

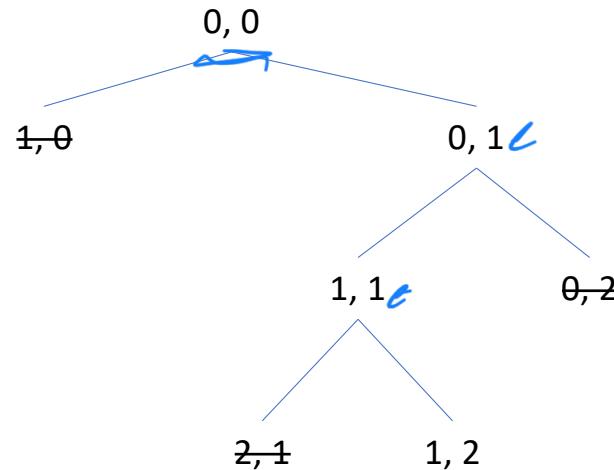
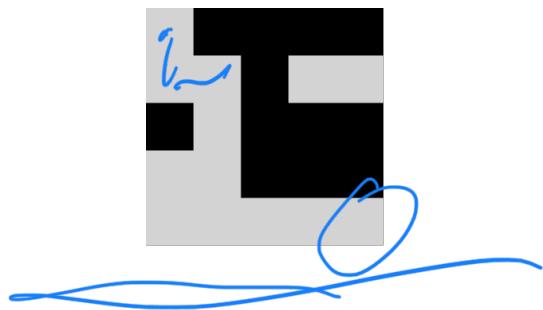
Search problems – day 2

- Last time – 2 way mazes (down and right only)
- Videos – Sudoku
- Problem bank – nqueens, programming contest
- Today – Triangle solitaire
- Lab - Clowns
- Problem set – TA lab schedule solver

- lot's of problems to practice with
 - work them from the starter
 - don't look at the solutions!!!

Domain analysis

Tree of x,y positions moving through this maze



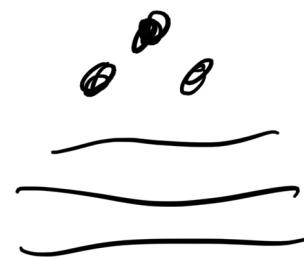
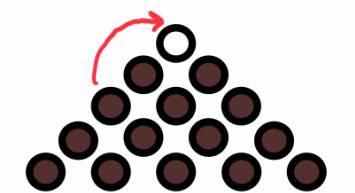
changing search state: current position

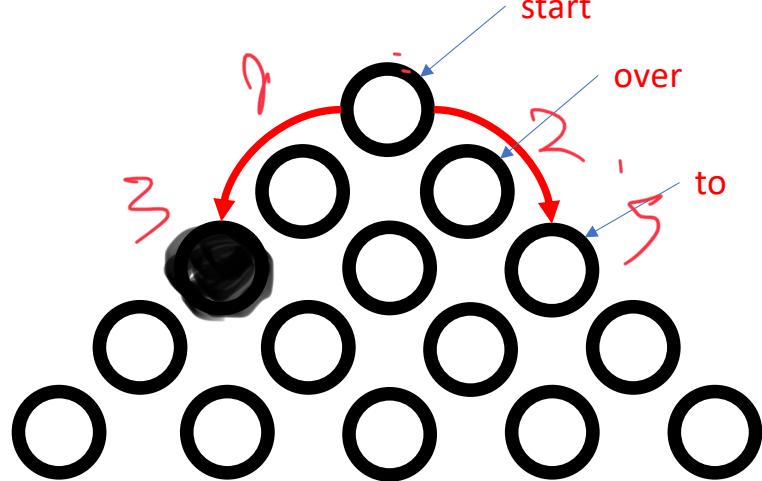
...

next search states: down and right, UNLESS wall or edges

done: solved when reach lower right corner
can also run out of moves

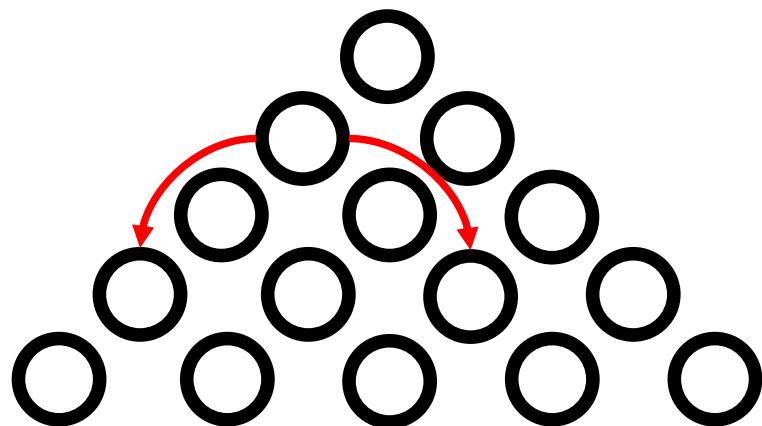
Peg solitaire



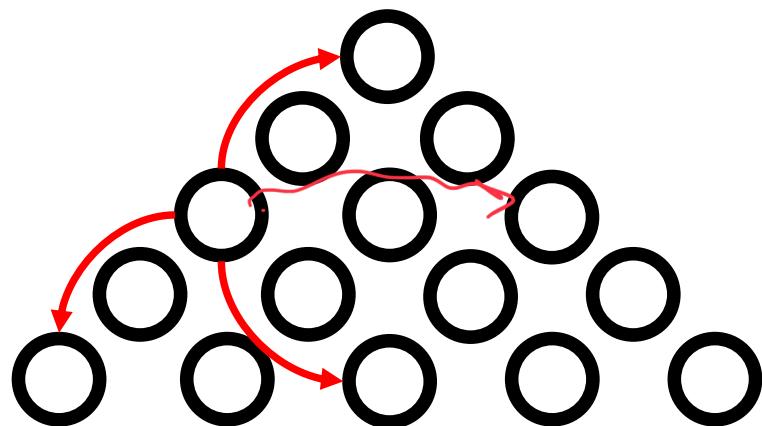


Jump has
start, over, to
To be valid,
start is full
over is full
to is empty

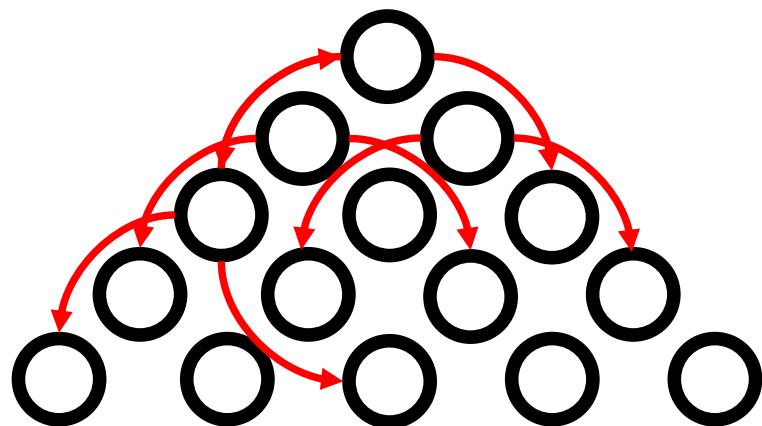
Jump has
start, over, to
To be valid,
start is full
over is full
to is empty



Jump has
start, over, to
To be valid,
start is full
over is full
to is empty

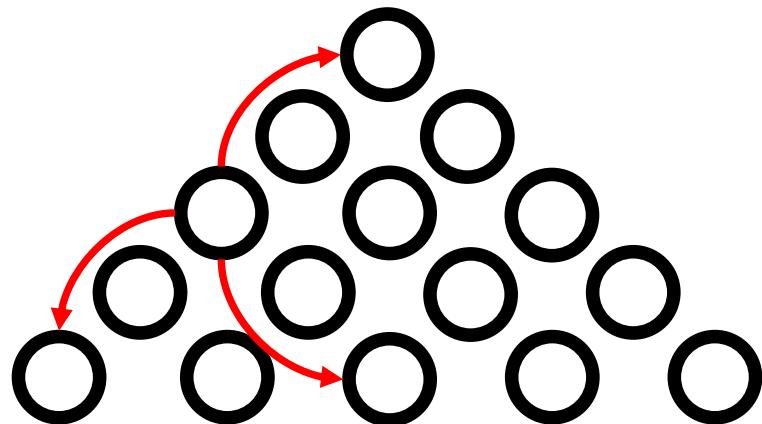


Jump has
start, over, to
To be valid,
start is full
over is full
to is empty



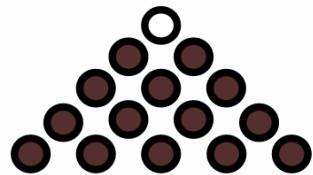
There are 36 possible jumps on the board

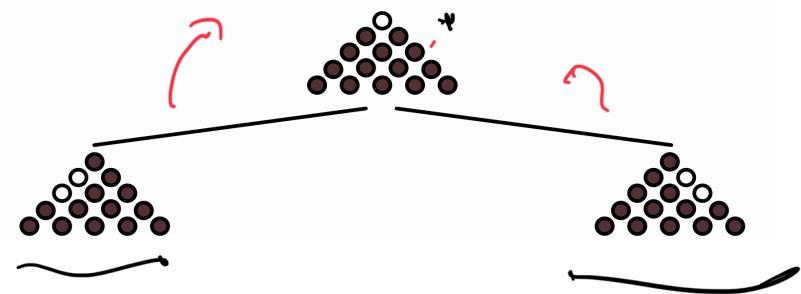
Jump has
start, over, to
To be valid,
start is full
over is full
to is empty



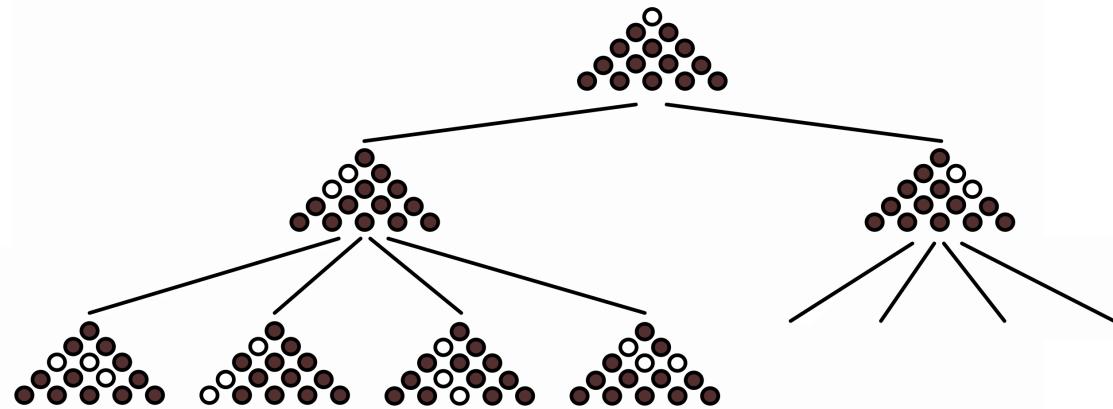
There are 36 possible jumps on the board

If we start here - what are possible next states?



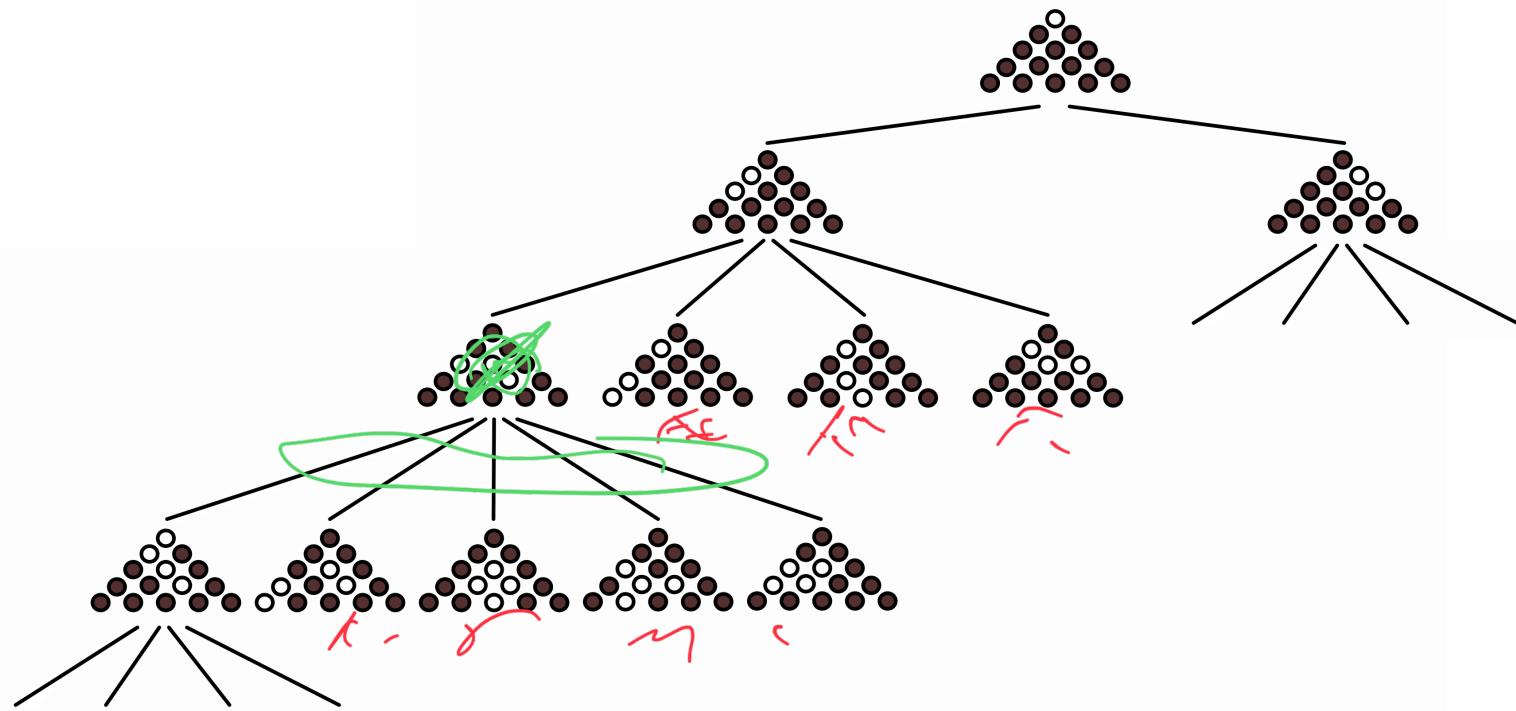


and after here?



and after here?

What is CHANGING???



Domain

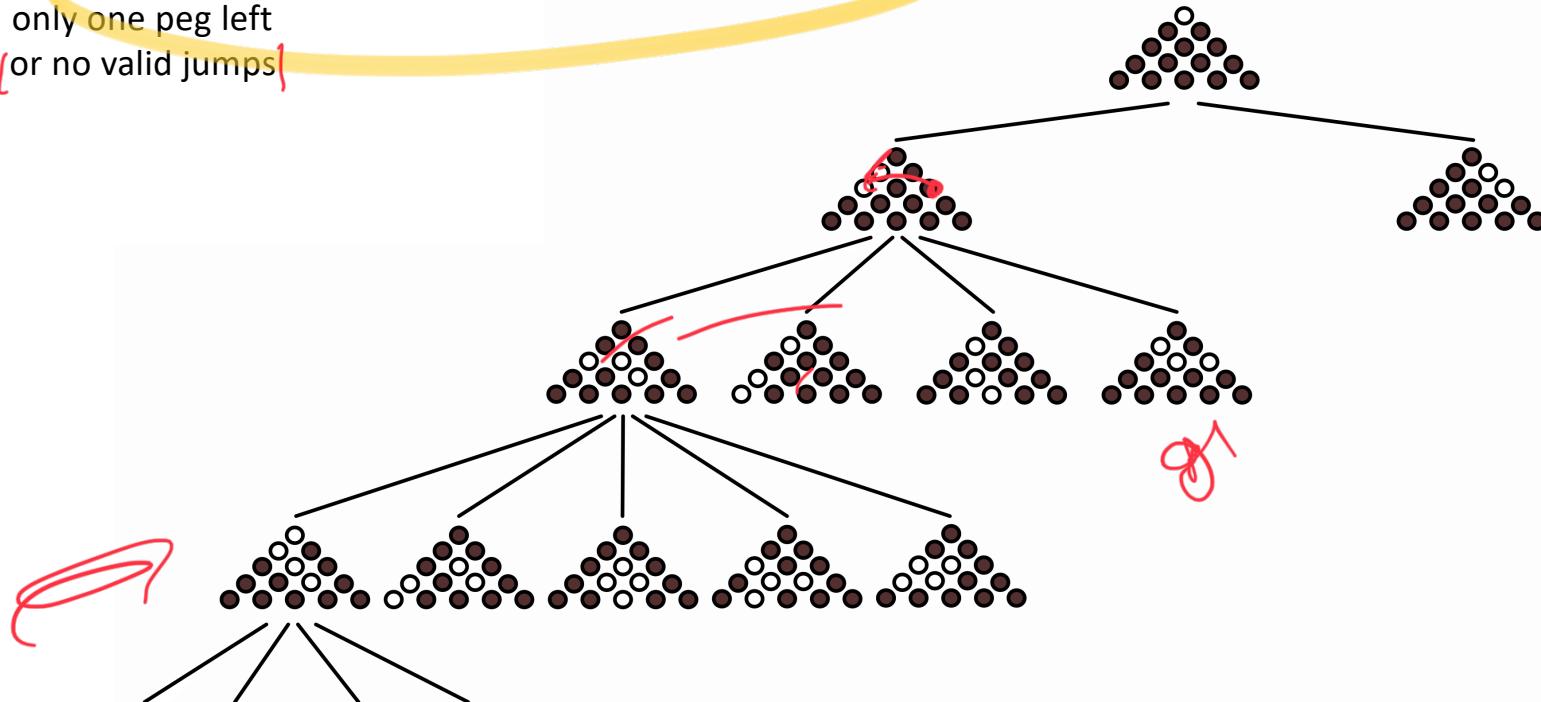
Analysis

changing: what holes have pegs in them

next states: 36 jumps \rightarrow n valid jumps \rightarrow n new boards

done: only one peg left

(or no valid jumps)



encapsulated generic advance try-catch

```
(define (solve bd)
  ;; Termination argument:
  ;; base: not possible to remove anymore pegs (solved or no valid jumps)
  ;; reduction: make one of n valid jumps
  ;; argument: making one jump at a time always reaches no more possible jumps
  (local [(define (solve-bd bd)
            (if (solved? bd)
                (list bd)
                (local [(define try (solve-lobd (next-boards bd)))]
                  (if (not (false? try))
                      (cons bd try)
                      false)))))

            (define (solve-lobd lobj)
              (cond [(empty? lobj) false]
                    [else
                     (local [(define try (solve-bd (first lobj)))]
                       (if (not (false? try))
                           try
                           (solve-lobd (rest lobj))))]))]
          (solve-bd bd)))
```

