Name

CMSC 420 Data Structures

Final Exam
Given Fall, 2000
Dr. Michelle Hugue

Problem 1: " Baron Harkonnen's Big Bad B-Tree Family" (30 pts)

(6) 1.1 Define a $B+$-tree of order m. That is, what conditions must all nodes in a $B+$-tree satisfy. Be sure to address both keys and records.

(6) 1.2 Define *overflow* in the context of a B^+-tree of order m, and explain briefly how the condition is corrected. Drawing a picture may be helpful here in explaining your answer.

(6) 1.3 Define *underflow* in the context of a B^+-tree of order m, and explain briefly how the condition is corrected. Drawing a picture may be helpful here in explaining your answer.

(6) 1.4 In performing a "search-and-insert-if-record-not-found" operation on a B^+-tree, the new key matched a value in the root node, yet the record was inserted.

Is the above scenario valid? If so, explain how. If not, explain why not.

(6) 1.5 Database applications typically use $B+$-trees where m is between 80 and 100. Give two reasons why. Explain your answer for full credit.

Problem 2:"Stilgar's Self-Adjusting Structures (24 pts)

Suppose you have a set of n distinct keys or elements stored in a list. Let $f(n)$ be the average number of elements accessed in satisfying a search query for a single element. Assume that each element is equally likely to be the target of a search.

2.1 (6) Is the following expression for $f(n)$ reasonable?

$$f(n) = \frac{(n+1)}{2}$$

If your answer is yes, explain why. If it is no, explain why not and give a more reasonable value, if one exists.

2.2 (6) Suppose now that we use a *self-adjusting list* to store these keys, where the list is modified after each data access. One way to implement a self-adjusting list is to move the key satisfying the search query to the head of the list, giving the method its name "move to front". New elements are inserted at the head of the list, following a deletion, the predecessor of a deleted element becomes the new list head. Let $g(n)$ represent the average number of elements accessed by a search query applied to the new list.

Would you expect $g(n)$ be less than $f(n)$? If your answer is yes, explain why. Otherwise, explain why not.

Note: you don't need to calculate $g(n)$ or $f(n)$ directly here; you merely need to reason about the effect of the reordering of list elements upon the search process

2.3 (6) When n exceeds a few hundred, the "move-to-front" operation described for problem 2.2 is replaced by a "transpose" strategy, which swaps the key targeted by the search with its predecessor in the list. Why might this be preferred for larger data sets?

2.4 (6) Give conditions under which the self-adjusting list described for this problem might be preferred over a splay tree. Or, if you don't believe such conditions exist, explain why not.

Problem 3: "A Melange of Spicy Bits" (48 pts)

Last, but not least, shorter focused topic questions. They are tailored to a variety of skills; so, don't panic until after you've read them all.

DO any 8 of the TEN problems here—that's right, you can skip TWO (2)!

(6) 3.1 Derive a minimum cost spanning tree for the following *undirected* graph, where the weights are listed after the edge. For example, the weight of the edge (A,B) is given by 2. For full credit, explain how you know that you have achieved a minimum?

Vertex	Edges				
A	<A,B> (2)	<A,C> (1)	<A,D> (3)	<A,E> (2)	<A,F> (4)
B	<B,A> (2)	<B,C> (4)	<B,D> (4)	<B,G> (2)	
C	<C,A> (1)	<C,B> (4)	<C, F> (1)		
D	<D,A> (3)	<D,B> (4)	<D, E> (3)		
E	<E,A> (2)	<E,D> (3)	<E,G> (2)		
F	<F,A> (4)	<F,C> (1)	<F,G> (5)		
G	<G,B> (2)	<G,E> (2)	<G,F>(5)		

(6) 3.2 What is meant by the term *amortized analysis*? Why might this method be preferred over asymptotic analysis for some applications?

(6) 3.3 What is the buddy algorithm? Give a definition of the buddy algorithm, and identify a benefit and a drawback of using it.

(6) 3.4 Each of the functions below has been proposed as a hashing function to map an integer key k to a hash table of of size n. Evaluate them for their acceptability for insert and search operations, and explain the flaw or benefit associated with each. That is, why is it acceptable, or why it is not acceptable.

A. $h(k) = (k + \textbf{random}(n))\ mod\ n$, where **random**$(n)$ is a randomly generated integer from the interval $[0, n-1]$ inclusive, and n is a prime number.

B. $h(k) = k\ mod\ n$, where n is a power of 2.

(6) 3.5 Sparse matrix schemes are only beneficial if the sparse representation takes less space than the entire matrix, including the explicit zeros, as this problem illustrates. Assume that a sparse n x n matrix contains k non-zero integers. Describe and draw a storage scheme that takes advantage of sparseness, and determine the cut-off value of k for your scheme. That is, determine the first value of k for which the sparse storage requirements exceed the space needed for the entire n x n matrix.

(6) 3.6 Explain in words how to perform a breadth first traversal algorithm of the graph G(V,E) using start vertex, s.

(6) 3.7 Explain why merge-sort is in $O(n \log n)$, and indicate what data structure should be used to make this algorithm space efficient. *You are welcomed to use an example to show how the algorithm works–no fancy definition or proof required.*

(6) 3.8 In the context of the project, briefly explain an algorithm for finding the closest base to a given raw material site. For full credit, explain why the algorithm is better than a brute force search of all possible bases.

(6) 3.9 In the context of the project, suppose that you were required to support continuous access to the data in the PM1 quadtree. That is, you should be able to satisfy any of the project functions that use the quad tree, while inserting or deleting sites and paths. However, you must use current data if at all possible, meaning, for example, that as soon as a pending insert has finished, any function initiated thereafter must be able to access that site. How would you modify your project code or operation to support this new constraint?

(6) 3.10 Suppose a depth first search (dfs) algorithm is performed on the graph $G(V, E)$, where the number of vertices is $v = |V|$, and the number of edges is $e = |E|$. Let $p(G, v)$ be the number of vertices visited by dfs. Initialize $p(G, v) = 0$, and add 1 to $p(G, v)$ every time the dfs algorithm examines a vertex, thus counting the examination of a vertex that has been seen before as a visit, which happens in the case of cycles.

Since the algorithm must examine each vertex at least once, we have $p(G, v) \in \Omega(v)$. Is $p(G, v) \in O(v)$? If not, explain why not. If so, explain why, and give a estimate of the worst case value of the growth constant $C > 0$ for which $p(G, v) \leq C * n$ when n is large enough .