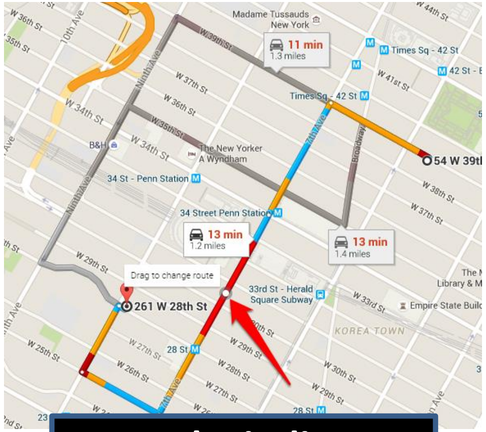
SEARCH METHODS IN AI

STATE SPACE SEARCH

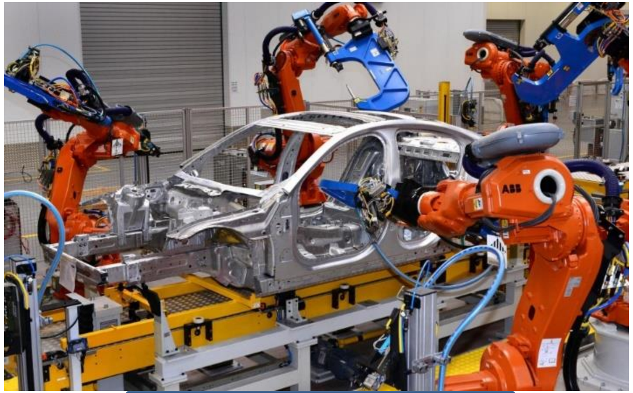
Partha P Chakrabarti
Indian Institute of Technology Kharagpur

COMPLEX PROBLEMS & SOLUTIONS

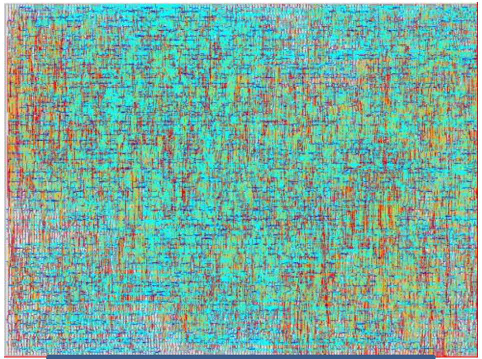


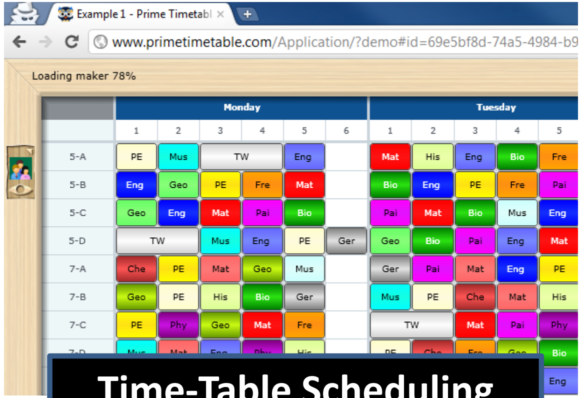
Path Finding

Chess Playing



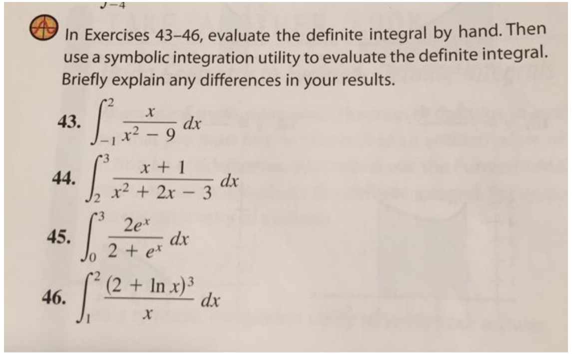
Robot Assembly


VLSI Chip Design



	Monday						Tuesday				
	1	2	3	4	5	6	1	2	3	4	5
5-A	PE	Mus	TW		Eng		Mat	His	Eng	Bio	Fre
5-B	Eng	Geo	PE	Fre	Mat		Bio	Eng	PE	Fre	Pai
5-C	Geo	Eng	Mat	Pai	Bio		Pai	Mat	Bio	Mus	Eng
5-D	TW		Mus	Eng	PE	Ger	Geo	Bio	Pai	Eng	Mat
7-A	Che	PE	Mat	Geo	Mus		Ger	Pai	Mat	Eng	PE
7-B	Geo	PE	His	Bio	Ger		Mus	PE	Che	Mat	His
7-C	PE	Phy	Geo	Mat	Fre		TW		Mat	Pai	Phy

Time-Table Scheduling


In Exercises 43–46, evaluate the definite integral by hand. Then use a symbolic integration utility to evaluate the definite integral. Briefly explain any differences in your results.

43. $\int_{-1}^{2} \frac{x}{x^2 - 9}\,dx$

44. $\int_{2}^{3} \frac{x + 1}{x^2 + 2x - 3}\,dx$

45. $\int_{0}^{3} \frac{2e^x}{2 + e^x}\,dx$

46. $\int_{1}^{2} \frac{(2 + \ln x)^3}{x}\,dx$

Symbolic Integration

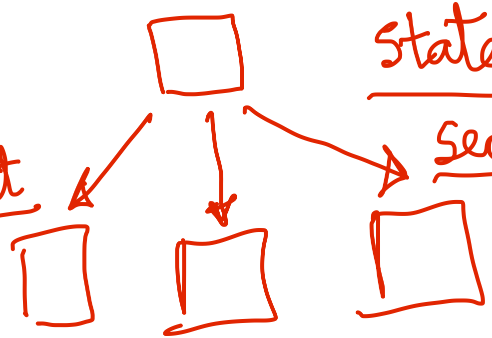
AUTOMATED PROBLEM SOLVING BY SEARCH

- Generalized Techniques for Solving Large Classes of Complex Problems
- Problem Statement is the Input and solution is the Output, sometimes even the problem specific algorithm or method could be the Output
- Problem Formulation by AI Search Methods consists of the following key concepts
  - Configuration or State
  - Constraints or Definitions of Valid Configurations
  - Rules for Change of State and their Outcomes
  - Initial or Start Configurations
  - Goal Satisfying Configurations
  - An Implicit State or Configuration Space
  - Valid Solutions from Start to Goal in the State Space
  - General Algorithms which SEARCH for Solutions in this State Space

State space

Search

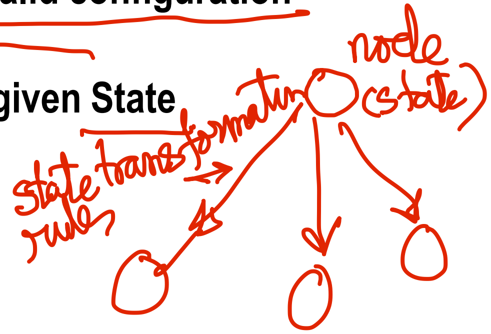
implicit



- **ISSUES**
  - Size of the Implicit Space, Capturing Domain Knowledge, Intelligent Algorithms that work in reasonable time and Memory, Handling Incompleteness and Uncertainty

BASICS OF STATE SPACE MODELLING

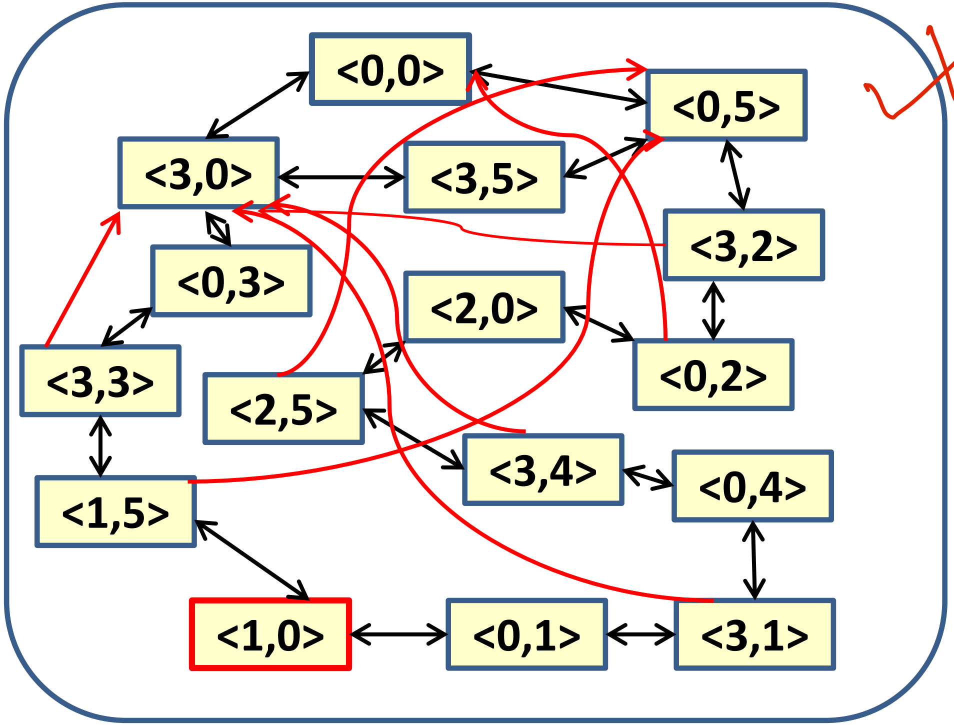
- **STATE or CONFIGURATION:**
  - A set of variables which define a state or configuration
  - Domains for every variable and constraints among variables to define a valid configuration
- **STATE TRANSFORMATION RULES or MOVES:**
  - A set of RULES which define which are the valid set of NEXT STATE of a given State
  - It also indicates who can make these Moves (OR Nodes, AND nodes, etc)
- **STATE SPACE or IMPLICIT GRAPH**
  - The Complete Graph produced out of the State Transformation Rules.
  - Typically too large to store. Could be Infinite.
- **INITIAL or START STATE(s), GOAL STATE(s)**
- **SOLUTION(s), COSTS**
  - Depending on the problem formulation, it can be a PATH from Start to Goal or a Sub-graph of And-ed Nodes
- **SEARCH ALGORITHMS**
  - Intelligently explore the Implicit Graph or State Space by examining only a small sub-set to find the solution
  - To use Domain Knowledge or HEURISTICS to try and reach Goals faster



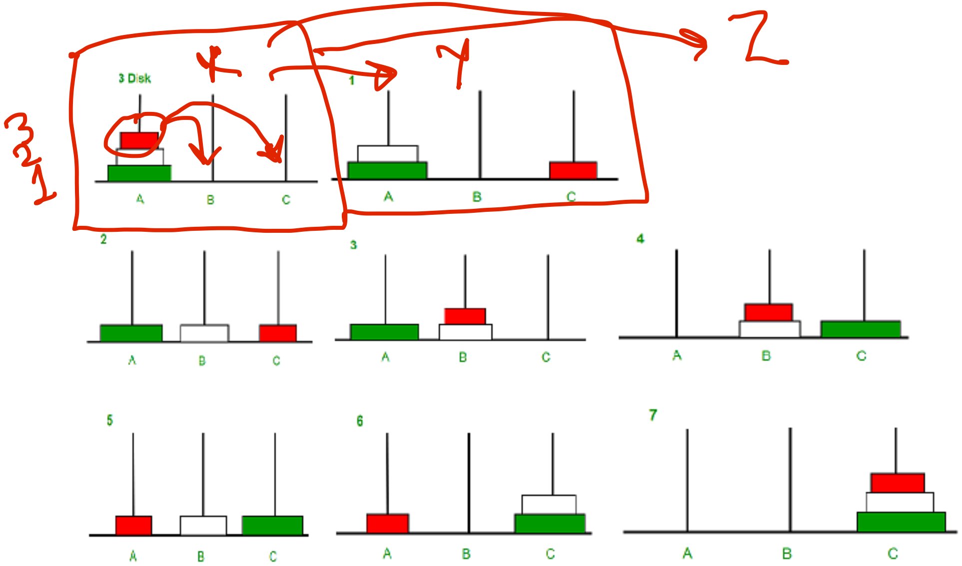
TWO JUG PROBLEM

- There is a large bucket B full of water and Two (02) jugs, J1 of volume 3 litre and J2 of volume 5 litre. You are allowed to fill up any empty jug from the bucket, pour all water back to the bucket from a jug or pour from one jug to another. The goal is to have jug J1 with exactly one (01) litre of water
- State Definition: <J1, J2>
- Rules:
  - Fill (J1): <J1, J2> to <3,J2>
  - Fill (J2): <J1, J2> to <J1, 5>
  - Empty (J1), Empty (J2): Similarly defined
  - Pour (J1, J2): <J1, J2> to <X,Y>, where
    - X = 0 and Y = J1 + J2 if J1+J2 ≤ 5,
    - Y = 5 and X = (J1+ J2) – 5, if J1+J2 > 5
  - Pour (J2, J1): Similarly defined
- Start: <0,0>, Goal: <1,0>
- Part of State Space Shown on the right

(Not all Links shown here)



3 DISK, 3 PEG TOWER of HANOI STATE SPACE



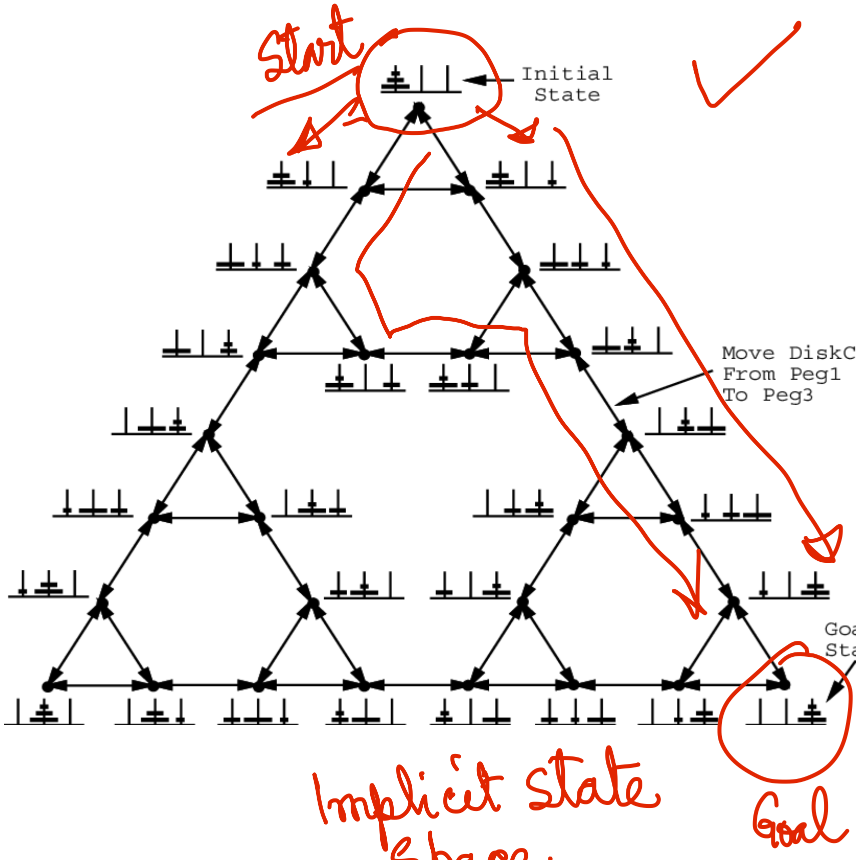
state definition

A: 1 2

B: –

C: 3

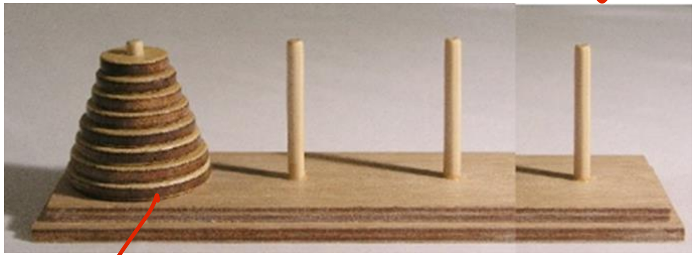
move(X, Y)



Implicit state space

STATES, SPACES, SOLUTIONS, SEARCH

- States
  - Full / Perfect Information and Partial Information States
- State Transformation Rules
  - Deterministic Outcomes
  - Non-Deterministic / Probabilistic Outcomes
- State Spaces As Generalized Games
  - Single Player: OR Graphs
  - Multi-Player: And / Or, Adversarial, Probabilistic Graphs
- Solutions
  - Paths
  - Sub-graphs
  - Expected Outcomes
- Costs
- Sizes
- Domain Knowledge
- Algorithms for Heuristic Search



BLOCK DIAGRAM OF A BOILER SYSTEM

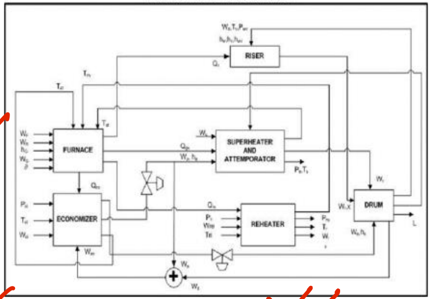
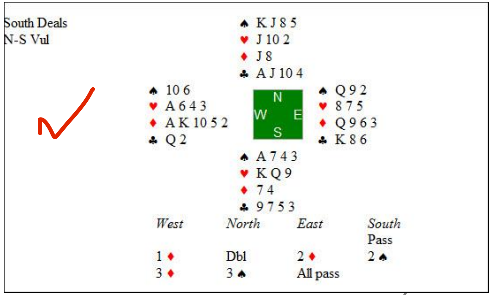
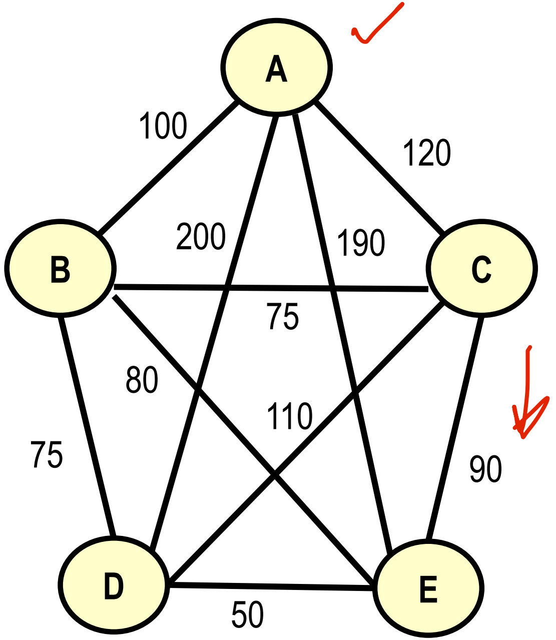


Plate Cutting



OR-Graph: TRAVELLING SALESPERSON PROBLEM



$A \xrightarrow[100]{} B \xrightarrow[75]{} C \xrightarrow[110]{} D \xrightarrow[50]{} E \xrightarrow[190]{} A$

515

$A \xrightarrow[100]{} B \xrightarrow[75]{} D \xrightarrow[50]{} E \xrightarrow[90]{} C \xrightarrow[120]{} A$

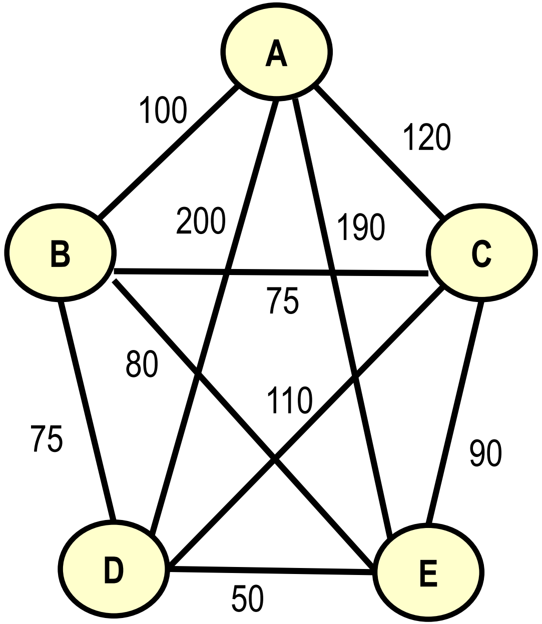
state : $\langle \{ \quad \}, c \rangle$ → current cost

→ sequence of states visited till now

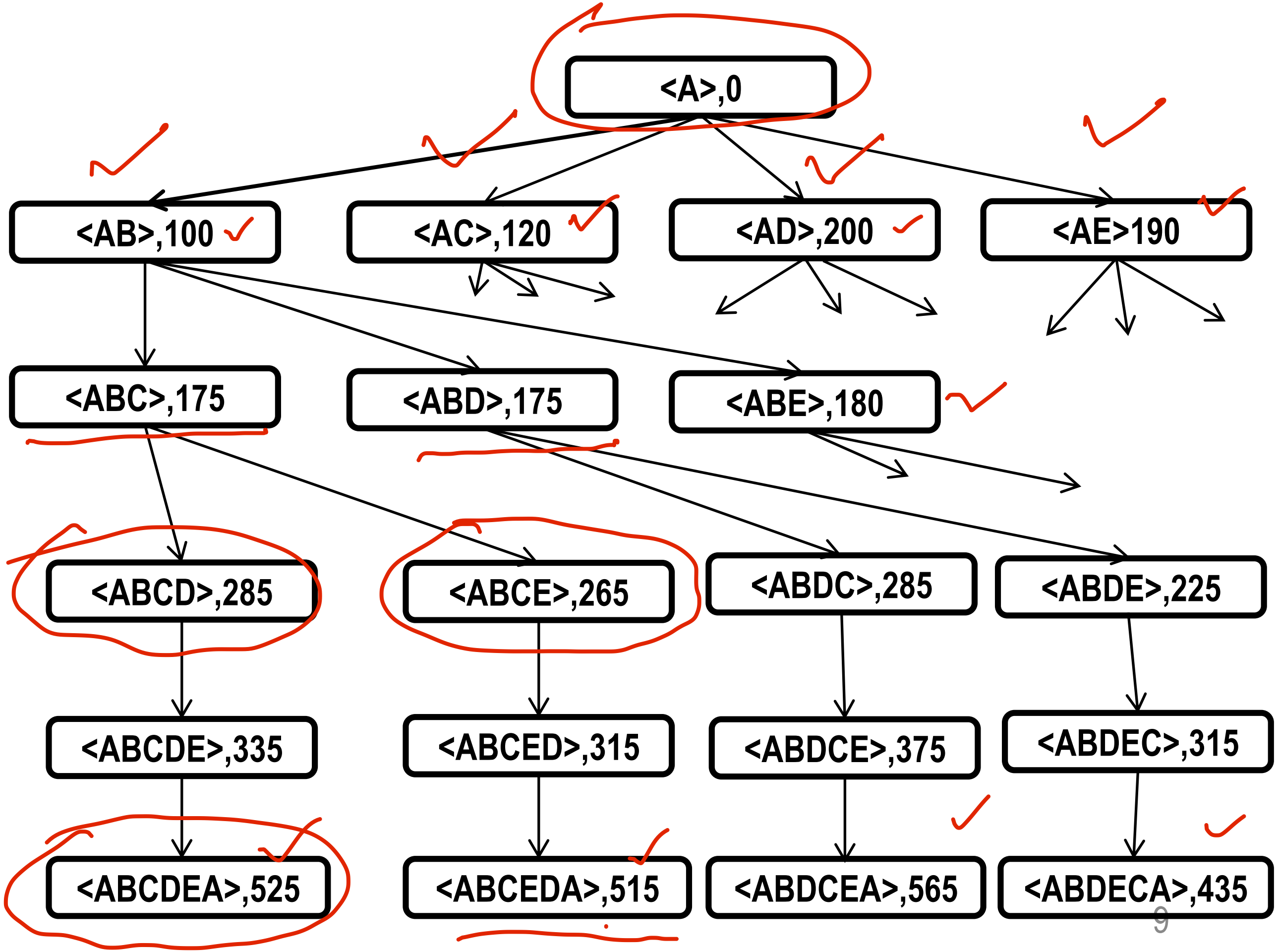
$\langle \{A, B, C\}, 175 \rangle$

state-transformation rules

OR-Graph: TRAVELLING SALESPERSON PROBLEM



implicit graph



MODELLING AND/OR GRAPHS:

OR Nodes are ones for which one has a choice.

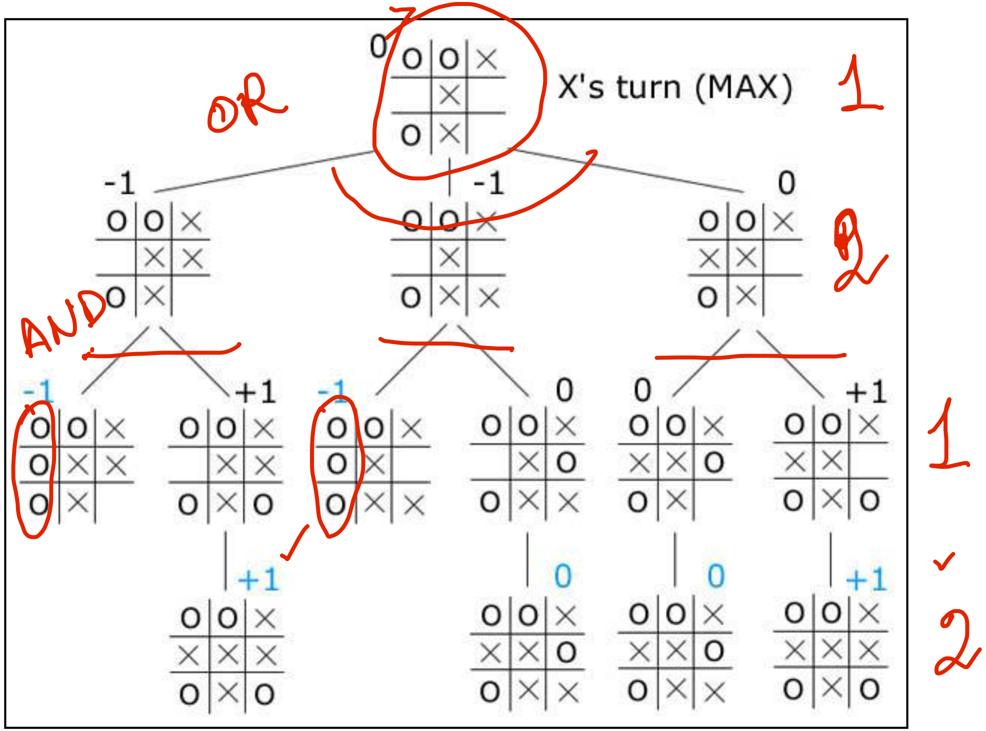
The AND nodes could be compositional (sum, product, min, max, etc., depending on the way the sub-problems are composed),

Adversarial (game where the other parties have a choice)

or

Probabilistic (Environmental Actions)

AND/OR GRAPHS: ADVERSARIAL

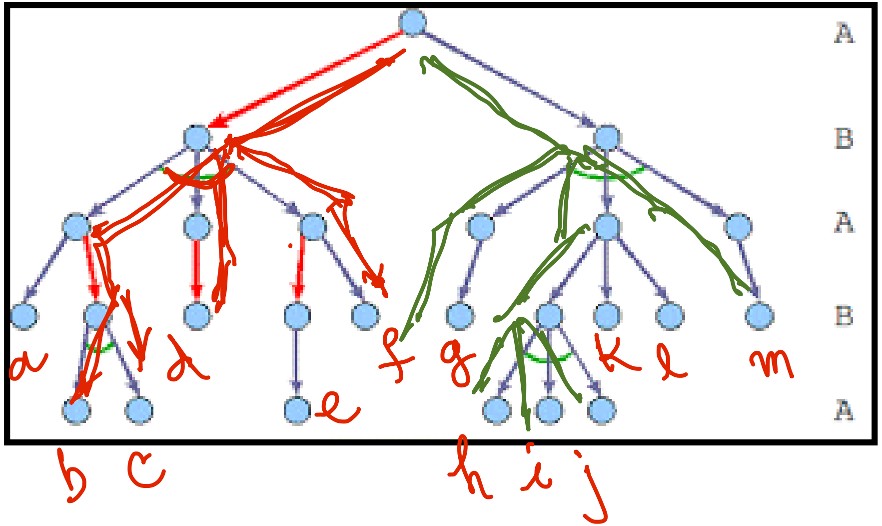


x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}

$\{\phi, O, X\}$

AND-OR graph

AND/OR GRAPHS: COMPOSITIONAL / ADVERSARIAL / PROBABILISTIC



→ OR → MIN

→ AND → SUM / PRODUCT

→ MAX

Solution is AND-subgraph which leads to terminal nodes

Costs

COMPOSITIONAL AND/OR GRAPHS – MATRIX CHAIN MULTIPLICATION

(M1 X (M2 X (M3 X M4))) = ((M1 x M2) X (M3 X M4)) = (((M1 x M2) X M3) X M4) = (M1 X (M2 X M3)) X M4)

BUT THE NUMBER OF MULTIPLICATIONS TO GET THE ANSWER DIFFER !!

Let A be a [p by q] Matrix and B be a [q by r] Matrix. The number of multiplications needed to compute A X B = p*q*r [p by r]

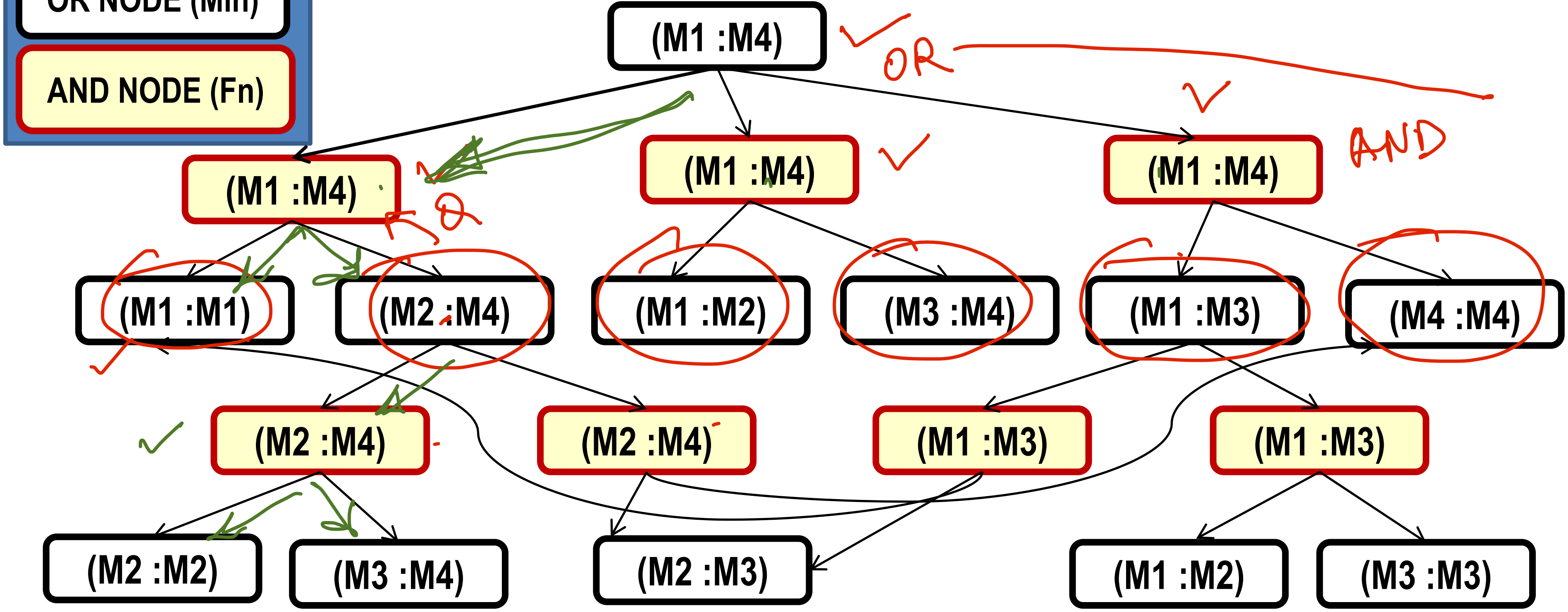
Thus if M1 be a [10 by 30] Matrix, M2 be a [30 by 5] Matrix and M3 be a [5 by 60] Matrix

Then the number of computations for

(M1 X M2) X M3 = 10*30*5 for P = (M1 X M2) and 10*5*60 for P X M3. Total = 4500

M1 X (M2 X M3) = 30*5*60 for Q = (M2 X M3) and 10*30*60 for M1 X Q. Total = 27000

COMPOSITIONAL AND/OR GRAPHS: MATRIX CHAIN MULTIPLICATION



(M1 X (M2 X (M3 X M4))) = (M1 x M2) X (M3 X M4) = (((M1 x M2) X M3) X M4) =

M1 X Q

(M1 X (M2 X M3)) X M4)

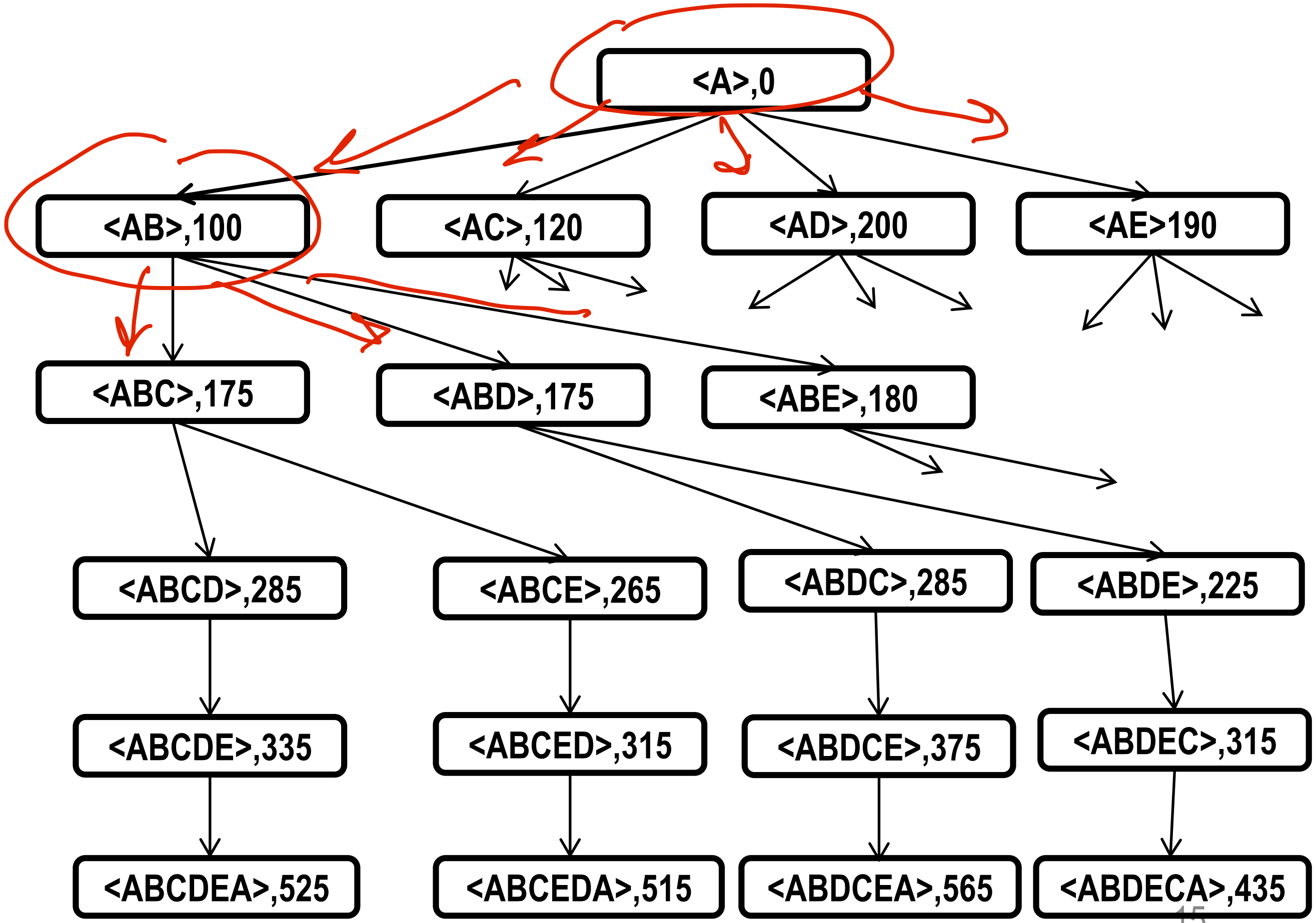
SEARCHING IMPLICIT GRAPHS

Given the start state the SEARCH Algorithm will create successors based on the State Transformation Rules and make part of the Graph EXPLICIT.

It will EXPAND the Explicit graph INTELLGENTLY to rapidly search for a solution without exploring the entire Implicit Graph or State Space

For OR Graphs, the solution is a PATH from start to Goal.

Cost is usually sum of the edge costs on the path, though it could be something based on the problem



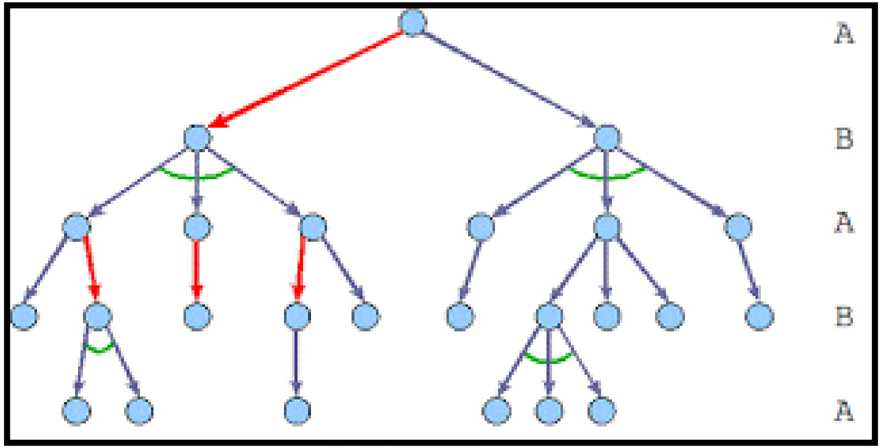
SEARCHING IMPLICIT GRAPHS

For And/OR Graphs, the Solution is an AND Subgraph rooted at the Start and each leaf is a Goal Node.

The Cost of OR Node is usually a Min or Max.

The Cost at the AND Node depends on the type of Node (Compositional, Adversarial, Probabilistic).

For Adversarial two player games, Max / Min is used at AND Node (reverse of Or Node)



MIN

MAX

-1 0 +1

SEARCHING IMPLICIT GRAPHS

The various Search Algorithms include

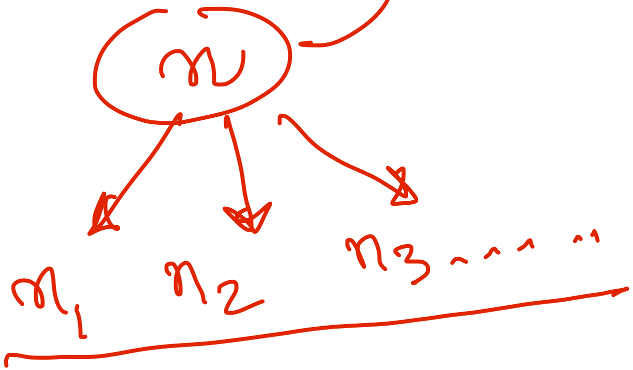
- BASIC Algorithms: Depth-First (DFS), Breadth-first (BFS), Iterative Deepening (IDS)
- COST-based Algorithms: Depth-First Branch-and-Bound, Best First Search, Best-First Iterative Deepening
- Widely Used Algorithms: A* and IDA* (Or Graphs), AO* (And/Or Graphs), Alpha-beta Pruning (Game-Trees)

BASIC ALGORITHMS in OR GRAPHS: DFS

1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, put n_i in the **FRONT** of OPEN List
   d) For each n_i already in OPEN or CLOSED decide based on cost of the paths
5. [Continue] Go to Step 2

OPEN = { s }

CLOSED = { }



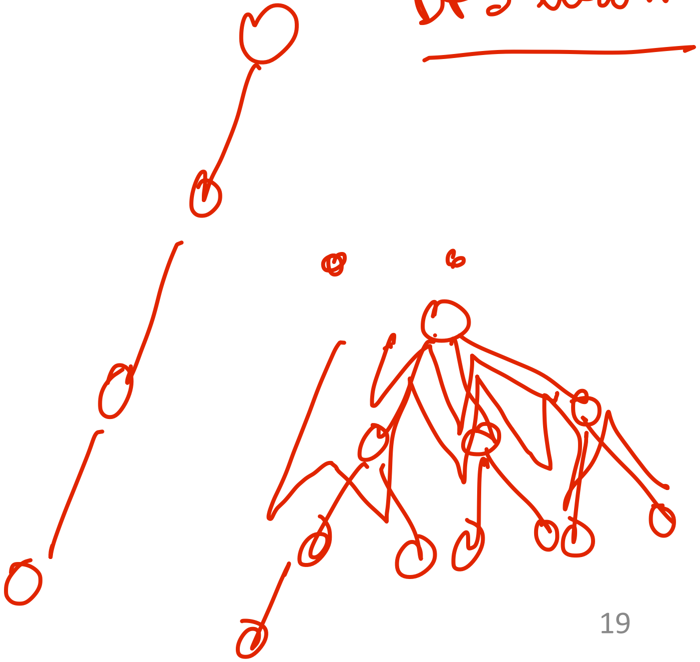
stack data structure

BASIC ALGORITHMS in OR GRAPHS: IDS

Iterative Deepening Search

1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, put n_i in the <u>FRONT</u> of OPEN List
   d) For each n_i already in OPEN or CLOSED decide based on cost of the paths
5. [Continue] Go to Step 2

DFS iteratively



Algorithm IDS Performs DFS Level by Level Iteratively (DFS (1), DFS (2), ……. and so on)

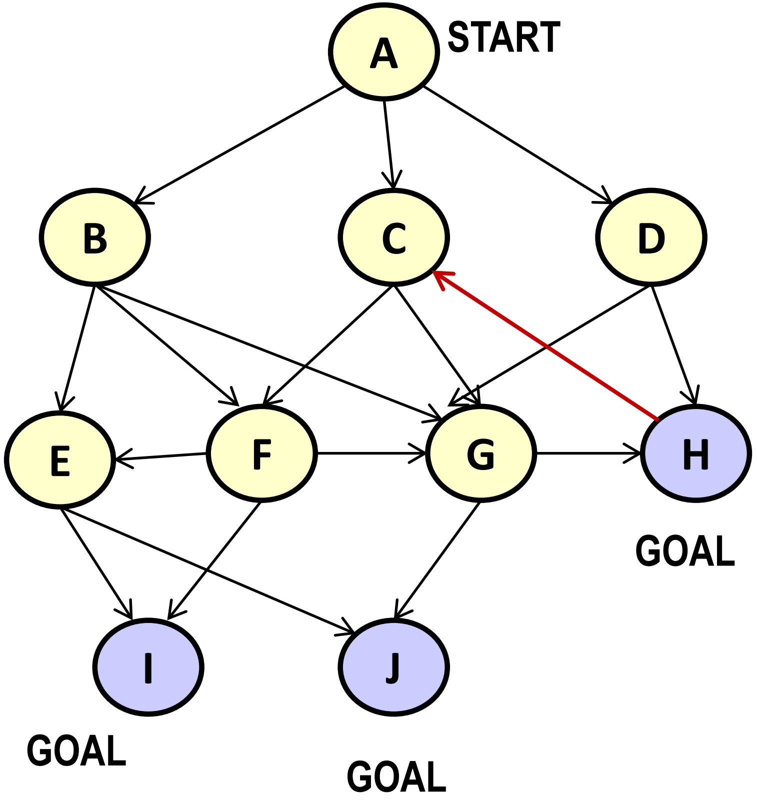
BASIC ALGORITHMS in OR GRAPHS: BFS

1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, put n_i in the **END** of OPEN List
   d) For each n_i already in OPEN or CLOSED decide based on cost of the paths
5. [Continue] Go to Step 2

Breadth-first Search

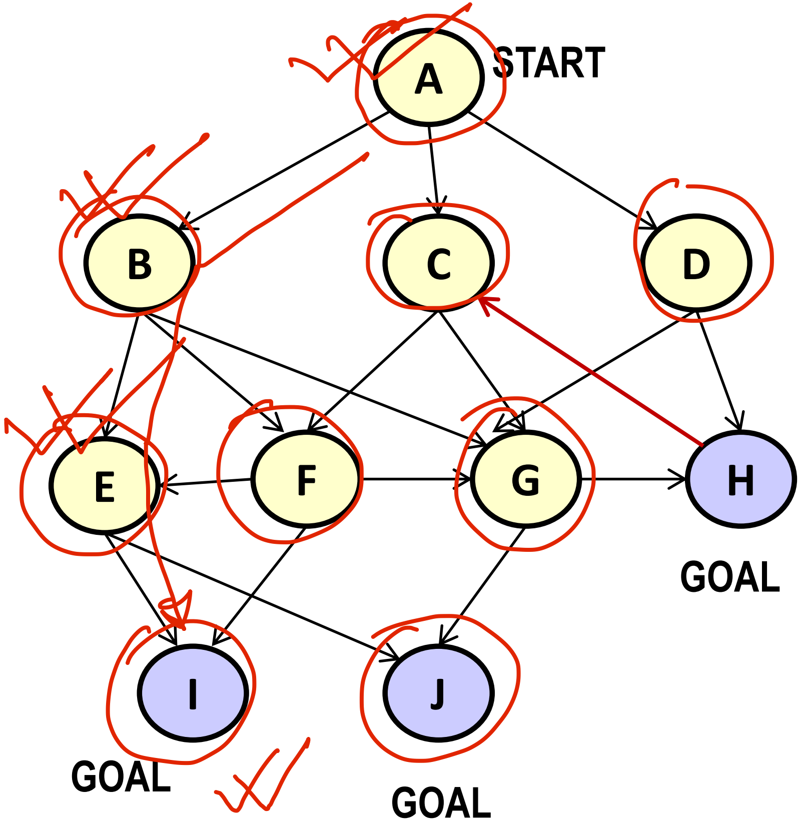
Open List is treated as a Queue

EXAMPLE: SEARCHING A STATE SPACE GRAPH



- DEPTH-FIRST SEARCH (DFS)
- BREADTH-FIRST SEARCH (BFS)
- ITERATIVE DEEPENDING SEARCH (IDS)
- PROPERTIES
  - SOLUTION GUARANTEES
  - MEMORY REQUIREMENTS

EXAMPLE: SEARCHING A STATE SPACE GRAPH



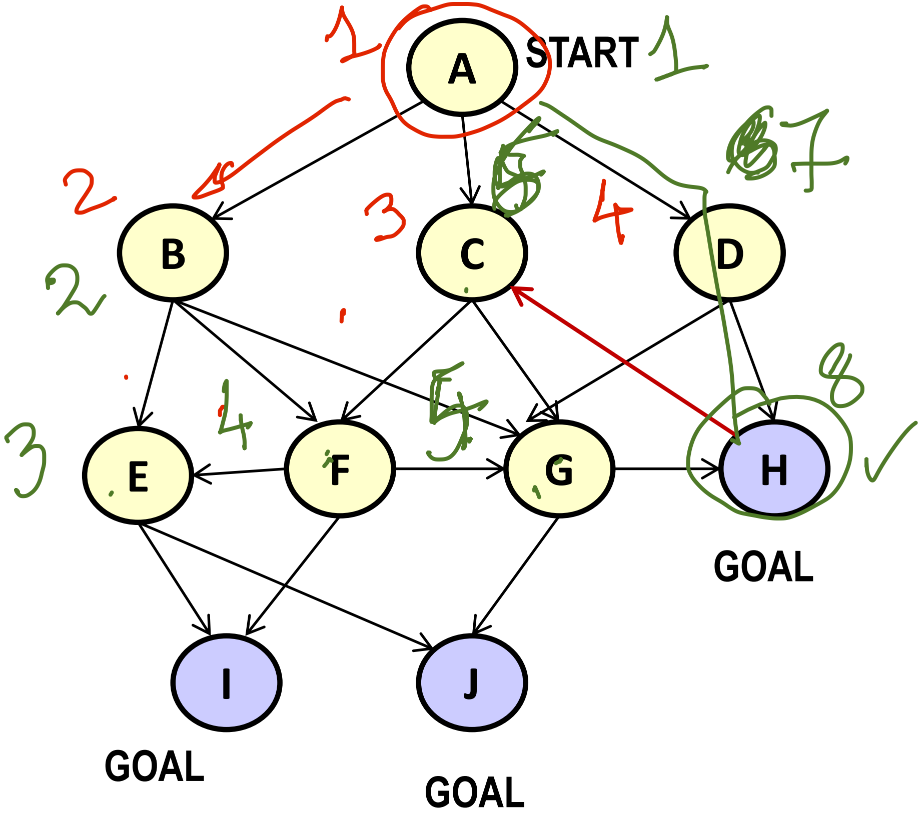
DEPTH-FIRST SEARCH:

1. OPEN ={A}, CLOSED = {}
2. OPEN = {B,C,D}, CLOSED = {A}
3. OPEN = {E,F,G,C,D}, CLOSED = {A,B}
4. OPEN = {I,J,F,G,C,D}, CLOSED = {A,B,E}
5. Goal Node I Found. Can Terminate with Path from A to I or may Continue for more Goal nodes if minimum length or cost is a criteria

DFS MAY NOT TERMINATE IF THERE IS AN INFINITE DEPTH PATH EVEN IF THERE IS A GOAL NODE AT FINITE DEPTH

DFS HAS LOW MEMORY REQUIREMENT

EXAMPLE: SEARCHING A STATE SPACE GRAPH



ITERATIVE DEEPENING SEARCH:

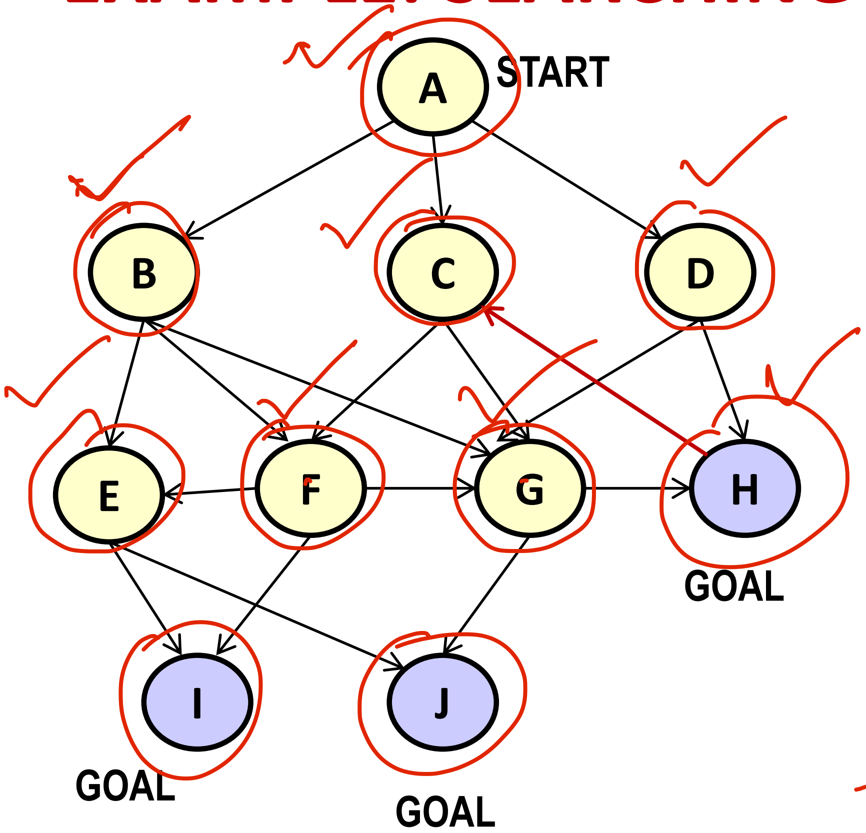
1. PERFORM DFS TILL LENGTH 1. NO SOLUTION FOUND
2. PERFORM DFS TILL LEVEL 2. GOAL NODE H REACHED.
3. Can Terminate with Path from A to H. This is guaranteed to be the minimum length path.

IDS GUARANTEES SHORTEST LENGTH PATH TO GOAL

IDS MAY RE-EXPAND NODES MANY TIMES

IDS HAS LOWER MEMORY REQUIREMENT THAN BFS

EXAMPLE: SEARCHING A STATE SPACE GRAPH

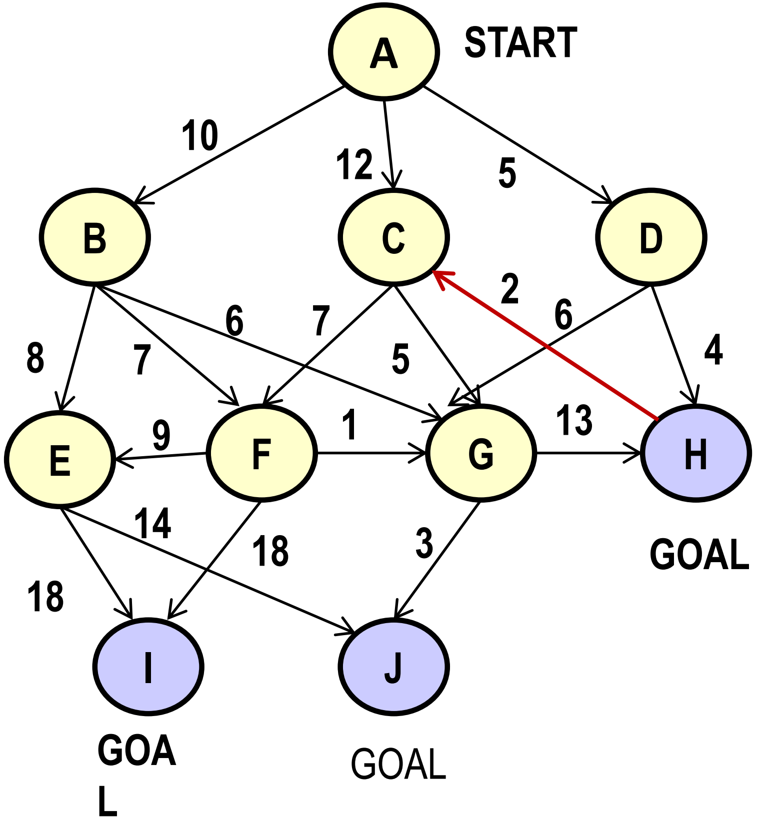


BREADTH-FIRST SEARCH:

1. OPEN ={A}, CLOSED = {}
2. OPEN = {B,C,D}, CLOSED = {A}
3. OPEN = {C,D,E,F,G}, CLOSED = (A,B}
4. OPEN = {D,E,F,G}, CLOSED = {A,B,C}
5. OPEN = {E,F,G,H}. CLOSED = {A,B,C,D}
6. OPEN = {F,G,H,I,J}, CLOSED = {A,B,C,D,E}
7. OPEN = {G,H,I,J}, CLOSED = {A,B,C,D,E,F}
8. OPEN = {H,I,J}, CLOSED = {A,B,C,D,E,F,G}
9. Goal Node H Found. Can Terminate with Path from A to H. This is guaranteed to be the minimum length path.

BFS GUARANTEES SHORTEST LENGTH PATH TO GOAL BUT HAS HIGHER MEMORY REQUIREMENT

SEARCHING STATE SPACE GRAPHS WITH EDGE COSTS



Find the best cost path from start to goal

n , g(n)

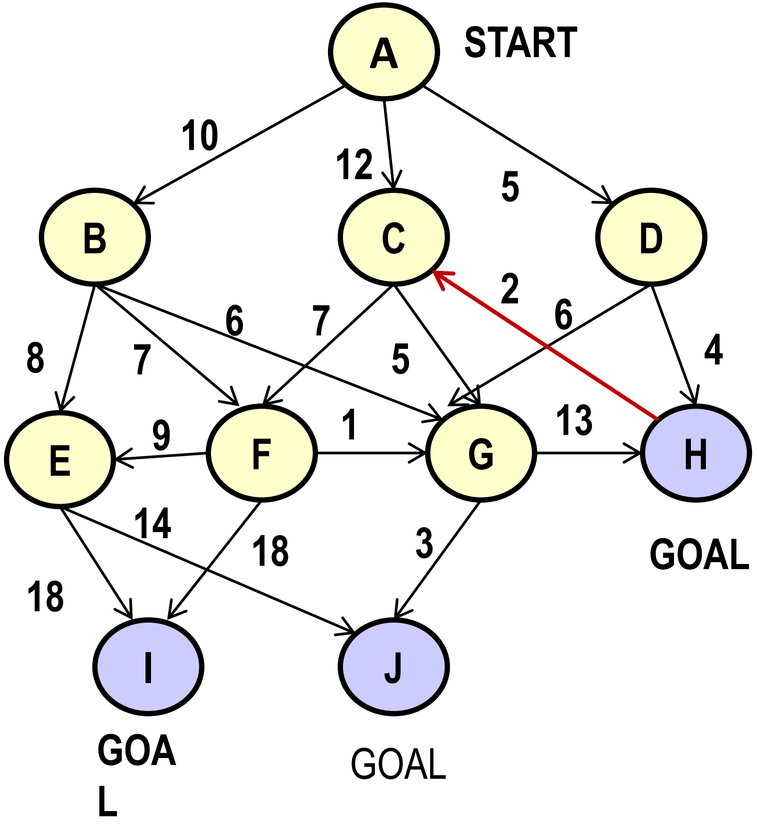
↳ cost of path (current path) from s to n

MODIFYING BASIC ALGORITHMS TO INCORPORATE COSTS

1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules
   b) Put n in LIST CLOSED
   c) For each n_i, not already in OPEN or CLOSED List, put n_i in the FRONT (for DFS) / END (for BFS) of OPEN List
   d) For each n_i already in OPEN or CLOSED decide based on cost of the paths
5. [Continue] Go to Step 2

Algorithm IDS Performs DFS Level by Level Iteratively (DFS (1), DFS (2), ……. and so on)

NEXT: SEARCHING STATE SPACE GRAPHS WITH EDGE COSTS



- COST ORDERED SEARCH:
  - DFBB
  - Best First Search,
  - Best First IDS
  - Use of HEURISTIC Estimates: Algorithm A* (Or Graphs), AO* (And/Or Graphs)
- PROPERTIES
  - SOLUTION GUARANTEES
  - MEMORY REQUIREMENTS

Heuristic Search

Thank you