Artificial Intelligence

AI2002

Rushda Muneer

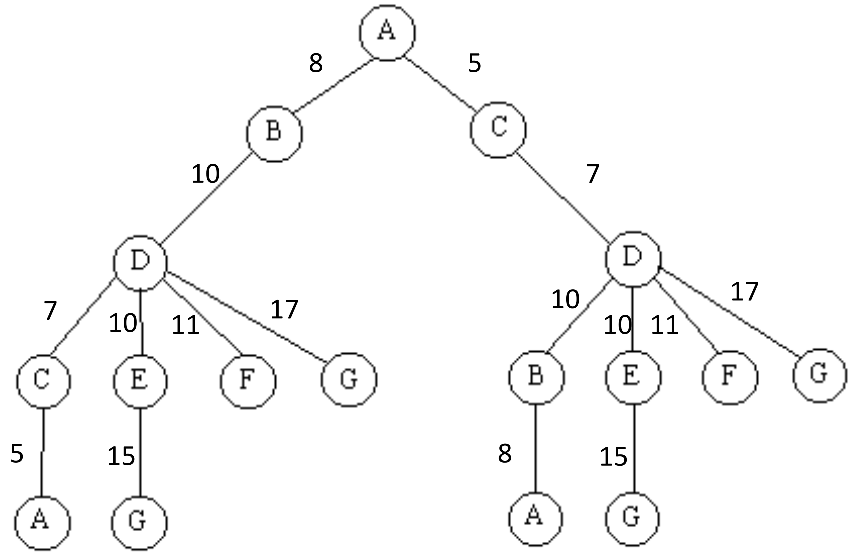
Spring 2026

Uniform Cost Search

- Expands the node n with the lowest path cost g(n).
- This is done by storing the frontier as a priority queue ordered by g.
- It explores cumulative path cost instead of heuristics that direct towards a goal making it a weighted version of BFS –an uninformed search.

UCS (Priority Queue)

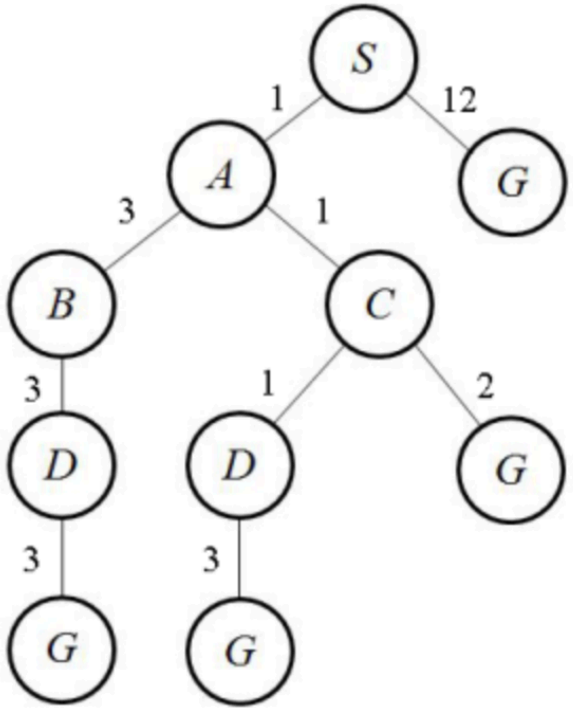
- begin
  - Open = [Start (path cost=0)];
  - Closed = [];
  - While open <> [] do
  - begin
    - remove highest priority (lowest path cost) state from open, call it X;
    - if X is a goal then return(success)
    - else begin
      - generate children of X;
      - put X on closed;
      - add the children of X if not in open or closed
      - replace the children of X if they already exist in open with higher path cost
  - end
  - return(failure)
- end.



1. Open = [A(0)]; closed = [ ];
2. Open = [B(8),C(5)]; closed = [A(0)];
3. Open = [B(8),D(12)]; closed = [C(5),A(0)];
4. Open = [D(12)]; closed = [B(8),C(5),A(0)];
5. Open = [E(22),F(23),G(29)]; closed = [D(12),B(8),C(5),A(0)];
6. Open = [F(23),G(29)]; closed = [E(22),D(12),B(8),C(5),A(0)];
7. Open = [G(29)]; closed = [F(23), E(22),D(12),B(8),C(5),A(0)];

Activity

- Perform Uniform cost search on the given tree
- Show how open and closed queues change in each iteration



Depth Limited Search

- The failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit.
- The nodes at depth are treated as if they have no successors.
- The depth limit solves the infinite-path problem.

Iterative Deepening Search

- This form of search is an **excellent compromise** between depth-first and breadth-first searches.
- The search **sets a depth limit** and then does **a depth-first search down** to this **limit** - if a solution is **found** it exits with **success**.
- If a solution **is not found** then the **limit is increased** and the search run again but down to this new depth.
- Obviously, a large number of nodes are **revisited a number of times**, but the **overall loss due to this becomes insignificant** for a search through a **large** tree.
- The **number of visited nodes grows** exponentially as you go down the tree so almost all the time is spent at **the deepest levels** which are **only visited once** unless the **search repeats to a deeper level**, at which the time spent in these levels will become insignificant.
- The memory consumption is the **same** as that for a **depth-first search**, but the search is always guaranteed to find a solution, as in **breadth-first search.**

IDS

- Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search, that finds the best depth limit.
- It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.
- This will occur when the depth limit reaches d, the depth of the shallowest goal node.
- Iterative deepening combines the benefits of depth-first and breadth-first search.

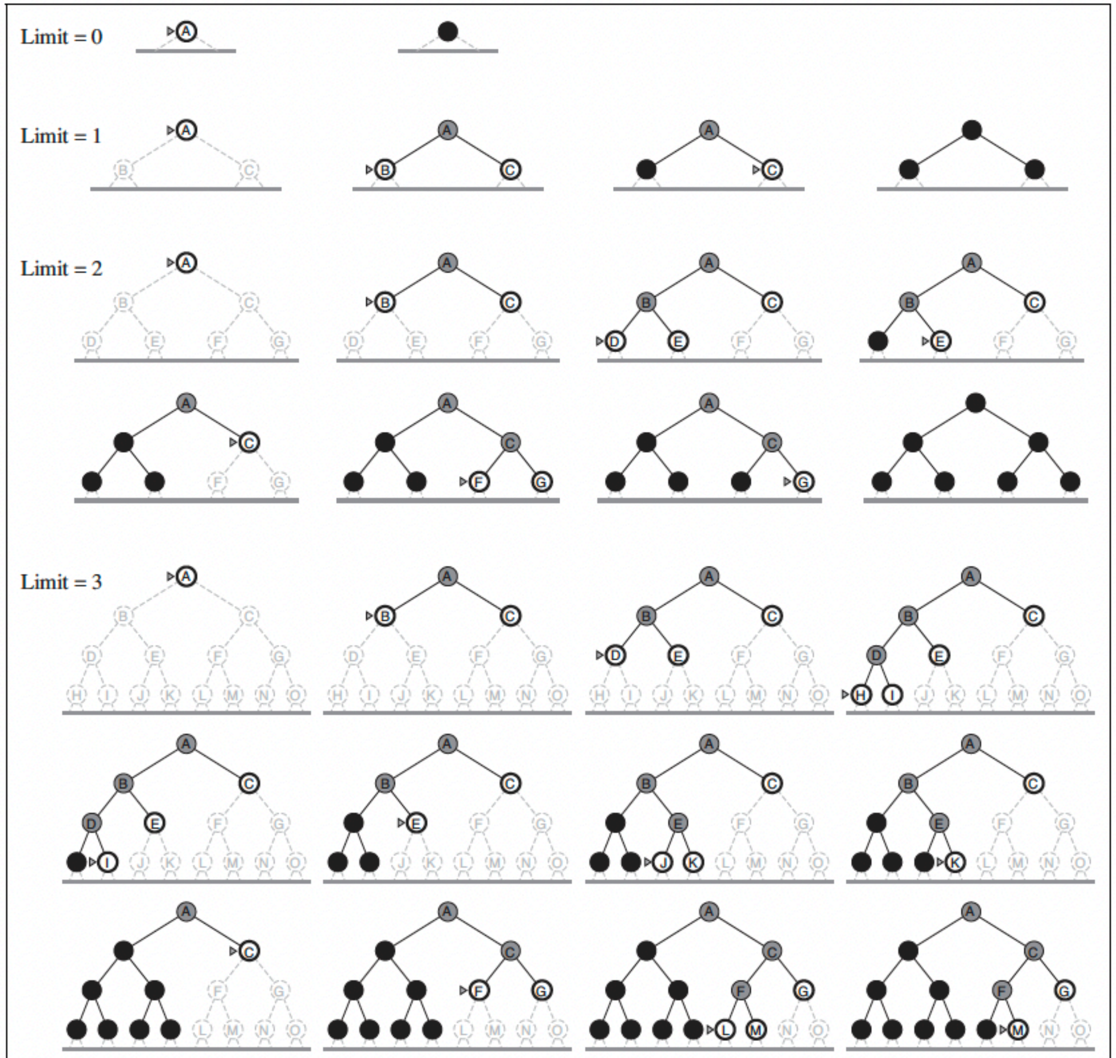
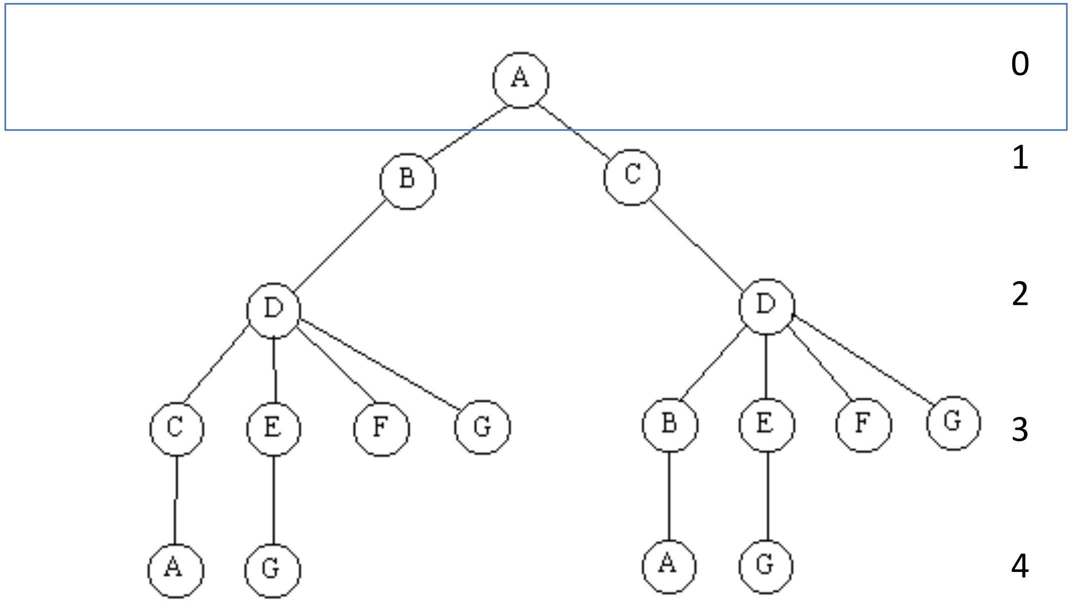


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

IDS

- begin
- for limit = 0 to ∞ do
- begin
  - Open = [(Start, 0)];
  - Closed = [];
  - while Open <> [] do
  - begin
    - remove leftmost state (X, d) from Open;
    - if X is a goal then
      - return(success);
    - else if d < limit then
    - begin
      - generate children of X;
      - put X on Closed;
      - discard children of X if already on Open or Closed;
      - put remaining children on left end of Open with depth d+1;
    - end
  - end
- end
- return(failure)
- end

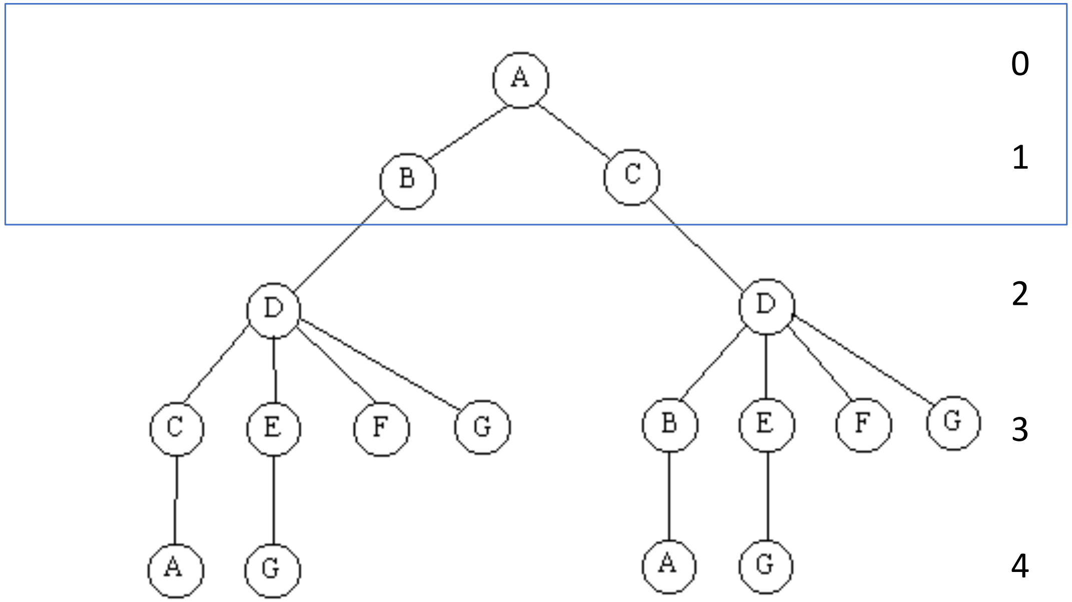


Iteration 1:

1. Open = [A]; closed = [ ];
2. Open = [ ]; closed = [A];

IDS

- begin
- for limit = 0 to ∞ do
- begin
-    Open = [(Start, 0)];
-    Closed = [];
-    while Open <> [] do
-    begin
-       remove leftmost state (X, d) from Open;
-       if X is a goal then
-          return(success);
-       else if d < limit then
-       begin
-          generate children of X;
-          put X on Closed;
-          discard children of X if already on Open or Closed;
-          put remaining children on left end of Open with depth d+1;
-       end
-    end
- end
- return(failure)
- end

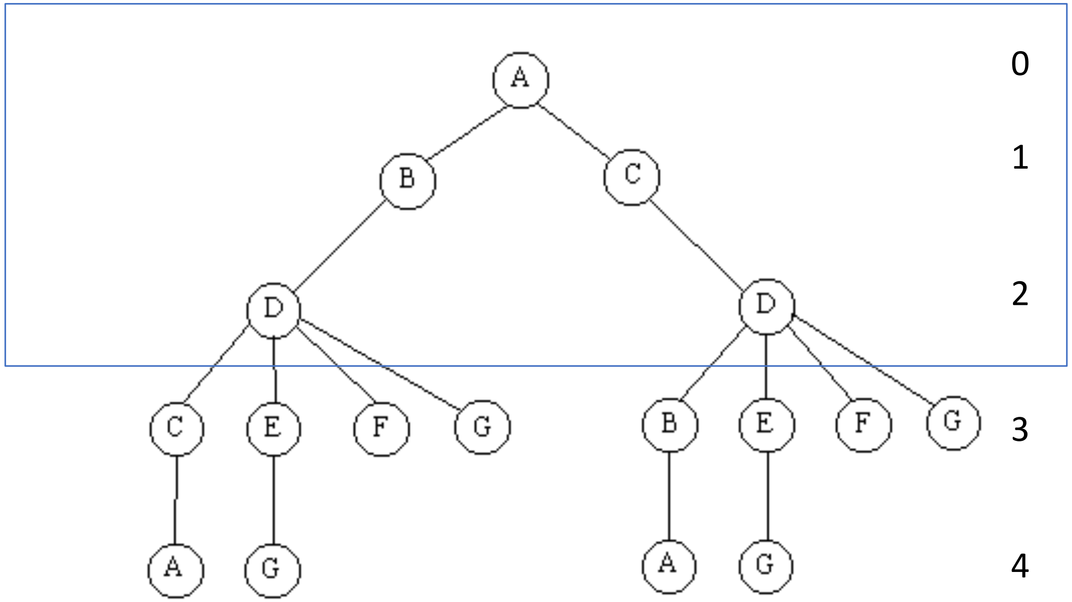


Iteration 2:

1. Open = [A]; closed = [ ];
2. Open = [B,C]; closed = [A];
3. Open = [C]; closed = [B,A];
4. Open = [ ]; closed = [C,B,A];

IDS

- begin
- for limit = 0 to ∞ do
- begin
- Open = [(Start, 0)];
- Closed = [];
- while Open <> [] do
- begin
- remove leftmost state (X, d) from Open;
- if X is a goal then
- return(success);
- else if d < limit then
- begin
- generate children of X;
- put X on Closed;
- discard children of X if already on Open or Closed;
- put remaining children on left end of Open with depth d+1;
- end
- end
- end
- return(failure)
- end

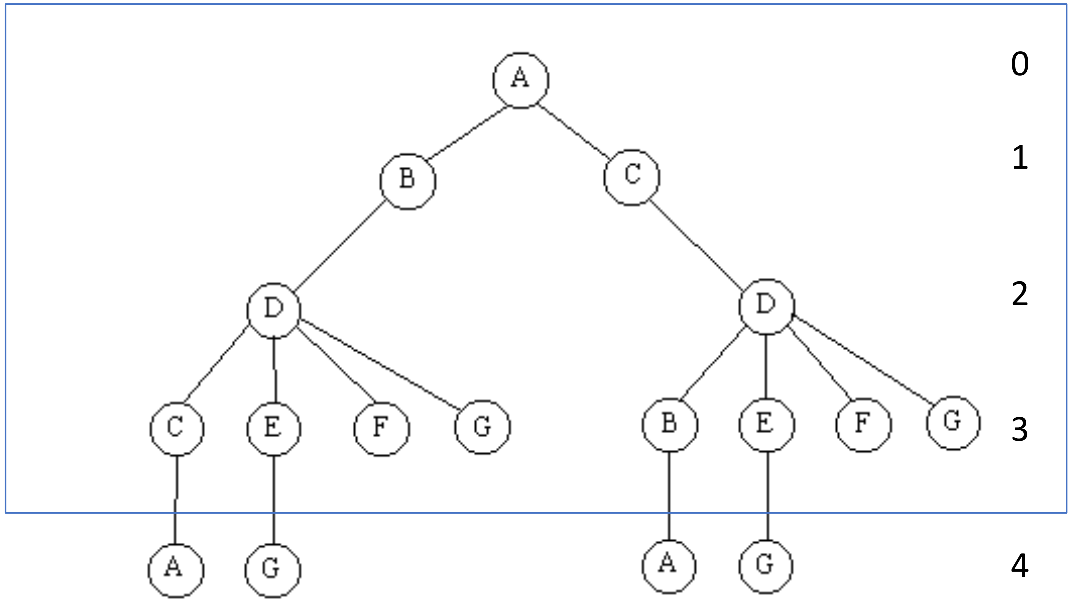


Iteration 3:

1. Open = [A]; closed = [ ];
2. Open = [B,C]; closed = [A];
3. Open = [D,C]; closed = [B,A];
4. Open = [C]; closed = [D,B,A];
5. Open = []; closed = [C,D,B,A];

IDS

- begin
- for limit = 0 to ∞ do
- begin
- Open = [(Start, 0)];
- Closed = [];
- while Open <> [] do
- begin
- remove leftmost state (X, d) from Open;
- if X is a goal then
- return(success);
- else if d < limit then
- begin
- generate children of X;
- put X on Closed;
- discard children of X if already on Open or Closed;
- put remaining children on left end of Open with depth d+1;
- end
- end
- end
- return(failure)
- end

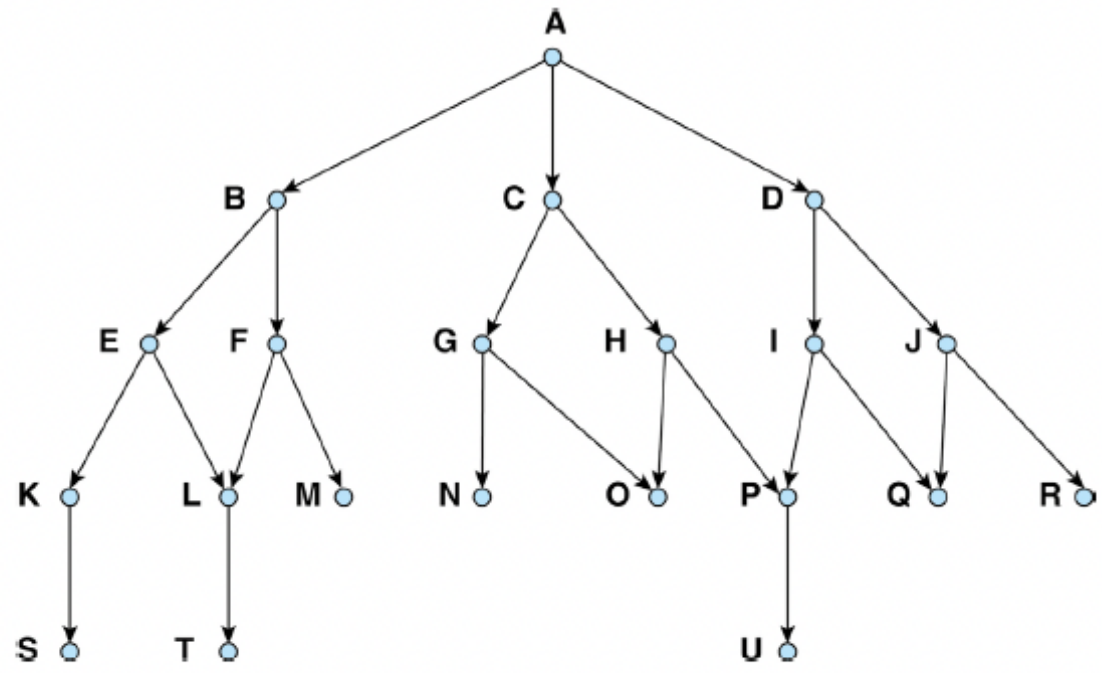


Iteration 4:

1. Open = [A]; closed = [ ];
2. Open = [B,C]; closed = [A];
3. Open = [D,C]; closed = [B,A];
4. Open = [E,F,G,C]; closed = [D,B,A];
5. Open = [F,G,C]; closed = [E,D,B,A];
6. Open = [G,C]; closed = [F,E,D,B,A];

Activity

- Perform IDS on the given tree from A to G
- Write down how each iteration is reflected in open and closed queues



Comparing uninformed search strategies

- For graph searches, the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Homework Readings

- Modern Approach Chapter 3 (3.4.2, 3.4.4, 3.4.5)
- G F Lugar Chapter 3 (3.2.4)