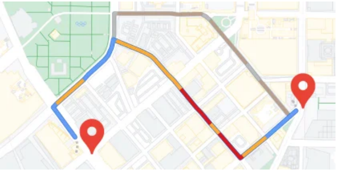
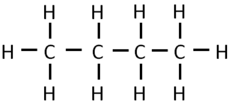
CS5201: Advanced Artificial Intelligence

State Space Search

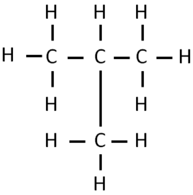
Arijit Mondal
Dept of Computer Science and Engineering
Indian Institute of Technology Patna
www.iitp.ac.in/~arijit/

Complex problems & solutions-1





Butane C_4H_{10}



Iso-Butane C_4H_{10}

$$\int \frac{x^4}{(1-x^2)^{\frac{5}{2}}} dx$$

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

Automated problem solving by search

- Generalized techniques for solving large class of complex problems
- Problem statement is the input and solution is the output (sometime problem specific algorithm / method could be the output)
- AI search based problem formulation requires following steps broadly
  - Configuration or state
  - Constraints or definitions of valid configuration
  - Rules for change of state and their outcomes
  - Initial state or start configurations
  - Goal satisfying configurations
  - An implicit state space
  - Valid solutions from start to goal in that state space
  - General algorithms which search for solutions in this state space
- Challenges: Size of implicit state space, Capturing domain knowledge, Intelligent algorithms that work in reasonable time and memory, Handling incompleteness and uncertainty

State space modeling

- **State or configurations**
  - **A set of variables which define a state or configuration**
  - **Domains for every variable and constraints among variables to define a valid configuration**

State space modeling

- **State or configurations**
  - **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 modeling

- State or configurations
  - 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.

State space modeling

- State or configurations
  - 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.
- Start and goal states

State space modeling

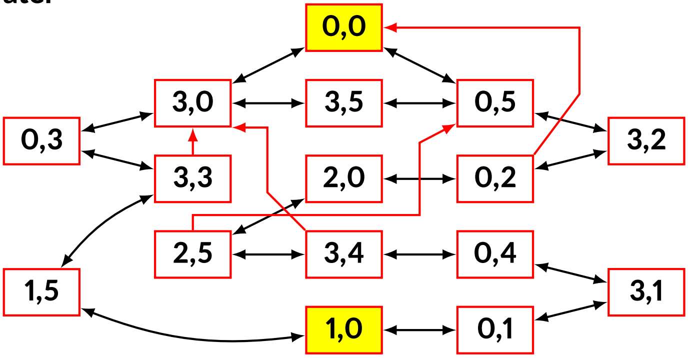
- State or configurations
  - A set of variables which define a state or configuration
  - Domains for every variable and constraints among variables to define a valid configuration
- State transformation rules or moves
  - A set of rules which define which are the valid set of next state of a given 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.
- Start and goal states
- Solutions, costs
  - Depending on the problem formulation, it can be a PATH from Start to Goal or a Sub-graph of And-ed Nodes

State space modeling

- State or configurations
  - 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.
- Start and goal states
- Solutions, 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 liter and J2 of volume 5 liter. 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) liter of water
- State-space modeling
  - State definition: (J1,J2)
  - Rules:
    - Fill(J1), Fill(J2)
    - Empty(J1), Empty(J2)
    - Pour(J1,J2), Pour(J2,J1)
  - Start state: (0,0)
  - Goal state: (1,0)



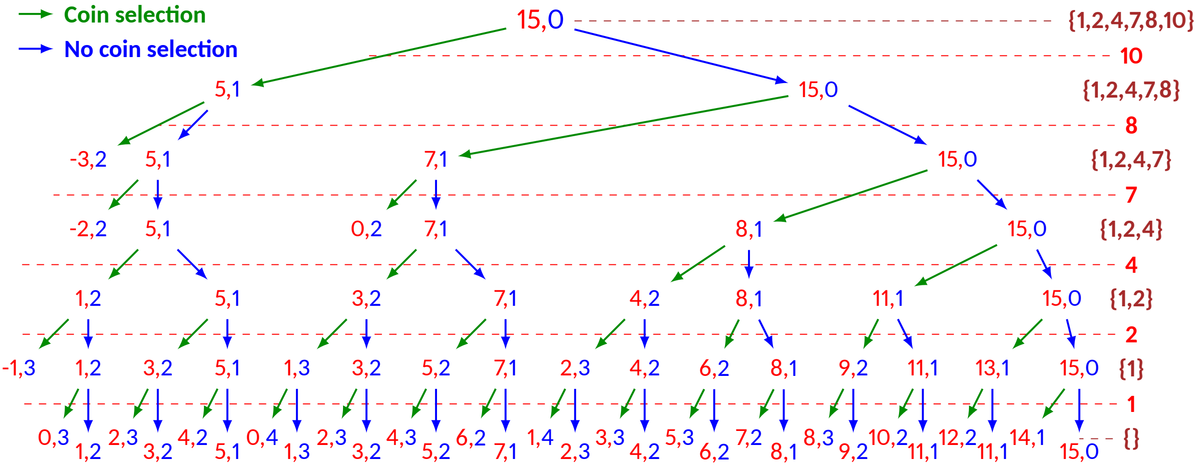
Partial state-space

Coin change: State-space

- **Given a set of coins C, what is the minimum number coins required to provide sum S?**
- **Example:** $C = \{1, 2, 4, 7, 8, 10\}, S = 15$

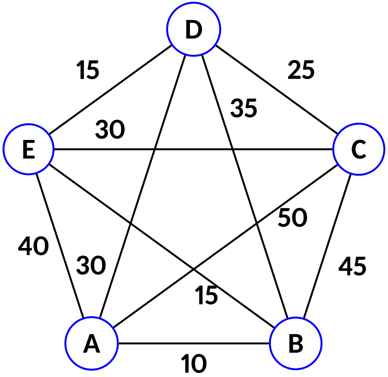
Coin change: State-space

- **Given a set of coins C, what is the minimum number coins required to provide sum S?**
- **Example:** $C = \{1, 2, 4, 7, 8, 10\}, S = 15$

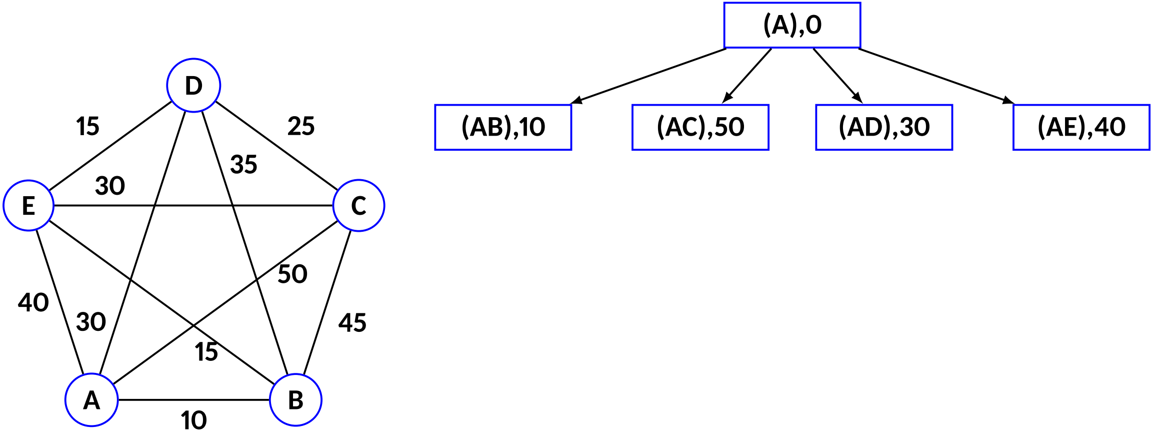


OR Graph: Travelling salesperson problem

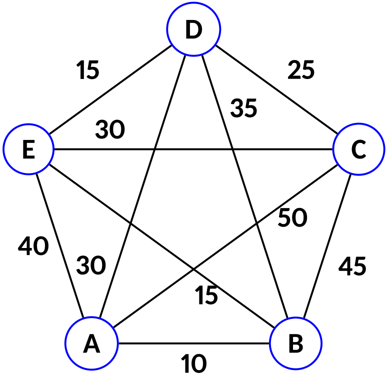
(A),0

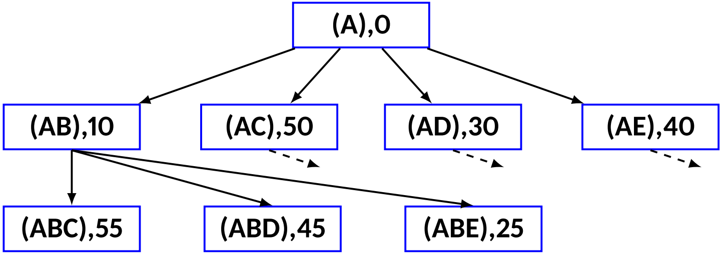


OR Graph: Travelling salesperson problem

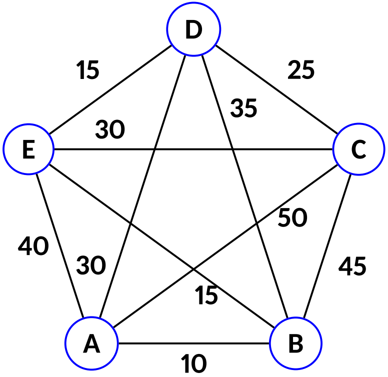


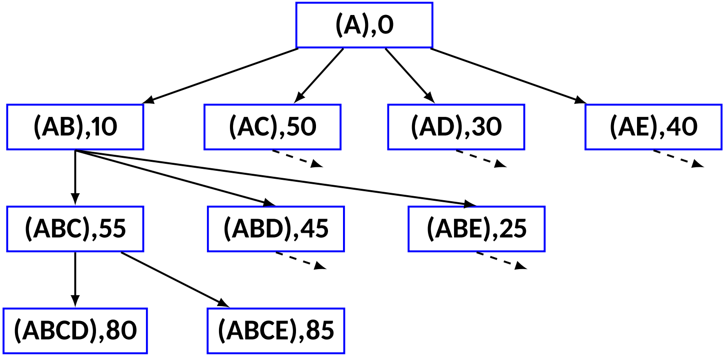
OR Graph: Travelling salesperson problem



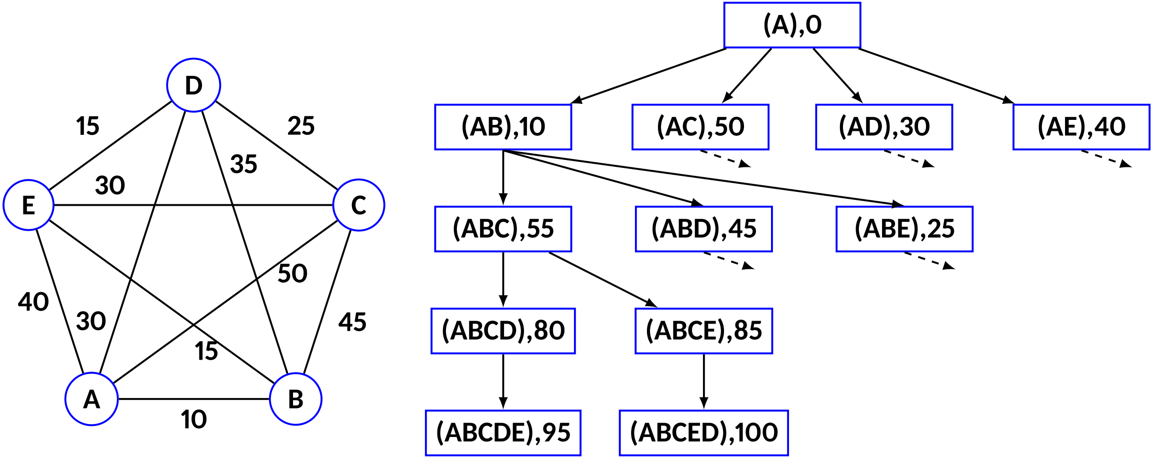


OR Graph: Travelling salesperson problem

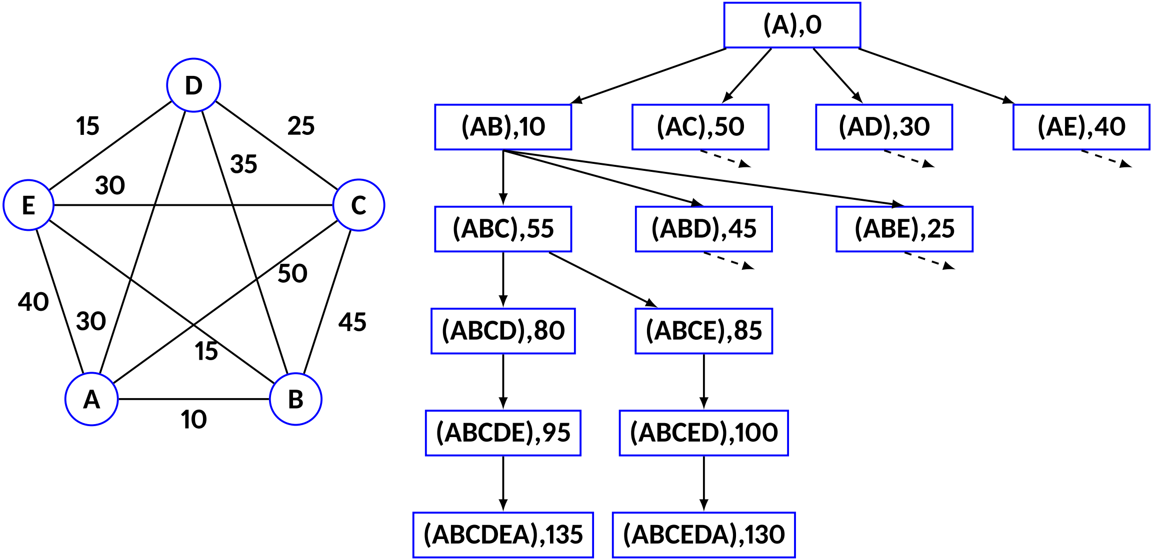




OR Graph: Travelling salesperson problem



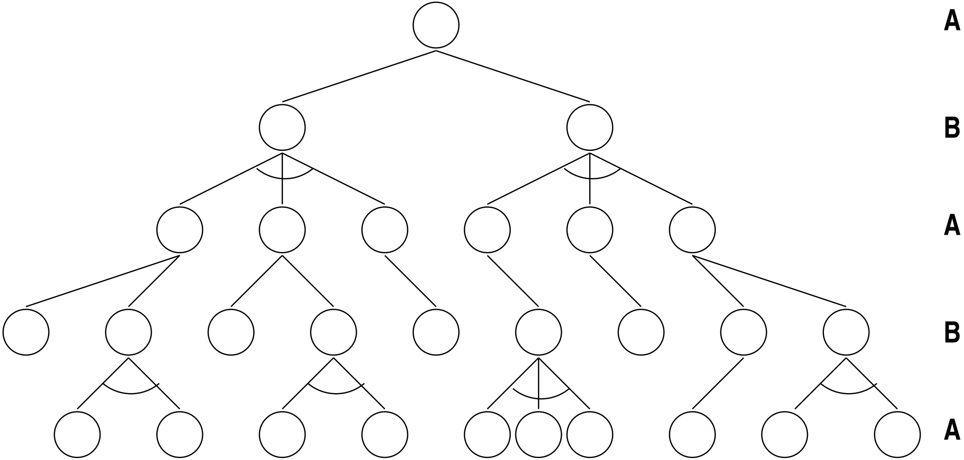
OR Graph: Travelling salesperson problem



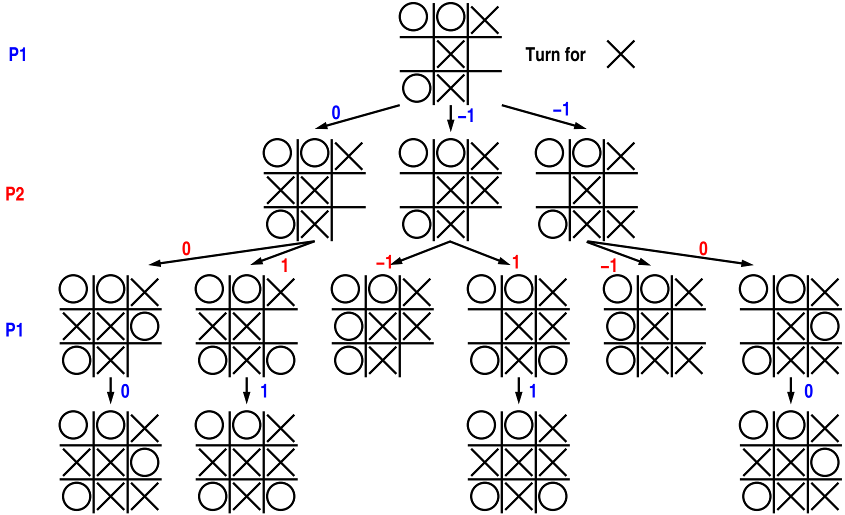
Modelling AND/OR graphs

- OR nodes are ones for which one has a choice
- The AND nodes could be compositional (sum, product, min, max, etc., depending on the subproblems are composed)
  - Adversarial — where the other parties have choice, usually in games
  - Probabilistic — environmental actions

AND/OR graphs



Adversarial AND/OR graphs



Compositional AND/OR graph

- **Let A and B be two matrices of size $p \times q$ and $q \times r$. Therefore, to determine $A \times B$ we need to perform $p \times q \times r$ number of multiplications**
- **Let A_1 be a 10×20 matrix, A_2 be a 20×5 and A_3 be a 5×50 matrix**
- **Then the number of computation**
  - $(A_1 \times A_2) \times A_3$

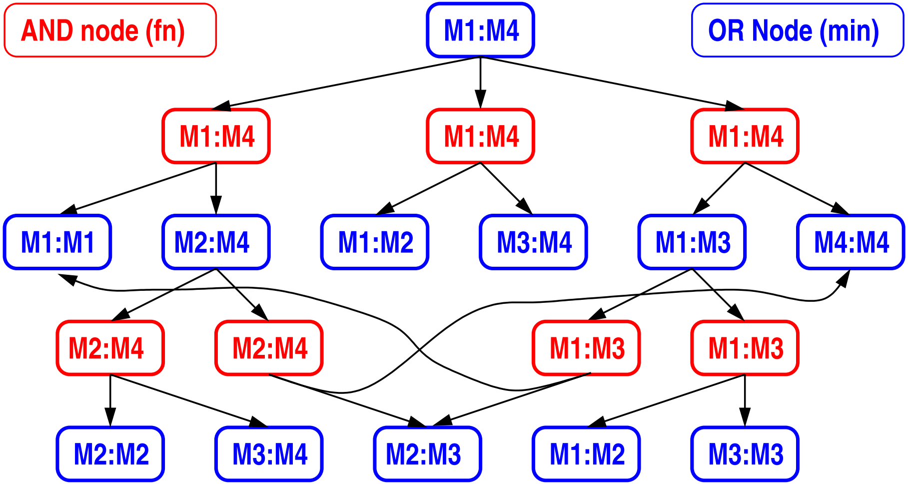
Compositional AND/OR graph

- **Let A and B be two matrices of size $p \times q$ and $q \times r$. Therefore, to determine $A \times B$ we need to perform $p \times q \times r$ number of multiplications**
- **Let A_1 be a 10×20 matrix, A_2 be a 20×5 and A_3 be a 5×50 matrix**
- **Then the number of computation**
  - $(A_1 \times A_2) \times A_3 - (10 \times 20 \times 5) + (10 \times 5 \times 50) = 3500$
  - $A_1 \times (A_2 \times A_3)$

Compositional AND/OR graph

- Let A and B be two matrices of size $p \times q$ and $q \times r$. Therefore, to determine $A \times B$ we need to perform $p \times q \times r$ number of multiplications
- Let A_1 be a 10×20 matrix, A_2 be a 20×5 and A_3 be a 5×50 matrix
- Then the number of computation
  - $(A_1 \times A_2) \times A_3$ **–** $(10 \times 20 \times 5) + (10 \times 5 \times 50) = 3500$
  - $A_1 \times (A_2 \times A_3)$ **–** $(20 \times 5 \times 50) + (10 \times 20 \times 50) = 15000$

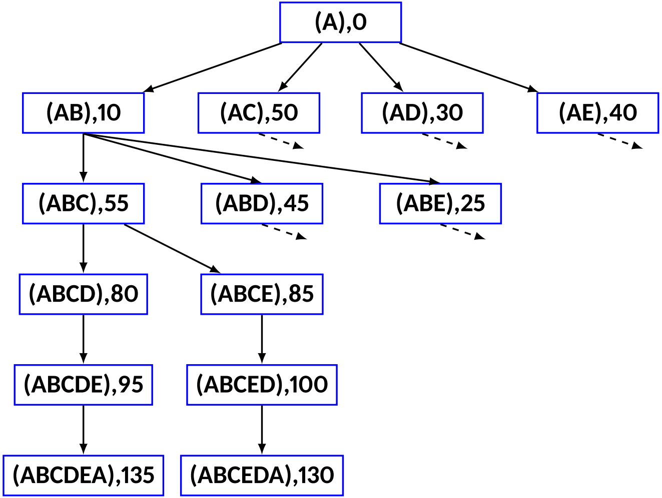
MCM: AND/OR graphs



M1(M2(M3M4)) = (M1M2)(M3M4) = ((M1M2)M3))M4 = (M1(M2M3))M4 = M1((M2M3)M4)

Searching implicit graph

- 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 INTELLIGENTLY 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 graph: Algorithms

- **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 $n_1, n_2, \ldots, n_k$, of node n, based on the State Transformation Rules
   b. Put n in CLOSED List
   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

Basic algorithms in OR graphs: IDS

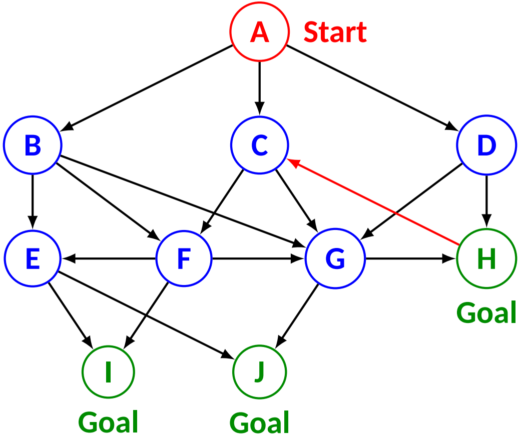
1. **[Initialize]** Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. **[Select]** Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. **[Goal Test]** If n is Goal, then decide on Termination or Continuation / Cost Updation
4. **[Expand]**
   a. Generate the successors $n_1, n_2, \ldots, n_k$, 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

- IDS performs DFS level wise manner (DFS(1), DFS(2), ...)

Basic algorithms in OR graphs: BFS

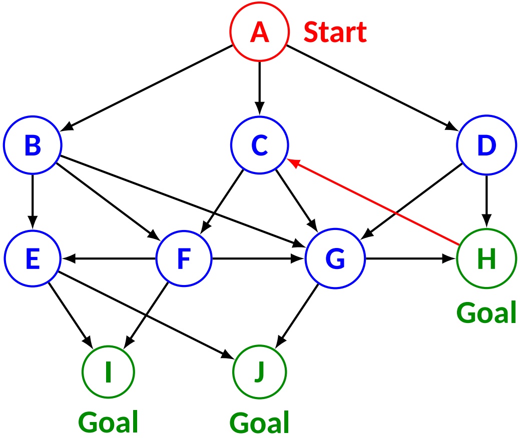
1. [Initialize] Initially the OPEN List contains the Start Node s. CLOSED List is Empty.
2. [Select] Select the first Node n on the OPEN List. If OPEN is empty, Terminate
3. [Goal Test] If n is Goal, then decide on Termination or Continuation / Cost Updation
4. [Expand]
   a. Generate the successors $n_1, n_2, \ldots, n_k$, 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

Searching state space graph



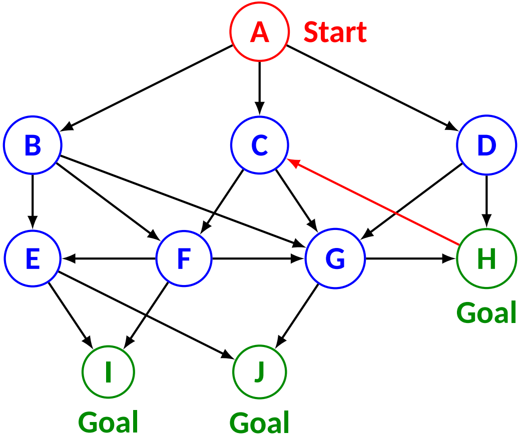
- Depth-first search
- Breadth-first search
- Iterative deepening search

Searching state space graph: DFS



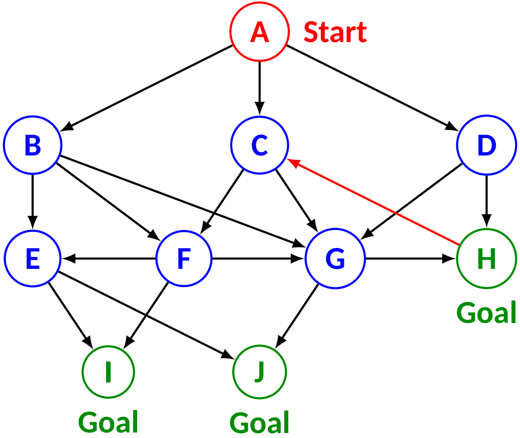
Step	OPEN	CLOSED

Searching state space graph: DFS



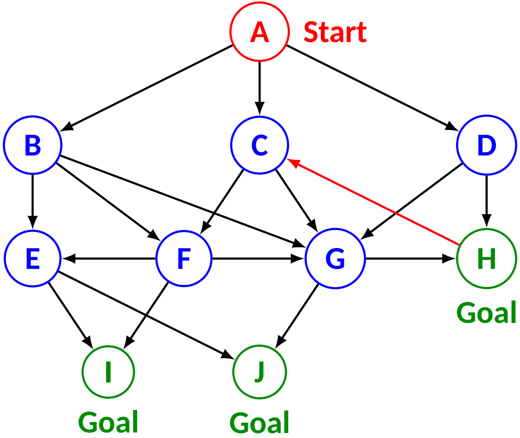
Step	OPEN	CLOSED
1	A	{}

Searching state space graph: DFS



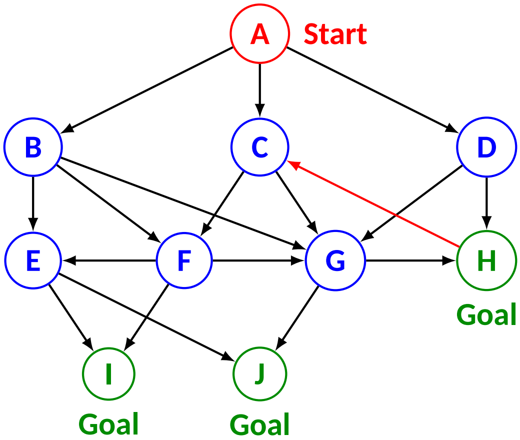
Step	OPEN	CLOSED
1	A	{}
2	BCD	A

Searching state space graph: DFS



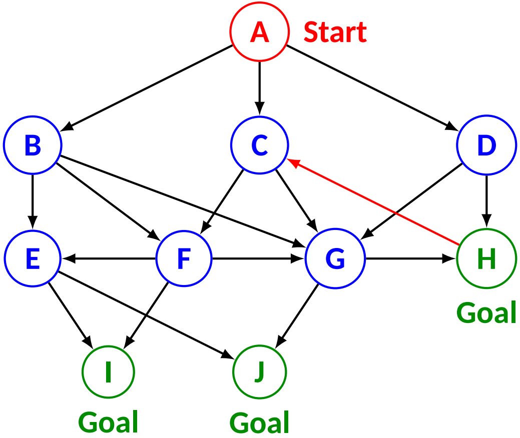
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	EFGCD	AB

Searching state space graph: DFS



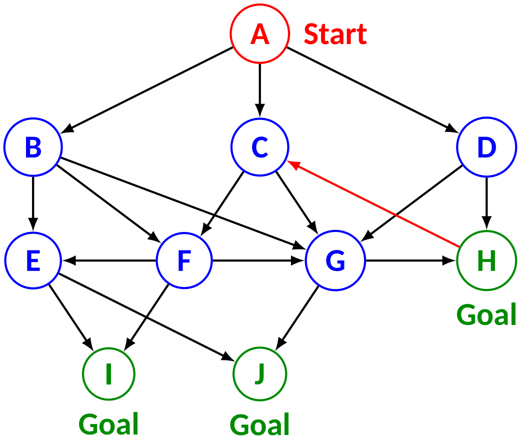
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	EFGCD	AB
4	IJFGCD	ABE

Searching state space graph: DFS



Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	EFGCD	AB
4	IJFGCD	ABE
5	I is goal node, can terminate	

Searching state space graph: DFS



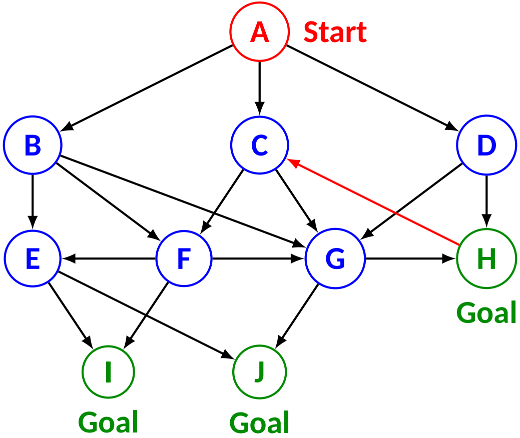
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	EFGCD	AB
4	IJFGCD	ABE
5	I is goal node, can terminate	

Search can 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.

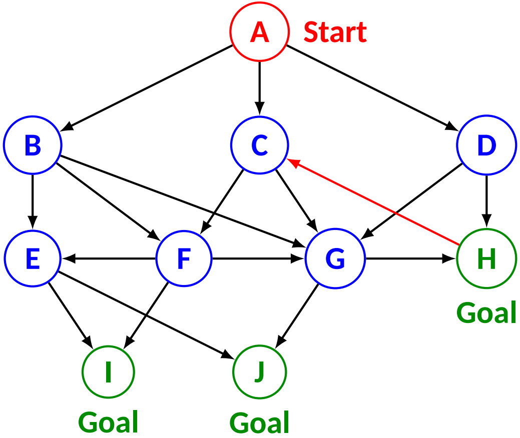
DFS has low memory overhead.

Searching state space graph: IDS



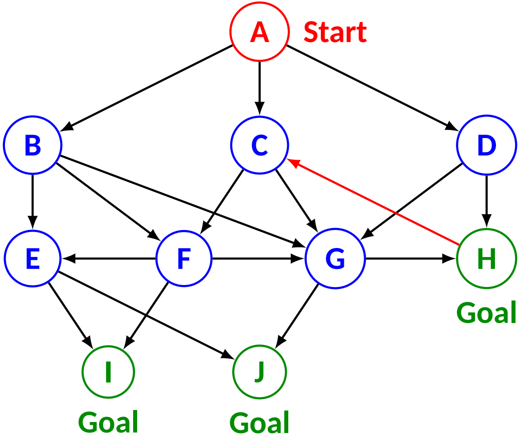
Step	Outcome

Searching state space graph: IDS



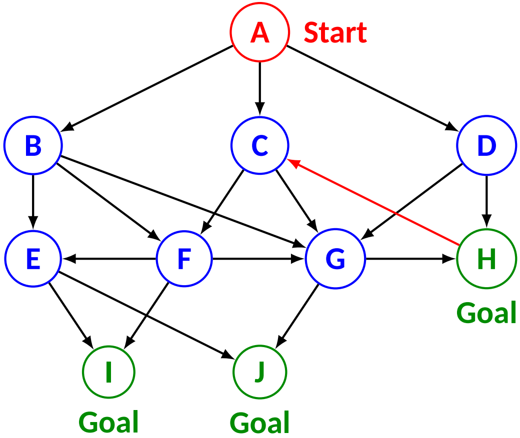
Step	Outcome
1	DFS(L=1) – No solution

Searching state space graph: IDS



Step	Outcome
1	DFS(L=1) – No solution
2	DFS(L=2) – Goal node H reached

Searching state space graph: IDS



Step	Outcome
1	DFS(L=1) – No solution
2	DFS(L=2) – Goal node H reached
3	Can terminate with path from A to H

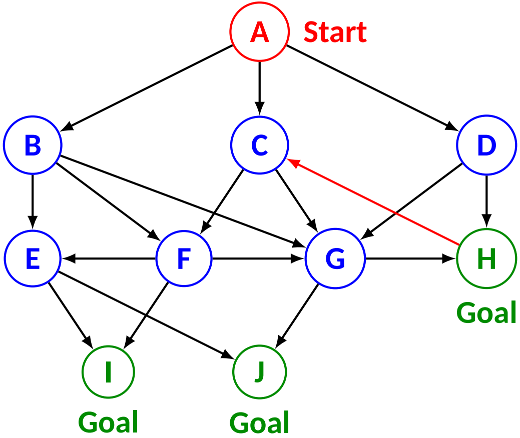
This is guaranteed to be the minimum length path.

IDS guarantees shortest length path to goal.

IDS may re-expand nodes many times.

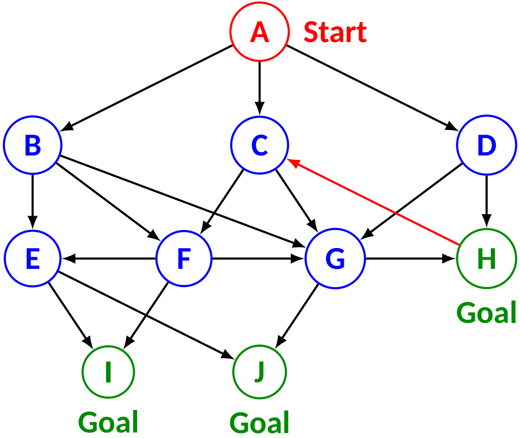
IDS has lower memory requirement than BFS.

Searching state space graph: BFS



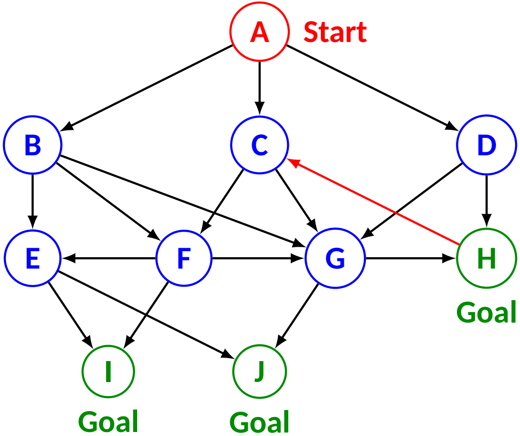
Step	OPEN	CLOSED

Searching state space graph: BFS



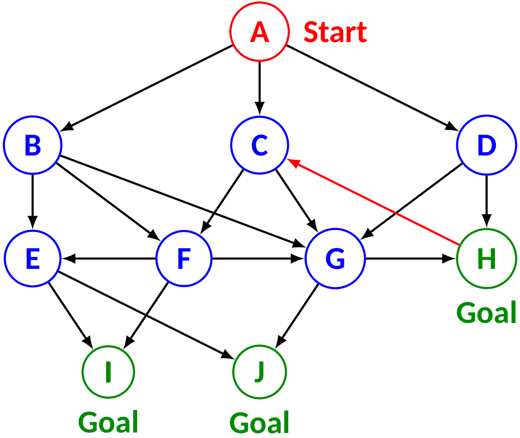
Step	OPEN	CLOSED
1	A	{}

Searching state space graph: BFS



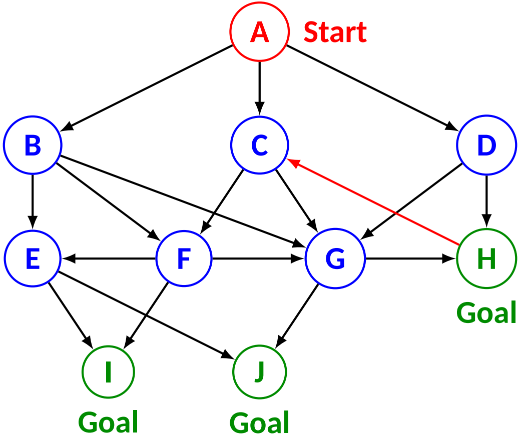
Step	OPEN	CLOSED
1	A	{}
2	BCD	A

Searching state space graph: BFS



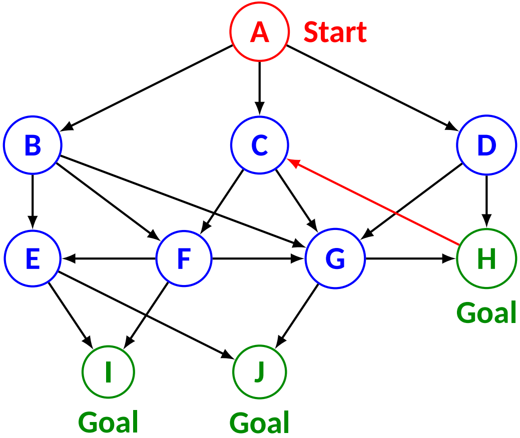
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB

Searching state space graph: BFS



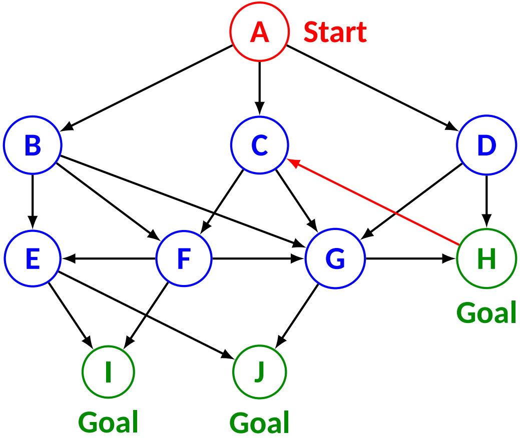
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC

Searching state space graph: BFS



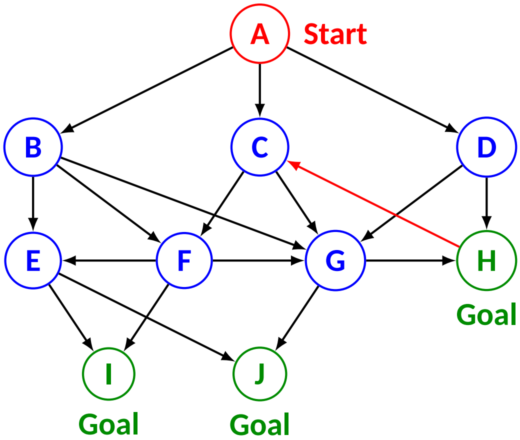
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD

Searching state space graph: BFS



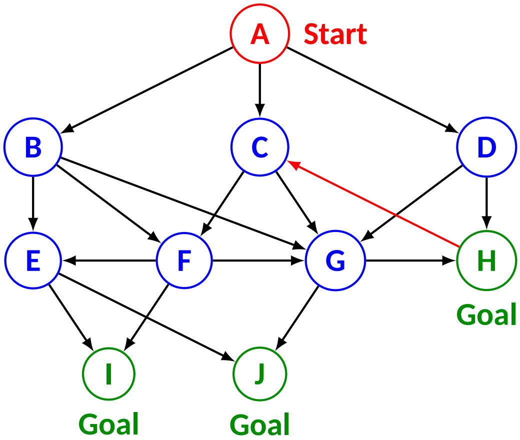
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD
6	FGHIJ	ABCDE

Searching state space graph: BFS



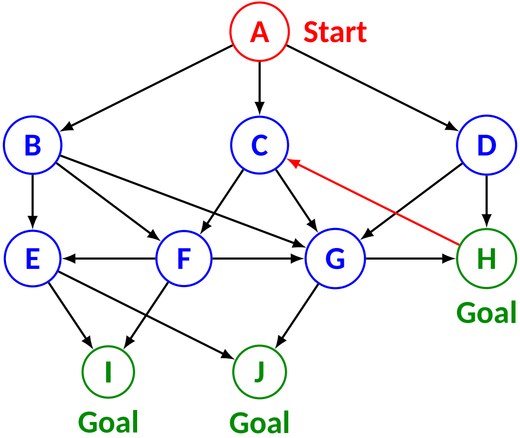
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD
6	FGHIJ	ABCDE
7	GHIJ	ABCDEF

Searching state space graph: BFS



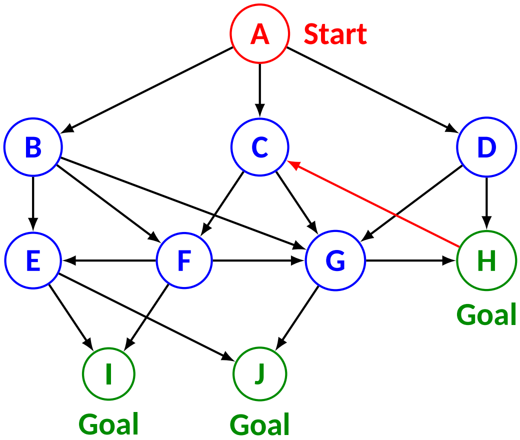
Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD
6	FGHIJ	ABCDE
7	GHIJ	ABCDEF
8	HIJ	ABCDEFG

Searching state space graph: BFS



Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD
6	FGHIJ	ABCDE
7	GHIJ	ABCDEF
8	HIJ	ABCDEFG
9	Goal node H found	

Searching state space graph: BFS



Step	OPEN	CLOSED
1	A	{}
2	BCD	A
3	CDEFG	AB
4	DEFG	ABC
5	EFGH	ABCD
6	FGHIJ	ABCDE
7	GHIJ	ABCDEF
8	HIJ	ABCDEFG
9	Goal node H found	

BFS guarantees shortest length path to goal but has higher memory requirement.

Comparison

- b — **branching factor,** d — **depth of shallowest soln,** m — **maximum depth**
- **Optimality under the assumption of identical cost for all steps**

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?			

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time			

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$
Space			

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$
Space	$O(bm)$	$O(bd)$	$O(b^d)$

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

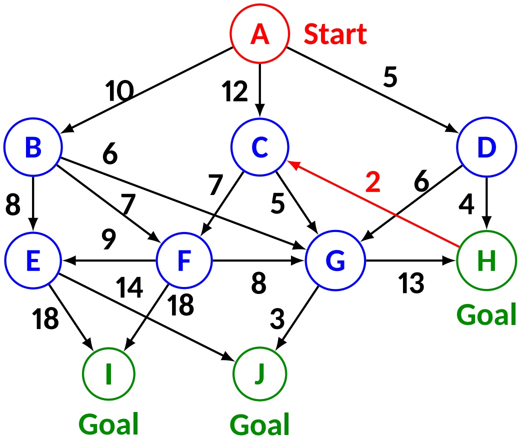
Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$
Space	$O(bm)$	$O(bd)$	$O(b^d)$
Optimal			

Comparison

- b — branching factor, d — depth of shallowest soln, m — maximum depth
- Optimality under the assumption of identical cost for all steps

Criterion	DFS	Iterative Deepening	BFS
Complete?	No	Yes	Yes
Time	$O(b^m)$	$O(b^d)$	$O(b^d)$
Space	$O(bm)$	$O(bd)$	$O(b^d)$
Optimal	No	Yes	Yes

Searching state space graph with edge cost

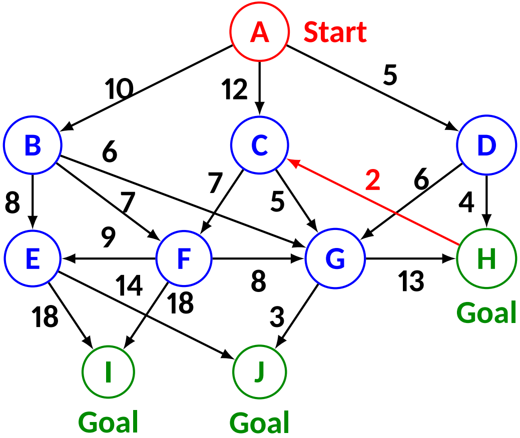


Modifying basic algorithms to incorporate cost

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 $n_1, n_2, \ldots, n_k$, 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)

Searching state space graph with edge cost



Cost ordered search:

- DFBB
- Best first search
- Best first IDS
- Use of heuristic estimates: A*, AO*

Thank you!