*COMP 410 Fall 2015*

**Final Exam**

This exam is closed book, notes, calculators, cell phones, classmates, everything but your own brain. You have 180 minutes to complete the exam but a well prepared student will more likely need 120 mins. Do all your work on these exam pages. Please sign here (and print under it) pledging that the work you submit is your own:

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name (print): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem 1 (6%)**

2

**True or False:** Consider the graph G to the right, **G:**

1

then answer T or F (true or false) for each of these:

1

1

3

1. \_T\_\_\_ G has exactly one minimum spanning tree
2. \_F\_\_\_ G has fewer than 4 spanning trees
3. \_T\_\_\_ G is a planar graph

1

2

1

1

1. \_F\_\_\_ G is a complete graph

**Problem 2 (9%)**

**True or False:**

1. \_\_F\_\_ A graph that is already a tree has no minimum spanning tree
2. \_\_F\_\_ A complete graph is always planar
3. \_\_T\_\_ If the number of nodes in a tree is an exact power of 2, then that tree is a bi-partite graph.
4. \_\_T\_\_ If the number of nodes in a tree is odd then that tree is a bi-partite graph.
5. \_\_F\_\_ If a graph is a tree then that graph has no Euler path.
6. \_\_F\_\_ Careful programming can produce a hash function that will work well for all data types.
7. \_\_F\_\_ Carefully programmed, heapsort is a stable sort algorithm.
8. \_\_T\_\_ Carefully programmed, insertion sort is a stable sort algorithm.
9. \_\_F\_\_ Optimal hash table size is a power of 2 that exceeds the expected number of elements to be stored.

**Problem 3 (15%)**

Below are two hash tables. The left one uses linear probing to resolve collisions. The right one will hash into lists to resolvecollisions.

1. First compute the hash value for each word and write it below the word in the blank.
2. Then fill in each table with the following input items inthe order given left to right

 chert garnet opal quartz calcite agate pyrite gem talc stone

hash: \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_\_

Use this hash function: take the position in the alphabet of the second letter, and do mod 13 on it. For example, the word *computer*: second letter is O, which has position 15. 15 mod 13 is 2. So *computer* hashes to slot 2

*an alphabet, with ordinal positions, for your viewing pleasure*

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

 ***0 1 2 3 4 5 6 7 8 9 10 11 12 0 mod 13***

 linear hash into lists

 probing

 0: \_\_\_\_\_\_\_\_\_\_ 0: \_\_\_\_\_

 1: \_garnet\_\_\_ 1: \_\_\_--> garnet 🡪 calcite 🡪 talc

 2: \_calcite\_\_ 2: \_\_\_\_\_

 3: \_opal\_\_\_\_\_ 3: \_\_\_--> opal

 4: \_talc\_\_\_\_\_ 4: \_\_\_\_\_

 5: \_gem\_\_\_\_\_\_ 5: \_\_\_--> gem

 6: \_\_\_\_\_\_\_\_\_\_ 6: \_\_\_\_\_

 7: \_agate\_\_\_\_ 7: \_\_\_--> agate 🡪 stone

 8: \_chert\_\_\_\_ 8: \_\_\_--> chert 🡪 quartz

 9: \_quartz\_\_\_ 9: \_\_\_\_\_

 10: \_stone\_\_\_\_ 10: \_\_\_\_\_

 11: \_\_\_\_\_\_\_\_\_\_ 11: \_\_\_\_\_

 12: \_pyrite\_\_\_ 12: \_\_\_--> pyrite

**Problem 4 (6%)**

Consider the following 3 connected undirected graphs:

 (a) (b) (c)

A

A

A

E

B

B

C

B

E

F

E

D

D

C

C

D

F

 (a) Is there an Euler path? **YES**

 How do you know, using Euler's theorems? **Exactly 2 odd vertices**

 If there is one, show one: **E B A D C A F E D**

 Is there an Euler circuit? **NO**

 How do you know, using Euler's theorems? **> 0 odd vertices**

 If there is one, show one:

 (b) Is there an Euler path? **YES**

 How do you know, using Euler's theorems? **Exactly 0 odd vertices**

 If there is one, show one: **D E A B C E B D**

 Is there an Euler circuit? **YES**

 How do you know, using Euler's theorems? **Exactly 0 odd vertices**

 If there is one, show one: **D E A B C E B D**

 (c) Is there an Euler path? **NO**

 How do you know, using Euler's theorems? **> 2 odd vertices**

 If there is one, show one:

 Is there an Euler circuit? **NO**

 How do you know, using Euler's theorems? **> 0 odd vertices**

 If there is one, show one:

**Problem 5 (5%)**

For this problem consider the graph from problem 4 part (b). This question is about node search order. When you have a choice of next nodes, use *alphabetic order* of node name to resolve the choice (if you can go to A or B next, choose A first for example).

1. Starting with A, write the order in which the nodes are visited in a depth-first search:

**A B C E D**

1. Starting with A, write the order in which the nodes are visited in a breadth-first search: **A B E C D**

**Problem 6 (15%)**

Consider the data structure called the unique queue, or UQUE. Let the operations be **new, add, peek, rest, in, size:**

 **add** puts an item on the tail of the queue

 **peek** returns the head item

 **rest** produces a queue that is whats left when the head item is taken of

 **size**  tells how many items are in the queue

 **in** tells is a particular item is in the queue, or not

In a UQUE, we add elements to the queue at the back, and we remove them from the front like a normal queue. However, we do not allow two or more elements in the queue to have the same value. We do this during the add operation. If I do (for example) add(Q,5) and 5 is already in Q, then nothing is added, and the queue remains the same length as before the add. If 5 is not in Q, then the length of Q grows by 1 and 5 is now at the back of the queue.

Consider implementing this UQUE with some of the data structures we have studied. Let’s consider that the UQUEs might get large… thousands of items in one queue. One way to do the “add” operation with the uniqueness constraint is to keep the queue as a list and search linearly through the list item by item to see if the item being added is already there. The “in” operation would be done similarly.

1. What is the worst-case time complexity of the “add” operation done this way?

O(N)

1. Give a better way to implement the “add” operation ; to do this, describe the data structure you would use and tell the worst-case complexity of the “add” operation using it.

Use a balanced BST to hold the operations in addition to the QUEUE

 Add operation uses O(log N) to check for uniqueness… another O(log N) to add it to the BST if its not there… the O(1) to add to QUEUE… total is O(log N) theoretically.

1. What is the worst-case time complexity of the “in” operation using the (b) approach?

Check the balanced BST takes O(log N)

1. Give another way from (b) to do “add” better; tell what data structure(s) you need, and tell the worst cast time complexity of “add” this other way.

Use a hashmap to keep elements for uniqueness check. Technically worst case for hashmap is O(N) but it is so very unlikely that we treat it as operating in average time, O(1) for uniqueness check.

Then “add” is check hashmap with O(1), and if not there add to hashmap O(1) and enque to QUEUE

at O(1).

1. Using the (d) way, give the worst-case time complexity of the “in” operation

**Worst case is O(N) technically to check hashmap but almost always O(1).**

**Problem 7 (10%)**

Consider the undirected graph below. Find a Minimum Spanning Tree using Kruskal’s Algorithm. Draw your final MST in the box to the right of the original graph using the same node layout so that it is easy to compare the two graphs. Show your work below for each step of the algorithm.

 **Original graph: MST:**

3

B

A

4

2

3

D

1

4

C

E

2

3

1

1

F

G

**Length of paths is 12 in the MST**

**Problem 8 (10%)**

A SET mathematically is a collection of elements where each element appears at most once (no matter how many times the element is inserted into the set). These axioms define abstractly SET of int:

 datatype IntSet =

 New

 | Insert of IntSet \* int ;

 fun member ( New, a ) = false

 | member ( Insert(S,a), b ) = if b=a then true

 else member(S,b) ;

 fun remove ( New, a ) = New

 | remove ( Insert(S,a), b ) = if b=a then remove(S,b)

 else Insert( remove(S,b), a) ;

 fun size(New) = 0

 | size( Insert(S,a) ) = if mem(S,a) then size(S)

 else size(S)+1 ;

An MSET (multi-set) is a SET where elements can appear 0 or more times; each time an element is inserted, the number of times it is in the MSET is incremented; removal causes the number of instances of that element to be decremented (no element can be in the MSET less than 0 times). The size operation will count the total number of *unique* items in the MSET (does not count the multiples).

***Examples:*** S = { 1, 2, 3, 3, 4, 5, 5, 5 } Insert(S,2) = { 1, 2, 2, 3, 3, 4, 5, 5, 5 }

 member(S, 3) = 2 member(S,4) = 1 member(S,8) = 0

 size(S) = 5 size(Insert(S,2)) = 5

**Alter the axioms above as needed to define an MSET of int. Do not make new operations.**

**We wont be writing axioms**

**Problem 9 (14%)**

Consider this adjacency matrix representation for a directed weighted graph (with 0 used for no edge):

 A B C D E F G

 A 0 4 2 0 0 0 3

 B 0 0 0 3 0 4 2

 C 0 1 0 0 0 0 1

 D 0 0 0 0 2 1 0

 E 0 0 0 0 0 0 0

 F 0 0 0 0 2 0 0

 G 0 0 0 1 0 2 0

1. Provide a drawing for this graph 🡺
2. Is the graph acyclic? \_\_\_\_\_\_\_\_\_

 If so, give a topological sort of the vertices in the graph (ignore the edge weights):

 If not, identify a cycle:

1. Using this graph with edge weights, determine the shortest paths from vertex A to all the other vertices in this graph. Use whatever algorithm you like. Fill in this table:

**From A to**

**vertex path length path**

**A** 0 --

**B**

**C**

**D**

**E**

**F**

**G**

***ANSWER THIS WITH YOUR CODE FOR TOPO AND DJIKSTRA’s…***

***Or use the JavaScript code I gave you***

**Problem 10 (10%)**

Consider the construction and use of a minimum binary heap. Show a data sequence containing duplicate values that will show why heap sort is unstable. This means find a data value sequence that will build a heap such that when the min values are taken out, the duplicate values come out in a different order than they went into the heap. The sequence does not have to be long, it just has to have at least 2 duplicate values. To distinguish the duplicates use something like subscripts or dot notation… i.e., if we have two 8 values, make one of them 8.a and the other 8.b.

Show your input sequence, the construction of the original heap, and the removal of min items enough to show the duplicates coming out in reverse order. Show proper delMin behavior.

***You have worked this out your self from my class questions***