**1. Basic Parent-Child Relationship (Binary Tree)**

prolog

Copy code

% Facts

parent(john, mary).

parent(john, tom).

parent(mary, alice).

parent(mary, bob).

parent(tom, charlie).

% Rules

ancestor(X, Y) :- parent(X, Y).

ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

% Queries

% ?- ancestor(john, alice).

% ?- ancestor(mary, bob).

**2. Sibling Relationship**

prolog

Copy code

% Facts

parent(john, mary).

parent(john, tom).

% Rules

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

% Queries

% ?- sibling(mary, tom).

**3. Descendant Relation**

prolog

Copy code

% Facts

parent(anne, brian).

parent(anne, chris).

parent(brian, david).

% Rules

descendant(X, Y) :- parent(Y, X).

descendant(X, Y) :- parent(Z, X), descendant(Z, Y).

% Queries

% ?- descendant(david, anne).

**4. Leaf Node Detection**

prolog

Copy code

% Facts

parent(alice, bob).

parent(bob, charlie).

parent(bob, diana).

% Rules

leaf(X) :- parent(\_, X), \+ parent(X, \_).

% Queries

% ?- leaf(charlie).

**5. Height of Binary Tree**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

parent(b, d).

parent(b, e).

parent(c, f).

% Rules

height(nil, 0).

height(X, H) :- parent(X, L), parent(X, R), height(L, LH), height(R, RH), H is max(LH, RH) + 1.

% Queries

% ?- height(a, H).

**6. Ancestor Search with Path**

prolog

Copy code

% Facts

parent(a, b).

parent(b, c).

parent(c, d).

% Rules

ancestor(X, Y, [X|Path]) :- parent(X, Y), Path = [].

ancestor(X, Y, [X|Path]) :- parent(X, Z), ancestor(Z, Y, Path).

% Queries

% ?- ancestor(a, d, Path).

**7. Common Ancestor**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

parent(b, d).

parent(c, e).

% Rules

ancestor(X, Y) :- parent(X, Y).

ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

common\_ancestor(X, Y, A) :- ancestor(A, X), ancestor(A, Y).

% Queries

% ?- common\_ancestor(d, e, A).

**8. Binary Search Tree Insertion**

prolog

Copy code

% Facts

bst(nil).

% Rules

bst\_insert(nil, X, tree(X, nil, nil)).

bst\_insert(tree(Root, Left, Right), X, tree(Root, Left1, Right)) :- X =< Root, bst\_insert(Left, X, Left1).

bst\_insert(tree(Root, Left, Right), X, tree(Root, Left, Right1)) :- X > Root, bst\_insert(Right, X, Right1).

% Queries

% ?- bst\_insert(nil, 5, T), bst\_insert(T, 3, T1), bst\_insert(T1, 7, T2).

**9. Mirror of Binary Tree**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

parent(b, d).

parent(b, e).

% Rules

mirror(nil, nil).

mirror(tree(X, L1, R1), tree(X, L2, R2)) :- mirror(L1, R2), mirror(R1, L2).

% Queries

% ?- mirror(tree(a, tree(b, nil, nil), tree(c, nil, nil)), M).

**10. Tree Traversal: In-Order**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

parent(b, d).

parent(b, e).

% Rules

inorder(nil, []).

inorder(tree(Root, Left, Right), List) :- inorder(Left, L1), inorder(Right, L2), append(L1, [Root|L2], List).

% Queries

% ?- inorder(tree(a, tree(b, nil, nil), tree(c, nil, nil)), L).

**11. Finding Maximum Element in a Binary Tree**

prolog

Copy code

% Facts

tree(tree(1, tree(2, nil, nil), tree(3, nil, nil))).

% Rules

max\_tree(tree(X, nil, nil), X).

max\_tree(tree(X, L, R), Max) :- max\_tree(L, MaxL), max\_tree(R, MaxR), Max is max(X, max(MaxL, MaxR)).

% Queries

% ?- max\_tree(tree(1, tree(2, nil, nil), tree(3, nil, nil)), Max).

**12. Subtree Search**

prolog

Copy code

% Facts

tree(a, tree(b, nil, nil), tree(c, nil, nil)).

% Rules

subtree(T, tree(T, \_, \_)).

subtree(T, tree(\_, L, \_)) :- subtree(T, L).

subtree(T, tree(\_, \_, R)) :- subtree(T, R).

% Queries

% ?- subtree(tree(b, nil, nil), tree(a, tree(b, nil, nil), tree(c, nil, nil))).

**13. Depth of a Node**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

parent(b, d).

% Rules

depth(X, Y, 0) :- X = Y.

depth(X, Y, D) :- parent(X, Z), depth(Z, Y, D1), D is D1 + 1.

% Queries

% ?- depth(a, d, Depth).

**14. Balanced Binary Tree**

prolog

Copy code

% Facts

parent(a, b).

parent(a, c).

% Rules

height(nil, 0).

height(tree(\_, L, R), H) :- height(L, LH), height(R, RH), H is max(LH, RH) + 1.

balanced(nil).

balanced(tree(\_, L, R)) :- balanced(L), balanced(R), height(L, LH), height(R, RH), abs(LH - RH) =< 1.

% Queries

% ?- balanced(tree(a, tree(b, nil, nil), tree(c, nil, nil))).

**15. Leftmost Node in Binary Tree**

prolog

Copy code

% Facts

parent(a, b).

parent(b, d).

% Rules

leftmost(X, X) :- \+ parent(X, \_).

leftmost(X, Y) :- parent(X, Z), leftmost(Z, Y).

% Queries

% ?- leftmost(a, L).