Dynamic Programming 101

- DP = recursion (divide-n-conquer) + caching (overlapping subproblems)

- the simplest example is Fibonacci

\[ f(n) = f(n-1) + f(n-2) \]
\[ f(1) = f(2) = 1 \]

```python
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)
```

```python
def fib1(n):
    if n not in fibs:
        fibs[n] = fib1(n-1) + fib1(n-2)
    return fibs[n]
```

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)
```

- naive recursion without memoization: \(O(1.618...n)\)

- DP1: top-down with memoization: \(O(n)\)

```
def fib0(n):
    a, b = 1, 1
    for i in range(3, n+1):
        a, b = a+b, a
    return a
```

```
def fib0(n):
    f = [1, 1]
    for i in range(3, n+1):
        f.append(f[-1]+f[-2])
    return f[-1]
```

- DP2: bottom-up: \(O(n)\)

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)
```

```
def fib1(n):
    if n not in fibs:
        fibs[n] = fib1(n-1) + fib1(n-2)
    return fibs[n]
```

```
def fib0(n):
    a, b = 1, 1
    for i in range(3, n+1):
        a, b = a+b, a
    return a
```

```
def fib0(n):
    f = [1, 1]
    for i in range(3, n+1):
        f.append(f[-1]+f[-2])
    return f[-1]
```
Number of Bitstrings

- number of \( n \)-bit strings that do not have 00 as a substring
  - e.g. \( n=1 \): 0, 1; \( n=2 \): 01, 10, 11; \( n=3 \): 010, 011, 101, 110, 111
  - what about \( n=0 \)?
  - last bit “1” followed by \( f(n-1) \) substrings
  - last two bits “01” followed by \( f(n-2) \) substrings

\[
f(n) = f(n - 1) + f(n - 2)
\]

\( f(1) = 2, f(0) = 1 \)
Max Independent Set (MIS)

- max weighted independent set on a linear-chain graph
- e.g. \[9 - 10 - 8 - 5 - 2 - 4\]; best MIS: \([9, 8, 4]\) = 21 (vs. greedy: \([10, 5, 4]\) = 19)
- subproblem: \(f(n)\) -- max independent set for \(a[1]..a[n]\) (1-based index)

\[
f(n) = \max\{f(n-1), f(n-2) + a[n]\}
\]

\(f(0) = 0; f(1) = a[1]? \text{ No! } f(1) = \max(a[1], 0)\)

or even better: \(f(0) = 0; f(-1) = 0\)

sum\_by\_operator (across divides) combination\_operator (within a divide)

bitstrings

\[
f(n) = \begin{cases} 
  f(n-1) \times 1 + f(n-2) \times 1 & \\
  \oplus & \text{summary operator} \\
  \otimes & \text{combination operator}
\end{cases}
\]

recursive backtracking to backtrack the optimal solution
Summary

- Divide-and-Conquer = divide + conquer + combine
- Dynamic Programming = multiple divides + memoized conquer + summarized combine
- two implementation styles
  - 1. recursive top-down + memoization
  - 2. bottom-up
- backtracking to recover best solution for optimization problems
  - 1. backpointers (recommended); 2. store subsolutions (not recommended — often slows down); 3. recompute on-the-fly
- two operators: ⊕ for summary (across multiple divides) and ⊗ for combine (within a divide)
- counting problems vs. optimization problems (“cost-reward model”)
- three steps in solving a DP problem
  - define the subproblem
  - recursive formula
  - base cases

\[ f(n) = \max \left\{ \begin{array}{ll} f(n-1) + 0 & \text{summary operator } \oplus \text{ (across divides)} \\ f(n-2) + a[n] & \text{combination operator } \otimes \text{ (within a divide)} \end{array} \right\} \]
Deeper Understanding of DP

- divide-n-conquer
  - single divide, independent conquer, combine
- DP = divide-n-conquer with multiple divides
  - for each possible divide
    - divide
    - conquer with memoization
    - combine subsolutions using the combination operator $\boxtimes$
  - summarize over all possible divides using the summary operator $\boxplus$
- multiple divides $\Rightarrow$ overlapping subproblems
- each single divide $\Rightarrow$ independent subproblems!

\[
B(n) = \bigoplus_{i=1}^{n} \left( B(i-1) \boxtimes B(n-i) \right)
\]

\[
B(0) = 1
\]
### Unary vs. Binary Divides

\[(a) : T(n) = 2T(n/2) + \ldots \quad (b) : T(n) = T(n - 1) + \ldots \quad (c) : T(n) = T(n/2) + \ldots \]

<table>
<thead>
<tr>
<th>Divide-n-Conquer</th>
<th>Branching (Binary Divide)</th>
<th>One-Sided (Unary Divide)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort, best-case</td>
<td>quicksort, worst-case ((b))</td>
<td></td>
</tr>
<tr>
<td>Mergesort</td>
<td>quickselect: worst ((b)), best ((c))</td>
<td></td>
</tr>
<tr>
<td>(Balanced) tree traversal (DFS)</td>
<td>binary search: ((c))</td>
<td></td>
</tr>
<tr>
<td>Heapify (top-down)</td>
<td>search in BST: worst ((b)), best ((c))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DP</th>
<th># of BSTs (hw5), midterm</th>
<th>Fib, # of bitstrings (hw5)…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal BST, final</td>
<td>max indep. set (hw5)</td>
<td></td>
</tr>
<tr>
<td>RNA folding (hw10)</td>
<td>knapsack (hw6), midterm</td>
<td></td>
</tr>
<tr>
<td>Context-free parsing</td>
<td>Viterbi (hw8), final</td>
<td></td>
</tr>
<tr>
<td>matrix-chain multiplication, …</td>
<td>LCS, LIS, edit-distance,…</td>
<td></td>
</tr>
</tbody>
</table>

\(T(n) = 2T(n/2) + \ldots\)
## Two Divides vs. Multiple Divides (# of Choices)

<table>
<thead>
<tr>
<th>DP</th>
<th>two divides</th>
<th>multiple divides</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fib, # of bitstrings (hw5)…</td>
<td># of BSTs (hw5)</td>
</tr>
<tr>
<td></td>
<td>max indep. set (hw5)</td>
<td>unbounded knapsack (hw6)</td>
</tr>
<tr>
<td></td>
<td>0-1 knapsack (hw6)</td>
<td>bounded knapsack (hw6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Viterbi (hw8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RNA folding (hw10)</td>
</tr>
</tbody>
</table>