CS 186 Fall 2024 | Introduction to Database Systems Alvin Cheung | DIS 4

1 Buffer Management

(a) Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern **A B C D A F A D G D G E D F**

Least Recently Used (LRU)

														Hit Rate
A														
	B													
		C												
			D											$\frac{}{14}$

Most Recently Used (MRU)

														Hit Rate
A														
	B													
		C												
			D											

CLOCK ("second chance LRU")

														Hit Rate
A														
	B													
		C												
			D											

(b) Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern **A, B, C, D, A, F (remains pinned), D, G, D, unpin F, G, E, D, F** Remember that unpinning does not contribute to the hit count!

Least Recently Used w/ Pinning

A															
	B														
		C													$\overline{13}$
			D												

Most Recently Used w/ Pinning

A															
	B														
		C													
			D												

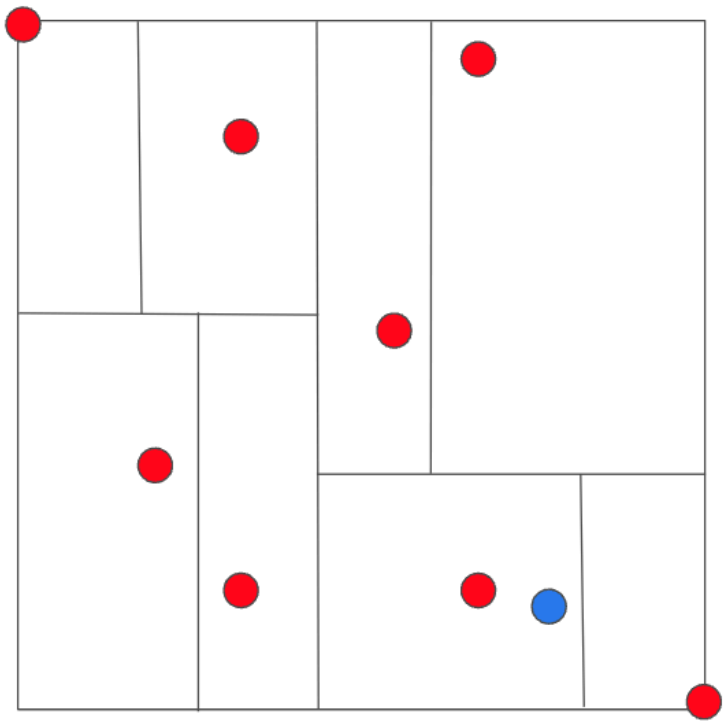
(c) Is MRU ever better than LRU?

(d) Why would we use a clock replacement policy over LRU?

(e) Why would it be useful for a database management system to implement its own buffer replacement policy? Why shouldn't we just rely on the operating system?

2 Nearest Neighbors with K-d Trees

Consider the image below, which shows two dimensional database points (red) partitioned by constructing a k-d tree. We wish to use our k-d tree to find the nearest neighbor to the query point (blue). Suppose that we use the Euclidean distance as our metric.



(a) How many distance calculations between our query point and database points would be needed in a brute force search?

(b) How many distance calculations between our query point and database points would be needed if we use a k-d tree instead?

3 Curse of Dimensionality (Optional)

In this question we will explore why nearest neighbor search using k-d trees degrades from logarithmic time to linear time complexity in higher dimensions. This provides motivation for approximate nearest neighbor indexes, which sacrifice accuracy in exchange for a reasonable runtime.

(a) Suppose we have a database of 1,000 d-dimensional random data points in [0, 1]^d. Given a d-dimensional query data point, we want to find the closest data point in the database. In a 1-dimensional space, starting from our query point, how much of the interval [0, 1] do we expect to search?

(b) Repeat the above but now in a 2-dimensional space. In each of the 2 dimensions, how much of the interval [0, 1] do we expect to search? What about in a 3-dimensional space?

(c) What about in a 128-dimensional space?

(d) Why is this problematic for k-d trees and other tree indexes which rely on spatial partitioning?