A screenshot of a computer

Description automatically generated

So what are connected components in a graph  
  
The connected components in a graph are groups of nodes that are directly or indirectly connected to each other, but no connections exist between nodes in different groups.

Step to consider a question like this

1. Check for direct connections
2. Look for isolated nodes
3. Carefully trace edges between nodes to avoid missing any indirect connections

2.2

**How many nodes and leaves has a complete binary tree of height h = 10?**

To calculate   
Nodes Formula: (2^h) – 1 =(2^10) -1 = 1023 nodes  
Leaves Formula: (2^h– 1) = (2^10-1) = (2^9) leaves = 512 Leaves

**What is the minimum height of a binary tree that consists of 1000 nodes**  
Analysis:  
First we see what is give:  
 1. Nodes are given and we need to find the height, so basically the opposite of what we   
 did in the last question.  
 2. What we are trying to achieve is that we need to find the height using the   
 formula of the nodes.  
  
  
Two Ways:

1. Using Log
   1. Nodes Formula: (2^h) – 1  
      1000 = (2^h-1):   
      1001 = 2^h
   2. Using log, we can see that  
      log (Base - 2)(512) = 9  
      log (Base - 2)(1024) = 10  
        
      So from above we can conclude that having a height of 10 in the tree can give us a limit to store 1000 nodes
2. Using the Formula:  
   Lowest height results when tree is as complete as possible  
   A complete tree of height h has 2^h – 1 nodes.
   1. h = 9: maximum 2^9 -1 = 511 nodes, i.e. h = 9 is not sufficient to store 1000 nodes
   2. h = 10: maximum 2^10 -1 = 1023 nodes  
        
      Minimum height of tree should be 10 to store 1000 nodes

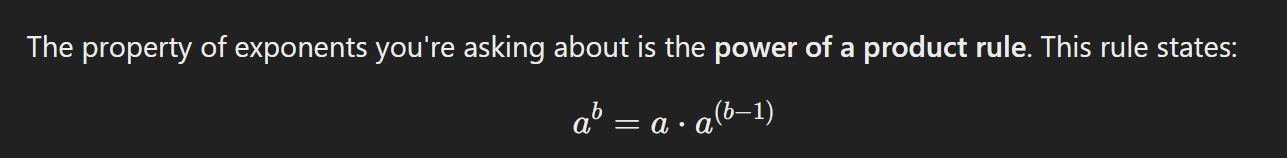
**Is the following statement true? “Every complete binary tree has more leaves than internal nodes” (give a short justification)  
  
Analysis:**

Internal Nodes = Total Nodes – Leaves  
Internal Nodes = (2^(h) -1) - (2^(h-1))

What the Prof. did here is that he asking to prove using those two formulas above True. Complete tree has,  
 2^(h-1) leaves,  
 2^h -1 nodes, hence  
 (2^h -1) – 2^(h-1) = 2\*2^(h-1) = 2^(h-1) – 1 internal node

So, a complete binary tree has always on leaf more than internal nodes

Keep in mind that there is an exponent property to simplify the expression in the calculations stated above



Exercise 2.3  
A diagram of a tree

Description automatically generated

1. **Steps for pre-order**
   1. Idea - MLR
      1. Answer to the question above: 124563
   2. Steps:
      1. Write the root nodes first. Continue Writing each root node until you reach the last one in the left subtree, why left subtree, as the next node to write will always be the left child of the current root.
      2. Traverse the entire left subtree.
      3. After the left subtree is fully traversed, move to and traverse the right subtree.
2. **Steps for In-order**
   1. Idea - LMR
      1. Answer to the question above: 426513
   2. Steps:
      1. Start by traversing the entire left subtree first, going as far left as possible.
      2. From the bottom of the tree, write the leftmost node first, then its root, and finally its right node. If there are multiple subtrees, you continue this process for each, always moving from bottom to top. The core idea remains always traverse left, then root, and then right.
3. **Steps for post-order**
   1. Idea - LRM
      1. Answer to the question above: 465231
   2. Steps:
      1. Start by traversing the entire left subtree first, going as far left as possible.
      2. From the bottom of the tree, write the leftmost node first, then the right node, and finally its root node. If there are multiple subtrees, you continue this process for each, always moving from bottom to top. The core idea remains always traverse left subtree first, then right subtree, and finally their root node.

Question: Where we need to construct a tree.  
  
A number of trees with black text

Description automatically generated

1. **Version 1 - Steps**
   1. **Using Post or Pre Order**
      1. **Find the Root node**
         1. In our case we write 6 first
      2. The Root Node that you selected from Pre/Post list (6 in our case) , Now check where it is in the “In order” list and you will see for the next number which number belongs to which sub tree.
         1. 1,4 nodes are part of left subtree of node 6
         2. 2,3,5 nodes are part of the right subtree of node 6
      3. **Now take the next number from the Pre/Post Order list (In our case 4)**
         1. (MAKE SURE IF YOU START WITH PRE OR POST ORDER LIST – THEN ONLY USE THAT SPECIFIC LIST, DONT INTERCHANGE THE LIST ALONG THE WAY).
      4. The number that you took which basically will be another root node for up comings nodes but sub node for the previous node that was taken in last steps should
         1. The idea in this step is to see (Node 4 in our case), where it is in the “In order” list and this number becomes the left or right sub node of the Root node above it
      5. **Repeat the steps till the point you reach the end of the Pre/Post list**
         1. With post-order to take the elements from the Left to Right from the list as the Root Node exists at the end
         2. With Pre-Order to take the elements from the Right to Left from the list as the Root Node exists at the end.
2. **Version 2 - Steps:**
   1. **Using post-order or Pre-order Traversal** i. **Find the root node**  
      - In our case, the root node is the first element of the **Pre-order** list: 6.
   2. **Check the position of the root node in the In-order list**
      1. Find where the root node (6) appears in the **In-order** list.
      2. Everything to the **left of 6** (1, 4) belongs to the **left subtree**.
      3. Everything to the **right of 6** (2, 3, 5) belongs to the **right subtree**.
   3. **Take the next number from the Pre/Post-orde**r list
      1. The next number in the Pre-order list is 4. -   
         Important: Once you start with Pre-order or post-order, stick to that list for the rest of the process (do not switch between lists).
   4. **Determine where the next number fits in the tree**
      1. The number you selected (4) is the root of a subtree. Check its position in the **In-order** list.
      2. In this case, 4 will be part of the **left subtree** of 6, as it is to the left of 6 in the **In-order** list
   5. **Repeat these steps for each node in the Pre/Post-order list**
      1. **Post-order**: Process elements from **left to right** in the list, since the root node will be at the **end** of the list.
      2. **Pre-order**: Process elements from **right to left**, since the root node will be at the **start** of the list.

Exercise 2.4

A diagram of a tree

Description automatically generated

Steps:

1. Stack-Machine Code is translated using the Post-Order Traversal
   1. Generated Code Key:
      1. Key
         1. Rule of Post-Order Traversal: LRM
         2. Syntax:
            1. For a number, write the term “const” before it
            2. For a variable, write the term “load “ before it
            3. For an operator just write the operator name

\*, stands for mult

-, stands for sub

+, stands for +

* 1. Steps
     1. Write the Post-Order Traversal of the tree, using the data in the nodes
        1. 2x\*y5+z\*-
     2. Now write the Generated Code using the Key stated above:
        1. Result
           1. Const 2
           2. Load x
           3. Mult
           4. Load y
           5. Const 5
           6. Add
           7. Load z
           8. Mult
           9. Sub

Exercise 2.5

A screenshot of a computer

Description automatically generated

Game Rules:

1. Player A starts by choosing either 1 or 2
2. Players alternate turns. After the first turn, they can choose from 1,2,3 but cannot pick the same number chosen by the opponent in the previous move
3. The goal is to reach a sum of exactly 6. The player who reaches 6 wins
4. If a players move cause the sum to exceed 6 they lose
5. **Once a player makes a choice, only the paths based on that choice are followed (invalidating other paths in the decision tree).**

Creating the Graph:

1. Start with root node.
   1. Nodes represent the sum of the numbers. So, root node will be 0
2. As stated at the start 1,2 will be the initial numbers that can be chosen, so create two sub trees, where now edges/lines need to have those number written down from the pool of numbers that are allowed to take on a turn
   1. So, at second step after 1.a. we write 1,2 but two different subtrees
3. Processed with the following logic, now those two subtrees will have their own subtrees, which means the values cannot be the same that were take one step previous.
   1. In this step make sure when you select the number as a node write the total sum of the number that you have been like with previous nodes. As you can see in the picture below at the step when the sum was 1, if one of the player for example 2 then the result will be the sum of the number at the previous node where we came from with the number that we know choose, which means 1+2 for example would result to a new node 3  
      A diagram of a triangle with numbers and lines

      Description automatically generated
   2. So the idea is you need to go down as much as possible until all the nodes either once exceed with a number more than 6 or the node 6 itself
      1. Mark nodes where the sum equals **6** as wins.
      2. Mark nodes where the sum exceeds **6** as losses.
4. On the right side you can write which players turns is it
   1. As stated in the question its players A turn at the start so we right in parallel of nodes which player turn is it and what possibilities they have ahead to select with number stated on top of the lines

A screenshot of a computer

Description automatically generated

Part a.

1. So in this part we are concatenating every string from L1 with every string from L2, so its not exactly like Cartesian Product but rather a concatenation of two languages
   1. What is concatenation
      1. The concatenation of two languages L1 ​ and L2 ​ is the set of all strings formed by taking a string from L1 ​ and appending a string from L2.
2. Steps:
   1. Concatenate ε from L1 with each string in L2:
      1. ε \* ε = εε = ε,
      2. ε\*ab=ab,
      3. ε\*bb=bb,
   2. Concatenate ab from L1 with each string in L2:
      1. ab\* ε=ab,
      2. ab\*ab=abab,
      3. ab\*bb=abbb,
   3. Result L1\*L2 = {ε,ab,bb,abab,abbb}

Part b.

1. The Kleene star operation (denoted as L∗ ) means that we take a language L and concatenate it with itself zero or more times.
2. Steps to solve:
   1. L1\*
      1. L1\*L1 = {ε,a,bb,aa,abb,bba,bbbb}
      2. L1\*L1\*L1 = {ε,a,bb,aa,abb,bba,bbbb,aaa,aabb,abba,abbbb,bbaa,bbabb,bbbba,bbbbbb}
      3. So, We can say
         1. L1\* = {ε, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba,… }
   2. L2\* = {ε, aa, aaaa, aaaaaa, ... }
      1. all strings consisting of an even number of a's
   3. L3\* = {ε}
   4. L4\*
      1. For an empty language you can show denote in this way while using the Kleene Star operation
      2. = L4(0) U L4(1) U L4(2) U ....  
          = {ε} U { } U { },  
          = {ε},  
         L(0) = {ε} for every language L

A close-up of a text

Description automatically generated

General Rules for Well-formed XML:

1. Unique Root Element:
   1. This means that XML document must have a single stand-alone root element that contains all other elements. No multiple root elements are allowed
2. Properly Nested Tags:
   1. Element must be properly nested. If an element opens inside another, it must close before the outer element closes
      1. For example:
         1. <a><b></b></a> <!-- Correct nesting -->
         2. <a><b></a></b> <!-- Incorrect nesting -->
3. Correct Tag Pairing:
   1. Every opening tag must have a corresponding closing tag in the correct order
      1. For example
         1. <tag>content</tag> <!-- Correct -->
         2. <tag>content</wrong> <!-- Incorrect -->
4. Not sure if the following are also part of our course work  
   A screenshot of a computer

   Description automatically generated