Problem Solving as State Space Search

Brian C.Williams

16.410-13

Sep 14th, 2004

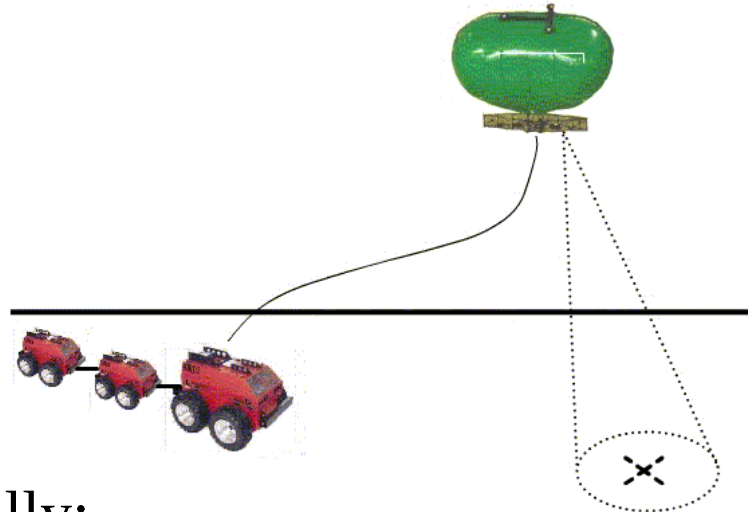
Slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA

Assignments

- Remember:

  Problem Set #1: Simple Scheme and Search due Monday, September 20th, 2003.

- Reading:
  - Solving problems by searching: AIMA Ch. 3



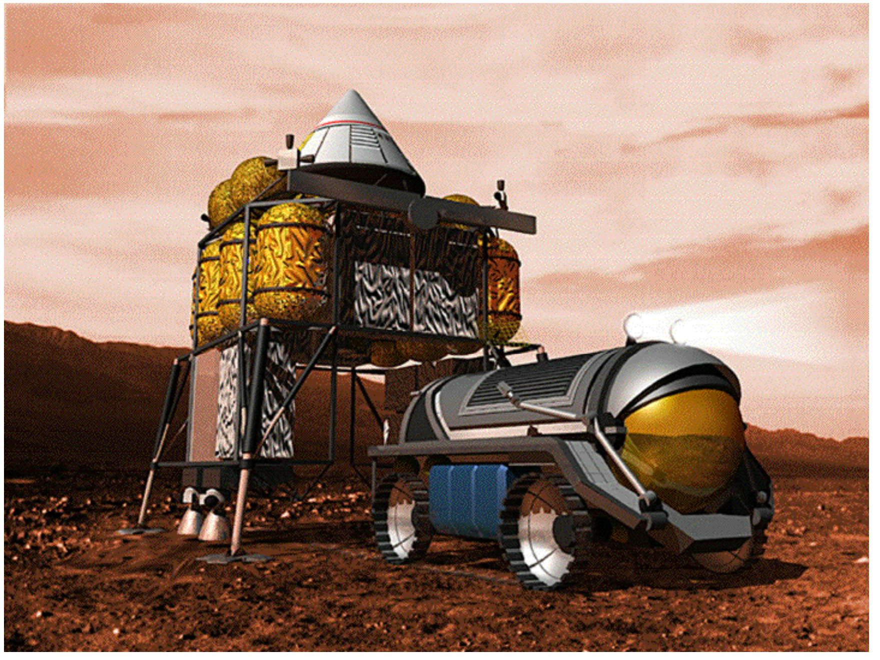
Complex missions must carefully:

- Plan complex sequences of actions
- Schedule actions
- Allocate tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

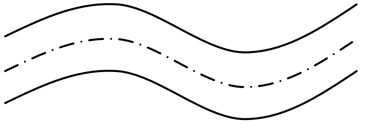
⇨ Most AI problems, like these, may be formulated as state space search.

Outline

- Problem Formulation
  - Problem solving as state space search
- Mathematical Model
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search



Astronaut
Goose
Grain
Fox

Rover 

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

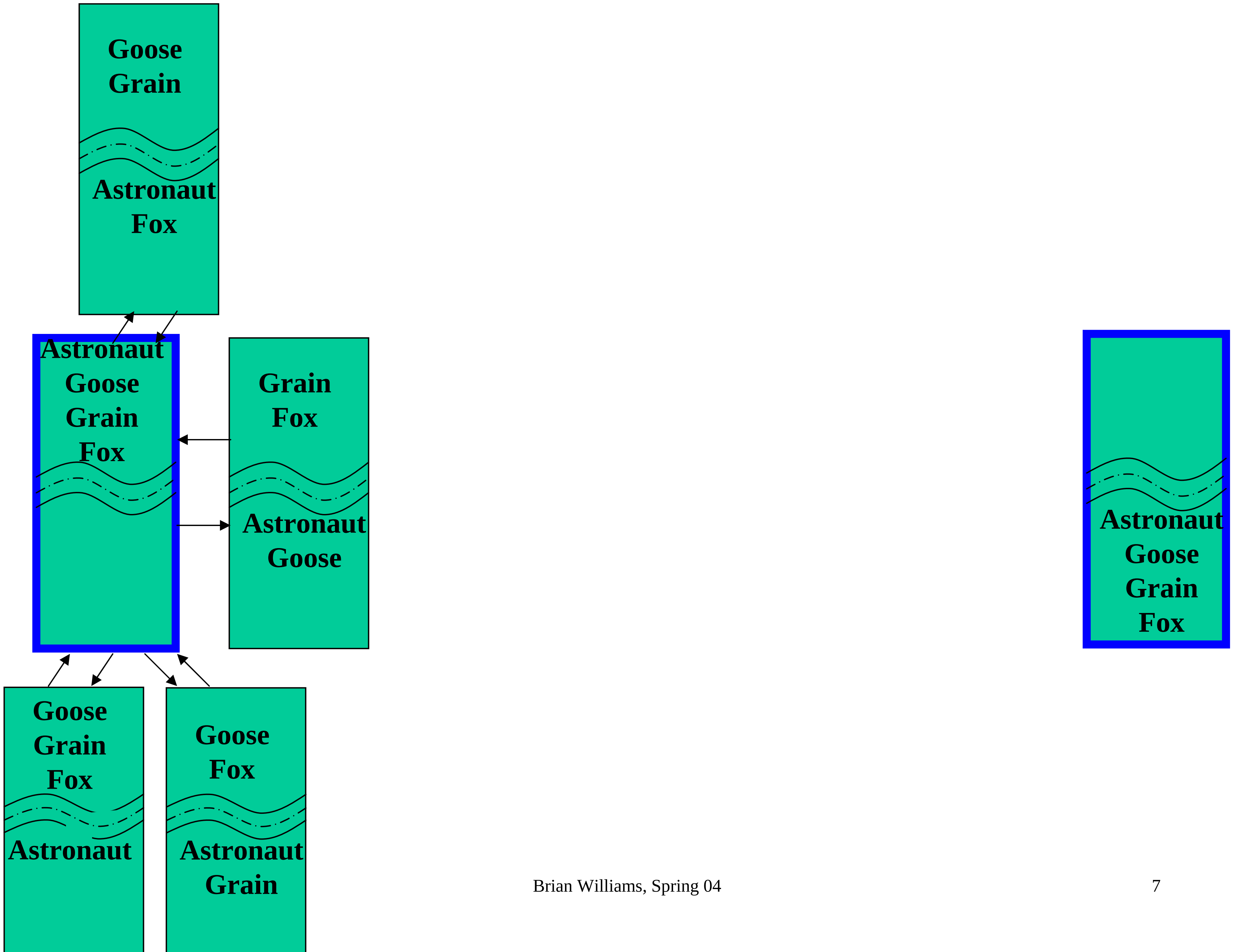
Can the astronaut get its supplies safely across the Martian canal?

Early AI: What are the universal problem solving methods?

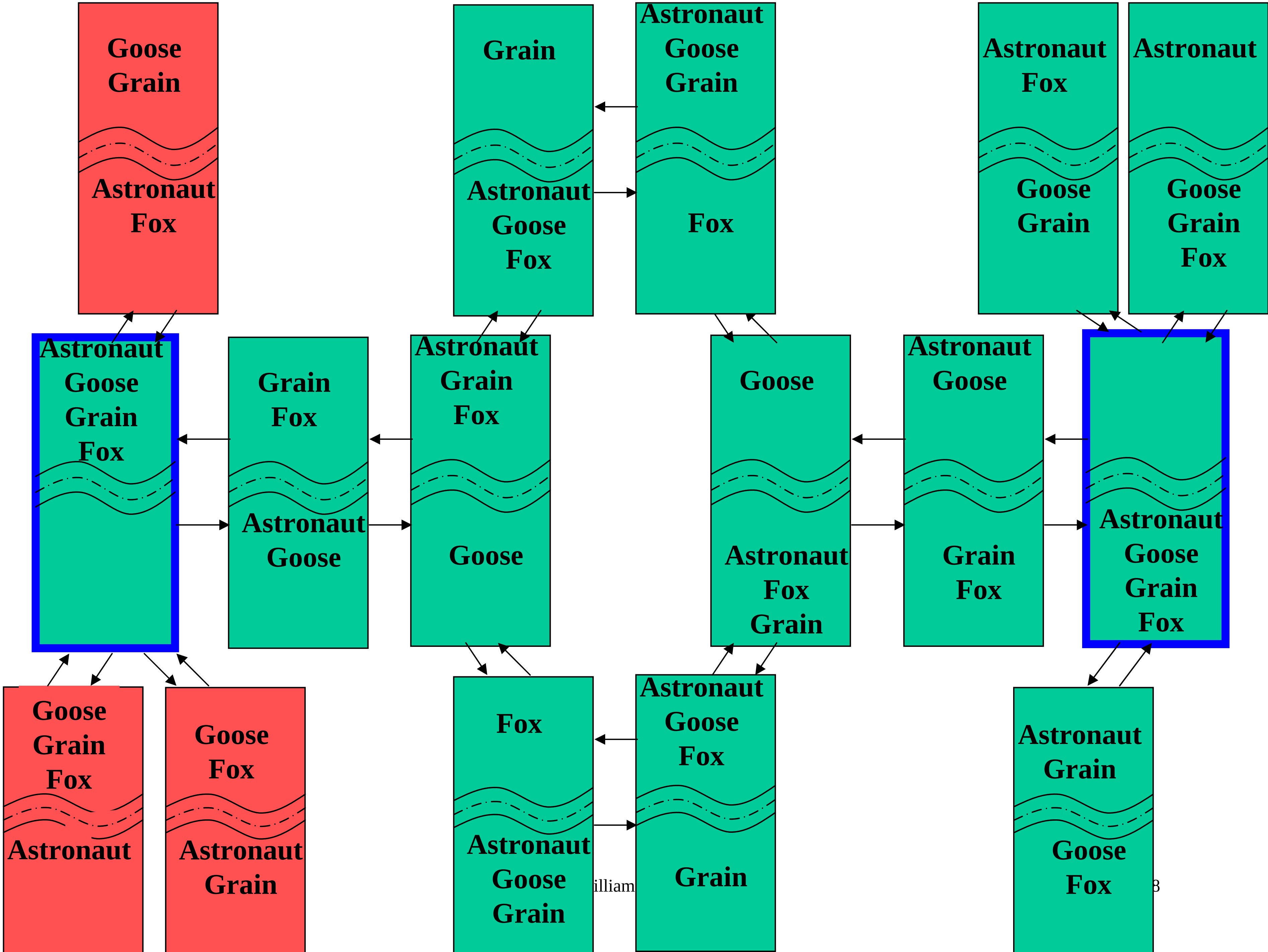
Simple $\not\longrightarrow$ Trivial

Problem Solving as State Space Search

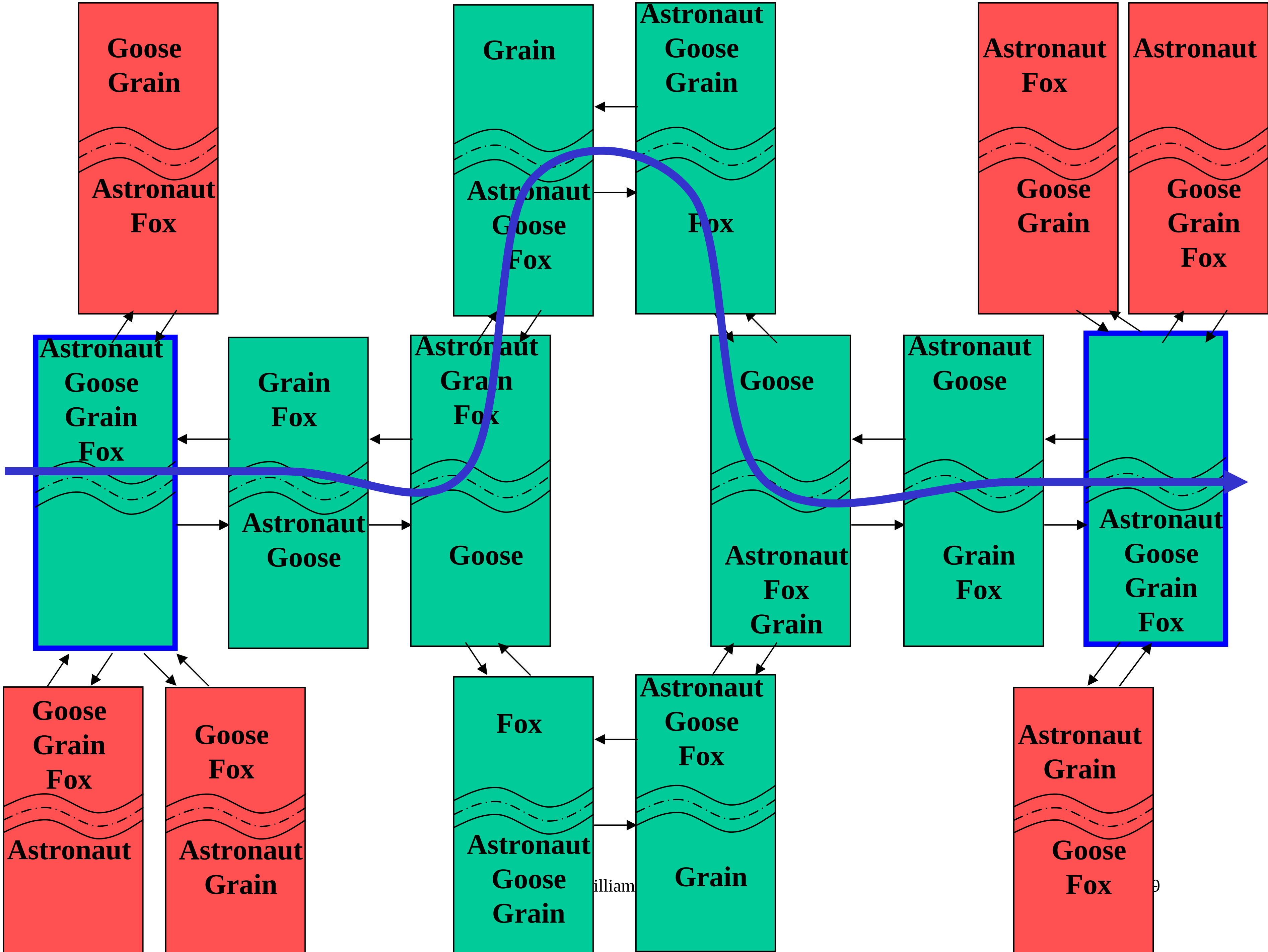
- Formulate Goal
  - State
    - Astronaut, Fox, Goose & Grain across river
- Formulate Problem
  - States
    - Location of Astronaut, Fox, Goose & Grain at top or bottom river bank
  - Operators
    - Astronaut drives rover and 1 or 0 items to other bank.
- Generate Solution
  - Sequence of Operators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



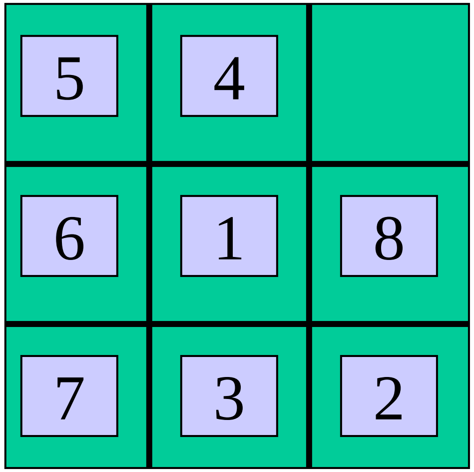
Goose
Grain
Astronaut
Fox
Astronaut
Goose
Grain
Fox
Grain
Fox
Astronaut
Goose
Astronaut
Goose
Grain
Fox
Goose
Grain
Fox
Astronaut
Goose
Fox
Astronaut
Grain



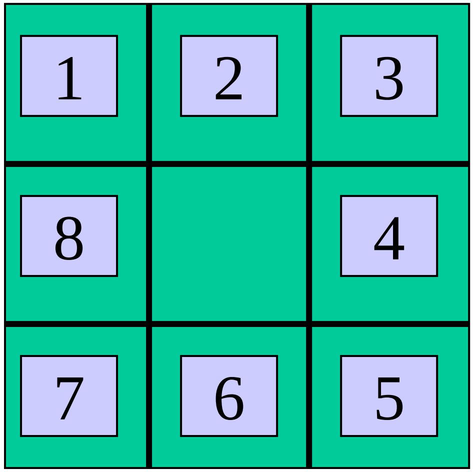
Goose Grain
Astronaut Fox
Grain
Astronaut Goose Fox
Astronaut Goose Grain
Fox
Astronaut Fox
Goose Grain
Astronaut
Goose Grain Fox
Astronaut Goose Grain Fox
Grain Fox
Astronaut Goose
Astronaut Grain Fox
Goose
Goose
Astronaut Fox Grain
Astronaut Goose
Grain Fox
Astronaut Goose Grain Fox
Goose Grain Fox
Astronaut
Goose Fox
Astronaut Grain
Fox
Astronaut Goose Grain
Astronaut Goose Fox
Grain
Astronaut Grain
Goose Fox


Goose
Grain
Astronaut
Fox
Grain
Astronaut
Goose
Fox
Astronaut
Goose
Grain
Fox
Astronaut
Fox
Goose
Grain
Astronaut
Goose
Grain
Fox
Astronaut
Goose
Grain
Fox
Grain
Fox
Astronaut
Goose
Astronaut
Grain
Fox
Goose
Goose
Astronaut
Fox
Grain
Astronaut
Goose
Grain
Fox
Astronaut
Goose
Grain
Fox
Goose
Grain
Fox
Astronaut
Goose
Fox
Astronaut
Grain
Fox
Astronaut
Goose
Grain
Astronaut
Goose
Fox
Grain
Astronaut
Grain
Goose
Fox

Example: 8-Puzzle



Start



Goal

- States: integer location for each tile AND …
- Operators: move empty square up, down, left, right
- Goal Test: goal state as given

Example: Planning Discrete Actions

- Swaggert & Lovell work on Apollo 13 emergency rig lithium hydroxide unit.
  - Assembly
- Mattingly works in ground simulator to identify new sequence handling severe power limitations.
  - Planning & Resource Allocation
- Mattingly identifies novel reconfiguration, exploiting LEM batteries for power.
  - Reconfiguration and Repair

Planning as State Space Search: STRIPS Operator Representation

Initial state:

(and (hose a)
(clamp b)
(hydroxide-unit c)
(on-table a)
(on-table b)
(clear a)
(clear b)
(clear c)
(arm-empty))

goal (partial state):

(and (connected a b)
(attached b a)))

Available actions Strips operators

precondition: (and (clear hose)
(on-table hose)
(empty arm))

pickup hose

effect: (and (not (clear hose))
(not (on-table hose))
(not (empty arm))
(holding arm hose)))

Note: strips doesn't allow derived effects; you must be complete!

⇨ Effects specify how to **change** the set of assertions.

STRIPS Action Assumptions

precondition: (and (clear hose)
(on-table hose)
(empty arm))

pickup hose

effect: (and (not (clear hose))
(not (on-table hose))
(not (empty arm))
(holding arm hose)))

- Atomic time.
- Agent is omniscient
(no sensing necessary).
- Agent is sole cause of change.
- Actions have deterministic effects.
- No indirect effects.

STRIPS Action Schemata

- Instead of defining:

  pickup-hose and **pickup-clamp** and …

- Define a schema (with variables ?v):

```
(:operator pick-up
        :parameters ((hose ?ob1))
        :precondition (and (clear ?ob1)
                           (on-table ?ob1)
                           (empty arm))
        :effect (and (not (clear ?ob1))
                     (not (on-table ?ob1))
                     (not (empty arm))
                     (holding arm ?ob1)))
```

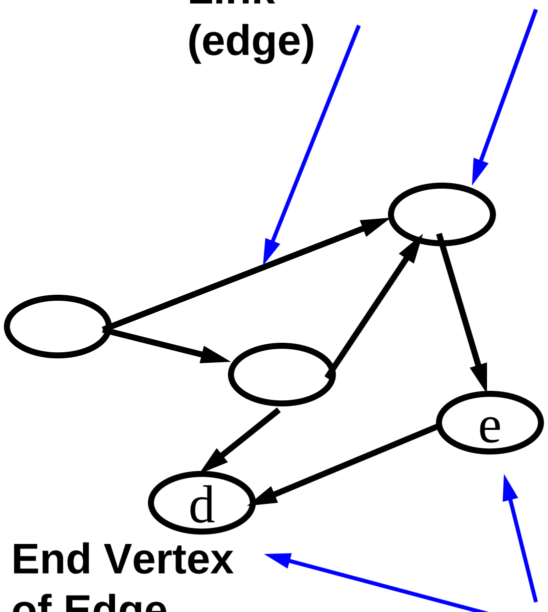
Outline

- Problem Formulation
  – Problem solving as state space search
- Mathematical Model
  – Graphs and search trees
- Reasoning Algorithms
  – Depth and breadth-first search

Problem Formulation: A Graph

Operator **State**

Link (edge) **Node (vertex)**

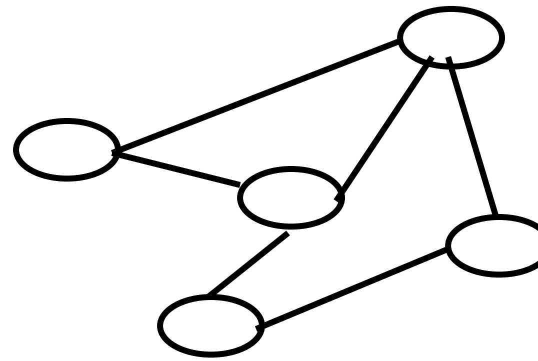


Start Vertex of Edge

End Vertex of Edge

neighbors (adjacent)

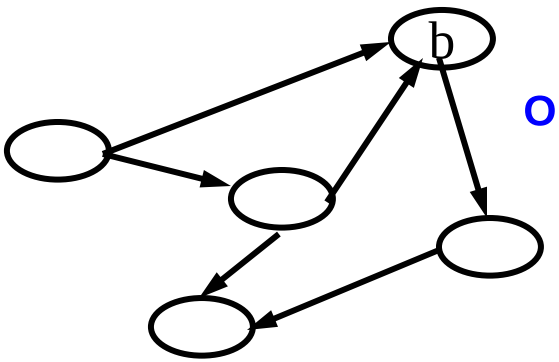
Incident to edge

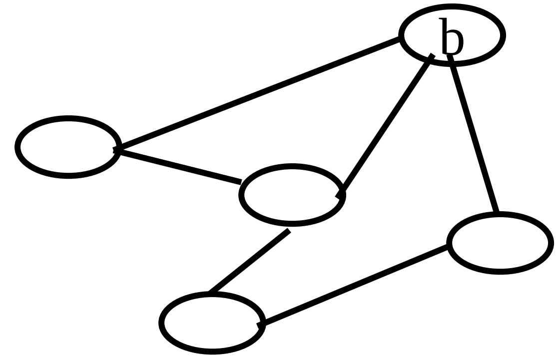


Directed Graph
(one-way streets)

Undirected Graph
(two-way streets)

Problem Formulation: A Graph





Directed Graph
(one-way streets)

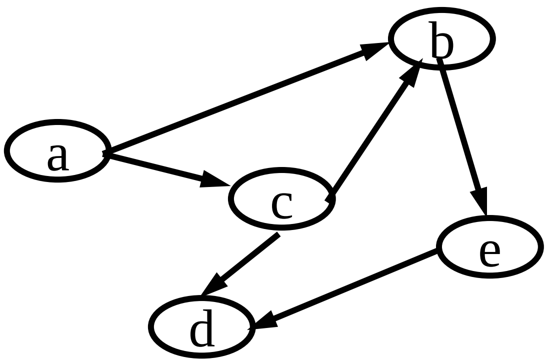
Undirected Graph
(two-way streets)

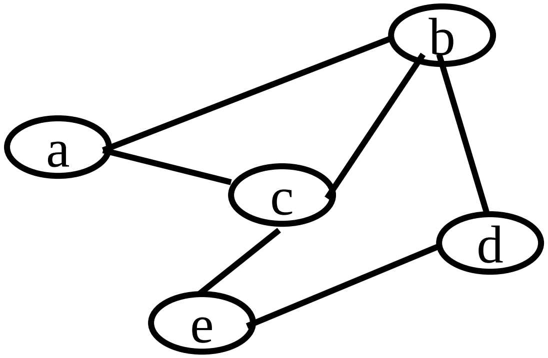
Problem Formulation: A Graph

Strongly connected graph
Directed path between all vertices.

Connected graph
Path between all vertices.

Complete graph
All vertices are adjacent.





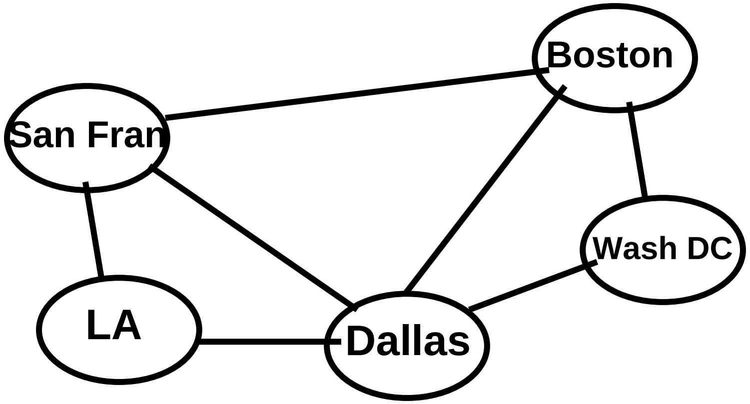
Sub graph
Subset of vertices
edges between vertices in Subset

Clique
A complete subgraph
All vertices are adjacent.

Directed Graph
(one-way streets)

Undirected Graph
(two-way streets)

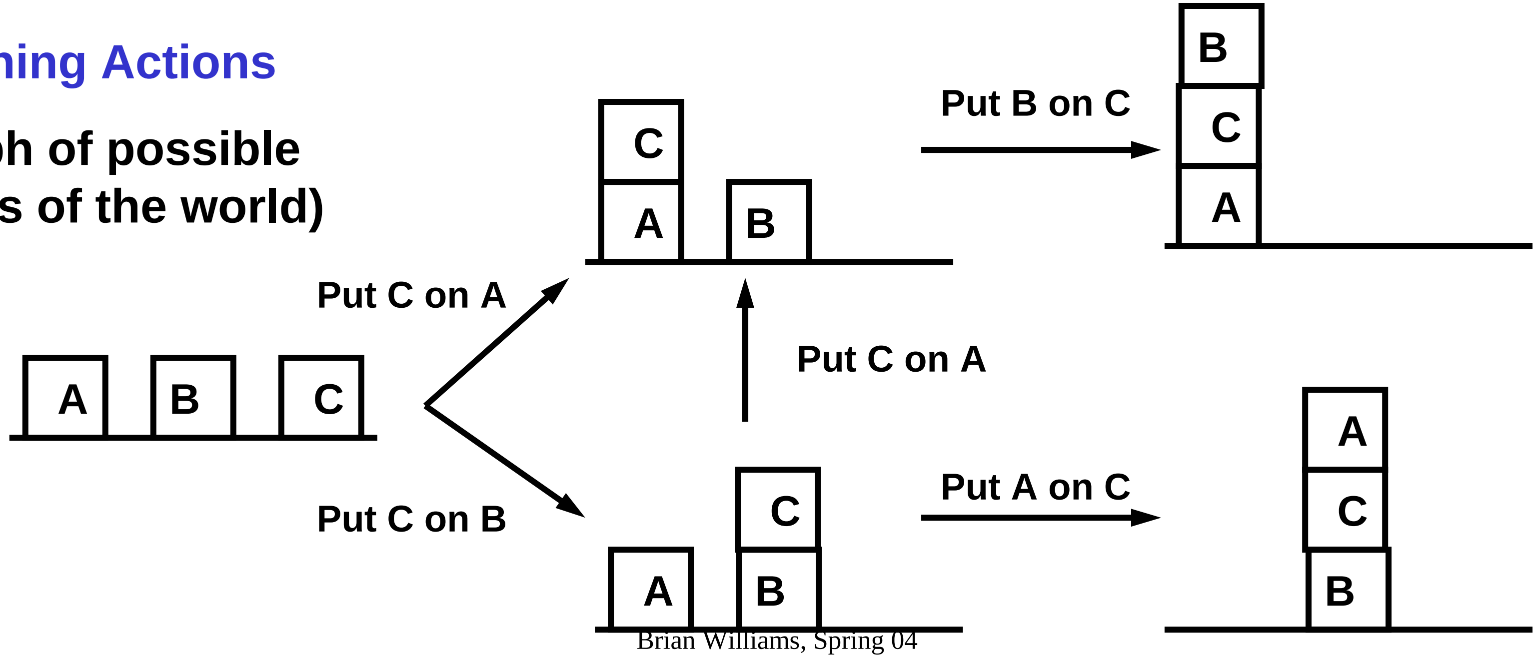
Examples of Graphs



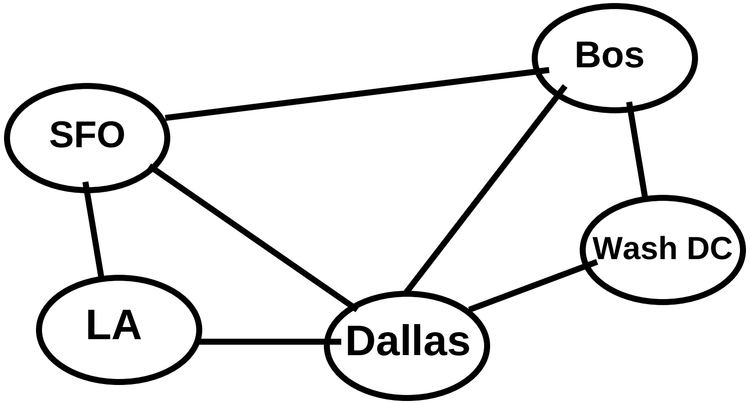
Roadmap

Planning Actions

(graph of possible states of the world)



A Graph



Airline Routes

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

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

{<SFO, Bos>,

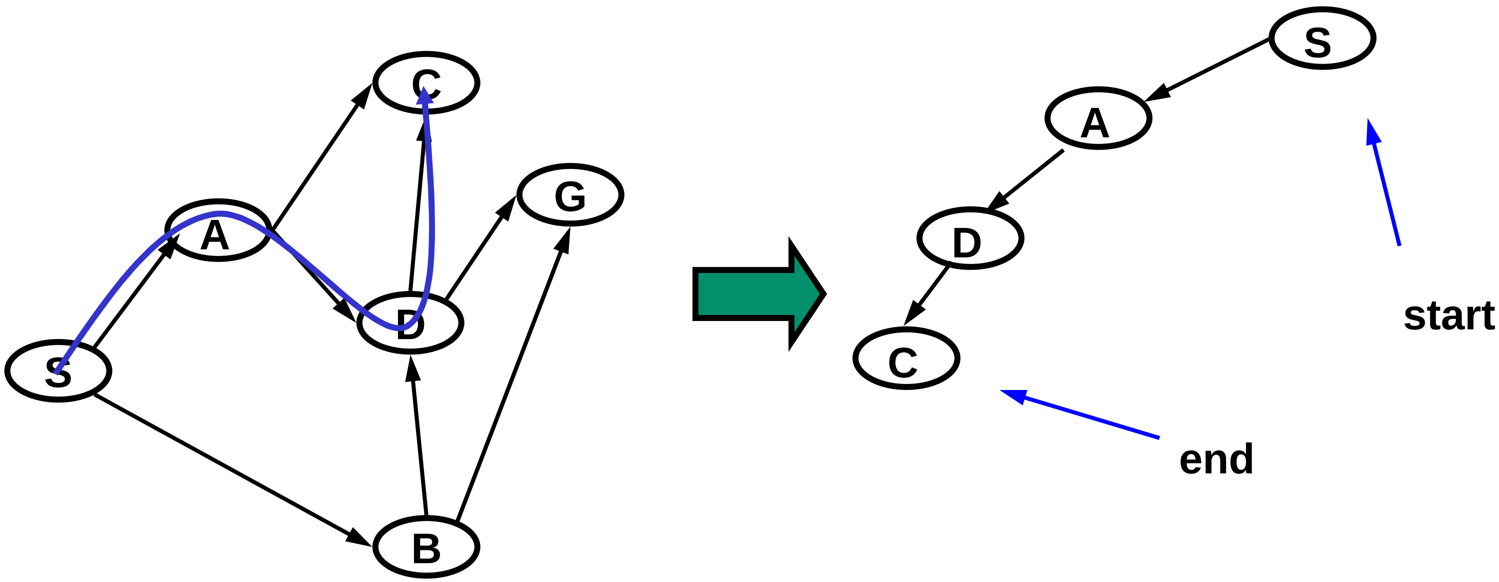
<SFO, LA>

<LA, Dallas>

<Dallas, Wash DC>

. . .} >

A Solution is a State Sequence: Problem Solving Searches Paths



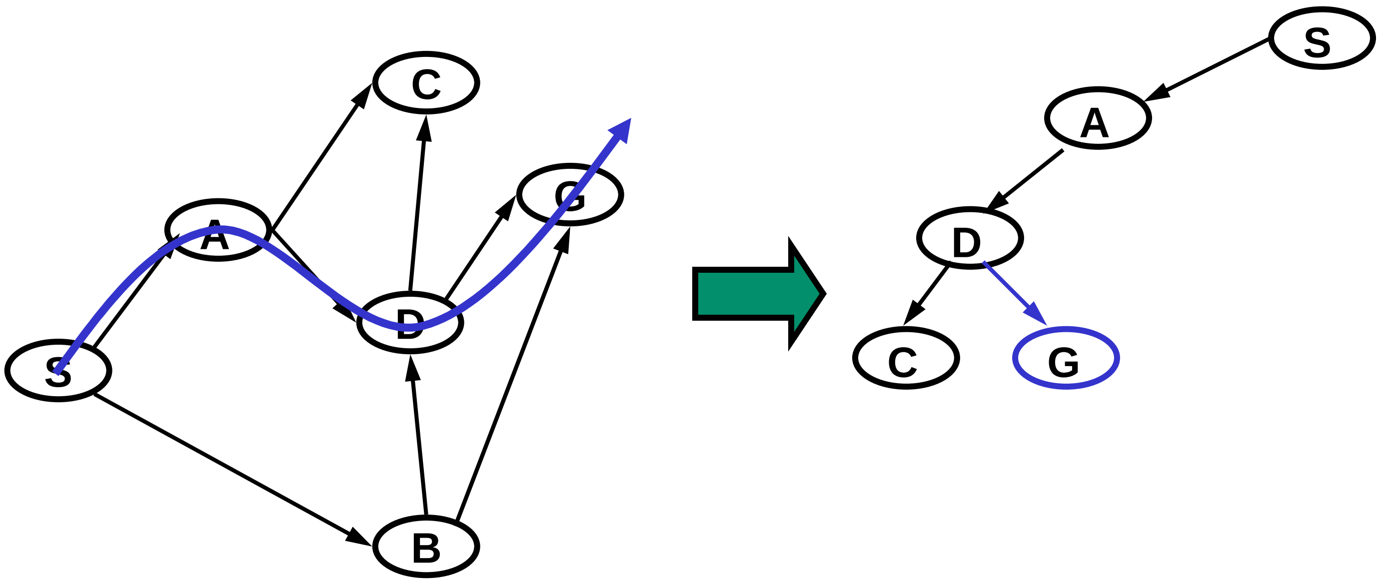
A path is a sequence of edges (or vertices)

<S, A, D, C>

Simple path has no repeated vertices.

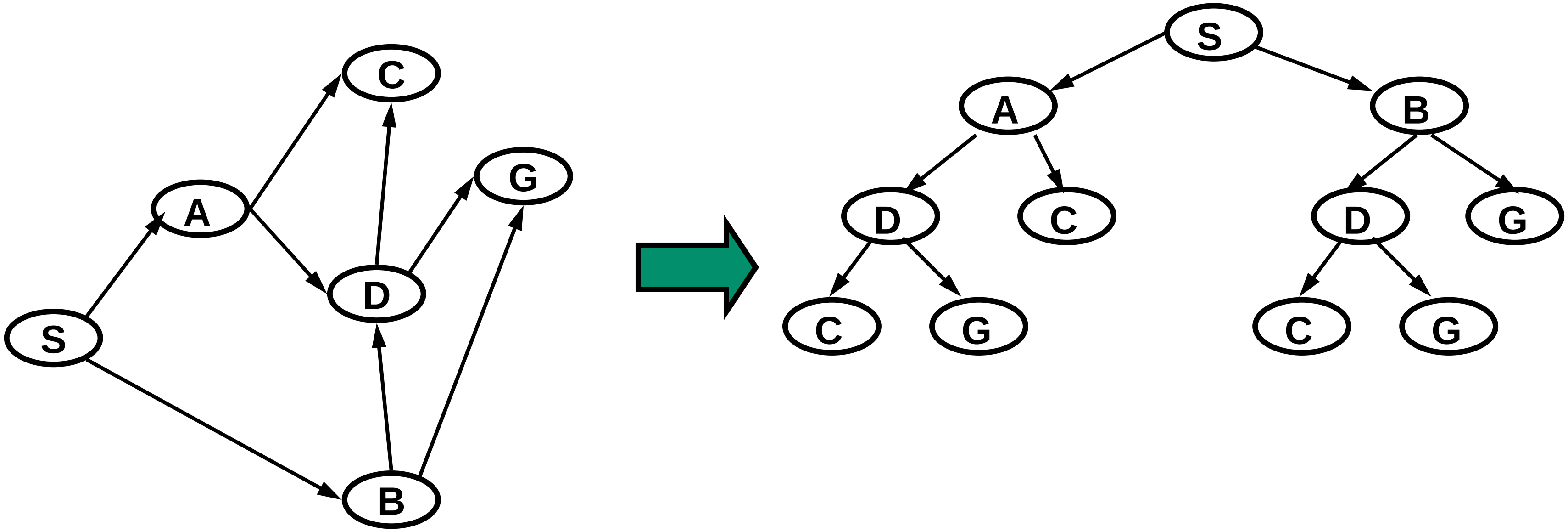
For a cycle, start = end.

A Solution is a State Sequence: Problem Solving Searches Paths



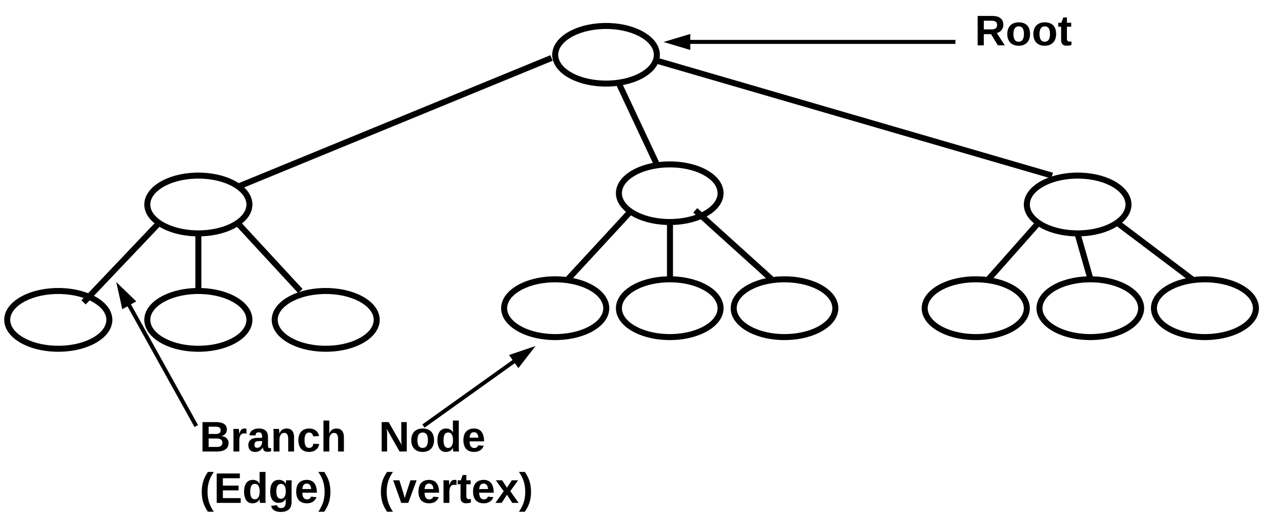
Represent searched paths using a tree.

A Solution is a State Sequence: Problem Solving Searches Paths

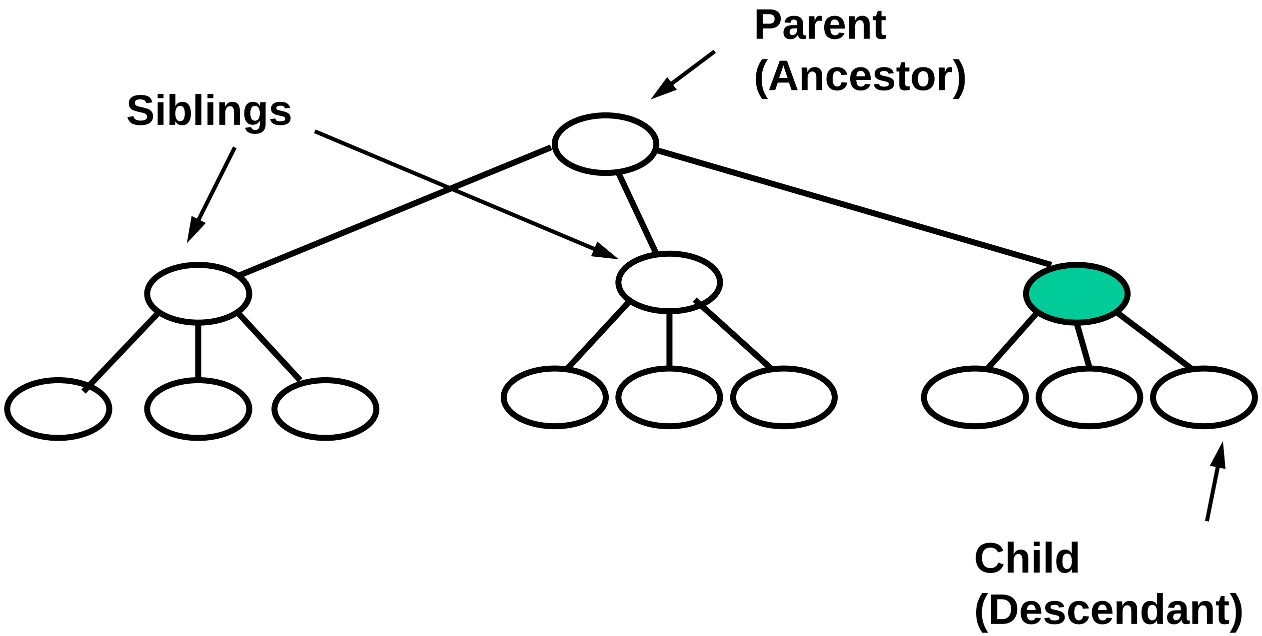


Represent searched paths using a tree.

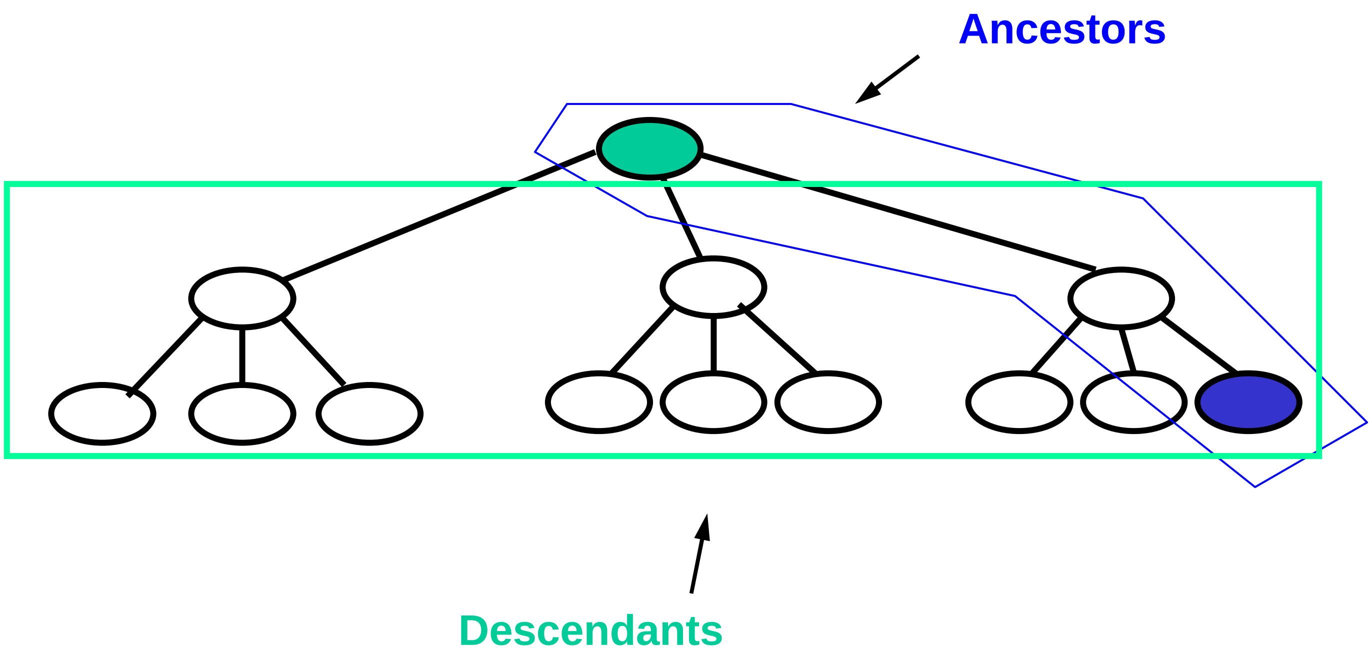
Search Trees



Search Trees



Search Trees



Outline

- Problem Formulation
  - Problem solving as state space search
- Mathematical Model
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

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

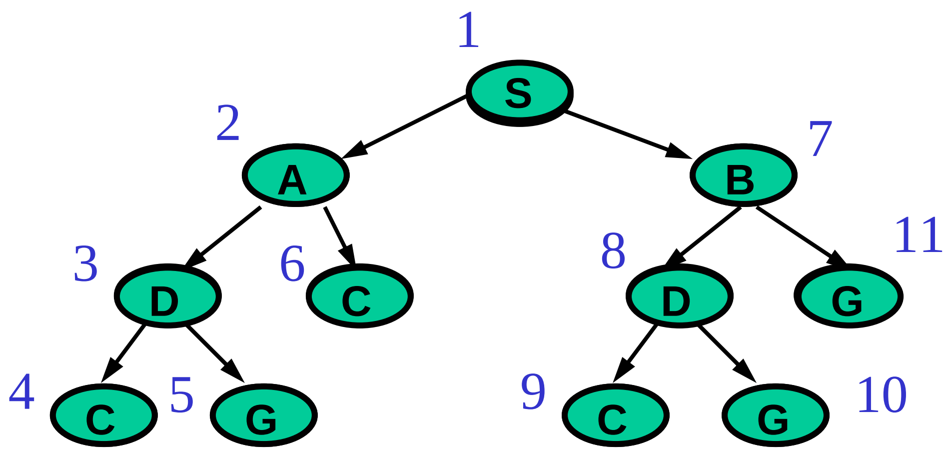
Classes of Search

Blind	Depth-First	Systematic exploration of whole tree
(uninformed)	Breadth-First	until the goal is found.
	Iterative-Deepening	

Depth First Search (DFS)

Idea: After visiting node

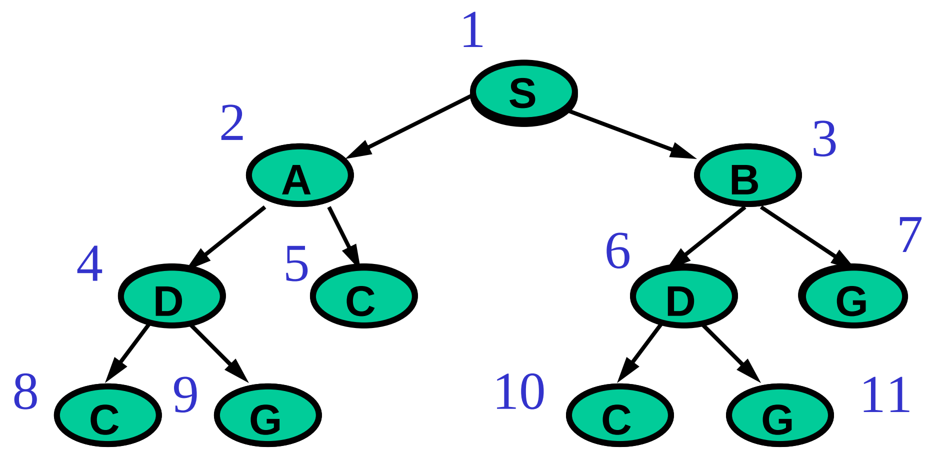
- Visit children, then siblings
- Visit siblings left to right



Breadth First Search (BFS)

Idea: After visiting node

- Visit siblings, then children
- Visit relatives left to right (top to bottom)



Elements of Algorithm Design

Description: (today)

- stylized pseudo code, sufficient to analyze and implement the algorithm (next Monday).

Analysis: (next Wednesday)

- Soundness:
  - when a solution is returned, is it guaranteed to be correct?
- Completeness:
  - is the algorithm guaranteed to find a solution when there is one?

- Time complexity:
  - how long does it take to find a solution?
- Space complexity:
  - how much memory does it need to perform search?

Outline

- Problem Formulation: State space search
- Model: Graphs and search trees
- Reasoning Algorithms: DFS and BFS
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

Simple Search Algorithm

How do we maintain the search state?

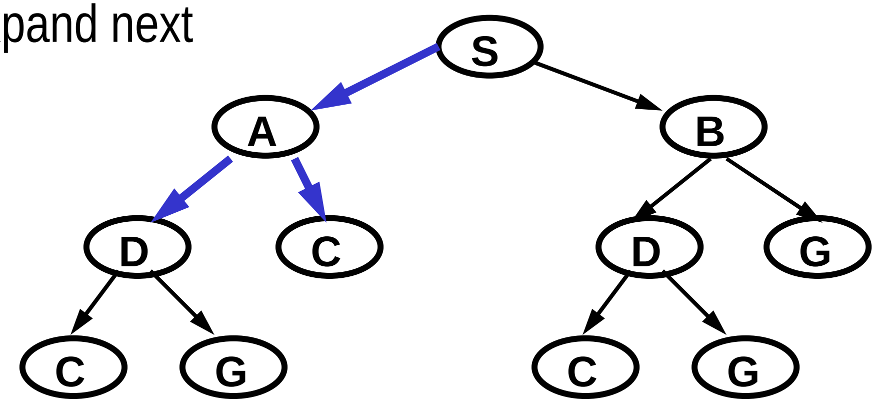
- A set of partial paths explored thus far.
- An ordering on which partial path to expand next
- called a queue Q.

How do we perform search?

- Repeatedly:
  - Select next partial path from Q.
  - Expand it.
  - Add expansions to Q.
- Terminate when goal found.

Going Meta:
Search as State Space Search

State Space

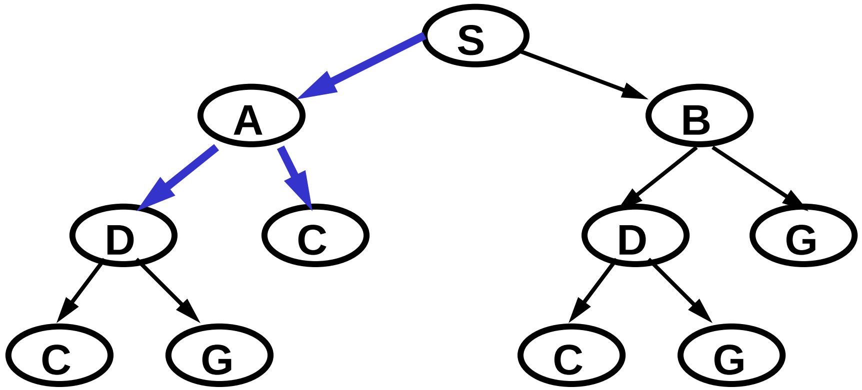


Operator

Goal-Test

Simple Search Algorithm

- S denotes the start node
- G denotes the goal node.
- A partial path is a path from S to some node D,
  - e.g., (D A S)



- The head of a partial path is the most recent node of the path,
  - e.g., D.
- The Q is a list of partial paths,
  - e.g. ((D A S) (C A S) …).

Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S)
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and create all one-step extensions of N to each child.
   c) Add all extended paths to Q
   d) Go to step 2.

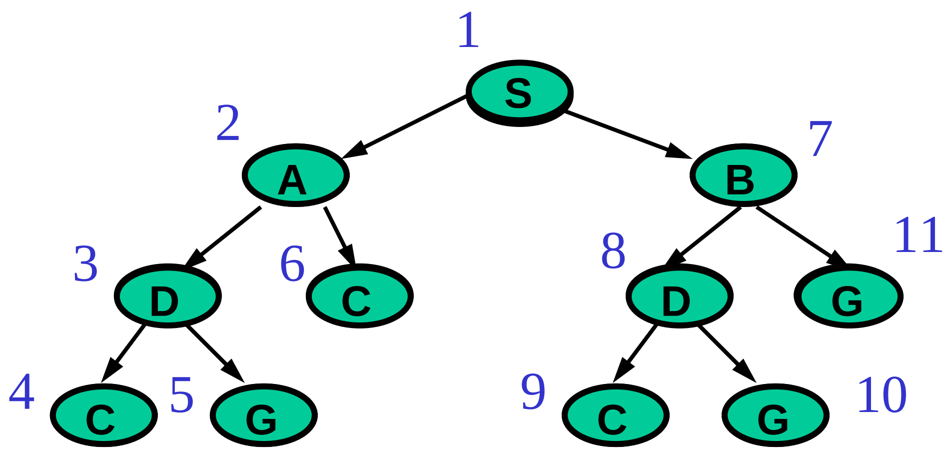
Outline

- Problem Formulation: State space search
- Model: Graphs and search trees
- Reasoning Algorithms: DFS and BFS
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

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?

Simple Search Algorithm

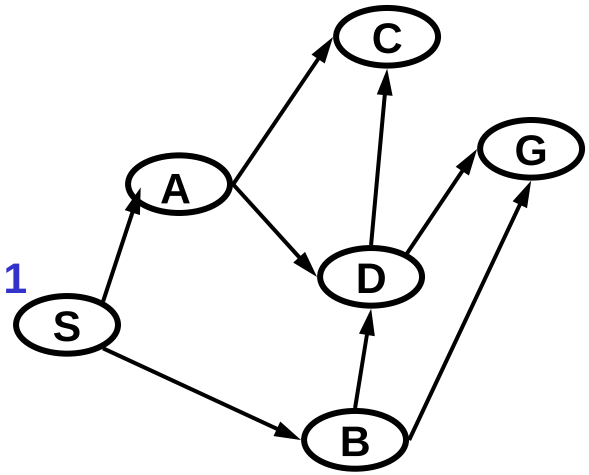
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S)
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and
   create all the one-step extensions of N to each child.
   c) Add all extended paths to Q
   d) Go to step 2.

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	(S)
2	
3	
4	
5	



Simple Search Algorithm

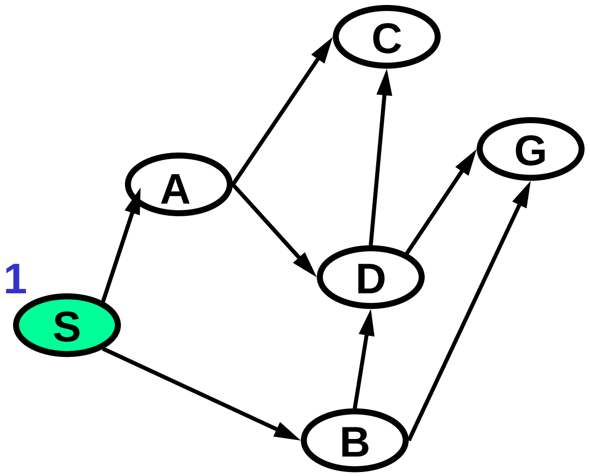
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S)
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and
      create all the one-step extensions of N to each child.
   c) Add all extended paths to Q
   d) Go to step 2.

Depth-First

Pick first element of Q; Add path extensions to front of Q

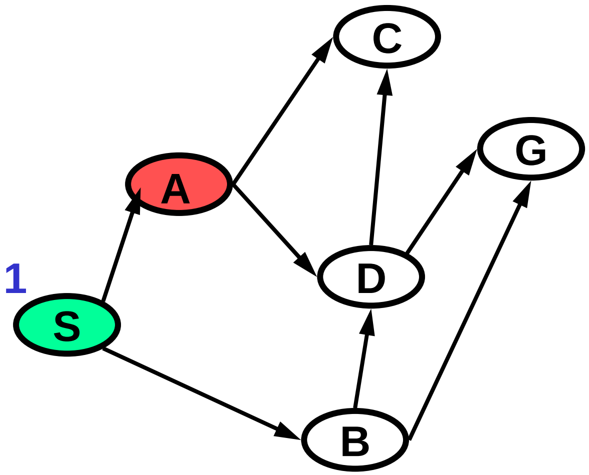
	Q
1	~~(S)~~
2	
3	
4	
5	



Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	(A S)
3	
4	
5	

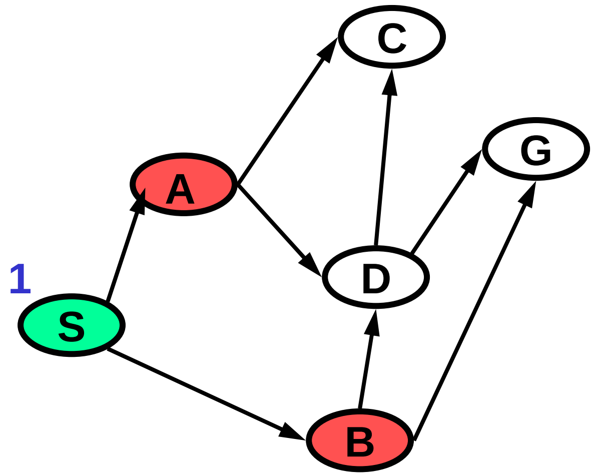


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	(A S) (B S)
3	
4	
5	



Added paths in blue

Simple Search Algorithm

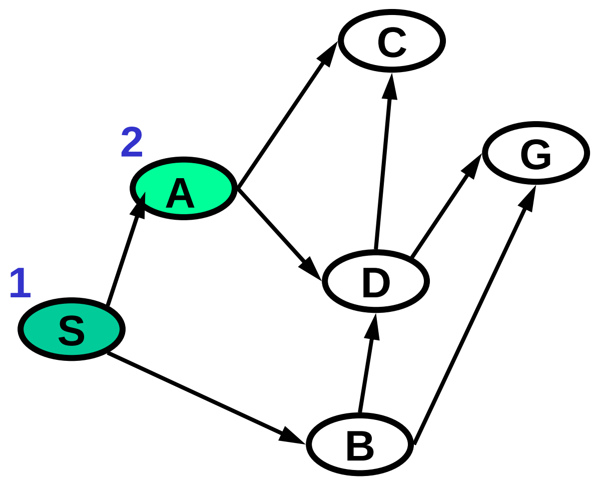
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S)
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and
      create all the one-step extensions of N to each child.
   c) Add all extended paths to Q
   d) Go to step 2.

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
~~1~~	~~(S)~~
~~2~~	~~(A S) (B S)~~
3	
4	
5	

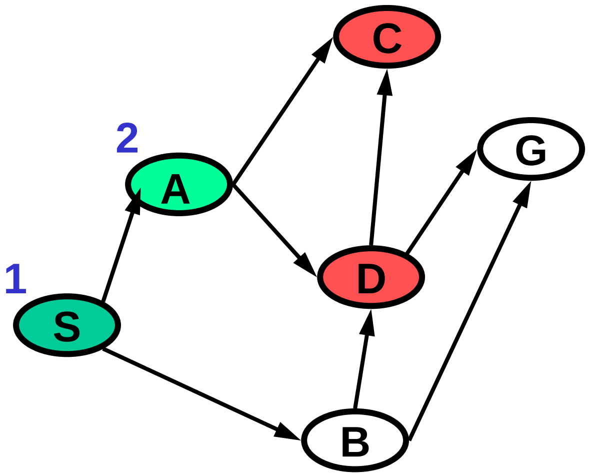


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	(C A S) (D A S) (B S)
4	
5	

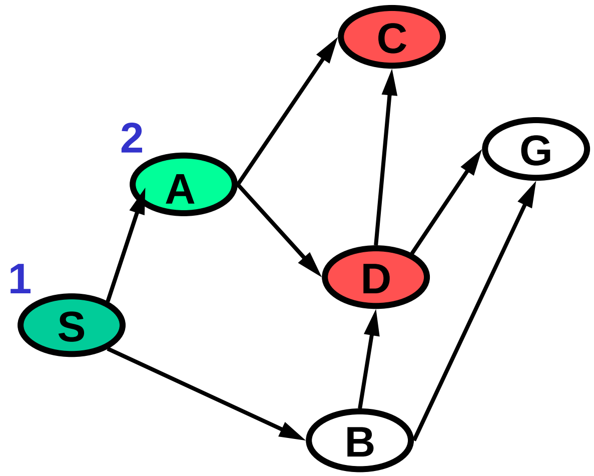


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	
5	

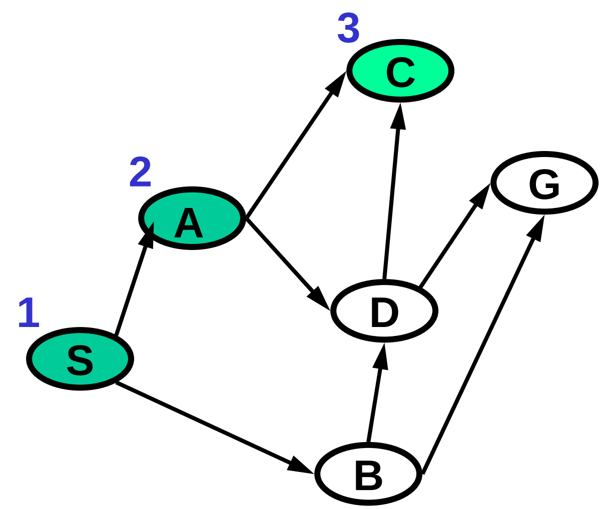


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	~~(A S) (B S)~~
3	~~(C A S)~~ (D A S) (B S)
4	
5	

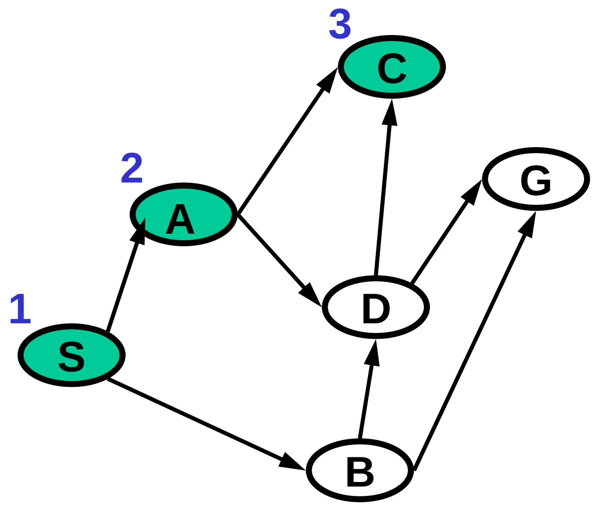


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	(D A S) (B S)
5	

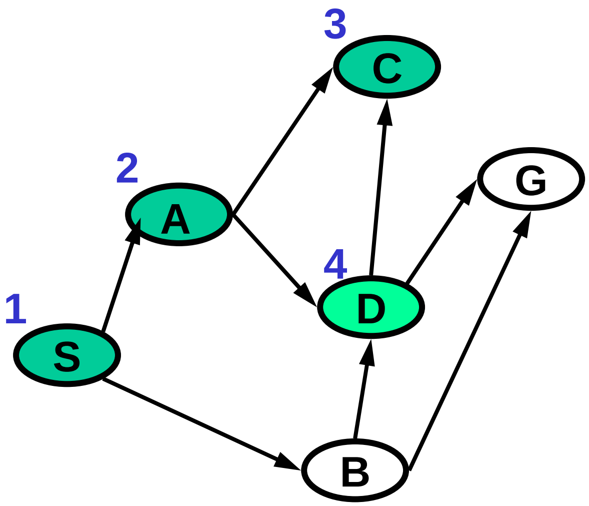


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	

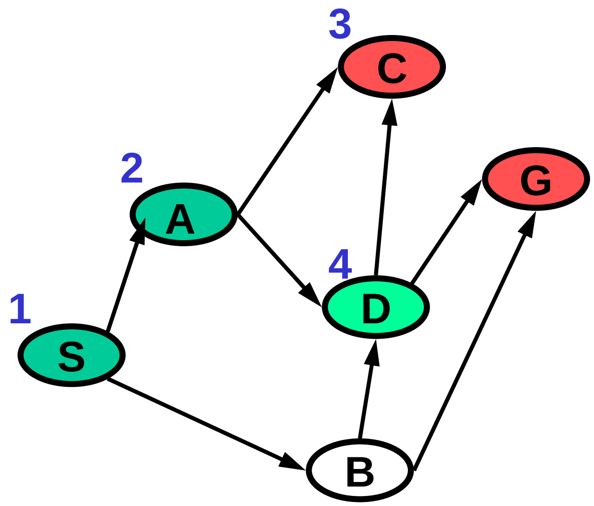


Added paths in blue

Depth-First

Pick first element of Q; Add path extensions to front of Q

	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	(C D A S)(G D A S) (B S)



Simple Search Algorithm

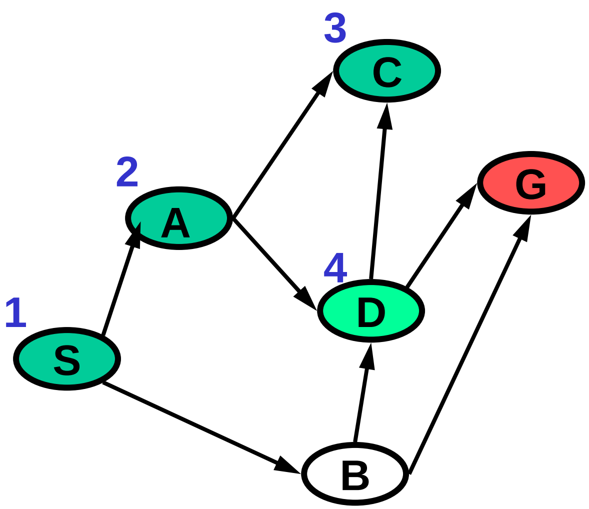
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S)
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and
      create all the one-step extensions of N to each child.
   c) Add all extended paths to Q
   d) Go to step 2.

Depth-First

Pick first element of Q; Add path extensions to front of Q

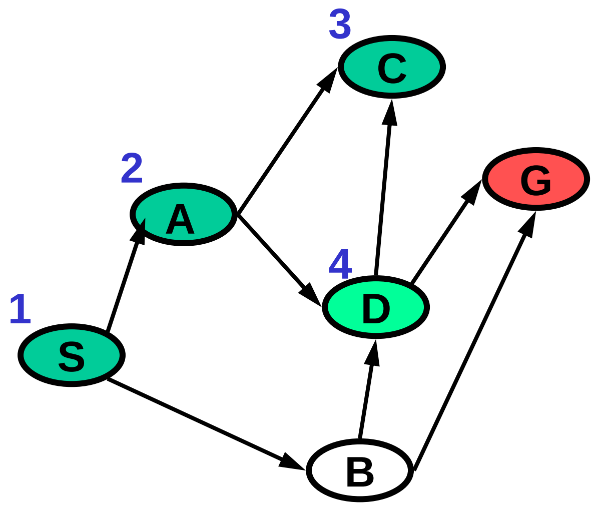
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	~~(C D A S)~~(G D A S) (B S)



Depth-First

Pick first element of Q; Add path extensions to front of Q

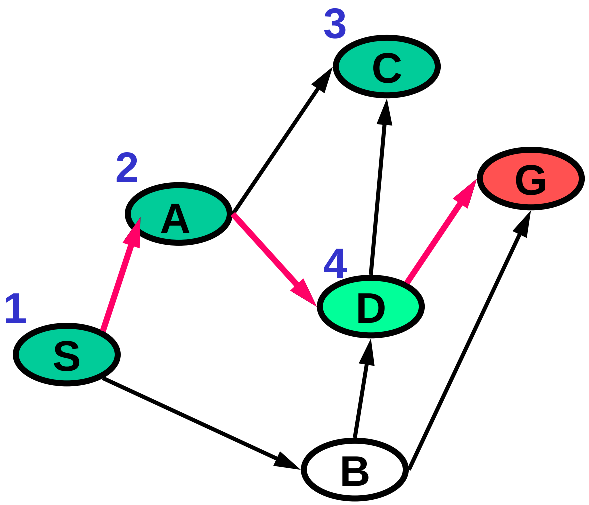
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	~~(C D A S)~~(G D A S) (B S)
6	(G D A S)(B S)



Depth-First

Pick first element of Q; Add path extensions to front of Q

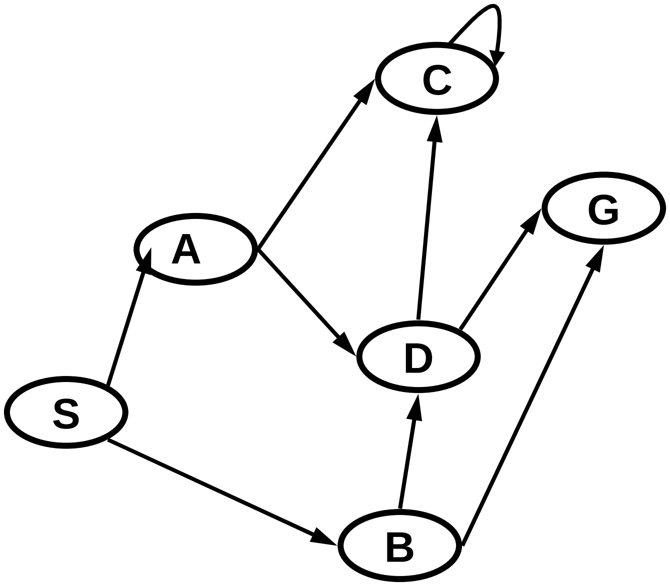
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	~~(C D A S)~~(G D A S) (B S)
6	$\boxed{\text{(G D A S)}}$(B S)



Outline

- Problem Formulation: State space search
- Model: Graphs and search trees
- Reasoning Algorithms: DFS and BFS
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

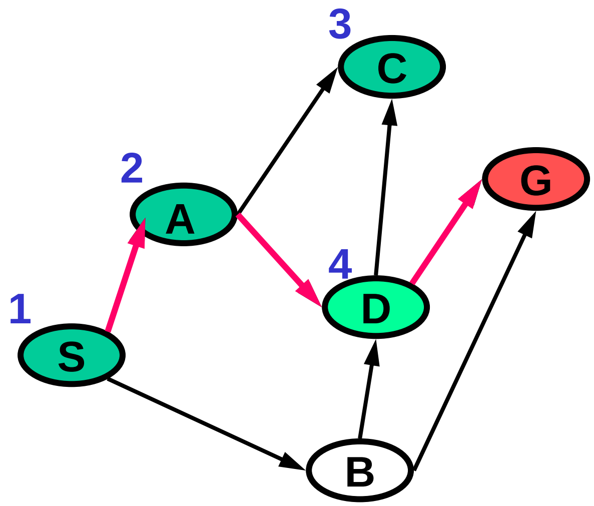
Issue: Starting at S and moving top to bottom, will depth-first search ever reach G?



Depth-First

Effort can be wasted in more mild cases

	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)
6	(G D A S)(B S)



- C visited multiple times
- Multiple paths to C, D & G

How much wasted effort can be incurred in the worst case?

How Do We Avoid Repeat Visits?

Idea:

- Keep track of nodes already visited.
- Do not place visited nodes on Q.

Does this maintain correctness?

- Any goal reachable from a node that was visited a second time would be reachable from that node the first time.

Does it always improve efficiency?

- Visits only a subset of the original paths, suc that each node appears at most once at the head of a path in Q.

How Do We Modify Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S) as only entry;
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else
   a) Remove N from Q
   b) Find all children of head(N) and create all the one-step extensions of N to each child.
   c) Add to Q all the extended paths;
   d) Go to step 2.

Simple Search Algorithm

Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. Initialize Q with partial path (S) as only entry; set Visited = ( )
2. If Q is empty, fail. Else, pick some partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else
   a) Remove N from Q
   b) Find all children of head(N) not in Visited and create all the one-step extensions of N to each child.
   c) Add to Q all the extended paths;
   d) Add children of head(N) to Visited

   e) Go to step 2.

Testing for the Goal

- **This algorithm stops (in step 3) when head(N) = G.**
- **We could have performed this test in step 6 as each extended path is added to Q. This would catch termination earlier and be perfectly correct for all the searches we have covered so far.**
- **However, performing the test in step 6 will be incorrect for the optimal searches we look at later. We have chosen to leave the test in step 3 to maintain uniformity with these future searches.**

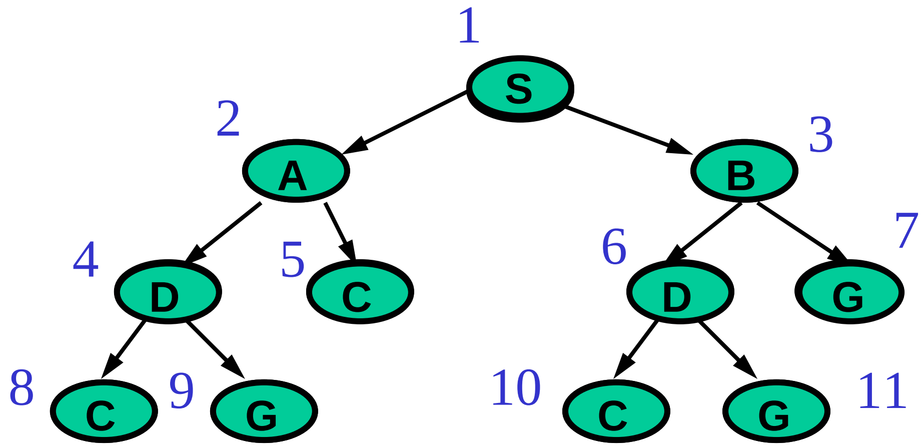
Outline

- Problem Formulation: State space search
- Model: Graphs and search trees
- Reasoning Algorithms: DFS and BFS
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

Breadth First Search (BFS)

Idea:

- Visit siblings before their children
- Visit relatives left to right

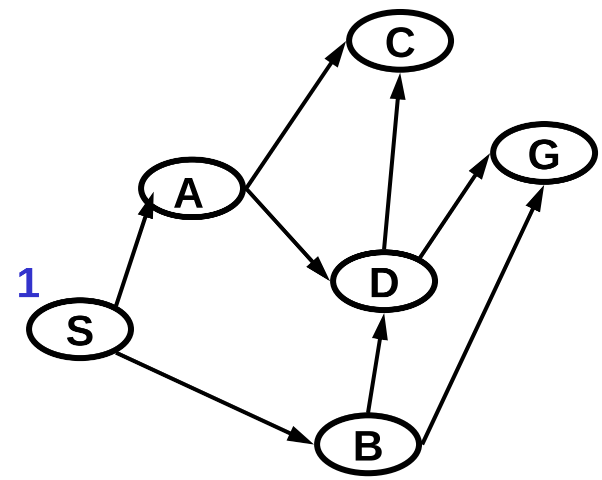


Assuming that we pick the first element of Q,
Then where do we add path extensions to the Q?

Breadth-First

Pick first element of Q; Add path extensions to end of Q

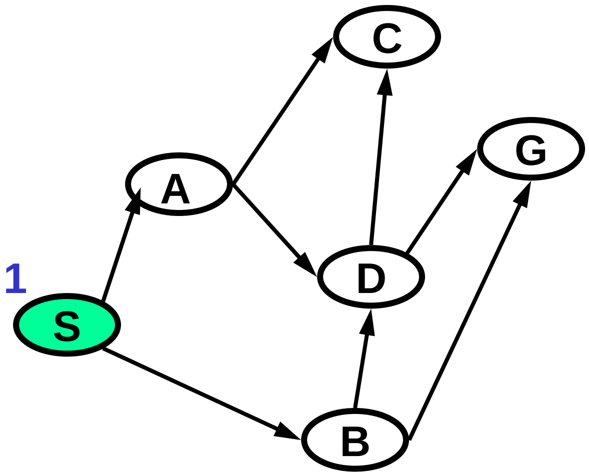
	Q	Visited
1	(S)	S
2		
3		
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

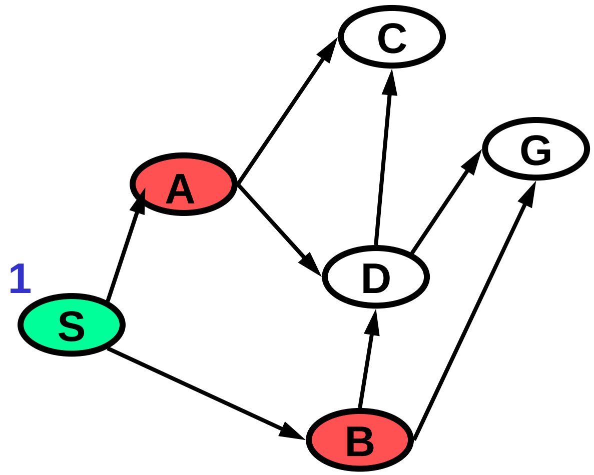
	Q	Visited
1	~~(S)~~	S
2		
3		
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

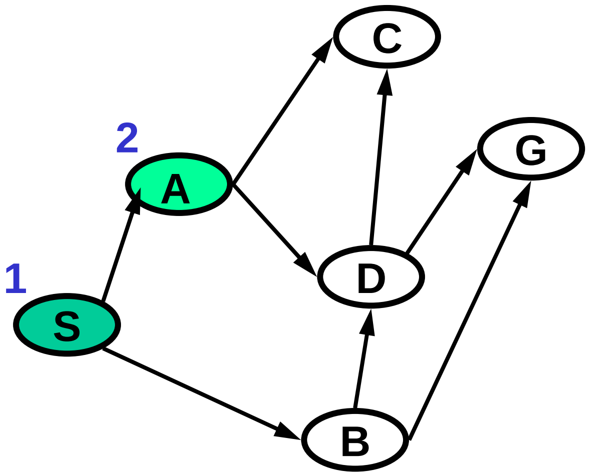
	Q	Visited
1	~~(S)~~	S
2	(A S) (B S)	A,B,S
3		
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

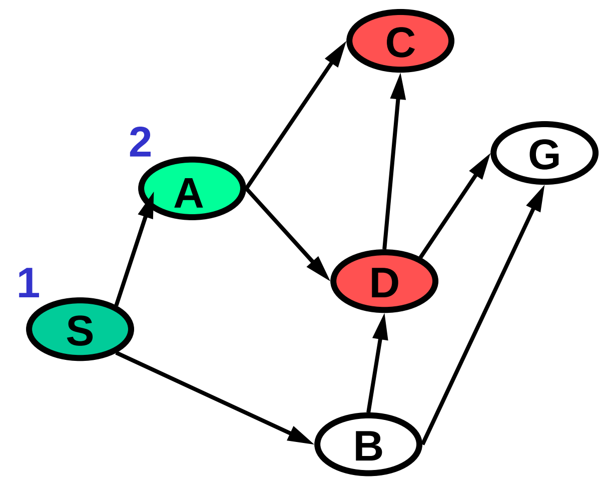
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3		
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

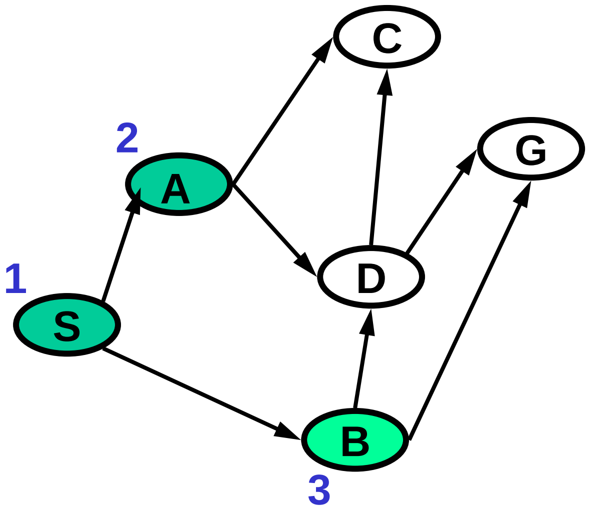
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

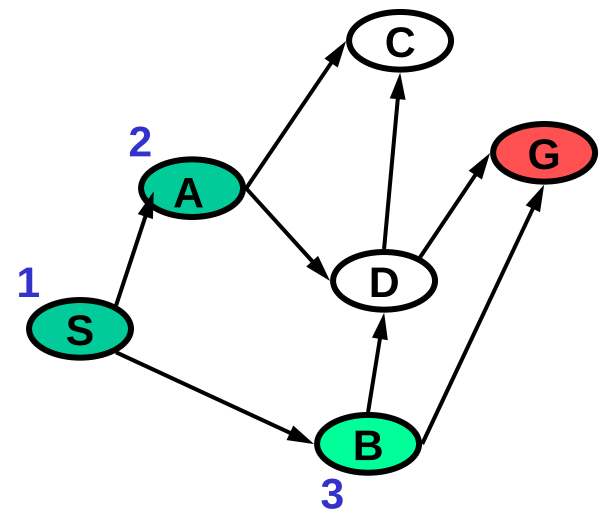
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	~~(B S)~~ (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	~~(B S)~~ (C A S) (D A S)	C,D,B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S
5		
6		

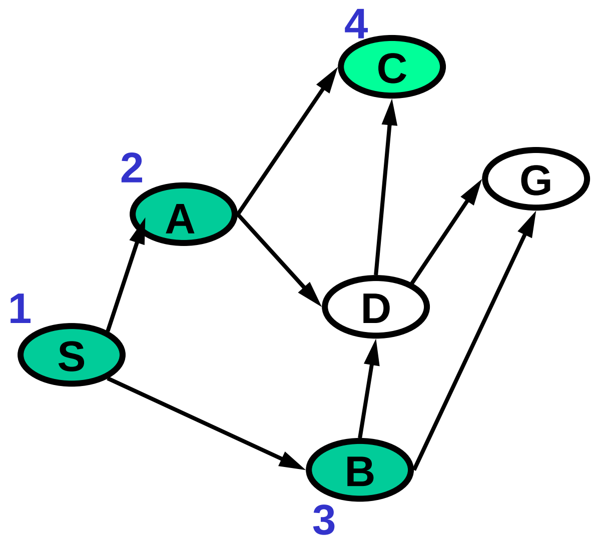


* We could stop here, when the first path to the goal is generated.

Breadth-First

Pick first element of Q; Add path extensions to end of Q

	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	~~(C A S)~~ (D A S) (G B S)*	G,C,D,B,A,S
5		
6		

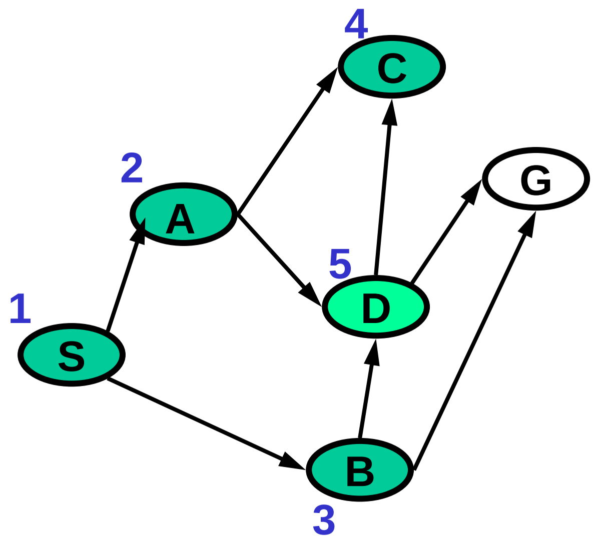


* We could stop here, when the first path to the goal is generated.

Breadth-First

Pick first element of Q; Add path extensions to end of Q

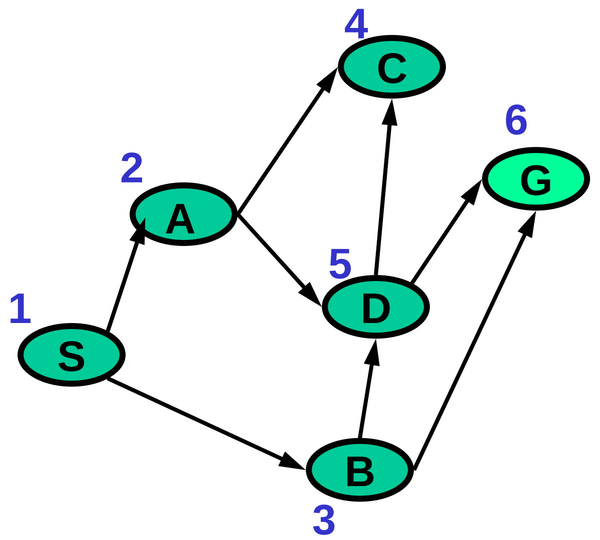
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	~~(C A S)~~ (D A S) (G B S)*	G,C,D,B,A,S
5	~~(D A S)~~ (G B S)	G,C,D,B,A,S
6		



Breadth-First

Pick first element of Q; Add path extensions to end of Q

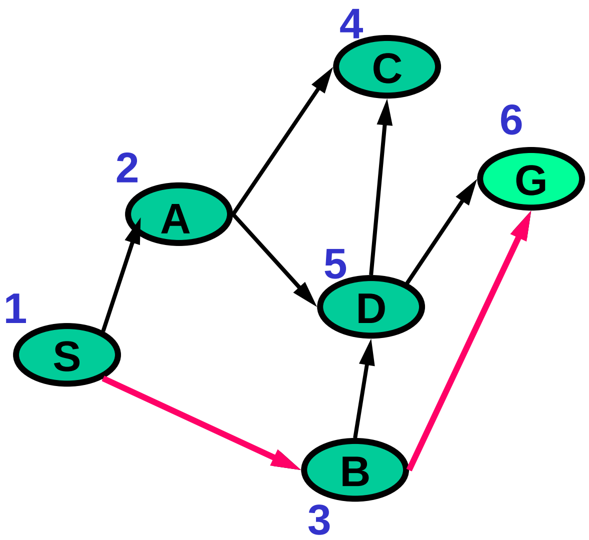
	Q	**Visited**
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	~~(C A S)~~ (D A S) (G B S)*	G,C,D,B,A,S
5	~~(D A S)~~ (G B S)	G,C,D,B,A,S
6	~~(G B S)~~	G,C,D,B,A,S



Breadth-First

Pick first element of Q; Add path extensions to end of Q

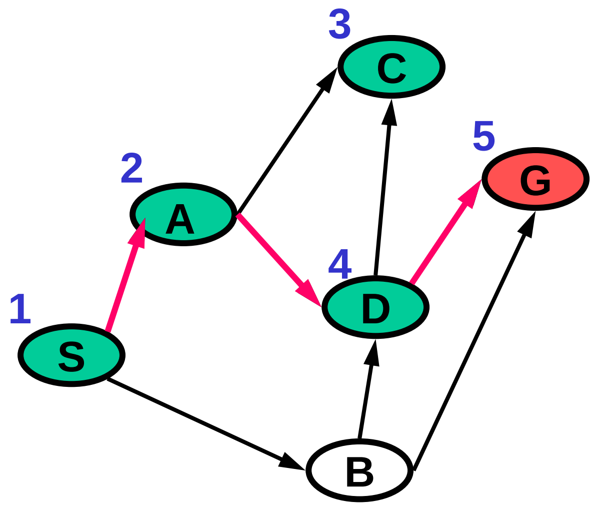
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S
5	(D A S) (G B S)	G,C,D,B,A,S
6	(G B S)	G,C,D,B,A,S



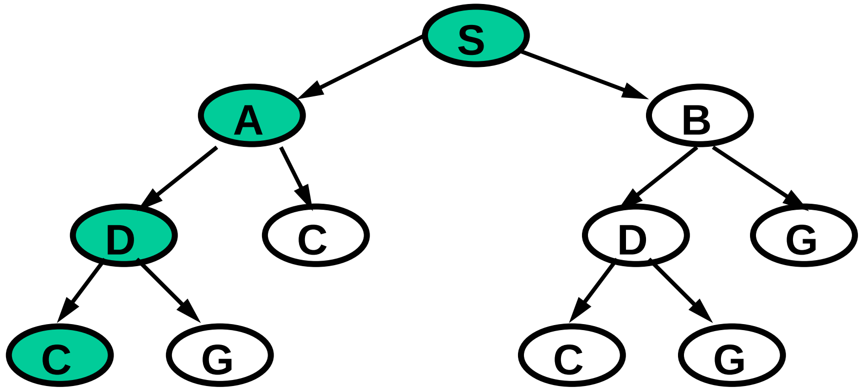
Depth-first with visited list

Pick first element of Q; Add path extensions to front of Q

	Q	Visited
1	(S)	S
2	(A S) (B S)	A, B, S
3	(C A S) (D A S) (B S)	C,D,B,A,S
4	(D A S) (B S)	C,D,B,A,S
5	$\boxed{\text{(G D A S)}}$ (B S)	G,C,D,B,A,S



Depth First Search (DFS)

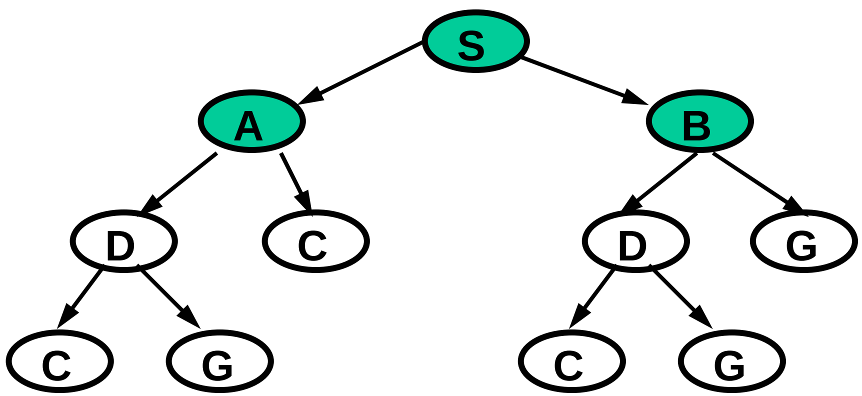


Depth-first:

Add path extensions to front of Q

Pick first element of Q

Breadth First Search (BFS)



Breadth-first:

Add path extensions to back of Q

Pick first element of Q

For each search type, where do we place the children on the queue?

What You Should Know

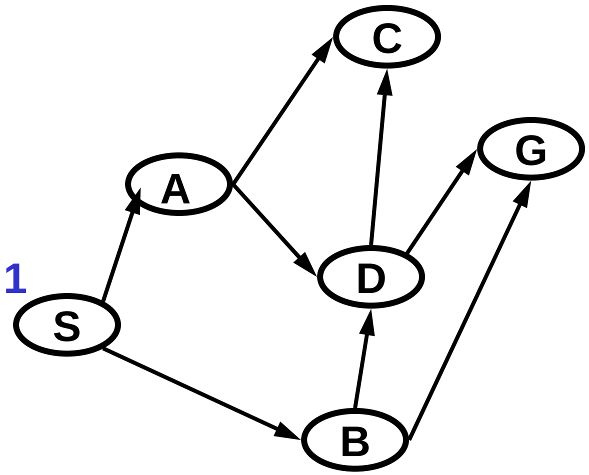
- Most problem solving tasks may be formulated as state space search.
- Mathematical representations for search are graphs and search trees.
- Depth-first and breadth-first search may be framed, among others, as instances of a generic search strategy.
- Cycle detection is required to achieve efficiency and completeness.

Appendix

Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

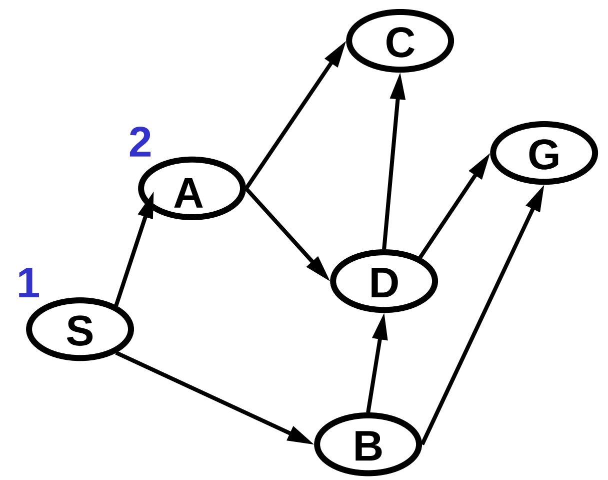
	Q
1	(S)
2	
3	
4	
5	
6	
7	



Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	
4	
5	
6	
7	

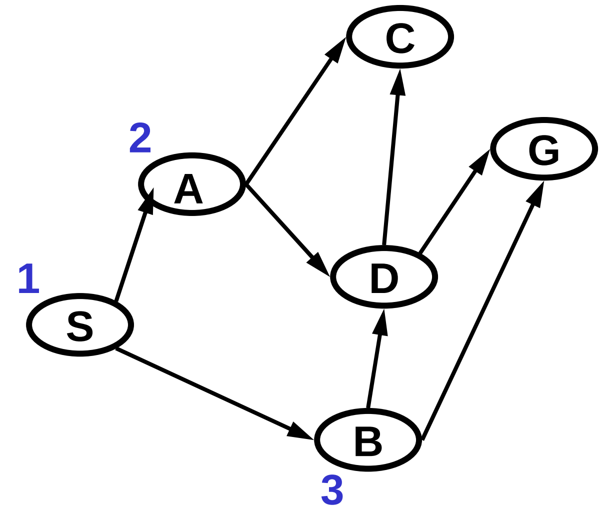


Added paths in blue

Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	
5	
6	
7	

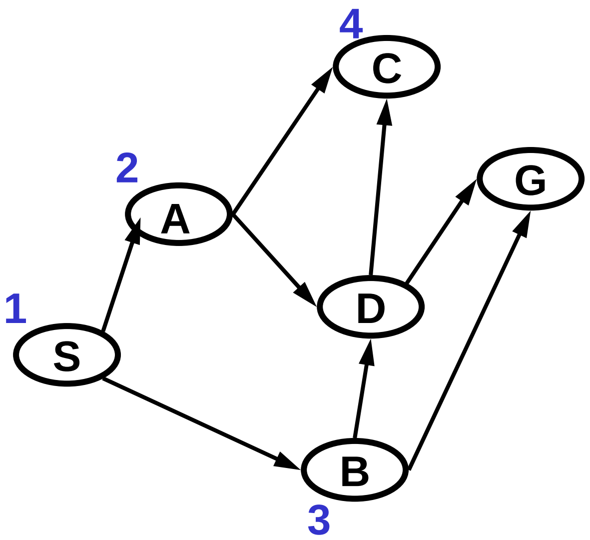


Added paths in blue

Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	
6	
7	



Added paths in blue

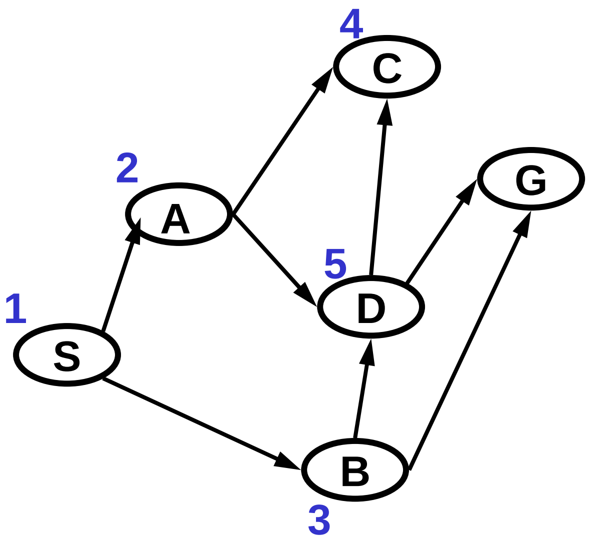
Revisited nodes in pink

* We could have stopped here, when the first path to the goal was generated.

Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

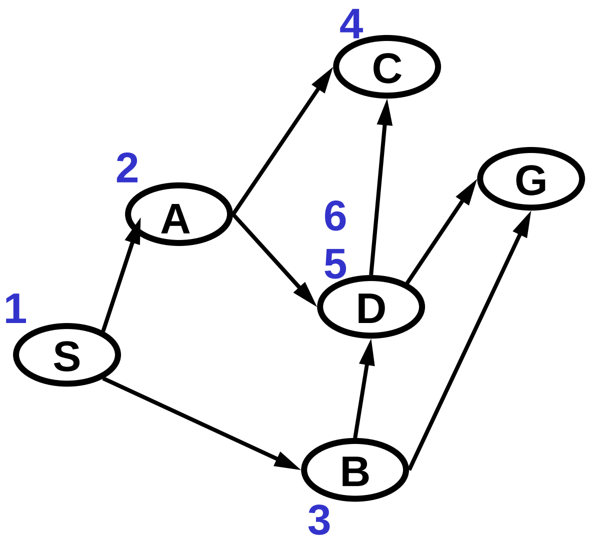
	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	(D A S) (D B S) (G B S)
6	
7	



Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	(D A S) (D B S) (G B S)
6	(D B S) (G B S) (C D A S) (G D A S)
7	



Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	(D A S) (D B S) (G B S)
6	(D B S) (G B S) (C D A S) (G D A S)
7	(G B S) (C D A S) (G D A S)