Problem Solving as State Space Search

State space search

- It is a process used in the field of computer science, including artificial intelligence (AI), in which states of an instance are considered, with the goal of finding a goal state with a desired property.

Can the astronaut get its supplies safely across the canal?

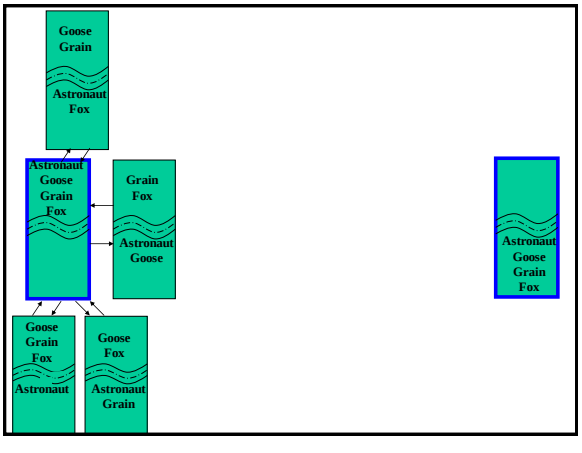
Astronaut
Goose
Grain
Fox

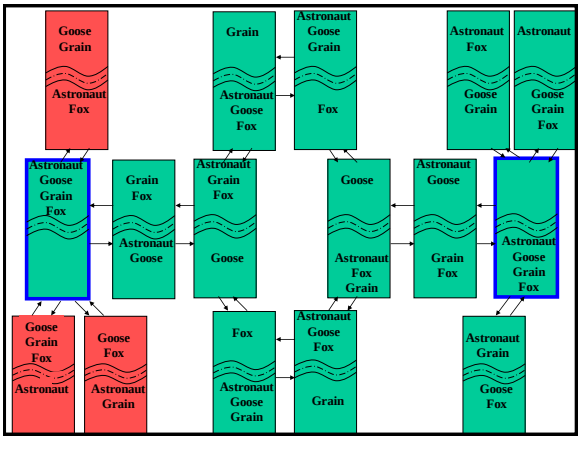
River

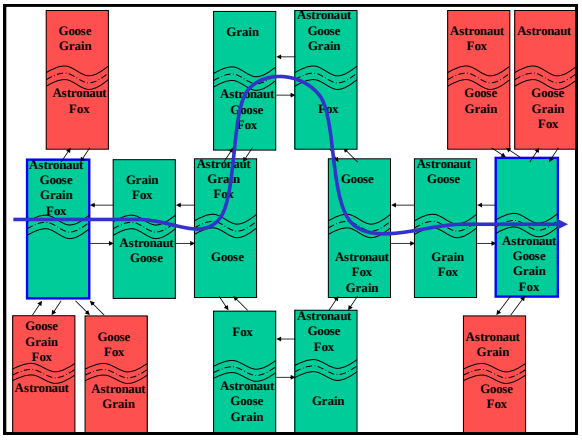
- Astronaut + 1 item allowed in the boat.
- Goose alone eats Grain
- Fox alone eats Goose

Problem Solving as State Space Search

- Formulate Goal
  - Astronaut, Fox, Goose & Grain across river
- Formulate Problem
  - States
    - Location of Astronaut, Fox, Goose & Grain at top or bottom river bank
  - Operators
    - Astronaut drives boat along 1 or 0 items to other bank.
- Generate Solution
  - Sequence of Operators (or States)
    - Move(goose,astronaut), Move(astronaut), . . .







Classes of Search

Blind (uninformed)	Depth-First Breadth-First Iterative-Deepening	Systematic exploration of whole tree until the goal is found.
Heuristic (informed)	Hill-Climbing Best-First Beam	Uses heuristic measure of goodness of a node,e.g. estimated distance to goal.
Optimal (informed)	Branch&Bound A*	Uses path "length" measure. Finds "shortest" path. A* also uses heuristic

A Graph

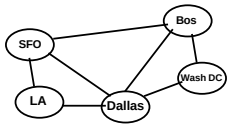
A Graph G is represented as a pair <V,E>, where:

- V is a set of vertices {v1 …}
- E is a set of (directed) edges {e1, …}

An edge is a pair <v1, v2> of vertices, where

- **v2** is the **head** of the edge and
- **v1** is the **tail** of the edge

Airline Routes



< {Bos, SFO, LA, Dallas, Wash DC}

{<SFO, Bos>,

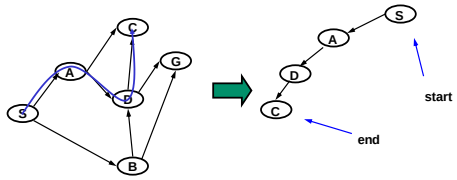
<SFO, LA>

<LA, Dallas>

<Dallas, Wash DC>

. . .} >

A Solution is a State Sequence: Problem Solving Searches Paths

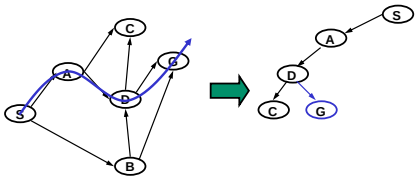


A path is a sequence of edges (or vertices)
<S, A, D, C>

Simple path has no repeated vertices.

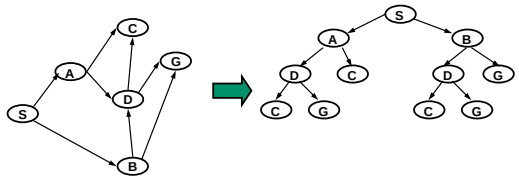
For a cycle, start = end.

A Solution is a State Sequence: Problem Solving Searches Paths



Represent searched paths using a tree.

A Solution is a State Sequence: Problem Solving Searches Paths



Represent searched paths using a tree.

Graph Searching/Traversing

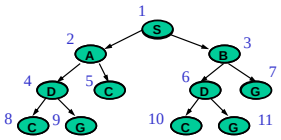
- Given: a graph G = (V, E), directed or undirected
- Explore every vertex and every edge
- Build a tree on the graph
  - Pick a vertex as the root
  - Choose certain edges to produce a tree

Breadth First Search

- Builds a tree over the graph
  - Pick a source vertex to be the root
  - Find ("discover") its children, then their children, etc. (Expand frontier of explored vertices across the breadth of the frontier)

Breadth First Search (BFS)

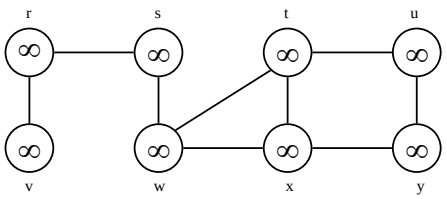
Idea: After visiting node
- Visit siblings, then children
- Visit relatives left to right (top to bottom)



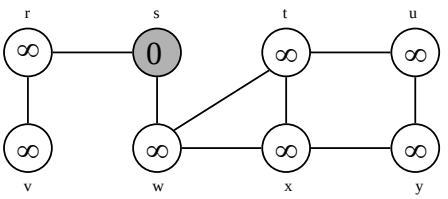
Breadth-First Search

- Associate vertex "colors"
  - **White** vertices have not been discovered
    - All vertices start out white
  - **Grey** vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - **Black** vertices are discovered and fully explored
    - They are adjacent only to black and grey vertices
- Explore vertices by scanning adjacency list of grey vertices

Breadth-First Search: Example

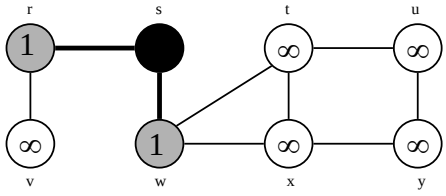


Breadth-First Search: Example



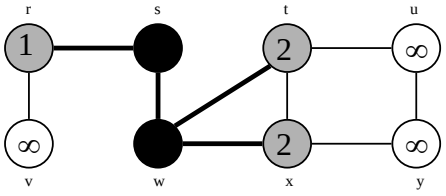
Q: s

Breadth-First Search: Example



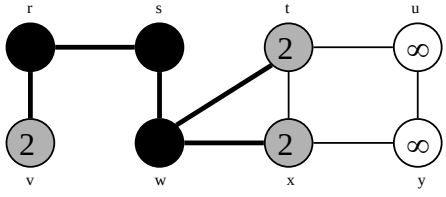
Q: | w | r |

Breadth-First Search: Example



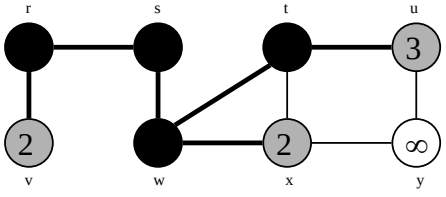
Q: | r | t | x |

Breadth-First Search: Example



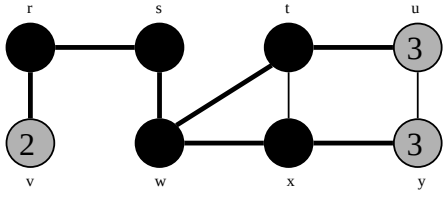
Q: | t | x | v |

Breadth-First Search: Example



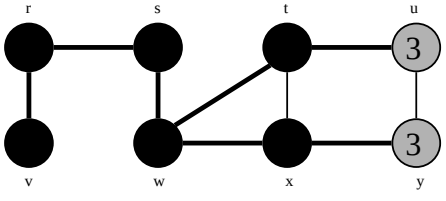
Q: | x | v | u |

Breadth-First Search: Example



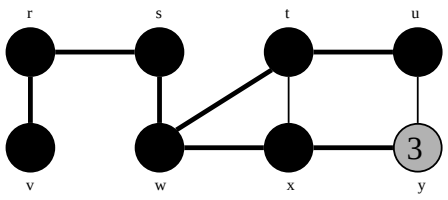
Q: | v | u | y |

Breadth-First Search: Example



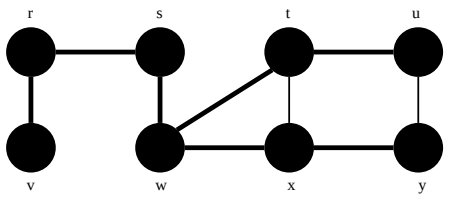
Q: | u | y |

Breadth-First Search: Example



Q: y

Breadth-First Search: Example



Q: Ø

Depth First Search

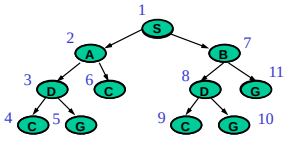
- Depth-first search is another strategy for exploring a graph
  - Explore "deeper" in the graph whenever possible
  - Edges are explored out of the most recently discovered vertex v that still has unexplored edges
  - When all of v's edges have been explored, backtrack to the vertex from which v was discovered

Depth First Search

- Vertices **initially** colored **white**
- Then **colored gray** when discovered
- Then **black** when **finished**

Depth First Search (DFS)

Idea:
- Visit children, then siblings
- Visit siblings left to right, (top to bottom).



Assuming that we pick the first element of Q,
Then where do we add path extensions to the Q?

Iterative deepening search

- To avoid the infinite depth problem of DFS, we can decide to **only search until depth L,** i.e. we don't expand beyond depth L.
  → Depth-Limited Search

- What of solution is deeper than L? → Increase L iteratively.
  → Iterative Deepening Search

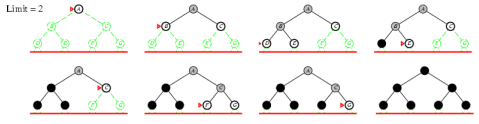
Iterative deepening search L=0

33

Iterative deepening search L=1

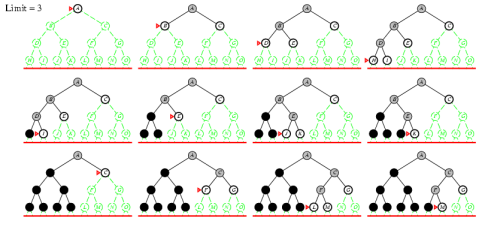
34

Iterative deepening search L=2



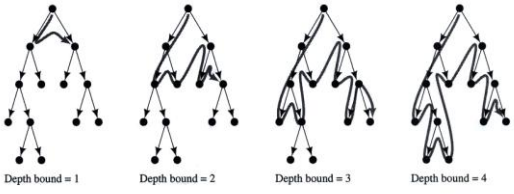
35

Iterative deepening search L=3



36

Example IDS



Stages in Iterative-Deepening Search

37