Sliding Tile

Fall Semester

Topic Outline

- Scale
  - Larger problems
  - U.S. waterways
  - Fairfax County roads
  - Internet servers
  - Sliding tile puzzles
- Statement of the Problem
- Tractable Cases
- Memory
  - Original work
- Proof of Solvability
- Graphics in Python
  - Tkinter module
- Heuristics
  - Tiles out of place
  - Manhattan distance
  - Linear conflict
  - Pattern databases
  - General or specific?
- Up Next: Local Search

How does our code scale?

- We want to be able to solve larger and larger problems.
- Examples:

Problem	Nodes	Edges
Romania Roads	20	23
U.S. Waterways	6014	7199
Fairfax Co. Roads	40,670	53,216
Internet Servers	192,244	609,066

- Sliding tile puzzles are easy to scale:
  - No data files to read or neighbor list to build.
  - Each node must have either 2, 3, or 4 neighbors.
  - An $N \times N$ puzzle has $O\left(N^2\right)$ tiles but $O\left(\left(N^2\right)!\right)$ nodes.

Sliding Tile Puzzle (1)

- Goal state for 15-puzzle:

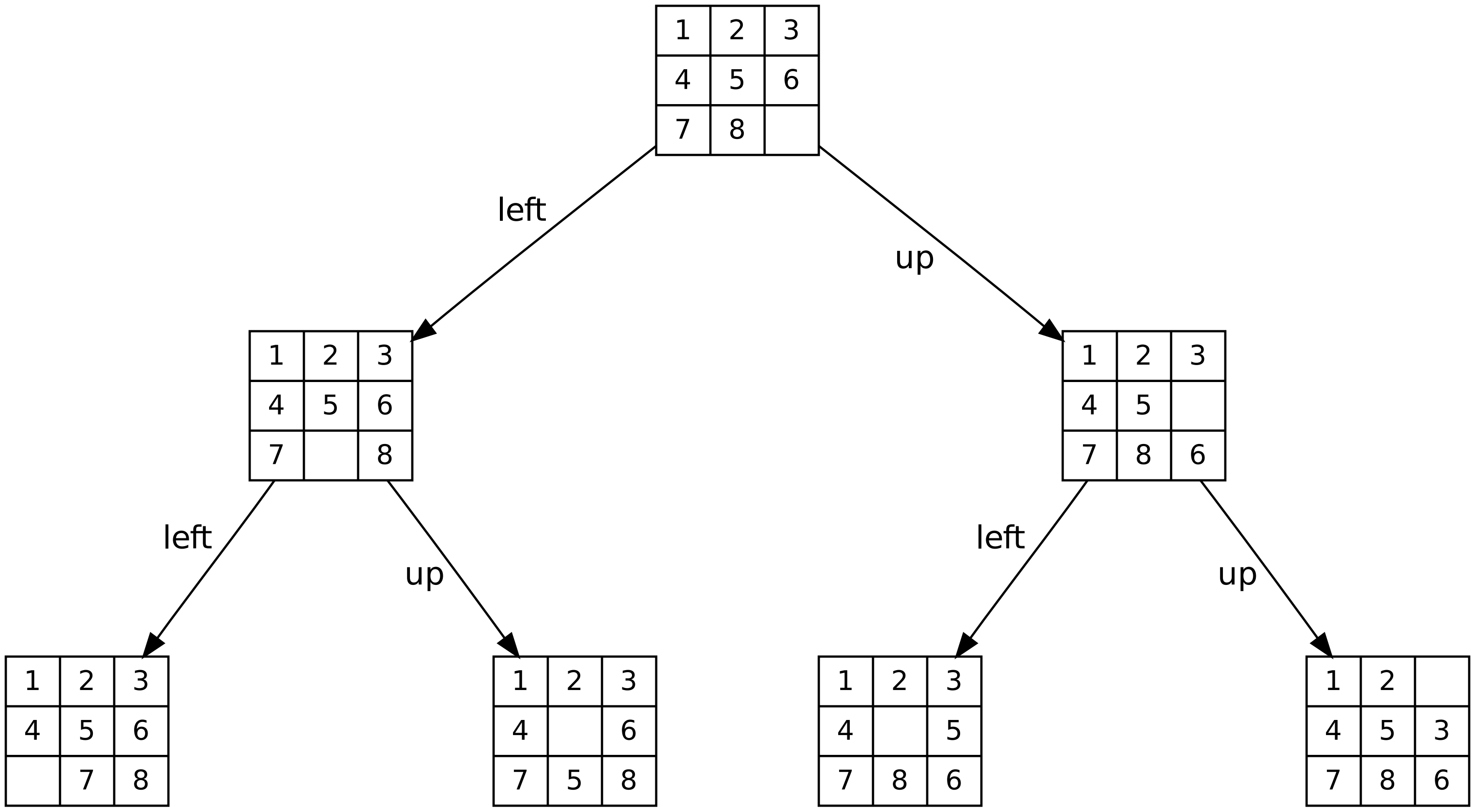
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- Initial state, for example:

5	1	2	3
9	10	6	4
13		7	8
14	15	11	12

- Half of all states are not connected by any path to the goal.
- Craze in 1880, cash prize for swapping two tiles. Impossible!

Sliding Tile Puzzle (2)



Statement of the Problem

- Input. The size of the puzzle N and the number of moves to make during the initial shuffling process.
- Generate an initial state:
  - Start with the goal state and randomly make valid moves.
  - Consider this shuffled puzzle as the initial state.
- Output. The intial state and a sequence of moves such that:
  - Each move is either **`up`**, **`down`**, **`left`**, or **`right`**.
  - We are led from the initial state to the goal state by applying these moves *in order* to the blank space.
  - The sequence is as short as possible. (Or does this matter?)

Testing

- Write a program that loads the initial state, applies the moves, and tests the end state against the goal. If it doesn't match, or if an illegal move is made along the way, something's wrong.
- This can be done graphically in Python using **`Tkinter`**.
- Do very little shuffling at first. If a code can't solve an easy problem then it's unlikely it would solve a harder problem.
- Stick to 8- and 15-puzzles early on. The 99-puzzle is intractable.

$N =$	3	4	5	$\cdots$	10
$\log_{10}\left((N^2)!/2\right) \approx$	5	13	24	$\cdots$	157

Path Storage Trick

- The nodes are boards. How will these be stored? One possibility is to use a string, for instance `goal='12345678X'`.
- We can keep track of each node's parent and then reconstruct the (backwards) path once the goal state is found.

Node	**Parent**
`12X453786`	`None`
`1X2453786`	`12X453786`
`12345X786`	`12X453786`
`12345678X`	`12345X786`

- This can even work for weighted graphs, calculate total path cost from individual edge costs stored in a double hash table.

Iterative Deepening A-Star

- But even then we could still run out of memory.
- These graphs are big!
- Recall DFSID solved the BFS memory problem.
- So, apply the same idea to a heuristic search.
- Turns out there are even better ideas than IDA*.
- Maybe in some cases we don't need the optimal path.
- Good area for original research.
- See also, Russell and Norvig page 101.

Parity Argument (1)

- Someone gives you a puzzle but can it be solved?

4	1	2
7		3
8	5	6

- Parity. Write the numbers down in row-major order and count up how many *bigger* values are *to the left* of each number.

4	1	2	7	X	3	8	5	6
0	1	1	0	-	2	0	2	2

- Ignore the blank space. The total is eight, an even number.

Parity Argument (2)

- Clearly the total is zero for the goal state. Zero is even.
- One can show:
  - Valid moves maintain the even-odd parity of this total.
  - Swapping two tiles does not maintain parity.
- Alternatively, start with the goal and swap two tiles. It will take an odd number of moves to unswap the tiles but an even number of moves to return the blank to its correct position. And the blank always moves! So the solution requires some number of moves that is both odd and even. Thus, there is no solution.
- Or, the mathematics of permutations.

Example Program: A Simple Tk Demo

- Frame and panel
  - root, canvas, pack, mainloop
- Event handling
  - bind
- Graphics objects
  - create, coords, itemcget, itemconfigure
- Animation
  - after

Heuristics (1)

- Number of tiles out of place.
- Admissibility. The tiles move one at a time and each requires at least one move to get to its correct location.
- Shown below $h = 6$.

1	2	3	4
5	6	7	8
11		12	9
13	15	10	14

Heuristics (2)

- Manhattan distance.
- Admissibility. Reduce the problem so that tiles can move independently. They still only gain one "city block" per move.
- Shown below $h = 3 + 2 + 2 + 1 + 2 + 1 = 11$.

1	2	3	4
5	6	7	8
11		12	9
13	15	10	14

Heuristics (3)

- Manhattan distance plus linear conflict.
- Admissibility. If two tiles are in the same correct row but must pass each other then add two moves for the out-and-around.
- Shown below $h = 3 + \mathbf{2} + 2 + 2 + 1 + 2 + \mathbf{2} + 1 = 15$.

1	2	3	4
5	6	7	8
11		12	9
13	15	10	14

Heuristics (4)

- Similar patterns recur in sliding tile puzzles.
- So, pre-compute an exact heuristic for those patterns.
- Then, identify the patterns and look up h in the database.
- Or, store entire move sequences (similar to how a human plays).
- Note. All these heuristics are problem specific.
- They only make sense in the context of sliding tile puzzles.
- Whereas the A-Star Search itself is completely general.
- There are generic heuristics for certain types of problems (CSP).

Lab Assignment: Sliding Tile

- Write a graphics program for sliding tile puzzles. Input the size and display a running update of the first three heuristics.
- The user should be able to click on a tile to swap it with the blank, if that's a legal move of course. Other possibilities include using arrow keys or even a solver that does the actual search.
- Fun alternatives to numbering the tiles:
  - Letters. The goal state spells a word. No profanity! Repeat letters are tricky. Be careful about which one goes where.
  - Break an image up into little pieces. Nothing obscene!
- Also, can you animate the tiles sliding into place?

End of Graph Problems

- The next topic is Local Search and will consider similar problems but with a totally different approach.
- This is a good time to make a note of possible project ideas related to graphs, for instance:
  - Data structures
  - Algorithms
  - Travelling Salesman Problem
  - Ant Colony Optimization
  - P versus NP
  - Memory-bounded search