HEURISTIC SEARCH I

Recap

The last lecture introduced

- Basic problem solving techniqus:
  - Breadth-first search
  - Depth-first search
- Breadth-first search is complete but expensive.
- Depth-first search is cheap but incomplete
- Can't we do better than this?
- That is what this lecture is about

Overview

Aims of this lecture:

- show how basic search (depth 1st, breadth 1st) can be improved;
- introduce:
  - *depth limited search*;
  - *iterative deepening*.
- show that even with such improvements, search is hopelessly unrealistic for real problems.

Algorithmic Improvements

- Are then any *algorithmic* improvements we can make to basic search algorithms that will improve overall performance?
- Try to get *optimality* and *completeness* of breadth 1st search with *space efficiency* of depth 1st.
- Not too much to be done about time complexity :-(

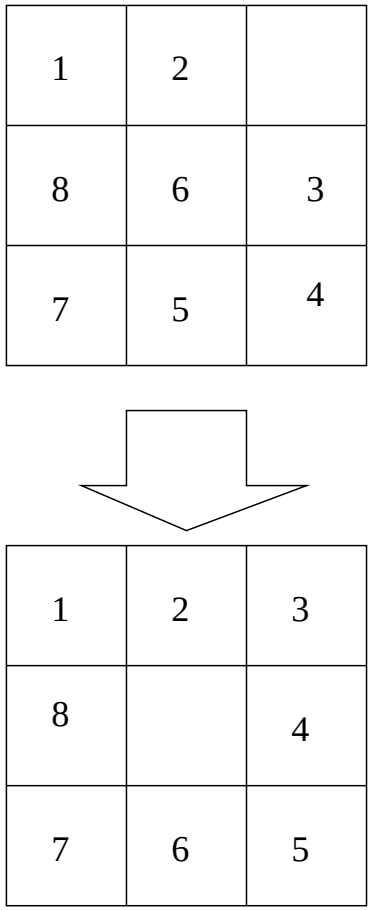
Depth Limited Search

- Depth first search has some desirable properties — space complexity.
- But if wrong branch expanded (with no solution on it), then it won't terminate.
- Idea: introduce a *depth limit* on branches to be expanded.
- Don't expand a branch below this depth.

- General algorithm for depth limited search:

```
depth limit = max depth to search to;
agenda = initial state;
while agenda not empty do
  take node from front of agenda;
  new nodes = apply operations to node;
  if goal state in new nodes then {
    return solution;
  }
  if depth(node) < depth limit then {
    add new nodes to front of agenda;
  }
}
```

- For the 8-puzzle setup as:



- . . . the search will be as follows:

Iterative Deepening

- Unfortunately, if we choose a max depth for d.l.s. such that shortest solution is longer, d.l.s. is not complete.
- Iterative deepning an ingenious *complete* version of it.
- Basic idea is:
  - do d.l.s. for depth 1; if solution found, return it;
  - otherwise do d.l.s. for depth n; if solution found, return it;
  - otherwise, . . .
- So we *repeat* d.l.s. for all depths until solution found.

- General algorithm for depth limited search:

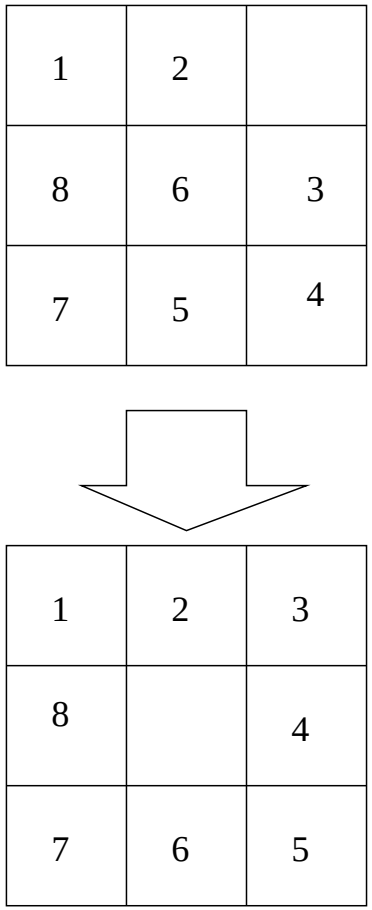
```
depth limit = 1;
repeat {
  result = depth_limited_search(
    max depth = depth limit;
    agenda = initial node;
  );
  if result contains goal then {
    return result;
  }
  depth limit = depth limit + 1;
} until false; /* i.e., forever */
```

- Calls d.l.s. as subroutine.

- Note that in iterative deepening, we *re-generate nodes on the fly.*
- Each time we do call on depth limited search for depth d, we need to regenerate the tree to depth $d - 1$.
- Isn't this inefficient?
- Tradeoff *time* for *memory*.
- In general we might take a *little* more time, but we save a *lot* of memory.

- Example: Suppose $b = 10$ and $d = 5$.
- Breadth first search would require:
  - examining $111,111$ nodes, and
  - memory requirement of $100,000$ nodes.
- Iterative deepening for same problem:
  - $123,456$ nodes to be searched, BUT
  - memory requirement only 50 nodes.
- Takes 11% longer in this case.

- For the 8-puzzle setup as:



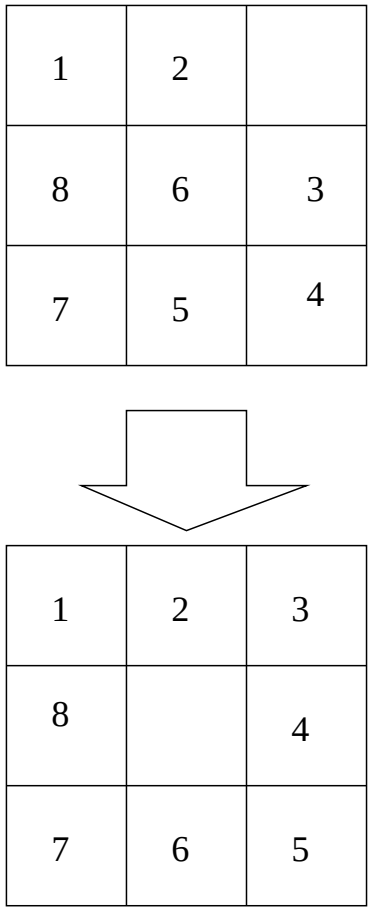
- . . . an iterative deepening search might be as follows:

Bi-directional Search

- Suppose we search from *the goal state backwards* as well as from *initial state forwards*.
- Involves determining *predecessor* nodes to goal, and then looking at predecessor nodes to this, ...
- Rather than doing one search of b^d, we do *two* $b^{d/2}$ searches.
- *Much* more efficient.

- Example:
  Suppose $b = 10$, $d = 6$.
  Breadth first search will examine nodes.
  Bidirectional search will examine nodes.
- Can combine different search strategies in different directions.
- For large d, is still impractical!

- For the 8-puzzle setup as:



- . . . a bi-directional search might be as follows:

Summary

- This lecture has looked at some more efficient techniques than breadth first and depth first search.
  - depth-limited search;
  - iterative-deepening search; and
  - bidirectional search.
- These all improve on depth-first and breadth-first search.
- However, all fail for big enough problems (too large state space).
- Next lecture, we will look at approaches that cut down the size of the state-space that is searched.