CS3000: Algorithms & Data
Jonathan Ullman

Lecture 6:
• Dynamic Programming:
  Fibonacci Numbers, Interval Scheduling

Sep 25, 2018
Dynamic Programming

• Don’t think too hard about the name
  • *I thought dynamic programming was a good name. It was something not even a congressman could object to.* So I used it as an umbrella for my activities. -Bellman

• Dynamic programming is careful recursion
  • Break the problem up into small pieces
  • Recursively solve the smaller pieces
  • **Key Challenge:** identifying the pieces

  Divide and Conquer: speeding up simple algorithms
  Dynamic Programming: often the only polynomial time alg
Warmup: Fibonacci Numbers
Fibonacci Numbers

• 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
• $F(n) = F(n - 1) + F(n - 2)$

• $F(n) \to \phi^n \approx 1.62^n$

• $\phi = \left(\frac{1+\sqrt{5}}{2}\right)$ is the golden ratio
Fibonacci Numbers: Take I

FibI(n):
  If (n = 0): return 0
  ElseIf (n = 1): return 1
  Else: return FibI(n-1) + FibI(n-2)

• How many recursive calls does FibI(n) make?
  • $2^n$
  • $2n$

\[
T(n) = \text{# of calls made by FibI(n)}
\]
\[
T(n) = T(n-1) + T(n-2)
\]
\[
T(0) = 0
\]
\[
T(1) = 0
\]
\[
T(2) = 2
\]
\[
T(n) = F(n) \approx 1.62^n
\]
Fibonacci Numbers: Take II

"Memoization", "Top-Down"

\[ M \leftarrow \text{empty array}, M[0] \leftarrow 0, M[1] \leftarrow 1 \]

\[
\text{FibII}(n):
\begin{align*}
\text{If (M}[n]\text{ is not empty): return M}[n]\text{]}
\text{ElseIf (M}[n]\text{ is empty):}
\quad M[n] \leftarrow \text{FibII}(n-1) + \text{FibII}(n-2)
\quad \text{return M}[n]\text{]}
\end{align*}
\]

- How many recursive calls does \textbf{FibII}(n) make?

Array has \( n+1 \) elements, need to fill \( n-1 \)

Each time we make a pair of recursive calls, we fill one \( M[i] \)

\[ \Rightarrow \leq 2(n-1) = O(n) \text{ recursive calls} \]
Fibonacci Numbers: Take III

"Bottom-Up"

FibIII(n):
M[0] ← 0, M[1] ← 1
For i = 2,...,n:
    M[i] ← M[i-1] + M[i-2]
return M[n]

• What is the running time of FibIII(n)?

\[ F(n) = \left( \frac{1+\sqrt{5}}{2} \right)^n \]

n-1 additions, each addition involves \( \Theta(n) \)-digit numbers
\( \Rightarrow \Theta(n^2) \) time
Fibonacci Numbers

• 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
• \( F(n) = F(n - 1) + F(n - 2) \)

• Solving the recurrence recursively takes \( \approx 1.62^n \) time
  • Problem: Recompute the same values \( F(i) \) many times

• Two ways to improve the running time
  • Remember values you’ve already computed (“top down”)
  • Iterate over all values \( F(i) \) (“bottom up”)

• **Fact:** Can solve even faster using Karatsuba’s algorithm!
Dynamic Programming: Interval Scheduling
Interval Scheduling (Weighted)

• How can we optimally schedule a resource?
  • This classroom, a computing cluster, ...

• **Input:** $n$ intervals $(s_i, f_i)$ each with value $v_i$
  • Assume intervals are sorted so $f_1 < f_2 < \cdots < f_n$

• **Output:** a compatible schedule $S$ maximizing the total value of all intervals
  • A **schedule** is a subset of intervals $S \subseteq \{1, \ldots, n\}$
  • A schedule $S$ is **compatible** if no $i, j \in S$ overlap
  • The **total value** of $S$ is $\sum_{i \in S} v_i$
Interval Scheduling

Index

1
\[ v_1 = 2 \]

2
\[ v_2 = 4 \]

3
\[ v_3 = 4 \]

4
\[ v_4 = 7 \]

5
\[ v_5 = 2 \]

6
\[ v_6 = 1 \]

\[ S = \{2, 5\} \]
\[ \text{value}(S) = 6 \]
A Recursive Formulation

• Let $O$ be the **optimal** schedule

• **Case 1:** Final interval is not in $O$ (i.e. $6 \notin O$)
  • Then $O$ must be the optimal solution for $\{1, \ldots, 5\}$

```latex
\begin{align*}
\text{If } O \text{ were not the optimal sched for } &\{1, \ldots, 5\} \\
\text{then } O \text{ is not the optimal sched for } &\{1, \ldots, 6\}.
\end{align*}
```

Index

1. $v_1 = 2$
2. $v_2 = 4$
3. $v_3 = 4$
4. $v_4 = 7$
5. $v_5 = 2$

$O$ is opt on these
A Recursive Formulation

• Let $O$ be the **optimal** schedule

• **Case 2:** Final interval is in $O$ (i.e. $6 \in O$)
  • Then $O$ must be $6 +$ the optimal solution for $\{1, \ldots, 3\}$

$$\text{If } O \setminus \{6\} \text{ were not opt for } \{1, \ldots, 3\} \text{ then } 6 + [\text{opt for } \{1, \ldots, 3\}] \text{ is better than } O$$

Index

1
2
3
4
5
6

$v_1 = 2$
$v_2 = 4$
$v_3 = 4$
$v_6 = 1$

which is better?

1. opt sched for $\{1, \ldots, 5\}$
2. opt sched for $\{1, \ldots, 3\}$ + 6

which is better?
A Recursive Formulation

- Let $O_i$ be the **optimal schedule** using only the intervals $\{1, \ldots, i\}$

- **Case 1:** Final interval is not in $O$ ($i \notin O_i$)
  - Then $O$ must be the optimal solution for $\{1, \ldots, i - 1\}$ ($O_{i-1}$)

- **Case 2:** Final interval is in $O$ ($i \in O_i$)
  - Assume intervals are sorted so that $f_1 < f_2 < \cdots < f_n$
  - Let $p(i)$ be the largest $j$ such that $f_j < s_i$
  - Then $O_i$ must be $i +$ the optimal solution for $\{1, \ldots, p(i)\}$

  $$O_i = i + O_{p(i)}$$
A Recursive Formulation

- Let $OPT(i)$ be the value of the optimal schedule using only the intervals $\{1, \ldots, i\}$

  - Case 1: Final interval is not in $O$ ($i \notin O_i$)
    - Then $O$ must be the optimal solution for $\{1, \ldots, i - 1\}$

  - Case 2: Final interval is in $O$ ($i \in O_i$)
    - Assume intervals are sorted so that $f_1 < f_2 < \cdots < f_n$
    - Let $p(i)$ be the largest $j$ such that $f_j < s_i$
    - Then $O$ must be $i +$ the optimal solution for $\{1, \ldots, p(i)\}$

- $OPT(i) = \max\{OPT(i - 1), v_i + OPT(p(i))\}$
- $OPT(0) = 0, OPT(1) = v_1$
Interval Scheduling: Take I

// All inputs are global vars
FindOPT(n):
    if (n = 0): return 0
    elseif (n = 1): return v_1
    else:
        return max{FindOPT(n-1), v_n + FindOPT(p(n))}

• What is the running time of FindOPT(n)?

As many as $1.62^n$ recursive calls

$\forall i \quad p(i) = i-2$
Interval Scheduling: Take II

// All inputs are global vars
M ← empty array, M[0] ← 0, M[1] ← Φ
FindOPT(n):
    if (M[n] is not empty): return M[n]
    else:
        M[n] ← max{FindOPT(n-1), v_n + FindOPT(p(n))}
    return M[n]

• What is the running time of FindOPT(n)?

    Need to fill ≤ n-1 entries of M

  × 2 recursive calls / entry

  ≤ 2(n-1) recursive calls

  O(n) running time

  + O(nlog n) to sort by f;
Interval Scheduling: Take II

Index
1 \[ v_1 = 2 \] \[ p(1) = 0 \]
2 \[ v_2 = 4 \] \[ p(2) = 0 \]
3 \[ v_3 = 4 \] \[ p(3) = 1 \]
4 \[ v_4 = 7 \] \[ p(4) = 0 \]
5 \[ v_5 = 2 \] \[ p(5) = 3 \]
6 \[ v_6 = 1 \] \[ p(6) = 3 \]

\[ M[4] = \max \{ 6, 7 + 0 \} \]
\[ M[5] = \max \{ 7, 2 + 6 \} \]
\[ M[6] = \max \{ 8, 1 + 6 \} \]

\[ M[i] = \text{OPT}(i) \]
\[ M[23] = \max \{ M[13], 4 + M[0] \} \]
\[ M[3] = \max \{ M[23], 4 + M[13] \} \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Interval Scheduling: Take III

// All inputs are global vars
FindOPT(n):
    M[0] ← 0, M[1] ← 1
    for (i = 2,...,n):
        M[i] ← max{FindOPT(n - i), vn + FindOPT(p(i))}
    return M[n]

• What is the running time of FindOPT(n)?
  \[ O(n) + O(n \log n) \text{ to sort if needed} \]
Finding the Optimal Solution

- Let $OPT(i)$ be the **value of the optimal schedule** using only the intervals $\{1, \ldots, i\}$

- **Case 1:** Final interval is not in $O$ ($i \notin O_i$) \(\Rightarrow\) $O_i = O_{i-1}$

- **Case 2:** Final interval is in $O$ ($i \in O$) \(\Rightarrow\) $O_i = \xi_i \beta + O_{p(i)}$

- \[OPT(i) = \max\{OPT(i-1), v_i + OPT(p(i))\}\]

  - then $O_i = O_{i-1}$
  - then $O_i = \xi_i \beta + O_{p(i)}$
Interval Scheduling: Take II

\[ M[0] = \text{OPT}(0) \]

\[ M[1] = \text{OPT}(1) \]

\[ M[2] = \text{OPT}(2) \]

\[ M[3] = \text{OPT}(3) \]

\[ M[4] = \text{OPT}(4) \]

\[ M[5] = \text{OPT}(5) \]

\[ M[6] = \text{OPT}(6) \]

\[ M[7] = \text{OPT}(7) \]

\[ M[8] = \text{OPT}(8) \]

\[ 0 = M[0], M[1], M[2], M[3] \]

\[ 2 = M[4], M[5], M[6], M[7] \]

\[ 4 = M[8] \]
Interval Scheduling: Take III

// All inputs are global vars
FindSched(M,n):
    if (n = 0): return ∅
    elseif (n = 1): return {1}
    elseif (vn + M[p(n)] > Mn-1):
        return {n} + FindSched(M,p(n))
    else:
        return FindSched(M,n-1)

• What is the running time of \text{FindSched}(n) ?
  \[O(n)\text{ time}\]
Now You Try

1. $v_1 = 3$
2. $v_2 = 5$
3. $v_3 = 9$
4. $v_4 = 6$
5. $v_5 = 13$
6. $v_6 = 3$

<table>
<thead>
<tr>
<th>$p(1)$</th>
<th>$p(2)$</th>
<th>$p(3)$</th>
<th>$p(4)$</th>
<th>$p(5)$</th>
<th>$p(6)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Programming Recap

• Express the optimal solution as a **recurrence**
  • Identify a small number of **subproblems**
  • Relate the optimal solution on subproblems

• Efficiently solve for the **value** of the optimum
  • Simple implementation is exponential time
  • **Top-Down:** store solution to subproblems
  • **Bottom-Up:** iterate through subproblems in order

• Find the **solution** using the table of **values**