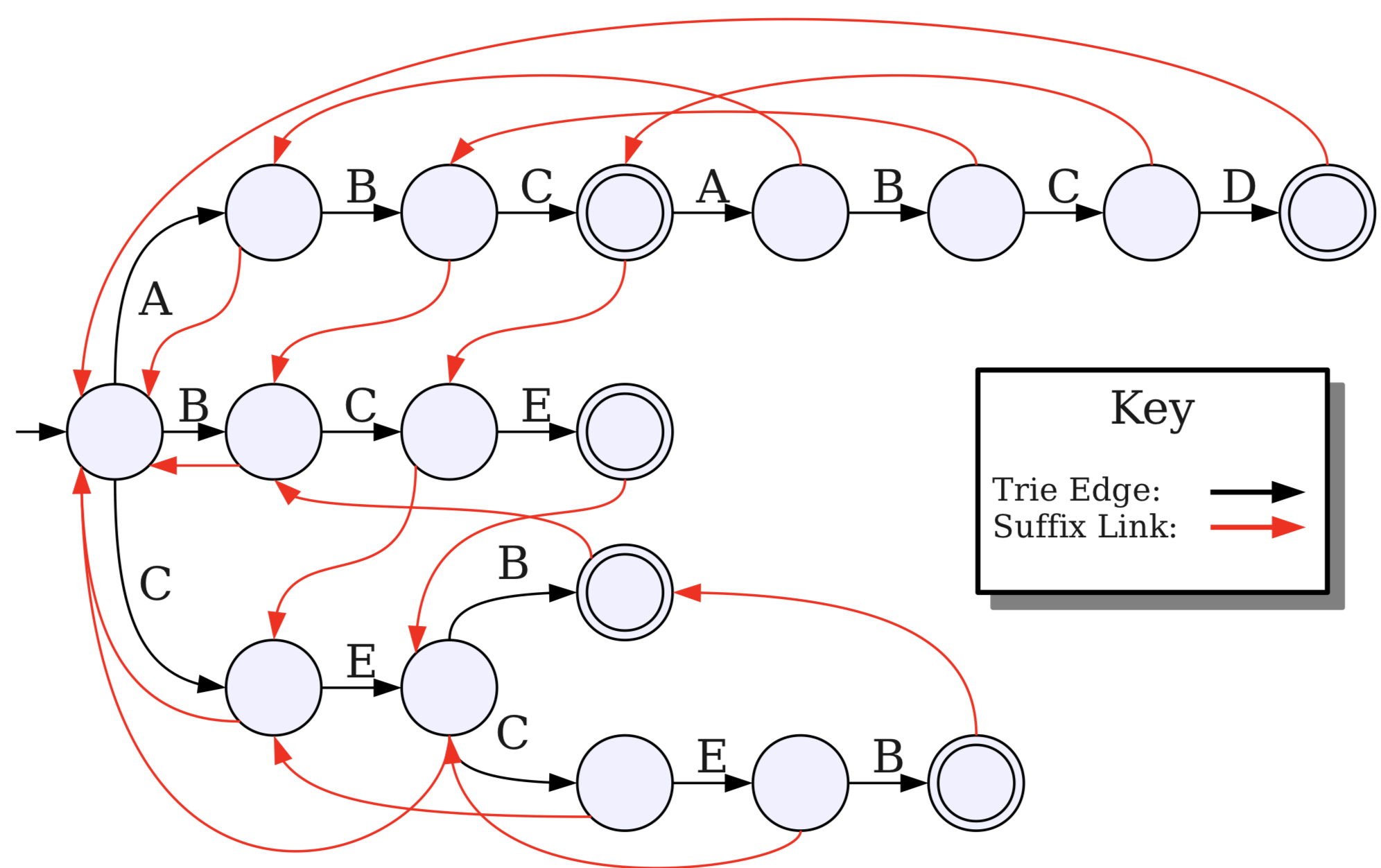
Suffix Algorithm:

1. Let *state* be the start state.
2. For *i* = 0 to *m* – 1

2.1 While *state* is not start and there is no trie edge labeled *T*[*i*]:

Follow the suffix link.

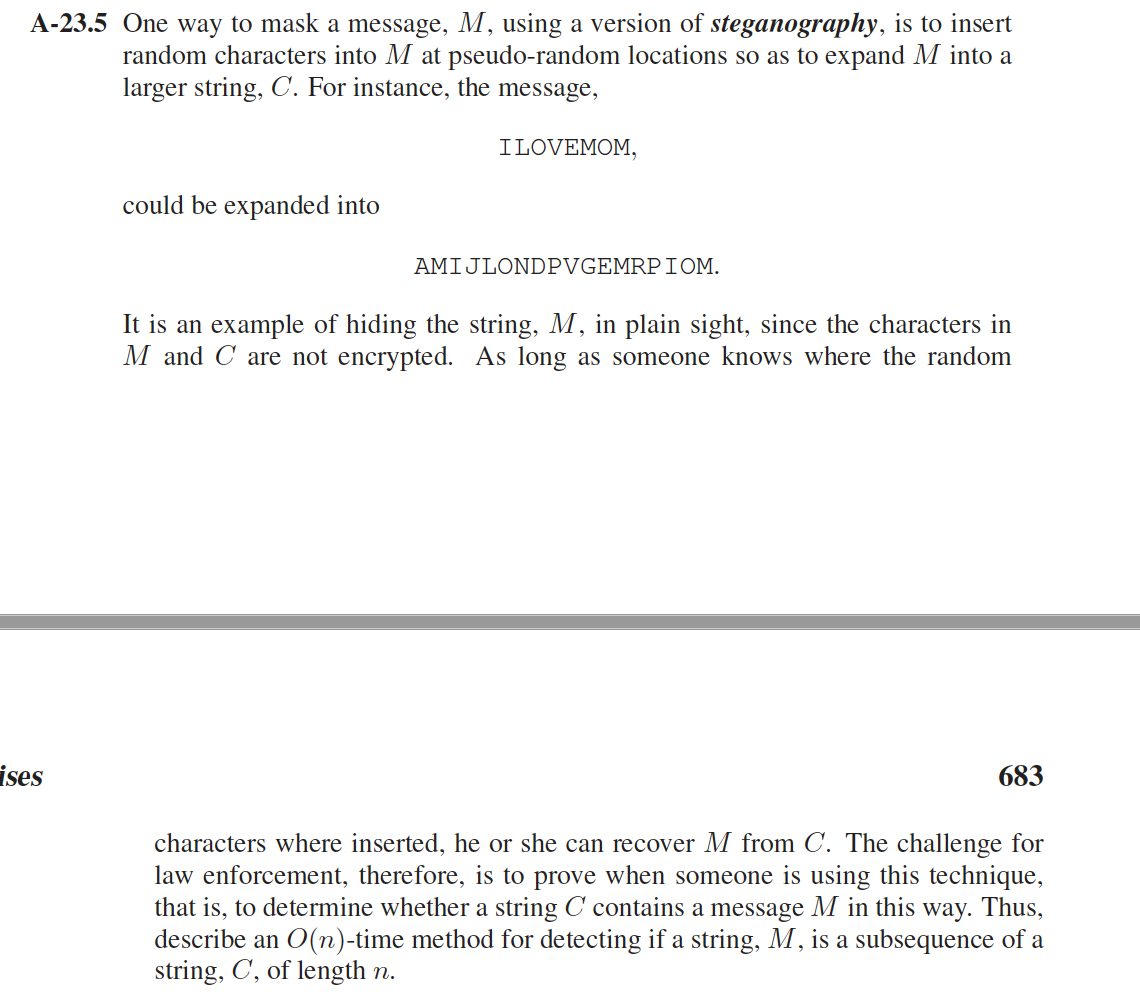
* 1. If there is a trie edge labeled *T*[*i*], follow that edge.



Once the trie is constructed and suffix links added, the runtime of searching through string is O(n).

And if the page is not found. When the search stops,it will be at the longest position in the trie where the page is matched. Then add the rest of the page in the trie with a new child under current node. The running time will be O(1).

A-23.5

****

**Answer:**

* Iterate over all the substring M letter by letter, for example "ILOVEMOM"
* Check if the current letter (iterate 0 is I) is on the string C, if/when it find , remove the ocurrences before it and let the string from that (JLONDPVGEMRPIOM.)
* Do this process for all left substrings
* use a control boolean variable to see if it has found or not.
* if in any pass of the iteration you did not find you return false.

As a result, we only scan the entire String C once, the time complexity is O(n)**.**