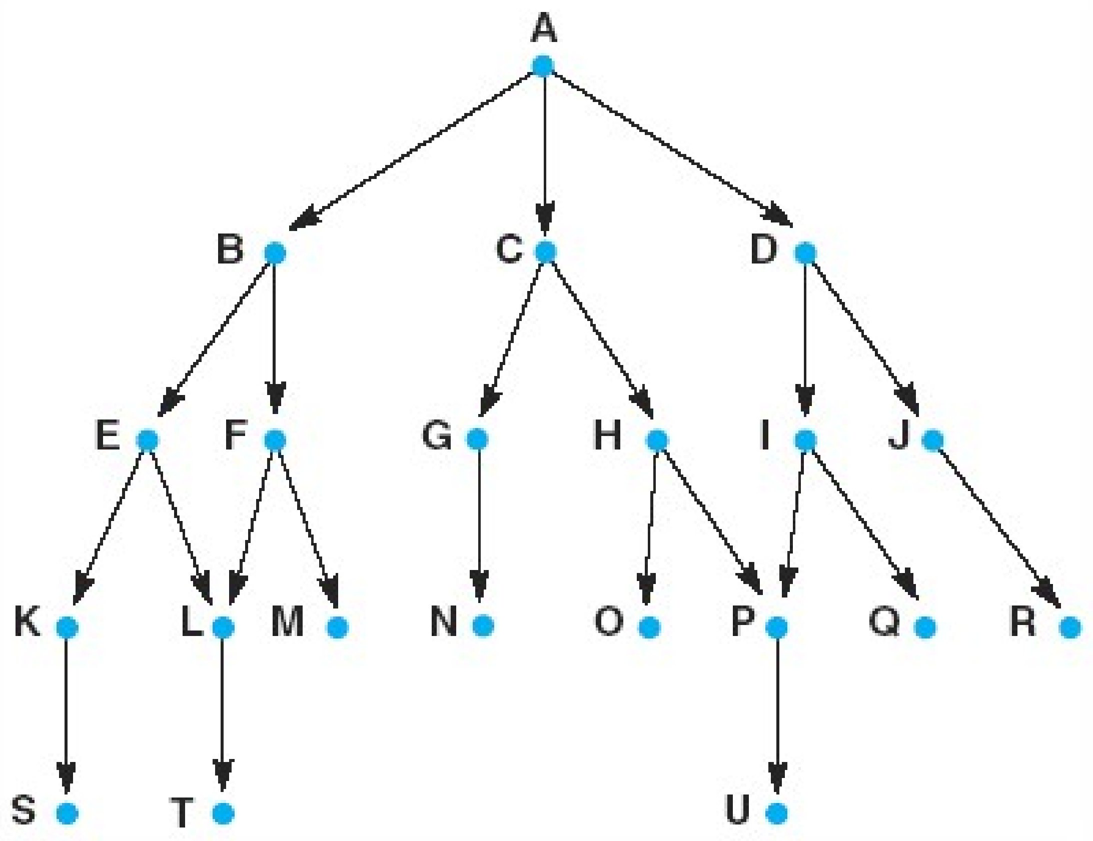
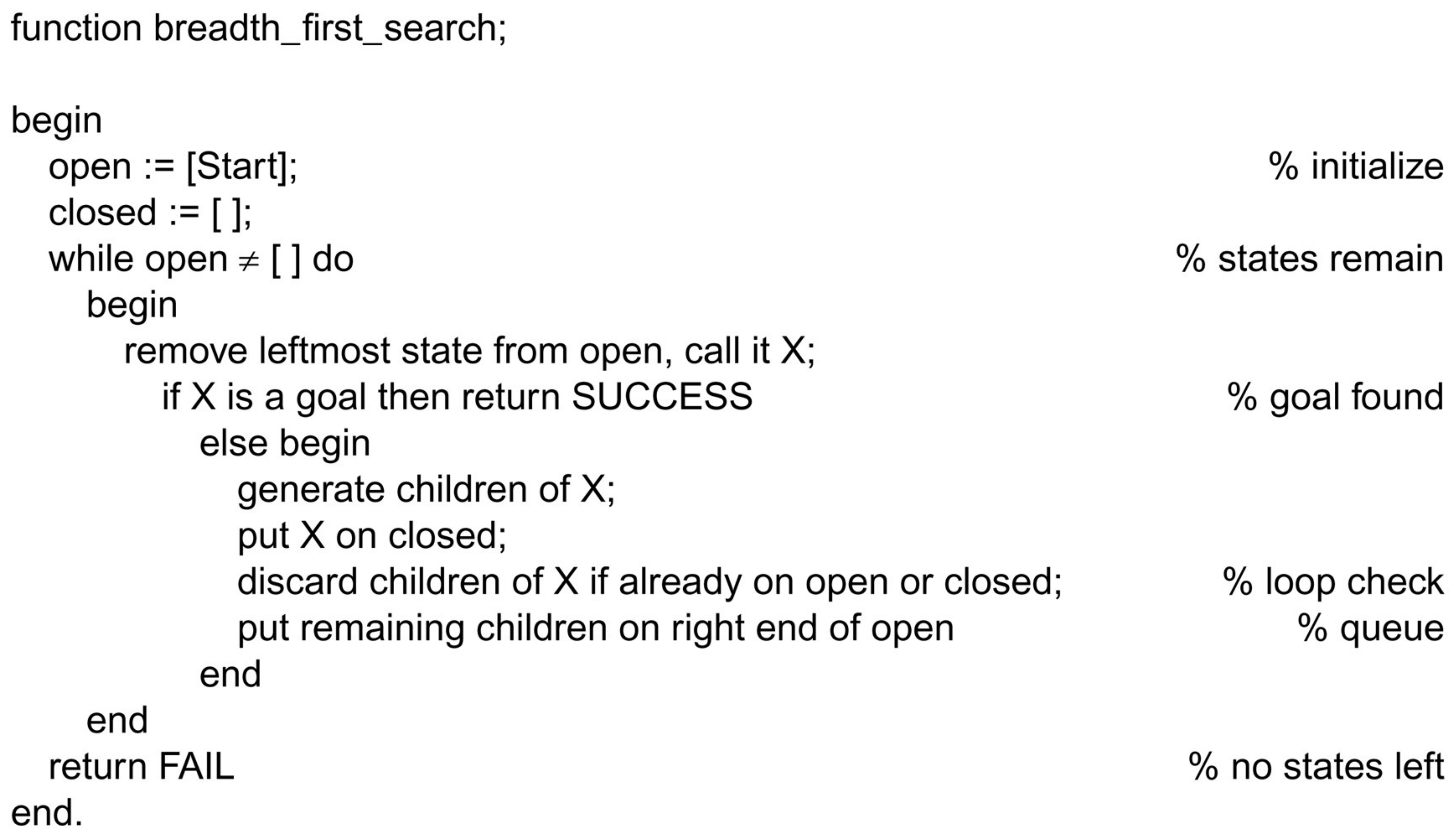
Fig 3.15 Graph for breadth - and depth - first search examples.



Luger: Artificial Intelligence, 6th edition. © Pearson Education Limited, 2009

Function breadth_first search algorithm



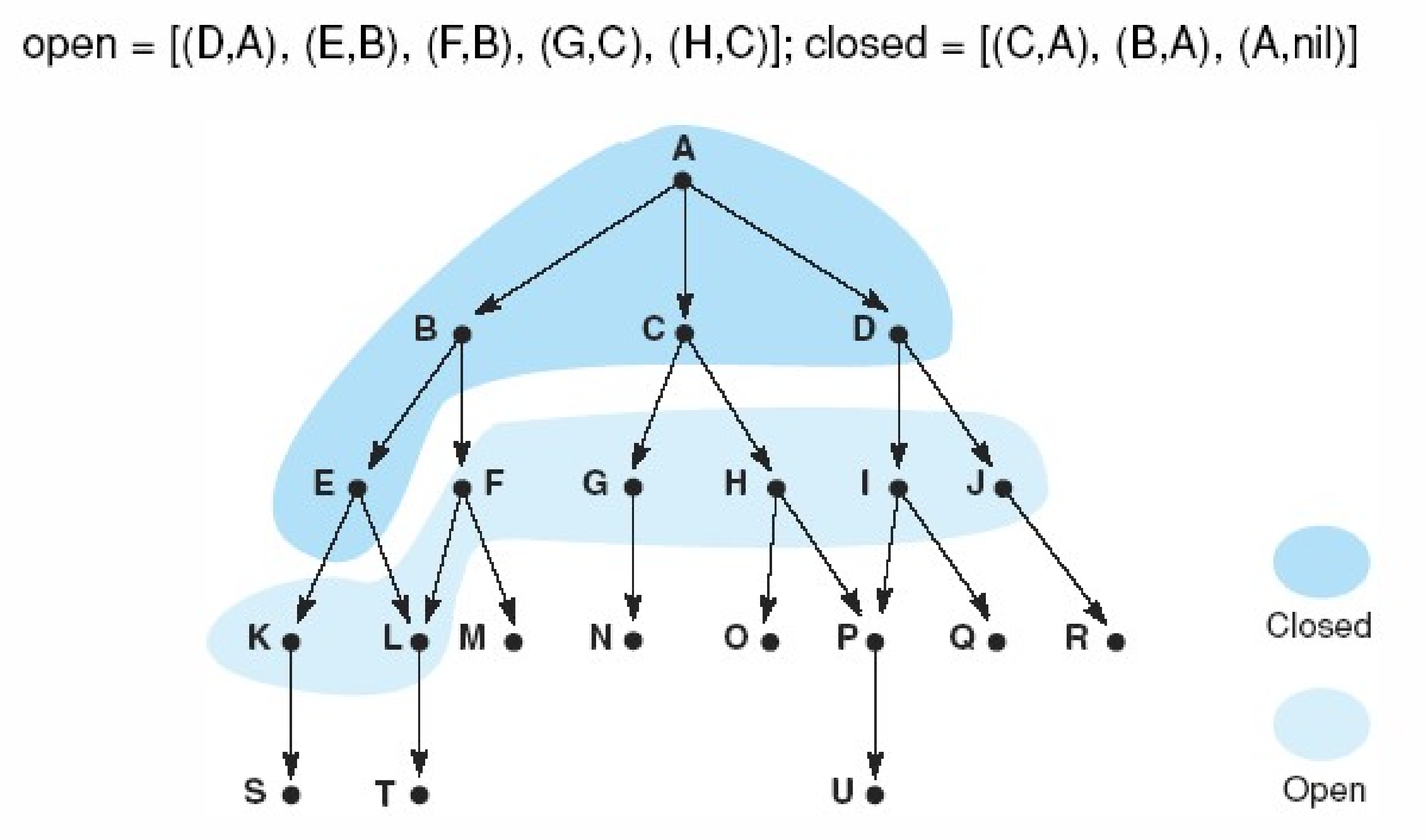
```
function breadth_first_search;

begin
  open := [Start];                                                  % initialize
  closed := [ ];
  while open ≠ [ ] do                                               % states remain
    begin
      remove leftmost state from open, call it X;
        if X is a goal then return SUCCESS                          % goal found
          else begin
            generate children of X;
            put X on closed;
            discard children of X if already on open or closed;     % loop check
            put remaining children on right end of open             % queue
          end
    end
  return FAIL                                                       % no states left
end.
```

A trace of breadth_first_search on the graph of Figure 3.13

1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [C,D,E,F]; closed = [B,A]**
4. **open = [D,E,F,G,H]; closed = [C,B,A]**
5. **open = [E,F,G,H,I,J]; closed = [D,C,B,A]**
6. **open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]**
7. **open = [G,H,I,J,K,L,M]** (as L is already on open); **closed = [F,E,D,C,B,A]**
8. **open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]**
9. and so on until either U is found or **open** = [ ]

Fig 3.16 Graph of Fig 3.15 at iteration 6 of breadth-first search. States on open and closed are highlighted.



open = [(D,A), (E,B), (F,B), (G,C), (H,C)]; closed = [(C,A), (B,A), (A,nil)]

Function depth_first_search algorithm

```
begin
  open := [Start];                                              % initialize
  closed := [ ];
  while open $\neq$ [ ] do                                       % states remain
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS                        % goal found
        else begin
          generate children of X;
          put X on closed;
          discard children of X if already on open or closed;   % loop check
          put remaining children on left end of open            % stack
        end
    end;
  return FAIL                                                   % no states left
end.
```

A trace of depth_first_search on the graph of Figure 3.13

1. **open = [A]; closed = [ ]**
2. **open = [B,C,D]; closed = [A]**
3. **open = [E,F,C,D]; closed = [B,A]**
4. **open = [K,L,F,C,D]; closed = [E,B,A]**
5. **open = [S,L,F,C,D]; closed = [K,E,B,A]**
6. **open = [L,F,C,D]; closed = [S,K,E,B,A]**
7. **open = [T,F,C,D]; closed = [L,S,K,E,B,A]**
8. **open = [F,C,D]; closed = [T,L,S,K,E,B,A]**
9. **open = [M,C,D],** as L is already on **closed**; **closed = [F,T,L,S,K,E,B,A]**
10. **open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]**
11. **open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]**