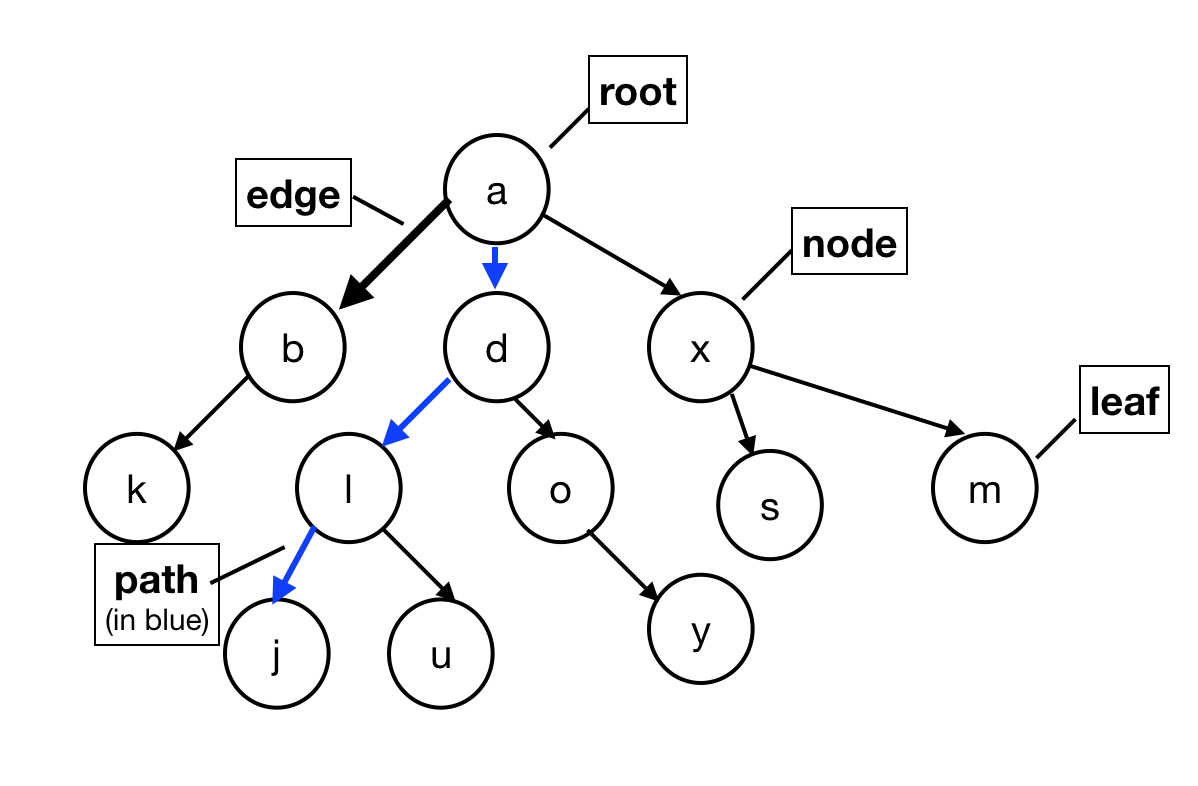
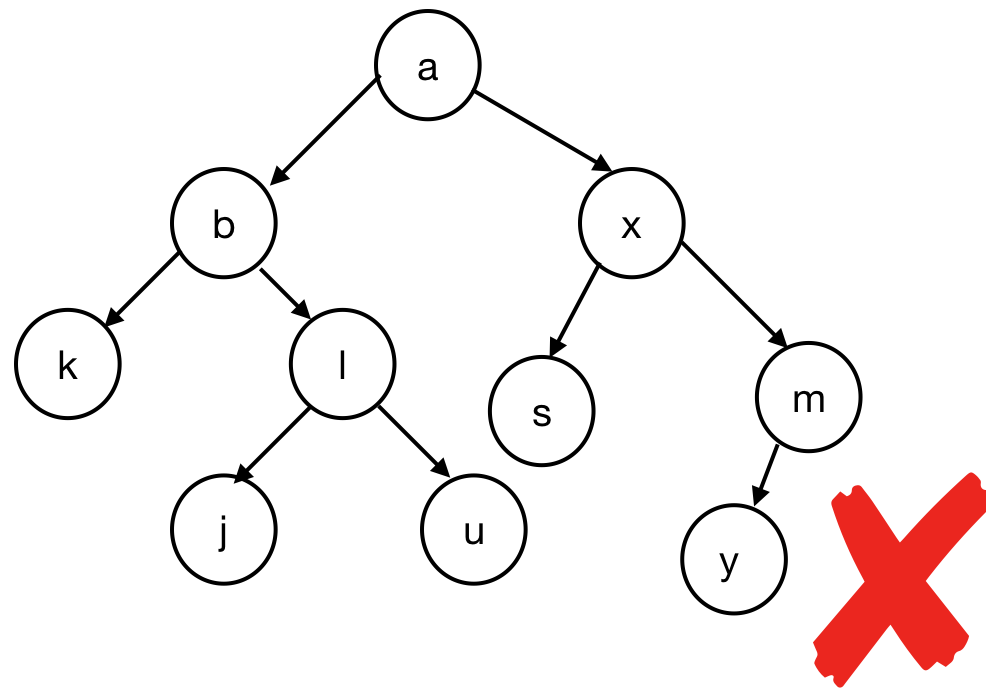
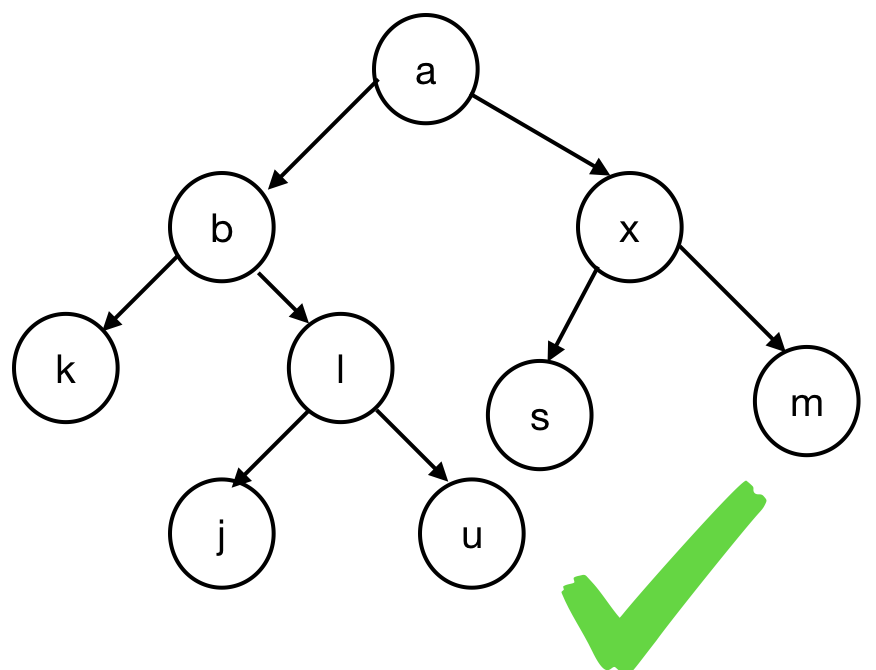
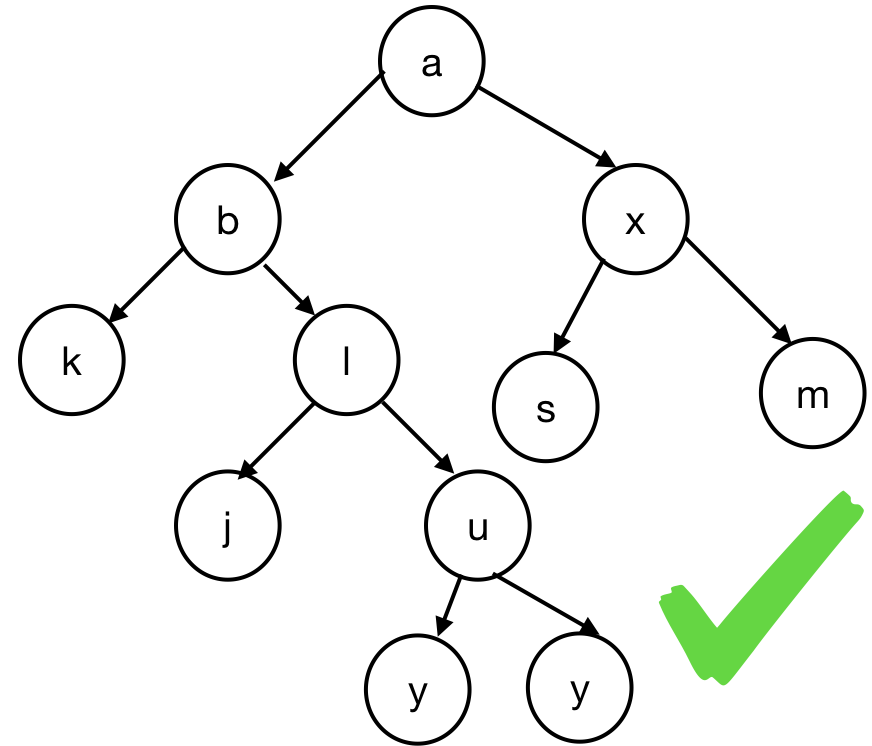
### ‘’ **Tree Terminology review**

* + Vertex: “nodes”
  + Edge: a connection between two vertices
  + Path: sequence of edges
  + Parents: Node **b, d, x** have Node **a** as their parent
  + Children: **b, d, x,** are the children of **a**
  + Siblings: **b, d, x,** are siblings of each other
  + Ancestors: **u** has ancestors **l, d, a**
  + Descendants: **x** has **s, m** as its descendants
  + Leaves: Vertices with no children

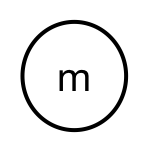
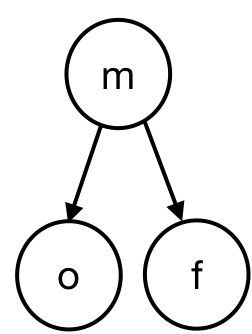
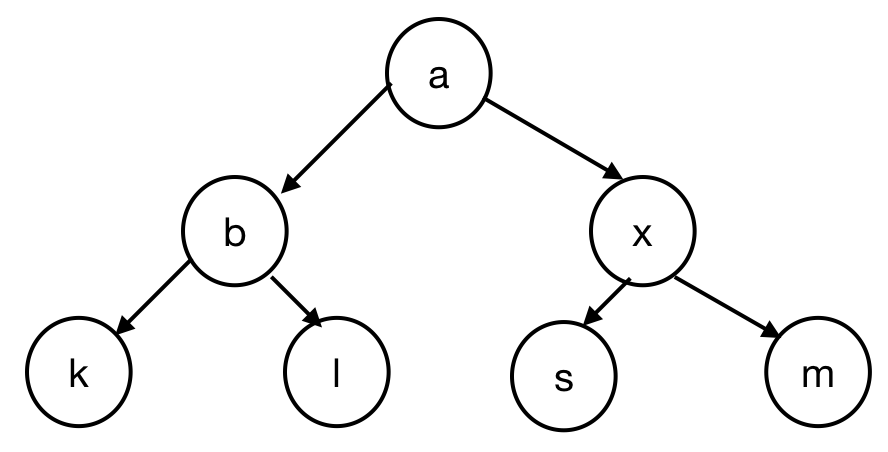


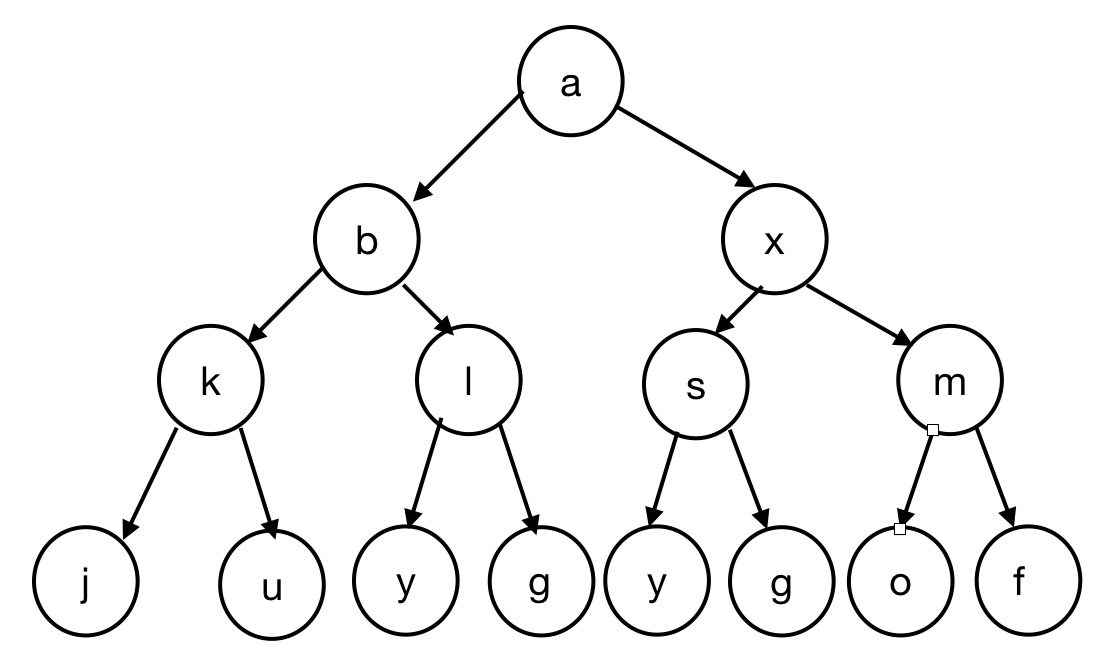
### **Binary Tree**

* + A binary tree is either
    - T = {TL, TR, r}, where TL, TR are binary trees
    - T = {} = ∅
* **Computation of the tree height**
  + The length of the longest path from the root to the leaf (count edges).
  + If we want to compute recursively:
    - height(T) = 1 + max(height(TL), height(TR)), where if height(null) = -1, which might be counter-intuitive but it follows the mathematical definition of tree height
* **Full Tree**:
  + A binary tree is **full** *if and only if*
    - Either: F = {}
    - Or: F = {TL, TR, r} where TL, TR both have either 0 or 2 children

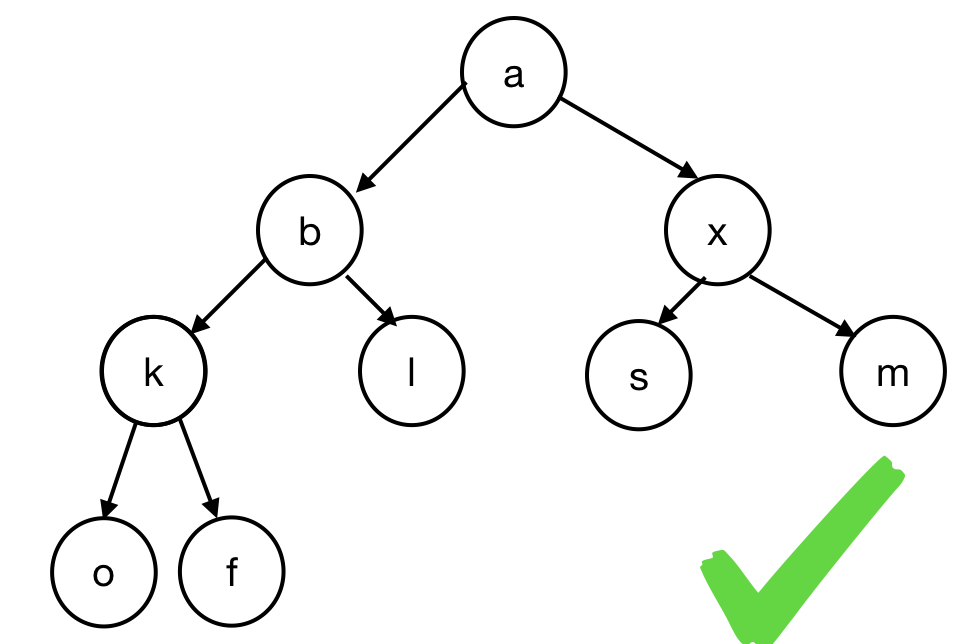
  

* **Perfect Tree**
  + A perfect tree **Ph** is defined by its height
    - Ph is a tree of height **h**, with
      * P-1 = {}
      * Ph = {r, Ph-1, Ph-1} when h>=0

P0 P1 P2 P3

* **Complete Tree**
  + A complete tree is
    - A perfect tree except for the last level
    - All leaves must be pushed to the **left**
  + Or, recursively, a complete tree **Ch** of height **h** is
    - C-1 = {}
    - Ch = {r, TL, TR} where
      * Either: TL = Ch-1 and TR = Ph-2
      * Or: TL = Ph-1 and TR = Ch-1

TL = Ch-1 and TR = Ph-2  TL = Ph-1 and TR = Ch-1

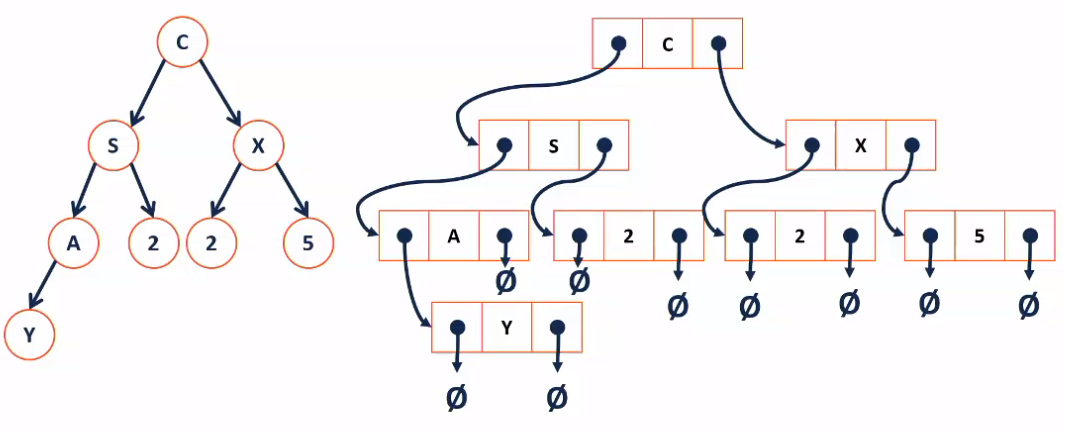
* **Tree property**
  + Is every full tree complete?
    - No



* + How about the other way - does every complete tree have to be full?
    - No



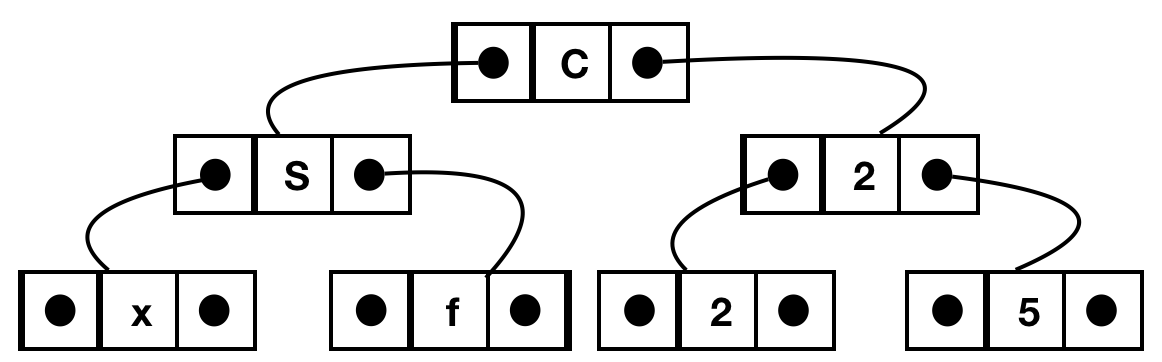
* Also,
  + Full does not imply perfect, so as complete does not imply perfect
  + Not full implies not perfect, thus perfect implies full; perfect also implies complete too.
* **Tree ADT**
  + Operations of Tree ADT
    - Insert
    - Remove
    - Traverse
  + A binary tree is just like a fancy linked list since they both traverse between nodes/TreeNode



|  |  |
| --- | --- |
| BinaryTree.h | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #pragma once  template <typename T>  class BinaryTree {  public:    /\* … \*/  private:  class TreeNode {  TreeNode \* left; // pointer to the left child  TreeNode \* right; // pointer to the right child  T & data;  TreeNode(T & t) :  data(t), left(NULL), right(NULL) {};  // constructor (initialization list)  };  TreeNode \* root\_;  // root of the tree: similar to head in linked list  } |

* Drawing

The actual tree



* + Every pointer not pointing to another node is NULL
* Number of null pointers in a binary tree
  + **Theorem**: A binary tree with n data items has n+1 null pointers.