

CMSC132

Partial Solutions to Final Exam Practice Questions

Problem 1 Software Engineering & Object Oriented Design

A. Software Development and Testing

- a. Software is difficult because programmers are slow T or F
- b. Software life cycle refers to how software is used T or F
- c. Problem specification is a component of software development T or F
- d. Problem specification is less important than program testing T or F
- e. Iterative development is a software development methodology T or F
- f. Black box testing is usually easier than clear box testing T or F
- g. Integration tests are usually more important than unit tests T or F
- h. Test coverage includes consideration of lines of code tested T or F
- i. Test drivers are only found on NASCAR training tracks T or F

B. Object-oriented design

- a. State, behavior, and identity are the main qualities of objects T or F
- b. Object oriented design produces faster programs T or F
- c. List two main principles of object-oriented design? **Abstraction & encapsulation**
- d. Inheritance describes a relationship between classes T or F
- e. Inheritance discourages code reuse T or F
- f. Extension is a form of inheritance T or F

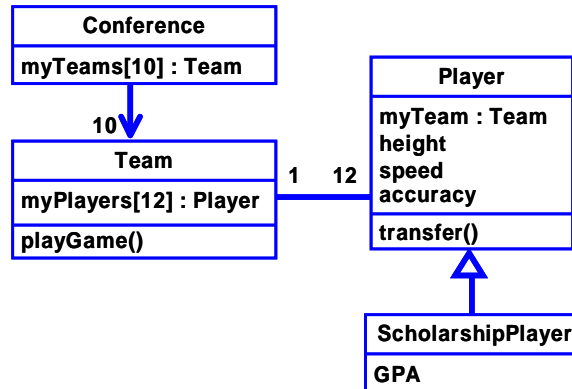
C. Object-oriented design II. Given the following problem description, produce an object-oriented solution. Answer the following questions about your object-oriented solution.

Design a simulation of a basketball conference. Each conference has 10 teams. Each team has 12 players. Each player has a specific height, speed, and accuracy. Players know which team they belong to. Some players are scholarship players. Scholarship players need to record their current grade-point average. Players may be transferred between teams. Teams play basketball games against other teams in the conference. The result of each game is determined using a function based on the height, strength, speed, and accuracy of the players on each team.

- a. What are the objects in your object-oriented solution?
Conference, Team, Player, ScholarshipPlayer
- b. What are the interactions between your objects?
Play Game, Transfer
- c. Which objects “have” other objects? (Also list target object)
Conferences have Teams, Teams have Players,
- d. Which objects “use” other objects? (Also list target object)
None
- e. Which objects “are” other objects? (Also list target object)

ScholarshipPlayers are Players

f. Draw a UML diagram of your solution.



Problem 2 Algorithmic Complexity

D. Algorithmic complexity

- What is algorithmic complexity?
Amount of resources required by algorithm with respect to problem size
- List a reason benchmarking is better than analyzing complexity
Measures performance for a particular hardware
- What is the difference between best case, worst case, and average case?
Minimum, maximum, and typical number of steps required by algorithm
- What does the Big-O notation represent?
Upper bound on the number of steps required by algorithm
- Why are $O(n^2)$ and $O(n)$ not considered equivalent?
The number of steps required by each algorithm grows at a different rate with respect to the problem size

E. Finding critical regions

Calculate the asymptotic complexity of the code snippets below (using big-O notation) with respect to the problem size n :

- ```

for (i = 0; i < n; i=i*2) {
 for (j = 1; j < n; j++) {
 ...
 }
}

```

$f(n) = O(\mathbf{n \log(n)})$
- ```

for (i = 0; i < n-2; i++) {
    for (j = 0; j < n; j=j*2) {
        for (k = 1; k < 5000; k=k*5) {
            ...
        }
    }
}

```

$f(n) = O(\mathbf{n^2})$

```

    }
    for (j = 0; j < 1000*n; j=j+1) {
        ...
    }
    for (j = 0; j < n/5; j=j+5) {
        ...
    }
}

```

Problem 3 Data Structures and Recursion

F. Taxonomy & properties

- Describe the main difference between linear and hierarchical data structures
Single successor vs. multiple successors
- What is the key property of a binary search tree?
Smaller values in left subtree, larger values in right subtree
- On average, what is the complexity of doing an insertion in a binary search tree?
 $O(\log(n))$
- Pre-order, in-order, and post-order are all depth-first traversals **T or F**
- What operation(s) supported by binary search trees are not supported by heaps?
find(n)
- What is the difference between a set and a map?
Maps have keys to their elements
- What happens when an open addressing hash table is close to full?
Operations take more steps
- Describe the 2 main parts of a recursive algorithm
Base case and recursive step

Problem 4 Graph Algorithms

G. Properties

- Describe the main difference between hierarchical and graph data structures
Single predecessor vs. multiple predecessors
- Describe the difference between a directed and undirected graph
Edges are one-way or bi-directional
- Describe the difference between a path and a cycle
Cycle returns to its starting node
- Describe two methods of storing edges in a graph. Which requires more space?
Edge list & adjacency matrix
Space usage depends on Edge/Node ratio of graph

H. Traversals

- Why is graph traversal more difficult than a tree traversal?
May encounter cycles
- Describe the difference between a breadth-first and depth-first traversal of a graph

Breadth-first visits closer neighbors first, depth-first finishes exploring all reachable nodes from current node first

I. Minimum spanning trees

- a. Given the following Java class definition for a graph

```
public class MyGraph<E> {
    public class Node<E> {
        E myValue;
        boolean tag;
        ArrayList<Node> myNeighbors;
    }
    ArrayList<Node> myNodes;
    void visitNode(Node n)    { /* Action to be performed when traversing node */ }
    void deptFirstSearch(Node n) { /* Perform depth-first search of graph starting at n */ }
}
```

- i. Write code for the method `depthFirstSearch(n)` that performs a depth first traversal starting at node `n`.
- ii. Write code for the method `breadthFirstSearch(n)` that performs a breadth first traversal starting at node `n`.

J. Minimum spanning trees

- a. What is a spanning tree?

Tree connecting all nodes in graph (no cycles, n-1 edges)

- b. Describe Kruskal's algorithm for finding minimum spanning trees

Sort edges by weight, continue adding remaining edge with lowest weight if does not create cycle, until spanning tree formed

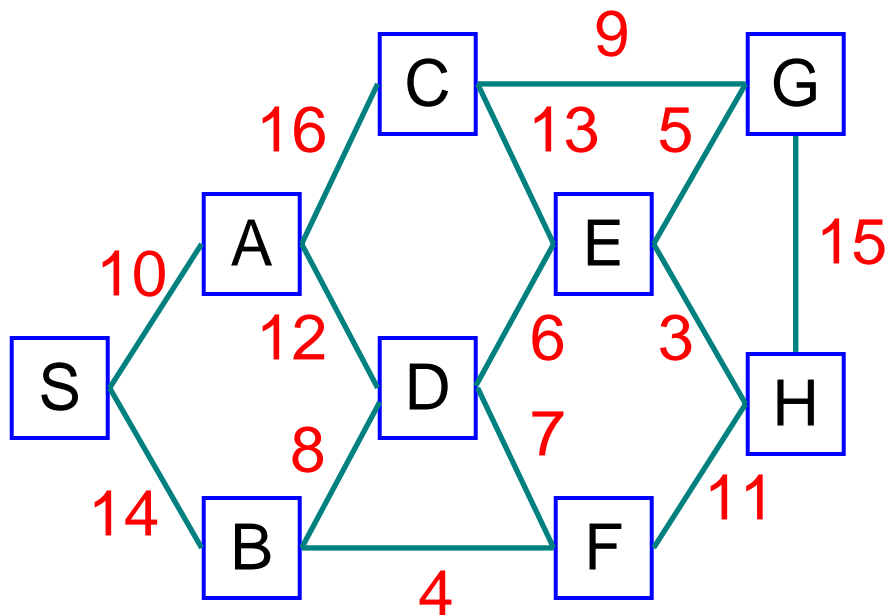
- c. Describe Prim's algorithm for finding minimum spanning trees

- d. Describe two methods for finding connected subgraphs

Traverse graph looking for cycle, or track connected subgraphs

- e. Consider the following graph. Using both Prim's and Kruskal's algorithm, calculate the minimum spanning tree, listing edges in the minimal spanning tree in the order they are added to the tree.

Kruskal - (E,H), (B,F), (E,G), (D,E), (D,F), (C,G), (S,A), (A,D)



K. Single source shortest path

- a. Describe Dijkstra's algorithm for finding shortest paths in a graph
Maintain set of nodes with known shortest path from start (and their costs & predecessors), repeatedly add node closest to set (updating table of costs & predecessors) until all nodes in set.
- b. Consider the previous graph. Apply Dijkstra's algorithm for this graph to calculate the shortest path from S to every other node. Store intermediate results in the table BestKnownDistances. Show the entries in the table after you finish computing the shortest distance from S to nodes in the set {S, A, B}.

Table BestKnownDistances

	S	A	B	C	D	E	F	G	H
LowestCost	0	10	14	26	22	infinity	18	infinity	infinity
Predecessor	none	S	S	A	A or B	none	B	none	none

- c. Which node would be processed next using Dijkstra's algorithm?
F
- d. Update the table BestKnownDistances after adding this node.

	S	A	B	C	D	E	F	G	H
LowestCost	0	10	14	26	22	infinity	18	infinity	29
Predecessor	none	S	S	A	A or B	none	B	none	F

- e. Using Dijkstra's algorithm, calculate the shortest path (and its cost) from S to every other vertex in the graph. List vertices in the order they are added to the table BestKnownDistances.

S, A, B, F, D, C, E, H, G

S, cost = 0

S->A, cost = 10

S->B, cost = 14

S->B->F, cost = 18

S->A->D or S->B->D, cost = 22

S->A->C, cost = 26

S->A->D->E or S->B->D->E, cost = 28

S->B->F->H, cost = 29

S->A->D->E->G or S->B->D->E->G, cost = 33

	S	A	B	C	D	E	F	G	H
LowestCost	0	10	14	26	22	28	18	33	29
Predecessor	none	S	S	A	A or B	D	B	E	F

Problem 5 Compression & Huffman Codes

L. Compression

- a. What are two sources of compressibility?

Redundancy & human perception

- b. What are two types of compression?

Lossy & lossless

- c. What is a prefix code?

No prefix of a code appears as another code

- d. What are Huffman codes?

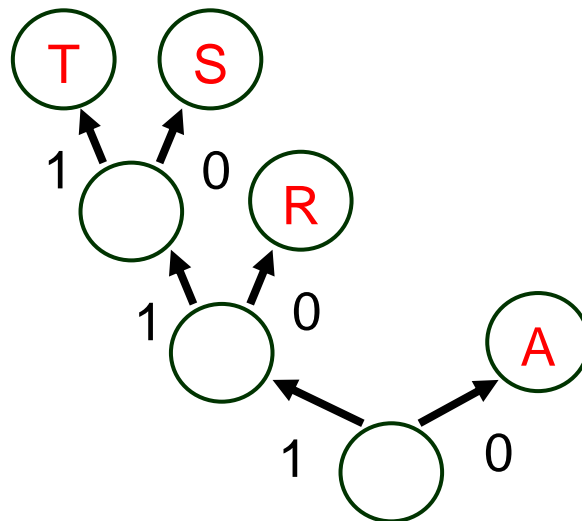
Binary prefix code based on symbol frequencies

- e. How do Huffman codes achieve compression?

Taking advantage of redundancy of frequently appearing symbols

- f. Given the following Huffman tree, decode the sequence "00110010"

AASAR



- g. Using the same Huffman tree, encode the string “arts” as a sequence of 0’s and 1’s
010111110
- h. Given the following symbol frequencies, create a Huffman tree for the symbols
A = 5, B = 8, C = 4, D = 6, E = 7

Problem 6 Java Language Features

M. Regular expressions in Java

- a. What are regular expressions?
Notation for describing simple string patterns
- b. What are finite automata?
Simple computing machines consisting of a finite # of states and transitions
- c. What is the relationship between regular expressions and finite automata?
Finite automata may be used to recognize strings generated by regular expressions, and vice versa
- d. What is the motivation for using regular expressions?
Useful for text processing, finding/extracting patterns
- e. Given the following Java code fragment using java.util.regex:

```
Pattern p = Pattern.compile("[a-z]+");
Matcher m = p.matcher("a0 ab5 42 cd ef13");
while (m.find()) { System.out.print(m.group() + "\n"); }
```

What will be printed out when the code is executed?

a ab cd ef

- f. Given the following Java code fragment using java.util.regex:

```
Pattern p = Pattern.compile("[a-z0-9]+");
Matcher m = p.matcher("a0 ab5 42 cd ef13");
while (m.find()) { System.out.print(m.group() + "\n"); }
```

What will be printed out when the code is executed?

a0 ab5 42 cd ef13

- g. Given the following Java code fragment using java.util.regex:

```
Pattern p = Pattern.compile("[a-z]+[0-9]");
Matcher m = p.matcher("a0 ab5 42 cd ef13");
while (m.find()) { System.out.print(m.group() + "\n"); }
```

What will be printed out when the code is executed?

a0 ab5 ef1

N. Java Inner Classes

- a. What are inner classes?
Classes defined in the scope of another class
- b. What are nested classes?
Static inner classes
- c. When should inner classes be used?
To define classes that need to access private members of enclosing class
- d. When should anonymous inner classes be used?
When name of inner class is not needed because it is used only once
- e. Write an example anonymous inner class in Java.

O. Java support for Object-Oriented programming

- a. The equals() method in Java is typically used to test for name equivalence T or F
- b. All non-static initialization blocks are executed when objects are created T or F
- c. Code in initialization blocks are executed at the end of every constructor T or F
- d. If no visibility modifier is specified, methods are private by default T or F
- e. Protected access is less visible than package access T or F

P. Exceptions in Java

- a. What are exceptions?
Run-time errors detected during program execution
- b. How are exceptions used in Java?
To represent run-time errors found by Java
- c. What should be made an exception in Java?
Serious run-time errors (should be made unchecked exceptions) and common errors the program can handle (should be made checked exceptions)
- d. What are the differences between try, catch, and finally in Java?
Try surrounds code that may cause exception, catch specifies exceptions caught and action performed, finally specifies action performed after try block exits (whether an exception is thrown or not)
- e. What is the difference between checked and unchecked exceptions?
The Java compiler attempts to require all checked exceptions to be caught at some point in the Java program.
- f. Given the following code

```
public static void f( int y ) {  
    try {  
        System.out.print("A");  
        int x = 1 / y ;    // generates ArithmeticException if y == 0  
        System.out.print("B");  
    }  
    catch (ArithmeticException e) {
```

```

    System.out.print("C");
}
finally {
    System.out.print("D");
}
}

```

What will be printed for the following method calls?

1. f(1) **ABD**
2. f(0) **ACD**

Q. (4 pts) Cloning and serialization in Java

- a. What is cloning?
Creating an identical copy of an object
- b. What is the relationship between clone() and the == operator?
A cloned object is not equivalent to the original when compared with ==
- c. What is the relationship between clone() and equals()?
A cloned object should be equivalent to the original when compared with equals()
- d. What is serialization?
Converting a graph of Java objects into a stream of data
- e. What is serialization used for?
Provide persistence, copy, or communicate a graph of Java objects
- f. What is the difference between making a shallow copy versus making a deep copy?
Shallow copy copies the object only, deep copy copies the object and all objects referred to by the object

Problem 7 Multithreading & Synchronization (20 pts)

R. Multithreading

- a. What is the motivation for writing multithreaded Java code?
Capture logical structure of problem, better utilize hardware resources
- b. What are possible states for Java threads?
New, Runnable, Running, Blocked, Dead
- c. What is the effect of invoking the start() method of the Thread class?
Tells JVM thread is now Runnable
- d. What is the effect of invoking the join() method of the Thread class?
Tells JVM to wait until thread is Dead before continuing execution
- e. What is scheduling?
Deciding which thread(s) to run, and for how long
- f. What is the difference between preemptive and non-preemptive scheduling?
Preemptive scheduling can stop threads, non-preemptive scheduling has to wait for threads to stop
- g. What are data races?
Concurrent accesses to shared variable, where at least one access is a write
- h. Write a Java program that can experience a data race.

- i. Why should programs avoid data races?
Results may differ based on scheduling, causing intermittent errors that are hard to debug

S. Synchronization & deadlocks

- a. What is synchronization?
Coordinating events with respect to time
- b. Why should programs use synchronization?
To eliminate data races in multithreaded code
- c. What are Java locks?
Entity associated with each Object that can be held by one thread at a time
- d. How may Java locks be used?
Java locks may be used to enforce mutual exclusion (by having multiple threads attempt to hold the same lock before executing) for a region of code
- e. What are deadlocks?
Threads are blocked waiting for locks with no possibility of making progress, program is unable to continue execution
- f. Why should programs avoid deadlocks?
Program will not complete execution
- g. Write a Java program that can experience deadlock.

T. Multithreading code

Consider the following code:

```
public class mySet {
    List myElements = new ArrayList();

    public boolean add( Object o ) {
        myElements.add( o );
    }

    public Object remove( ) {
        if (myElements.isEmpty() == false)
            return myElements.remove( 0 ); // removes & returns object at position 0
        return null;
    }
}
```

- a. What may happen if an object of the class mySet is used by multiple threads calling add() and remove() at the same time?
Data race on shared variable myElements
- b. Change the add() and remove() methods so that the class mySet can be safely used by multiple threads at once.

Add synchronized keyword to both methods

- c. Change the add() and remove() methods so that the method remove() will always return an object when used by multiple threads (by waiting until an object has been added).

Problem 8 Networking & Networking Support in Java

U. Networking

- a. What are protocols?
A formal description of formats and rules
- b. What is the internet?
A combination of multiple layers of networking protocols
- c. What are packets?
A fixed-size piece of data (with header information & actual data)
- d. What is IP? UDP? TCP/IP?
All 3 are protocols: Internet Protocol, User Datagram Protocol, Transmission Control Protocol
- e. What is a socket? Port? URL?
All three are abstractions: application-level abstraction of network connection (socket), abstraction of destination at IP address (port), abstraction of web resource (Uniform Resource Locator)
- f. What is the difference between reliable and unreliable network connections?
Reliable network connections send data in order & report lost data
- g. How can a reliable connection be built on top of an unreliable network?
Using a protocol based on round-trip communication between sender/receiver
- h. What is a server? Client?
Server waits for communication and provides services, client initiate communication and requests services
- i. What is the difference between Java Socket, ServerSocket, and DatagramSocket?
Java socket connections using TCP (client) or TCP (server), vs. UDP
- j. How is data transported across a Java Socket? Across a DatagramSocket?
Using TCP vs. UDP

Problem 9 Graphic User Interfaces

V. GUIs and MVC

- a. In a GUI, what is the model? The view? The controller?
Model is the data, View is what is displayed, Controller is used by user to interact with the model & view
- b. Why should they be kept separate?
Reduce complexity of software
- c. What are events?
External events that occur outside the control of the program that must be handled by the program
- d. Why are events used in GUIs?

To represent user actions

- e. How are events handled in the Java Swing library?
Events are generated by individual components (e.g., JButton) and handled using ActionListeners registered for the component

Problem 10 Sorting & Algorithm Strategies

W. Sorting algorithms

- a. What is a comparison sort?
Sorting algorithm using only pairwise key comparisons
- b. When is a sorting algorithm not a comparison sort?
Depends on qualities beyond pairwise key comparisons (e.g., all keys have values between X and Y)
- c. What is a stable sort?
Sorting algorithm leaves relative order of keys with equal value unchanged
- d. What is an in-place sort?
Sorting algorithm only needs small constant amount of additional space
- e. What is an external sort?
Sorting algorithm is designed for efficiency when keys do not all fit in memory (i.e., very large number of keys)
- f. What is the average case complexity of sorting using
 - i. bubble sort **$O(n^2)$**
 - ii. heap sort **$O(n \log(n))$**
 - iii. quick sort **$O(n \log(n))$**
 - iv. counting sort **$O(n+k)$ for keys from 1..k**
- g. What is the worst case complexity of sorting using
 - i. selection sort **$O(n^2)$**
 - ii. tree sort **$O(n^2)$**
 - iii. heap sort **$O(n \log(n))$**
 - iv. radix sort **$O(d(n+k))$ for keys = d components from 1..k**
- h. Can the following sort be performed in a stable manner?
 - i. bubble sort **No**
 - ii. quick sort **No (stable version would require extra memory)**
 - iii. counting sort **Yes**
- i. Can the following sort be performed using an in-place algorithm?
 - i. selection sort **Yes**
 - ii. tree sort **No**
 - iii. merge sort **No**

X. Algorithm strategies

- a. What is divide-and-conquer?
Type of algorithm that divides problem into smaller problems of the same type and recursively solves them, combining their solutions for overall solution
- b. What is dynamic programming?

Type of algorithm that divides problem into smaller overlapping problems of the same type and recursively solves them, remembering previous solutions to subproblems.

- c. What is the difference between divide-and-conquer and dynamic programming?
Dynamic programming remembers solutions to subproblems, divide-and-conquer resolves subproblems
- d. What is the difference between recursive and backtracking algorithms?
Backtracking algorithms may use recursion to implement backtracking
- e. What is the difference between a greedy algorithm and heuristics?
Greedy algorithms achieve optimal solutions by selecting best (local) choice at each step. Heuristics use rules of thumb (such as greedy) to make a choice at each step, but are not guaranteed to achieve the best solution
- f. What is the difference between brute force and branch-and-bound algorithms?
Branch-and-bound algorithms attempt to reduce searching by eliminating partial solutions that are already worse than the best current solution found
- g. List a reason to use dynamic programming
Improve efficiency by avoiding repeating work
- h. List a reason to use backtracking
Simple approach to searching for all possible solutions
- i. List a reason to use a brute force algorithm
Need optimal solution but more efficient algorithms not known
- j. What type of algorithm is Kruskal's algorithm for finding minimum spanning trees?
Greedy

Problem 11 Design Patterns

Y. Design patterns

- a. What is a design pattern?
Descriptions of reusable solutions to common software design problems
- b. How were design patterns discovered?
Programmers found certain problems were solved repeatedly
- c. When are design patterns used?
When common software design problems are encountered that have been previously solved
- d. List 5 components of a design pattern
- e. List 3 types of design patterns. Give 2 example patterns for each type.
- f. What type of design pattern is the factory pattern? Visitor pattern?
- g. Write a Java example of the Singleton pattern
- h. Write a Java example of the Factory pattern
- i. Write a Java example of the Visitor pattern.
- j. List 2 examples of design patterns used in the Java class libraries
Iterator, Decorator, Marker, Observer
- k. Given the following code, complete the code for a BoatFactory class so it can be used to create big and small boat objects:

```
public interface Boat {
```

```

    int maxCapacity;
    int topSpeed()
}

class CruiseShip implements Boat { // big boat
    int topSpeed() { return 20; }
}

class SpeedBoat implements Boat { // small boat
    int topSpeed() { return 40; }
}

Boat myBigBoat = BoatFactory.create("big");
Boat mySmallBoat = BoatFactory.create("small");

public class BoatFactory {
    static Boat create(String s) {
        // your code here
        if (s.equals("big")) return new CruiseShip();
        else if (s.equals("small")) return new SpeedBoat();
        else return null; // error
    }
}

```

1. Using the same code, use the Decorator design pattern to
 - i. Add a BoatDecorator class implementing the Boat interface

```

public class BoatDecorator implements Boat {
    Boat b;
    BoatDecorator (Boat b) { this.b = b; }
    int topSpeed() { return b.topSpeed(); }
}

```

- ii. Create two BoatDecorators withBarnacle() and withTurboEngine() that change the result returned by topSpeed() by -1 and +10, respectively

```

public class withBarnacle() extends BoatDecorator {
    int topSpeed() { return b.topSpeed() - 1; }
}
public class withTurboEngine() extends BoatDecorator {
    int topSpeed() { return b.topSpeed() + 10; }
}

```

- iii. Use BoatDecorators to create a Boat object for a SpeedBoat with 2 barnacles and 1 turbo engine whose topSpeed() method returns 48.

```

Boat myBoat = new withBarnacle( new withBarnacle(
    new withTurboEngine( new SpeedBoat() ) ) );

```

Problem 12 Effective Java

Z. Effective Java

- Give 3 examples of possibly confusing Java features.
hashCode() contract, overloading + operator, ambiguous overloading methods, floats for decimals, private subclass fields
- Write an example of potentially confusing Java code.
- Name 2 approaches to Java programming styles that avoids confusing Java features.
avoiding ambiguous overloading methods, using ints decimals, avoiding public fields
- Give an example of potentially confusing Java code, and how to avoid it

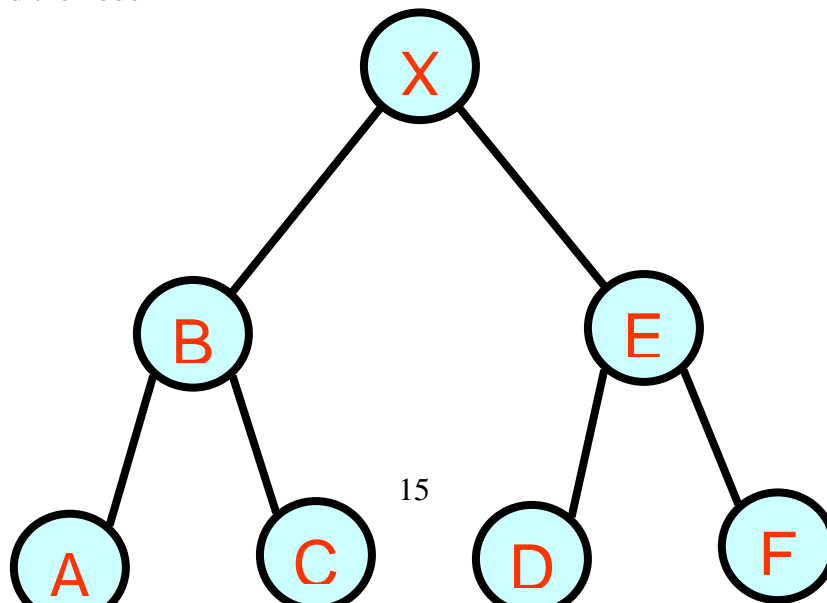
Problem 13 Advanced Tree Structures (Honors Section Only)

AA. Indexed search trees

- What is the motivation for using an indexed search tree (trie)?
Fast searches for keys that can be decomposed (with redundancy)
- What is a compressed trie?
Multiple connecting single edge chains are compressed into single edge
- What is a compact trie?
Nodes store indices into original text, rather than key
- What is a suffix trie?
A trie of all suffixes of each key
- Draw the suffix trie for the string “google”

BB. (4 pts) Balanced search trees

- What is the motivation for using balanced search trees?
Avoid worst case $O(n)$ behavior for find/insert/delete operations
- Name two algorithms for maintaining balanced search trees
AVL & Red-Black
- What is the mechanism used to balance search trees?
Rotation
- Given the following binary search tree, draw the tree resulting from performing a single right rotation around the node X
- For the same search tree, draw the tree resulting from performing a single left rotation around the node X



CC. (4 pts) Multi-way search trees

a. What is the motivation for using multi-way search trees?

Reduces number of nodes accessed (and hopefully disk accesses)

b. What is a 2-3 tree?

Multi-way tree where nodes have 2 or 3 children

c. Describe the algorithm for finding items in a 2-3 tree

If node has 1 key, same as binary tree. If node has 2 keys, search left subtree if value < smaller key, right subtree if value > larger key, else search middle subtree

d. Given the following 2-3 tree, draw the tree resulting from inserting the value 11.

e. Given the same 2-3 tree, draw the tree resulting from inserting the value 22.

