Problem Solving as State Space Search

Brian C. Williams
16.410-13
Sept 13th, 2010

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

Assignments

- Remember:
  Problem Set #1: Java warm up
  Out last Wednesday,
  Due this Wednesday, September 15th
- Reading:
  - Today: Solving problems through search *[AIMA]* Ch. 3.1-4
  - Wednesday: Asymptotic Analysis Lecture 2 Notes of 6.046J;
    Recurrences, Lecture 12 Notes of 6.042J.

Recap - Course Objectives

1. Understand the major types of agents and architectures:
   – goal-directed vs utility-based
   – deliberative vs reactive
   – model-based vs model-free

2. Learn the modeling and algorithmic building blocks for creating agents:
   – Model problem in an appropriate formal representation.
   – Specify, analyze, apply and implement reasoning algorithms to solve the problem formulations.

Recap – Agent Architectures

Functions: Robust, coordinated operation + mobility

It Begins with State Space Search!

Problem Solving as State Space Search

- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

Most Agent Building Block Implementations Use Search

Robust Operations:

- Activity Planning
- Diagnosis
- Repair
- Scheduling
- Resource Allocation

Mobility:

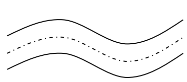
- Path Planning
- Localization
- Map Building
- Control Trajectory Design

Example: Outpost Logistics Planning

Image produced for NASA by John Frassanito and Associates.

Can the astronaut get its supplies safely across a Lunar crevasse?

Astronaut
Goose
Grain
Fox

Rover 

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

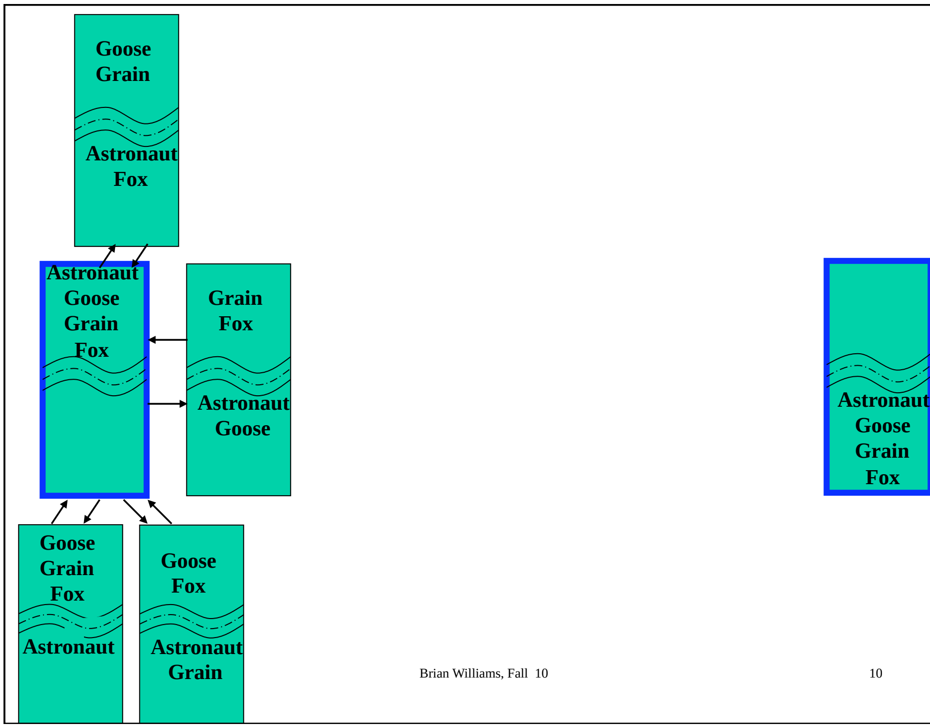
Early AI: What are the universal problem solving methods?

Simple $\not\rightarrow$ Trivial

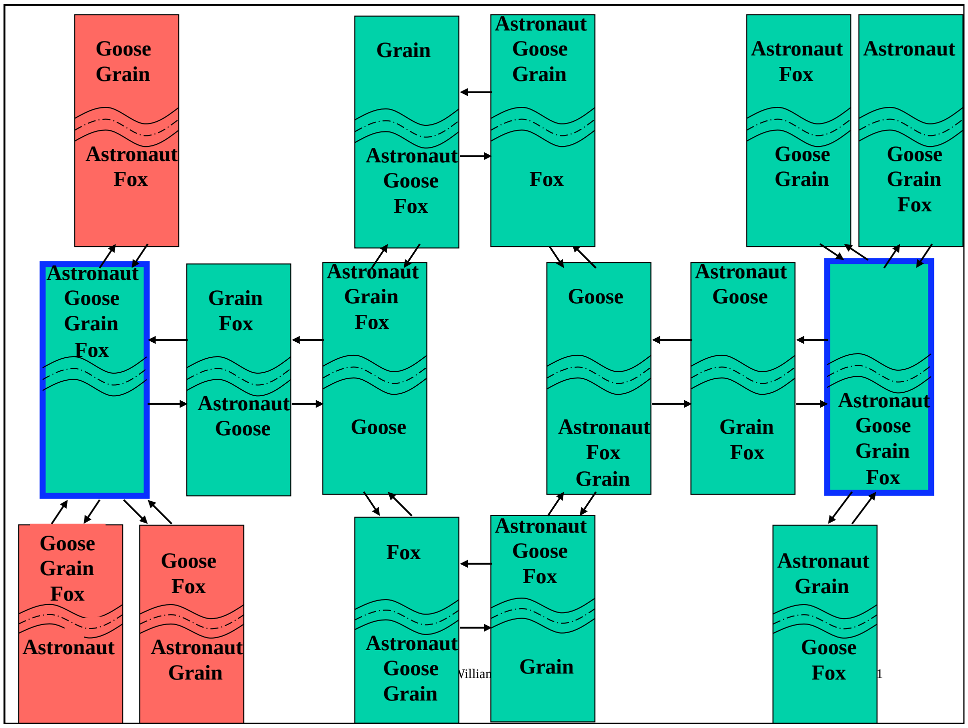
Problem Solving as State Space Search

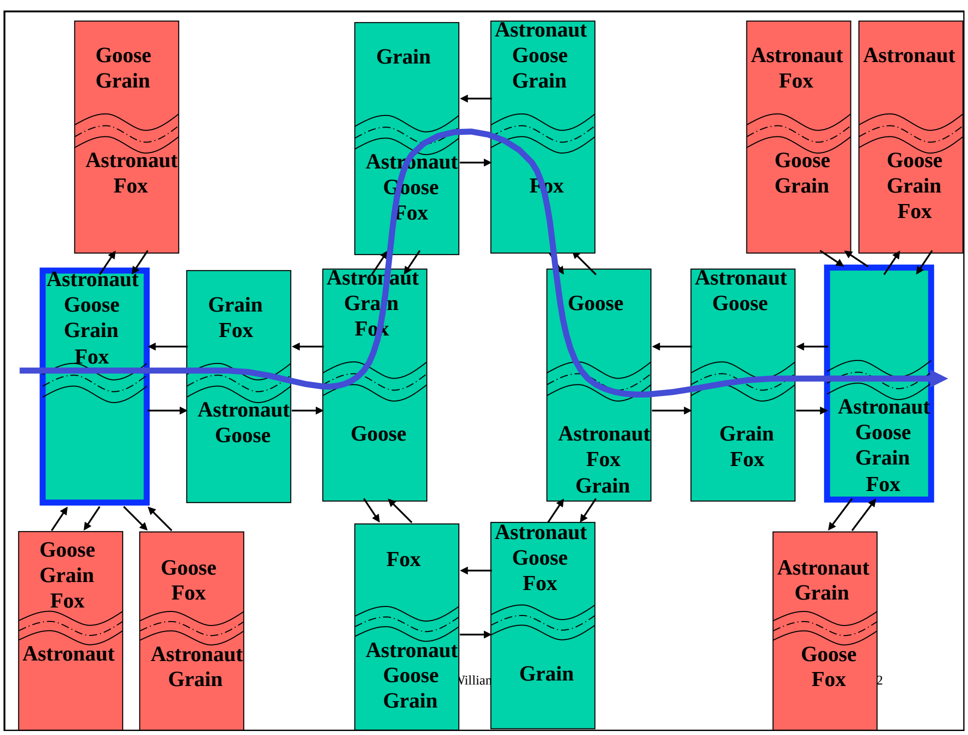
- Formulate Goal
  - State
    - Astronaut, Fox, Goose & Grain below crevasse.
- Formulate Problem
  - States
    - Astronaut, Fox, Goose & Grain above or below the crevasse.
  - Operators
    - Move: Astronaut drives rover and 1 or 0 items to other side of crevasse.
  - Initial State
    - Astronaut, Fox, Goose & Grain above crevasse.
- Generate Solution
  - Sequence of Operators (or States)
    - Move(goose,astronaut), Move(astronaut), . . .

Brian Williams, Fall 10 9



Brian Williams, Fall 10 10


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

Formulation Example: 8-Puzzle

5	4	
6	1	8
7	3	2

Start

1	2	3
8		4
7	6	5

Goal

- States: integer location for each tile AND …??
- Operators: move empty square up, down, left, right
- Initial and Goal States: as shown above

Languages for Expressing States and Operators for Complex Tasks

- Swaggert & Lovell assemble emergency rig for Apollo 13 lithium hydroxide unit.

Image source: NASA.

Example: STRIPS Planning Language

Initial state:

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

goal (partial state):

(and (connected a b)
(connected b c)))

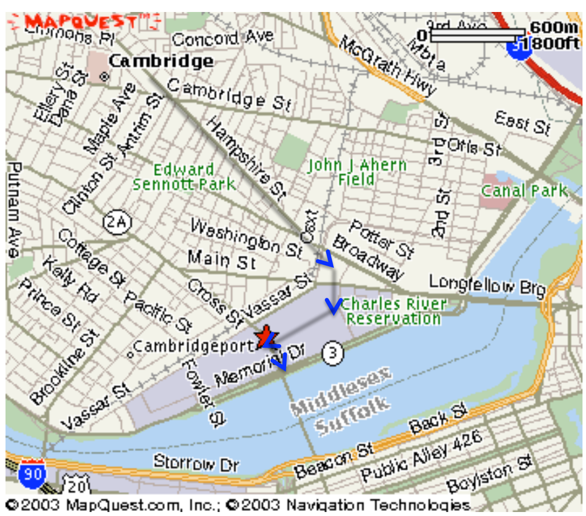
Operators

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

pickup hose

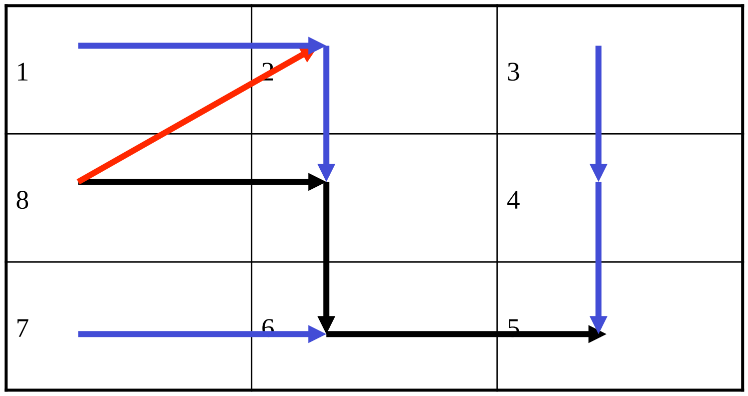
effect: (and (not (clear hose))
(not (on-table hose))
(not (empty arm))
(holding arm hose)))

Problem:
Find a Route from home to MIT



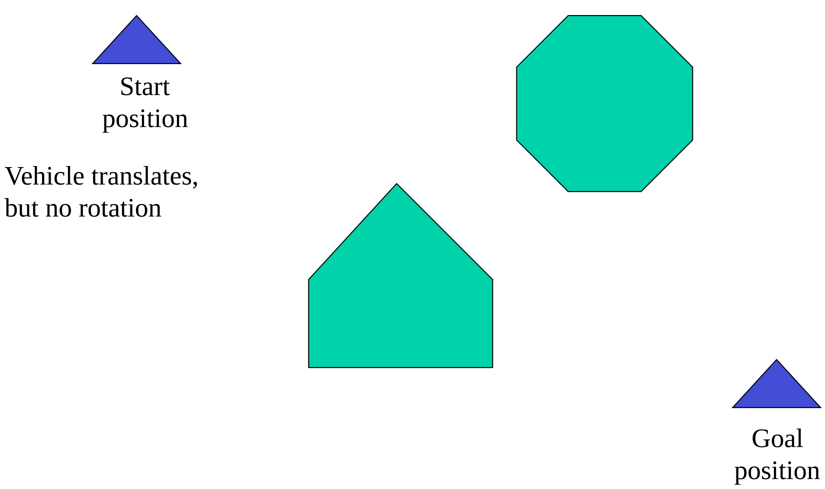
States? Operators?, Initial and Goal State?

Problem: Compensating for Error Online

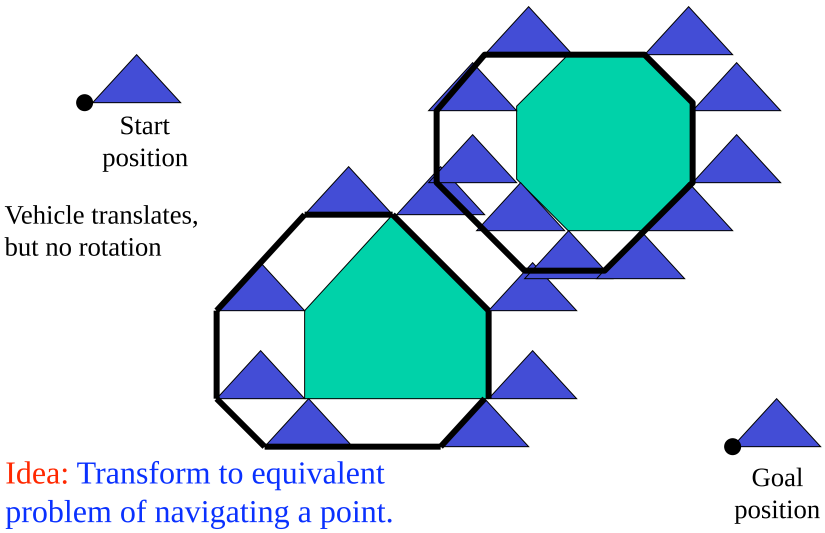


- Policy, $\pi(v) \rightarrow e$, dictates how to act in all states.
- Policy π corresponds to a shortest path tree from all vertices to the destination.

How do we Map Path Planning to State Space Search?



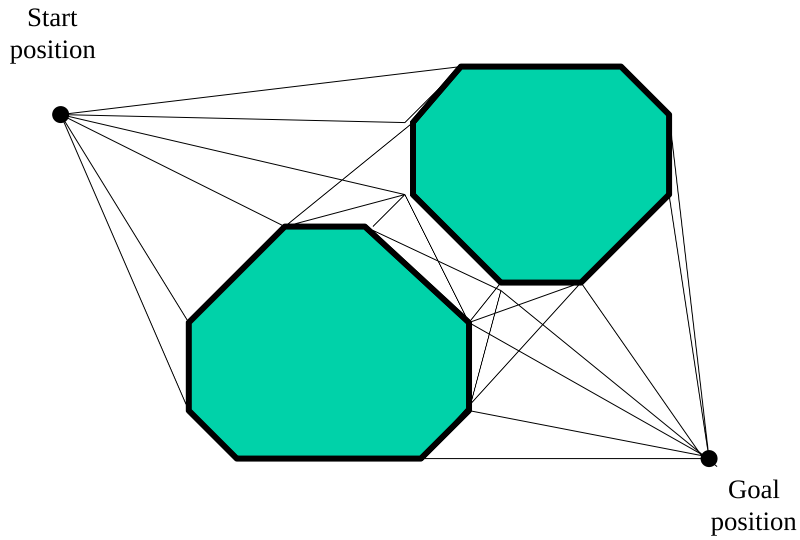
1. Create Configuration Space



Idea: Transform to equivalent problem of navigating a point.

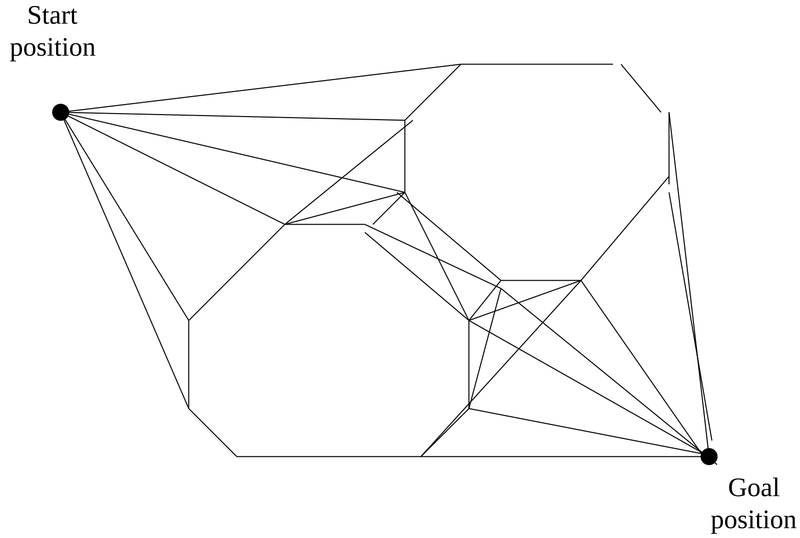
19

2. Map From Continuous Problem to Graph Search: Create Visibility Graph



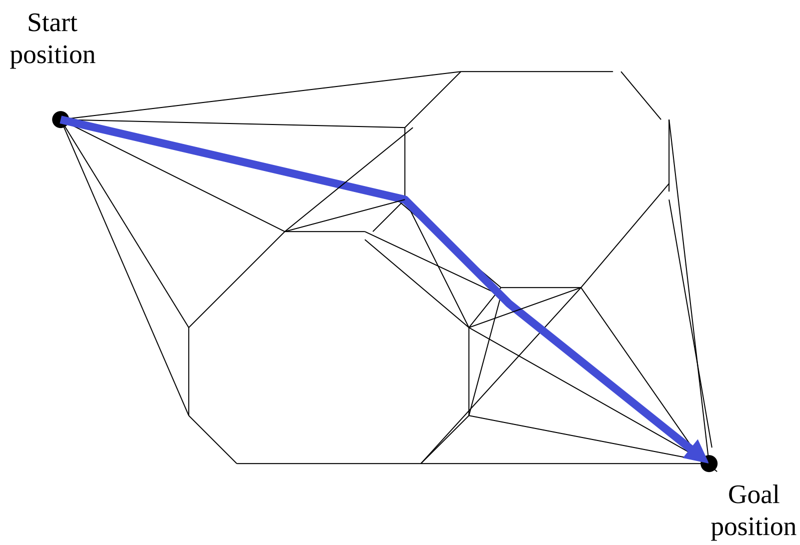
20

2. Map From Continuous Problem to Graph Search: Create Visibility Graph



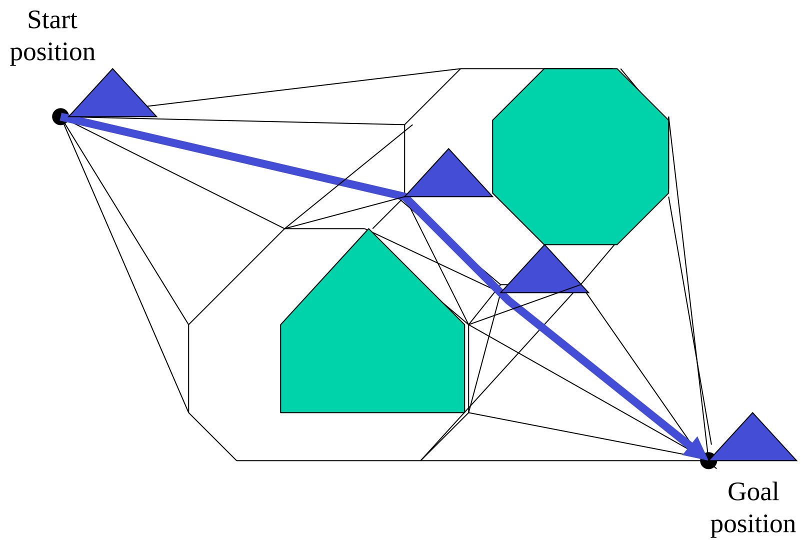
21

3. Find Shortest Path (e.g., A*)



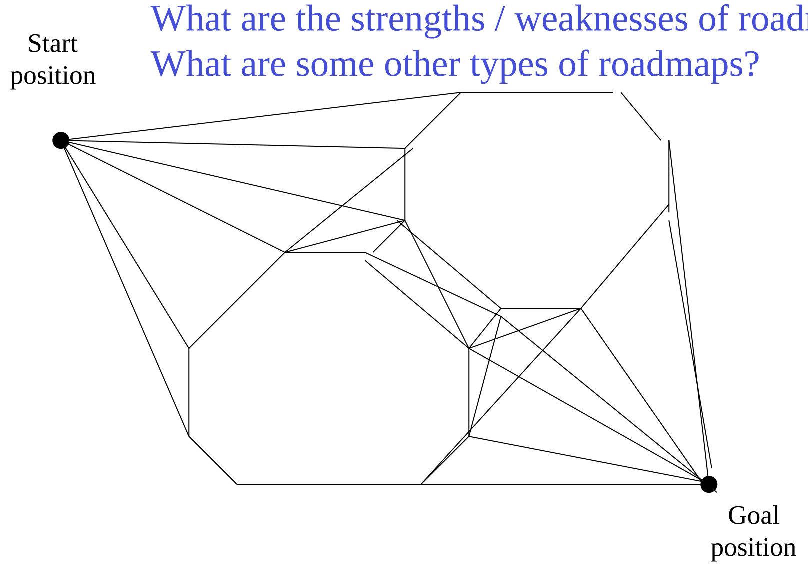
22

Resulting Solution

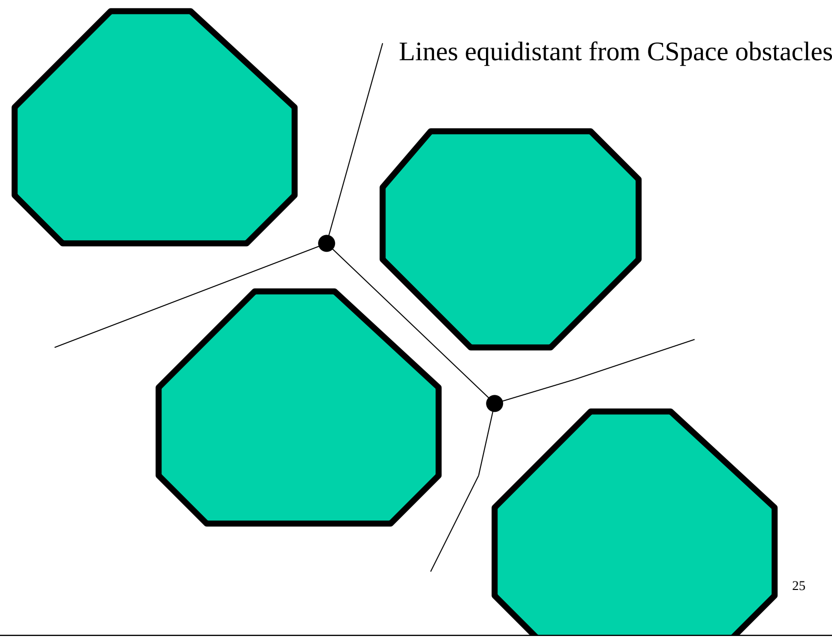


A Visibility Graph is a Kind of Roadmap

What are the strengths / weaknesses of roadmaps?
What are some other types of roadmaps?

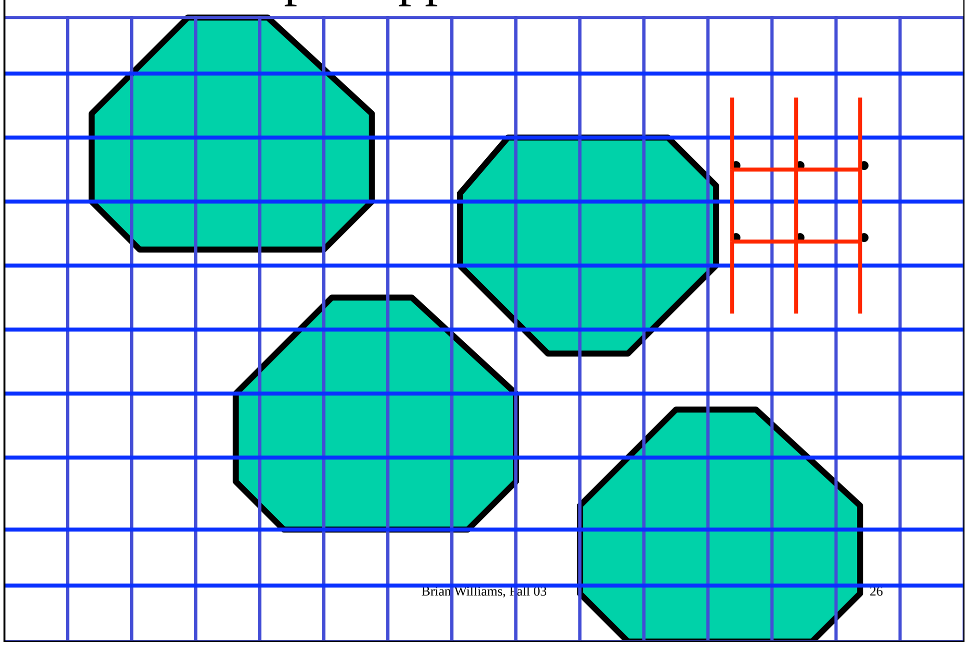


Voronoi Diagrams



25

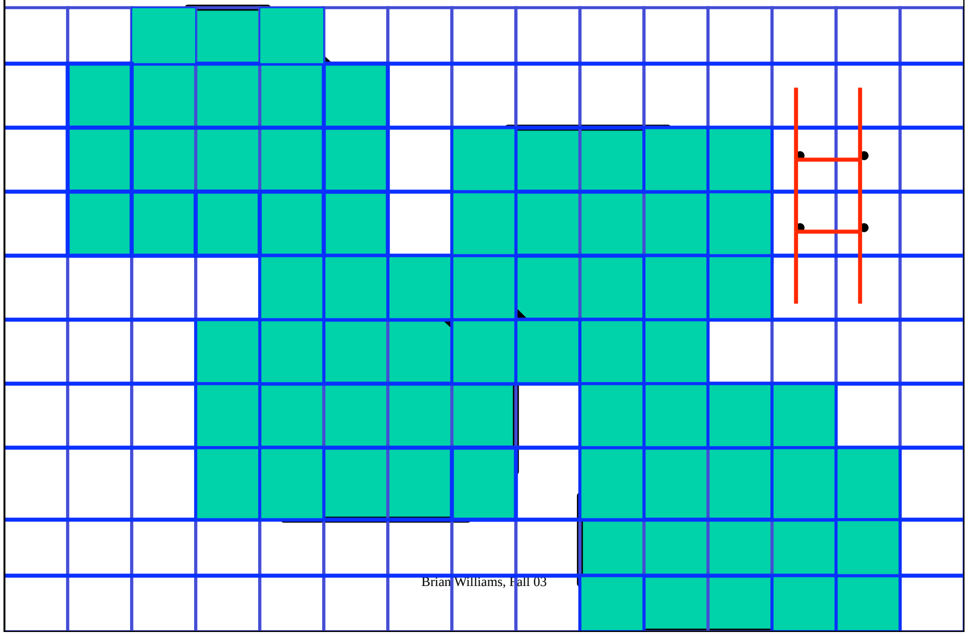
Roadmaps: Approximate Fixed Cell



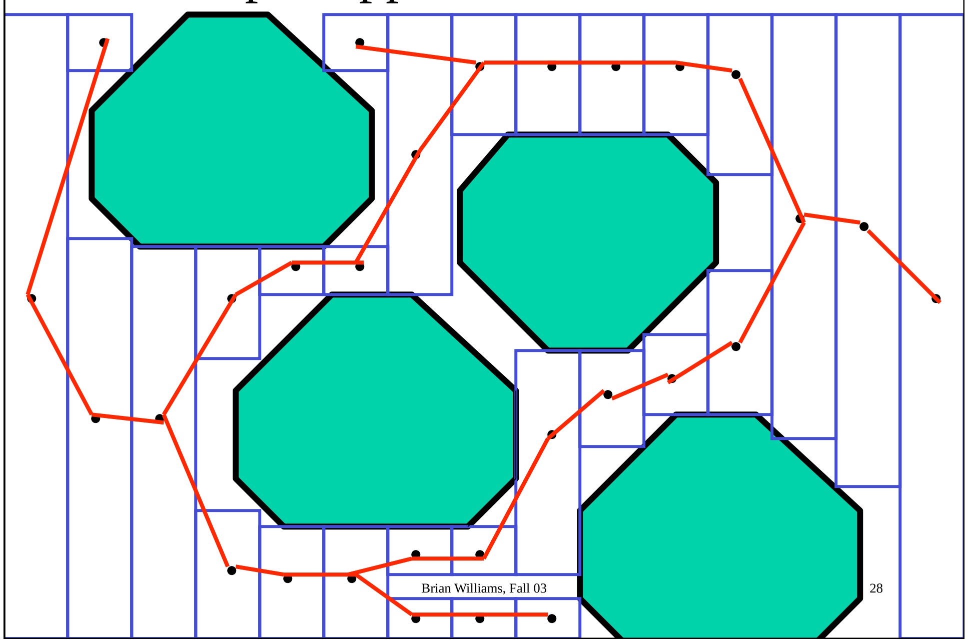
Brian Williams, Fall 03

26

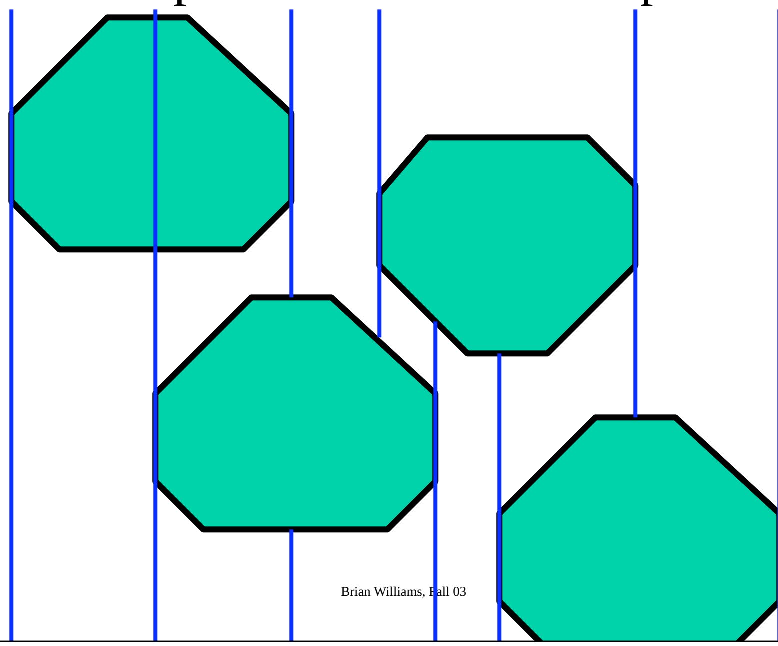
Roadmaps: Approximate Fixed Cell



Roadmaps: Approximate Variable Cell

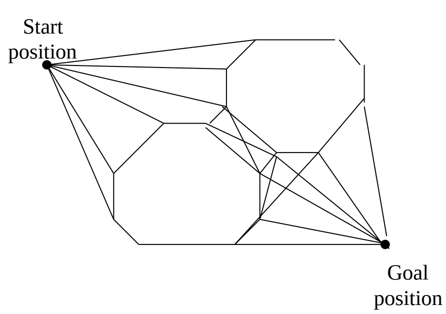


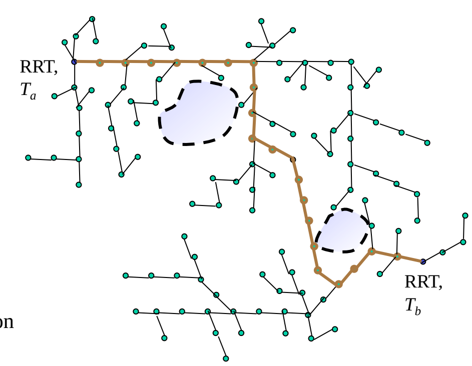
Roadmaps: Exact Cell Decomposition



Brian Williams, Fall 03 29

How do we handle large state spaces?





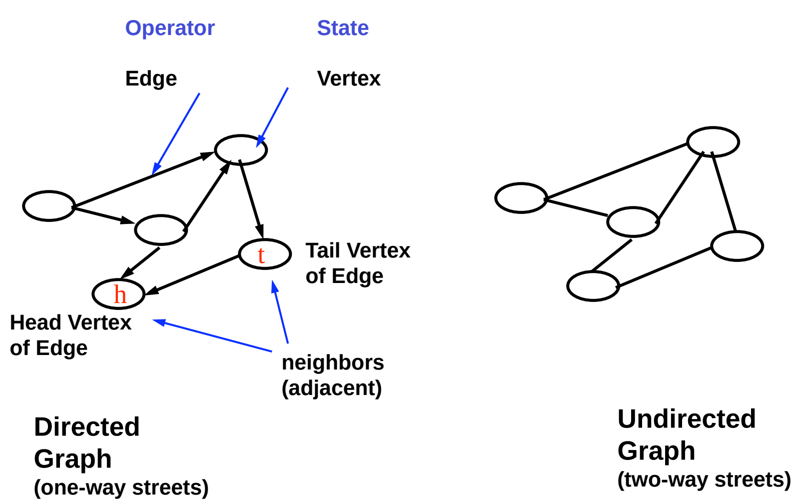
Rapid exploring Random Trees

Brian Williams, Fall 03 30

Problem Solving as State Space Search

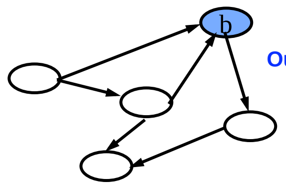
- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

Problem Formulation: A Graph



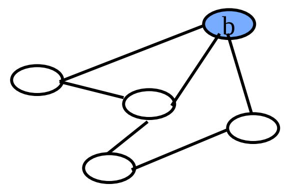
Problem Formulation: A Graph

In Degree (2)



Out Degree (1)

Degree (3)



Directed Graph
(one-way streets)

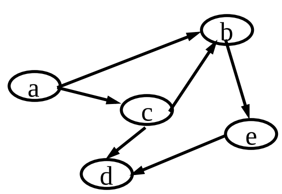
Undirected Graph
(two-way streets)

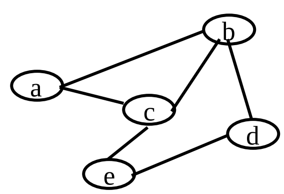
Problem Formulation: A Graph

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

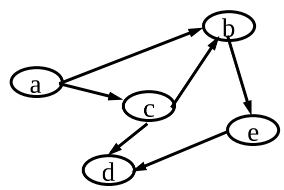
Clique
A complete subgraph
(All vertices are adjacent).

Directed Graph
(one-way streets)

Undirected Graph
(two-way streets)

Specifying a Graph: G = <V, E>

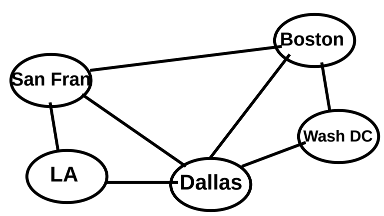
Notation:
<a, b, … n> an ordered list of elements a, b …
{a, b, … n} an unordered set of distinguished elements.



Vertices V = {a, b, c, d, e}

Edges E = {<a, b>, <a, c>,
<b, e>,
<c, b>, <c,d>,
<e, d>}

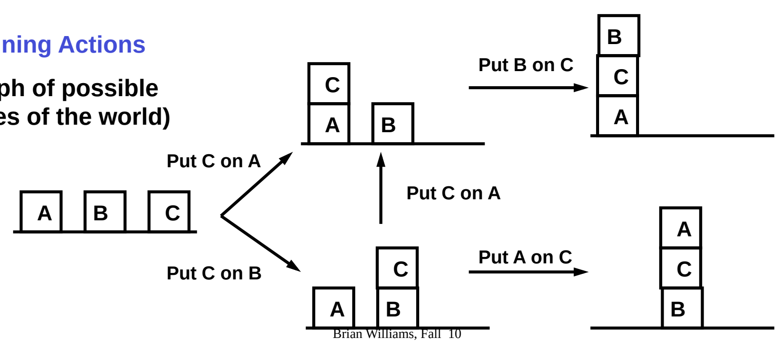
Examples of Graphs



Roadmap

Planning Actions

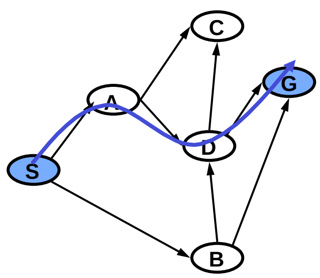
(graph of possible states of the world)



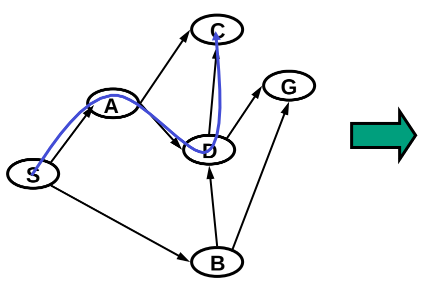
Formalizing State Space Search

Input: A search problem S = <g, S, G> where
- **graph g = <V, E>,**
- **start vertex S in V, and**
- **goal vertex G in V.**

Output: A simple path P = <S, v2, … G> in g from S to G.



Simple Paths of Graph g = <V, E>



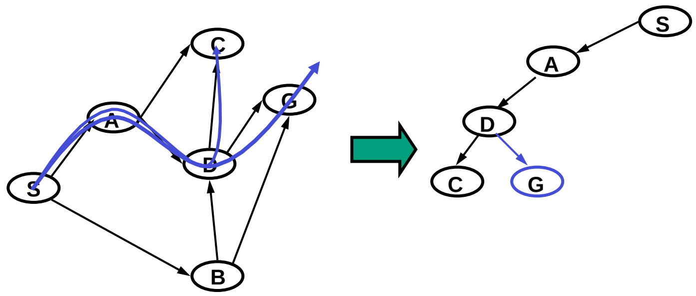
<S, A, D, C>

start **end**

A (directed) path P of graph g is
a sequence of vertices $\langle v_1, \dots v_n \rangle$ in V
such that each successive pair $\langle v_i, v_{i+1} \rangle$ is a (directed) edge in E

A simple path is a path that has no cycles.
A cycle is a subpath where start = end (*i.e.*, repeated vertices).

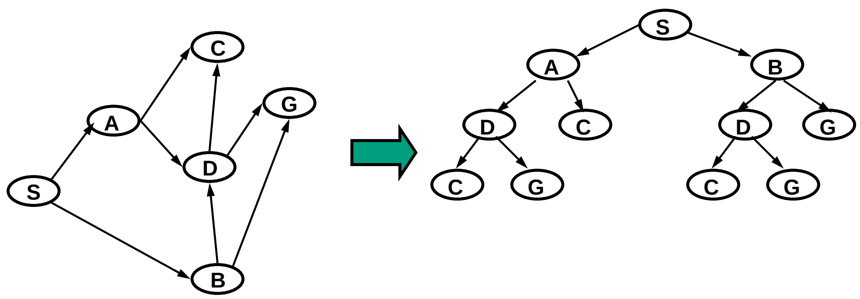
A Problem Solver Searches through all Simple Paths



Brian Williams, Fall 10

39

A Search Tree Denotes All Simple Paths



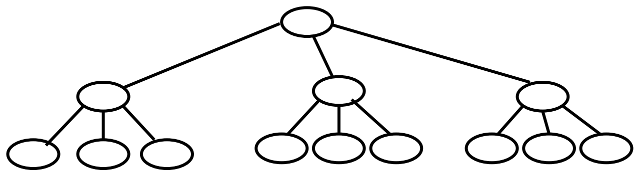
Enumeration is:
- Complete
- Systematic
- Sound

Brian Williams, Fall 10

40

Search Trees

think of a tree as a “family” tree



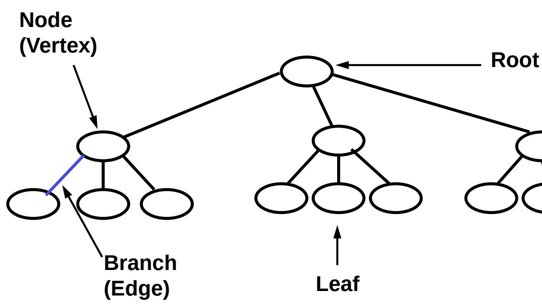
A tree T is a directed graph, such that

• there exists exactly one undirected path between any pair of vertices.

•In degree of each vertex is 1

Brian Williams, Fall 10 41

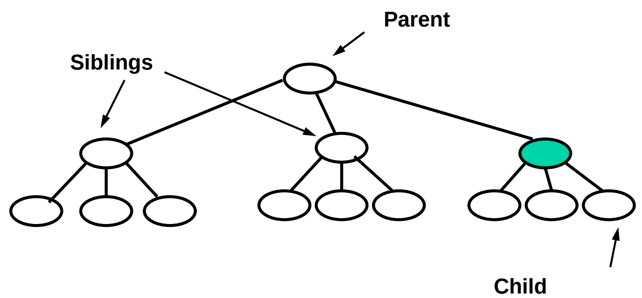
Search Trees



think of a tree as a “family” tree

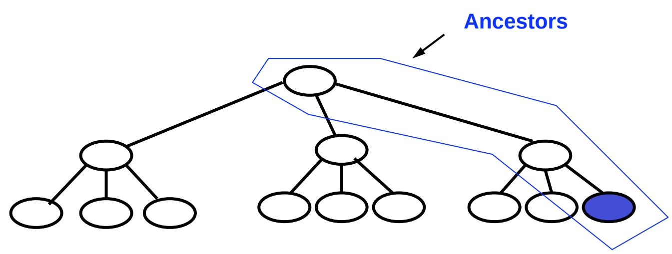
Brian Williams, Fall 10 42

Search Trees



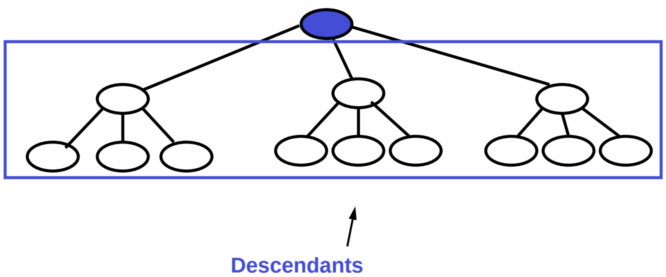
think of a tree as a “family” tree

Search Trees



think of a tree as a “family” tree

Search Trees



think of a tree as a "family" tree

Problem Solving as State Space Search

- Problem Formulation (Modeling)
  - Problem solving as state space search
- Formal Representation
  - Graphs and search trees
- Reasoning Algorithms
  - Depth and breadth-first search

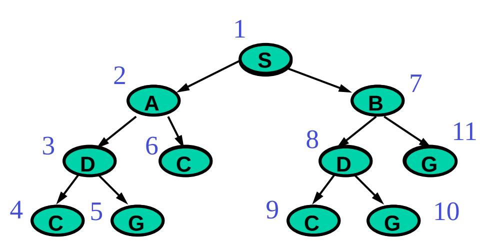
Classes of Search

Blind (uninformed)	Depth-First Breadth-First Iterative-Deepening	Systematic exploration of whole tree until the goal is found.
Optimal (informed)	**A*** **Branch&Bound**	Use path "length" measure. Find "shortest" path.
Heuristic (informed)	Hill-Climbing Best-First Beam	Use heuristic measure of goodness of a node.

Depth First Search (DFS)

Local Rule: After visiting node…

- Visit its children before its siblings
- Visit its children left to right

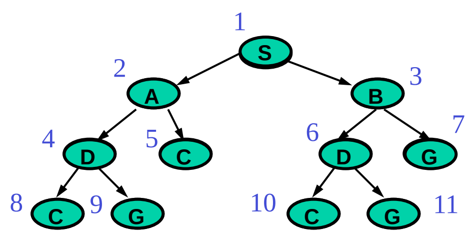


Breadth First Search (BFS)

Local Rule: After visiting node…

• Visit its siblings, before its children

• Visit its children left to right



Elements of Algorithm Design

Algorithm Description: (Today)

– stylized pseudo code, sufficient to analyze and implement the algorithm (implementation next Wednesday).

Algorithm Analysis: (Wednesday & Monday)

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

Problem Solving as State Space Search

- Problem Formulation (Modeling)
- Formal Representation
- Reasoning Algorithms
  - A generic search algorithm description
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

Solve <g = <V, E>, S, G> using State Space Search

Search States:

- All simple paths <S, …v> in g starting at S

Initial State:

- <S>

Operator:

- Extend a path <S, … v> to <S, … v, u> for each <v, u> in E
  - call u a child of v

Goal:

- A simple path <S, …, G> in g

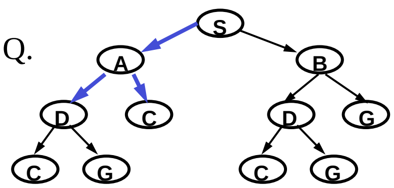
Solve <g = <V, E>, S, G> using State Space Search

How do we maintain the search state?

- An ordering on partial paths yet to be expanded (called a queue Q).

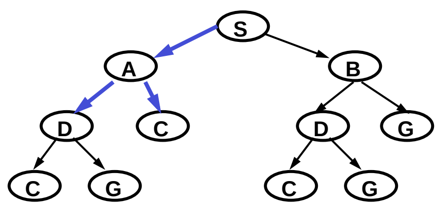
How do we perform search?

- Repeatedly:
    1. Select next partial path from Q.
    2. Expand it.
    3. Add expansions to Q.
- Terminate when goal G is found.



Simple Search Algorithm: Preliminaries

- A partial path from S to D is listed in reverse order,
    - e.g., <D, A, S>
- The head of a partial path is its most recent visited node,
    - 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,
S be the Start node and
G be the Goal node.

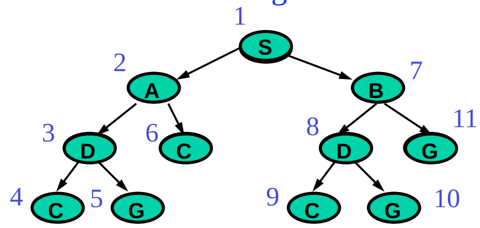
1. Initialize Q with partial path <S>
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
   a) Remove N from Q
   b) Find all children of head(N) and
   create a one-step extension of N to each child
   c) Add all extended paths to Q
   d)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

Problem Solving as State Space Search

- Problem Formulation (Modeling)
- Formal Representation
- Reasoning Algorithms
  - A generic search algorithm description
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

Depth First Search (DFS)

Idea: After visiting node
- Visit its children left to right (or top to bottom)
- Visit its children before its siblings



Assume we remove the first element of Q,

Where to Q do we add the path extensions?

Brian Williams, Fall 10 57

Simple Search Algorithm

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

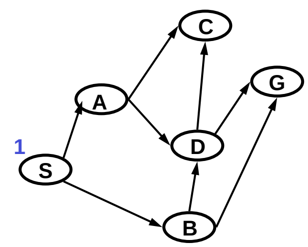
1. Initialize Q with partial path <S>
2. If Q is empty, fail. Else, pick a partial path N from Q
3. If head(N) = G, return N (goal reached!)
4. Else:
    a) Remove N from Q
    b) Find all children of head(N) and create a one-step extension of N to each child
    c) Add all extended paths to Q
    d)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

Brian Williams, Fall 10 58

Depth-First

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

	Q
1	(S)
2	
3	
4	
5	



Simple Search Algorithm

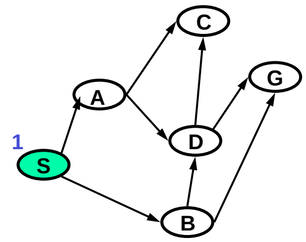
Let Q be a list of partial paths,
S be the Start node and
G be the Goal node.

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 a one-step extension 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

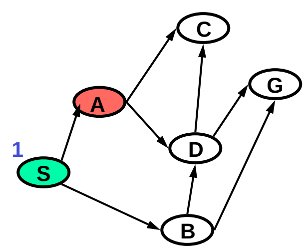
	Q
1	~~(S)~~
2	
3	
4	
5	



Depth-First

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

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

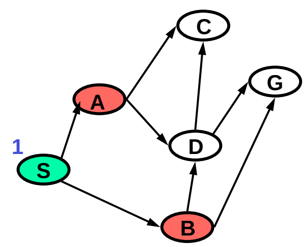


Added paths in blue

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

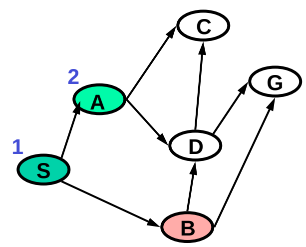
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

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 a one-step extension 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	

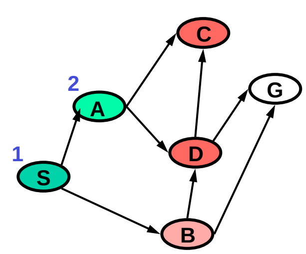


Added paths in blue

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	

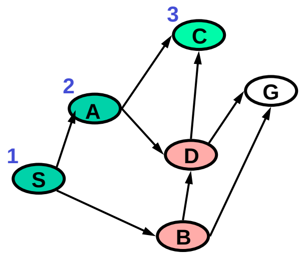


Added paths in blue

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	

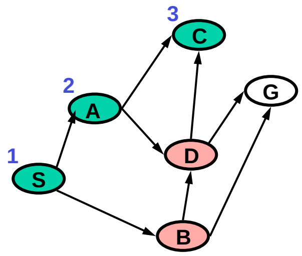


Added paths in blue

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	(D A S) (B S)
5	

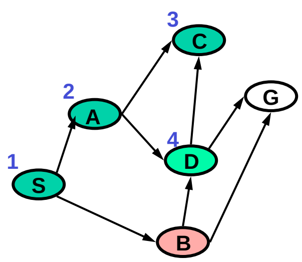


Added paths in blue

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	

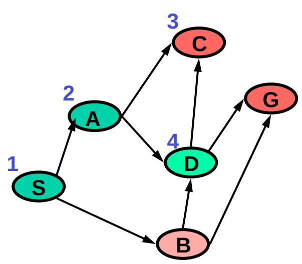


Added paths in blue

Depth-First

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

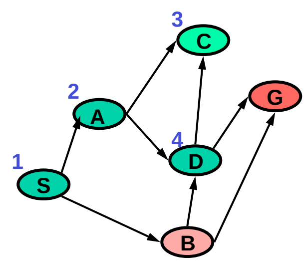
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	(C D A S)(G D A S) (B S)



Depth-First

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

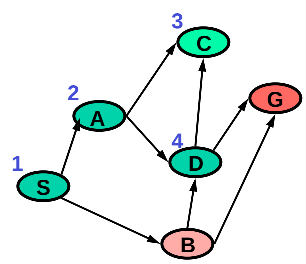
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	~~(C D A S)~~(G D A S) (B S)



Depth-First

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

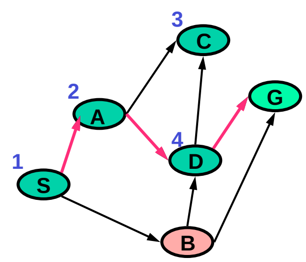
	Q
1	~~(S)~~
2	~~(A S)~~ (B S)
3	~~(C A S)~~ (D A S) (B S)
4	~~(D A S)~~ (B S)
5	~~(C D A S)~~(G D A S) (B S)
6	(G D A S)(B S)



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)



Simple Search Algorithm

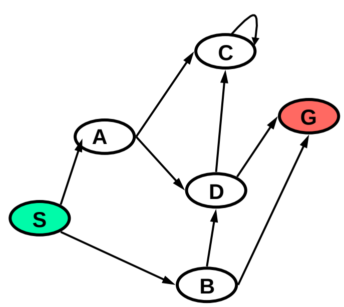
Let Q be a list of partial paths,
Let S be the start node and
Let G be the Goal node.

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 a one-step extension of N to each child
   c) Add all extended paths to Q
   d) Go to step 2

Problem Solving as State Space Search

- Problem Formulation (Modeling)
- Formal Representation
- Reasoning Algorithms
  - A generic search algorithm description
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

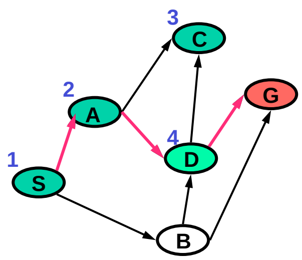
Issue: Starting at S and moving top to bottom, will depth-first search ever reach G?



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 expanded path on Q if head is a visited node.

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 this always improve efficiency?

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

How Do We Modify The 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 a one-step extension 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 a one-step extension 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

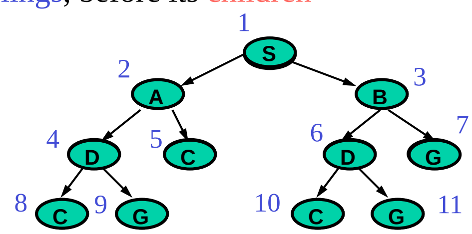
- **This algorithm stops (in step 3) when head(N) = G.**
- **We could have performed this test in step 6 as each extended path is added to Q. This would catch termination earlier and be perfectly correct for all the searches we have covered so far.**
- **However, performing the test in step 6 will be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

Problem Solving as State Space Search

- Problem Formulation (modeling)
- Formal Representation
- Reasoning Algorithms
  - A generic search algorithm description
  - Depth-first search example
  - Handling cycles
  - Breadth-first search example

Breadth First Search (BFS)

Idea: After visiting node
- Visit its children left to right
- Visit its siblings, before its children



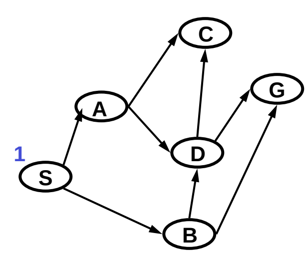
Assume we remove the first element of Q,

Where to Q do we add the path extensions?

Breadth-First with Visited List

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

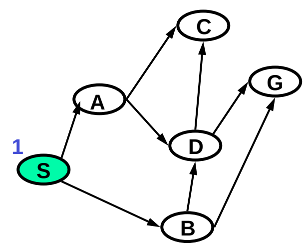
	Q	Visited
1	(S)	S
2		
3		
4		
5		
6		



Breadth-First with Visited List

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

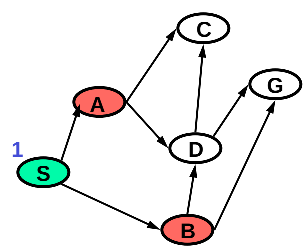
	Q	Visited
1	~~(S)~~	S
2		
3		
4		
5		
6		



Breadth-First with Visited List

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

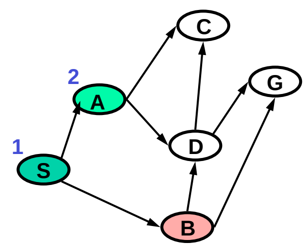
	Q	Visited
1	~~(S)~~	S
2	(A S) (B S)	A,B,S
3		
4		
5		
6		



Breadth-First with Visited List

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

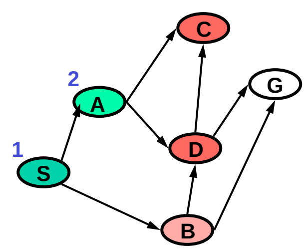
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3		
4		
5		
6		



Breadth-First with Visited List

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

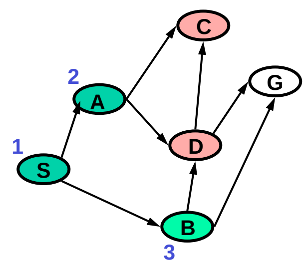
	Q	Visited
1	(S)	S
2	~~(A S)~~ (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First with Visited List

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

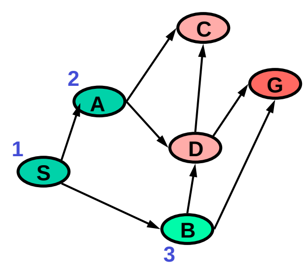
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	~~(B S)~~ (C A S) (D A S)	C,D,B,A,S
4		
5		
6		



Breadth-First with Visited List

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		

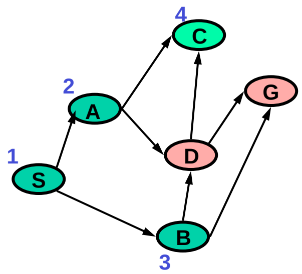


* We could stop here, when the first path to the goal is generated.

Breadth-First with Visited List

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		

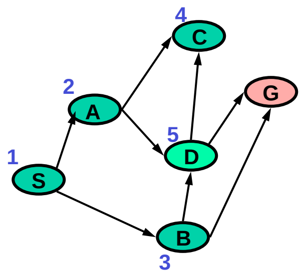


* We could stop here, when the first path to the goal is generated.

Breadth-First with Visited List

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

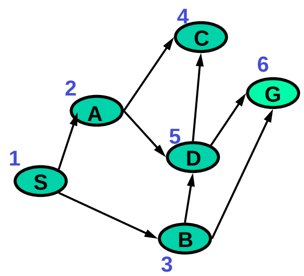
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	~~(C A S)~~ (D A S) (G B S)*	G,C,D,B,A,S
5	~~(D A S)~~ (G B S)	G,C,D,B,A,S
6		



Breadth-First with Visited List

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

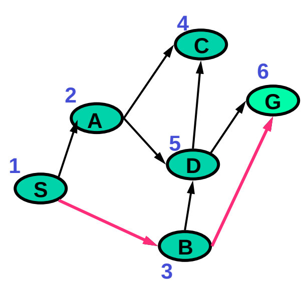
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	(~~C A~~ S) (D A S) (G B S)*	G,C,D,B,A,S
5	(~~D A~~ S) (G B S)	G,C,D,B,A,S
6	(~~G~~ B S)	G,C,D,B,A,S



Breadth-First with Visited List

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

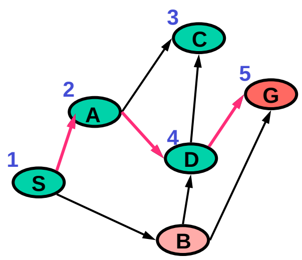
	Q	Visited
1	(S)	S
2	(A S) (B S)	A,B,S
3	(B S) (C A S) (D A S)	C,D,B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S
5	(D A S) (G B S)	G,C,D,B,A,S
6	(G B S)	G,C,D,B,A,S



Depth-first with Visited List

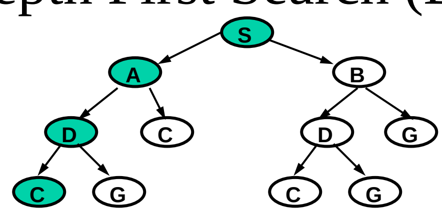
Pick first element of Q; Add path extensions to front of Q

	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



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

Depth First Search (DFS)

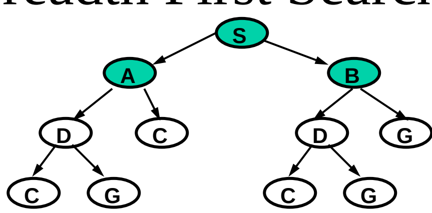


Depth-first:

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

What You Should Know

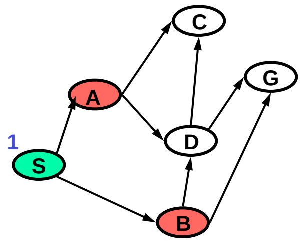
- Most problem solving tasks may be formulated as state space search.
- State space search is formalized using graphs, simple paths, search trees, and pseudo code.
- Depth-first and breadth-first search are framed, among others, as instances of a generic search strategy.
- Cycle detection is required to achieve efficiency and completeness.

Appendix

Breadth-First (without Visited list)

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

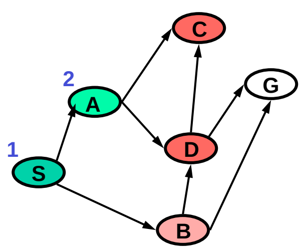
	Q
1	(S)
2	
3	
4	
5	
6	
7	



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	

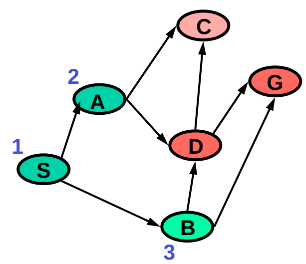


Added paths in blue

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	

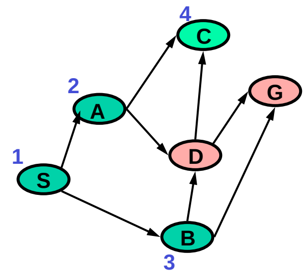


Added paths in blue

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

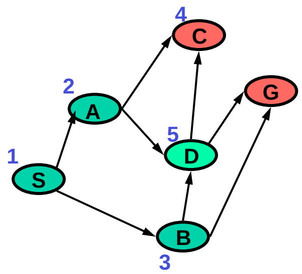
Revisited nodes in pink

* 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

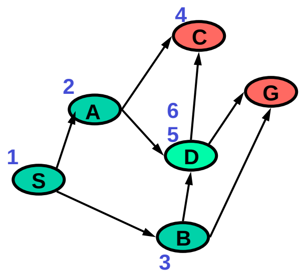
	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	(D A S) (D B S) (G B S)
6	
7	



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)(C D B S)(G D B S)