Lecture 6

- Conclusion of KR
- Search
  - Blind search

Concept Description Language

- A specialized language for efficient inference.
- Represent
  - classes of objects,
  - sub-classes of classes,
  - instances of classes,
  - properties of instances (and classes).
- Akin to inheritance in object-oriented programming.

- A **semantic network** is a graph-based representation that addresses the same idea.

(See DAA pp. 107-109.)

Nonmonotonic Logic

A monotonic logic:

Things are are theorems remains theorems as we add additional axioms.

Non-monotonic logic: formulas that were once theorems may not remain so as the theory is augmented.

- Idea: add "default" assumptions to the theory in the absence of complete knowledge.
- These assumptions may be **retracted** later, as we learn more.
- As a result, conclusions based on those assumptions are invalidated.

Minimal (nonmonotonic) models

- Can induce a preference ordering on interpretations.

Deductive retrieval

- Deductive retrieval uses a KB to store information, and uses rules to achieve goals.
- A system for maintaining a knowledge base. Includes retracting conclusions based on information that changes or which is deleted.
- Use **forward chaining** and **backward chaining**.
  - Forward: start from KB and see what can be inferred (leading to the goal, we hope). Especially happens when a question involves new knowledge.
    - Q. if pigs fly then will I pass?
  - Backwards: we already saw **goal reduction.**
    - Start from the goal.
    - See what we need to infer it.

Search

- Reference: DAA Chapter 4.

- The process of explicitly examining a set of objects to find a particular one, or satisfy some goal.
- In the contact, the objects are typically possible configurations of some problem representation.

- Eg.
  - Towers of Hanoi problem.
  - Path planning
  - Theorem proving!
  - Game playing.

Search

- Search is a central topic in AI
  - Originated with Newell and Simon's work on problem solving.

    Famous book:

    ``Human Problem Solving'' (1972)

- Automated reasoning is a natural search task
  More recently: Given that almost all AI formalisms (planning, learning, etc.) are NP-Complete or worse,some from of search is generally **unavoidable** (no ``smarter'' alg. Available)

State space

- In AI, search usually refers to search of a state space.
- State space: the ensemble of possible configurations of the domain of interest.
  - Like phase space in physics
- E.g.
  - Chess: The set of allowed arrangements of pieces on a chess board.
  - Speech understanding: The set of possible arrangements of words that make valid sentences.
  - Robot motion planning: the set of accessible & safe locations for the robot.

Search: Problem Definition

State space -- described by an **initial state** and the set of possible actions available (**operators**).

- A path is any sequence of actions that lead from one state to another.

Goal test -- applicable to a single state to determine if it is a (the) goal state.

Path cost -- relevant if more than one path leads to the goal, and we want the shortest path.

- Note: very general formulation. Can be somewhat unnatural.

Graphs, Trees & Search

- We can visualize generic state space search in terms of searching a graph or tree.

- Graph search corresponds to looking for a particular state given an arbitrary transition diagram.
  - A graph is defined as G = (V, E)
  - V: set of vertices (i.e. states)
  - E : set of transitions $e_i = (v_j, v_k)$. Can be directed or undirected.

- Tree search results when there is a natural ordering, as with game playing.

Traversal

- To traverse means to visit the vertices in some systematic order. You should be familiar with various traversal methods for trees:

  preorder: visit each node before its children.
  postorder: visit each node after its children.
  inorder (for binary trees only): visit left subtree, node, right subtree.

Familiar ideas….

- **Blind search**: just examine "successive" alternative possibilities.
- Does not exploit knowledge of what states to examine first.
  - It what order should we consider the states?

- Sequentially along a path? Leads to **Depth First Search** (DFS)

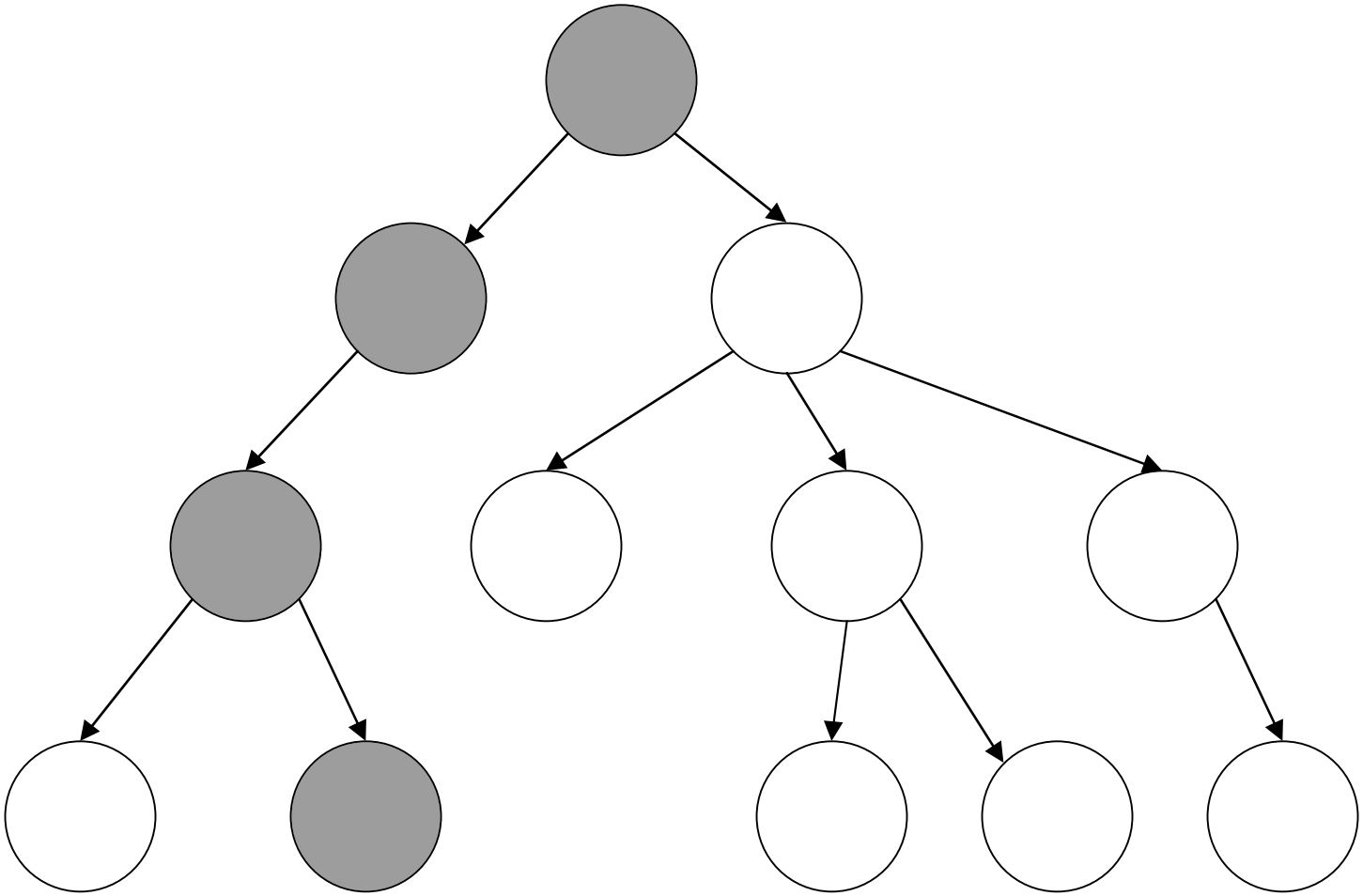
- A little bit along each possible path in turn? Leads to **Breadth First Search** (BFS).

Depth First Search

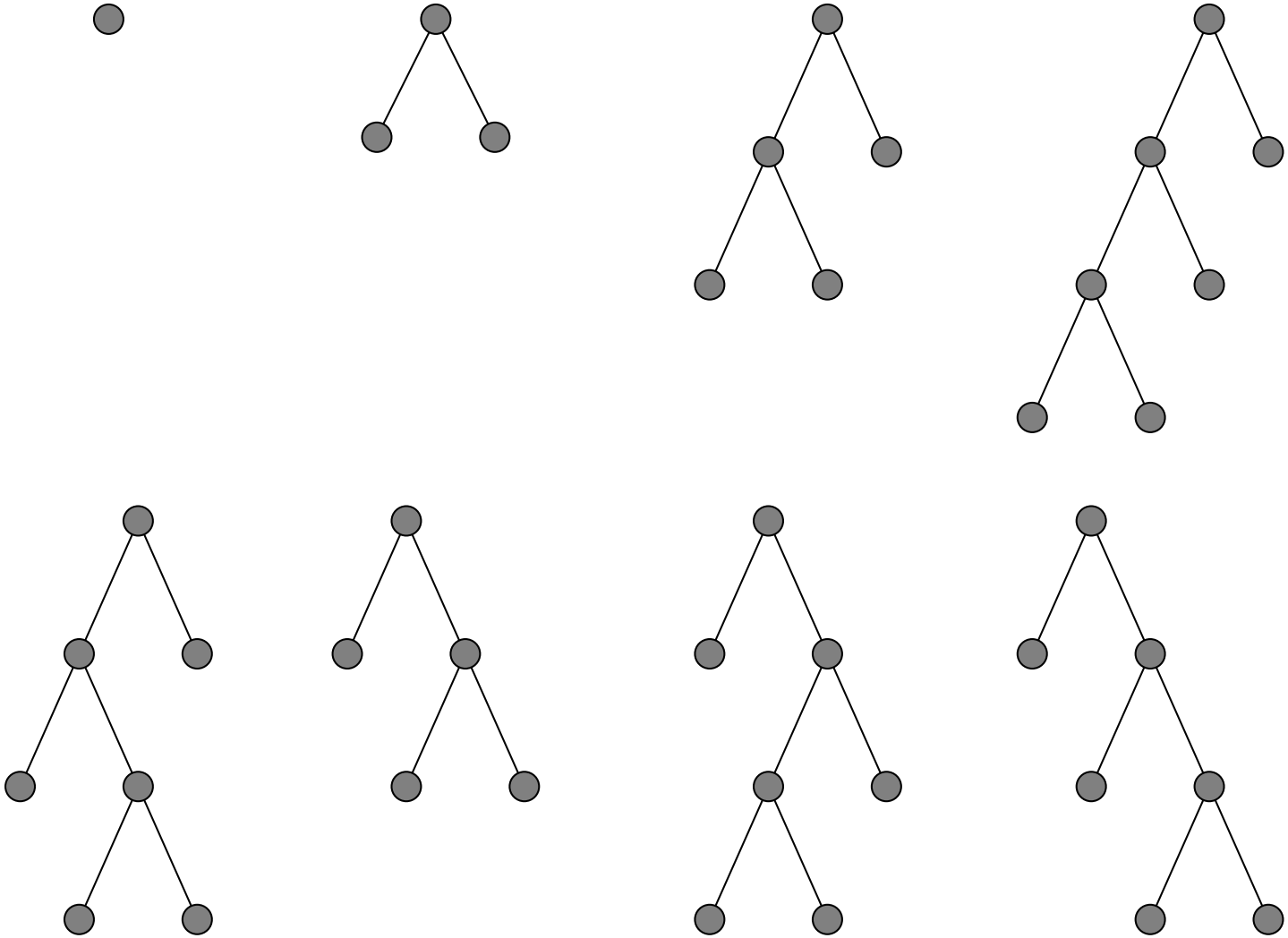
- Key idea: pursue a sequence of successive states *as long as possible.*

```
unmark all vertices
    choose some starting vertex x
    mark x
    list L = x
    tree T = x
    while L nonempty
        choose the vertex v from front of list
        visit v
        for each unmarked neighbor w
            mark w
            add it to front of list
            add edge (v,w) to T
```

Depth First Search



DFS illustrated

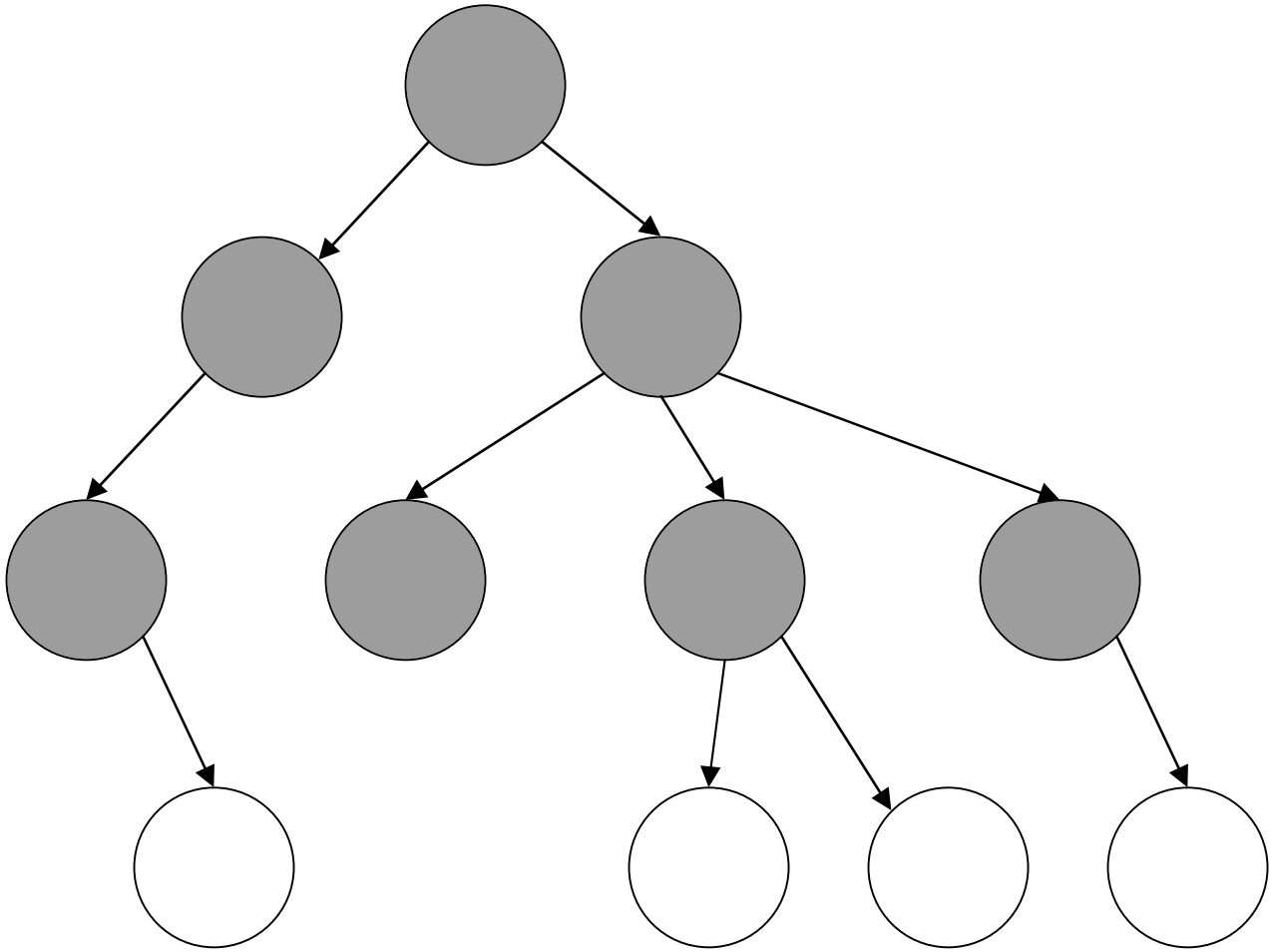


BFS

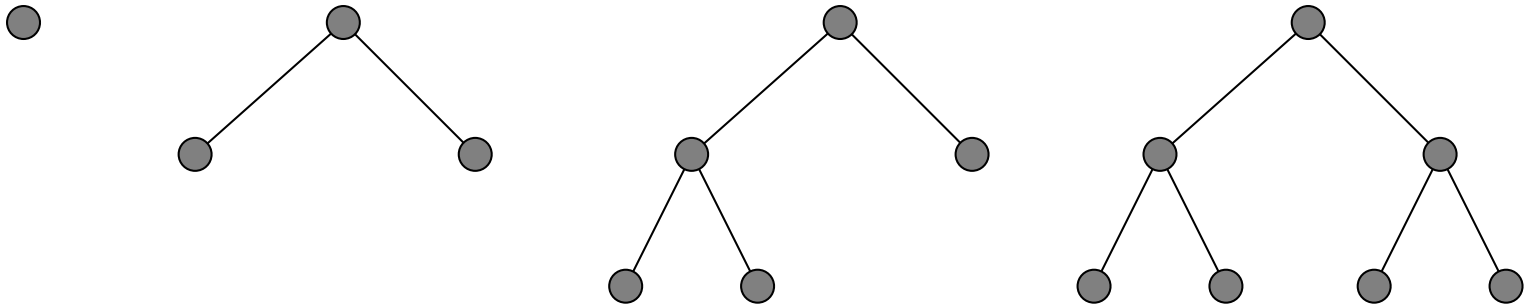
- Key: explore nodes at the same distance from the start at the same time

```
unmark all vertices
    choose some starting vertex x
    mark x
    list L = x
    tree T = x
    while L nonempty
        choose the vertex v from front of list
        visit v
        for each unmarked neighbor w
            mark w
            add it to end of list
            add edge (v,w) to T
```

Breadth First Search



BFS illustrated



Key issues in search

- Here's what to keep in mind.
- Completeness: are we assured to find a solution (if one exists)?
- Space complexity: how much storage do we need?
- Time complexity: how many operations do we need?
- Solution quality: how good is the solution

- Also…
  - Can we expect an interim solution?
  - Can we refine a partial/inadequate solution?
  - Can we cope with imperfect knowledge?

Example I : Cryptarithmetic

```
      SEND
+     MORE
--------
     MONEY
```

- Find substitution of digits for letters such
- that the resulting sum is arithmetically correct.
- Each letter must stand for a different digit.

Cryptarithmetic, cont.

- **States**: a (partial) assignment of digits to letters.
- **Operators**: the act of assigning digits to letters.
- **Goal test**: all letters have been assigned digits and sum is correct.
- **Path cost:** zero. All solutions are equally valid.

- Solution method?
- Depth first search:

Search performance

- Key issues that determine the nature of the problem are:
  - **Branching factor** of the search space: how many options do we have at any time?
    - Typically summarized by the worst-case braching factor, which may be quite pessimistic.
  - **Solution depth**: how long is the path to the first solution?

Example: knight's tour

- Tour executed by a chess knight to cover (touch) every square on a chess board.
  - Sub-problem: Find a path from one position to another.
- States: possible positions on the chess board.
- Operators: the ways a knight moves.
- Goal: the positions of the pawns.
- Path cost: the number of moves used to get to a position.

Knight's

- How large is the state space?
  - A knight has up to 8 moves per turn.
  - Each possible tour must be verified up to the end of the trip. For a board of width N, there are N*N squares.
    - Thus, each tour can be up to N*N states in length.
  - If the correct solutions is found last, we might consider up every wrong tour first.

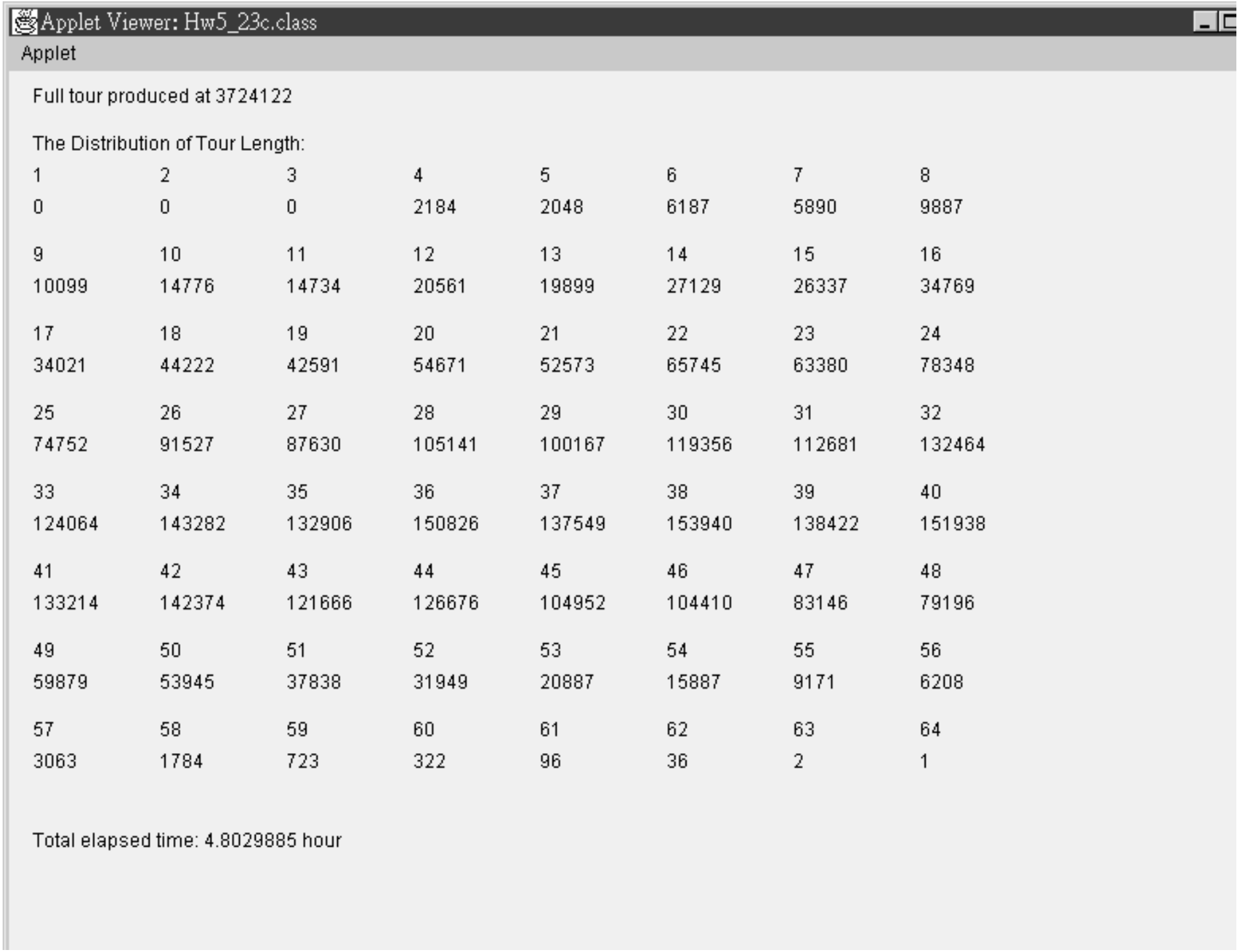
  $O(8^{N\wedge 2})$ states to examine!

What does this say about BFS? DFS?

Example: Knight's heuristics

- **Zero**: a board can be considered a dead end if any square has zero paths remaining to it. Any square with no paths to it would be unreachable, so no Knight's Tour would exist.
- **Two ones**: a board with more than one square with only one path to it is a dead end. A square with one path left is necessarily a dead end, so two of them indicate a dead end position.
- **Move to one**: the move finder should never choose a move to a square with one path left unless it is the last move; such a choice would otherwise lead to a dead end.

- How well will we do if we use “blind” DFS?
- That is, if we consider random tours?


Applet Viewer: Hw5_23c.class
Applet
Full tour produced at 3724122
The Distribution of Tour Length:
1 2 3 4 5 6 7 8
0 0 0 2184 2048 6187 5890 9887
9 10 11 12 13 14 15 16
10099 14776 14734 20561 19899 27129 26337 34769
17 18 19 20 21 22 23 24
34021 44222 42591 54671 52573 65745 63380 78348
25 26 27 28 29 30 31 32
74752 91527 87630 105141 100167 119356 112681 132464
33 34 35 36 37 38 39 40
124064 143282 132906 150826 137549 153940 138422 151938
41 42 43 44 45 46 47 48
133214 142374 121666 126676 104952 104410 83146 79196
49 50 51 52 53 54 55 56
59879 53945 37838 31949 20887 15887 9171 6208
57 58 59 60 61 62 63 64
3063 1784 723 322 96 36 2 1
Total elapsed time: 4.8029885 hour

- Three heuristics based on the number of paths remaining to each square were implemented and tested in combination, as well as a representational speedup. The optimal combination of the heuristics is to eliminate boards with either a square with zero paths remaining or a square with two ones remaining; this combination led to a 950-fold speedup on a 6x6 board. The representational speedup led to a 2.5-fold additional speedup on a 6x6 board.
  - Michael Bernstein

Heuristics: Knight's Performance

ZERO	TWO ONES	MOVE TO ONE	Time
0	0	0	9500 ms
1	0	0	415 ms
0	1	0	13.7 ms
0	0	1	6430 ms
0	1	1	13.2 ms
1	1	0	10.0 ms
1	0	1	350 ms
1	1	1	10.4 ms

BFS

- Consider a state space with a uniform branching factor of *b*
- At each level we have b nodes for every node we had before

  $1 + b + b^2 + b^3 + b^4 \ldots + b^d$

So, solution at depth d implies we must expand

$O(b^d)$ nodes!

- internal nodes (b**d-1)/(b-1)
- Leaf nodes: (b**d+1)/2
- why 2?

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes