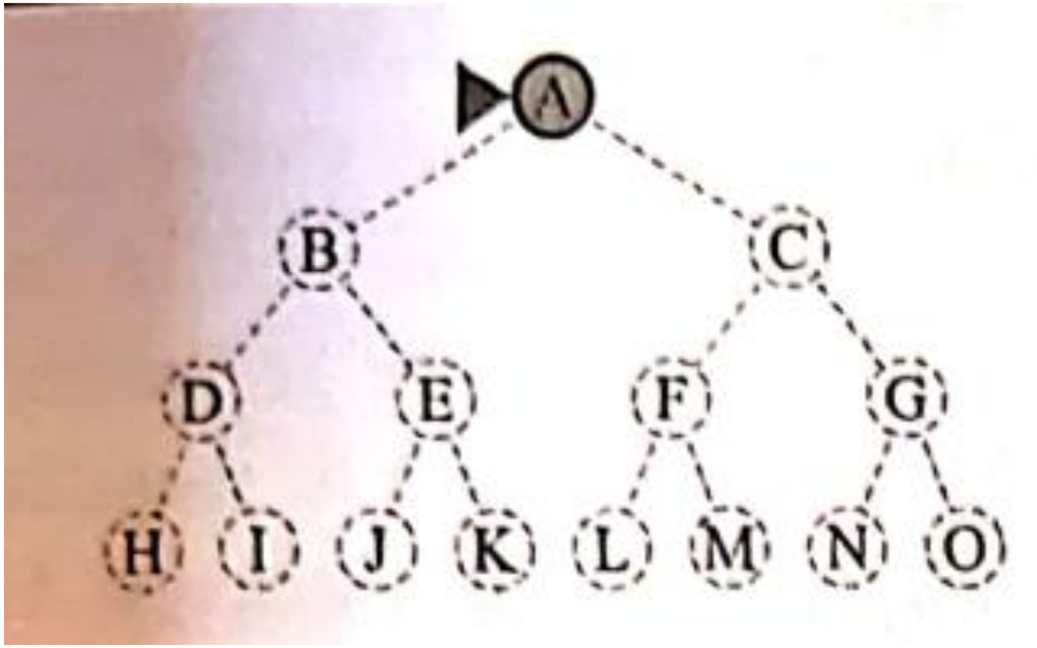
BCSE306
Artificial Intelligence
Module 3: Genetic Algorithm



Dr. Durgesh Kumar

Assistant Professor (Senior)

School of Computer Science and Engineering (SCOPE),

VIT Vellore,

Lecture Outline

- The Problem solving based on Search
  - Search Algorithm and Search Tree
    - Search tree, node, parent, successor, predecessor, fringe
    - Search data structures
    - Different types of Queue – Priority queue, FIFO, LIFO
    - Measuring Problem solving performance
  - Uninformed search
    - Breadth First Search (BFS)
    - Depth First Search (DFS)
    - Uniform cost Search (Dijkstra)

History of Genetic Algorithm (GA)

- As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments

History of Genetic Algorithm (GA)

- A **genetic algorithm** is a **heuristic search** algorithm that is inspired by Charles Darwin's theory of natural evolution.
- This algorithm reflects the process of **natural selection** where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.
- Genetic Algorithms are being widely used in different **real-world applications**, for example, **image processing**, **Designing electronic circuits**, **code-breaking**, and **artificial creativity**.

What is GA ?

- A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- (GA)s are categorized as global search heuristics.
- (GA)s are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

What is GA ?

- The evolution usually starts from a population of randomly generated individuals and happens in generations.

- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified to form a new population.

What is GA ?

- The new population is used in the next iteration of the algorithm.
- The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

No convergence rule or guarantee!

Vocabulary of GA ?

- **Individual** - Any possible solution
- **Population** - Group of all individuals
- **Fitness** – Target function that we are optimizing (each individual has a fitness)
- **Trait** - Possible aspect (features) of an individual
- **Genome** - Collection of all chromosomes (traits) for an individual.

GA Example-1

- Consider the function of maximizing the function

$$f(x) = x^2$$

- where x in permitted to vary between 0 to 31

Vocabulary of GA ?

- **Individual** - Any possible solution
- **Population** - Group of all individuals
- **Fitness** – Target function that we are optimizing (each individual has a fitness)
- **Trait** - Possible aspect (features) of an individual
- **Genome** - Collection of all chromosomes (traits) for an individual.

Phases/Steps of GA

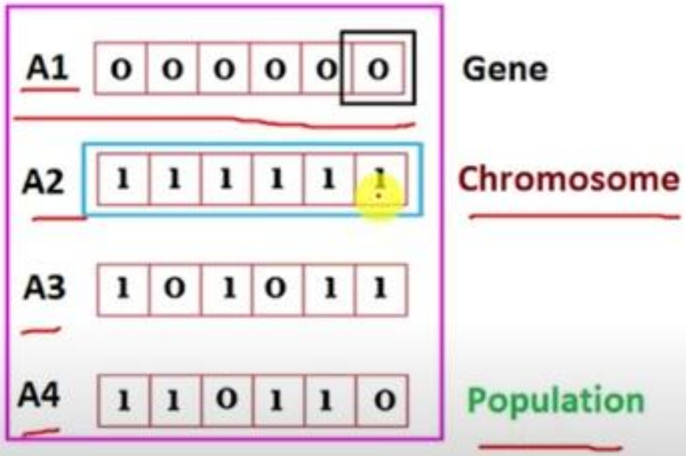
There are five phases in Genetic Algorithm:

- Initialization
- Fitness Assignment
- Selection
- Crossover (Reproduction)
- Termination

Phases/Steps of GA- Initialization

Initial Population

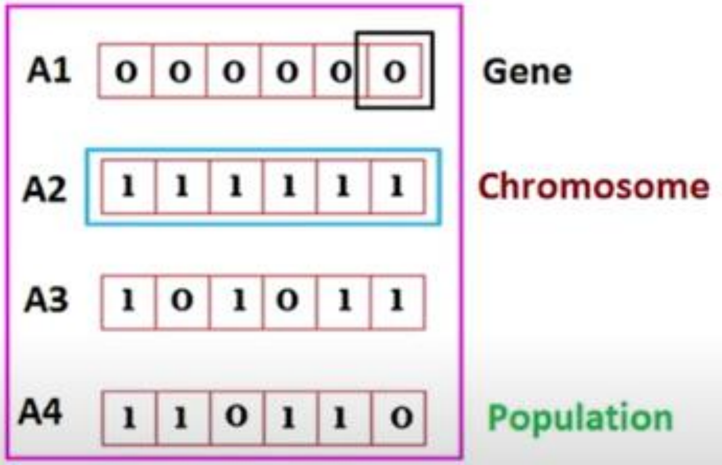
- The process begins with a set of individuals which is called a Population.
- Each individual is a solution to the problem you want to solve known as Chromosome.
- An individual is characterized by a set of parameters (variables) known as Genes.
- Genes are joined into a string to form a Chromosome (solution).



Phases/Steps of GA-Fitness function

Fitness Function

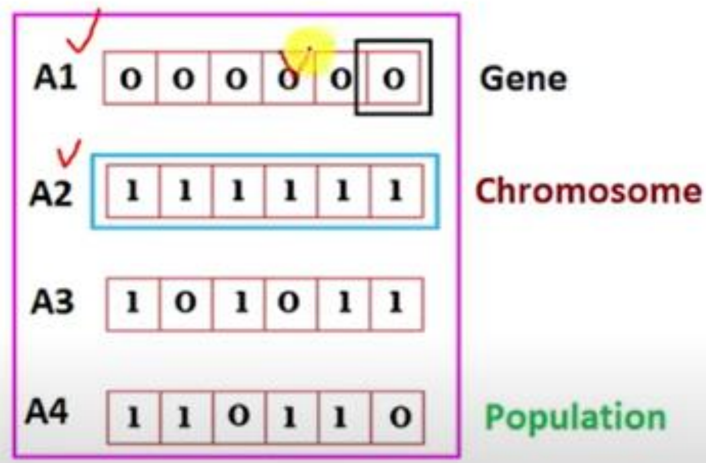
- The fitness function determines **how fit** an individual is? (the ability of an individual to compete with other individuals).
- It gives a fitness score to each individual.



Phases/Steps of GA- Selection

Selection

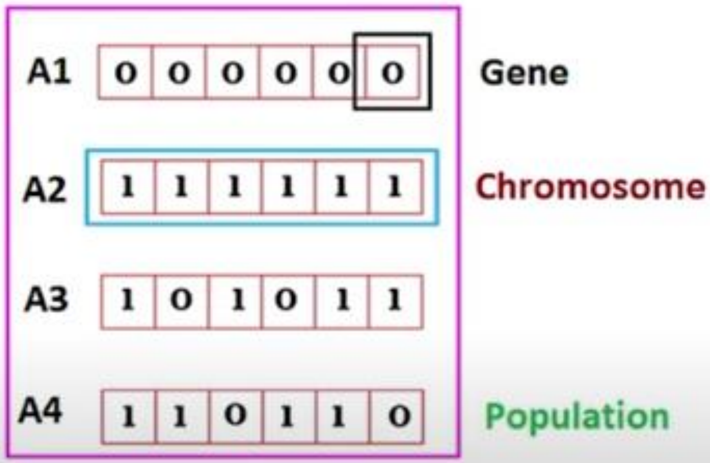
- The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.



Phases/Steps of GA- Selection

There are three types of Selection methods available, which are:

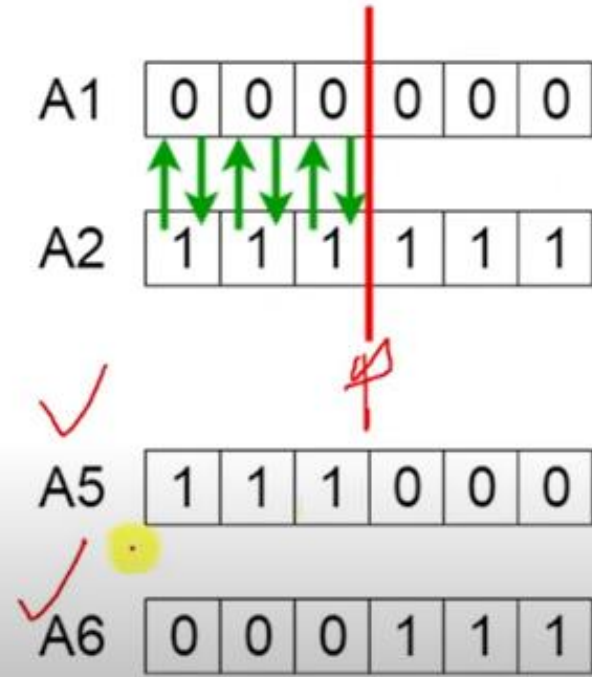
1. Roulette wheel selection
2. Tournament selection
3. Rank-based selection



Phases/Steps of GA- Cross Over

Offspring

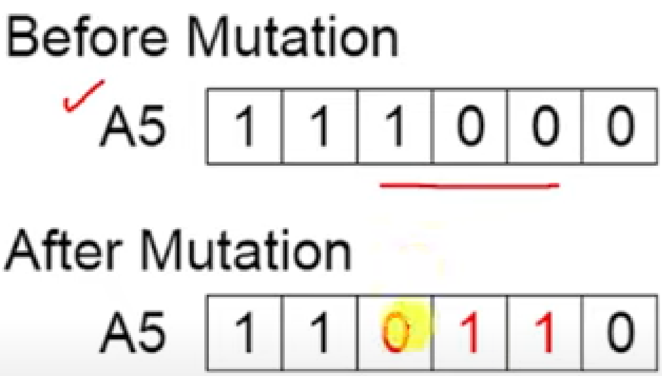
- **Offspring** are created by exchanging the genes of parents among themselves until the crossover point is reached.
- The new offspring are added to the population.



Phases/Steps of GA-Mutation

Mutation

- In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability.
- This implies that some of the bits in the bit string can be flipped.

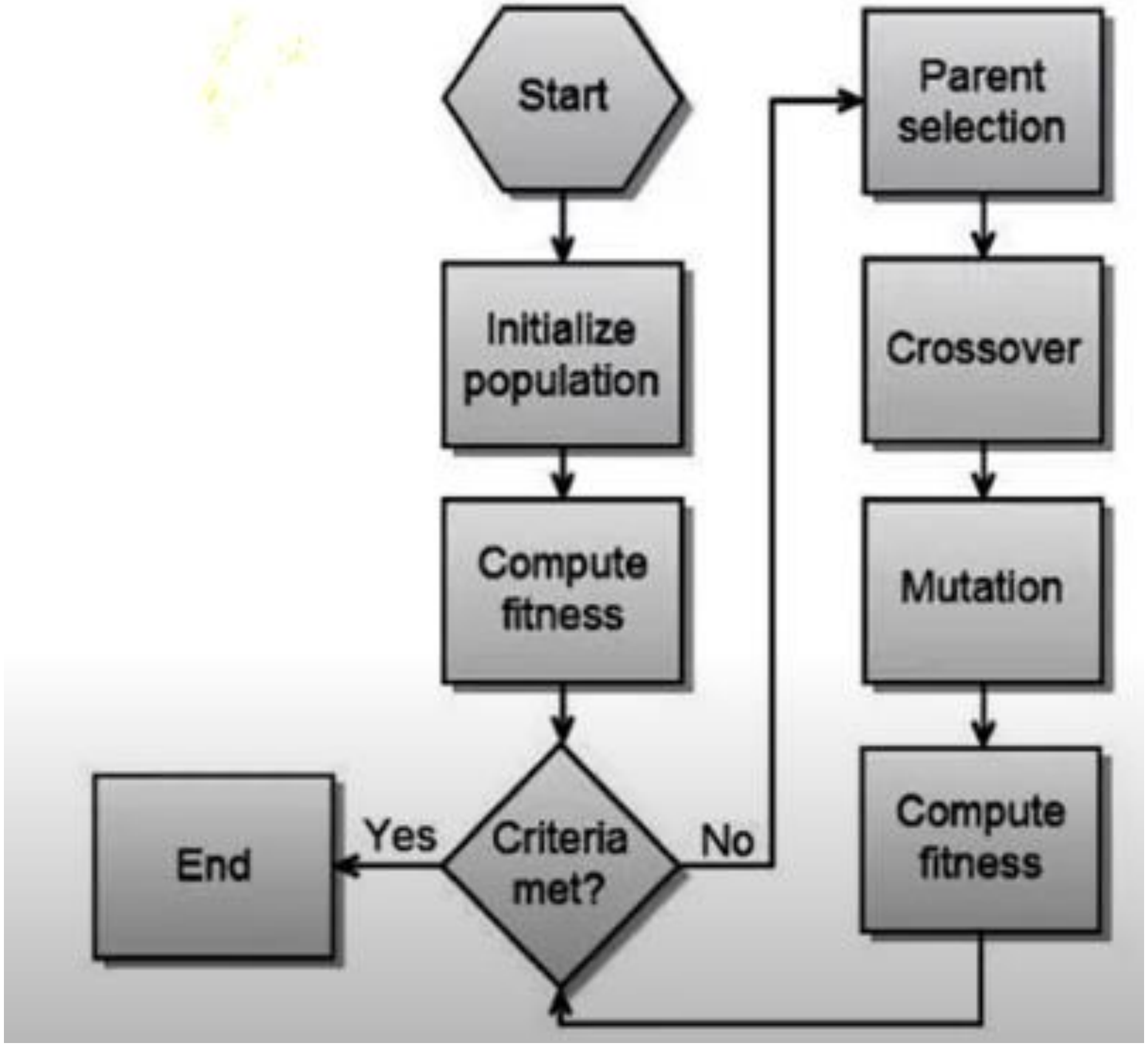


Phases/Steps of GS - Termination

Termination

- The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation).

Genetic Algorithm Flowchart



GA Example-1

- Consider the function of maximizing the function

$$f(x) = x^2$$

- where x in permitted to vary between 0 to 31

GA Example-1

- **Select Encoding Technique**
- The minimum value is 0 and maximum value is 31
- Using a five-bit binary integer, numbers between **0 (00000)** and **31 (11111)** can be obtained.
- The objective function here *is* $f(x) = x^2$,which *is* to be maximized.

GA Example-1

Select Initial Population

- To start with, select initial population are random.
- Here initial population of size 4 is chosen, but any number of populations can be selected based on the requirement and application.

GA Example-1

String No.	Initial Population (Randomly Selected)	X Value	Fitness $f(x) = x^2$	Prob	% Prob	Expected Count	Actual Count
1							
2							
3							
4							
Sum							
Average							
Maximum							

GA Example-1

String No.	Initial Population (Randomly Selected)	X Value	Fitness $f(x) = x^2$	Prob	% Prob	Expected Count	Actual Count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	181	0.3126	31.26	1.2502	1
Sum			1155	1.0	100	4	4
Average			288.75	0.25	25	1	1
Maximum			625	0.5411	54.11	2.1645	2

GA Example-1

String No.	Mating Pool	Crossover Point	Offspring after crossover	X Value	Fitness $f(x) = x^2$
1	01100	4	01101	13	169
2	11001		11000	24	576
3	11001	2	11011	27	729
4	10011		10001	17	289
Sum					1763
Average					440.75
Maximum					729

GA Example-1

String No.	Offspring after crossover	Mutation Chromosome for flipping	Offspring after mutation	X Value	Fitness $f(x) = x^2$
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00101	10100	20	400
Sum					2546
Average					636.5
Maximum					841

Genetic Algorithm Example-1

Search Data structures - Queue

Three kinds of queues used in search algorithm:

- **Priority queue:** pops the node with minimum coset according to some evaluation function f. Used in best-first search.

- **FIFO queue**: QUEUE pops the node which was inserted first, used in breadth-first search.

- **LIFO queue** : STACK pops the most recently added node, used in depth-first search.

Search Data structures - Queue

Three kinds of queues used in search algorithm:

▪**Priority queue**: pops the node with minimum coset according to some evaluation function f. Used in best-first search.

▪**FIFO queue**: QUEUE pops the node which was inserted first, used in breadth-first search.

▪**LIFO queue** : STACK pops the most recently added node, used in depth-first search.

Measuring problem-solving performance

Three kinds of queues used in search algorithm:

- **Completness:** Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not ?
- **Cost Optimality**: Does it find the a solution with lowest cost of all solutions ?
- **Time Complexity** : How long it takes to find a solution?
- **Space Complexity :** How much memory is needed to perform search?

Measuring problem-solving performance

Time and space complexity are measured in terms of

- *b:* maximum branching factor of the search tree
- *d:* depth of the least-cost solution
- *m*: maximum depth of the state space (may be ∞)

Breadth First Search (BFS)

- Initialize a FIFO queue with start state.
- Visit every node level by level and keep Checking for the Goal state.

BFS - Algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

BFS – Example 1

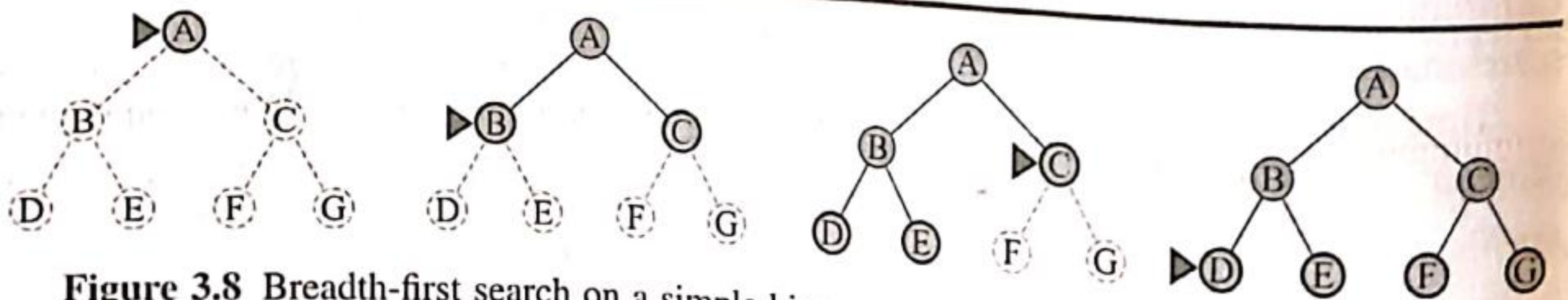
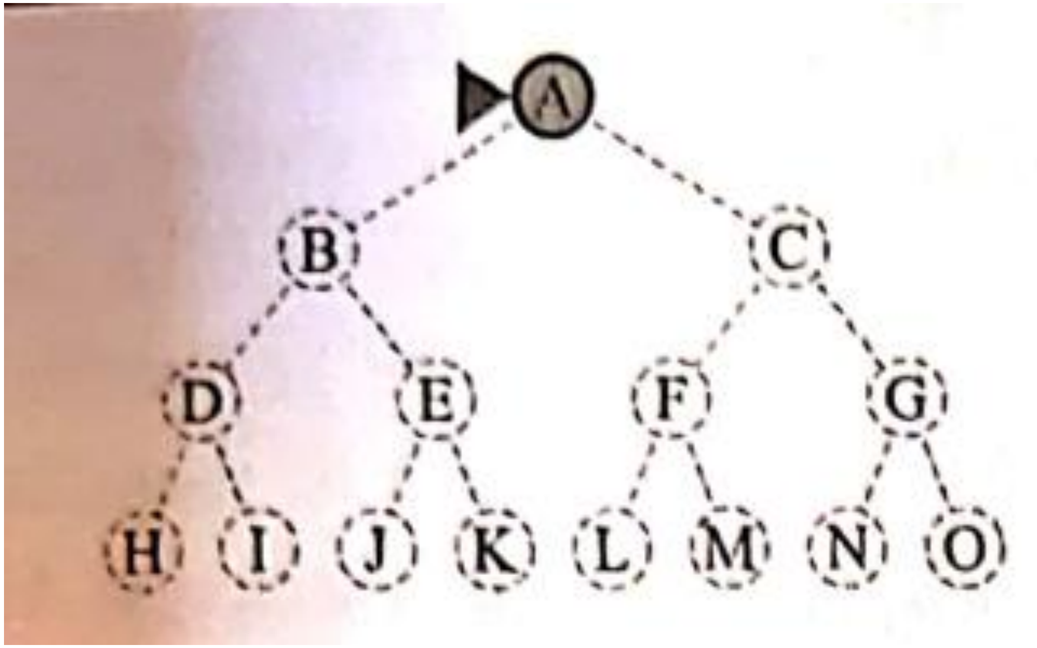
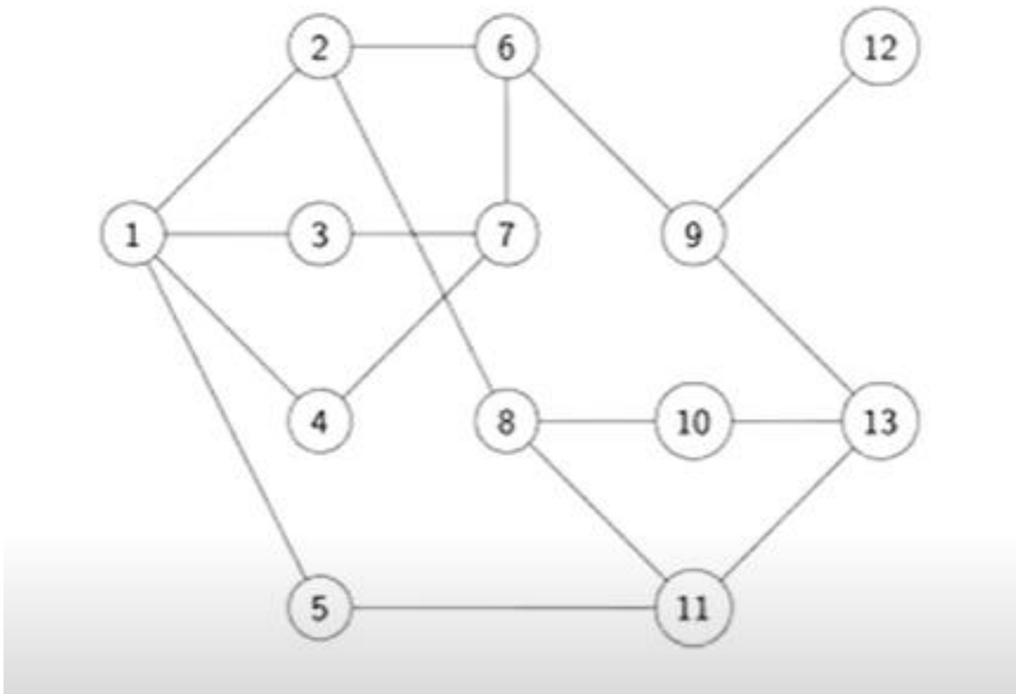


Figure 3.8 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

BFS – Example 2



BFS – Example 3



BFS – properties

- Complete? Yes (if *b* is finite)

- Time? ***O(b^d)***
  - *[1+ b + b^2 + …+ b^d] = O(b^d)*
- Space? *O(b^d)* (keeps every node in memory)

- Optimal? Yes (if cost = 1 per step)

.- where b is the branching factor for each node and
- d is the maximum depth at which goal state is found

- BFS always finds a solution with minimum number of actions

BFS – Drawback

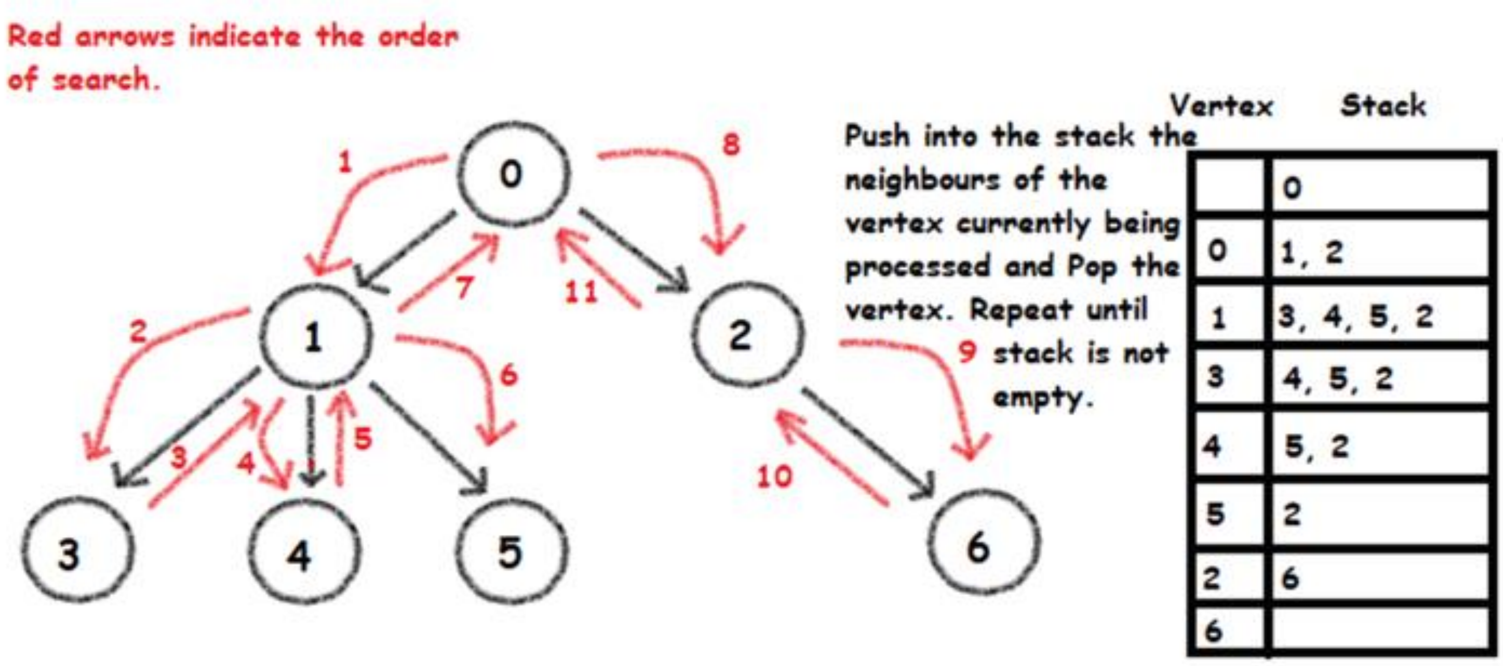
- When b=10, and d=10, BFS would take 3 hrs and 10 TB of memory.

- With d=14, search would take 3.5 years even with infinite memory.

- Exponential complexity search problems can not be solved by uninformed search for any but smallest instance.

Depth First Search (DFS)

- **Depth-First Search (DFS)** always expands the **deepest node** in the frontier first.



Depth First Search

Properties of DFS

- Depth-First Search (DFS) search is **not cost-optimal**; it returns the first solution it finds, even if it is not cheapest.

- For finite state spaces that are trees, it is efficient and complete.

- For acyclic state space, it may end up expanding the same state many times via different paths, but will (eventually) systematically explore the entire space.

Properties of DFS -2

- In the cyclic state spaces, it can get stuck in an infinite loop; therefore, some implementations of **depth-first search** check each new node for **cycles.**

- In infinite state spaces, DFS search is not systematic, it may stuck going down in **infinite path**, even if there are no **cycles**.

- Therefore, DFS search is **incomplete**.

Advantage of DFS -2

- For problems, where a tree like search is **feasible**, **vanilla DFS** has much smaller needs for memory.

- As **vanilla DFS** don't keep a reached table at all.

- For the finite tree-shaped state-space, **space complexity** of **DFS** search = **O(*b m*)**
    - Where ***b*** is branching factor
    - ***m*** is maximum depth of the tree.

Analysis of DFS -2

- Complete? No: fails in infinite-depth spaces, spaces with loops

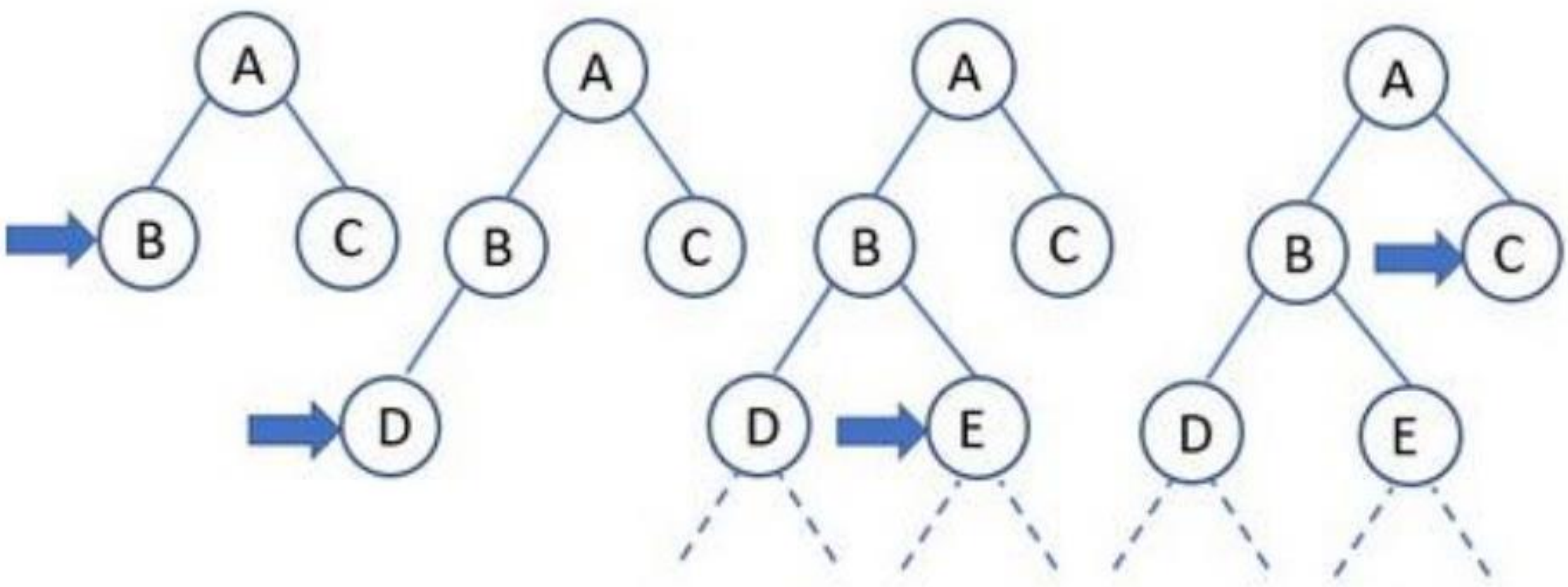
  → complete in finite spaces

- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first

- Space? $O(bm)$, i.e., linear space!

Depth limited DFS

- **depth-first search** with **depth limit *l***,
  i.e., it is assumed that nodes at **depth *l* have no successors**



Iterative Deepening DFS

- **DFS** which tries to find best value of l **depth-first search** with **depth limit *l***, for l = 1, 1, 2, .. and so on
  i.e., it is assumed that nodes at **depth *l* have no successors**

Iterative Deepening DFS

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Comparison of different algorithm

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
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	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$


Thank You
For Your Attention!
Any Questions