Dynamic Programming 101

- DP = recursion (divide-n-conquer) + caching (overlapping subproblems)
Dynamic Programming 101

- DP = recursion (divide-n-conquer) + caching (overlapping subproblems)
- the simplest example is Fibonacci
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)
```
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)
```
Dynamic Programming 101

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

- the simplest example is Fibonacci

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

  naive recursion without memoization: \( O(1.618^{\ldots n}) \)

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

- Fibonacci sequence:
  - Level 0: 5
  - Level 1: 4, 3
  - Level 2: 3, 2, 2, 1
  - Level 3: 2, 1, 1, 1
Dynamic Programming 101

- DP = recursion (divide-n-conquer) + caching (overlapping subproblems)
- the simplest example is Fibonacci

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

\[ f(1) = f(2) = 1 \]

```
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 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]
```

DP1: top-down with memoization: \( O(n) \)
Dynamic Programming 101

- DP = recursion (divide-n-conquer) + caching (overlapping subproblems)
- the simplest example is Fibonacci
  \[ f(n) = f(n-1) + f(n-2) \]
  \[ f(1) = f(2) = 1 \]

**Naive recursion without memoization:**
\[ O(1.618...^n) \]

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

```python
fibs={1:1, 2:1} # hash table (dict)
def fib1(n):
    if n not in fibs:
        fibs[n] = fib1(n-1) + fib1(n-2)
    return fibs[n]
```

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

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

```python
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 fib0(n):
    fibs={1:1, 2:1} # hash table (dict)
def fib1(n):
    if n not in fibs:
        fibs[n] = fib1(n-1) + fib1(n-2)
    return fibs[n]
```
Number of Bitstrings
Number of Bitstrings

- number of $n$-bit strings that do not have 00 as a substring
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
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 $n=0$?
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
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
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
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

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 2, \quad f(0) = 1$$
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: $f(i)$ -- max independent set for $a[1]..a[i]$ (1-based index)
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: $f(i)$ -- max independent set for $a[1]..a[i]$ (1-based index)

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

  $b(i) = [f(i) \neq f(i-1)] : \text{take } a[i] \text{ for } f(i)\?$

  $f(0) = 0; f(1) = a[1]?$  

  No! $f(1) = \max\{a[1], 0\}$
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \)  
    (1-based index)
    \[
    f(i) = \max\{f(i-1), f(i-2) + a[i]\}
    \]
    \[
    b(i) = [f(i) \neq f(i-1)]: \text{take } a[i] \text{ for } f(i)?
    \]
    \[
    f(0) = 0; f(1) = a[1]?
    \]
    No! \( f(1) = \max\{a[1], 0\} \)
    or even better: \( f(0) = 0; f(-1) = 0 \)
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \)

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

\[
b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i)\?
\]

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( f(i) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{best value backpointer}
\]
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(i) -- \text{max independent set for } a[1]..a[i] (1-based index)\n  f(i) = \max\{f(i - 1), f(i - 2) + a[i]\} \n  b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i)？\n
\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
i & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
a[i] & & & 9 & 10 & 8 & 5 & 2 & 4 \\
\hline
f(i) & 0 & & & & & & & \\
\hline
\end{array}
\]

*best value backpointer*
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: $f(i)$ -- max independent set for $a[1]..a[i]$
  
$f(i) = \max\{f(i-1), f(i-2) + a[i]\}$

$b(i) = [f(i) \neq f(i-1)]$ : take $a[i]$ for $f(i)$?

<table>
<thead>
<tr>
<th>$i$</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best value backpointer
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: $f(i) \rightarrow$ max independent set for $a[1]..a[i]$

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

$b(i) = [f(i) \neq f(i-1)]$: take $a[i]$ for $f(i)$?

$$f(0) = 0; \; f(1) = a[1]$$?

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

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

---

<table>
<thead>
<tr>
<th>$i$</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**best value backpointer**
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \) (1-based index)
  
  \[
  f(i) = \max\{f(i - 1), f(i - 2) + a[i]\}
  \]
  
  \[
  b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i) ?
  \]

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

best value

backpointer
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: \( f(i) \) -- max independent set for \( a[1..i] \) (1-based index)

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

\[
b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i)?
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**best value**

**backpointer**
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \)

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

\[
b(i) = [f(i) \neq f(i-1)] : \text{take } a[i] \text{ for } f(i)\?
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best value

Backpointer
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \) (1-based index)

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

\( b(i) = [f(i) \neq f(i-1)] : \text{take } a[i] \text{ for } f(i) \)?
Max Independent Set (MIS)

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

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

$$b(i) = \begin{cases} 
\text{true} & \text{if } f(i) \neq f(i-1) \text{; take } a[i] \text{ for } f(i) \\
\text{false} & \text{otherwise}
\end{cases}$$

$$f(0) = 0; f(1) = a[1]?$$
No! $f(1) = \max\{a[1], 0\}$
or even better: $f(0) = 0; f(-1) = 0$

<table>
<thead>
<tr>
<th>$i$</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>$b(i)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*best value backpointer*
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: $f(i)$ -- max independent set for $a[1]..a[i]$
  
  $f(i) = \max\{f(i-1), f(i-2) + a[i]\}$
  
  $b(i) = [f(i) \neq f(i-1)]$: take $a[i]$ for $f(i)$?

<table>
<thead>
<tr>
<th>$i$</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>$b(i)$</td>
<td></td>
<td></td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best value $\text{backpointer}$
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: $f(i) -- \text{max independent set for } a[1]..a[i]$
  
  
$f(i) = \max\{f(i - 1), f(i - 2) + a[i]\}$

$b(i) = [f(i) \neq f(i - 1)]: \text{take } a[i] \text{ for } f(i)?$

<table>
<thead>
<tr>
<th>$i$</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>$b(i)$</td>
<td></td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*best value
backpointer*
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \) (1-based index)

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

\[
b(i) = [f(i) \neq f(i-1)]: \text{take } a[i] \text{ for } f(i)?
\]

\[
\begin{array}{cccccc}
 i & -1 & 0 & 1 & 2 & 3 & 4 \\
 a[i] & & 9 & 10 & 8 & 5 & 2 & 4 \\
 f(i) & 0 & 0 & 9 & 10 & 17 & 17 & 19 & 21 \\
 b(i) & \text{T} & \text{T} & \text{T} & \text{T} & \text{T} & \text{T} & \text{T} & \text{T} & \text{T} \\
\end{array}
\]

(best value backpointer)
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \)
    
    \[
    f(i) = \max\{f(i - 1), f(i - 2) + a[i]\}
    
    b(i) = \left[ f(i) \neq f(i - 1) \right] : \text{take } a[i] \text{ for } f(i) ?
    
    \]

\[
\begin{array}{ccccccc}
  i & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
  \hline
  a[i] & \ & \ & \ & \ & \ & \ & \ & \ \\
  f(i) & 0 & 0 & 9 & 10 & 17 & 17 & 19 & 21 \\
  b(i) & \ & \ & \ & \ & \ & \ & \ & \\
\end{array}
\]

best value
backpointer

\[
f(0) = 0; f(1) = a[1]?
No! \ f(1) = \max\{a[1],0\}
\text{or even better: } f(0) = 0; f(-1) = 0
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: \(f(i)\) -- max independent set for \(a[1]..a[i]\) (1-based index)

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

\[
b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i)?
\]

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a[i])</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>(f(i))</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>(b(i))</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Best value backpointer: start here

Recursively backtrack the optimal solution.
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: $f(i) \rightarrow$ max independent set for $a[1]..a[i]$
  
  $f(i) = \max\{f(i-1), f(i-2) + a[i]\}$

  $b(i) = [f(i) \neq f(i-1)] :$ take $a[i]$ for $f(i)$?

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[i]$</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>$f(i)$</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>$b(i)$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

best value
backpointer
start here

recursively backtrack the optimal solution

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

or even better: $f(0) = 0; f(-1) = 0$
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \) (1-based index)

\[
f(i) = \max\{f(i - 1), f(i - 2) + a[i]\}
\]
\[
b(i) = [f(i) \neq f(i - 1)] : \text{take } a[i] \text{ for } f(i)?
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>( b(i) )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

best value backpointer

start here

backtrack

take

not

take

recursively backtrack the optimal solution
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: \( f(i) \) -- max independent set for \( a[1]..a[i] \)

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

\[
f(0) = 0; \quad f(1) = a[1]?
\]

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

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

---

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>( b(i) )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\( b(i) = [f(i) \neq f(i-1)] : \text{ take } a[i] \text{ for } f(i)? \)

---

<table>
<thead>
<tr>
<th>( i )</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a[i] )</td>
<td></td>
<td></td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>( b(i) )</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

---

**MIS**

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

best value

backpointer

start here
Graph Interpretation of DP
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)

- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)
- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)
- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)

- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)

- $f(i)$: longest path between source and node $i$
• MIS: longest path between source and target (see lecture video)
  
• each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \rightarrow 0 i$ (not take)

• $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)
- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)
- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- MIS: longest path between source and target (see lecture video)
- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)
- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)
- each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)
- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)

- each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \xrightarrow{0} i$ (not take)

- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- MIS: longest path between source and target (see lecture video)
- Each node $i$ has two incoming edges: $(i - 2) \xrightarrow{a[i]} i$ (take) and $(i - 1) \rightarrow i$ (not take)
- $f(i)$: longest path between source and node $i$
Graph Interpretation of DP

- **MIS**: longest path between source and target (see lecture video)
  - each node $i$ has two incoming edges: $(i-2) \xrightarrow{a[i]} i$ (take) and $(i-1) \xrightarrow{0} i$ (not take)
  - $f(i)$: longest path between source and node $i$
- fibonacci & bitstrings: number of paths between source and target
Summary

• Divide-and-Conquer $= \text{divide} + \text{conquer} + \text{combine}$

• Dynamic Programming $= \text{multiple divides} + \text{memoized conquer} + \text{summarized combine}$

• 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
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

\[ f(n) = \max \begin{cases} f(n-1) + 0 \\ f(n-2) + a[n] \end{cases} \]

- summary operator ⊕ (across divides)
- combination operator ⊗ (within a divide)
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!

$$B(n) = \bigoplus_{i=1}^{n} \left( B(i-1) \otimes B(n-i) \right)$$

$$B(0) = 1$$

<table>
<thead>
<tr>
<th>Problem</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib</td>
<td>$+$</td>
<td>$\times$</td>
</tr>
<tr>
<td>MIS</td>
<td>max</td>
<td>$+$</td>
</tr>
<tr>
<td># BSTs</td>
<td>$+$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Knapsack</td>
<td>max</td>
<td>$+$</td>
</tr>
<tr>
<td>Shortest path</td>
<td>min</td>
<td>$+$</td>
</tr>
</tbody>
</table>
## 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, best</td>
</tr>
<tr>
<td>(balanced) tree traversal (DFS)</td>
<td></td>
<td>binary search:</td>
</tr>
<tr>
<td></td>
<td>heapify (top-down)</td>
<td>search in BST: worst, best</td>
</tr>
<tr>
<td>DP</td>
<td># of BSTs (hw5), midterm</td>
<td>Fib, # of bitstrings</td>
</tr>
<tr>
<td></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>
<tr>
<td>DP</td>
<td>Two Divides</td>
<td>Multiple Divides</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>DP</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>Divide-n-Conquer</th>
<th>two-way (binary divide)</th>
<th>one-way (unary divide)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary tree</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DP</th>
<th># of BSTs (hw5)</th>
<th>Fib, # of bitstrings (hw5)…</th>
</tr>
</thead>
<tbody>
<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>

**Graph**

- $v$
- $u_1$
- $u_2$
- $e$

**Hypergraph**

- $v$
- $u_1$
- $u_2$
- $e$
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: $O(V + E)$
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

   ![Diagram](image)

   - 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

\[
\begin{align*}
&\text{all } O(n^3) \\
&\text{A} \\
&\text{B} & \text{C} \\
&i & j & k
\end{align*}
\]
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

\[ \text{all } O(n^3) \]
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

\[
(i, n) \quad \text{bottom-up} \quad \text{left-to-right}
\]

all \(O(n^3)\)
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

\[ \text{all } O(n^3) \]
RNA Folding Example

Nussinov Algorithm — Traceback Example

Bottom-up

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 k-best distances from the k-best of each incoming node \( u \)

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

  Can improve it to: (cf. midterm & teams, w/ quickselect)
  
  k-best: \( O(E + Vk \log k) \) (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

\[ \text{opt}[1, 8] \]

\[ \text{opt}[i, j] = \bigoplus \left\{ \text{opt}[i, j-1], \bigotimes_{i \leq p < j} \left( \text{opt}[i, p-1] \bigotimes \text{opt}[p+1, j-1] \right) \otimes 1 \right\} \]

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

\[ \text{opt} \]

<table>
<thead>
<tr>
<th>op</th>
<th>( \oplus )</th>
<th>( \otimes )</th>
<th>1_\oplus</th>
</tr>
</thead>
<tbody>
<tr>
<td>best</td>
<td>max</td>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>total</td>
<td>+</td>
<td>( \times )</td>
<td>1</td>
</tr>
</tbody>
</table>

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