Problem Solving as State Space Search

Brian C. Williams

16.070

April 15th, 2003

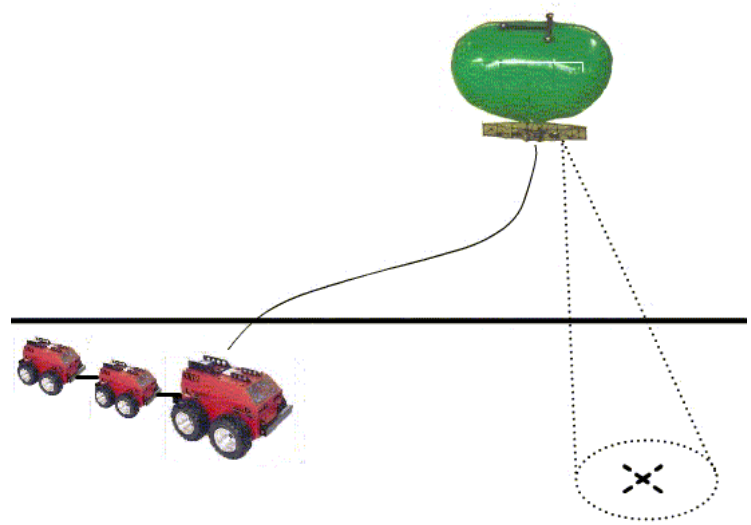
Slides adapted from:
6.034 Tomas Lozano Perez,
Russell and Norvig AIMA


Self-Diagnosing Explorers
courtesy of JPL

In Space The Exception is the Rule

- Quintuple fault occurs (three shorts, tank-line and pressure jacket burst, panel flies off).
- Power limitations too severe to perform new mission..
- Novel reconfiguration identified, exploiting LEM batteries for power.
- Swaggert & Lovell work on Apollo 13 emergency rig lithium hydroxide unit.

APOLLO 13



Complex missions must carefully:

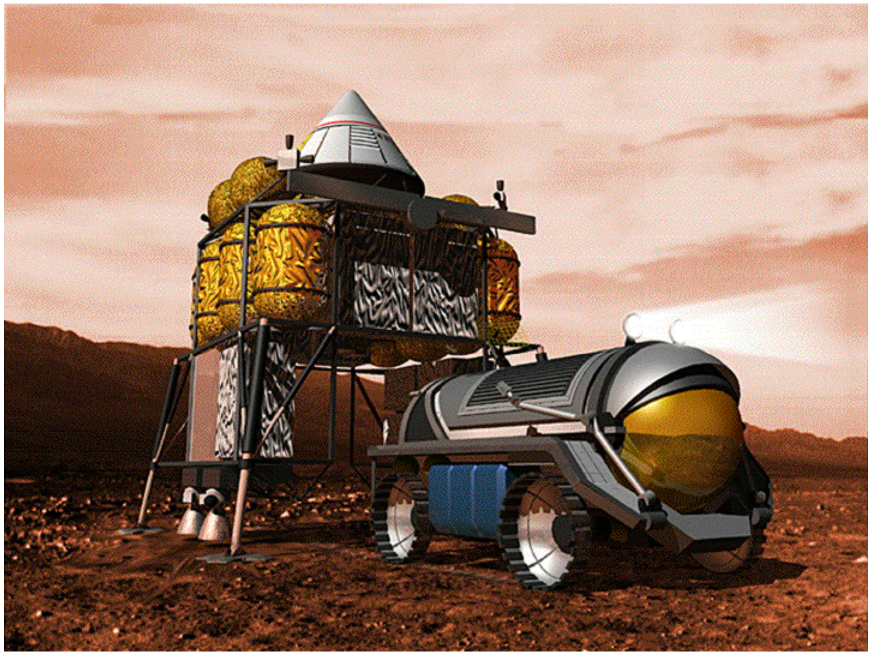
- Plan complex sequences of actions
- Schedule tight resources
- Monitor and diagnose behavior
- Repair or reconfigure hardware.

⇨ Most Autonomy problems, search through a space of options.

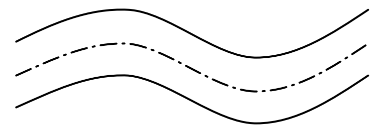
⇨ We formulate as state space search.

Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search



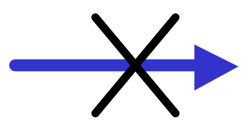
Astronaut
Goose
Grain
Fox

Rover 

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

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

Early AI: What are the universal problem solving methods?

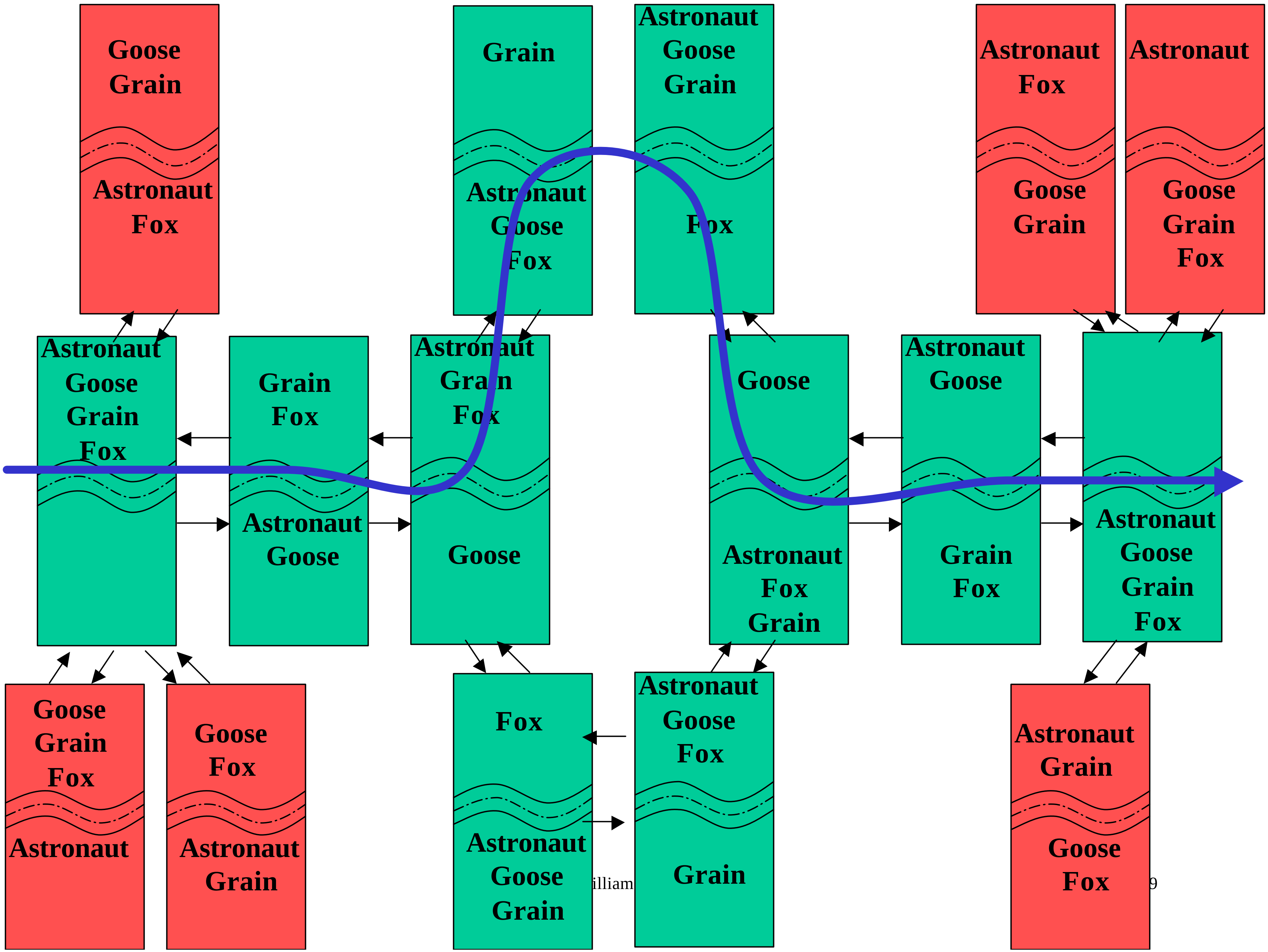
Simple  Trivial

Problem Solving as State Space Search

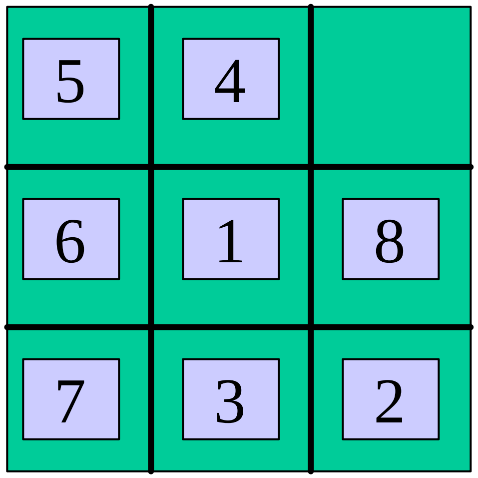
- Formulate Goal
- Formulate Problem
  - States
  - Operators
- Generate Solution
  - Sequence of states

Problem Solving as State Space Search

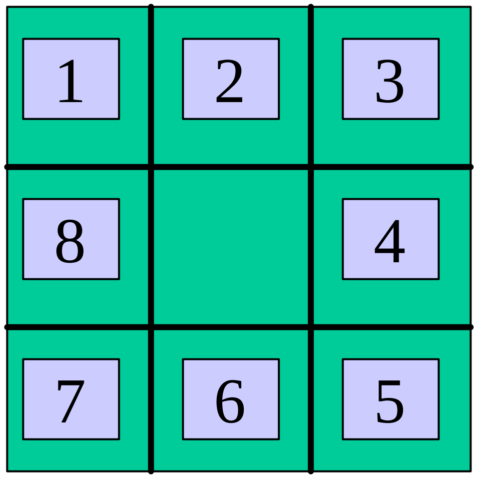
- Formulate Goal
  - Astronaut, Fox, Goose & Grain across river
- Formulate Problem
  - States
    - Location of Astronaut, Fox, Goose & Grain at top or bottom river bank
  - Operators
    - Move rover with astronaut & 1 or 0 items to other bank.


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

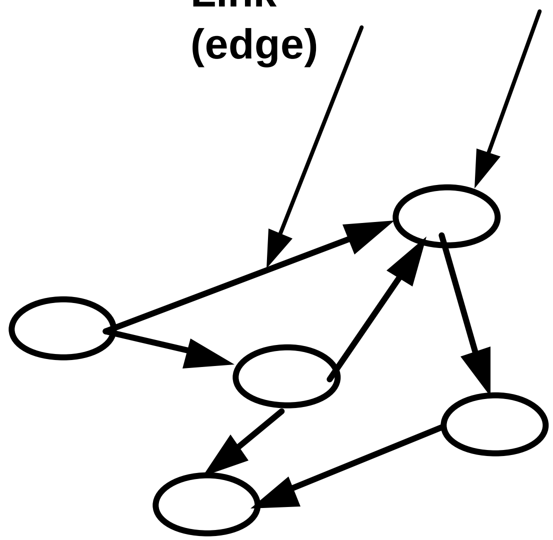
Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search

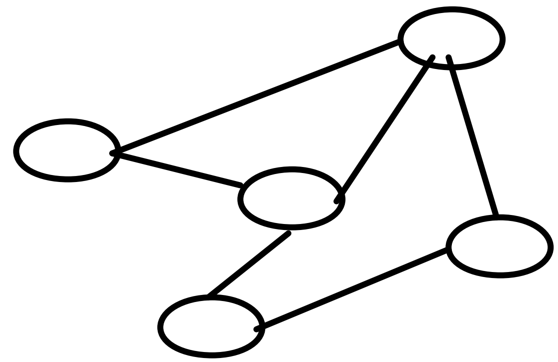
Problem Formulation: A Graph

Operator **State**

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

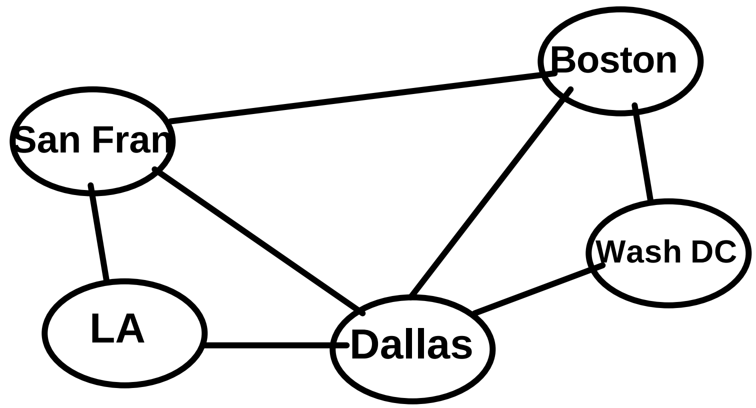


Directed Graph
(one-way streets)



Undirected Graph
(two-way streets)

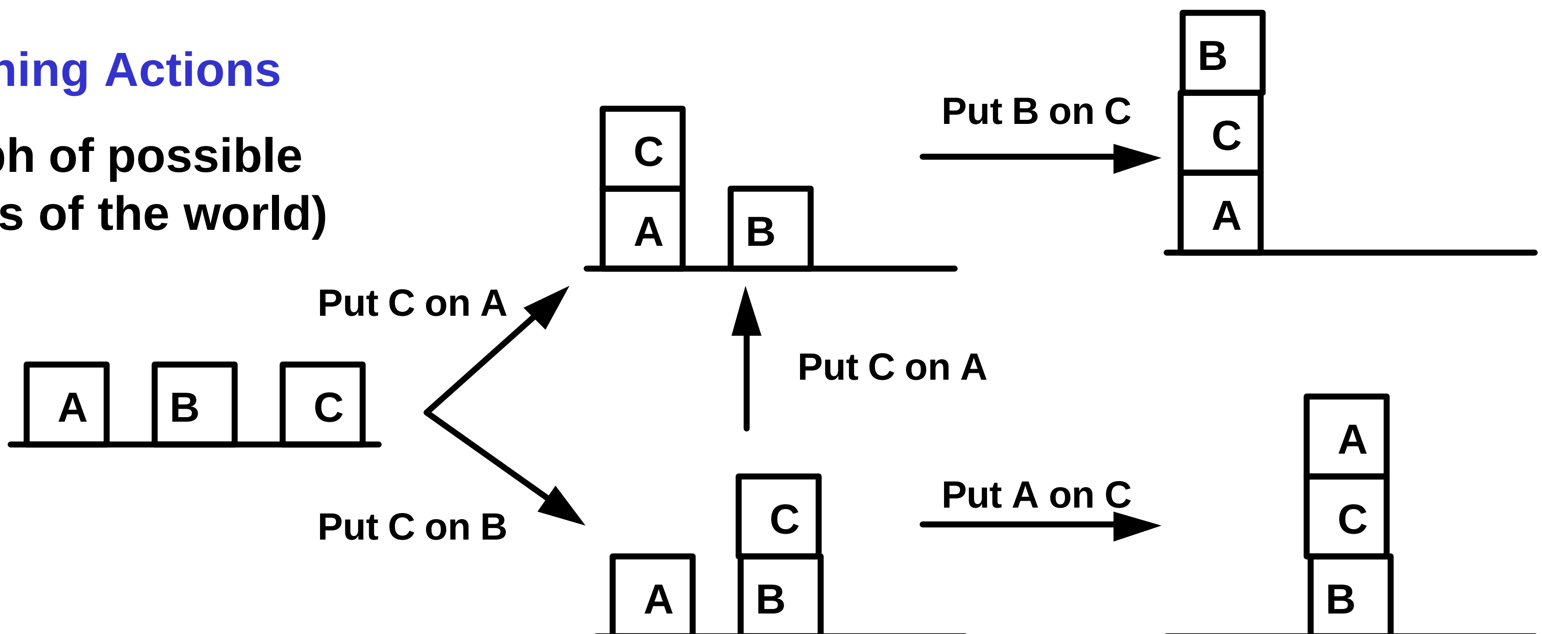
Examples of Graphs



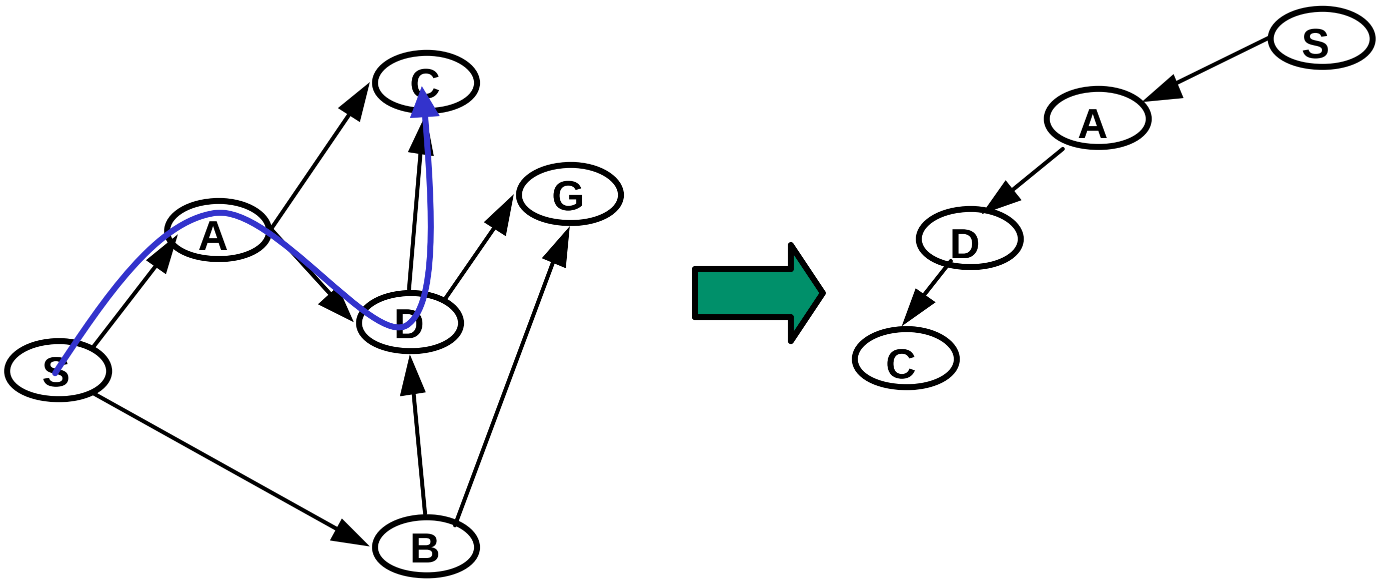
Airline Routes

Planning Actions

(graph of possible states of the world)

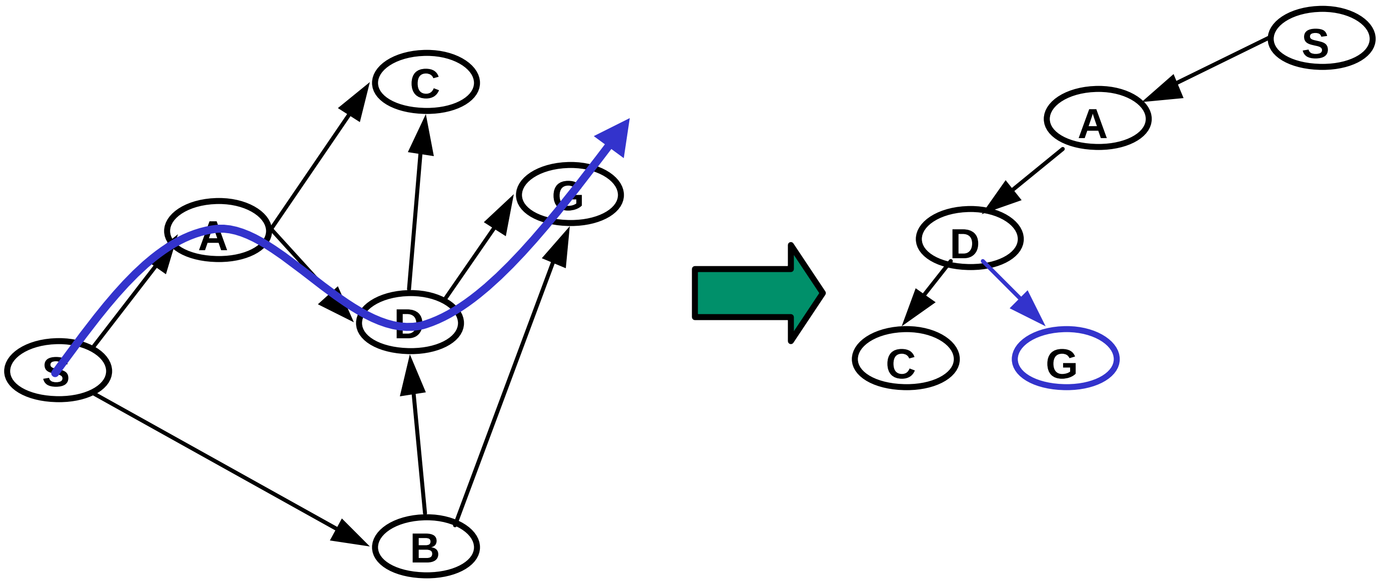


A Solution is a State Sequence: Problem Solving Searches Paths



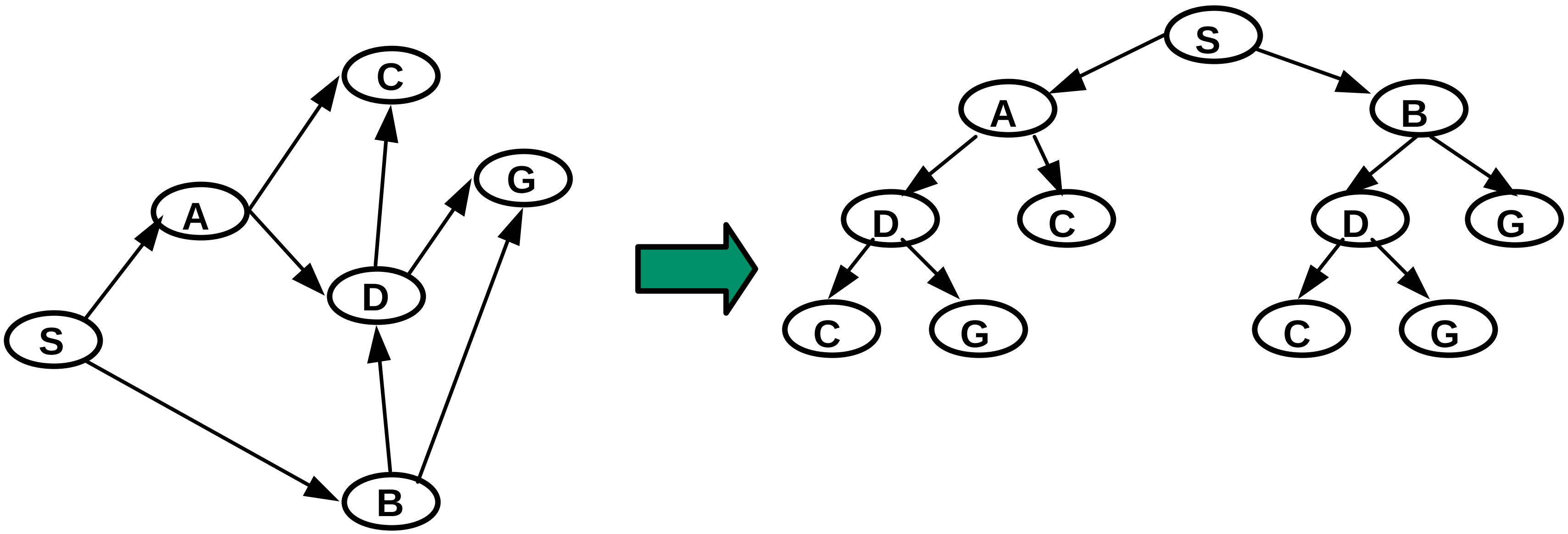
Represent searched paths using a tree.

A Solution is a State Sequence: Problem Solving Searches Paths



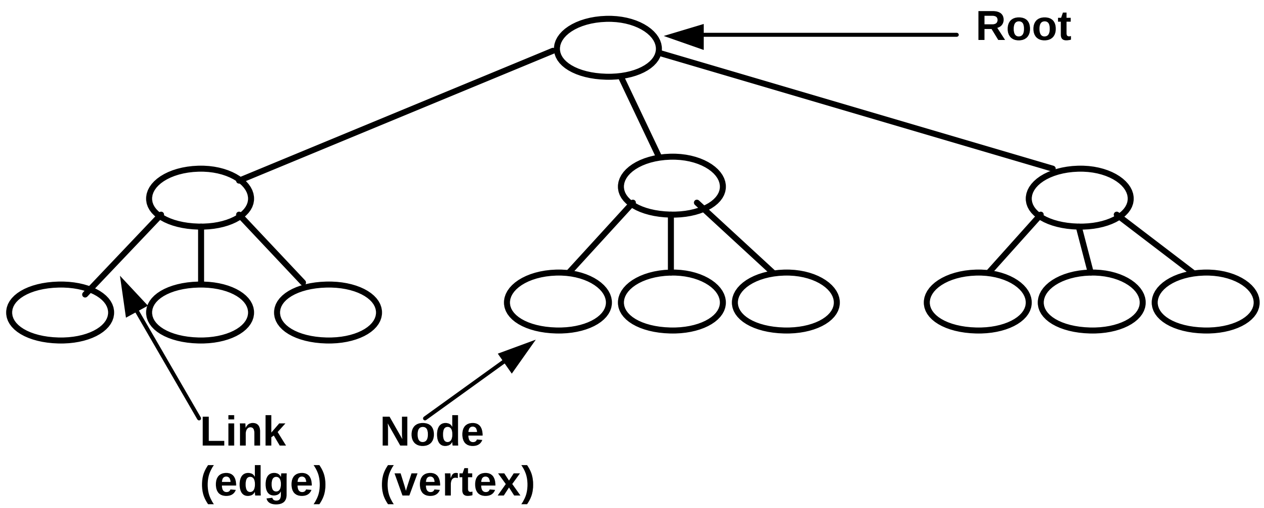
Represent searched paths using a tree.

A Solution is a State Sequence: Problem Solving Searches Paths

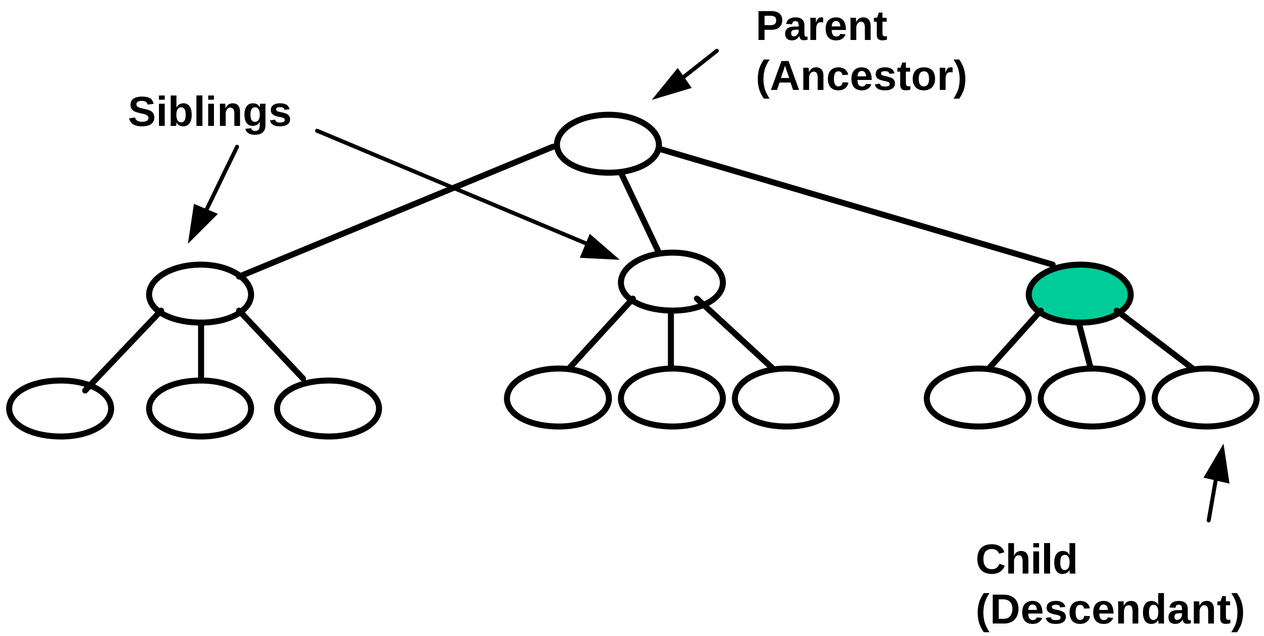


Represent searched paths using a tree.

Search Trees



Search Trees



Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search
  - Depth first in lecture
  - Breadth first at home

Classes of Search

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

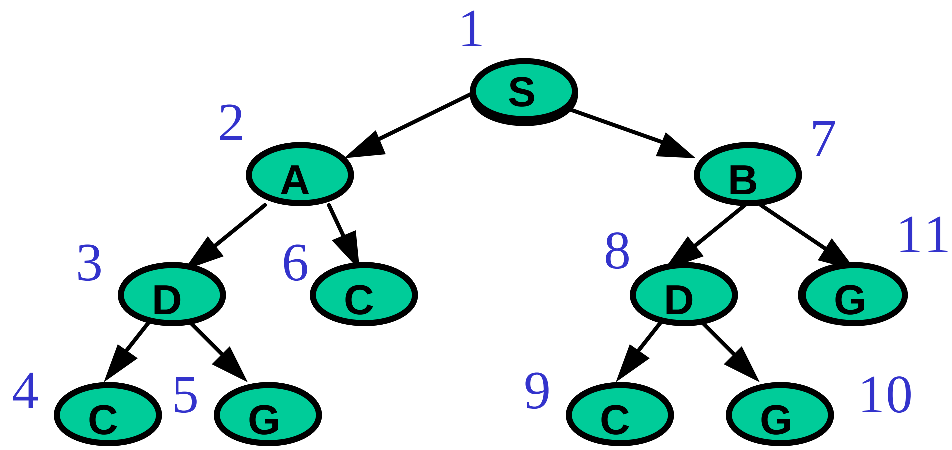
Classes of Search

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

Depth First Search (DFS)

Idea:

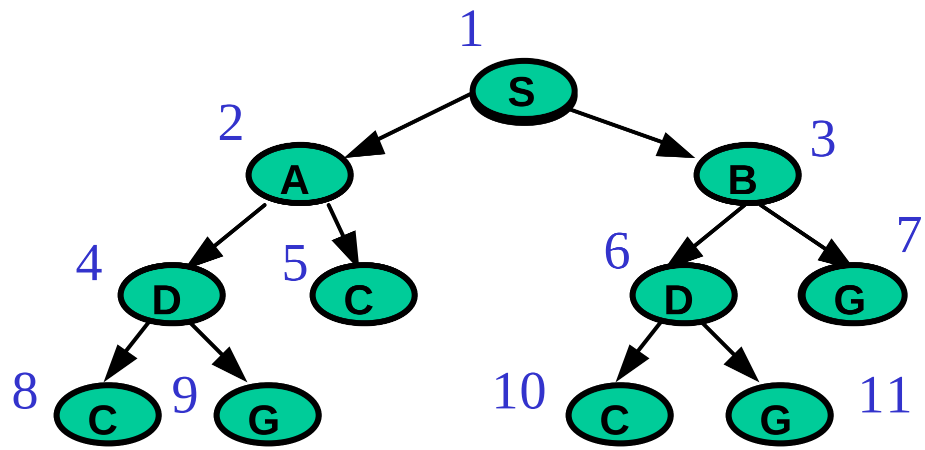
- Explore descendants before siblings
- Explore siblings left to right



Breadth First Search (BFS)

Idea:

- Explore relatives at same level before their children
- Explore relatives left to right



Elements of Algorithm Design

Description:

- stylized pseudo code, sufficient to analyze and implement the algorithm.

Analysis:

- Soundness:
  - is a solution returned by the algorithm guaranteed to be correct?
- Completeness:
  - is the algorithm guaranteed to find a solution when there is one?
- Optimality:
  - is the algorithm guaranteed to find a best 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?

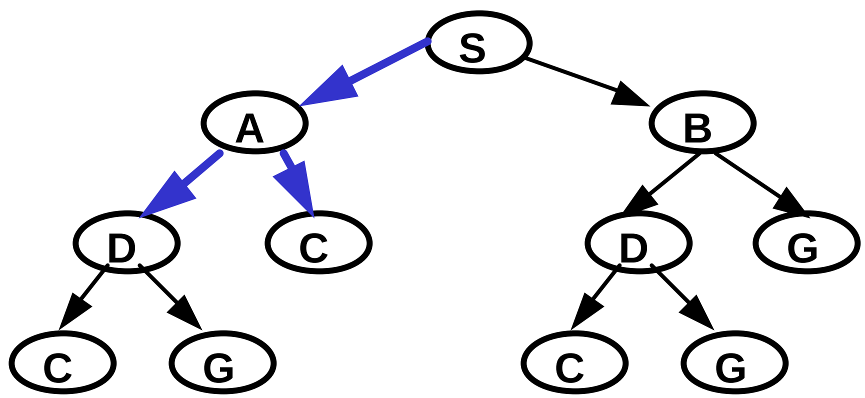
Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example (do at home)

Simple Search Algorithm

How do we maintain the Search State?

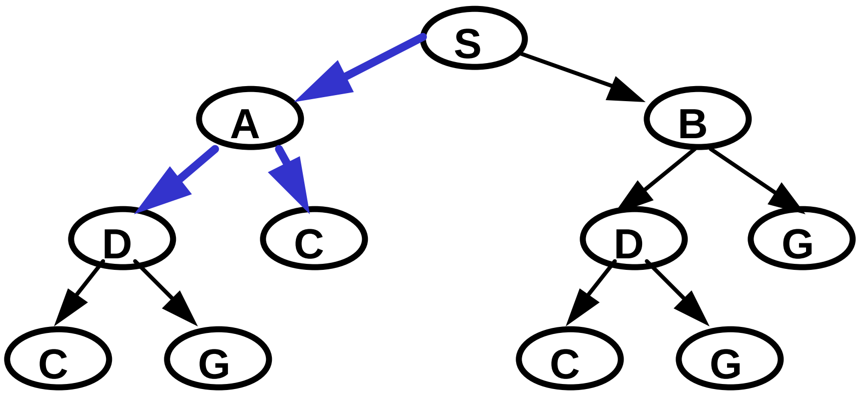
- A set of partial paths explored thus far.
- An ordering on which partial path to expand next (called a queue Q).



- Search repeatedly:
    - Selects next partial path
    - Expands it.
- Terminates when goal found.

Simple Search Algorithm

- Let S denote the start node and G a 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

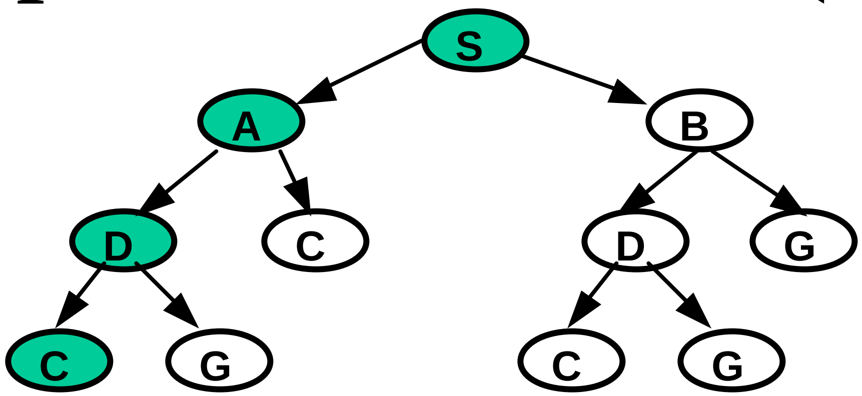
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

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 the one-step extensions of N to each child.**
   c) **Add all extended paths to Q**
   d) **Go to step 2.**

Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example (do at home)

Depth First Search (DFS)



Depth-first:

Add path extensions to **front** of Q

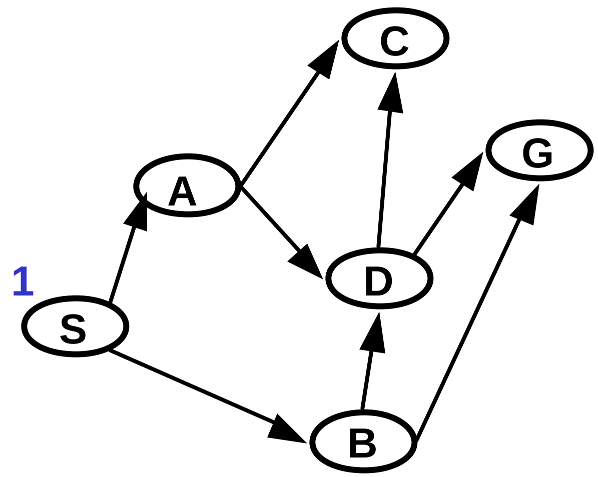
Pick first element of Q

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

Depth-First

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

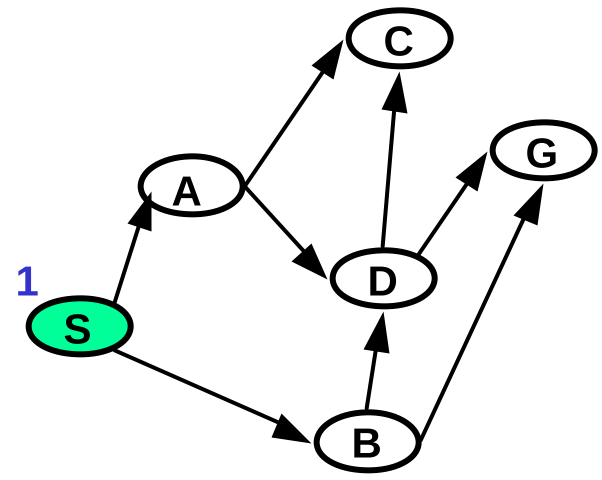
	Q
1	(S)
2	
3	
4	
5	



Depth-First

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

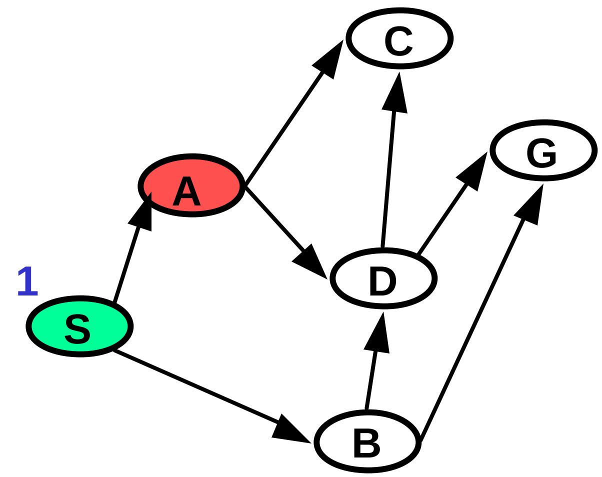
	Q
1	~~(S)~~
2	
3	
4	
5	



Depth-First

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

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

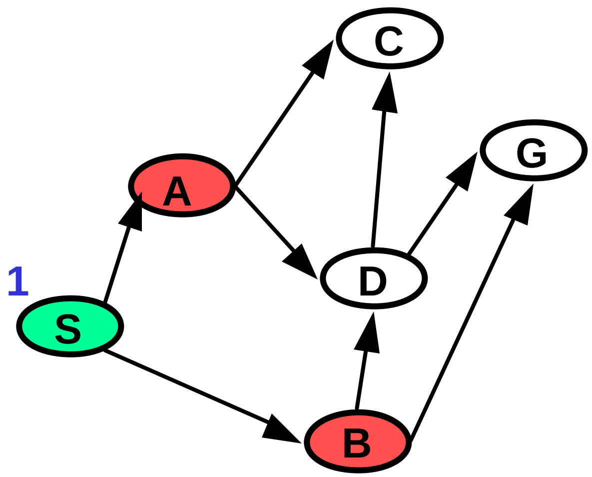


Added paths in blue

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	

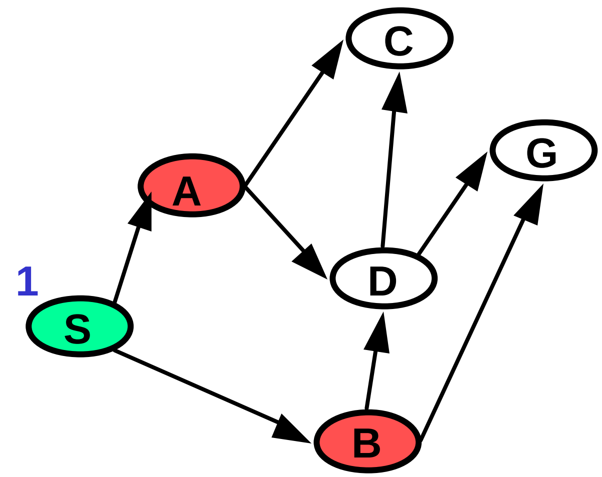


Added paths in blue

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	

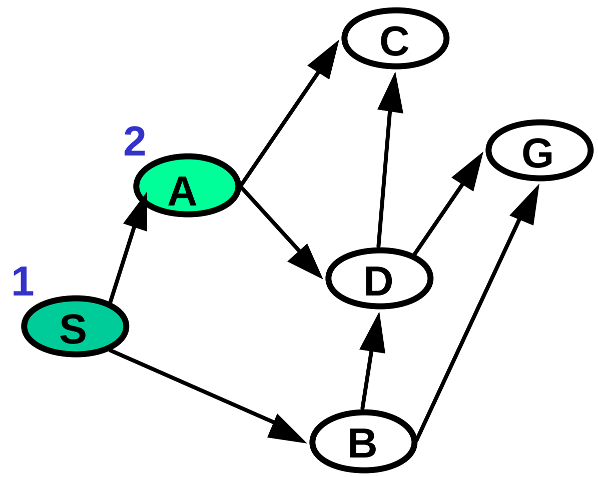


Added paths in blue

Depth-First

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

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



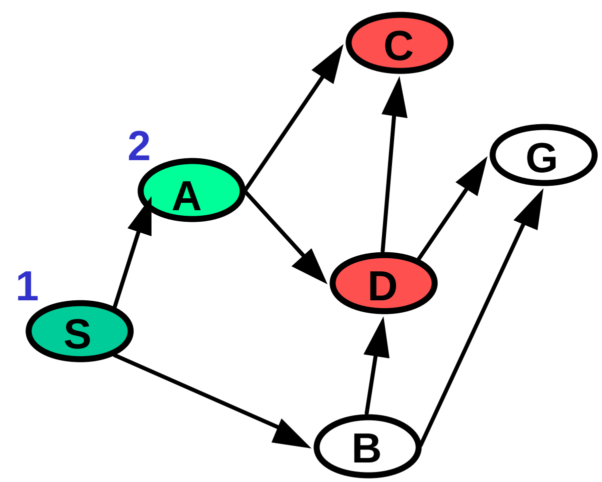
Added paths in blue

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	

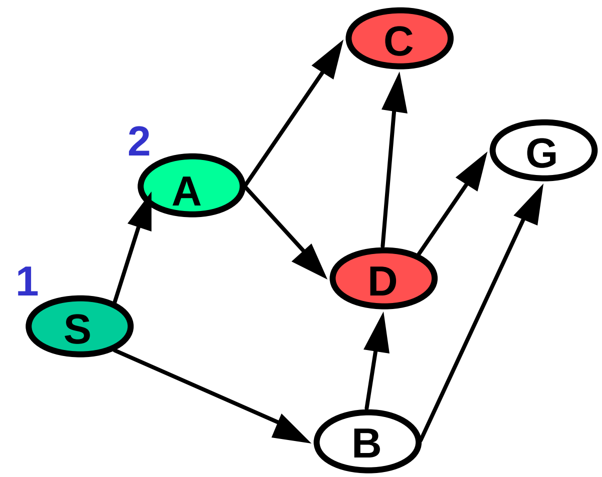
Added paths in blue



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	



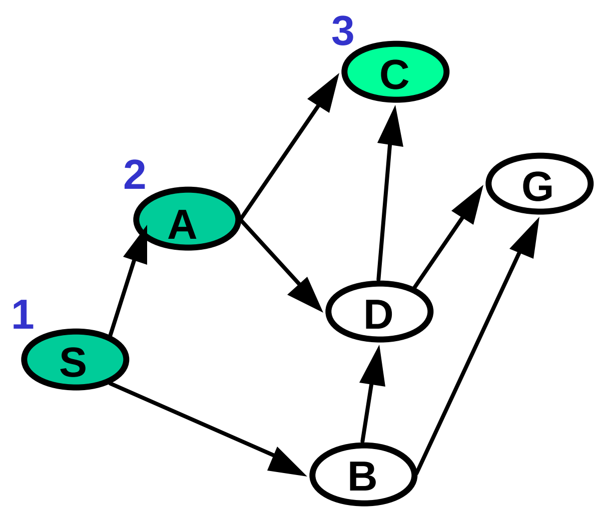
Added paths in blue

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	

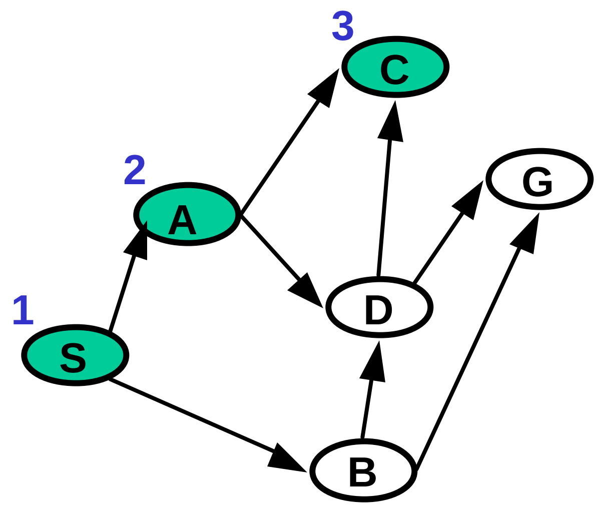
Added paths in blue



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	

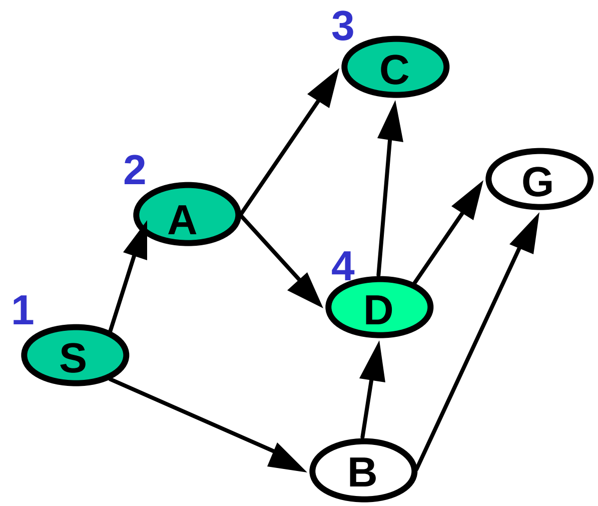


Added paths in blue

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	

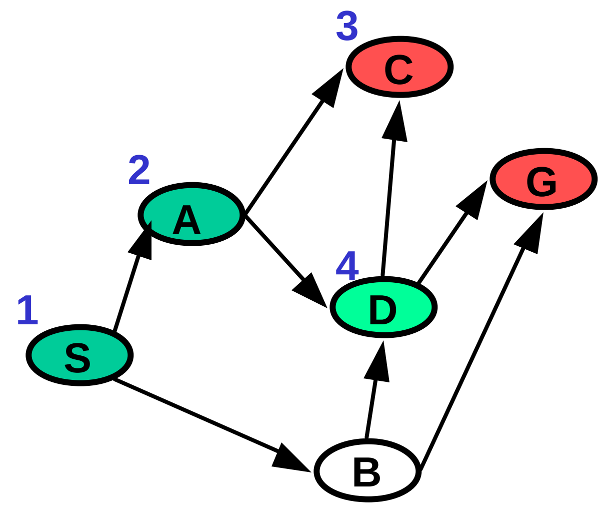


Added paths in blue

Depth-First

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

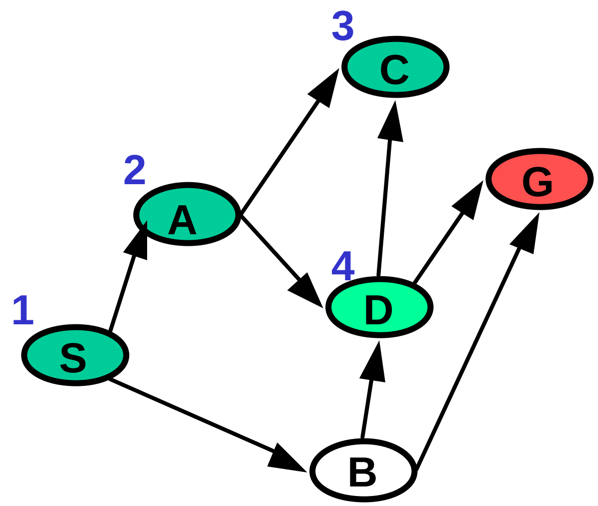
	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)



Depth-First

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

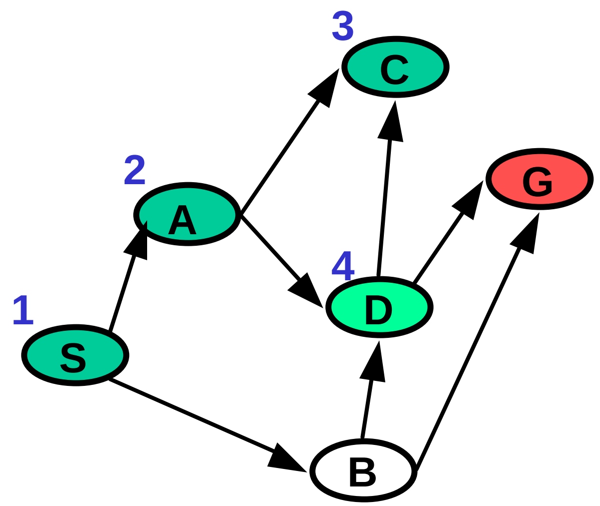
	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)



Depth-First

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

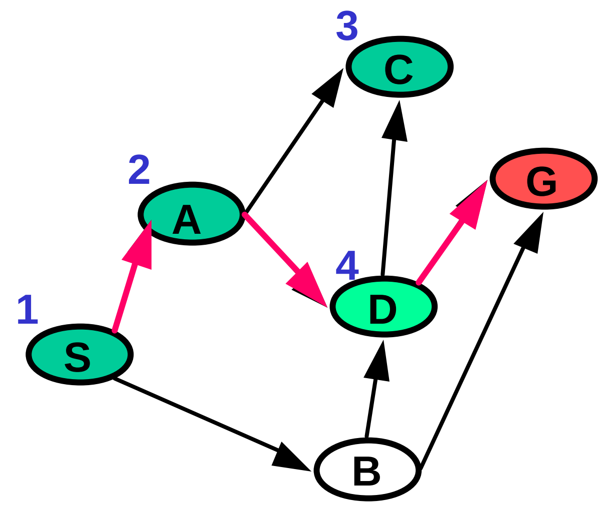
	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)
6	(G D A S)(B S)



Depth-First

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

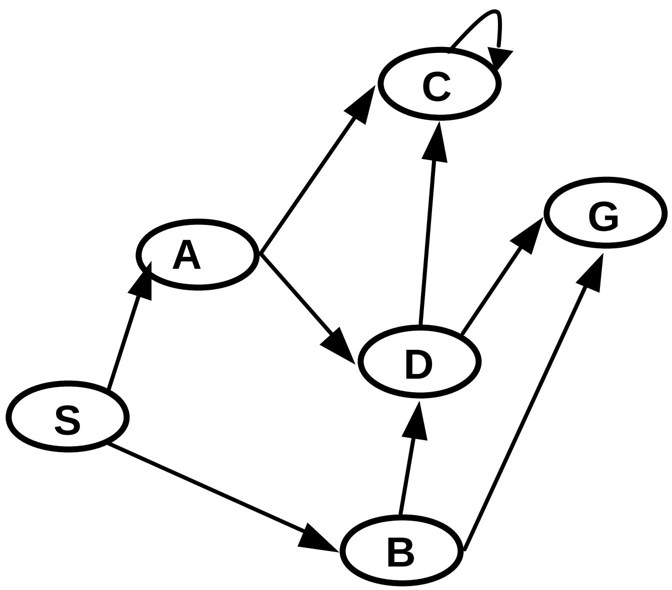
	Q
1	(S)
2	(A S) (B S)
3	(C A S) (D A S) (B S)
4	(D A S) (B S)
5	(C D A S)(G D A S) (B S)
6	(G D A S)(B S)



Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example (do at home)

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

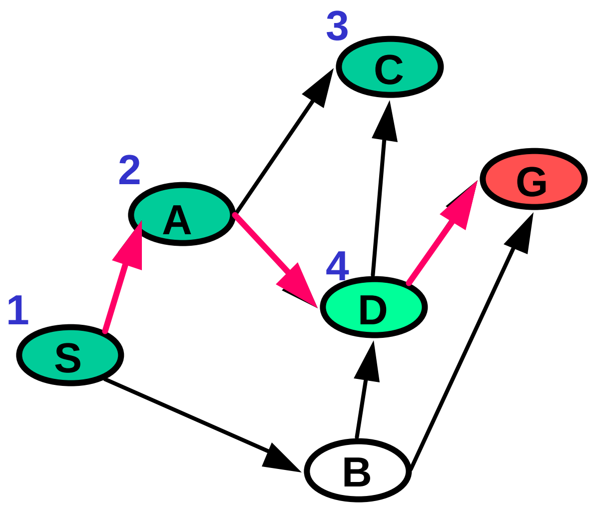


Can effort be wasted in more mild cases?

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)



- 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 it 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?

- **Guarantees each node appears at most once at the head of a path in Q.**

Simple Search Algorithm

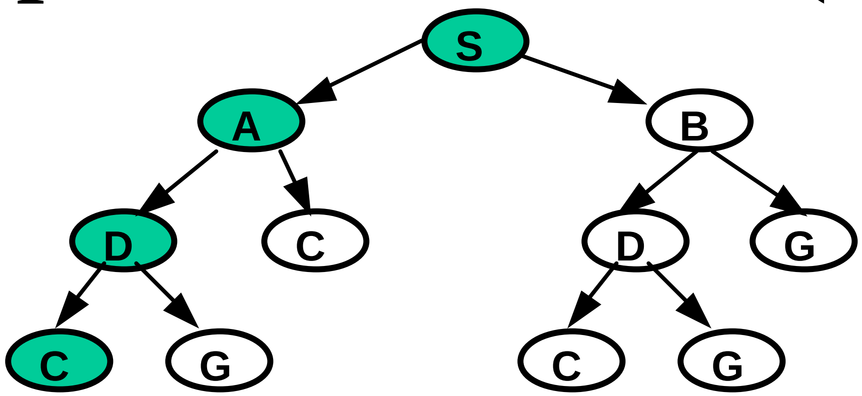
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

1. **Initialize Q with partial path (S) 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.**

Outline

- Problem encoding as state space search
- Graphs and search trees
- Depth and breadth-first search
  - A generic search algorithm
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example (do at home)

Depth First Search (DFS)

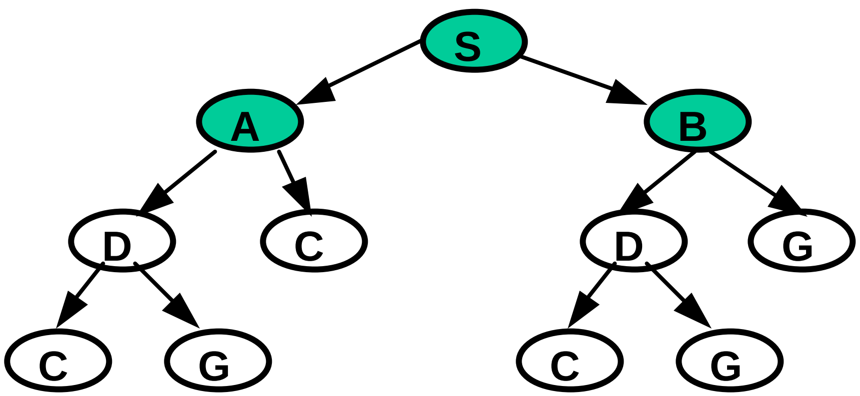


Depth-first:

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

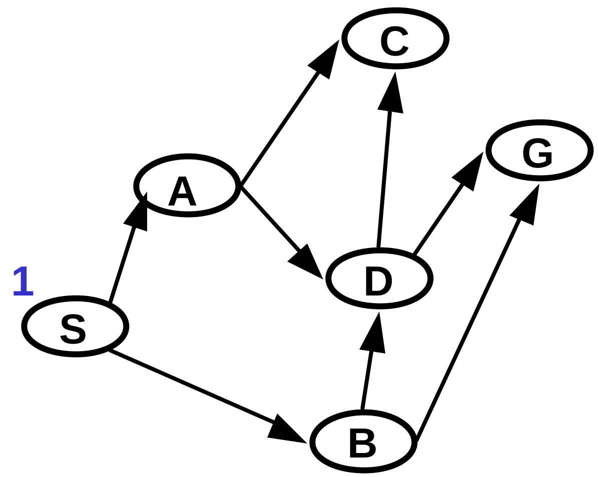
Pick first element of Q

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

Breadth-First

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

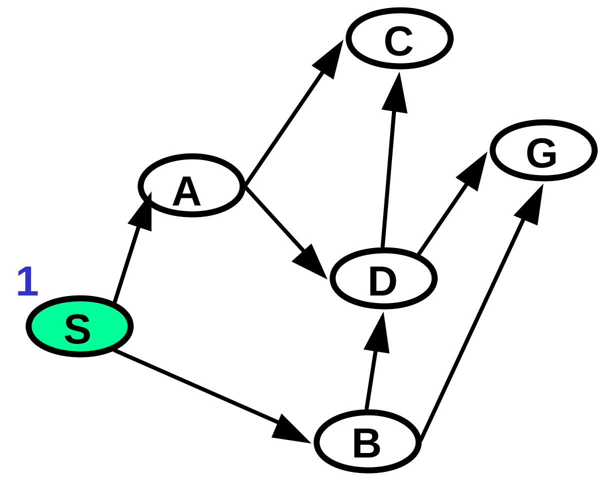
	Q	Visited
1	(S)	S
2		
3		
4		
5		
6		



Breadth-First

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

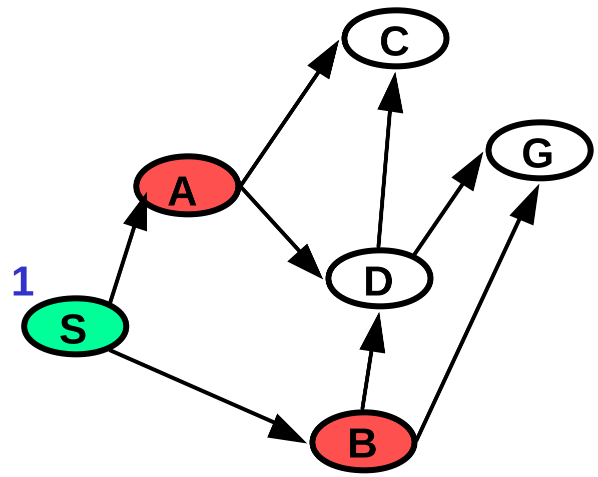
	Q	Visited
1	~~(S)~~	S
2		
3		
4		
5		
6		



Breadth-First

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

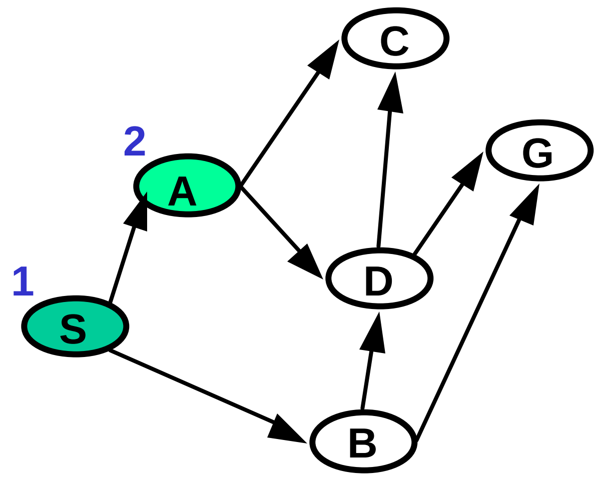
	Q	Visited
1	~~(S)~~	S
2	(A S) (B S)	A,B,S
3		
4		
5		
6		



Breadth-First

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

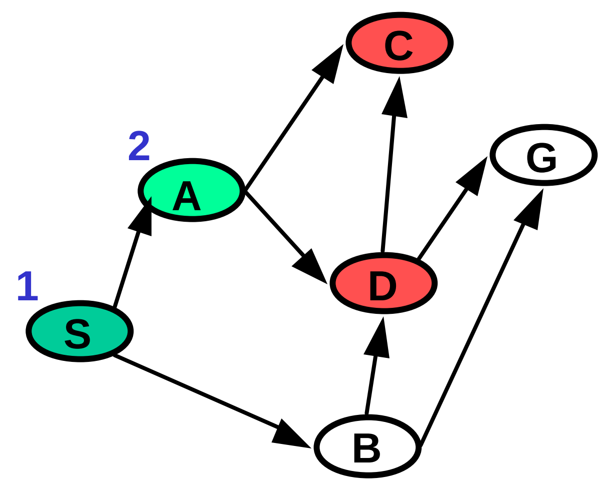
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3		
4		
5		
6		



Breadth-First

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

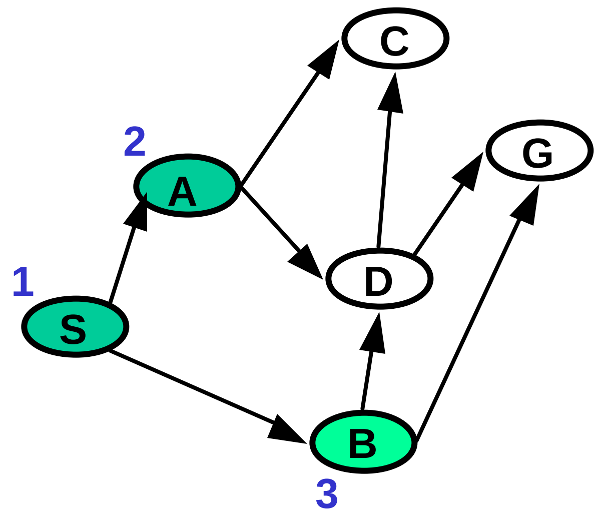
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First

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

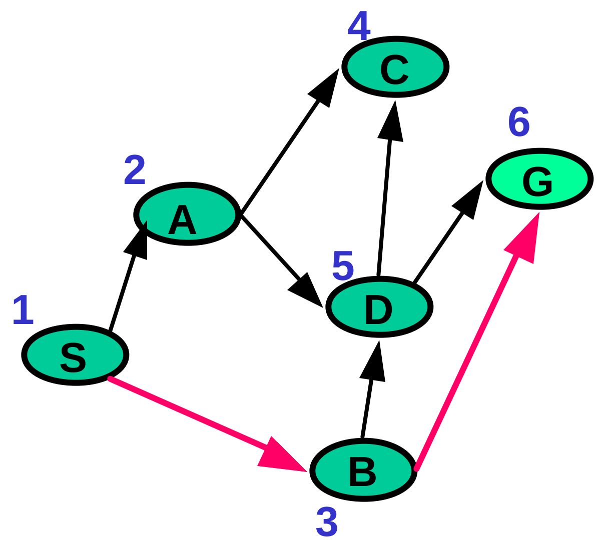
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	~~(B S)~~ (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First

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

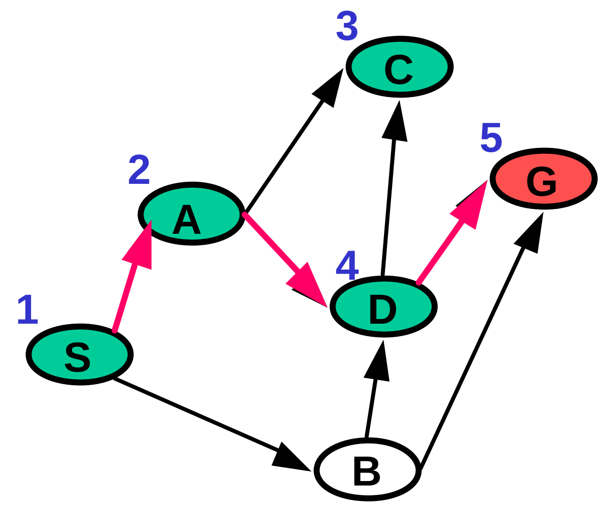
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S
5	(D A S) (G B S)	G,C,D,B,A,S
6	(G B S)	G,C,D,B,A,S



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	(G D A S) (B S)	G,C,D,B,A,S



Summary

- Most problem solving tasks may be encoded as state space search.
- Basic data structures 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	(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)