COMP7505 Group Project & Presentation

Presentation Dates: Thu 10 Oct - Thu 17 Oct - Thu 24 Oct

Submit Slides By: Thu 10 Oct 8:00am

Overview

Algorithms and data structures are crucial to ensuring a large-scale software system is efficient and reliable. Hence, many software companies (such as Google, Amazon, or Atlassian) tend to require its employees to have a very strong understanding of how to use algorithms and data structures to solve complex problems.

For this assessment item, you will be required to present the solution to a question similar to one that might be given in a technical interview at a large software company OR perform research into an algorithm/data structure that is currently a popular research topic. These presentations will be completed in groups of three during the week 11 to 13 lectures.

Presentation Structure

There is no required structure of your presentation, and you can include whatever sections you see necessary. However, it is strongly recommended that the sections described below are included in the presentation.

Interview style questions should include the following sections as a minimum:

- **Introduction:** Briefly introduce the problem that you are solving. This does not need to be very in-depth but should provide the audience with any information they need in order to understand the rest of the presentation.
- **Solution:** Explain your team's solution to the problem. This solution should be as efficient as possible and make use of relevant algorithms or data structures taught in this course. You should present the solution's pseudocode or Java code and provide an overview of how your team arrived at the solution presented.
- **Evaluation:** Explain why your solution is optimal for solving the problem by comparing your solution against alternative ones (eg. a brute-force solution).

Research questions should include the following sections as a minimum:

- **Introduction:** Briefly introduce the research scenario and the problem(s) the scenario is trying to solve. This section should also cover any background material the audience may need to know to understand the problem.
- **Research Results:** Give a detailed description of how the data structure/algorithm you are researching operates.
- **Evaluation:** Analyse the efficiency and usefulness of the data structure/algorithm you are researching. How can the data structure/algorithm be applied in practice?

Details and Constraints

You should consider the following details/constraints when presenting:

- The presentation's content should build upon the material presented so far in COMP3506/7505, and should be able to be understood by any other student in the course. There is no need to explain any content that has already been taught in the course.
- You should make use of visual aids (eg. a powerpoint) to help your audience follow your presentation. (Tip: try to minimize the amount of text on each slide).
- The solutions to interview-style questions should be presented in either pseudocode or Java code. Pseudocode or Java code snippets can also be shown to help explain a research topic. Your solution should be readable and well-styled regardless of what you write it in. You must include this code in your presentation slides.
- The first slide of your presentation should include the name/number of your topic, and the names and student numbers of your group members in the order they will present and the group member's contribution.
- You should present the material without a script, palm cards, or by reading directly from the slides.
- The presentation should last 5-6 minutes. Each group member should talk for a roughly equal amount of time (1-2 minutes). You will be cut off if your presentation lasts longer than 6 minutes and your mark will be determined from the material presented by that time.
- There will be 1-2 minutes for questions at the end of each presentation.
- You must make the presentation in-person and may not show any videos as a part of your presentation.

Slides Submission and Extensions

You must submit your team's slides in `pptx` or `pdf` format by 8am on Thursday the 10th of October, regardless of when you are actually presenting. Failure to do so by this time will result in your team being denied the usage of visual aids during the presentation.

You must be ready to present at 8am on the day of your presentation. If you are not ready to present when called upon, you will receive no marks for this assessment task. Due to this being a piece of group assessment, extensions are not possible.

Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. If you are doing an interview-style question, your entire solution should be your own work. If you are doing a research-style question, you must cite any sources you use appropriately (IEEE referencing style is recommended). See the course profile and the School web page for more information:
`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.

Topic Selection and Team Formation

You are responsible for forming your own teams. Each team must have exactly three students, and **you may not sign up for a topic unless you have a team of exactly three students**. You are encouraged to use the course Piazza page to search for team members. If you have not formed a team by the Friday of week 9, you will be randomly allocated a team and a topic. Each team will be randomly allocated one of the three lecture timeslots to present in. It is expected that you are available for the entire duration of all three timeslots.

There are 50 topics in total (42 interview-style questions and 8 research questions), outlined across the following pages. You are required to select one of these topics to present on. **Each topic can only be presented by one team. Topic allocation will be on a first-come first-served basis.** You will need to have a strong understanding of the algorithms and data structures taught over the span of this entire course to tackle these problems. By the end of week 10 you should be comfortable completing any of the problems below. Not all problems are equal difficulty, but this should be accounted for during marking (eg. if you pick an easier problem there is a much higher expectation that your answer is optimal). You will have the opportunity to make clarifications about these problems during the week 10 and 11 tutorial sessions. Note that many of the interview-style questions are not "real-world" problems and the scenarios given are simply designed for the question.

Sign-up for a team and a topic by the 20th of September via `Blackboard > Wikis > Presentation Topic Selection`.

Here are the problems available for you to choose for your presentation. Problems 1-42 are interview-style questions. Problems 43-50 are research-based topics.

1 A Re-mark-able Problem

Mahsa has an array, A, containing students and their final mark for COMP7505. Students are marked out of 100. Mahsa wants to sort the list in descending order of marks. If two students have the same mark, the first in the original array should come first in the returned array.

Example:

```
Input:  [["Jeff", 65], ["John", 77], ["Jim", 65]]
Output:  [["John", 77], ["Jeff", 65], ["Jim", 65]]
Explanation:  John has the highest mark so he should come first.  Jeff comes before Jim because
he is listed first in the original array.
```

2 Prerequisite Courses

You are given an array, A, containing pairs of courses. The pair $[x, y]$ indicates that x is a prerequisite for the course y. For example, the list `[["CSSE1001", "CSSE2002"], ["CSSE2002", "COMP3506"]]` indicates that CSSE1001 should be completed before CSSE2002 which should be completed before COMP3506. Write an algorithm that takes A and determines if it is possible to complete all courses while satisfying all prerequisites. Assume that only one course can be completed at a time and all courses are available every semester.

Example:

```
Input:  A = [["CSSE1001", "CSSE2002"], ["CSSE2002", "CSSE1001"]]
Output:  false
Explanation:  It is impossible to finish both courses, since you must take CSSE2002 before
CSSE1001, and vice versa.
```

3 LaTeX Compilation

Henry is struggling to compile the latest COMP7505 assignment spec in LaTeX. Environments are a LaTeX feature which allow special formatting to be applied to blocks of text. Valid environments begin with `\begin{command}` and end with `\end{command}`, where `command` could be any string but both values of `command` should be the same (assume any values for this command are valid). For example, the following is a valid LaTeX environment:

“`\begin{document} some text \end{document}`”

While the following are not valid LaTeX environments:

“`\begin{document} some text`”

“`\begin{document} some text \end{itemize}`”

Environments may be nested. For example, the following is valid:

“`\begin{document} \begin{center} some text \end{center} \end{document}`”

The following is not valid (the most recently opened environment should be closed first):

“`\begin{document} \begin{center} some text \end{document} \end{center}`”

You are to write an algorithm that takes a large section of LaTeX text (S), and determines if all environments are valid.

Example:

```
Input:  S = "some text \end{document}"
Output:  false
Explanation:  The end tag has no associated begin tag
```

4 Treasure Hunt

Arr, Henry the pirate has arrived on a deserted island looking for treasure! Your goal is to help Henry by writing an algorithm to determine the shortest path to find the treasure (in terms of the number of hops). Your algorithm should take a multi-line string representation of the island (S). Each character in the string is one of the following:

- 'H' is Henry's starting location
- 'X' is the location of the treasure
- 'W' is a stream of water which can't be crossed
- 'L' is an empty space of land

Henry can make one hop left, right, down, or up at a time (no diagonal movements are allowed) and is unable to cross water. Your algorithm should return the length of the shortest path, or -1 if no path exists.

Example:

```
Input:  S = "WWWWW
             WLLLX
             WLWLW
             WLLLH
             WWWWW"
Output:  4
Explanation:  It takes 4 hops to go from 'H' to 'X'
```

5 Landlubber

Henry the pirate has a map of the ocean. Your goal is to write an algorithm to count the number of islands on the map. Your algorithm should take a multi-line string representation of the map (S). Each character in the string is one of the following:

- 'W' represents water
- 'L' represents land

Adjacent land squares (in the left/right/up/down direction) form a single island.

Example:

```
Input:  S = "LLWL
             LLWL
             WWLL
             WLWW"
Output:  3
Explanation:  There are three islands of connected 'L's
```

6 A Cra-sea Problem

Henry the pirate has the same map as described in the previous problem. This time he wants to determine which of the islands has the largest area. Your algorithm should take a multi-line string representation of the map (as in the previous problem), and return the size of the largest island. If multiple largest islands exist you can return any of them.

Example:

```
Input:  S = "LLWL
             LLWL
             WWLL
             WLWW"
Output:  4
Explanation:  There are two islands of size 4, and one of size 1
```

7 A Farm Down Udder

James is on his hobby farm in the Queensland outback and has a large plot of land divided into smaller subplots in a straight line. He has a group of farm animals to put into these subplots but knows they will not cooperate if two animals of the same species are in adjacent subplots.

Write an algorithm that takes an array of animals (A) and returns an array of these animals where no two animals of the same species are adjacent to eachother. Assume that a valid answer always exists.

Example:

```
Input:  A = ["Sheep", "Cow", "Cow", "Sheep", "Sheep", "Cow"]
Output:  ["Sheep", "Cow", "Sheep", "Cow", "Sheep", "Cow"]
Explanation:  No sheep are adjacent to eachother, nor are any cows.
```

8 The Taskmaster

James is setting up a system to help keep track of tasks he needs to complete on the farm. Each task has a description and an urgency (the higher the more urgent). The maximum urgency possible is 100 and the minimum urgency possible is 0. The system should have the following methods:

- `add(description, urgency)` : add a new task with the given urgency
- `complete()` : remove and return the description of the task with the highest urgency. If multiple tasks have equal urgency they should be returned in the order they were added. You can assume this method will only be called when there are tasks to be completed.

Example:

```
T = TaskSystem()
T.add("Feed the chickens", 5)
T.add("Milk the cows", 10)
T.complete() -> "Milk the cows"
T.add("Harvest the crops", 5)
T.complete() -> "Feed the chickens"
T.complete() -> "Harvest the crops"
```

9 Leonardo DiCowprio's Great Adventure

Leonardo DiCowprio the cow is inside a large plot of land. His plot of land has width and height n. There is a water trough at position $p = (a, b)$ (where a is the row and b is the column) that Leo cannot pass through. Leo starts at position (0, 0). He can make only four movements: Left (`L`), Right (`R`), Up (`U`), or Down (`D`).

You are to write an algorithm that takes the values of n, p, and an integer k. Find all paths that Leo can take that end in a dead end after exactly k movements. Leo cannot visit the same position more than once. Your algorithm should return an array of paths (strings containing the characters L, R, U and D).

Example:

```
Input:  n = 4, p = [1, 1], k = 5
Output:  ["LLDLU", "DDRDL"]
Explanation:  The plot of land looks as follows ('E' is an empty cell, 'T' is the location of
the trough, 'L' is Leo's starting location):
LEEE
ETEE
EEEE
EEEE
There are only two paths of length 5 that result in a dead end
```

10 Terrafarming

A group of planets are arranged in a straight line. The animals of James' farm have flown to space and have arrived on these planets (one animal species per planet) to try to terraform these into a new home.

You are given an array, A, containing the species currently occupying each planet. You are to write an algorithm that finds the largest distance between a planet and the next planet occupied by the same species.

Example:

```
Input:  A = ["Cow", "Sheep", "Pig", "Cow", "Chicken", "Sheep", "Cow"]
Output:  4
Explanation:  Pigs and Chickens only occupy a single planet each.  The distance between each
pair of Cow planets is three.  The distance from the first Sheep planet to the second Sheep
planet is four, which is the largest distance.
```

11 Destroyed Asteroid

An asteroid field in deep space is represented as an array of integers, A. Each element in the array represents an asteroid, as follows:

- The absolute value of each integer represents the mass of each asteroid
- Positive asteroids move right, negative asteroids move left
- Each asteroid moves at the same speed

When two asteroids collide, the smaller asteroid is destroyed (the one with the lesser weight). If both asteroids have equal weight they are both destroyed.

You are to write an algorithm that takes an asteroid field and determines the state of the field after all collisions have occurred.

Example:

```
Input:  A = [10, 2, -5]
Output:  [10]
Explanation:  Firstly, asteroids 2 and -5 collide (2 is destroyed as it is smaller).  Then 10
and -5 collide (-5 is destroyed as it is smaller).
```

12 Satellite Links

A group of satellites are arranged in a straight line. Each satellite has a communication frequency. Each satellite can only send messages to satellites on its right, and only if the communication frequency of the receiving end is strictly greater than the communication frequency of the sending end.

Design an algorithm that takes an array (A) representing the communication frequency of each satellite (and the order they are arranged in). That is, the element $A[i]$ stores the communication frequency of satellite i. The algorithm should return an array of the same size containing the number of other satellites each satellite can send messages to.

Example:

```
Input:  A = [4, 6, 3, 9, 7, 10]
Output:  [4, 3, 3, 1, 1, 0]
Explanation:  The first satellite has four satellites to its right with a greater communication
frequency (6, 9, 7, 10).  The second satellite has three satellites to its right with greater
communication frequencies (9, 7, 10).  This carries on for each element in the array.  Note
that the last element will always be 0 because the last satellite will never have others to its
right.
```

13 In Plane Sight

Emily is going on holiday and has a bunch of plane tickets. Unfortunately, she does not remember which order they need to be used.

You are to design an algorithm which takes an array, A, of plane tickets. Each ticket is represented as a pair of strings representing the departure and arrival airports. For example, the pair `["BNE", "SYD"]` represents a ticket from Brisbane to Sydney. Your algorithm should sort the tickets in the order they should be used. If there are multiple solutions you can return any of then. Assume Emily starts in Brisbane (BNE) and the tickets can be redeemed on any day or time.

Example:

```
Input:  A = [["SIN", "LHR"], ["BNE", "SIN"], ["SFO", "LAX"], ["LHR", "SFO"]]
Output:  ["BNE", "SIN", "LHR", "SFO", "LAX"]
Explanation:  The tickets should be used in the following order (indices of tickets are given):
1, 0, 3, 2
```

14 Flight of Passage

Emily still has her plane tickets (as per the previous problem) and wants to know how many return ticket pairs she owns. A pair of tickets $[a, b]$ and $[c, d]$ count as a return ticket pair if $a = d$ and $c = b$. Design an algorithm that takes an array of tickets and counts the number of return ticket pairs. A single ticket cannot be a part of multiple return ticket pairs, but it is possible for Emily to own duplicate tickets.

Example:

```
Input:  A = [["BNE", "SYD"], ["BNE", "SYD"], ["SYD", "BNE"], ["BNE", "MEL"]]
Output:  1
Explanation:  There is only one return ticket pair.  There is also an extra ["BNE", "SYD"]
ticket and a ["BNE", "MEL"] ticket with no return ticket
```

15 The Airspace Less Travelled

Emily is recording the airports she visits on her trip. Given an array, A, of airports that have been visited, find the top k most frequently visited airports. The answer should be in sorted order of frequency (the most frequent first).

Example:

```
Input:  A = ["BNE", "SYD", "BNE", "MEL", "SYD", "BNE", "SIN"], k = 2
Output:  ["BNE", "SYD"]
Explanation:  BNE has been visited three times.  SYD has been visited twice.  All other
airports have been visited once.
```

16 The Magic Method

Alex has come across an ancient magical language that does not follow the same alphabetical ordering conventions as the English language. She has a book of words in this language and knows that these words are in alphabetical order, but does not know the ordering of the letters themselves.

Write an algorithm that takes an ordered array of words in the language (A) and returns an ordered array of the letters in the language.

Example:

```
Input:  A = ["caa", "aaa", "aab"]
Output:  [‘c’, ‘a’, ‘b’]
Explanation:  It is clear that ‘c’ is the first letter in the language as the word starting
with ‘c’ comes before the other words.  Since "aaa" comes before "aab" it can also be derived
that ‘a’ comes before ‘b’.
```

17 Alex and her Axle-ent Anagrams

Alex still has her magical book (as above), and is interested in finding anagrams of various words in the book. Given a string (S) of space separated words, group any anagrams in the string together. You should return an array of arrays.

Example:

```
Input:  S = "dads can braid rabid dogs while Alex adds gods"
Output:  [["dads", "adds"], ["braid", "rabid"] ["dogs", "gods"]]
Explanation:  "dads" is an anagram of "adds", "braid" of "rabid", and "dogs" of "gods".
```

18 Accio

Searching through her magic book (as above) is a slow process for Alex. She wants to build a database that allows her to search for words in the book. The database should have the following operations:

- `save(word)` : adds a word to the database. This word can only be made up of lowercase letters.
- `accio(word)` : returns whether the given word is in the data structure or not. Each character in the `word` parameter can be any lowercase letter OR the ‘.’ character, which represents any letter.

Example:

```
D = Database()
D.save("hello")
D.accio("hello") -> true
D.accio("h..lo") -> true
D.accio("he..lo") -> false
```

19 Zombie Outbreak!

There is a zombie outbreak at UQ! Everyone passes by catastrophe, but all students have to go to a quarantine area. Unfortunately, some of the students in the quarantine area are already infected with the zombie virus. The quarantine area is represented as a string (S) containing multiple lines.

- 'E' represents an empty cell with no humans or zombies
- 'H' represents a cell containing a human
- 'Z' contains a cell containing a zombie

Each day, the zombies infect all humans next to them (up, down, left or right), turning them into zombies. Write an algorithm that takes S and finds the number of days until all humans have been turned into zombies. If any human is guaranteed to survive, return -1.

Example:

```
Input:  S = "ZHEZH
             HEHZH
             HEEEE"
Output:  2
Explanation:  After one day all humans in the first two rows are zombies.  After another day
the human in the bottom left is also a zombie.
```

20 Brain Drain

Tom the zombie is in a neighbourhood with many houses full of human occupants. His goal is to eat all the humans' brains, but wants to savour the experience by going as slowly as possible.

The neighbourhood is represented as an array, A, of length n. The following rules are in place:

- Houses in the neighbourhood are numbered from 0 to $n-1$
- $A[i]$ represents the number of humans in house number i
- Each human has exactly one brain
- Tom's eating speed is k brains per hour - each hour, tom breaks into one house and eats k humans' brains
- If a house has less than k occupants, Tom eats all the brains in that house but cannot move to another house until the next hour
- Tom's maximum eating speed is 10^9 brains per hour
- Tom needs to eat all the brains within h hours before the sun rises and he is disintegrated (Tom lives in Antarctica, so the length of a single night could be huge)

You are to write an algorithm that takes A and h as parameters and calculates the minimum value of k that allows Tom to eat all the brains in the neighbourhood within h hours.

Example:

```
Input:  A = [3, 6, 7, 11], h = 8
Output:  4
Explanation:  Tom eats all the brains in house 0 in one hour.  Tom takes two hours to eat all
the brains in houses 1 and 2.  Tom takes three hours to eat all the brains in house 3.  This
takes 8 hours in total, so an eating speed of 4 brains per hour is appropriate.
```

21 Dungeon Escape

Paul is locked in a medieval dungeon. There are n cells, each numbered from 0 to $n - 1$. Each cell contains one or more keys which can open other cells. Paul starts in cell 0, which the guards have accidentally left unlocked. Cells can only be unlocked from the outside.

You are given an array, A, representing the keys in each cell. $A[i]$ is a (possibly empty) subarray of the keys in cell i, and $A[i][j]$ is a single integer representing the room key j can unlock. Paul is able to walk between unlocked cells freely and can do so in any order. Write an algorithm that determines if it is possible to free all prisoners by unlocking all the cells.

Example:

```
Input:  A = [[1, 3], [3, 0, 1], [2], [0]]
Output:  false
Explanation:  There is no way to enter room 2
```

22 Long Live the King!

The King is dead! Long live the King! Unfortunately, there is no-one to take his place. The medieval kingdom is required to vote for the next king. Given an array of strings representing votes for people, find the person with the most votes - that person will become the next king! If a tie exists, the person who reaches the most votes first should become King.

Example:

```
Input:  A = ["Henry", "Henry", "Tom", "James", "James", "Tom", "James", "Henry"]
Output:  "James"
Explanation:  Henry and James both receive 3 votes, but James gets to 3 votes first so he is
crowned King
```

23 Union of Linked Lists

Given two singly-linked lists, $L1$ and $L2$, find the union of the two linked lists. The order of the elements in the output list does not matter but duplicate elements should not be present. You may assume the elements of the linked lists are comparable but the elements are not guaranteed to be hashable.

Example:

```
Input:  L1 = 10->15->4->20, L2 = 8->4->2->10
Output:  2->8->20->40->15->10
Explanation:  Each element in both L1 and L2 are in the output.  4 and 10 are only present once
in the output.
```

24 Intersection of Linked Lists

This is the same as the previous problem, however you are now to find the intersection of the two linked lists. Again, the order of the elements in the output does not matter but duplicate elements should not be present. Elements are guaranteed to be comparable but not hashable.

Example:

```
Input:  L1 = 10->15->4->20, L2 = 8->4->2->10
Output:  4->10
Explanation:  4 and 10 are the only elements in both lists
```

25 Newspaper

You are given:

- An array of strings, A
- A string, S

Write an algorithm that determines if S can be formed by concatenating any two strings in A.

Example:

```
Input:  A = ["news", "abcd", "tree", "paper"], S = "newspaper"
Output:  true
Explanation:  The words "news" and "paper" make up "newspaper"
```

26 Paper News

You are given:

- An array of strings, A
- A string, S

Write an algorithm that determines if S can be space-segmented into strings only in A.

Example:

```
Input:  A = ["news", "paper", "yes", "no"], S = "papernews"
Output:  true
Explanation:  S can be segmented as "paper news"
```

27 Footpath

A council is planning to build a footpath along the side of a road. Unfortunately some parts of the footpath are duplicated in the plans. Given an array (A) of intervals representing areas to fill in with concrete, return an array with any overlapping intervals merged.

Example:

```
Input:  A = [[0, 5], [10, 20], [5, 7], [15, 30]]
Output:  [0, 7], [10, 30]
Explanation:  [0, 5] and [5, 7] have an overlapping element (5).  [10, 20] and [15, 30] have
several overlapping elements (15 to 20).
```

28 Stacks of Fun

Design a data structure that operates in a similar fashion to a stack of integers and has the following functions:

- `push(v)` : pushes an element to the top of the stack
- `pop()` : pops and returns the element at the top of the stack
- `getMin()` : returns the element on the stack with the smallest value

Example:

```
S = Stack()
S.push(2)
S.push(5)
S.push(1)
S.getMin() -> 1
S.pop() -> 1
S.getMin() -> 2
```

29 A Sort of Problem

Given an array of unsorted integers, A, find the smallest subarray to sort which will result in the entire array being sorted. Return the start and end indexes of that subarray.

Example:

```
Input:  A = [1, 2, 3, 7, 5, 6, 4, 8]
Output:  [3, 6]
Explanation:  Sorting the array from index 3 to 6 will sort the entire array
```

30 Another Sort of Problem

Given a string, S, sort the characters of the string in decreasing order of frequency.

Example:

```
Input:  S = "tree"
Output:  "eert"
‘e’ appears twice so it appears first in the output.  "eetr" is also a valid output.
```

31 Minimising Problems

Given a value k and a non-negative integer represented as a string, S, remove any k digits from S to make the integer as small as possible. You can assume the integer won't be prepended with zeroes.

Example:

```
Input:  n = "1432219", k = 3
Output:  "1219"
Explanation:  Remove digits 4, 3, and 2
```

32 Misalignment

Given a string, S, determine each character's misalignment. A character's misalignment is the number of characters that come before it in the string but after the character alphabetically. Return an array, A, where the element $A[i]$ is the misalignment of the ith character in the string.

Example:

```
Input:  S = "bdacea"
Output:  [0, 0, 2, 1, 0, 4]
‘b’ and ‘d’ come alphabetically after ‘a’, so the misalignment of the first ‘a’ is 2.  ‘d’
comes alphabetically after ‘c’ so the misalignment of ‘c’ is 1.  All characters except for ‘a’
come alphabetically after ‘a’, so the misalignment of the last ‘a’ is 4
```

33 Longest Consecutive Subsequence

Given an array of integers (A), find the length of the longest subsequence, such that when the elements of that subsequence are ordered they are consecutive integers. The consecutive elements could be in any order in the original array.

Example:

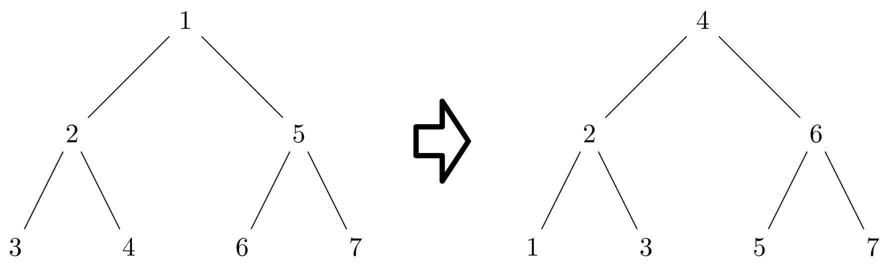
```
Input:  A = [1, 9, 3, 10, 4, 20, 2]
Output:  4
Explanation:  The longest subsequence of consecutive elements is [1, 3, 4, 2] which has length
4
```

Problems 34 to 37 take a reference to the root node of a binary tree as input. Each tree node has a reference to each child, but no node has a parent reference.

34 toBST

Convert the given binary tree to a binary search tree. The structure of the tree should not change - only the values of the elements.

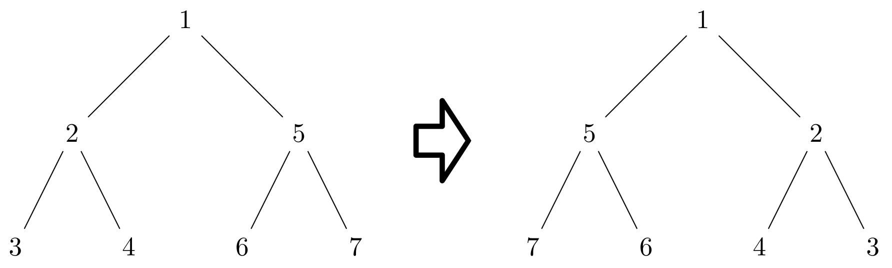
Example:



35 Inversion

Invert the input tree around its vertical axis.

Example:

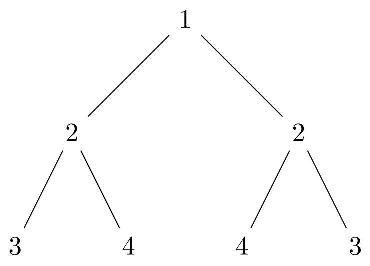


36 Symme-tree

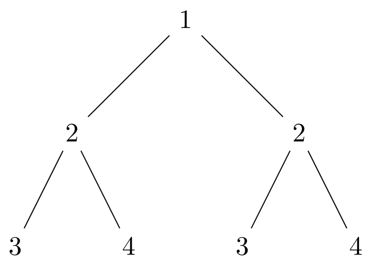
Determine if the input tree is symmetrical around its vertical axis.

Example:

This tree would yield true:



This tree would yield false:

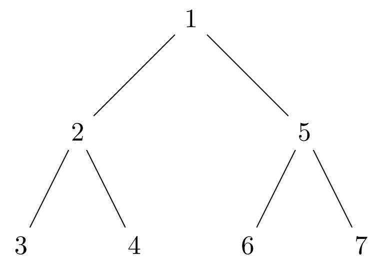


37 Tree Climbing

In addition to a root node reference, you have references to two other nodes in the same tree. Compute the shortest distance between those two nodes, in terms of the number of branches between those nodes.

Example:

Assume we have references to nodes 3 and 6 in the following tree:



The distance between these nodes is 4 (the shortest path is `3 -> 2 -> 1 -> 5 -> 6`)

Problems 38 to 40 take a graph as input. Each graph has n nodes (numbered from 0 to $n-1$) and an array of edges, E. Each element in the array is a pair, $[a, b]$, representing an undirected and unweighted edge from a to b.

38 Forest Adventure

Given a forest of trees, find the number of trees in that forest.

```
Input:  n = 5, E = [[0, 1], [0, 2], [3, 4]]
Output:  2
Explanation:  There are two trees (the first contains 0, 1 and 2 and the second contains 3 and
4)
```

39 Planting Trees

Given a graph, determine if it is a tree or not.

Example:

```
Input:  n = 4, E = [[0, 3], [1, 3], [2, 3]]
Output:  true
Explanation:  This graph is a tree rooted at node 3
```

40 Red-Black Graph

Consider a new data structure, called a red-black graph. In this graph, the edge array (E) described above is split into two separate arrays, R and B. The R array contains edges coloured red, while the B array contains edges coloured black. An alternating path between any two nodes i and j is a path from i to j where no two consecutive edges in the path have the same colour. You are to write an algorithm that finds the length (in terms of the number of edges) of the shortest alternating path from node 0 to node $n-1$, or -1 if no such path exists.

Example:

```
Input:  n = 3, B = [[1, 2]], R = [[0, 1]]
Output:  2
Explanation:  The shortest path from node 0 to node 2 is 0 -> 1 -> 2, which has alternating
colours.
```

Problems 41 and 42 are distributed problems and are only recommended to those who have done or are doing CSSE7231, CSSE7610, or CSSE7014. You must consider the effects of concurrency on any data structures you pick.

41 Distributed File System

You are to design a distributed file system that has the following functions:

- `record(username, ip)` : store the username of the user alongside their IP address. Both of these are strings, but the username may contain special characters and therefore can't be hashed. A user may have multiple IP addresses, so this method may be called more than once with the same username.
- `request(username)` : return an array of all IP addresses associated with the given username, or `null` if that user is not stored

Example:

```
F = FileSystem()
Concurrent[F.record("hob", "10.0.0.1"), F.record("emily", "8.8.8.8" )]
F.record("hob", "10.0.0.2")
F.request("hob") -> ["10.0.0.1", "10.0.0.2"]
F.request("emily") -> ["8.8.8.8"]
```

42 Concurrent Testing Server

You are to write an application that can concurrently record testing results of various algorithms (which may be in shared memory). Test results can be represented as strings. The application should have the following functions:

- `add(id, test)` : record a new test in the system, which has an integer identifier
- `result(id)` : return the test result with the given identifier - this may be called in parallel to other `run` or `add` calls
- `allResults()` : return an array of all test results in order of the test identifiers

Example:

```
T = TestingServer()
T.add(1, "TEST1")
Concurrent[T.add(3, "TEST2"), T.add(4, "TEST4")]
T.add(2, "TEST3")
T.result(1) -> "TEST1"
T.allResults() -> ["TEST1", "TEST3", "TEST2", "TEST4"]
```

Research-based projects
(Courtesy: Keith Schwarz's 2019 Data Structures class, Stanford)

Students interested in research in the field of algorithms and data structures can consider exploring one of these data structures/algorithms. You must find appropriate sources to help you learn about these topics and cite them appropriately in the presentation.

43 Tango Trees

Is there a single best binary search tree for a set of data given a particular access pattern? We asked this question when exploring splay trees. Tango trees are a data structure that are at most $O(\log(\log n))$ times slower than the optimal BST for a set of data, even if that optimal BST is allowed to reshape itself between operations. By exploring tango trees, you'll get a much better feel for an active area of CS research and will learn a totally novel way of analyzing data structure efficiency. *Consider looking up the original paper on tango trees ("Dynamic Optimality – Almost").*

44 Purely Functional Red/Black Trees

In purely functional programming languages, it's not possible to modify data structures after they've been constructed. Any updates that happen must be performed by building a new version of the data structure, possibly sharing some existing pieces of the original structure in a read-only way. Although red/black trees are tricky to implement in imperative languages, it's possible to implement them in purely functional programming languages, maintaining all the existing time bounds, in about a page of code. This means that it's possible to implement red/black trees in a way that lets us "go back in time" to see what the data structure looked like in the past and to use them in programming languages and settings where updates aren't permitted. Purely functional red/black trees are a good first taste of what data structures look like in a purely functional setting.

Note: Having a reasonable grasp of functional programming is recommended for this question.

45 Min-Hashing

How do you determine how similar two documents are? That's a major question you'd need to answer if you were trying to build a search engine or indexing huge amounts of text. One technique for solving this problem involves techniques reminiscent of the cardinality estimation techniques: hash the contents of the documents with different hash functions and take the smallest value produced. Amazingly, these minimum hash values reveal a huge amount about the documents in question. Min-hashing was initially developed to solve practical, real-world problems, and it's one of those beautiful approaches that combines theoretical elegance and pragmatic utility.

46 Signature Sort

It's possible to sort in time $O(n \log n)$ if the items to sort are integers (for example, using radix sort). What are the limits of our ability to sort integers? Using advanced techniques, signature sort can sort integers in time $O(n)$ – assuming that the machine word size is $\Omega((logn)^{2+\epsilon})$.

Signature sort employs a number of clever techniques: using bitwise operations to perform multiple operations in parallel, using tries to sort integers as though they were strings on a small alphabet, etc.

47 Geometric Data Structures

Geometric data structures are designed for storing information in multiple dimensions. For example, you might want to store points in a plane or in 3D space, or perhaps the connections between vertices of a 3D solid. *Consider looking up: k-d trees.*

48 Parallel Data Structures

Traditional data structures assume a single-threaded execution model and break if multiple operations can be performed at once. (Just imagine how awful it would be if you tried to access a splay tree with multiple threads.) Can you design data structures that work safely in a parallel model – or, better yet, take maximum advantage of parallelism? In many cases, the answer is yes, but the data structures look nothing like their single-threaded counterparts. *Consider looking up: concurrent priority queues*

Note: Unlike problems 41 and 42, which require you to choose parallel data structures, this question is asking for a more detailed overview of how parallel data structures operate. Like problems 41 and 42, you should have a strong understanding of concurrency.

49 The Commentz-Walter String Matching Algorithm

The Commentz-Walter string matching algorithm is a generalization of the Boyer-Moore string searching algorithm, which (surprisingly) can match strings in sublinear time. Commentz-Walter is known to run in sublinear time on many inputs and to have a quadratic worst-case complexity.

50 Approximate Distance Oracles

Computing the shortest paths between all pairs of nodes in a graph can be done in time $O(n^3)$ using the Floyd-Warshall algorithm. What if you want to get the distances between many pairs of nodes, but not all of them? If you're willing to settle for an approximate answer, you can use subcubic preprocessing time to estimate distances in time $O(1)$.

Pathfinding is as important as ever, and the sizes of the data sets keeps increasing. Approximate distance oracles are one possible approach to try to build scalable pathfinding algorithms, though others exist as well.