# CS4618: Artificial Intelligence I

# Searching State Spaces


### Derek Bridge
### School of Computer Science and Information Technology
### University College Cork


# Initialization

In [1]:

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

# State space search

- If the current state is not a goal state,
  - **Expand** the current state, i.e. generate its **successor** states
  - One successor becomes the new current state
  - Others must be kept in case we want to come back to them
    - Keep them on an **agenda**, i.e. a list of unexplored options
- **Search strategy** determines which state to expand next

# A general state space search algorithm

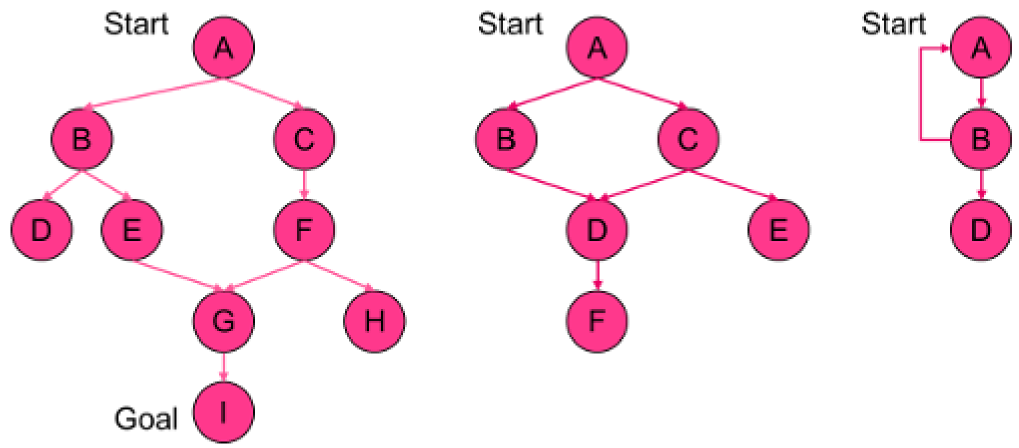
StateSpaceSearch()

- insert start state onto $agenda$;
- while $agenda$ is not empty
  - $currentState$ = remove from front of $agenda$;
  - if $currentState$ satisfies goal condition
    - return the path of actions that led to $currentState$;

    else
    - $successors$ = states that result from expanding $currentState$;
    - insert $successors$ onto $agenda$;
- return fail;

# Search tree

- The parts of the state space that the search algorithm actually visits are made explicit in a **search tree**
- State space and search tree are different:
  - Some search strategies may leave parts of the state space unexplored
  - Some search strategies may re-explore parts of the state space
- These ideas will be made clearer in the lecture using these three example state spaces:




# Avoiding re-exploration

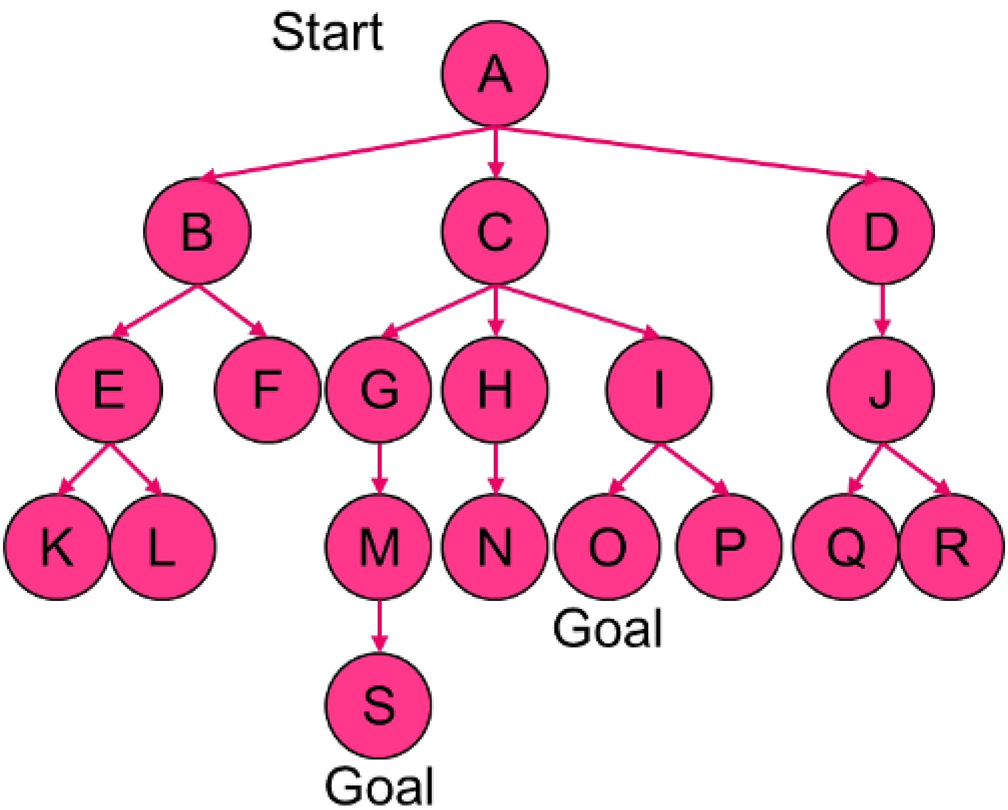
- Three options, varying in cost and effectiveness:
  - Discard any successor that is the same as the current node's parent
  - Discard any successor that is the same as another node on the path
  - Discard any successor if it is the same as any previously-generated node
- Class exercise: How effective at avoiding re-exploration are these three options? What do they cost in time & space?

# Search strategy

- **Search strategy** decides which state to expand next
- **Uninformed** search strategies:
  - No problem-specific heuristic knowledge is used in the decision
  - E.g. breadth-first; depth-first; least-cost
- **Informed** search strategies:
  - Problem-specific heuristic knowledge is used
  - E.g. greedy; $A^*$
- Search strategy is determined by where nodes are added to the agenda

# Breadth-first search

- Treat the agenda as a **queue**:
  - nodes are removed from the front (as always)
  - successors are added to the *back*
- Hence, all nodes at depth $i$ in the search tree will be expanded before any at $i + 1$
- We will illustrate in the lecture using this state space



# Depth-first search

- Treat the agenda as a **stack**:
  - nodes are removed from the front (as always)
  - successors are added to the *front*
- Hence, it always choses to expand one of the nodes that is at the deepest level of the search tree (it only expands nodes at a shallower level if the search has hit a dead-end at the deepest level)

# Evaluating a search strategy

- **Completeness**:
  - A search strategy is complete if it guarantees to find a solution (path to goal) when there is one
- Class exercise:
  - Is breadth-first complete?
  - Is depth-first complete?

# Evaluating a search strategy

- **Optimality**:
  - A search strategy is optimal if it guarantees to find the highest-quality solution (least-cost path to goal) when there is a solution
    - i.e. the first solution it finds is the one with lowest-cost
- Class exercise:
  - Is breadth-first optimal?
  - Is depth-first optimal?

# Evaluating a search strategy

- Time-complexity:
  - How long does it take to find a solution, in terms of size of state space?
  - Worst-case (also best-case and average-case)
- Space-complexity:
  - How much memory is needed, in terms of size of state space?
  - Worst-case (also best-case and average-case)
- Class exercise: For both breadth-first and depth-first search, work out the time- and space-complexity

In [ ]:

```

```