Lecture 7: Recursion with Backtracking

Adam Hartz
hz@mit.edu

30 March 2020
Recursion

In a general sense, *recursion* occurs when a thing is defined in terms of itself.

Example: For nonnegative integer $n$,

$$ n! = \begin{cases} 
  1 & \text{if } n = 0 \\
  n \times (n - 1)! & \text{otherwise} 
\end{cases} $$

To solve a problem recursively, we typically identify:

- One or more **base cases** (a terminating scenario that does not use recursion to produce an answer), and
- One or more **recursive cases** (a set of rules that reduce all other cases toward the base case).
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
Recursion vs Iteration?

Factorials can also be computed iteratively.

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```python
def factorial(n):
    out = 1
    for i in range(1, n+1):
        out *= i
    return out
```

Which would you choose? Why?
Recursion vs Iteration?

Factorials can also be computed iteratively.

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```python
def factorial(n):
    out = 1
    for i in range(1, n+1):
        out *= i
    return out
```

Which would you choose? Why?

Do we even need recursion?
Example Problem: Mixtape

Given a dictionary mapping song titles to durations, find a set of songs whose durations sum to a desired duration. If no such set exists, return None.
Recursive Backtracking Search: General Outline

- For each possible ”move” at the next position:
  - if move is legal: recursively search for solution to new problem incorporating the consequences of this move
    ★ if succeeds: return result from subproblem, incorporating this move
    ★ if fails: try next move
- if all moves exhausted with no success: return value indicating failure