Dynamic Programming 101
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)
```

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)
```
Dynamic Programming 101

- DP = recursion (divide-and-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)
```

naive recursion without memoization: \( O(1.618^{n}) \)
Dynamic Programming 101

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

- the simplest example is Fibonacci

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

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

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

```
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: \(O(n)\)

---

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

```
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: \( O(n) \)

```
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):
        fibs.append(f[-1]+f[-2])
    return f[-1]
```

- DP2: bottom-up: \( O(n) \)
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 \( n \)-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 $n$-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 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 — 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! $f(1) = \max\{a[1], 0\}$
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) ≠ 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
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)

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

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

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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)

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

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

  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></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: \(f(i)\) -- 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>9</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</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>

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

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

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

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

\(\text{best value}\)

\(\text{backpointer}\)
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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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 $a[1]..a[i]$ (l-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>

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

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

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 \(f(i)\)?

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

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>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 $a[1]..a[i]$

$l$-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></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>19</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>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 — 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)

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

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

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>T</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)

\[
\begin{align*}
\text{f}(i) & = \max\{\text{f}(i - 1), \text{f}(i - 2) + a[i]\} \\
b(i) & = [\text{f}(i) \neq \text{f}(i - 1)] : \text{take } a[i] \text{ for } \text{f}(i)\
\end{align*}
\]

<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></td>
<td></td>
<td>T</td>
<td></td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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></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>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

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

\[ \text{best value backpointer} \]
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: $f(i)$ -- max independent set for $a[1]..a[i]$

  (1-based index)

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

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

  or even better: $f(0) = 0; f(1) = a[1]$?

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

  $f(0) = 0; f(-1) = 0$

<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>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td></td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
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 \( a[1]..a[i] \)

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

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

\[
\begin{array}{ccccccc}
    i & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
    \hline
    a[i] & & 9 & 10 & 8 & 5 & 2 & 4 & \\
    f(i) & 0 & 0 & 9 & 10 & 17 & 17 & 19 & 21 \\
    b(i) & & T & T & T & T & F & T & T \\
\end{array}
\]

Recursively backtrack to find the optimal solution

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

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>$i$</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>

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

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

Start here

Best value

Backpointer

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

<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>

**Backpointer**
- \( f(0) = 0; f(1) = a[1] \)?
- No! \( f(1) = \max \{ a[1], 0 \} \)
- or even better: \( f(0) = 0; f(-1) = 0 \)
- recursively backtrack the optimal solution
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: $f(i) \rightarrow$ max independent set for $a[1]..a[i]$

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

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

<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>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>backtrack</td>
<td>*</td>
<td>*</td>
<td>take</td>
<td>take</td>
<td>not</td>
<td>take</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{MIS} \quad f(n) = \max \left\{ \begin{array}{l}
  f(n - 1) + 0 \\
  f(n - 2) + a[n]
\end{array} \right\}$

**Summary operator** (across divides) $\oplus$

**Combination operator** (within a divide) $\otimes$

$\text{best value backpointer start here}$

<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>F</td>
<td>T</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>backtrack</td>
<td>*</td>
<td>*</td>
<td>take</td>
<td>take</td>
<td>not</td>
<td>take</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $i$
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 $i$
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 $i$
- Fibonacci & bitstrings: number of paths between source and target
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
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

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

- **divide-n-conquer**
  - single division, independent conquer, combine

- **DP = divide-n-conquer with multiple divisions**
  - for each possible division
    - divide
    - conquer with memoization
    - combine subsolutions using the combination operator $\otimes$
  - summarize over all possible divisions using the summary operator $\oplus$
  - multiple divisions $\Rightarrow$ overlapping subproblems
  - each single division $\Rightarrow$ independent subproblems!

- **Examples**
  - Fibonacci: $+$, $\times$
  - Maximum Independent Set (MIS): $\max$, $+$
  - Number of Binary Search Trees (BSTs): $+$, $\times$
  - Knapsack: $\max$, $+$
  - Shortest Path: $\min$, $+$

- **Formula**
  - $B(n) = \bigoplus_{i=1}^{n} \left( B(i-1) \otimes B(n-i) \right)$
  - $B(0) = 1$
## Unary vs. Binary Divisions

\[(a) : T(n) = 2T(n/2) + \ldots\] \quad \begin{align*}
(b) : T(n) &= T(n - 1) + \ldots \\
(c) : T(n) &= T(n/2) + \ldots
\end{align*}

<table>
<thead>
<tr>
<th>Divide-n-conquer</th>
<th>Branching (binary division)</th>
<th>One-sided (unary division)</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>

- **(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></th>
<th>two divisions</th>
<th>multiple division</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DP</strong></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></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>Divide-n-conquer</td>
<td>DP</td>
<td></td>
</tr>
<tr>
<td></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>
Graph Interpretation of Unbounded Knapsack
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: \( O(V + E) \)
Generalized Viterbi for DAHs (Hypergraphs)

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

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

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

\[ all \ O(n^3) \]
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

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

\[ \begin{align*}
(1, n) \\
\text{bottom-up} \\
(1, n) \\
\text{left-to-right} \\
(1, n) \\
\text{right-to-left}
\end{align*} \]

all \( O(n^3) \)
Example: RNA Folding as CKY Parsing

• Dynamic Programming — $O(n^3)$
• bottom-up CKY parsing
• example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — \( O(n^3) \)
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- **Dynamic Programming** — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
Example: RNA Folding as CKY Parsing

- Dynamic Programming — $O(n^3)$
- bottom-up CKY parsing
- example: maximize # of pairs (A-U, G-C, or G-U)
RNA Folding Example (1-best)

$opt[1,8] = 3$

RNA Folding Example (1-best)

```
(1, n)
```

```
G C A C G A C G
```

```
1 2 3 4 5 6 7 8
```

```
G C A
```

```
A C G
```

```
( ) . ) . . )
```

```
G C A C G A C G
```

```
( )
```

```
. .
```

```
G C A C G A C G
```

```
12345678
```

```
G C A C G A C G
```

```
xxx(xxx)
```

```
G C A
```

```
A C G
```

```
) . . )
```

```
() . ( ()
```

```
().(().)
```

```
().((.))
```

```
G C A C G A C G
```

```
() . . . )
```

```
G C A C G A C G
```

```
. .
```

```
G C A C G A C G
```

RNA Folding Example (1-best)

opt[1,8] = 3

GCA
G
A
C
G
A
C
G

RNA Folding Example (1-best)

opt[1,8] = 3

RNA Folding Example (1-best)

```
12345678
GCACGACG
xxx(XXX)
GCA
xx.
GC
().
().
GAC
().
A
().
().
().
().
GAC
().
A
().
().
().
().
().
().
().
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
()
().
()
()
RNA Folding Example (1-best)

opt[1,8] = 3

12345678
GCACGACG
xxx(×××)
GCA
xx .
GC ()  
() .

GAC (x)
A .
( .)  
().(()

GCA
().
..()
..()
..()
..()
..()
RNA Folding Example (1-best)

opt[1,8] = 3

GCA
xxx(xxx)
GCA
xx.
GAC
().
GAC
A.
().
().
().

RNA Folding Example (1-best)

opt[1,8] = 3

G C A C G A C G

GCA

xx.

G

() (.).

().((().))

GAC

A

(xx)()

().

().

(x)

().

().

().

12345678

GACGACG

xxx(XXX)

GCA

xx.

G

() (.).

().((().))

RNA Folding Example (1-best)

\[ \text{opt}[1,8] = 3 \]

GCA

\[ \text{opt}[i, i] \]

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

GCA

\[ \text{bottom-up} \]

RNA Folding Example (1-best)

\[
opt[1,8] = 3
\]

\[
\text{GCACGACG}
\]

RNA Folding Example (1-best)

$opt[1,8] = 3$

From 1-best to k-best

- each subproblem will now store top-k best answers instead of a single best
- we’ll first extend Viterbi on DAGs to k-best Viterbi
- then extend generalized Viterbi on DAHs (e.g., CKY or Nussinov) to k-best
k-best Viterbi on Graph

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

```latex
\begin{align*}
\text{for each node } v, \\
\quad &\text{compute its k-best distances} \\
\quad &\text{from the k-best of each incoming node } u
\end{align*}
```

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”)

\[\begin{array}{c}
\text{Incoming}[v] \\
\hspace{1cm} k\text{best}[u] \\
\hspace{2cm} \ldots \\
\hspace{3cm} u \\
\hspace{2cm} k\text{best}[v] \\
\hspace{1cm} \ldots \\
\hspace{1cm} k\text{best}[p] \\
\hspace{2cm} \ldots \\
\hspace{3cm} p \\
\hspace{2cm} \ldots \\
\hspace{1cm} k\text{best}[q] \\
\hspace{2cm} \ldots \\
\hspace{3cm} q
\end{array}\]
k-best Viterbi on Hypergraph

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

```
\begin{align*}
\text{kbest("GCACGACG", 3) =} & \quad [ ] \\
\end{align*}
```
k-best Viterbi on Hypergraph

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

\[ \text{k-best}("GCACGACG", 3) = [(3, '(().((.))))')] \]
**k-best Viterbi on Hypergraph**

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

\[
k\text{-best}("GCACGACG", 3) = [(3, '()().()'), (3, '()().()')]\]
k-best Viterbi on Hypergraph

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

```
kbest("GCACGACG", 3) = [(3, '().((.))'), (3, '().().()'), (2, '().().()'), (2, '().().()')]
```