Fundamental Algorithms

Topic 9: Dynamic Programming

What is this Topic?

- **Dynamic Programming** is a technique for designing algorithms.
  - Fits problems where we wish to find an **optimal solution**.
    - Like finding a **shortest path** or a **max matching**.
    - Unlike soring or searching.
  - Based on **solving smaller instances of the same problem**.
    - Similarly to **divide and conquer**, except that the smaller problems overlap.
Cutting Steel Rods

- Problem.
  - A company produced a steel rod of length $n$.
  - We have a table of the prices of steel rods of lengths 1 up to $n$.
  - Find a way to cut the rod that maximizes the revenue.

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

Examples

(a) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(b) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(c) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(d) \[ \begin{array}{c} \text{\textbullet} \\
\end{array} \]
(e) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(f) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(g) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]
(h) \[ \begin{array}{c} \text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \]

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
Maximize Profit from Rod of Length 5

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

Naïve Algorithm

- **Algorithm**.
  - Go over every possible way to cut the rod.
  - Check the profit for each option.
  - Find the option that maximizes the profit.

- **Issue**.
  - There are $2^{n-1}$ ways of cutting a rod of length $n$.
  - The running time would be $\Omega(2^n)$. 
Relying on Smaller Problems

- \( p_i \) – price of a piece of length \( i \).
- \( r_i \) – max revenue for rod of length \( i \).

We have \( r_0 = p_0 = 0 \), \( r_1 = p_1 \).

\[
r_2 = \max\{p_2 + r_0, p_1 + r_1\}
\]

\[
r_3 = \max\{p_3 + r_0, p_2 + r_1, p_1 + r_2\}
\]

- Each option corresponds to a different length of the first piece.

- In general:

\[
r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\}.
\]

Example

\[
\begin{align*}
  r_0 &= 0 \\
  r_1 &= 1 \\
  r_2 &= \max\{p_2 + r_0, p_1 + r_1\} = \max\{5,2\} = 5 \\
  r_3 &= \max\{p_3 + r_0, p_2 + r_1, p_1 + r_2\} = 8 \\
  r_4 &= \max\{p_4 + r_0, p_3 + r_1, p_2 + r_2, p_1 + r_3\} = 10
\end{align*}
\]

<table>
<thead>
<tr>
<th>length ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price ( p_i )</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>
Dynamic Programming Solution

- $r_0 = p_0 = 0, r_1 = p_1$.
  \[ r_n = \max_{1 \leq i \leq n} \{ p_i + r_{n-i} \} \]

- **Algorithm.**
  - Use the formula to find $r_2$, then $r_3$, ..., up to $r_n$.
  - Computing $r_k$ takes $O(k)$ time.
  - Total running time is $O(1 + 2 + 3 + \cdots + n) = O(n^2)$.

Some Intuition

- **Recap.**
  - Checking every possible cut took $\Omega(2^n)$ time.
  - Splitting into smaller problems took $O(n^2)$.

- Why is there such a big difference?
  - The first approach has a lot of repetitions!
  - **Example.** There are four ways to cut first 3 inches, separating them from other $n - 3$.
  - In each of these four cases, we repeat the exact same calculations for the larger piece.
Finding the Optimal Cut

- Our algorithm finds the optimal profit.
- How can we also find the structure of a cut leading to this profit?
  - When finding $r_i$, we also store the size of the first piece – the one maximizing the profit.
  - If we set $r_i = p_j + r_{i-j}$, then the cut starts with a piece of size $j$, followed by the optimal cut stored with $r_{i-j}$.
  - Running time remains $O(n^2)$.

Dynamic Programming Principles

- The dynamic programming approach.
  - First handle the smallest instances of the problem.
  - Gradually handle larger and larger instances, by relying on the smaller ones.
  - Prevents repetitions in the calculations.
Longest Increasing Subsequence

- **Problem.** Given a list of $n$ numbers, find the longest increasing subsequence in it.
  - You may remove elements from the list, but you may not change the order.
  - The longest increasing subsequence in $[40, 12, 45, 22, 50, 37, 5, 10, 46]$ is $[12, 22, 37, 46]$.
  - What is the longest increasing subsequence? $[9, 8, 7, 10, 6, 5, 11, 4, 3, 12]$
Discuss in Groups

• Problem.
  ◦ Describe a dynamic programming algorithm for the longest increasing subsequence problem.
  ◦ List $L$ of $n$ numbers.
  ◦ Define $S[k]$ to be the length of the longest increasing subsequence ending in the $k$’th number of $L$.
  ◦ Running time should be $O(n^2)$.

$[40, 12, 45, 22, 50, 37, 5, 10, 46] \rightarrow [12, 22, 37, 46]$.

“I spent the Fall quarter of 1950 at RAND... We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research... he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical...

I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. In the first place I was interested in planning, in decision making, in thinking... I decided therefore to use the word programming. I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying... Thus, 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.” --- Richard Bellman.
Finding DNA Similarities

• Problem.
  ◦ A DNA sequence is a sequence of the letters C,G,A,T.
  ◦ We are given two DNA sequences $S,S'$ of lengths $m$ and $n$.
  ◦ We wish to find the longest DNA sequence that appears in both $S$ and $S'$.
  ◦ The sequence may appear non-consecutively.

Example

$S = \text{CCGCCTACCTG}$
$S' = \text{ATATCGCTCGC}$

• Both sequences contain $CC$ (non-consecutively in $S'$).
• Both sequences contain $CCCC$.
• Both sequences contain $CCTCC$. 
Solving with Dynamic Programming

To use dynamic programming, we need to define subproblems.

- $c(k, \ell) =$ length of longest common subsequence in the first $k$ letters of $S$ and the first $\ell$ letters of $S'$.

**Warmup.**

$c(1,1) = \begin{cases} 
1, & \text{if } S[1] = S'[1] \\
0, & \text{otherwise}
\end{cases}$

Assume $S[k] = S'[\ell]$. For example:

$$S = CAGCCT ... \quad S' = ATT ...$$

Then $c(k, \ell) = 1 + c(k - 1, \ell - 1)$
Solving with Dynamic Programming

- Subproblems.
  - $c(k, \ell) =$ length of longest common subsequence in the first $k$ letters of $S$ and the first $\ell$ letters of $S'$.

\[
c(k, \ell) = \begin{cases} 
0, & \text{if } \ell = 0 \text{ or } k = 0 \\
c(k-1, \ell-1) + 1, & \text{if } S[k] = S'[\ell] \\
\max\{c(k, \ell-1), c(k-1, \ell)\}, & \text{otherwise}
\end{cases}
\]
Towards an Algorithm

- We need to compute the values of $c(k, \ell)$ from $c(0,0)$ up to $c(m, n)$.
- If we have $c(k-1, \ell - 1)$, $c(k, \ell - 1)$, and $c(k - 1, \ell)$, we can compute $c(k, \ell)$ in $O(1)$ time.
- In what order should we compute the $c(k, \ell)$?

$$c(k, \ell) = \begin{cases} 
0, & \text{if } \ell = 0 \text{ or } k = 0 \\
c(k - 1, \ell - 1) + 1, & \text{if } S[k] = S'[\ell] \\
\max\{c(k, \ell - 1), c(k - 1, \ell)\}, & \text{otherwise}
\end{cases}$$

The Algorithm

- For $i$ going from 0 to $m$:
  - For $j$ going from 0 to $n$:
    - Compute $c(i, j)$.
    - When we compute $c(i, j)$, we already have $c(i - 1, j - 1), c(i - 1, j)$ and $c(i, j - 1)$.

$$c(k, \ell) = \begin{cases} 
0, & \text{if } \ell = 0 \text{ or } k = 0 \\
c(k - 1, \ell - 1) + 1, & \text{if } S[k] = S'[\ell] \\
\max\{c(k, \ell - 1), c(k - 1, \ell)\}, & \text{otherwise}
\end{cases}$$
Running Time

• For \( i \) going from 0 to \( m \):
  ◦ For \( j \) going from 0 to \( n \):
    • Compute \( c(i, j) \).
  ◦ When we compute \( c(i, j) \), we already have \( c(i - 1, j - 1), c(i - 1, j) \) and \( c(i, j - 1) \).

• Running time.
  ◦ We compute \( mn \) values of \( c(i, j) \).
  ◦ Each computation takes \( O(1) \) times.
  ◦ Total time: \( O(mn) \).

General Guidelines

• How to determine if a problem can be handled using dynamic programming?
  ◦ The problem asks to find an optimal solution.
  ◦ It is easy to obtain an optimal solution when given solutions to slightly smaller subproblems.
  ◦ Likely to lead to an efficient algorithm when the subproblems overlap.
Group Exercise

- Solve using dynamic programming.
  - \( L \) – list of \( n \) positive integers.
  - Find an increasing subsequence of elements in \( L \) whose sum is as large as possible.
    - The number of elements in the sequence does not matter.
  - When \( L = \{100, 10, 20, 30, 35\} \), the best sequence is \( \{100\} \).
    - There are more elements in \( \{10, 20, 30, 35\} \) but it sums up only to 95.

Group Exercise

- Solve using dynamic programming.
  - We change the DNA sequence question.
  - Two DNA sequences \( S, S' \) of lengths \( m \) and \( n \).
  - We wish to find the longest DNA sequence that appears in both \( S \) and \( S' \).
    - The sequence must appear consecutively in both \( S \) and \( S' \)!
      - (We cannot remove letters in the middle of a sequence).
      - You might need to slightly change the subproblem definition.
The Best Party

- Your knowledge of algorithms led you to a job in a fancy high-tech company.
- You need to plan a company party!
- Every employee has a “fun level”, which is a positive number. You need to choose who to invite to maximize the sum of the fun levels.
- The company hierarchy is a tree. If the direct boss of an employee is present, their fun level drops to zero.

Example

fun = 37  
fun = 24  
fun = 29
Find the Best Party

Discuss in Groups

- Your knowledge of algorithms led you to a job in a fancy high-tech company.
- You need to plan a company party!
- Every employee has a "fun level", which is a positive number. You need to choose who to invite to maximize the sum of the fun levels.
- The company hierarchy is a tree. If the direct boss of an employee is present, their fun level drops to zero.
Solution

- For a **vertex** \( v \) in the tree:
  - \( s^+(v) \) – Max sum of fun levels in subtree whose root is \( v \), when \( v \) is invited.
  - \( s^-(v) \) – Max sum of fun levels in subtree whose root is \( v \), when \( v \) is not invited.

- The solution to the problem is \( \max\{s^+(r), s^-(r)\} \), when \( r \) is the vertex of the CEO.

Example

\[
\begin{align*}
  s^-(r) &= 37 \\
  s^+(r) &= 29
\end{align*}
\]
Solving the Subproblems

- \( f_{\text{un}}(v) \) – fun level of \( v \).
- If \( v \) is a leaf of the tree:
  \[
  s^+(v) = f_{\text{un}}(v) \\
  s^-(v) = 0
  \]
- \( T(v) \) – the set of direct children of \( v \).
- If \( v \) is not a leaf:
  \[
  s^+(v) = f_{\text{un}}(v) + \sum_{u \in T(v)} s^-(u) \\
  s^-(v) = \sum_{u \in T(v)} \max\{s^+(u), s^-(u)\}
  \]

Running Time

- \( n \) employees in company.
- We compute \( s^+(v) \) and \( s^-(v) \) for every employee, starting at the bottom, and gradually going higher.
  - Computing one \( s^+(v) \) takes \( O(n) \) time.
  - This leads to a total running time of \( O(n^2) \).
  - We can do better!
Improved Running Time

- \( n \) employees in company.
- We compute \( s^+(v) \) and \( s^-(v) \) for every employee, starting at the bottom, and gradually going higher.
  - Each \( s^+(v) \) and \( s^-(v) \) appear only in the calculation of the two values of the direct parent of \( v \).
  - Overall, computing all of the \( s^+(v) \) and \( s^-(v) \) values takes a total of \( O(n) \) time.

Those who do not remember the past are condemned to repeat it.

George Santayana

(Use Dynamic Programming!)
Travelling Salesman Problem

- **Problem.** Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Example

- The shortest trip that visit 50 famous U.S. landmarks.
Formalizing the Problem

• From an algorithmic perspective, how can we think of the problem?
  ◦ Build a graph.
  ◦ Vertices are cities.
  ◦ Directed graph with edge from every city to every other city.
  ◦ Weights are the cost / distance / time (quantity we wish to minimize).

Find the Cheapest Trip
The Naïve Solution

- There are \((n - 1)!\) permutations of the cities starting at city 1. Check each path takes \(O(n)\) operation. So \(O(n!)\).

\[
(1,2,3,4,5,6) \rightarrow \text{cost} = 19 \\
(1,2,3,4,6,5) \rightarrow \text{cost} = 32 \\
(1,2,3,5,4,6) \rightarrow \text{cost} = 28 \\
(1,2,3,5,6,4) \rightarrow \text{cost} = 15 \\
(1,2,3,6,4,5) \rightarrow \text{cost} = 22 \\
(1,2,3,6,5,4) \rightarrow \text{cost} = 19 \\
\ldots
\]

Using Dynamic Programming

- For a subset \(A\) of \(\{2,3,\ldots,n\}\) and an integer \(2 \leq k \leq n\), let \(c(A,k)\) be the lightest path starting at 1, passing exactly through the cities of \(A\), and ending at \(k\).
  - Assume that \(k\) is not in \(A\).
  - \(p(\ell,k)\) – cost of direct travel from \(\ell\) to \(k\).
  - Need to find \(\min_k\{c(\{2,3,\ldots,n\}\setminus\{k\},k) + p(k,1)\}\).
Notation: Set Difference

• In \{2,3, ..., n\}\{k\}, the symbol ‘\’ is called set difference.
  ◦ Removing the number \(k\) from \{2,3, ..., n\}.
  ◦ For example, when \(k = 3\), we have \{2,3, ..., n\}\{3\} = \{2,4,5, ..., n\}.

\[
A \setminus B
\]

Discuss in Groups

• Problem.
  ◦ Now that we defined the subproblems, complete the dynamic programming alg.
  ◦ What is the running time? (very large!)

• For a subset \(A\) of \{2,3, ..., n\} and an integer \(2 \leq k \leq n\), let

\[
c(A, k)
\]

be the shortest path starting at 1, passing exactly through the cities of \(A\), and ending at \(k\).
The Algorithm

- $p(\ell, k)$ – cost of direct travel from $\ell$ to $k$.

$$c(\{\}, k) = p(1, k)$$

$$c(A, k) = \min_{i \in A} \{ c(A \setminus \{i\}, i) + p(i, k) \}$$

- First compute $c(A, k)$ for all sets $A$ of size one, then of size two, and so on...
  - We compute $2^{n-1}$ values of $c(A, k)$.
  - Each value requires $O(n)$ time.
  - Total running time: $O(2^n \cdot n^2)$.

Faster Algorithms?

- Using dynamic programming, we obtained a running time of $O(2^n \cdot n^2)$.
- It is not known whether an algorithm with running time $O(1.99^n \cdot n^{100})$ exists.
- Finding a polynomial time algorithm or proving that such an algorithm does not exist would get you $1,000,000$!
The Millennium Prize Problems

- P versus NP.
- The Hodge conjecture.
- The Poincaré conjecture. Solved!
- The Riemann hypothesis.
- Yang–Mills existence and mass gap.
- Navier–Stokes existence and smoothness.
- The Birch and Swinnerton-Dyer conjecture.

The End

<table>
<thead>
<tr>
<th>Brute-Force Solution: O(n!)</th>
<th>Dynamic Programming Algorithms: O(n^2 2^n)</th>
<th>Selling on eBay: O(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Brute-Force Solution" /></td>
<td><img src="image2.png" alt="Dynamic Programming Algorithms" /></td>
<td><img src="image3.png" alt="Selling on eBay" /></td>
</tr>
</tbody>
</table>

Still working on your route?

SHUT THE HELL UP!