# EE 312 Fall ’13 Exam 2 NAME:

39 Points

1. (3 pts) I have a HashTable and I foolishly forgot to keep track of how many keys I’ve inserted. Write a function that calculates the load factor under the following assumptions

The keys are String objects. String objects work reasonably and have operators defined for comparisons (== and <).

There is a function called hash which has a String parameter. Hash returns a uint32\_t value. You should assume that the hash function provides Simple Uniform Hashing.

The HashTable uses the % array\_size method to select an array position.

The HashTable struct is implemented using the same Linked List chains as we did in class. All the structs necessary for this are shown below.

Assume that the HashTable has been properly initialized to contain N random String keys.

uint32\_t hash(String s); // returns a good hash code in O(1) time

struct Node { // used to make chains in the HashTable

String key;

Node\* next;

};

struct HashTable {

typedef Node\* NodePtr; // for convenience

NodePtr\* table; // also Node\*\* table (same type) and it points to the heap

uint32\_t array\_size; // the size of the array that table points to

};

Write a function that calculates and returns the Load Factor of my HashTable

double loadFactor(HashTable ht) {

1. (3 pts) Write a function that returns a pointer to the node in the tree that contains the median value. You can use any of the standard functions we wrote in class (a partial list is provided below). Please assume that the parameter *root* points to the root of the tree and that the nodes in the tree are built with the Node struct shown below. Your solution should not depend on what type is used for PayLoad type. For partial credit, you can write a solution where you assume PayLoad is an int. Please assume that the tree has N nodes (the value N is provided to you as a global variable), and that N is an odd number (so the median is well-defined).

uint32\_t N; // number of nodes in the tree

struct Node {

PayLoad value;

Node\* left;

Node\* right;

Node\* parent;

};

Node\* insert(Node\* root, PayLoad value);

Node\* remove(Node\* root, Payload value);

Node\* smallest(Node\* root);

Node\* successor(Node\* root);

/\* return the node containing the median value in a BST with N nodes (N is odd) \*/

Node\* median(Node\* root) {

1. (3 pts) Sometimes people do stupid things. The following two hash functions are almost identical (they differ by one “&” character). The first function approximates Simple Uniform Hashing when applied to a random collection of English language strings. The second… well, who knows? Note that the missing & causes the loop to terminate as soon as h is set to an even number (the function performs bitwise AND between h and a Boolean value).

uint32\_t hash1(char\* key) {

uint32\_t h = 0;

while ((\*key != 0) && (h = (h \* 33) + \*key++))

;

return h;

}

uint32\_t hash2(char\* key) {

uint32\_t h = 0;

while ((\*key != 0) & (h = (h \* 33) + \*key++))

;

return h;

}

* 1. Assuming I’ve created a HashTable using hash1 that contains N random English strings. What is the (typical) time complexity to determine if the string “blueberry” is stored in that HashTable (please assume the Load Factor is O(1) and the array size is a power of two).
  2. Assuming I’ve created a HashTable using hash2 that contains N random English strings. What is the (typical) time complexity to determine if the string “blueberry” is stored in that HashTable (again, assume the Load Factor is O(1) and the array size is a power of 2).
  3. So, clearly hash2 has a bias for returning even numbers. Recognizing this bias, I am now going to use array sizes that are odd numbers. Specifically, I’m going to use array sizes that are a power of 2 minus 1 (e.g., 127 or 1023). What is the (typical) time complexity to determine if the string “blueberry” is stored in a HashTable created using hash2 with N random English strings with this assumption (assuming the Load Factor is again O(1)).

1. (5 pts) Shown below I have a String struct that is very similar to the one we’ve used in class on several occasions. You may want to read part (b) before answering part (a). Note that part (b) is on the next page.
   1. (3 pts) I want the operator+= function to concatenate strings. I’ve already declared this function for you as a member function. Please write this function so that the expression “x += y” changes the String x so that x becomes the concatenation of x and y (i.e., append y to the end of x).

struct String {

private:

char\* ptr; // points to a null-terminated string on the heap

uint32\_t len; // number of characters in the string

public:

String(char\* str) {

len = 0;

while (str[len] != 0) { len += 1; }

ptr = new char[len+1];

for (uint32\_t k = 0; k < len; k += 1) { ptr[k] = str[k]; }

ptr[len] = 0;

}

~String(void) {

delete[] ptr;

}

char\* c\_str(void) { return ptr; }

/\* append that to the end of this, do not modify that \*/

void operator+=(String that) {

}

};

* 1. (2 pts) I wrote a small program that tested my solution to part a) and while my solution seemed to concatenate the strings correctly, the program seemed to have some sort of double-delete bug. Without changing the way that operator+= is declared, fix the double-delete bug (you can add other member functions to the String struct if this will help fix the bug). You may attempt to describe how you fix the problem (for partial credit), but to receive full credit you must provide C++ code that fixes the problem. Please put that code on this page (i.e, please don’t change the struct on the previous page, as I might not see your changes).

1. (10 pts) Consider the problem of tracking and managing “friends” lists. In this problem, we’ll assume that we have a large number of People objects and that each People object has N friends. This question is concerned with how to best store the information about the N friends. The People objects are identified by email address (i.e., the email address is used as a “key” in this problem). In the questions below, please assume that x and y are People objects
   1. What is the worst case time complexity to determine if [chase@ece.utexas.edu](mailto:chase@ece.utexas.edu) is a friend of x (assume x has N friends)?
   2. What is the worst case time complexity to determine if x and y have any friends in common (assume x has N friends and y has N friends)?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Hash Table with Simple Uniform Hashing (Load Factor < 1.0) | Hash Table without Uniform Hashing (Load Factor < 1.0) | Balanced Binary Search Tree | Unbalanced Binary Search Tree | Sorted Vector |
| a) |  |  |  |  |  |
| b) |  |  |  |  |  |

Place your time complexity responses in the table below, row a) is for question a) and row b) is for question b). You should have a time complexity expression in each column in the table. The various columns refer to different data structures that can be used to implement the friends list.

1. (8 pts) Consider the following C++ code and indicate what the output is for each of the following short programs. If the program does not produce any output, write “NOTHING”. Do not leave any blank. Some statements may print more than one thing. For each statement write on the line provided the output produced by that statement.

struct Foo {

int x;

Foo(void) { this->x = 10; printf(“basic constructor\n”); }

Foo(int x) { this->x = x; printf(“int constructor\n”); }

Foo(Foo& that) {

this->x = that.x;

printf(“copy constructor\n”);

}

~Foo(void) { printf(“destructor for %d\n”, this->x); }

};

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo f(10);

}

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo\* f;

}

1. int main(void) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Foo\* f = new Foo(42);

}

1. void doit(Foo\* g) { \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

int main(void) {

Foo f;

doit(&f);

}

1. (3 pts) Multiple choice/short answer questions
   1. Which of the following errors are exhibited by the program below (choose one response)?
      1. Memory Leak
      2. Double-delete (i.e, use-after-free)
      3. Both of the above
      4. None of the above

int main(void) {

Foo\* f = new Foo;

f->~Foo();

}

* 1. Recall that our basic implementation of *remove* from a binary search tree returned a pointer to the root of the resulting tree. We did this because in some circumstances the act of removing a value from a binary search tree can result in changing which node is the root of the tree. Assume we removed a value and the root pointer was updated and points to a different node as a result. Which of the following **must** have been true?
     1. The tree contained only one value and after removing that value the tree is empty
     2. The tree was not balanced (i.e., the height of the tree was much larger than log N).
     3. The value removed from the tree was the largest value in the tree.
     4. None of the above *have* to have been true (although any one of them *might* have been true).
  2. The benefits of using public/private in C++ is best described by which of the following?
     1. Using private helps avoid buffer overflow errors (e.g., making the data outside of the array private prevents that data from being accessed).
     2. Using private helps avoid double delete bugs and other use-after-free errors (e.g., making a pointer private once you’ve deleted the pointer ensures that the pointer cannot be deleted a second time).
     3. Using private helps discourage programmers from coupling the design of one module in a program to the design of a different module (e.g., making stuff private prevents that stuff from being used in (some) other functions).
     4. Private is used to identify variables whose values should never be displayed on the screen or produced as output (e.g., social security numbers should be private so that the data stays secure).

1. (2 pts) Draw the binary search tree that is produced when the values 1, 2, 3, 42, 15, 50 are inserted in precisely that order.
2. (2 pts) Show the state of the following binary search tree after the value 30 has been removed (assume no changes were made, other than removing 30, draw the resulting tree to the right).

20

10

5

15

30

25

42

40