CS4618: Artificial Intelligence I

Search Strategies

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

Initialization

In [1]:

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

In [2]:

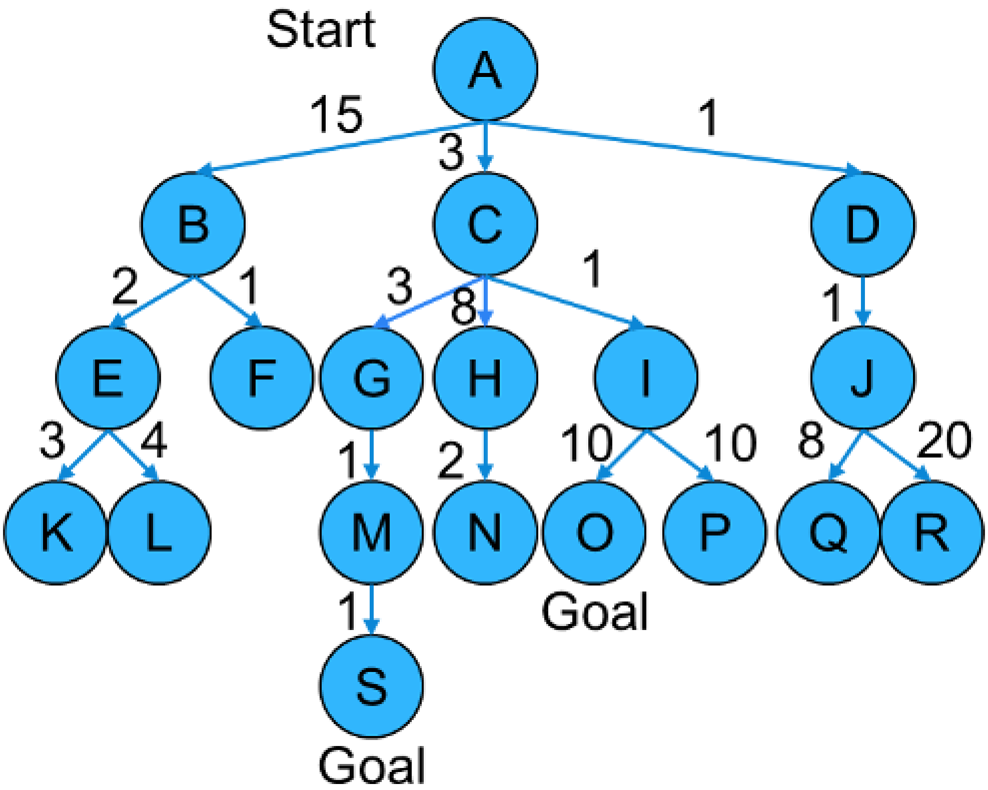
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Class exercise

- Consider using a breadth-first strategy on the 8-puzzle
- Will switching to depth-first search increase or decrease the size of the state space?

Least-cost search

- Treat the agenda as a **priority-ordered queue**:
  - nodes are ordered by ascending cost
  - (in the case of ties, we'll assume an arbitrary order for those that tie)
- Hence, the least-cost path is extended at every step
- This, in effect, is *Dijkstra's Algorithm*, that you met in previous modules
- We will illustrate in the lecture using this state space:



Evaluation

- Is least-cost search complete?
- Is least-cost search optimal?
- What is its time complexity?
- What is its space complexity?

Informed search

- In **informed search** (heuristic search, directed search), the agenda again is a **priority-ordered queue**
- But nodes are ordered by their ‘promise’, computed by an **evaluation function**
  - Perhaps counter-intuitively, the convention is that smaller number designate higher 'promise'
  - So the queue will be in ascedning order
- The evaluation function is typically a **heuristic** function, which *estimates* the cost of the cheapest path from the state to a goal state
  - Note that heuristic functions evaluate *states*, not actions
  - Note that heuristic functions are problem-specific

Heuristic function

- For the 8-tiles puzzle, e.g.
$$h_1(n) = \text{the number of tiles out of place in this state relative to the goal state}$$
- Example:
  - State being evaluated:

8	2	7
6	1	3
4		5

  - Goal state:

1	2	3
8		4
7	6	5

Heuristic function

- For the 8-tiles puzzle, e.g.
  $h_2(n) =$ the sum, for each tile, of the Manhattan distance between its position in this state and its
- Example:
  - State being evaluated:

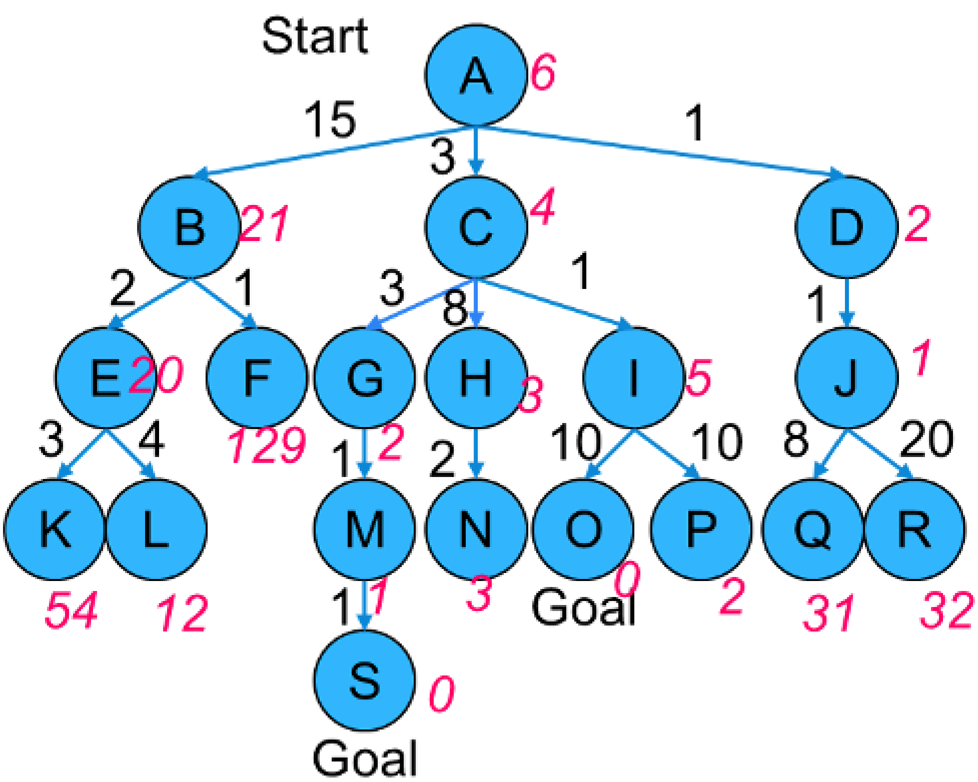
8	2	7
6	1	3
4		5

  - Goal state:

1	2	3
8		4
7	6	5

Greedy search

- Evaluation function consists only of heuristic function
- Hence, most promising node (according to heuristic) is always the one expanded next
- We will illustrate in the lecture using this state space:



Evaluation

- Is greedy search complete?
- Is greedy search optimal?
- What is its time complexity?
- What is its space complexity?

A^* search

- In A^* search,
  - the evaluation function consists of the path cost as well as the heuristic function:
    $$f(n) = g(n) + h(n)$$
  - furthermore, h must be an **admissible** heuristic:
    - one that *never over-estimates* the cost of the path to the nearest goal
- Class exercise: Was h_1 for the 8-tiles puzzle (see earlier) admissible? What about h_2?
- We will illustrate in the lecture using the same state space that we used for greedy search

(Advanced) Strictly speaking…

- One way to avoid re-exploration was:
  - Discard any successor if it is the same as any previously-generated node
- If you want to do something like this for A^* but you want A^* still to be optimal, then:
  - Discard either the successor or the previously-generated node — whichever has the higher path-cost
  - (Alternatively, discard the successor, as above, but preserve optimality by making sure your heuristic is not just admissible, but also *consistent*)
- Alternatively, don't worry about avoiding re-exploration! Maybe the cost of re-exploration is less than the cost of checking & discarding)

Evaluation

- Is A^* search complete?
- Is A^* search optimal?
- What is its time complexity?
- What is its space complexity?

Class exercise

- You are asked to compare two heuristic functions. What would cause you to prefer one over the other?

In [ ]:

```

```