Exam # 381 Pile # 12 Room # SIEBL 1304 Seat # 12

University of Illinois at Urbana-Champaign
Computer Science

Second Midterm Exam

CS 225 — Data Structures
Fall, 2015
Tuesday, November 3, 2015 19:00–22:00

- This is a **closed book** and **closed notes** exam.
- You are **not** allowed to use electronic devicess during this exam.
- Do **ALL** problems in this booklet. Read each question very carefully.
- You may detach pages, but **you must return all pages of this exam.**
- Your exam will be digitized for grading. **The last few pages of the exam are your answer book, and only this and the scantron will be used for grading.**
- You are allowed to write helper functions if you think they will help.
- Enter your **NETID** and **UIN** on your scantron. Please double check them for accuracy.
- Enter “**381**” in the `special code` section on your scantron. Bubble it in all the way to the right, and do not enter leading zeros.

Name	
Netid	

I affirm that I neither gave nor received aid when taking this exam.

Signature: ______________________

Multiple Choice

Question 1) (2.5 points)

Here is a preorder traversal of a binary tree (**not** a BST).

`* + 5 * 3 2 9.`

The + and * nodes have two children, and the integer nodes have no children. Which of the following is the postorder traversal of this tree?

a) `* + 9 5 * 3 2`

b) `2 3 5 9 + * *`

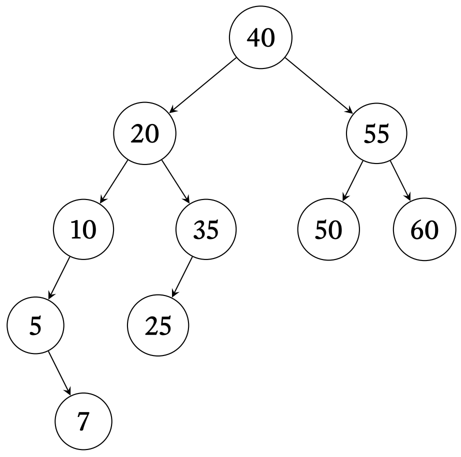
c) `5 3 2 * + 9 *`

d) `9 2 3 * 5 + *`

e) None of these are valid postorder traversals for this tree.

Question 2) (2.5 points)

A node has just been inserted into this AVL tree, causing it to become out of balance.



What kind of rotation is needed to rebalance this tree?

a) A `leftRightRotate` on node 10.

b) A `leftRotate` on node 5.

c) A `leftRightRotate` on node 40.

d) A `rightRotate` on node 40.

e) None of these rebalance the tree.

Question 3) (2.5 points)

Which of the following sequences, if inserted into an empty binary search tree (**not** self-balancing) in the order given, will cause worst-case behavior?

a) `6, 4, 3, 5, 8, 7, 10`

b) `3, 10, 8, 4, 7, 5, 6`

c) `40, 20, 55, 10, 35, 50, 60`

d) `40, 20, 10, 35, 55, 60, 50`

e) None of these cause worst-case behavior.

Question 4) (2.5 points)

You are working on a real-time system and need a very large queue that will **always** run quickly when `enqueue` and `dequeue` are called. You find an implementation that uses two stacks for its implementation. Should you use it?

a) Yes, this will work.

b) No, you cannot implement a queue using stacks.

c) No, because `enqueue` runs in $\mathcal{O}(n)$ time.

d) No, because `dequeue` could suffer from long pauses.

Question 5) (2.5 points)

You need to validate a stream of data consisting of the letters "a" and "b".

A valid stream of data has the same number of a's and b's, but at any point you should never have seen more b's than a's. For example `aababb` is valid, but `abbaab` is not.

What data structure allows you to write your validator using space most efficiently?

a) An integer.

b) A stack.

c) A tree.

d) An iterator.

e) None of these will work.

Question 6) (2.5 points)

You have been given a data structure that is like a binary tree, except some of the child pointers are "back-edges," which point to some ancestor node. You need to search this "tree" for a key that is known to be in the tree. You cannot modify it to keep track of where you have been. It's okay to re-examine parts of the "tree," as long as you are able to search the whole "tree" eventually. How should you proceed?

a) Use an inorder traversal.

b) Use a preorder traversal.

c) Use a depth-first search.

d) Use a level-order traversal.

e) The back edges mean that none of these techniques will work.

Question 7) (2.5 points)

If you have a full tree containing 11 nodes, how many of them are leaf nodes?

a) 4

b) 5

c) 6

d) 7

e) It cannot be determined.

Question 8) (2.5 points)

You want to build a queue using a singly linked list. What does this list need to be able to implement queues efficiently?

a) A tail pointer.

b) An iterator.

c) A sentinel.

d) A `reverse` method.

e) None of these are sufficient.

Question 9) (2.5 points)
Suppose you have an arbitrary B-tree, and consider the left-most leaf node in the tree. You discover that it is of depth d in this tree. What will be the depth of the right-most leaf node?

a) 1

b) $\frac{d}{2}$

c) d

d) It depends on the value of the right-most leaf node.

e) None of these.

Question 10) (2.5 points)
Suppose you have a B-tree that has maximum of 5 children in its internal nodes. What would you expect the minimum number of children in an internal (non-root) node to be?

a) 1

b) 2

c) 3

d) 4

e) 5

Efficiency Analysys

Each item below is a description of a data structure, its implementation, and an operation on the structure. In each case, choose the tightest running time or memory use from the list below. The variable n represents the number of items (keys, data, or key/data pairs) in the structure. In answering this question you should assume the best possible implementation given the constraints, and also assume that every array is sufficiently large to handle all items unless we say otherwise. Please use the scantron sheets for your answers.

a) $\mathcal{O}(1)$

b) $\mathcal{O}(\log n)$

c) $\mathcal{O}(n)$

d) $\mathcal{O}(n \log n)$

e) None of these.

Question 11) (3 points) _____ Push an element onto a stack using an array implementation.

Question 12) (3 points) _____ Insert n elements into an AVL tree.

Question 13) (3 points) _____ Insert an element into a binary search tree (not necessarily balanced) of n elements.

Question 14) (3 points) _____ Size of the call-stack needed to do a preorder traversal of a balanced tree.

Question 15) (3 points) _____ Resize a full array-based queue.

Question 16) (3 points) _____ Calculate the height of each node in a tree.

Question 17) (3 points) _____ Dequeue an element from an array-based queue.

Question 18) (3 points) _____ Print the list of paths from each leaf node of a balanced binary tree to the root. (Printing one node is $\mathcal{O}(1)$.)

Question 19) (3 points) _____ Amount of memory needed for a level-order traversal of a tree.

Question 20) (3 points) _____ Amount of memory needed to implement a forward iterator for a linked list.

A Stepable Inorder Traversal

Write your solutions on the "answer pages" at the back of this exam.

In this section you will implement a class that will perform an inorder traversal of a binary tree, k steps at a time. In other words, we do not want to precompute the traversal, we want to compute it as necessary.

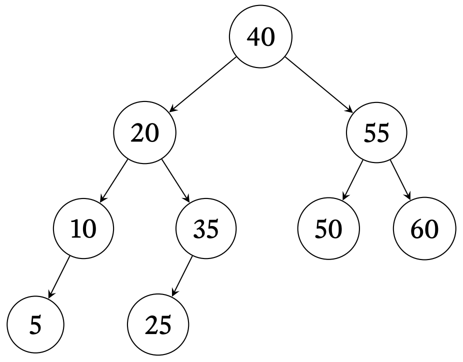
For this problem we will use the following tree implementation. You can assume that correct implementations of usual things like constructors, destructors, add, find, etc. are provided, but they are not shown here to save space.

```
template<class T>
class Stack {
    private:
       // doesn't matter
    public:
       T pop();
       void push(T data);
       T peek();
       bool isEmpty();
};

class Tree {
    private:
        class TreeNode {
            public:
                int data;
                TreeNode *left;
                TreeNode *right;
            };

        TreeNode *root;

    public:
        class KStep {
            private:
                Stack<TreeNode *> st;
                void pushAll(TreeNode *n);
                int step1();
            public:
                KStep();
                bool hasMore();
                int step(int k);
        };

    // assume other reasonable methods as well
};
```

Question 21) (5 points)

Write the `pushAll` function. Given a node `n`, `pushAll` pushes it and all its left descendents onto the stack `st`.

For example, given this tree:



`pushAll` on 55 would push 55, then 50.
`pushAll` on 40 would push 40, then 20, then 10, then 5.

Question 22) (5 points)

`KStep` uses `pushAll` to step through a traversal. The top of the stack will contain the next element of the traversal. After a node n is removed from the stack, we call `pushAll` on n's right child, if it exists. You might want to sketch this out to be sure you understand why this works.

Now that you have `pushAll`, write the `KStep` constructor. For the tree given above, calling the constructor results in a stack having values 40, 20, 10, and 5, with 5 being at the top of the stack.

Question 23) (3 points)

Write the `hasMore` function. It returns `true` if there are elements left in the tree to traverse, and `false` if all nodes have been traversed.

Question 24) (5 points)

Write the function `step1`. It will return the current element of the traversal and adjust the stack for the next element. Assume `step1` is never called on an empty stack.

Question 25) (4 points)

Write the function `step(k)`. It will step `k` steps of the traversal, returning the initial element. E.g., if the traversal is on the node containing 5 in the above tree, and you then call `step(4)`, it will return 5 and leave the stack with 35 as the top element. If `step(k)` exhausts the stack, it should return, leaving the stack empty.

By the way, congratulations. You have implemented an inorder iterator!

Question 26) (5 points)

What is the running time of `step1`? We want to know lower bound, average, and upper bound.

Quadtrees

For this question, consider the following partial class definition for the `Quadtree` class, which uses a quadtree to represent a square bitmap image as in MP5.

```
class Quadtree
{
  public:
     // constructors and destructor; all of the public methods from MP5, including:

   void buildTree(PNG const & source, int resolution);
   RGBAPixel getPixel(int x, int y) const;
   PNG decompress() const;
   void clockwiseRotate();
   void prune(int tolerance);
   int pruneSize(int tolerance) const;
   int idealPrune(int numLeaves) const;

   void perfectify(int levels); // for question 28

  private:
     class QuadtreeNode
     {
        QuadtreeNode* nwChild;  // pointer to northwest child
        QuadtreeNode* neChild;  // pointer to northeast child
        QuadtreeNode* swChild;  // pointer to southwest child
        QuadtreeNode* seChild;  // pointer to southeast child

        RGBAPixel element;  // the pixel stored as this node's "data"
     };

     QuadtreeNode* root;    // pointer to root of quadtree, NULL if tree is empty.

     QuadtreeNode* buildPerfectTree(int k, RGBAPixel p); // for question 27

};
```

You may assume that the quadtree is complete and that it has been built from an image that has size $2^k \times 2^k$. As in MP5, the element field of each leaf of the quadtree stores the color of a square block of the underlying bitmap image; for this question, you may assume, if you like, that each non-leaf node contains the component-wise average of the colors of its children. You may not use any methods or member data of the `Quadtree` or `QuadtreeNode` classes which are not explicitly listed in the partial class declaration above. You may assume that each child pointer in each leaf of the quadtree is NULL.

Quad Trees

Write your solutions to these questions on the answer pages at the back of this exam.

Question 27) (5 points)
Write the function `buildPerfectTree(int k, RGBAPixel p)` mentioned on the previous page. It returns a pointer to the root of a perfect quadtree of k levels where every leaf contains value `p`.

Question 28) (5 points)
Write a function `perfectify(h)` mentioned on the previous page. It is the opposite of pruning. It will use `buildPerfectTree` to make the current tree a perfect tree of height h. You can assume that h will be greater or equal to the height of the quadtree.

Question 29) (3 points)
What is the running time of `perfectify` in terms of h?

Sloppy AVL Trees

Question 30) (10 points)
You know and love the fact that the height of an n-node AVL tree is $\mathcal{O}(\log n)$, so you instruct your programming team to impement one for you. But the programming team you manage did not follow your instructions! Instead of every node having a balance b such that $-1 \leq b \leq 1$, their implementation allows balances such that $-2 \leq b \leq 2$.

We'll call this implementation a "Sloppy AVL Tree," or SAVL tree for short.

The product has to ship tonight, so there's no time to fix this. Is the height still $\mathcal{O}(\log n)$, or do we need to cancel the project?

We'll use a proof by induction to find out.

Fill in the blanks on the answer sheets to complete the proof.

Answer Pages

Question 21 (`pushAll`) answer:

Question 22 (`KStep`) answer:

Question 23 (`hasMore`) answer:

Question 24 (`step1`) answer:

Question 25 (`step1` running time) answer:

Question 26 answer:

Lower Bound	
Average	
Upper Bound Case	

Question 27 (`buildPerfectTree`) answer:

Question 28 (`perfectify`) answer:

Question 29 (`perfectify` running time) answer:

Question 30 answer:

You may answer this question by filling in these blanks, or use the blank space for your own proof/disproof.

Preliminaries Let $H(n)$ denote the maximum height of an n-node SAVL tree, and let $N(h)$ denote the minimum number of nodes in an SAVL tree of height h. To prove (or disprove!) that $H(n) = \mathcal{O}(\log n)$, we attempt to argue that

$$H(n) \leq 3 \log_2 n, \text{for all } n$$

Rather than prove this directly, we'll show equivalently that

a) $N(h) \geq$ ____________________, (1pt)

Proof For an arbitrary value of h, the following recurrence holds for all SAVL Trees:

b) $N(h) =$ __________ $+$ __________ $+$ __________, (2pt)

c) and $N(0) =$ __________, $N(1) =$ __________, $N(2) =$ __________, (2pt)

We can simplify this expression to the following inequality, which is a function of $N(h-3)$:

d) $N(h) \geq$ __________ $\times$ __________, (1pt)

By an inductive hypothesis, which states:

e) ____, (1pt)

we now have

f) $N(h) \geq$ ____________________ = part (a) answer, (1pt)

which is what we wanted to show.
Given that $2^0 = 1, 2^{1/3} \approx 1.25$, and $2^{2/3} \approx 1.58$, what is your conclusion?
Is an SAVL tree $\mathcal{O}(\log n)$ or not? (Circle one): (2pt)

YES NO

Overflow Page

Use this space if you need more room for your answers.