Dynamic Programming 101

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

- the simplest example is Fibonacci

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

DP1: top-down with memoization:
\[ O(n) \]
\[
\begin{align*}
\text{def } \text{fib}(n):
& \quad \text{if } n \leq 2: \\
& \quad \quad \text{return } 1 \\
& \quad \text{return } \text{fib}(n-1) + \text{fib}(n-2)
\end{align*}
\]

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

\[
\begin{align*}
\text{def } \text{fib0}(n):
& \quad a, b = 1, 1 \\
& \quad \text{for } i \text{ in range}(3, n+1):
& \quad \quad a, b = a+b, a \\
& \quad \text{return } a
\end{align*}
\]

naive recursion without memoization:
\[ O(1.618...n) \]
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. $9 \rightarrow 10 \rightarrow 8 \rightarrow 5 \rightarrow 2 \rightarrow 4$; 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)
  \[
  f(n) = \max\{f(n-1), f(n-2) + a[n]\}
  \]
  \[
  f(1) = 1 \\
  f(0) = 0 \\
  f(-1) = 0
  \]
  
  MIS
  \[
  f(n) = \max\left\{\begin{array}{ll}
  f(n-1) & \text{cost}
  \\
  f(n-2) + a[n] & \text{reward}
  \end{array}\right.
  \]

  bitstrings
  \[
  f(n) = + \left\{\begin{array}{ll}
  f(n-1) \times 1 & \text{summary operator } \oplus \\
  f(n-2) \times 1 & \text{combination operator } \otimes
  \end{array}\right.
  \]

  recursively 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: \( \oplus \) for summary (across multiple divides) and \( \otimes \) 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

\[
f(n) = \max \begin{cases} f(n-1) + 0 \\ f(n-2) + a[n] \end{cases}
\]
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 $\otimes$
  - summarize over all possible divides using the summary operator $\oplus$
- multiple divides $\Rightarrow$ overlapping subproblems
- each single divide $\Rightarrow$ independent subproblems!

\[
B(n) = \bigoplus_{i=1}^{n} \left( B(i - 1) \otimes B(n - i) \right) \\
B(0) = 1
\]
## Unary vs. Binary Divides

(a) \( T(n) = 2T(n/2) + \ldots \)  
(b) \( T(n) = T(n - 1) + \ldots \)  
(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></td>
<td>quicksort, best-case</td>
<td>quicksort, worst-case</td>
</tr>
<tr>
<td></td>
<td>mergesort</td>
<td>quickselect: worst</td>
</tr>
<tr>
<td>(balanced) tree traversal (DFS)</td>
<td></td>
<td>(b), best (c)</td>
</tr>
<tr>
<td></td>
<td>heapify (top-down)</td>
<td>binary search: (c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>search in BST: worst</td>
</tr>
<tr>
<td></td>
<td># of BSTs (hw5), midterm</td>
<td>Fib, # of bitstrings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(hw5)…</td>
</tr>
<tr>
<td>DP</td>
<td>optimal BST, final</td>
<td>max indep. set (hw5)</td>
</tr>
<tr>
<td></td>
<td>RNA folding (hw10)</td>
<td>knapsack (hw6), midterm</td>
</tr>
<tr>
<td></td>
<td>context-free parsing</td>
<td>Viterbi (hw8), final</td>
</tr>
<tr>
<td></td>
<td>matrix-chain multiplication, …</td>
<td>LCS, LIS, edit-distance, …</td>
</tr>
</tbody>
</table>
## 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>
Viterbi Algorithm for DAGs

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each incoming edge $(u, v)$ in $E$
   - use $d(u)$ to update $d(v)$: $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: $d(u)$ is fixed to optimal at this time

- time complexity: $O(V + E)$
Variant 1: forward-update

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each outgoing edge \((v, u)\) in \( E \)
   - use \( d(v) \) to update \( d(u) \):
     \[
     d(u) \oplus = d(v) \otimes w(v, u)
     \]
   - key observation: \( d(v) \) is fixed to optimal at this time

- time complexity: \( O(V + E) \)
Variant 2: Recursive Descent

- Top-down Recursion + Memoization = Bottom-up
- Start from the target vertex, going backwards
  - remember each visited vertex
- Sometimes easier to implement
- There is a tradeoff b/w top-down and bottom-up
## One-way vs. Two-way Divides (Graph vs. Hypergraph)

<table>
<thead>
<tr>
<th></th>
<th>two-way (binary divide)</th>
<th>one-way (unary divide)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>divide-n-conquer</strong></td>
<td>quicksort, best-case</td>
<td>quicksort, worst-case</td>
</tr>
<tr>
<td></td>
<td>mergesort</td>
<td>quickselect</td>
</tr>
<tr>
<td></td>
<td>tree traversal (DFS)</td>
<td>binary search</td>
</tr>
<tr>
<td></td>
<td>heapify (top-down)</td>
<td>search in BST</td>
</tr>
<tr>
<td><strong>DP</strong></td>
<td># of BSTs (hw5)</td>
<td>Fib, # of bitstrings (hw5)…</td>
</tr>
<tr>
<td></td>
<td>optimal BST</td>
<td>max indep. set (hw5)</td>
</tr>
<tr>
<td></td>
<td>RNA folding (hw10)</td>
<td>knapsack (all kinds, hw6)</td>
</tr>
<tr>
<td></td>
<td>context-free parsing</td>
<td>Viterbi (hw8)</td>
</tr>
<tr>
<td></td>
<td>matrix-chain multiplication, …</td>
<td>LCS, LIS, edit-distance,…</td>
</tr>
</tbody>
</table>
Viterbi Algorithm for DAGs

1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each incoming edge \((u, v)\) in \(E\)
   - use \(d(u)\) to update \(d(v)\): \(d(v) \oplus = d(u) \otimes w(u, v)\)
   - key observation: \(d(u)\) is fixed to optimal at this time

\[
\begin{align*}
  u & \quad w(u, v) \\
  \downarrow & \quad \downarrow \\
  v & \quad \uparrow \\
  \downarrow & \quad \downarrow \\
  \text{v} & \quad \text{w(u, v)}
\end{align*}
\]

- time complexity: \(O(V + E)\)
Viterbi Algorithm for DAHs

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming hyperedge \( e = ((u_1, \ldots, u_{|e|}), v, w(e)) \)
   - use \( d(u_i) \)'s to update \( d(v) \)
   - key observation: \( d(u_i) \)'s are fixed to optimal at this time

\[
d(v) \oplus = d(u_1) \otimes d(u_2) \otimes w(e)
\]

- time complexity: \( O(V + E) \) (assuming constant arity)
Example: RNA Folding and CKY Parsing

- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering
- Nussinov algorithm in RNA is almost identical to CKY but w/o overcounting

all $O(n^3)$
RNA Folding Example

Nussinov Algorithm — Traceback Example

k-best Viterbi on Graph

- simple extension of Viterbi to solve k-best on graphs and hyper graphs

cf. teams problem in HW4

for each node v,
- compute its kbest distances
- from the kbest of each incoming node u

1-best: $O(E + V)$

k-best: $O(E + V k \log d_{\text{max}})$ where $d_{\text{max}}$ is the max in-degree

can improve it to: (cf. midterm & teams, w/ quickselect)
k-best: $O(E + V k \log k)$ (assume $k \ll d_{\text{max}}$)
("most states do not have anybody on team USA")
k-best Viterbi on Hypergraph

- simple extension of Viterbi to solve k-best on graphs and hyper graphs

\[ \text{opt}[i, i] = \begin{cases} \text{opt}[i, j-1] & \text{if } i < j \\ (\text{opt}[i, p-1] \otimes \text{opt}[p+1, j-1] \otimes 1) & \text{if } i = j \\ 1 & \text{if } i = i-1 \end{cases} \]

\[ \text{opt}[i, j] = \oplus_{i \leq p < j} (\text{opt}[i, p-1] \otimes \text{opt}[p+1, j-1] \otimes 1) \]

\[ \text{opt}[i, i] = \text{opt}[i, i-1] = 1 \]

**Example:**

\[ \text{kbest("GCACGACG", 3)} = [(3, ".\ldots\ldots\ldots"), (3, ".\ldots\ldots\ldots"), (2, ".\ldots\ldots\ldots")\ldots\ldots\ldots] \]