www.utm.my

Search Algorithms

Artificial Intelligence

School of Computing

Universiti Teknologi Malaysia

Premier Digital Tech University™

Formulation

- State Space
- Problem Formulation
- Graph Theory
- State Space Mapping

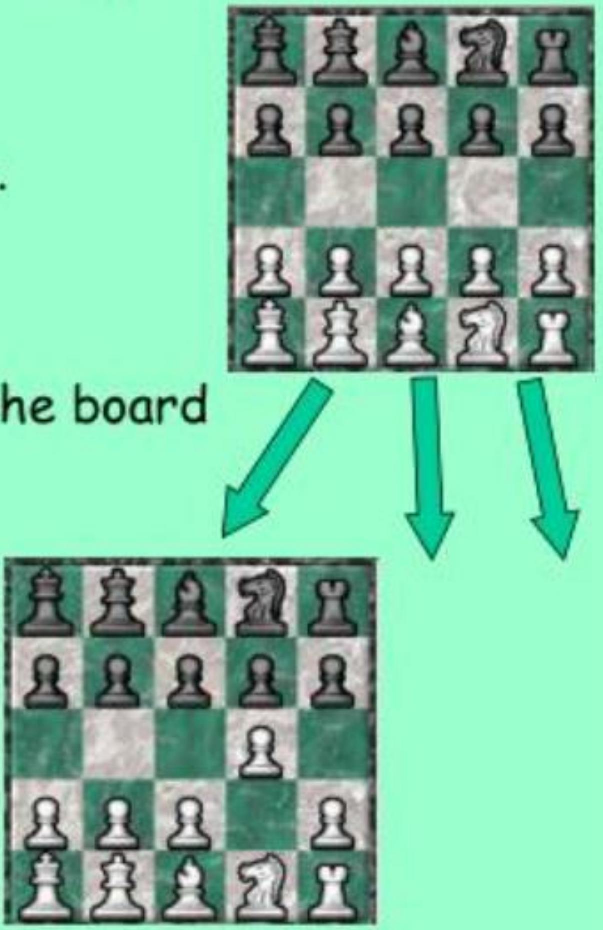
Searching Algorithm in AI

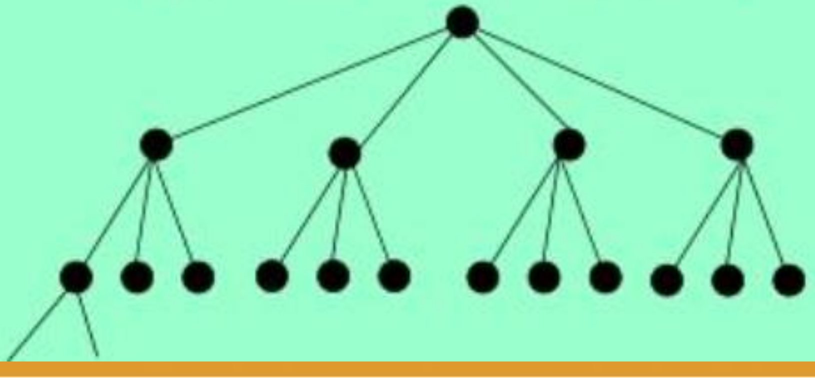
- In Artificial Intelligence, Search techniques are universal problem-solving methods.
- **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem-solving agents are the goal-based agents and use atomic representation.

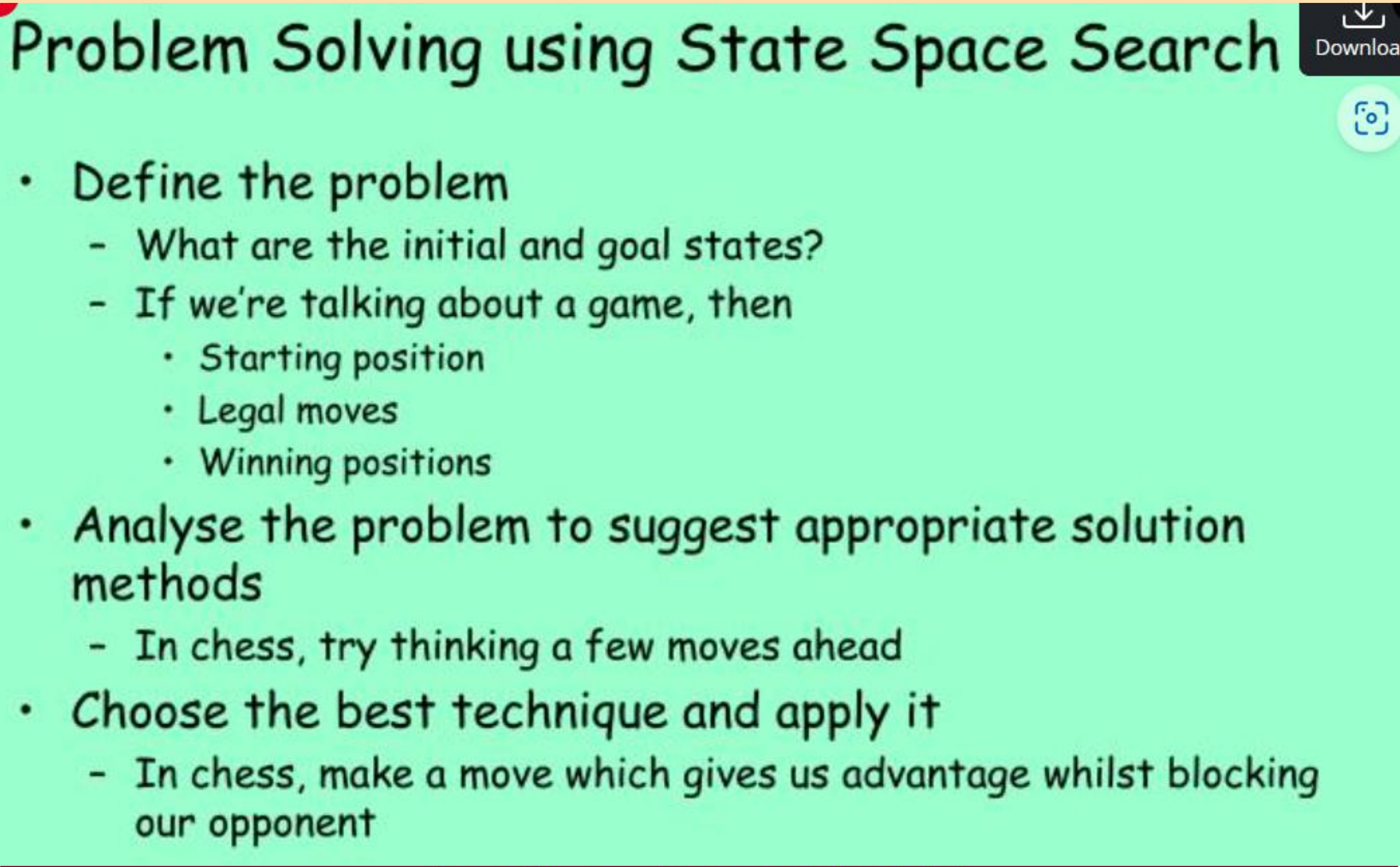
State Space Representation

- To represent a problem as a state-space search
  - Define the states
    - Define the initial and goal states
  - Define the possible operators in any given state.

- Example: Chess
  - State = position; what occupies each square of the board
    - Initial state – traditional starting position
    - Final state – checkmate or draw
  - Operators correspond to legal chess moves





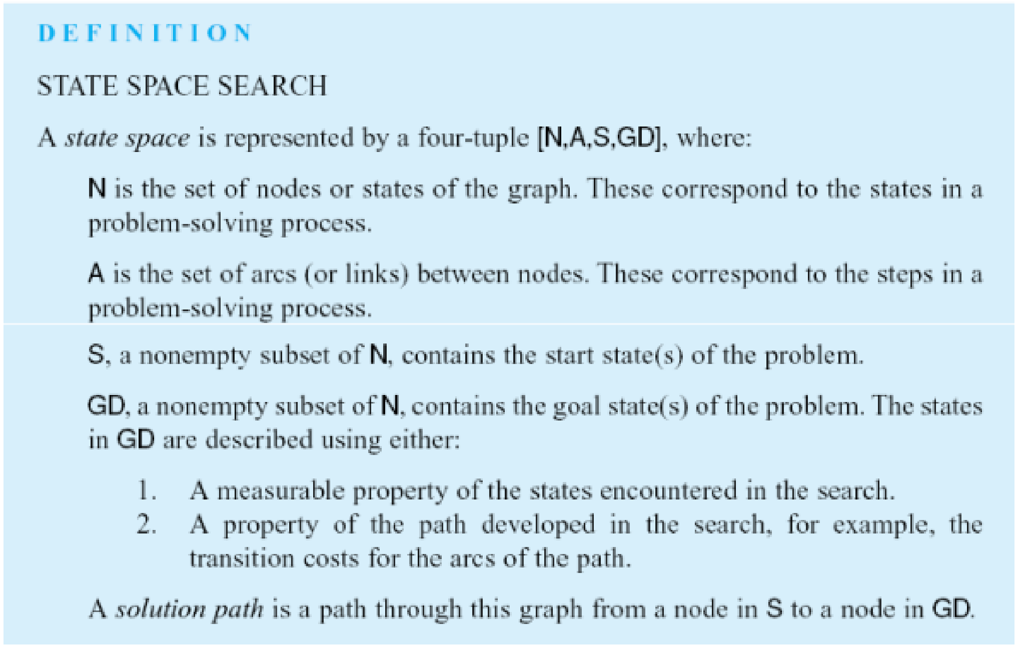


Problem Solving using State Space Search

- Define the problem
  - What are the initial and goal states?
  - If we're talking about a game, then
    - Starting position
    - Legal moves
    - Winning positions
- Analyse the problem to suggest appropriate solution methods
  - In chess, try thinking a few moves ahead
- Choose the best technique and apply it
  - In chess, make a move which gives us advantage whilst blocking our opponent

Problem Solving using State Space Representation

- A state space representation of a problem is a directed graph where the nodes correspond to partial problem solution states, and the arcs correspond to steps in a problem-solving process
- State space search characterizes problem solving as a process of finding a solution path from a start state to a goal state


DEFINITION

STATE SPACE SEARCH

A *state space* is represented by a four-tuple [N,A,S,GD], where:

N is the set of nodes or states of the graph. These correspond to the states in a problem-solving process.

A is the set of arcs (or links) between nodes. These correspond to the steps in a problem-solving process.

S, a nonempty subset of N, contains the start state(s) of the problem.

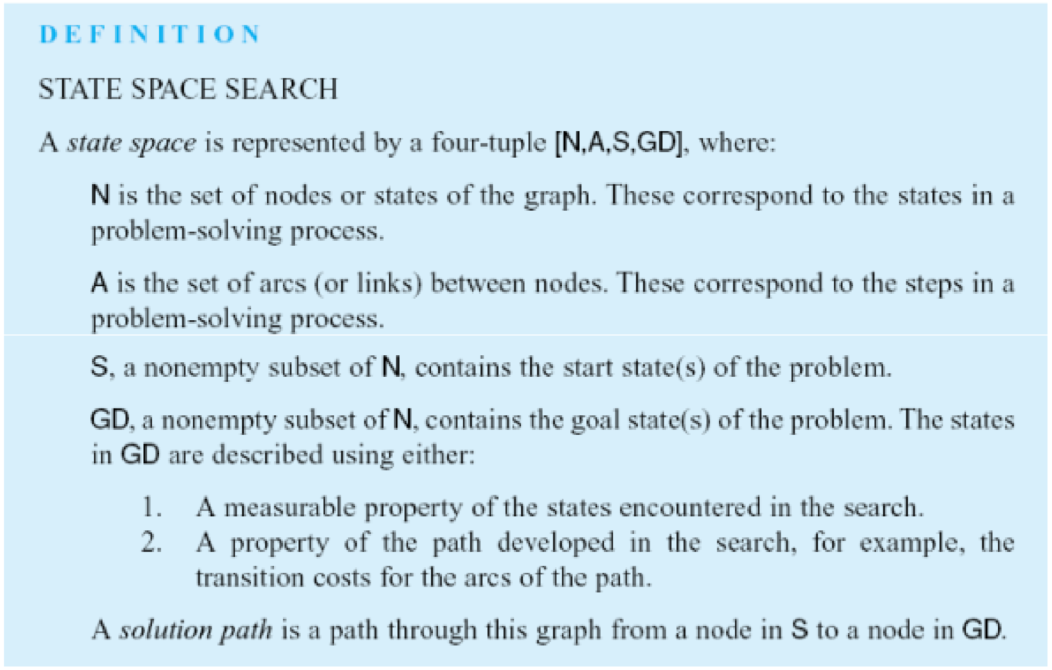
GD, a nonempty subset of N, contains the goal state(s) of the problem. The states in GD are described using either:

1. A measurable property of the states encountered in the search.
2. A property of the path developed in the search, for example, the transition costs for the arcs of the path.

A *solution path* is a path through this graph from a node in S to a node in GD.

What do you need to know?

- Represent a problem as – A STATE SPACE
- Analyze the structure and complexity of problem and search procedures using – GRAPH THEORY
  - Nodes represent discrete states
    Example: Different configuration of game board
  - Path/arcs/links represent action between states
    Example: Movement in a game



DEFINITION

STATE SPACE SEARCH

A *state space* is represented by a four-tuple [N,A,S,GD], where:

N is the set of nodes or states of the graph. These correspond to the states in a problem-solving process.

A is the set of arcs (or links) between nodes. These correspond to the steps in a problem-solving process.

S, a nonempty subset of N, contains the start state(s) of the problem.

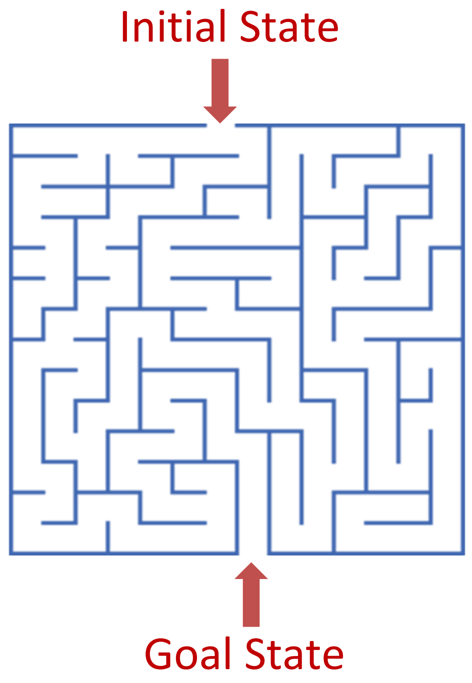
GD, a nonempty subset of N, contains the goal state(s) of the problem. The states in GD are described using either:

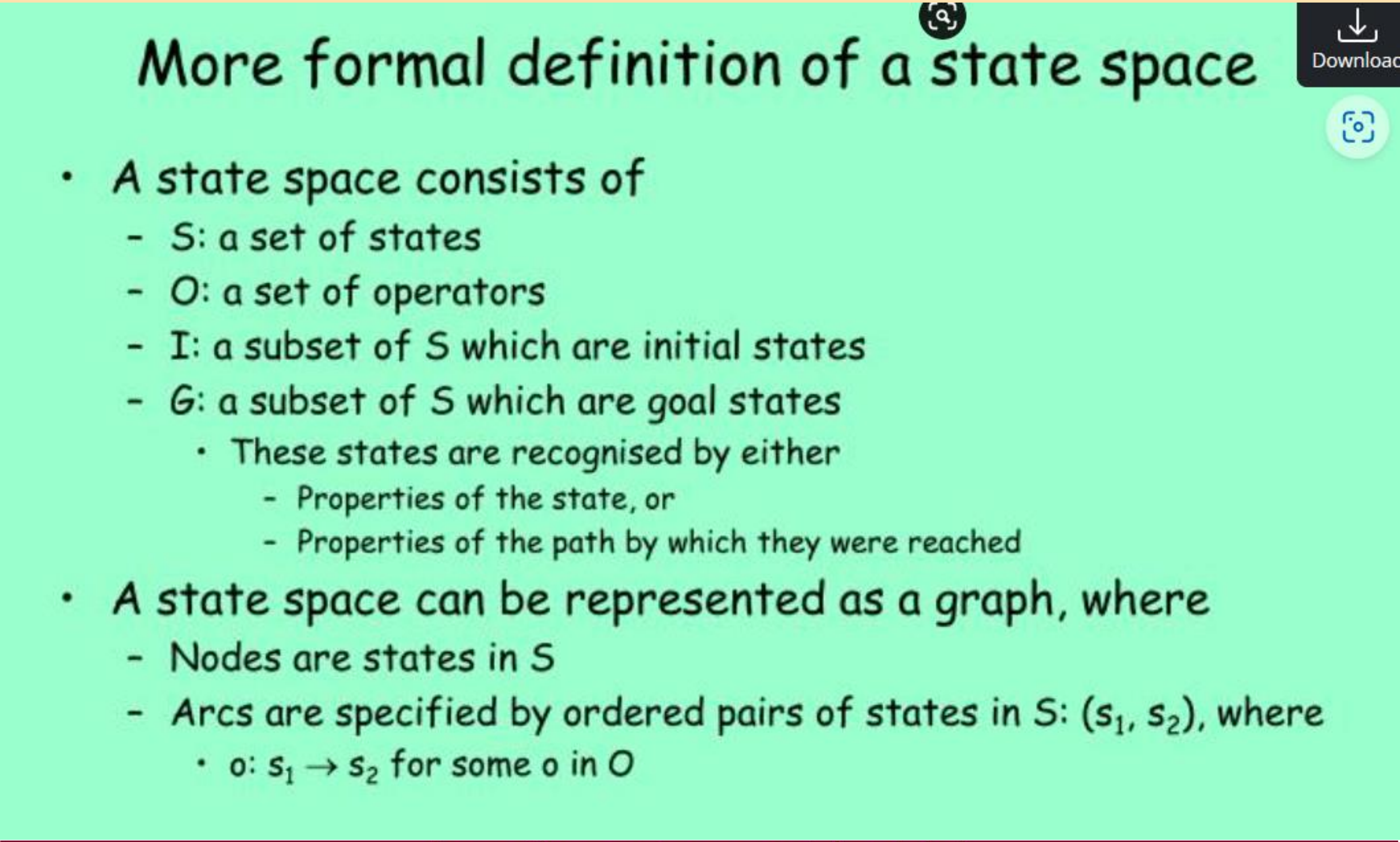
1. A measurable property of the states encountered in the search.
2. A property of the path developed in the search, for example, the transition costs for the arcs of the path.

A *solution path* is a path through this graph from a node in S to a node in GD.

Problem Formulation

- Problem formulation is the process of deciding what actions and states to consider, given a goal
- Problem formulated by specifying four things:
  - Initial state
  - Actions
  - Goal test
  - Path cost
- Solution: sequence of actions leading from initial state to goal state.





More formal definition of a state space

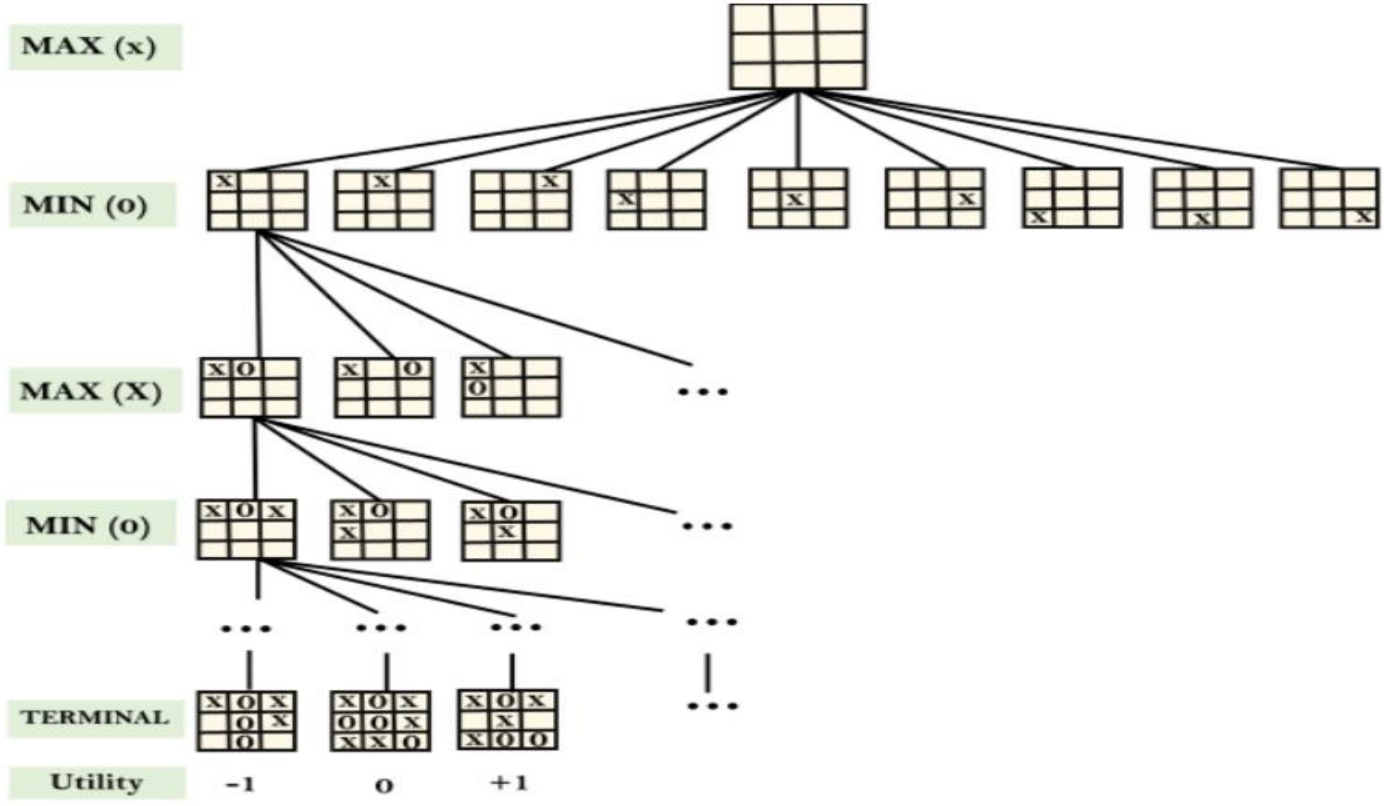
- A state space consists of
  - S: a set of states
  - O: a set of operators
  - I: a subset of S which are initial states
  - G: a subset of S which are goal states
    - These states are recognised by either
      - Properties of the state, or
      - Properties of the path by which they were reached
- A state space can be represented as a graph, where
  - Nodes are states in S
  - Arcs are specified by ordered pairs of states in S: (s_1, s_2), where
    - $o: s_1 \rightarrow s_2$ for some o in O

Example of State Space – Tic Tac Toe Game

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

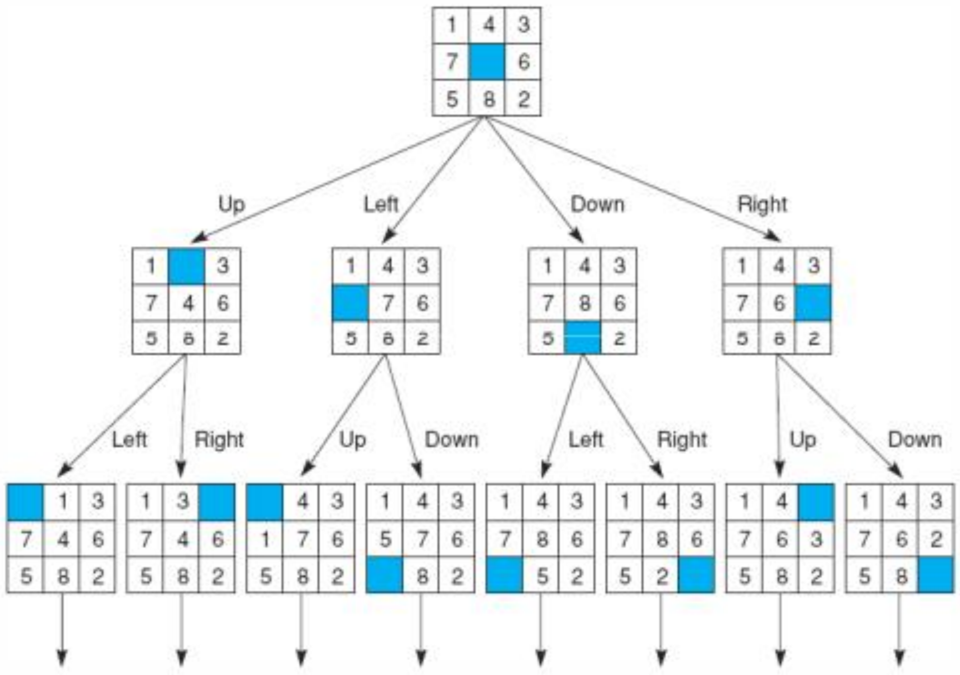
- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.



Example of State Space – The 8-Puzzle

- 8 differently numbered tiles are fitted into 9 spaces. One space is left blank so that tiles can be moved around to form different patterns.
- The goal is to find a series of moves of tiles to place the board in a goal configuration.
- Number of states of the space = 9!

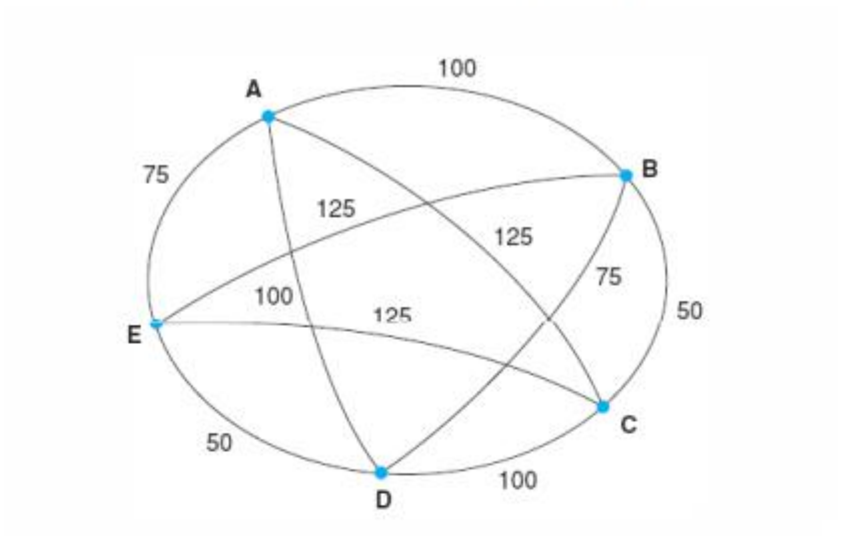
State space of the 8-puzzle generated by "move blank" operations



Example of State Space – The Traveling Salesperson

- A salesperson needs to visit five cities and then return home.
- The goal of the problem is to find the shortest path for the salesperson to travel, visiting each city, and return to the starting city

An instance of the travelling salesperson problem

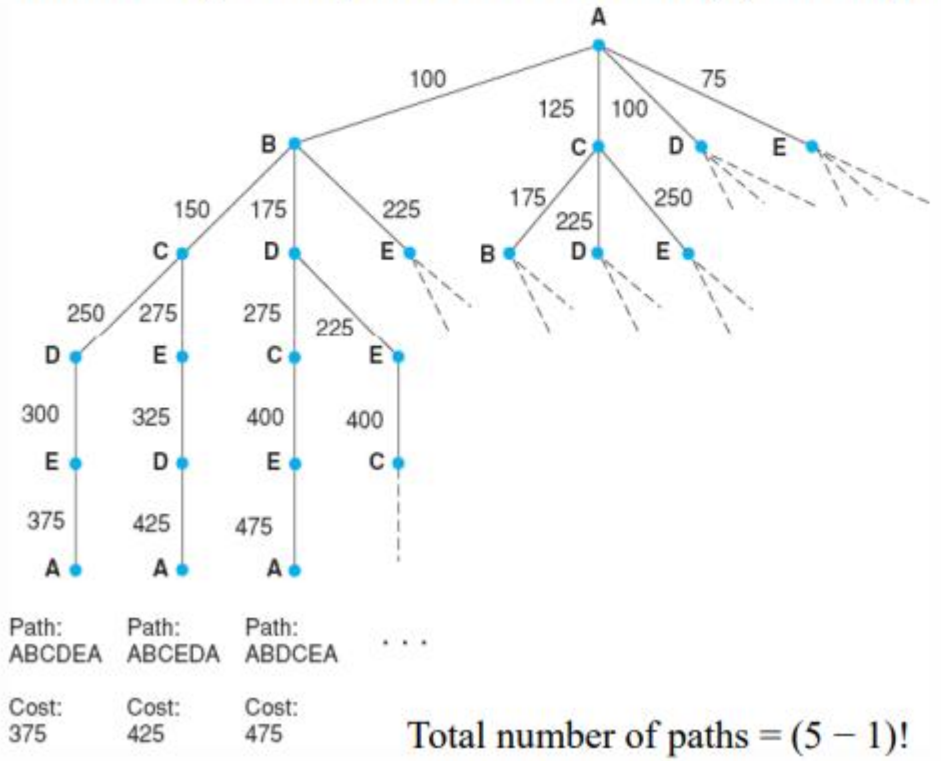


- The nodes of the graph represent the cities.
- The labels on the arcs represent the distances.
- Assume the salesperson lives in city A.

Example of State Space – The Traveling Salesperson

- A salesperson needs to visit five cities and then return home.
- The goal of the problem is to find the shortest path for the salesperson to travel, visiting each city, and return to the starting city

Search for the travelling salesperson problem. Each arc is marked with the total weight of all paths from the start node (A) to its endpoint.



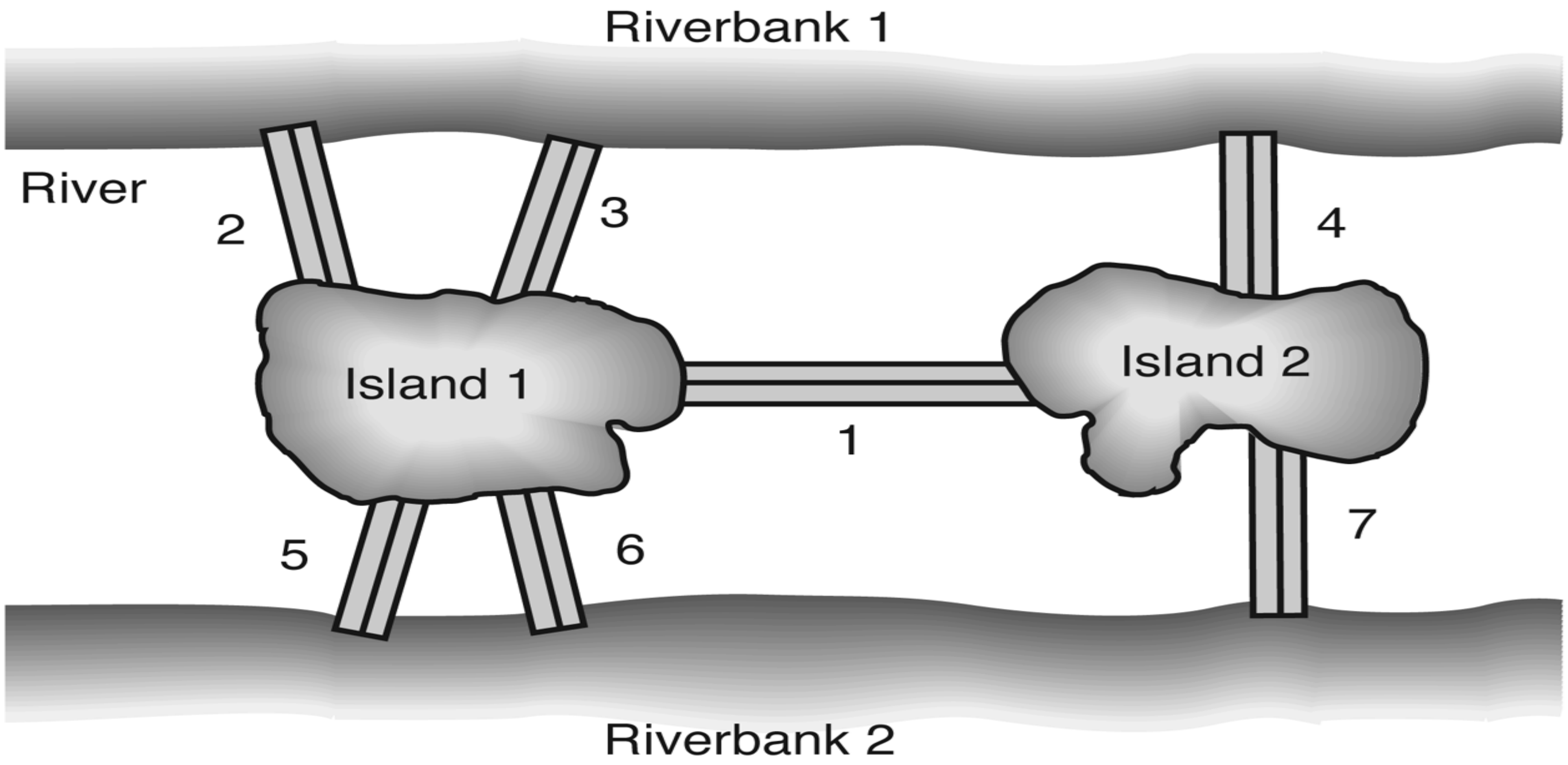
Total number of paths = (5 − 1)!

Graph Theory: Intro

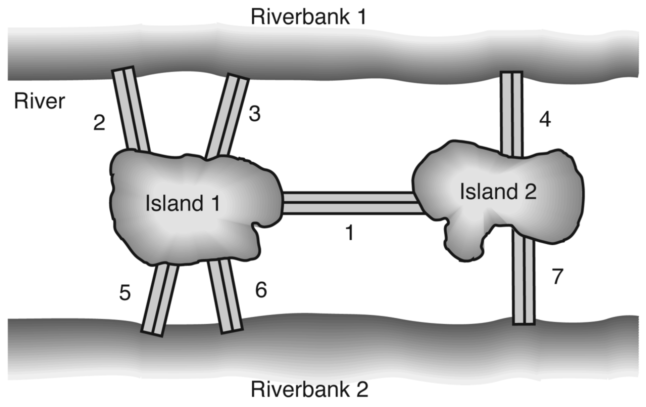
- Tool for reasoning about the structure of objects and relations
- Structure of the problem can be VISUALISED more directly.
- Invented by Swiss Mathematician Leonhard to solve "Bridges of Konigsberg Problem" (Newman 1965)
- Problem: Is there a walk around the city that crosses each bridge exactly once?

Graph Theory

The city of Königsberg- 2 islands, 2 riverbanks and 7 bridges



Represent using Graph Theory

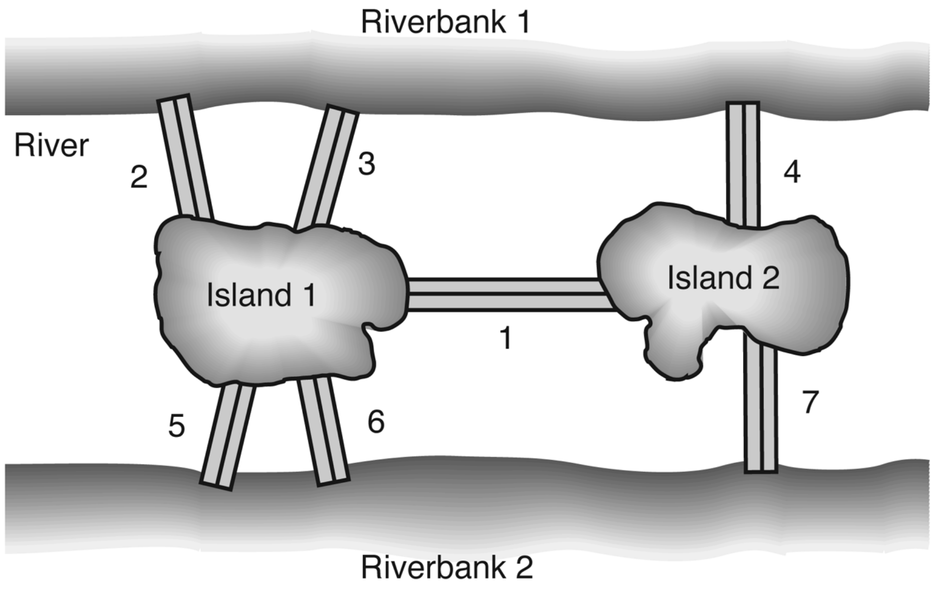


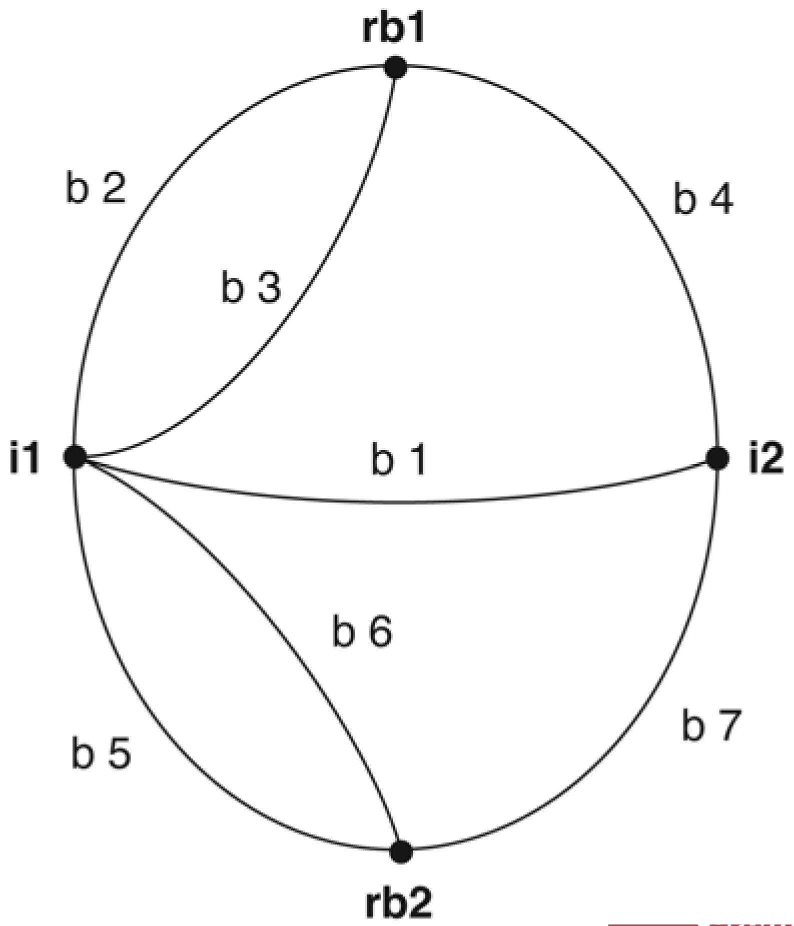
Activity 1.0 (10 mins):

- Assumes that 2 islands and 2 riverbanks become the nodes of the graph. Meanwhile 7 bridges become the arcs.
- Draw a graph that consist all those nodes and arcs.
- Share in the class.

2019

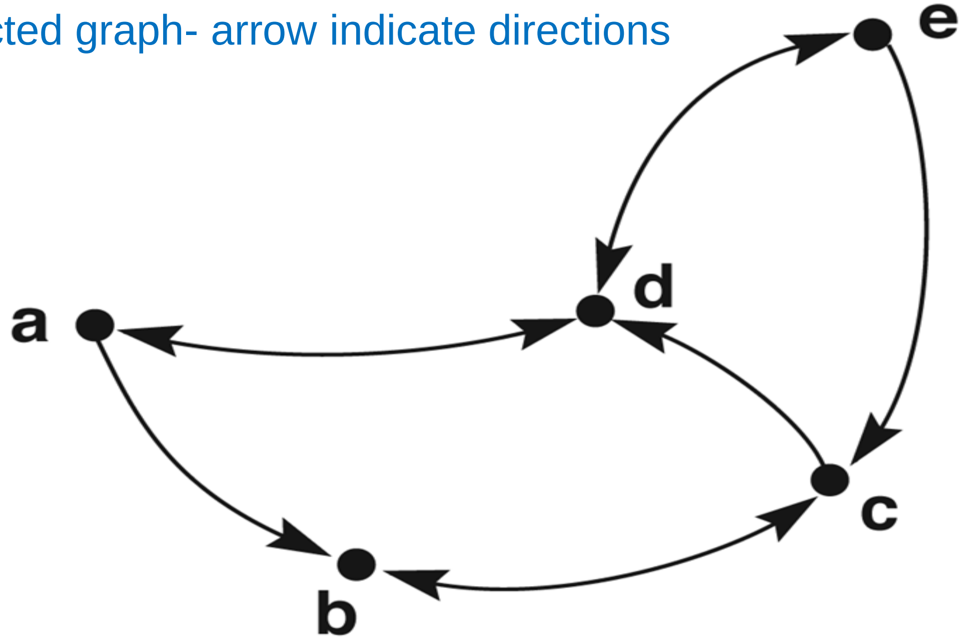
Possible Answer





Graph Theory : Structure

a) A labeled directed graph- arrow indicate directions

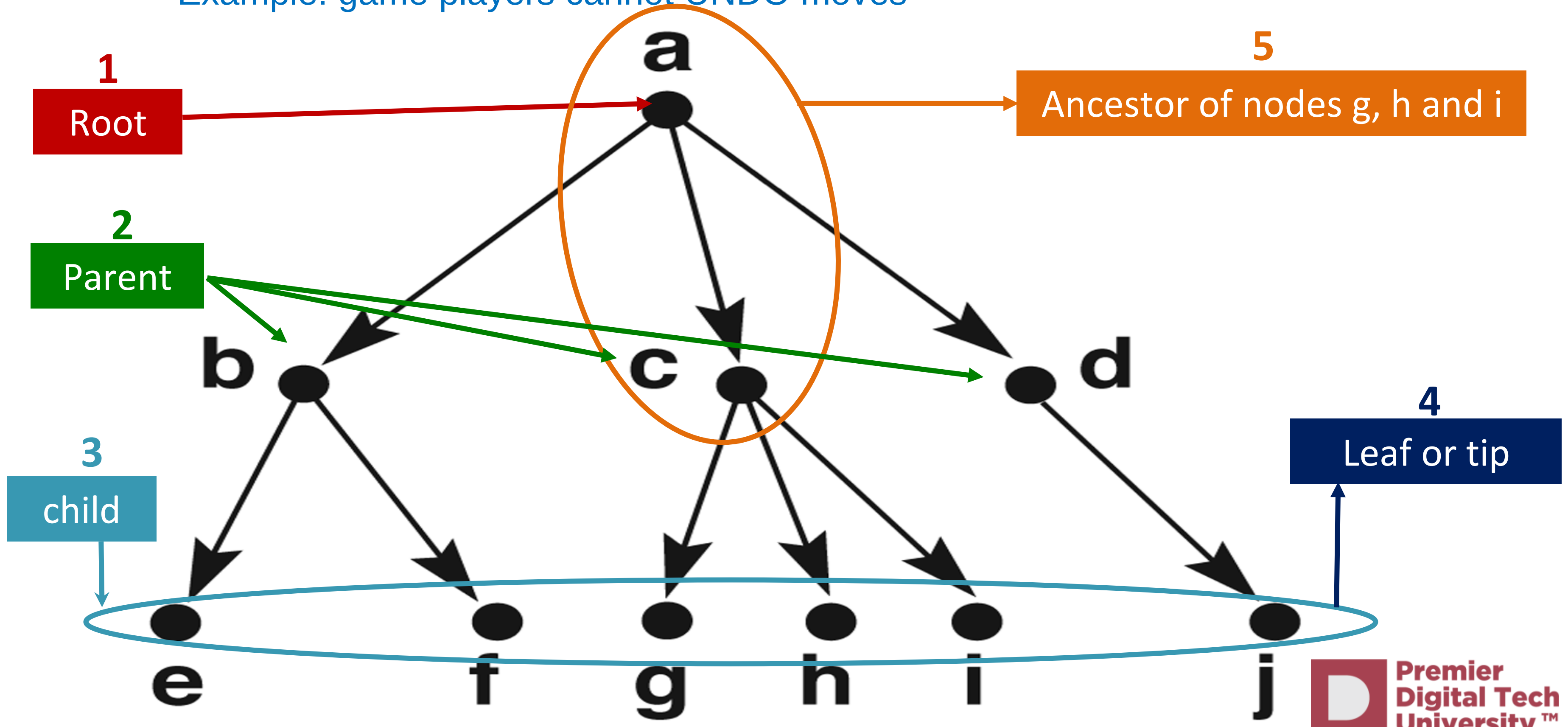


Nodes	{a, b, c, d, e}
Arcs	{(a,b),(a,d), (b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}

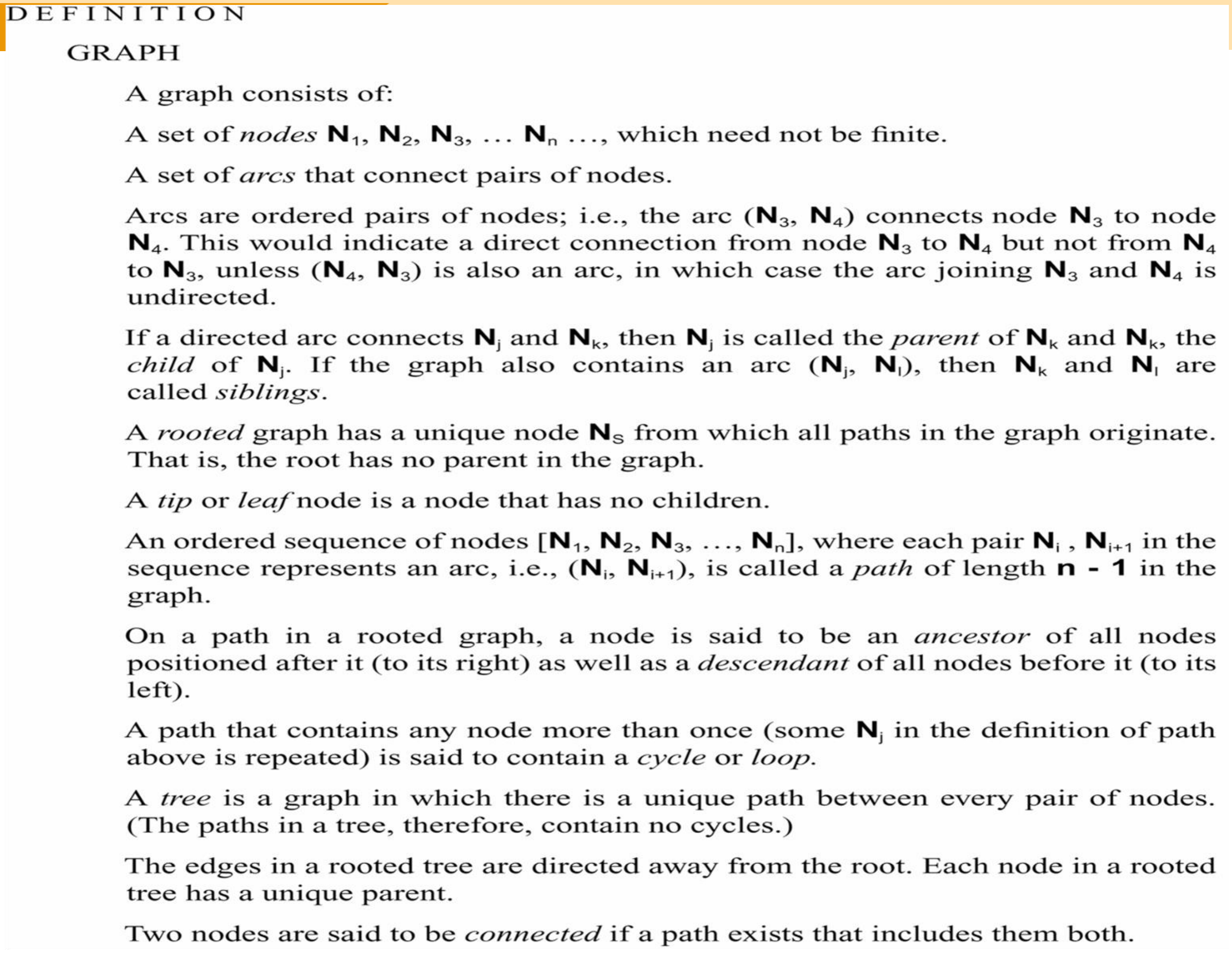
Graph Theory: Structure

b) A rooted tree, exemplifying family relationships(parents, child, sibling)

- Path from root to all nodes
- Directed graph with arcs having one direction ~no cycle
- Example: game players cannot UNDO moves



Graph Theory: Definition



DEFINITION

GRAPH

A graph consists of:

A set of *nodes* $\mathbf{N}_1$, $\mathbf{N}_2$, $\mathbf{N}_3$, … $\mathbf{N}_n$ …, which need not be finite.

A set of *arcs* that connect pairs of nodes.

Arcs are ordered pairs of nodes; i.e., the arc ($\mathbf{N}_3$, $\mathbf{N}_4$) connects node $\mathbf{N}_3$ to node $\mathbf{N}_4$. This would indicate a direct connection from node $\mathbf{N}_3$ to $\mathbf{N}_4$ but not from $\mathbf{N}_4$ to $\mathbf{N}_3$, unless ($\mathbf{N}_4$, $\mathbf{N}_3$) is also an arc, in which case the arc joining $\mathbf{N}_3$ and $\mathbf{N}_4$ is undirected.

If a directed arc connects $\mathbf{N}_j$ and $\mathbf{N}_k$, then $\mathbf{N}_j$ is called the *parent* of $\mathbf{N}_k$ and $\mathbf{N}_k$, the *child* of $\mathbf{N}_j$. If the graph also contains an arc ($\mathbf{N}_j$, $\mathbf{N}_l$), then $\mathbf{N}_k$ and $\mathbf{N}_l$ are called *siblings*.

A *rooted* graph has a unique node $\mathbf{N}_S$ from which all paths in the graph originate. That is, the root has no parent in the graph.

A *tip* or *leaf* node is a node that has no children.

An ordered sequence of nodes [$\mathbf{N}_1$, $\mathbf{N}_2$, $\mathbf{N}_3$, …, $\mathbf{N}_n$], where each pair $\mathbf{N}_i$, $\mathbf{N}_{i+1}$ in the sequence represents an arc, i.e., ($\mathbf{N}_i$, $\mathbf{N}_{i+1}$), is called a *path* of length $\mathbf{n - 1}$ in the graph.

On a path in a rooted graph, a node is said to be an *ancestor* of all nodes positioned after it (to its right) as well as a *descendant* of all nodes before it (to its left).

A path that contains any node more than once (some $\mathbf{N}_j$ in the definition of path above is repeated) is said to contain a *cycle* or *loop*.

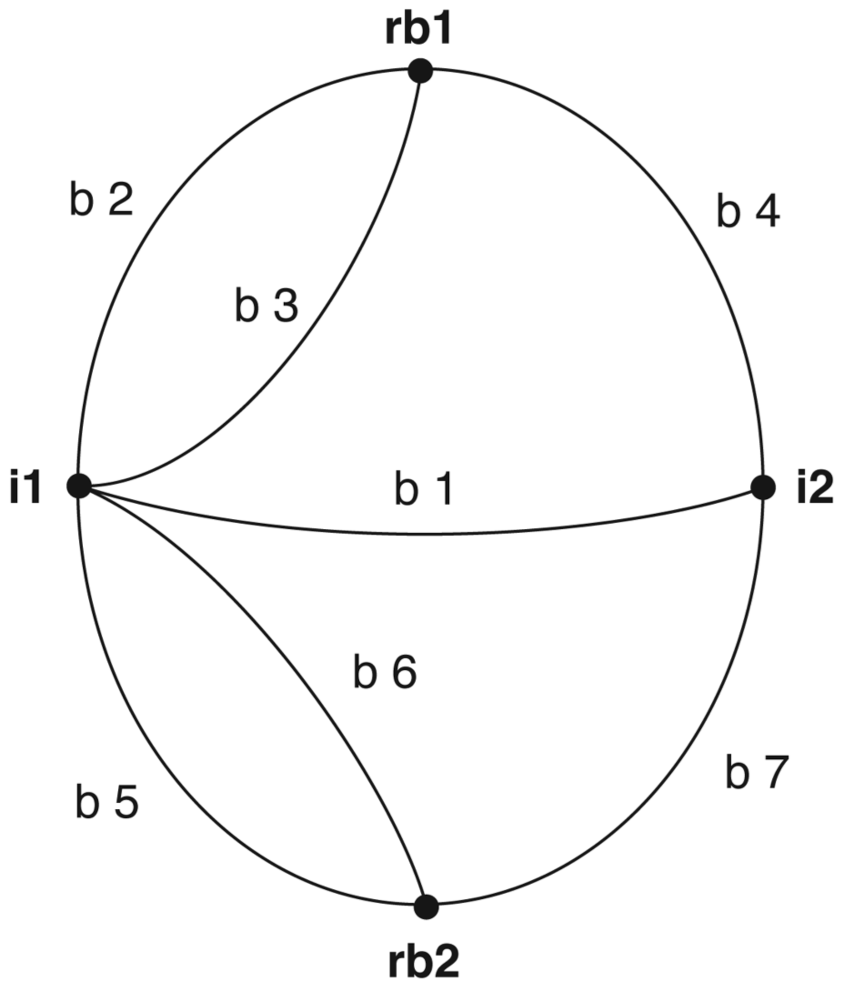
A *tree* is a graph in which there is a unique path between every pair of nodes. (The paths in a tree, therefore, contain no cycles.)

The edges in a rooted tree are directed away from the root. Each node in a rooted tree has a unique parent.

Two nodes are said to be *connected* if a path exists that includes them both.

Graph Theory: Using Predicate Logic

Euler Graph Theory: Graph of the Königsberg bridge system.

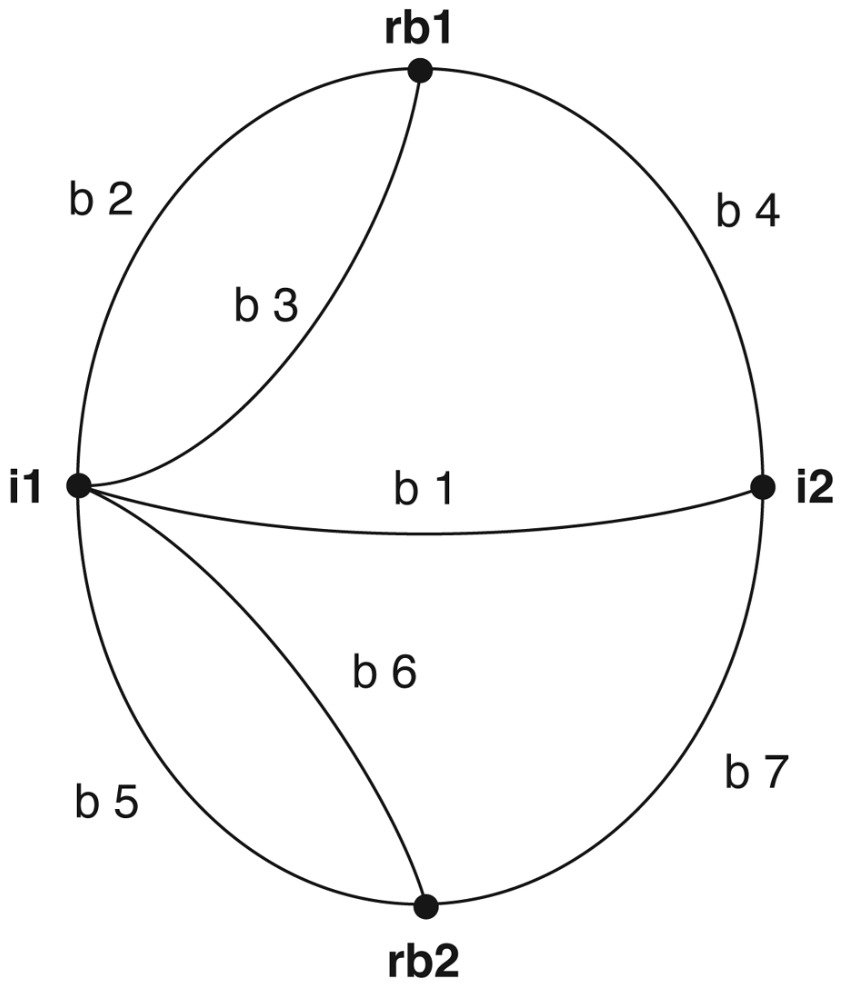


Activity 2.0 (5 mins):

- Based on the given graph of Konigsberg bridge system, two nodes can be connected by an edge. For example, using predicate logic, it can be represented using Connect(i1, i2, b1).
- List all possible connections in a form of predicate logic.
- Share in the class.

Graph Theory: Using Predicate Logic - Answer

Euler Graph Theory: Graph of the Königsberg bridge system.

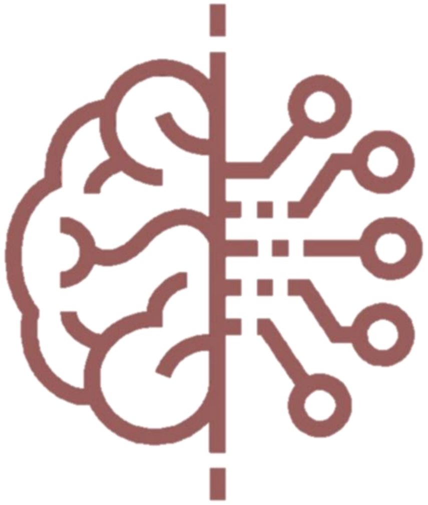


Represent using predicate calculus:
Connect(i1, i2,b1) Connect(i2,i1,b1)
Connect(rb1, i1,b2)Connect(i1,rb1,b2)
Connect(rb1, i1,b3) Connect(i1,rb1,b3)
Connect(rb1, i2, b4) Connect(i2,rb1, b4)
Connect(rb2,i1,b5) Connect(i1,rb2, b5)
Connect(rb2,i1,b6) Connect(i1,rb2,b6)
Connect(rb2,i2,b7) Connect(i2,rb2, b7)

EULER noted WALK WAS IMPOSSIBLE UNLESS A GRAPH HAS EXACTLY ZERO OR TWO NODES OF ODD DEGREES

Graph theory problem

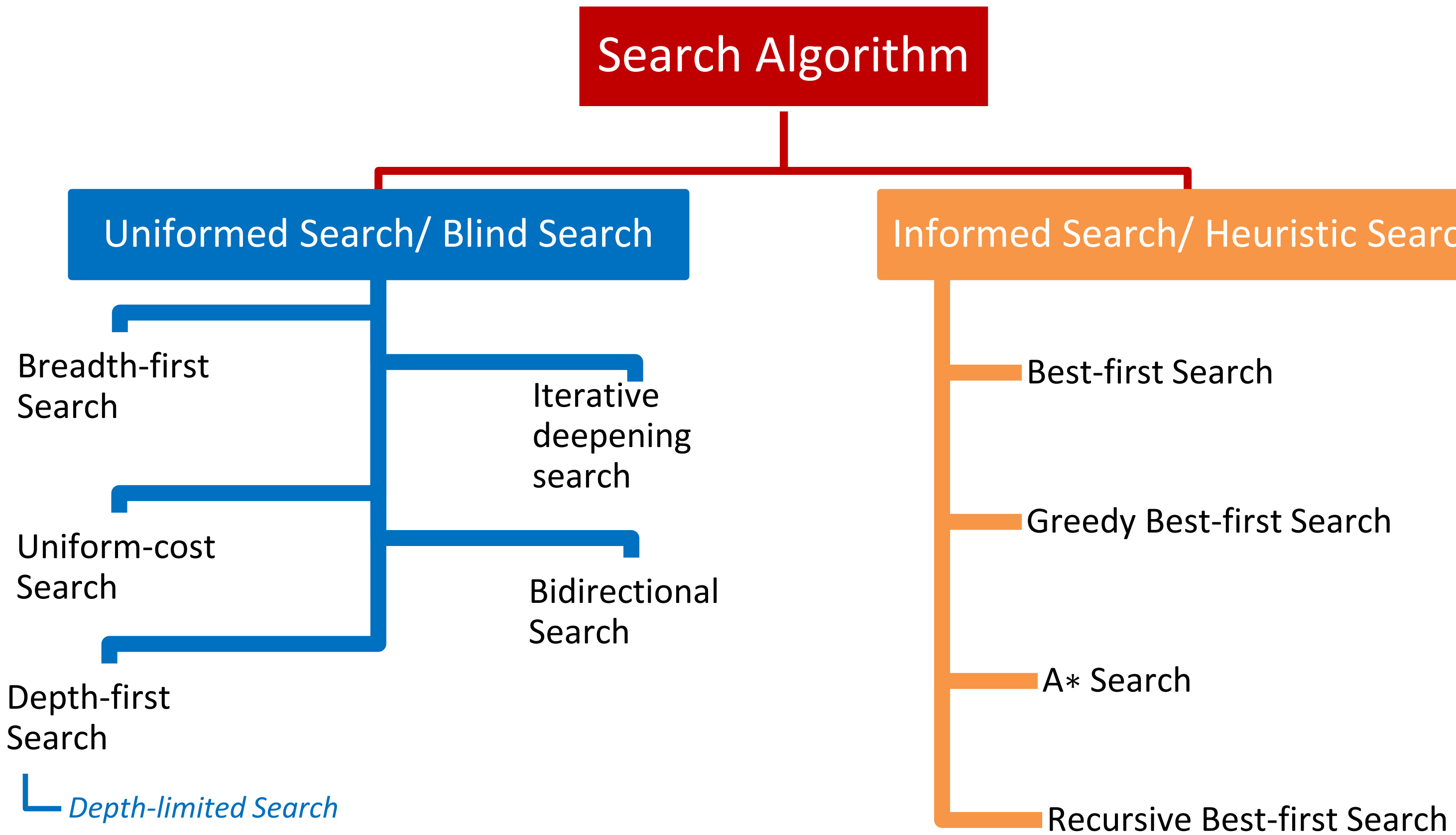
- Wolf, sheep and cabbage problem https://www.youtube.com/watch?v=pBT-8gqhHzo
- A great video explaining graph theory



Search Algorithm

- Search Algorithms
- Selection of a Search Strategy
- Implementation of Graph Search

Search Algorithm : Taxonomy



Search Algorithm : Criteria

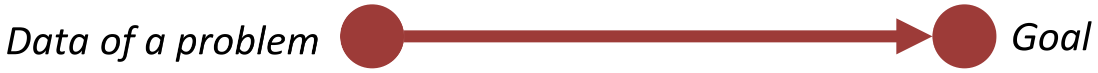
Uninformed Search Algorithms	Informed Search Algorithms
• Uninformed search strategies use only the information available in the problem definition.	• Informed search strategies use other domain specific information
• No additional information on the goal node other than the one provided in the problem definition	• The algorithms have information on the goal state
• The plans to reach the goal state from the start state differ only by the order and/or length of actions.	
• Uninformed search is also called Blind search.	

Selection of a Search Strategy : Intro

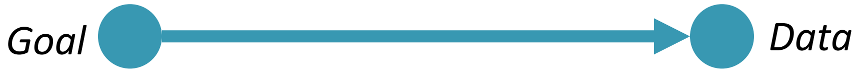
Data Driven & Goal Driven Approach

State space may be searched in two directions:

- From given data of a problem instance towards a goal



- Goal From a goal back to the data



Selection of a Search Strategy

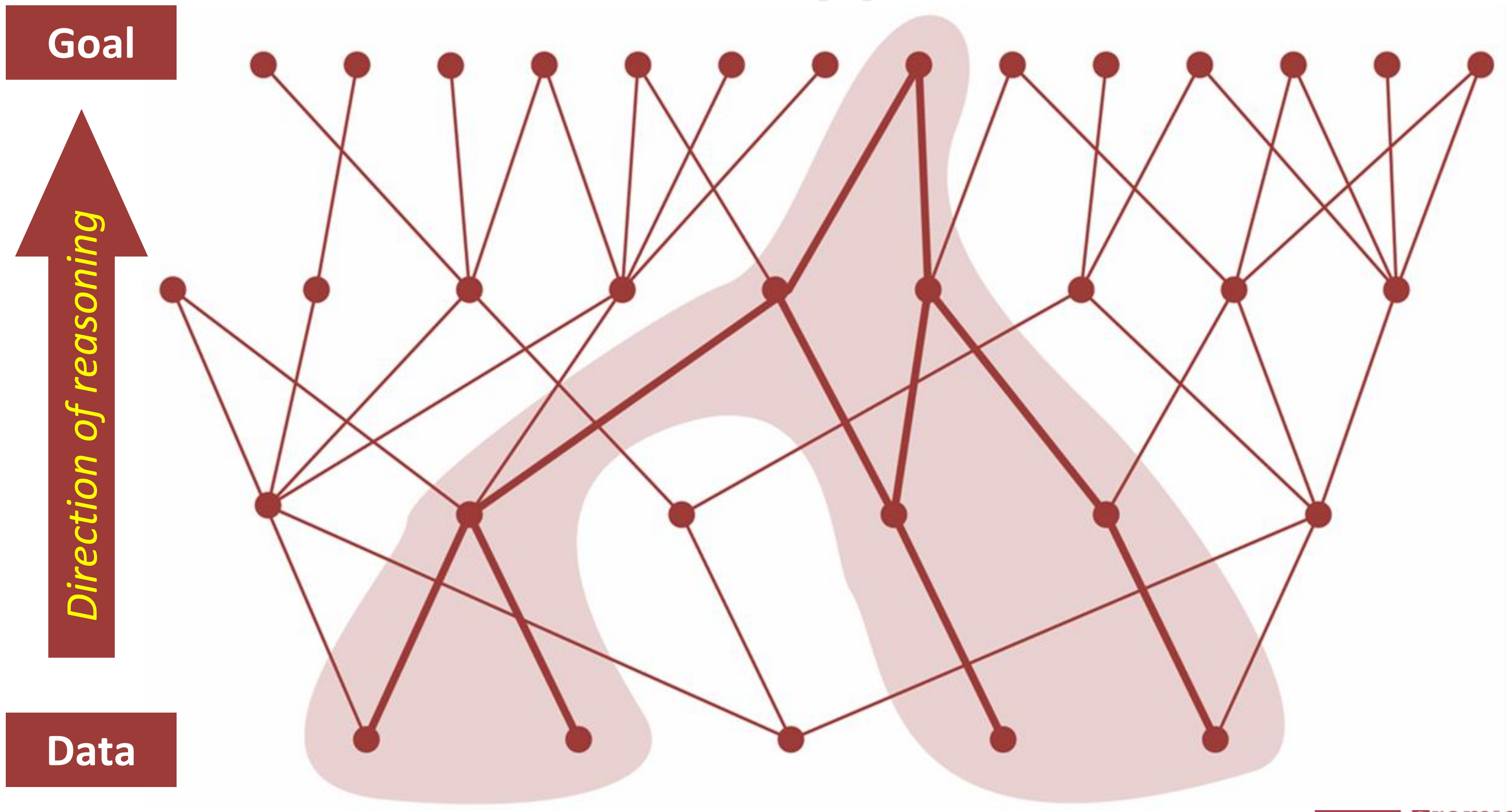
Data-Driven Approach

- Also called as data-directed search, forward chaining
- The problem solver begins with given facts of the problem and a set of legal moves or rules for changing states.

- Search proceeds as follows:-
  - Applying rules to facts to produce new facts.
  - New facts are used by rules to produce more new facts.
  - Process continues until it generates a path that satisfies the goal condition.
  - Data driven search uses the knowledge and constraints found in the given data of a problem to guide search along lines known to be true

Selection of a Search Strategy

Data-Driven Approach



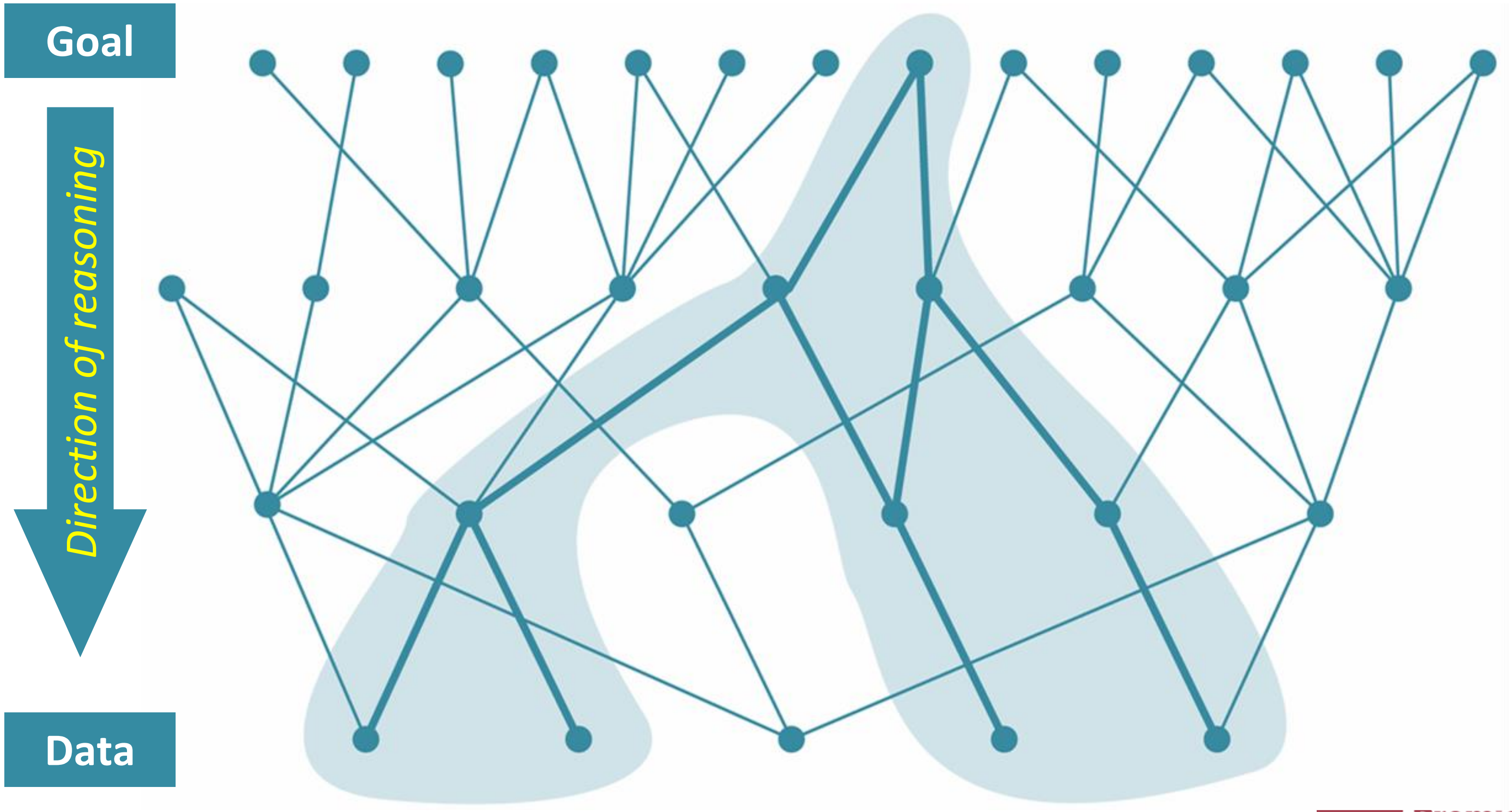
Selection of a Search Strategy

Goal-Driven Approach

- Also called as goal-directed search, backward chaining
- Take the goal that we want to solve and see what rules or legal moves could be used to generate this goal. Also determine what conditions must be true to use them.
- These conditions become the new goals or subgoals for the search. Reaching subgoals, determining new subgoals and so on..
- Search continues, working backward through successive subgoals until it works back to the facts of the problem.
- Goal driven search thus uses knowledge of the desired goal to guide the search through relevant rules and eliminate branches of the space.

Selection of a Search Strategy

Goal-Driven Approach



Selection of a Search Strategy

- Both goal-driven and data-driven approaches search the same state space graph
- Order and number of states are different
- Preferred strategy is determined by:
  - Complexity of the rules
  - Shape of state space
  - Availability of problem data

Selection of a Search Strategy : Summary

Strategies

- **Data-Driven Approach**
  - From a given data of a problem toward a goal
  - Also called forward chaining
  - Given facts, rules
  - Apply rules to facts to produce new facts
  - Rules use new facts to produce more new facts
  - Search continues until a path that satisfies goal is generated
- **Goal-Driven Approach**
  - From a goal back to data
  - Also called backward chaining
  - Take the goal, see what rules apply and which conditions are true to use
  - The condition becomes subgoal
  - Search continues backward through rules and subgoals to the given facts

Selection of a Search Strategy

Data-driven search will by preferred for problems in which:

- All are most of the data are given in the initial problem statement
- There are a large number of potential goals, but there are only a few ways to use the facts and given information of a particular problem instance
- It is difficult to form a goal or hypothesis.

Goal-driven search will be preferred for problems in which:

- A goal or hypothesis is given in the initial problem statement or can easily be formulated
- There are a large number of rules that match the facts of the problem and thus produce an increasing number of conclusions or goals. Early selection of a goal can eliminate most of these branches, making goal driven search more effective
- Problem data are not given but must be acquired by the problem solver. In this case goal driven search can help guide data acquisition.

Implementation of Graph Search : Intro

- In solving a problem using either Data-driven or Goal-driven approaches, a problem solver must find a path from a start to a goal through the state space graph.
- The sequence of the arcs in this path corresponds to the ordered steps of the solution.
- A problem solver must consider different paths until it finds a goal.

Implementation of Graph Search : Backtracking

- The algorithm begins at start state, pursue a path until reaches either a goal or a dead end
- If it finds a goal, it returns the path
- It finds a dead end:
  - It backtracks to the most recent node on the path (node S)
  - Pursue a new path along one of unexamined child of node S
  - If the backtrack does not find a goal in this subgraph. Repeat the procedure for all siblings of node S
  - If none of the siblings leads to a goal. Then backtrack to the parent node of node S
  - The procedure may be applied to all siblings of node S, and so on

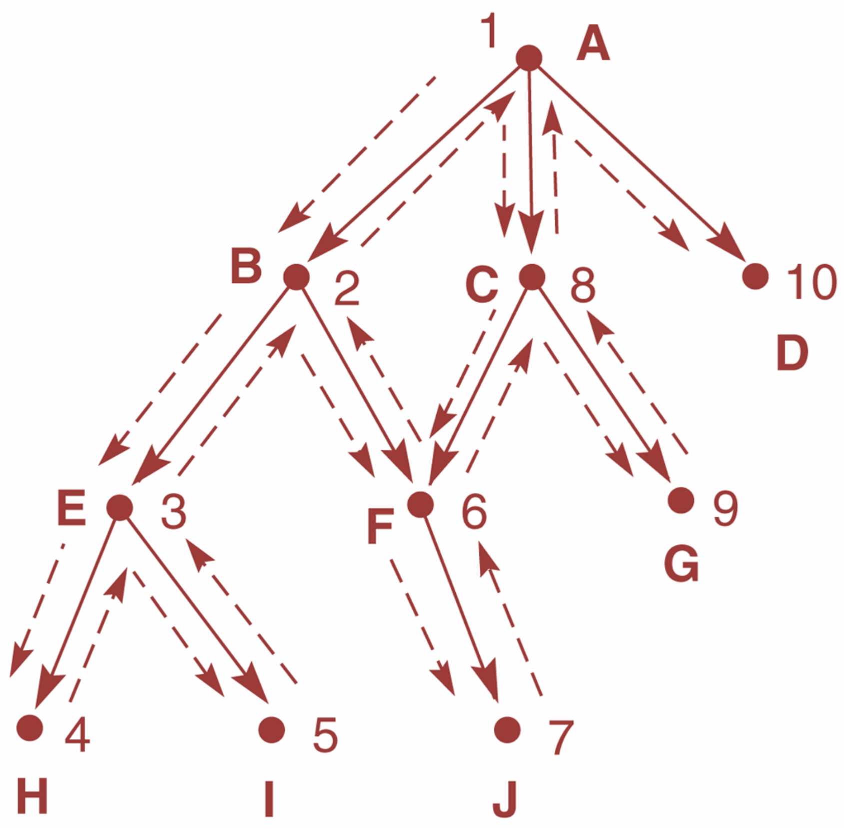
Implementation of Graph Search

- An algorithm which performs a backtrack search uses three lists to keep track of nodes in the state space. It begins search at start state and continues until it reaches a GOAL or 'DEAD END'
- If found GOAL, it quits and return solution path. If found dead-end, it backtracks to the most recent unvisited nodes.
- Use lists to keep track nodes in state space:
    - STATE LIST(SL): current path tried
    - NEW STATE LIST(NSTL): nodes to be visited
    - DEAD END (DE): states failed nodes and eliminated from evaluation
- To avoid re-entry of a state already occurred (to avoid loops), each newly generated state is tested for membership in above lists. If new state belongs to any of these lists, it has already been visited so may be ignored.
- Advantage: keep track states in search; breadth-first, depth-first and best-first exploits this idea in backtracking

Implementation of Graph Search

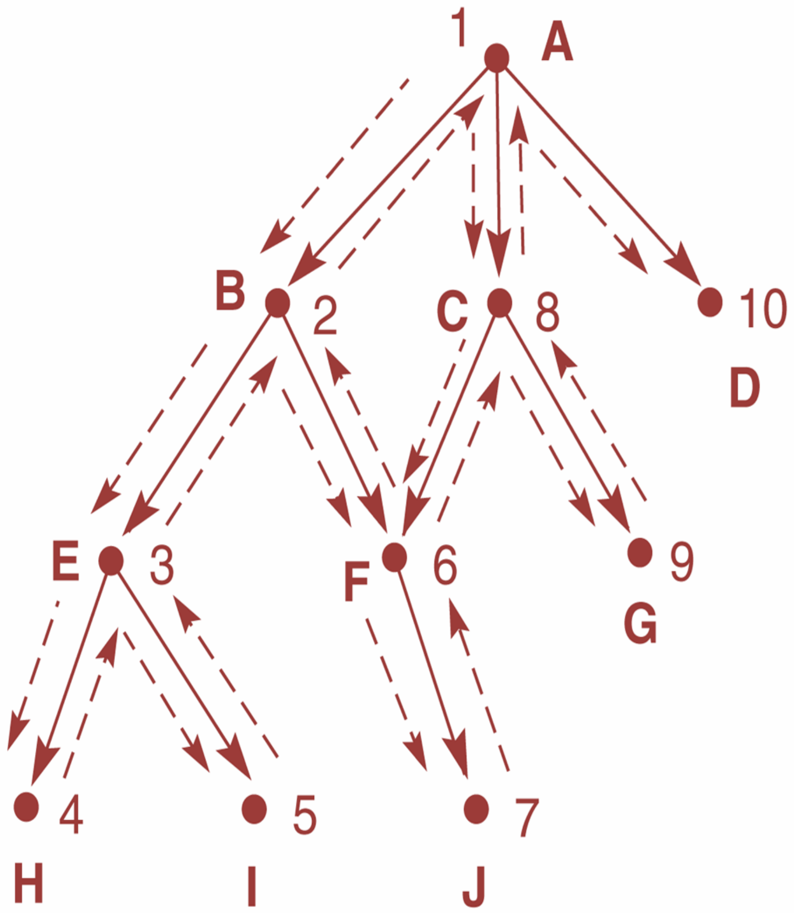
```
function backtrack;

begin
  SL := [Start];  NSL := [Start];  DE := [ ];  CS := Start;                 % initialize:
  while NSL ≠ [ ] do                                    % while there are states to be tried
    begin
      if CS = goal (or meets goal description)
        then return SL;                       % on success, return list of states in path.
      if CS has no children (excluding nodes already on DE, SL, and NSL)
        then begin
          while SL is not empty and CS = the first element of SL do
            begin
              add CS to DE;                              % record state as dead end
              remove first element from SL;                               %backtrack
              remove first element from NSL;
              CS := first element of NSL;
            end
          add CS to SL;
        end
      else begin
          place children of CS (except nodes already on DE, SL, or NSL) on NSL;
          CS := first element of NSL;
          add CS to SL
        end
    end;
    return FAIL;
end.
```



Implementation of Graph Search

Backtracking search of a hypothetical state space.



Initialize: SL = [A]; NSL = [A]; DE = [ ]; CS = A;

Current node

Node to visit

Goal

AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

Implementation of Graph Search

Depth-First and Breadth-First Search

- In addition to specifying a search direction (data-driven or goal- driven), a search algorithm must determine the order in which states are examined in the tree or the graph.
- We will consider two possibilities for the order in which the nodes of the graph are considered:-
  - Depth-first search
  - Breadth-first search
- Depth-first search, when a state is examined, all of its children and their descendants are examined before any of its siblings.
- Depth-first search goes deeper into the search space whenever this is possible. Only when no further descendants of a state can be found are its siblings considered.
- Depth-first search examines the states in the graph of Figure, in the order A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R.
- This search can be implemented with backtrack algorithm.

Implementation of Graph Search

Depth-First and Breadth-First Search

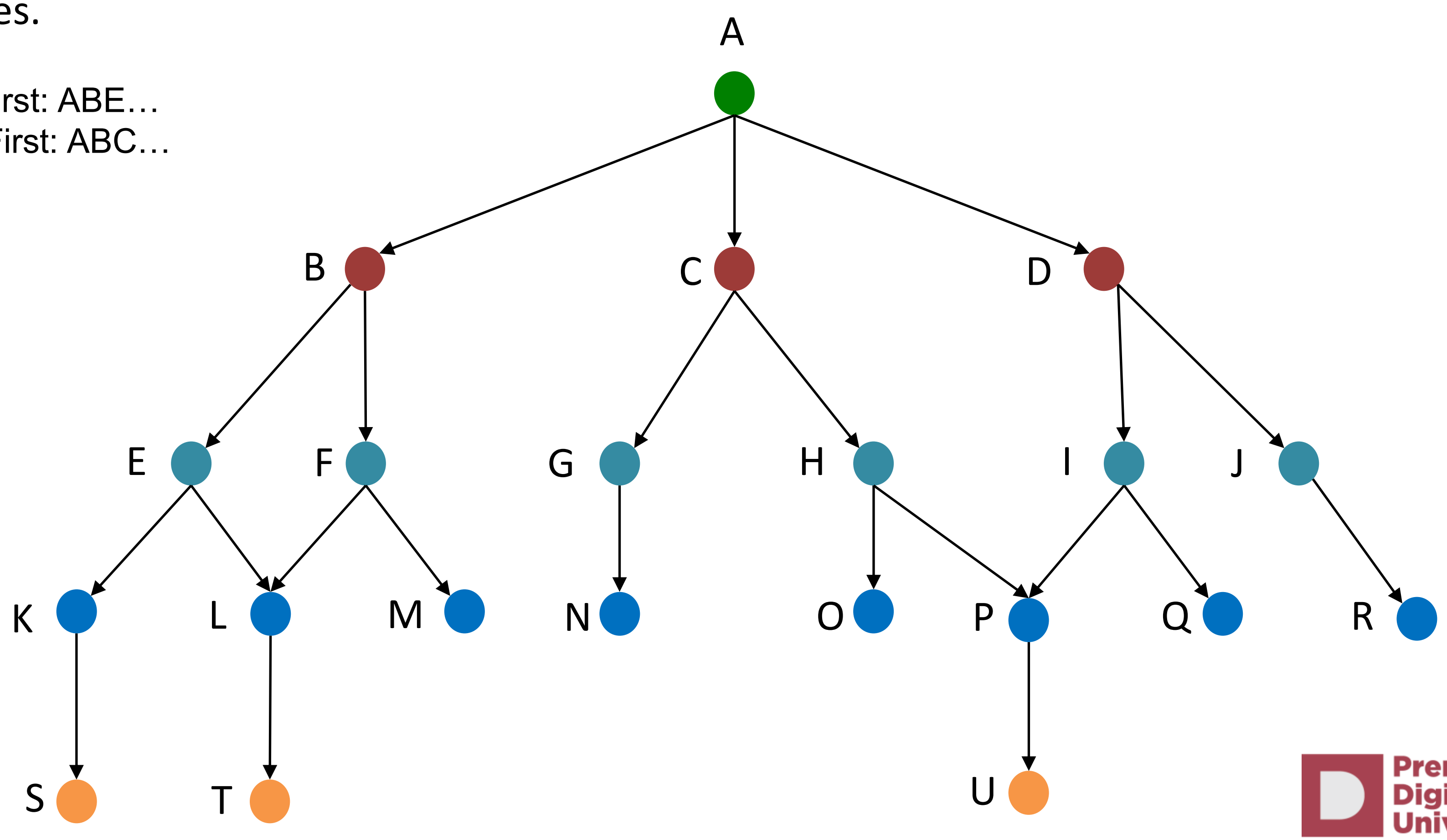
- Breadth-first search, in contrast, explores the space in a level- by-level fashion.
- Only when there are no more states to be explored at a given level does the algorithm move on to the next level.
- A breadth-first search of the graph of Figure considers the states in the order A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.
- We implement breadth-first search using lists, open and closed, to keep track of progress through the state space. Open, like NSL in backtrack, lists the states that have been generated but whose children have not been examined. The order in which states are removed from open determines the order of the search.
- Closed records states that have already been examined.

Activity 3

Question :

Give the possibilities path for Depth-first search and Breadth-first search to complete traverse all nodes.

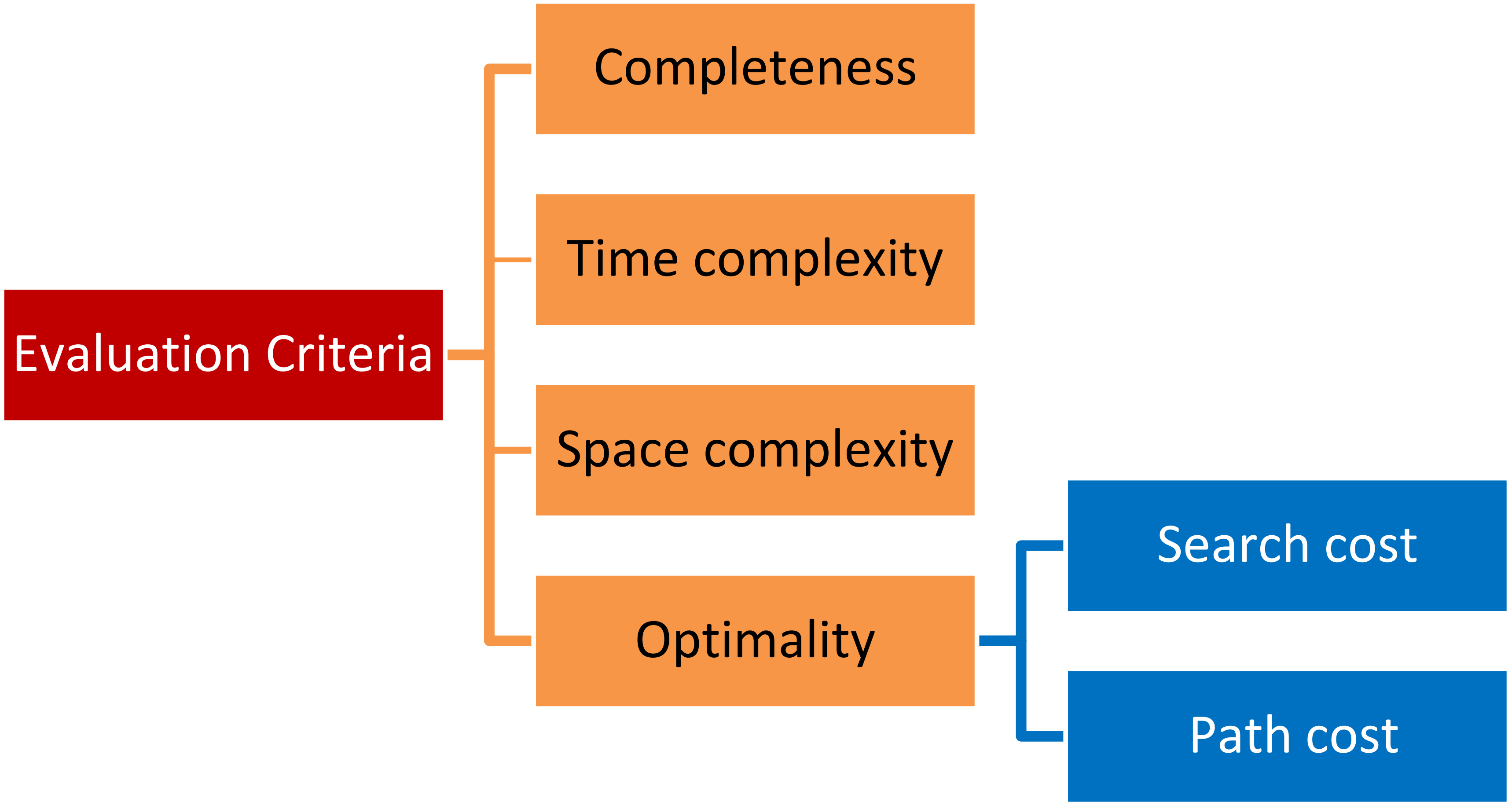
Depth-First: ABE…
Breath-First: ABC…



Evaluation Criteria

- Completeness
- Time Complexity
- Space Complexity
- Optimality

Evaluation Criteria



Evaluation Criteria

- completeness
  - if there is a solution, will it be found
- time complexity
  - how long does it take to find the solution
  - does not include the time to perform actions
- space complexity
  - memory required for the search
- optimality
  - will the best solution be found

Time and space complexity are measured in terms of *b, d, m*

main factors for complexity considerations:

branching factor *b*, depth *d* of the shallowest goal node, maximum path length, *m*

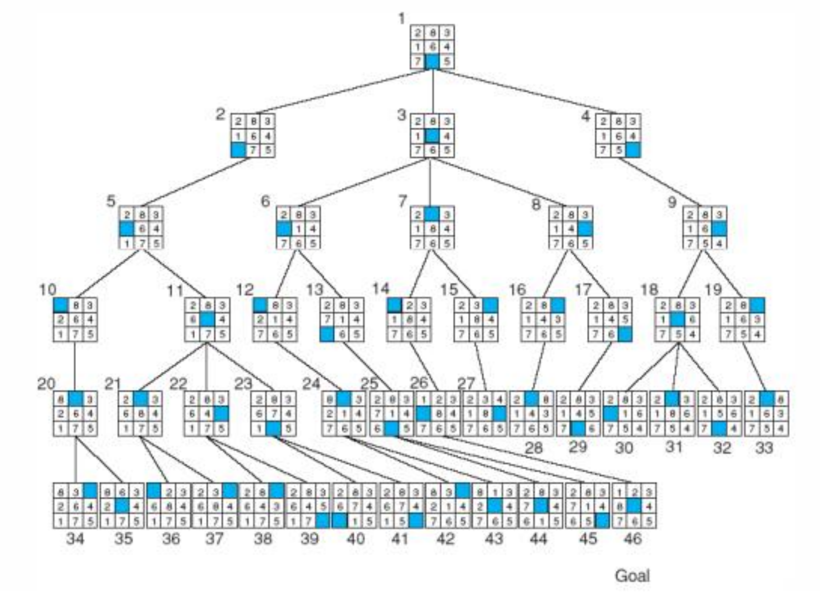
Search and Path Cost

- The search cost indicates how expensive it is to generate a solution
  - Time complexity (e.g. number of nodes generated) is usually the main factor
  - Sometimes space complexity (memory usage) is considered as well
- Path cost indicates how expensive it is to execute the solution found in the search
  - distinct from the search cost, but often related
- Total cost is the sum of search and path costs

DFS vs BFS in solving 8-puzzle game

- Assume that the Path cost indicates the number of steps to reach goal node. Which algorithm wins? Why?

Breadth-first search of the 8-puzzle, showing order in which states were searched.



Depth-first search of the 8-puzzle with a depth bound of 5.