

# Graph theory, Recursion and its implementation in board games

Ibrahim Butt 24043  
Muhammad Tahir Ahmed 24151  
Ammara Khan 24133  
Sara Abid 24112  
Israr Hussain 24045

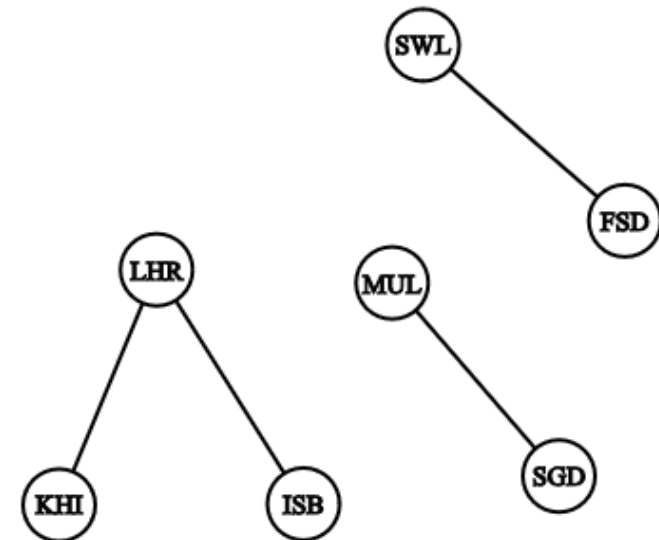
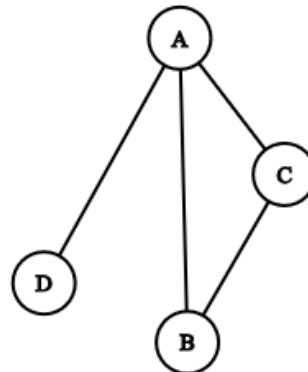
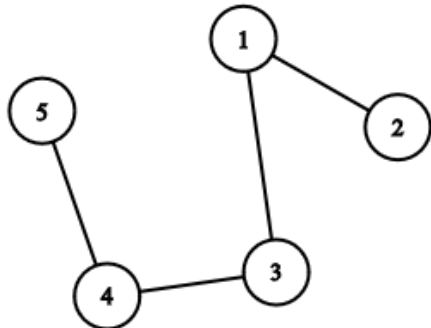
# GRAPH

A list of pairs of “things” called vertex/vertices, and lines between those points, called edges

$A = \{(1, 2), (1, 3), (3, 4), (4, 5)\}$

$B = \{(A, B), (B, C), (A, C), (A, D)\}$

$C = \{(LHR, KRA), (LHR, ISB), (FSD, SWL), (STD, MUL)\}$

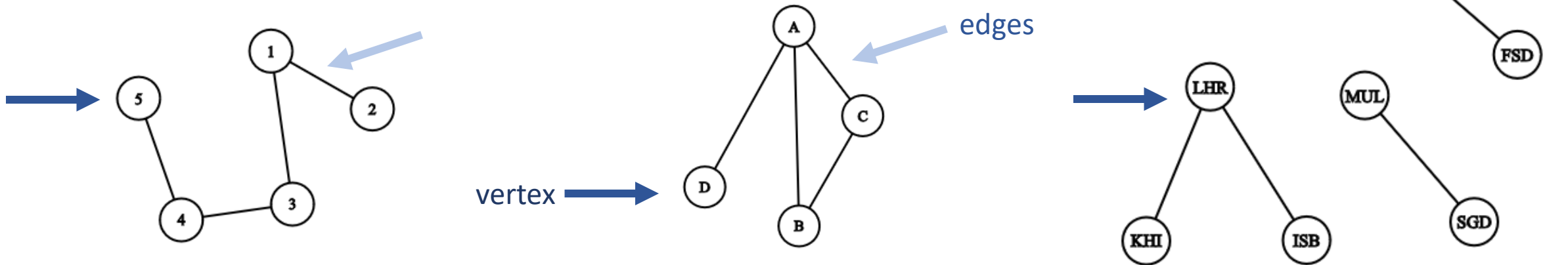


**GRAPH** : a list of pairs of “things” called vertex/vertices, and lines between those points, called edges

$A = \{(1, 2), (1, 3), (3, 4), (4, 5)\}$

$B = \{(A, B), (B, C), (A, C), (A, D)\}$

$C = \{(LHR, KRA), (LHR, ISB), (FSD, SWL), (STD, MUL)\}$

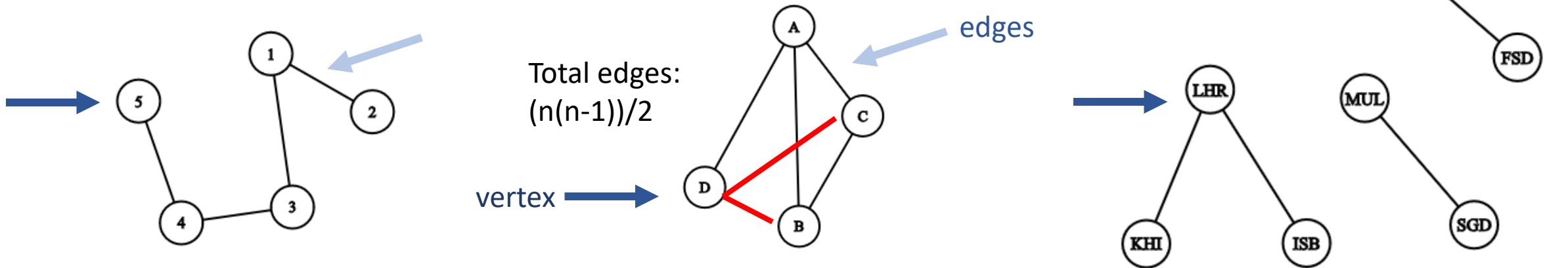


**GRAPH :** a list of pairs of “things” called vertex/vertices, and lines between those points, called edges

$A = \{(1, 2), (1, 3), (3, 4), (4, 5)\}$

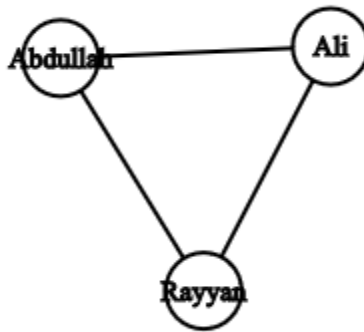
$B = \{(A, B), (B, C), (A, C), (A, D)\}$

$C = \{(LHR, KRA), (LHR, ISB), (FSD, SWL), (STD, MUL)\}$

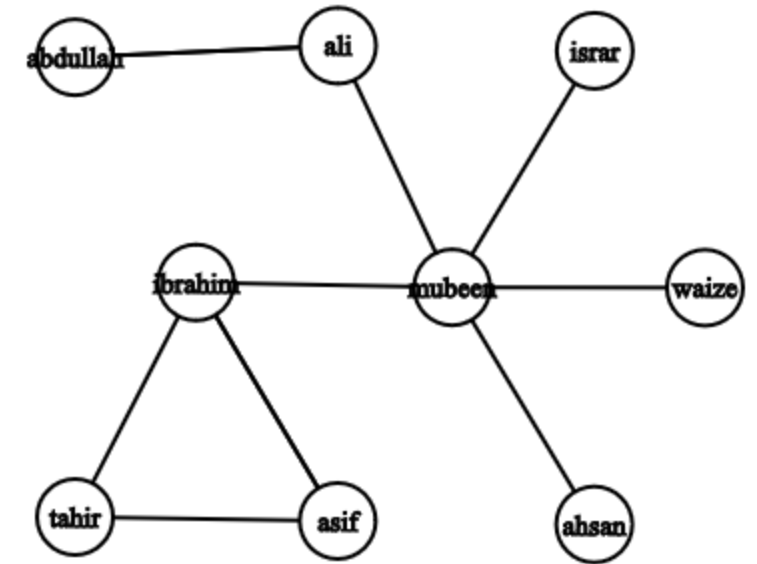


# Difference between Directed and undirected graphs?

Siblings Relation

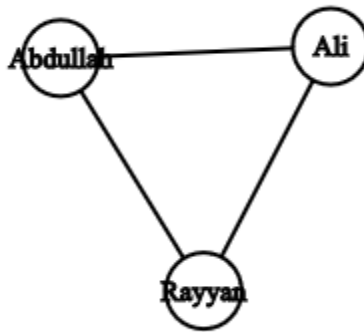


Instagram followers

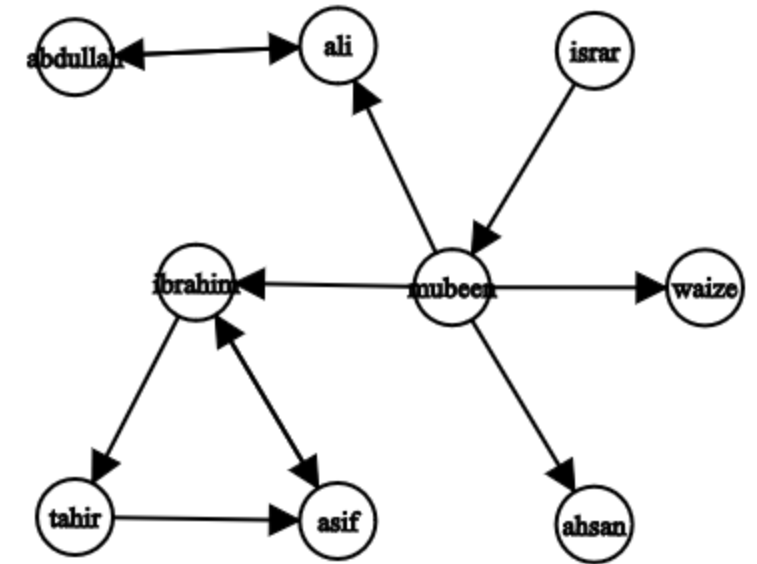


# Difference between Directed and undirected graphs?

Siblings Relation



Instagram followers



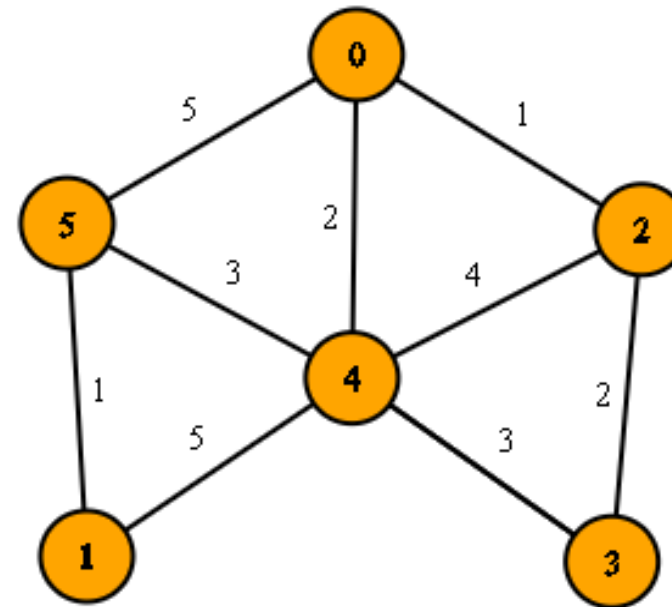
# Weighted Graphs

Many graphs can have edges that contain a certain weight to represent an arbitrary value such as cost, distance, quantity, etc.

This suggests that all edges cannot be treated equally in certain cases.

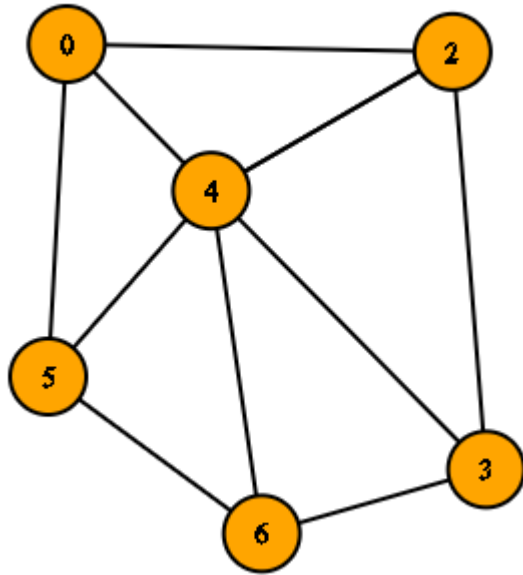
## USE IN GAME:

- to represent the potential impact or strategic value of certain moves.
- Weights could represent how close a sequence is to completion
- In more advanced AI implementations, weighted graphs can help find paths that maximize player's advantage while minimizing the opponent's.

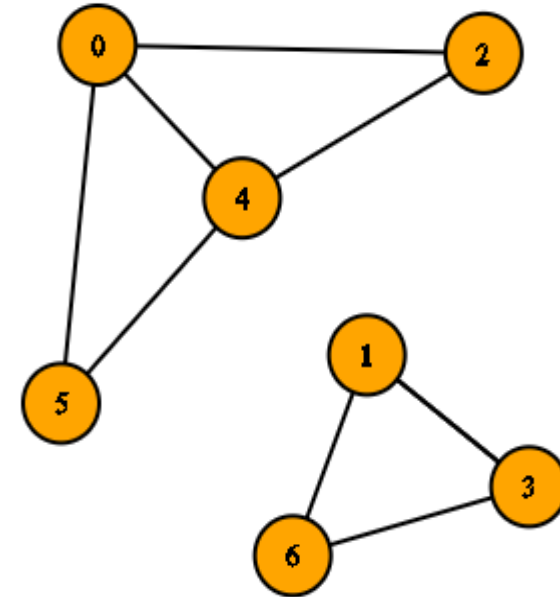


# Connected vs Disconnected Graphs

A graph is called connected when a path exist between all of its vertices.



A graph is called disconnected when a path doesn't exist between all of its vertices.





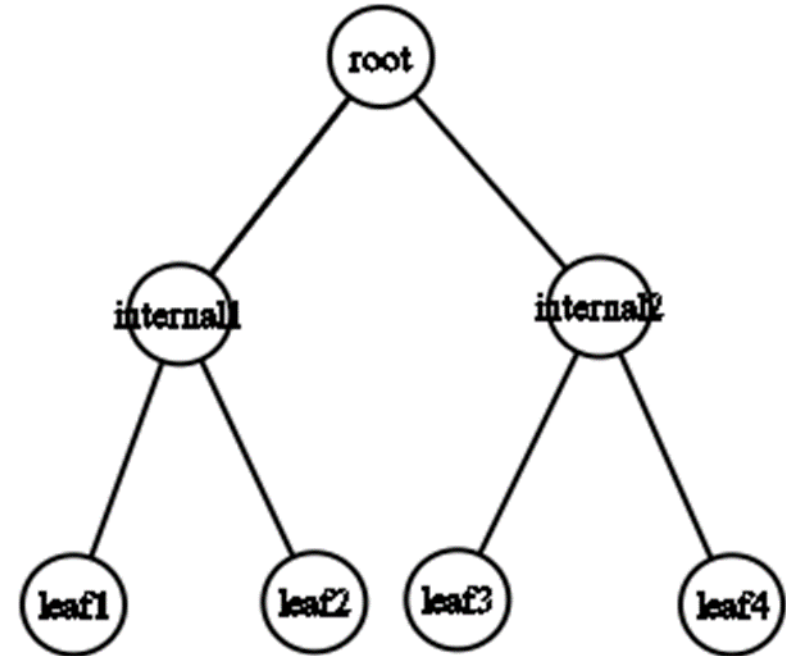
# TREES AND SPANNING TREES

Tree is a simple, connected ,undirected ,acyclic(having a unique path between any two vertices) graph.

A **spanning tree** of a graph is a subgraph that includes all vertices of the graph and is a tree.

- Spanning trees connect all vertices with the minimum possible edges.

In games, spanning trees can help create unique paths with no loops.



# GRAPH REPRESENTATION

1. Edge List
  1. Individual track of all edges
  2. Need to check all edges
2. Adjacency List
  1. Track of all adjacent to a particular edge
  2. Need to check relevant list of adjacent
3. Adjacency Matrix
  1. Table in form of rows and columns that keep track of all possible edges in terms of YES/NO
  2. Only need to check one cell.

# Things we need for our area of research:

1. How to traverse a graph
2. Count neighbors of a vertex
3. Count degree of a vertex(number of edges meeting at that vertex)
4. Deleting and inserting edges
5. Path and cycle of vertices

# RECURSION:

- **BASE CASE:**

a condition that allows an algorithm stops further recursive calls and return a result.

- **Recursive case:**

It is the part of a recursive function that breaks down a problem into smaller subproblems, which moves the problem closer to the base case.

## **RECURSIVE FUNCTION:**

Recursive function is a function that values at any point can be calculated from the values of the function at some previous points

# HOW RECURSION WORKS :

- Step By Step :

Step 2: Step 1 + lowest step.

Step 3: Step 2 + Step 1 + lowest step.

Step 4: Step 3 + step 2 + step 1+ lowest step, and so on.

The smallest argument is donated by  $f(0)$  or  $f(1)$  .

The nth argument is donated by  $f(n)$ .

- Example:  
fibonacci sequence.

## Fibonacci Series Formula

$$F_0 = 0, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

For  $n > 1$

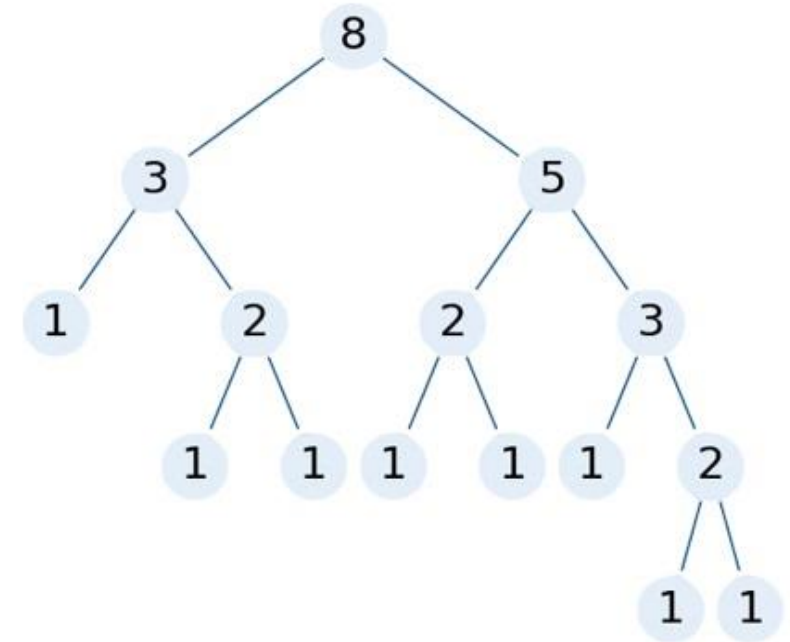
⇒ Fibonacci Series:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

# Recursive function:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

1. Fibonacci(5) calls  $\rightarrow$  Fibonacci(4) + Fibonacci(3)
2. Fibonacci(4) calls  $\rightarrow$  Fibonacci(3) + Fibonacci(2)
3. Fibonacci(3) calls  $\rightarrow$  Fibonacci(2) + Fibonacci(1)
4. Fibonacci(2) calls  $\rightarrow$  Fibonacci(1) + Fibonacci(0)
5. Base case:  
    Fibonacci(1) returns 1  
    Fibonacci(0) returns 0



```
def fibonacci(n):  
    # Base cases  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    # Recursive case  
    return fibonacci(n - 1) + fibonacci(n - 2)
```

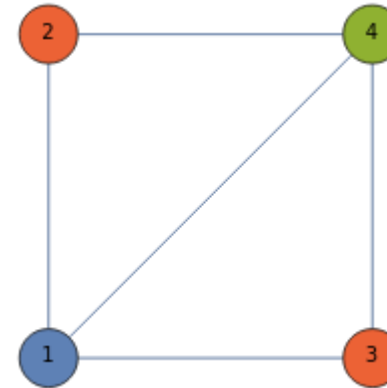
# Graph in Board Games

Most board games are played two-dimensional grid.

- Nodes represent the cells where game pieces are placed.
- Winning conditions involve finding paths (edges) with consecutive stones.

## Graph Coloring in Board Games

- involves assigning different colors to the vertices of a graph such that no two adjacent vertices share the same color.
- Simplifies game design by managing player interactions and organizing regions.



1. Territory Colouring: Assigning different colours to neighbouring territories.
2. Player moves: Useful in multiplayer games to minimize conflicts.
3. Puzzle Solving: No two adjacent players have the same colours.



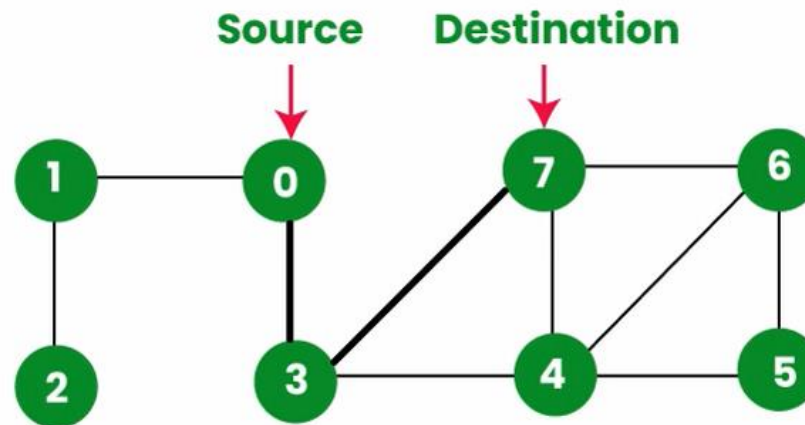
# Path finding

## Definition:

“**Pathfinding** refers to the process of finding a path from one point (vertex) to another in a graph.”

A **path** in a graph is a sequence of edges that connects a series of vertices.

It can be represented as a list of vertices starting from a source node and ending at a destination node, with each consecutive vertex connected by an edge.



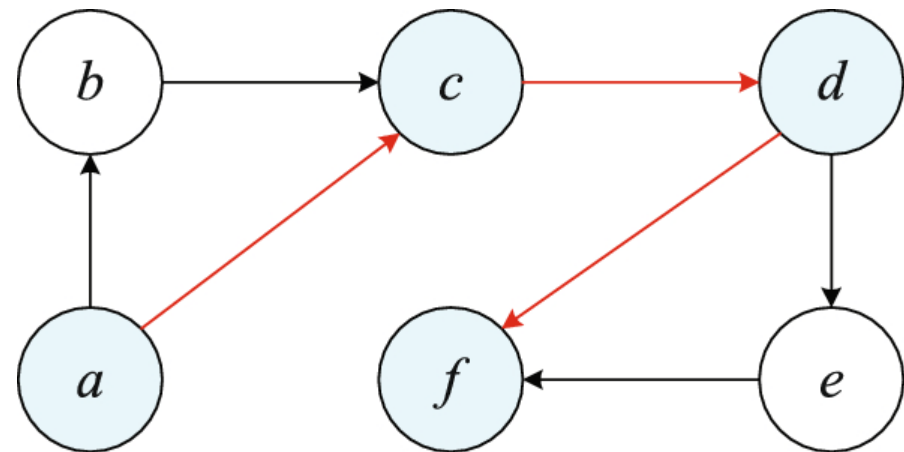
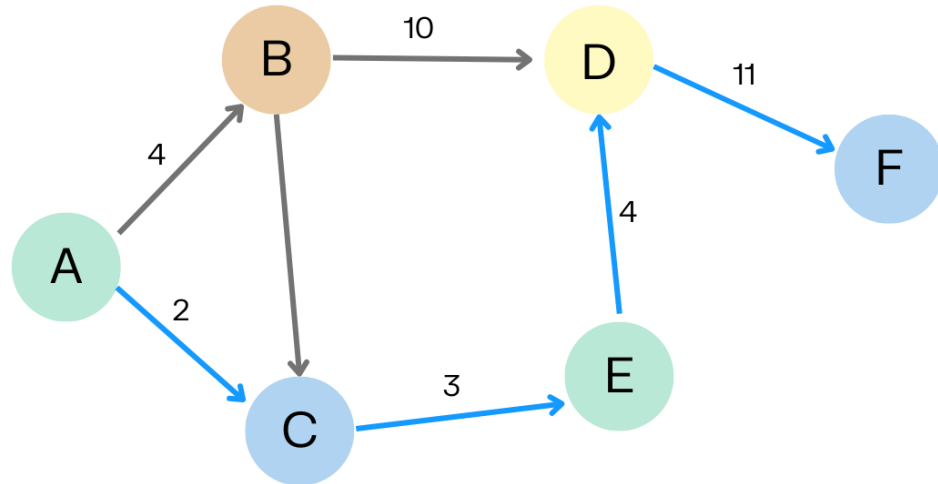
# Shortest Path

## Definition:

“The **shortest path** is a path between two vertices such that the sum of the weights of the edges in the path is minimized.”

In unweighted graphs, the shortest path is simply the path with the least number of edges.

In weighted graphs, it's the path with the minimum total weight.



# Some Algorithms Used for Path Finding

- Dijkstra's Algorithm
  - A\* Algorithm
- Bellman-Ford Algorithm
- Floyd-Warshall Algorithm
- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Johnson's Algorithm

# Some Algorithms Used for Path Finding

- Dijkstra's Algorithm
  - A\* Algorithm
- Bellman-Ford Algorithm
- Floyd-Warshall Algorithm
- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Johnson's Algorithm

We have used Breadth-First Search (BFS) in our code..

# Breadth-First Search Algorithm

BFS is the best algorithm used for path finding in **unweighted graphs**.

This BFS implementation explores close moves first and extends outward, efficiently finding potential moves for the AI to play based on nearby pieces.

# Recursive Algorithms in Board Games

Recursion is often used in board games to explore all the possible game states or moves to determine which one's the best for you.

Minimax Algorithm

Alpha-Beta Pruning

Monte Carlo Tree Search (MCTS)

Negamax

Depth-First Search (DFS)

Breadth-First Search (BFS)

A\* Search Algorithm

Minimax with Heuristics

# Minimax Algorithm

It is commonly used in **two-player games** like chess which helps to make the best decisions by assuming that both players play **optimally**.

## How It Works

The game is represented as a **tree**, where each node represents a game state.

Edges represent moves.

The algorithm alternates between two players:

**Maximizer:** Tries to maximize the score (e.g., their chance of winning).

**Minimizer:** Tries to minimize the score (e.g., their opponent's chance of winning).

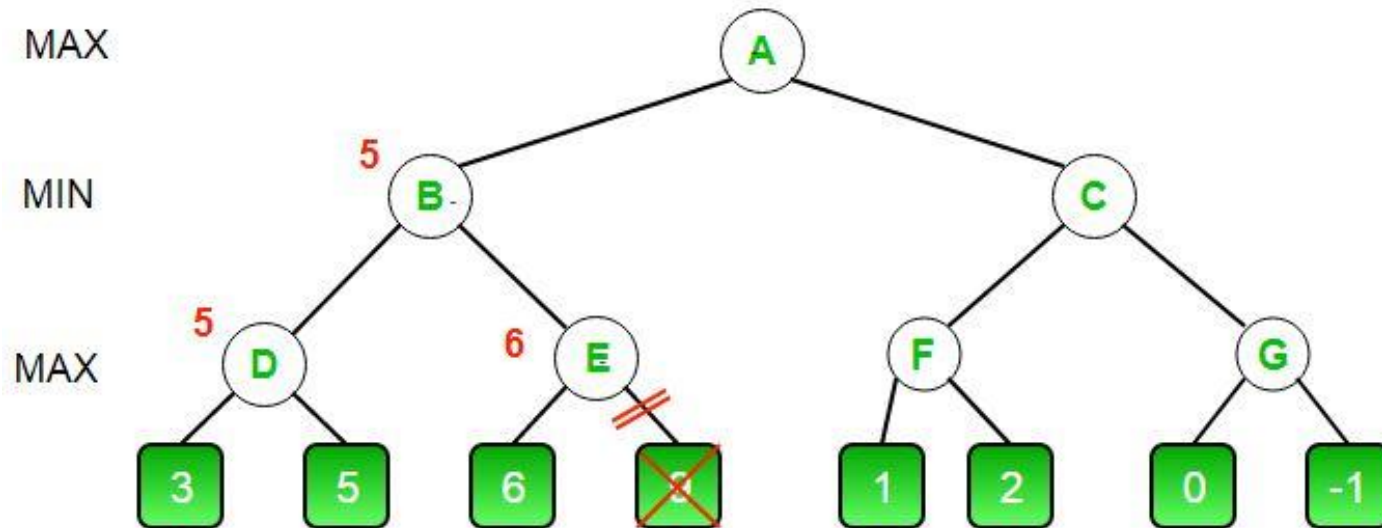
It **recursively** evaluates all possible game states down to the terminal states (e.g., win, lose, or draw).

Backtracking from the terminal states, it assigns a score to each move by choosing the best option for the player whose turn it is.



# Alpha-Beta Pruning

Alpha-Beta Pruning is an optimization technique for the minimax algorithm. It reduces the number of nodes evaluated in the game tree by **pruning** or eliminating branches that cannot influence the final decision.



# How it Works

**Alpha ( $\alpha$ ):** The best value the maximizer currently can guarantee.

**Beta ( $\beta$ ):** The best value the minimizer currently can guarantee.

During the traversal:

If a branch's potential outcome is worse than an already evaluated branch for either player, that branch is pruned (skipped).

The order in which nodes are explored determines the effectiveness of pruning.

**Good move ordering** can significantly reduce the number of nodes evaluated.

(There may be cases where no pruning occurs but it is faster most of the times.)

# Implementation

Topics like graph theory, and recursion play crucial roles in modeling and solving the game.

# Introduction to Gomoku

**Gomoku** rules and win condition

**Graph Model:** Gomoku's grid can be represented as a graph where:

- **Vertices (Nodes)** represent each board position.
- **Edges (Connections)** represent possible moves between adjacent positions (up, down, left, right, or diagonally).
- **Significance:** This representation enables the game to leverage graph algorithms to analyze the board for optimal moves and winning patterns.
- **Recursion in Gomoku:**
- **Exploring Moves Recursively:** AI can use recursion to evaluate multiple future moves, creating a **game tree** structure. Each recursive call simulates a potential move, building layers of possible future game states.
- **AI Strategy:**
  - **Block Opponent Moves:** The AI recursively checks potential moves that could lead to an opponent's victory, blocking these to defend.
  - **Find Winning Paths:** It evaluates sequences of moves to identify paths where it can form five in a row.

- **Graph Theory:** By modeling the board as a graph, the AI can:
  - Track connections between moves.
  - Evaluate clusters of pieces that could lead to potential wins.
- **Recursion:** Recursively explores all potential moves, simulating different sequences and choosing the path that maximizes the chance of winning.
- **Python:** Recommended for ease of use and development speed, especially with libraries that simplify board representation and recursive functions.
- **Basic Algorithm for AI Move Selection**
- **Recursive Evaluation:**
  - Start with an empty board state, checking all valid moves.
  - For each move, recursively simulate possible opponent responses.
  - Limit recursive checks

# CONCLUSION

- **Graph Theory:**

Helps us map and understand game boards and other complex structures by showing connections and paths, making strategic planning easier.

- **Recursion:**

A flexible tool for solving repetitive tasks and layered challenges, especially useful in game logic to solve problems efficiently.

- **AI in Board Games**

AI can be enhanced for games like Gomoku by using graph theory and recursion to develop smarter, more adaptable strategies.

## Final Thought:

Graph theory and recursion deepen our understanding of mathematical structures while transforming practical areas like game development. These concepts enable smarter AI, richer game mechanics, and effective problem-solving, bridging gap between theoretical knowledge and real-world applications.

# Thank you for your attention!

Feel free to ask any questions, share  
thoughts or discuss further.

Looking forward to insightful discussions!

