COMP2511

Object-Oriented Design and Programming

Basic Search Algorithms

Wayne Wobcke

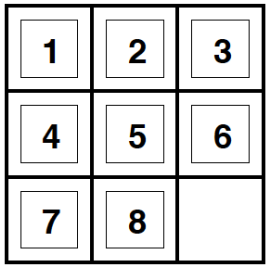
w.wobcke@unsw.edu.au

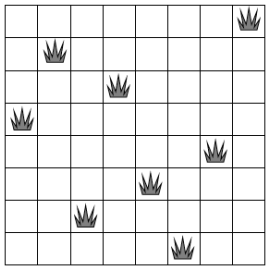
Today's Lecture

- Graphs and Problem Solving
- Basic Search Algorithms
  - Breadth-First
  - Depth-First
- Design Patterns
  - Iterator Pattern
  - Strategy Pattern

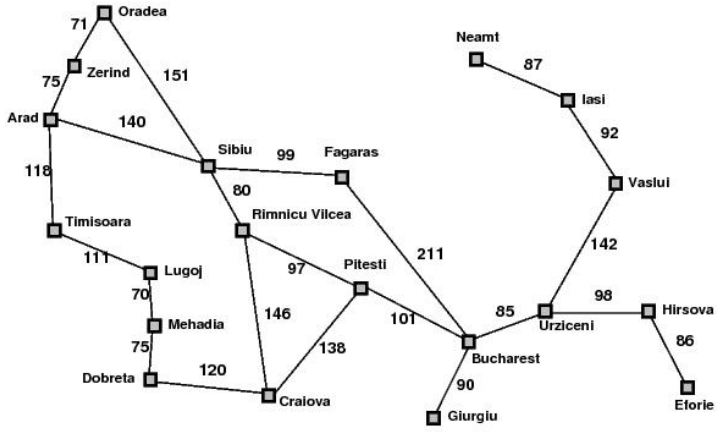
Graphs and Problem Solving

- State Space Search
  - Initial state, successor function, goal state

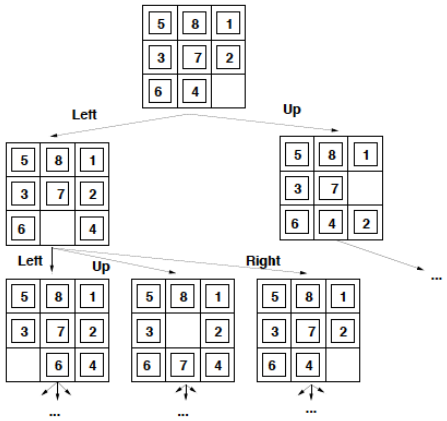




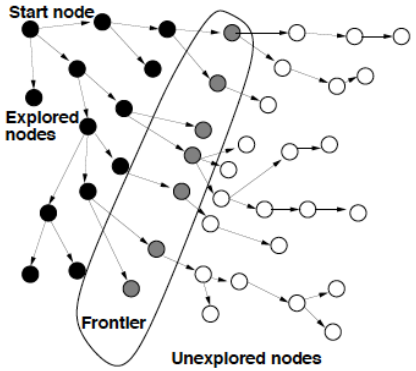
Romania Map



Search Tree: Nodes and States



General Search Procedure



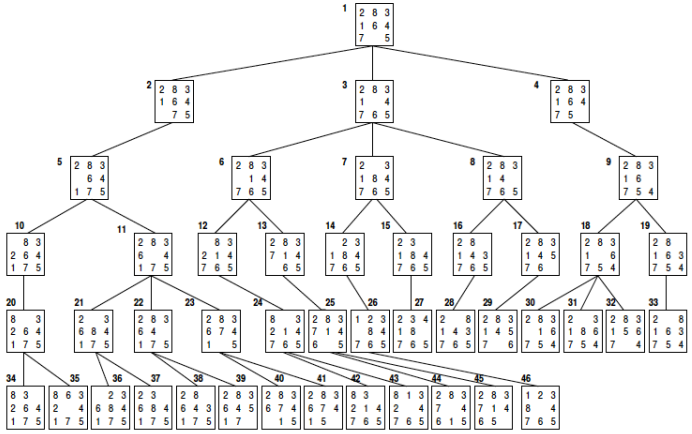
■ Search strategy = way frontier expands

Breadth-First Search

```
Queue<Node> frontier = new LinkedList<Node>();
ArrayList<Node> explored = new ArrayList<Node>();

while (frontier.peek() != null)
   Node current = frontier.poll();
   explored.add(current);
   successors = current.successors();     //sorted how?
   for (Node n : successors)
      if (n.goalState()) return success;     // stop search
      if (!explored.contains(n) && !frontier.contains(n))
         frontier.add(n);
return failure;
```


Search Tree

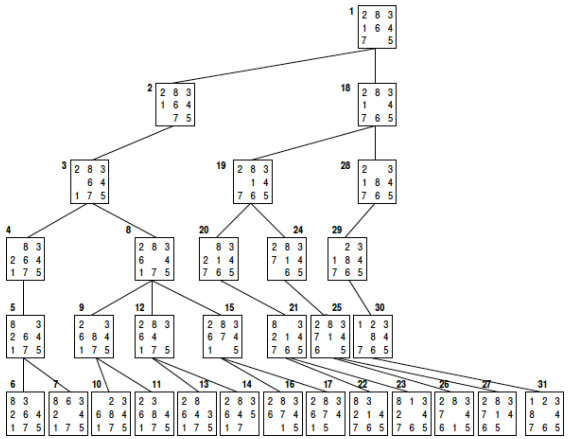


Depth-First Search

```
Stack<Node> stack = new Stack<Node>();    // one path on explored

public boolean dfs(Stack<Node> stack)
   if (!stack.empty())
      Node current = stack.peek();
      if (current.goalState()) return success;
      successors = current.successors();          //sorted how?
      for (Node n : successors)
         if (!stack.search(n) == -1)          // cycle check: n not on stack
            stack.push(n);
            if (dfs(stack)) return success
            stack.pop();                      // backtrack
   return failure;
```


Search Tree



Comparison

	Breadth-First	Depth-First
Time	b^d	b^m
Space	b^d	bm
Optimal	Yes	No
Complete	Yes	No

- b = branching factor
- d = depth of shallowest solution
- m = maxiumum depth of tree

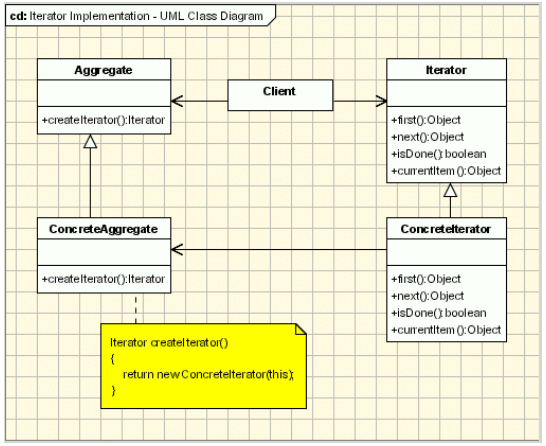
Design Patterns

- Web reference
  - Motivation: General problem scenario
  - Intent: Description of solution to problem
  - Implementation: UML class design
- Horstmann
  - Context: Typical problem scenario
  - Solution: Description and UML class design

Iterator Pattern

- Motivation
  - Need to traverse items of diverse sorts of collections (lists, arrays) in uniform manner
- Intent
  - Create an iterator object that maintains a reference to one item, and methods for advancing through the collection

Iterator Pattern Implementation



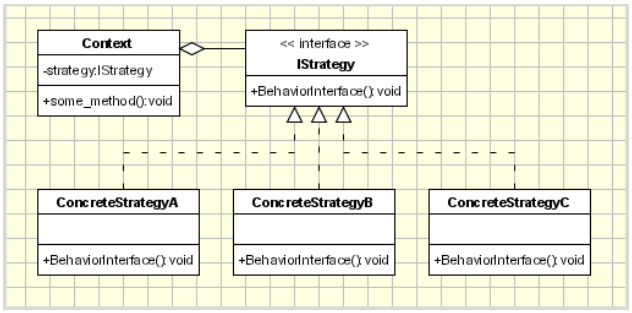
Strategy Pattern

- Motivation
  - Need a way to adapt the behaviour of an algorithm at runtime by varying one part of the procedure (giving algorithm variants)
- Intent
  - Define an interface that abstracts the varying functionality, have the client use that interface to supply a concrete variant

Strategy Pattern

- Examples
  - Sorting a list (vary orderings)
  - AI/Robot behaviour (vary game play)
  - Graph search (vary order of successors)

Strategy Pattern Implementation



Strategy Pattern in Search

```
successors = current.successors();

// sort using natural ordering on element type
Collections.sort(successors);

// or using Comparator – anonymous inner class declared as interface
Collections.sort(successors, new Comparator<Node>() {
   public int compare(Node n, Node o) {
      return o.getName().compareTo(n.getName());
   }
});
```


Next Week

- A* Search
- Assignment 2