CS 600 HW 8
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

- 20.6.3

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

T is a valid (a, b) tree for (4, 8) or (5, 9) tree in the previous exercise.
As the value order-d B-tree in (a, b) with $a = d/2$ and $b = d$
If we take (4,8), the result of d will be 8.
And if we take (5, 9), the result of d will be 9.

- 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 $O(n/B)$.

A queue can be implemented using a linked list to reduce the amount of disk transfers required to complete a sequence of n enqueue and dequeue operations.

Inserting an element at the end of the list will take $O(1)$ disk transfers.

When the block is empty for dequeuing elements from the front, it will return it to the free memory heap.

Thus, all these operations will perform $O(n/B)$ disk transfers.

- 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 B-tree to implement a union-find data structure (from Section 7.2) so that **union** and **find** operations each use at most $O(\log\ n/\log\ B)$ disk transfers each.

B-tree to implement the union-find data structure. Let's say that initially, singleton trees contain all the nodes (with the height of 1).

When a node is connected to a larger group and the number of nodes in the tree is at least doubled, the height of the tree increases by one.

The height of the resulting tree can only be at most $logn$, because there can never be more nodes than n in a tree.

Because there are $O(log\ n)$ nodes to visit, each union operation will take $O(1)$, and doing union will take $O(log\ n)$, the Find operation will therefore take $O(log\ n)$

Thus, implementing a union-find data structure using B-tree with n items executes $O(log_B\ n)$ disk transfers in union and find operation.

- 23.7.11

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

The longest prefix of the string “cgtacgttcgtacg” that is also a suffix of the string is “cgtacg”.

- 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 $\Omega(nm)$.

Let’s consider a text T of length n as AAAAAAAAA………….AS and a pattern P of length m as AAAS While comparing each letter of pattern P in text T, the match will be found at the end of the string.

As, the worst-case running time for brute force is mn then the running time will be $\Omega(nm)$.

- 23.7.32

One way to mask a message, M, using a version of ***steganography***, is to insert random characters into M at pseudo-random locations so as to expand M into a larger string, C. For instance, the message,

$$\text{ILOVEMOM,}$$

could be expanded into

$$\text{AMIJLONDPVGEMRPIOM.}$$

It is an example of hiding the string, M, in plain sight, since the characters in M and C are not encrypted. As long as someone knows where the random characters where inserted, he or she can recover M from C. The challenge for law enforcement, therefore, is to prove when someone is using this technique, that is, to determine whether a string C contains a message M in this way. Thus, describe an $O(n)$-time method for detecting if a string, M, is a subsequence of a string, C, of length n.

If we move left to right through the strings M and C. And a matching character is found, then the pointer in both strings will be increased. Else, we shall only increase the pointer for string C.

If we go through all of M's characters, then M is a subsequence of C.

Time Complexity:
As we traverse the string C, which is of length n, the time complexity will be $O(n)$.