Backtracking

Dr. Bart

Algorithms

Problem:
Generate all possible values from a pool

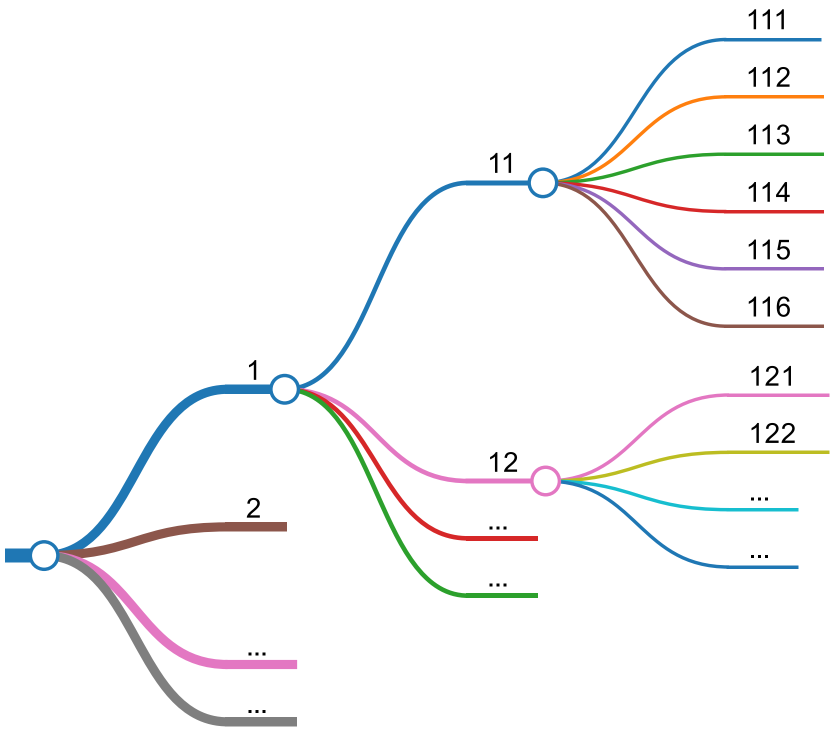
Example: All dice roll combinations

```
For (each possible first die value):
    For (each possible second die value):
        For (each possible third die value):
            …
                Print!
```

But what if we don't want to keep nesting for loops?

The **Search Space**

- Graph/Tree of possibilities
  - Internal vertices are partial candidates
  - Leaves are complete candidates
  - Edges represent adding a part
- Depth-First Search
  - Systematically construct
    all possible candidates



The Backtracking Algorithm

- Backtracking: Exhaustively find solutions by recursively building partial solutions and then abandoning them if they don't work

- Characteristics:
  - Brute force
  - Exhaustive
  - Recursive

What can we do with it?

- Produce all permutations/combinations of
  - A set
  - A string
  - …
- Parsing text (inefficiently)
- Solve games completely
- Logic programming

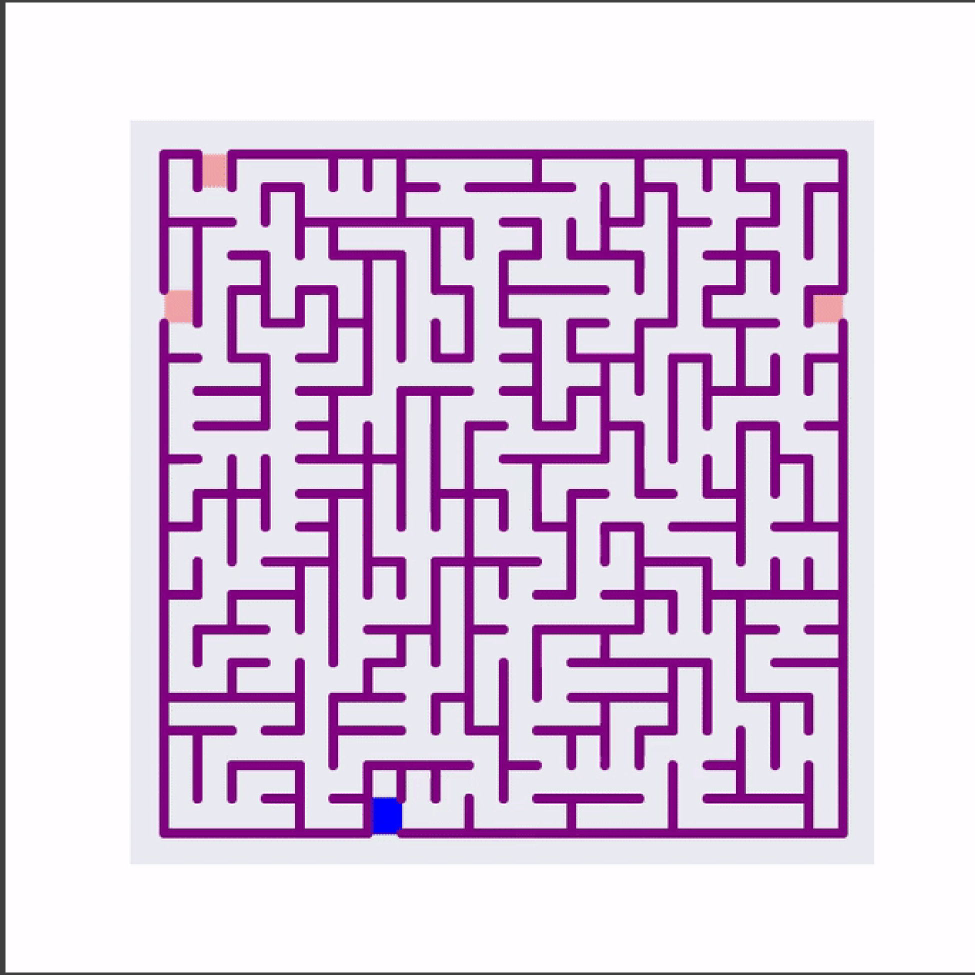
Solving Sudoku by trying all possible combinations

File:Sudoku solved by bactracking.gif. (2018, October 9). *Wikimedia Commons, the free media repository*. Retrieved 04:44, April 12, 2020 from https://commons.wikimedia.org/w/index.php?title=File:Sudoku_solved_by_bactracking.gif&oldid=323389836.

Exploring a maze

- Explore possible paths
  - If we hit the end,
    - We're done
  - Otherwise,
    - Try out each possible junction

- Orange: Internal
- Grey: Leaf



Algorithm and Psuedocode

- Explore(choices):
  - If there are no more choices
    - Then stop
  - Otherwise
    - Make a single choice C
    - Explore the remaining choices
    - Un-make choice C (backtrack)

```
def backtrack(candidate):
    if is_solution(candidate):
        process_solution(candidate)
    else:
        next_choices = make_candidates(candidate)
        for next_choice in next_choices:
            add_choice(candidate, next_choice)
            backtrack(candidate)
            remove_last(candidate)
```

Algorithm Implementation Questions

- ❑What does the solution look like?
- ❑What does a candidate look like?
- ❑How do we know if a candidate is a solution?
- ❑Do we stop when we find a solution?
- ❑What are the next choices at each vertex?
- ❑How do we add a choice to a candidate?
- ❑How do we "undo" a choice from a candidate?
- ❑How do we iterate through the choices?
- ❑How do we combine the choices solutions?

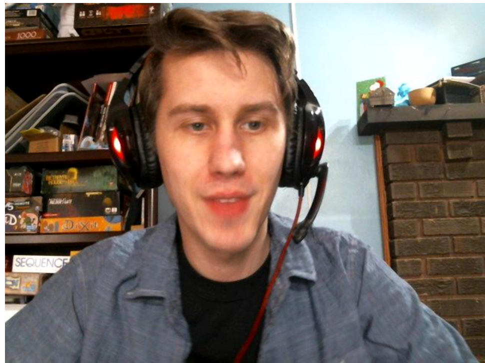
Pruning

- Sometimes we hit a dead-end internal vertex
    - If a partial candidate is clearly not going to lead to success,
    - We can stop pre-emptively

- Also known as "Branch and Bound"
    - Branching is the way we explore the DFS
    - Bounding is the way we stop at dead-ends

Time Analysis

- DFS takes V+E time
  - AKA its time complexity is linear on the vertices and edges, whichever is bigger.
- But the number of vertices is the number of partial candidates
  - So how many partial candidates are there?



Time Analysis: Rolling Dice

If we roll N 6-sided dice, then there are 6^N permutations

- So what if we rolled 5 dice?
  - $6^5 = 6 * 6 * 6 * 6 * 6$
  - $= 7776$
- What about 10 dice?
  - $6^{10} = 6*6*6*6*6*6*6*6*6*6$
  - $= 60466176$
- 20 dice?
  - Really big - quadrillion options!

Time Analysis: Drawing Cards

If we draw N cards from a 52-card deck, there are 52_n combinations

- So what if we draw 3 cards?
  - $52_3 = \frac{52!}{(52-3)!} = 52 * 51 * 50$
  - $= 132600$
- What about 6 cards?
  - $52_{10} = \frac{52!}{(52-6)!} = 52*51*50*49*48*47$
  - $= 14658134400$
- 20 cards?
  - Really, really big! Nonillion options!

Time Analysis: Conclusion

- Unfortunately, backtracking does not scale well
  - Brute force rarely works out well for us
- But if you have the computation power and time…
  - It's an easy way to get things done!