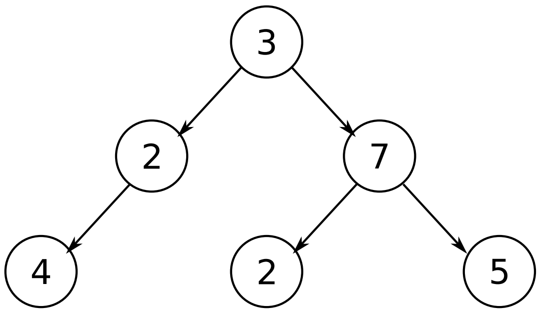
8 Prolog (ACR)

You are asked to write a Prolog program to work with binary trees. Your code should not rely on *any* library predicates and you should assume that the interpreter is running without occurs checking.



(*a*) Describe a data representation scheme for such trees in Prolog and demonstrate it by encoding the tree shown above. [3 marks]

(*b*) Implement a Prolog predicate `bfs/2` which effects a *breadth-first* traversal of a tree passed as the first argument and unifies the resulting list with its second argument. For example, when given the tree shown above as the first argument the predicate should unify the second argument with the list `[3,2,7,4,2,5]`. [4 marks]

(*c*) Explain why the `bfs/2` predicate might benefit from being converted to use difference lists. [2 marks]

(*d*) Implement a new predicate `diffbfs/2` which makes use of a difference list to exploit the benefit you identified in part (c). Your predicate should take the same arguments as `bfs/2`. [6 marks]

(*e*) A friend observes that a clause in `diffbfs/2` will need to contain an empty difference list and proposes two possible ways of representing it, either `[]-[]` or `A-A`.

Consider your implementation of `diffbfs/2`. For each use of an empty difference list, justify your choice and explain what can go wrong using the alternative form. [2 marks]

(*f*) Is your implementation amenable to *last call optimisation* (LCO)? If so, explain why. If not, give details of the minimal changes you would make to make LCO possible. [3 marks]