Other Dynamic Programming Problems

View in slide-show mode
Problem 1
Subset Sum
Subset-Sum Problem

Given:

- a set of integers \( X = \{x_1, x_2, \ldots, x_n\} \), and
- an integer \( B \)

Find:

- a subset of \( X \) that has maximum sum not exceeding \( B \).

Notation: \( S_{n,B} = \{x_1, x_2, \ldots, x_n : B\} \) is the subset-sum problem

- The integers to choose from: \( x_1, x_2, \ldots, x_n \)
- Desired sum: \( B \)
Subset-Sum Problem

**Example:**

Let $S_{12,99} = \{20, 30, 14, 70, 40, 50, 15, 25, 80, 60, 10, 95: 99\}$.

Find a subset of $X$ with maximum sum not exceeding 99.

An optimal solution:

$$N_{\text{opt}} = \{20, 14, 40, 25\}$$

with sum $20 + 14 + 40 + 25 = 99$. 
Consider the solution as a sequence of $n$ decisions:

$$i^{th} \text{ decision}: \text{ whether we pick number } x_i \text{ or not}$$

Let $N_{\text{opt}}$ be an optimal solution for $S_{n,B}$.

Let $x_k$ be the highest-indexed number in $N_{\text{opt}}$.

$N_{\text{opt}}' = N_{\text{opt}} - \{x_k\}$
Optimal Substructure Property

**Lemma**: \( N'_\text{opt} = N_{\text{opt}} - \{x_k\} \) is an optimal solution for the subproblem \( S_{k-1,B-x_k} = \{x_1, x_2, \ldots, x_{k-1} : B-x_k\} \)

and

\[
c(N_{\text{opt}}) = x_k + c(N'_{\text{opt}})
\]

where \( c(N) \) is the sum of all numbers in subset \( N \)
**Optimal Substructure Property - Proof**

**Proof**: By contradiction, assume that there exists another solution $A'$ for $S_{k-1, B-x_k}$ for which:

\[ c(A') > c(N'_\text{opt}) \text{ and } c(A') \leq B - x_k \]

*i.e. $A'$ is a better solution than $N'_\text{opt}$ for $S_{k-1, B-x_k}$*

Then, we can construct $A = A' \cup \{x_k\}$ as a solution to $S_{k, B}$.

We have:

\[ c(A) = c(A') + x_k \]

\[ > c(N'_\text{opt}) + x_k = c(N_{\text{opt}}) \]

Contradiction! $N_{\text{opt}}$ was assumed to be optimal for $S_{k,B}$.

Proof complete.
Optimal Substructure Property - Example

Example:

\[ S_{12,99}: \{20, 30, 14, 70, 40, 50, 15, 25, 80, 60, 10, 95: 99\} \]

\[ N_{\text{opt}} = \{20, 14, 40, 25\} \text{ is optimal for } S_{12,99} \]

\[ N'_{\text{opt}} = N_{\text{opt}} - \{x_8\} = \{20, 14, 40\} \text{ is optimal for the subproblem } S_{7,74} = \{20, 30, 14, 70, 40, 50, 15: 74\} \]

and

\[ c(N_{\text{opt}}) = 25 + c(N'_{\text{opt}}) = 25 + 74 = 99 \]
Recursive Definition an Optimal Solution

\[ c[i, b] \text{: the value of an optimal solution for } S_{i,b} = \{ x_1, \ldots, x_i : b \} \]

\[
\begin{align*}
C_i, b &= C_{i-1}, b, C_{i-1}, b-x_i + x_i \\
\text{if } i = 0 \text{ or } b = 0 \\
c[i, b] &= c[i-1, b] \quad \text{if } x_i > b \\
C_i, b &= \text{Max} \{ x_i + c[i-1, b-x_i], c[i-1, b] \} \quad \text{if } i > 0 \text{ and } b \geq x_i 
\end{align*}
\]

According to this recurrence, an optimal solution \( N_{i,b} \) for \( S_{i,b} \):

- **either contains** \( x_i \) \( \Rightarrow c(N_{i,b}) = x_i + c(N_{i-1,b-x_i}) \)
- **or does not contain** \( x_i \) \( \Rightarrow c(N_{i,b}) = c(N_{i-1,b}) \)
Subset Sum Problem: A special case of 0/1 Knapsack Problem

\[ C_i b = \max \left\{ x_i + C_{i-1, b-x_i}, C_{i-1, b} \right\} \]
\[ C_i b = \max \left\{ x_i + C_{i-1}, w - w_i \right\}, C_{i-1, w} \]

Subset Sum: \( S_{n, b} = \sum x_i, x_i \leq x_n = b \)

0/1 Knapsack: \( K_{n, W} = \sum \omega_i, \omega_i = W \)

\[ \omega_i \text{ weight of article } i \]

\[ v_i = \omega_i / \omega_j \text{ unless } \]
Need to process: $c[i, b]$ after computing: $c[i - 1, b], c[i - 1, b - x_i]$
\[ c[i, b] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } b = 0 \\
c[i-1, b] & \text{if } x_i > b \\
\max \{ x_i + c[i-1, b-x_i], c[i-1, b] \} & \text{if } i > 0 \text{ and } b \geq x_i 
\end{cases} \]
Computing the Optimal Subset-Sum Value

SUBSET-SUM \((x, n, B)\)

\[
\begin{align*}
&\text{for } b \leftarrow 0 \text{ to } B \text{ do} \\
&\quad c[0, b] \leftarrow 0 \\
&\quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
&\quad\quad c[i, 0] \leftarrow 0 \\
&\quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
&\quad\quad \text{for } b \leftarrow 1 \text{ to } B \text{ do} \\
&\quad\quad\quad \text{if } x_i \leq b \text{ then} \\
&\quad\quad\quad\quad c[i, b] \leftarrow \text{Max}\{x_i + c[i-1, b-x_i], c[i-1, b]\} \\
&\quad\quad\quad \text{else } x_i > b \\
&\quad\quad\quad c[i, b] \leftarrow c[i-1, b] \\
&\quad \text{return } c[n, B]
\end{align*}
\]
Finding an Optimal Subset

SOLUTION-SUBSET-SUM \((x, b, B, c)\)

\[
i \leftarrow n \\
b \leftarrow B \\
N \leftarrow \emptyset \\
\text{while } i > 0 \text{ do} \\
\quad \text{if } c[i, b] = c[i-1, b] \text{ then} \\
\quad \quad i \leftarrow i - 1 \\
\quad \text{else} \\
\quad \quad c[i, b] \neq c[i-1, b] \\
\quad \quad N \leftarrow N \cup \{x_i\} \\
\quad \quad i \leftarrow i - 1 \\
\quad \quad b \leftarrow b - x_i \\
\text{return } N
\]
Number Partitioning Problem

Given a set of n integers \( X = \{x_1, x_2, \ldots, x_n\} \)

Partition \( X \) into two subsets \( \Pi(X) = \{A, B\} \), where

- \( A, B \subseteq X \)
- \( A \cap B = \emptyset \) (mutually disjoint)
- \( A \cup B = X \) (mutually exhaustive)
- \( \exists \) no elt replication
- \( \exists \) no elt in such a way that

\[
\left| \sum_{x_i \in A} x_i - \sum_{x_j \in B} x_j \right| \text{ is minimized}
\]
How can we solve the number partitioning problem as an instance of the subset-sum problem?

\[ \text{totSum} = \sum_{x_i \in X} x_i, \quad B = \left\lfloor \frac{\text{totSum}}{2} \right\rfloor \]

\[ \text{Solve:} \quad S_0, B = \sum x_1, x_2, \ldots, x_n : B = \left\lfloor \frac{\text{totSum}}{2} \right\rfloor \]

Let \( N_{opt} \subseteq X \) be an opt. \( S_0 \). \( B \) this

\[ \text{opt} \left( f(X) \right) = \sum_{N_{opt} \subseteq X} |X - N_{opt}| \]

bipartition

\[ \min \sum_{x_i \in A} x_i - \sum_{x_i \in B} x_i \rightarrow \text{balanced partition as much as possible} \]
Subset Sum  \( \text{input instance size} = n + d \)!

- Running Time
  \( T(n) = \Theta(n^B) \)

- Input instance
  \( X = \{1, 3, 2, 10^6\} \)

- Value of \( d \)
  \( d = 2^{10} \)

- Pseudo-polynomial
  \( T(n, 1) = \Theta(3n10^6) \)
Problem 2
Optimal Binary Search Tree
Reminder: Binary Search Tree (BST)

All keys in the left subtree less than 8

All keys in the right subtree greater than 8

This property holds for all nodes.
Example: English-to-French translation

Organize (English, French) word pairs in a BST

- **Keyword:** English word
- **Satellite data:** French word

We can search for an English word (node key) efficiently, and return the corresponding French word (satellite data).
Binary Search Tree Example

Suppose we know the frequency of each keyword in texts:

\[ k_1, k_2, k_3, k_4, k_5, k_6, k_7 \]

5% 40% 8% 4% 10% 10% 23%

\[ k_1 < k_2 < k_3 < k_4 \]
Cost of a Binary Search Tree

Example: If we search for keyword “while”, we need to access 3 nodes. So, 23% of the queries will have cost of 3.

Total cost = \( (\text{depth}(i)+1) \times \text{freq}(i) \)

\[
\sum_{i} = 1 \times 0.04 + 2 \times 0.4 + 2 \times 0.1 + 3 \times 0.05 + 3 \times 0.08 + 3 \times 0.1 + 3 \times 0.23 = 2.42
\]
A different binary search tree (BST) leads to a different total cost:

\[
\text{Total cost} = 1 \times 0.4 + 2 \times 0.05 + 2 \times 0.23 + 3 \times 0.1 + 4 \times 0.08 + 4 \times 0.1 + 5 \times 0.04
\]

\[= 2.18\]

This is in fact an optimal BST.
Optimal Binary Search Tree Problem

Given:

A collection of \( n \) keys \( K_1 < K_2 < \ldots K_n \) to be stored in a BST.
The corresponding \( p_i \) values for \( 1 \leq i \leq n \)
\( p_i \): probability of searching for key \( K_i \)

Find:

An optimal BST with minimum total cost:

\[
\text{Total cost} = \sum_{i} (\text{depth}(i)+1) \times \text{freq}(i)
\]

Note: The BST will be static. Only search operations will be performed. No insert, no delete, etc.
Problem 7
\[
K_1 < K_2 < \cdots < K_n
\]

\[
P_1 \quad P_2 \quad \cdots \quad P_n
\]

Key:
10 < 20 < 30 < 40 < 50 < 60 < 70 < 80 < 90

Matrix Chain:
\[
A_{1 \cdots n} = (A_1 \ A_2 \cdots \ A_k \ \underbrace{A_{k+1} \cdots \ A_n})^S
\]
Optimal BST for $T_m$: $T_{m+1,n}$ should be an optimal BST for $K_1 < K_2 < K_3 < \ldots < K_m < K_{m+1}$.

Optimal substructure property: $T_{i,m-1}$ for $K_i < K_{i+1} < \ldots < K_m$.
Let $T_{in}$ be an optimal BST for $K_{i} \leq C < K_{m}$.

\[
\text{cost}(T_{in}) = \text{cost}(T_{m}, m-1) + \text{cost}(T_{m+1}, n) + \sum_{h=1}^{n} P_h
\]
Lemma 1: Let $T_{ij}$ be a BST containing keys $K_i < K_{i+1} < \ldots < K_j$. Let $T_L$ and $T_R$ be the left and right subtrees of $T$. Then we have:

\[
\text{cost}(T_{ij}) = \text{cost}(T_L) + \text{cost}(T_R) + \sum_{h=i}^{j} p_h
\]

Intuition: When we add the root node, the depth of each node in $T_L$ and $T_R$ increases by 1. So, the cost of node $h$ increases by $p_h$. In addition, the cost of root node $r$ is $p_r$. That’s why, we have the last term at the end of the formula above.
Lemma 2: Optimal substructure property

Consider an optimal BST $T_{ij}$ for keys $K_i < K_{i+1} < \ldots < K_j$

Let $K_m$ be the key at the root of $T_{ij}$

Then:

- $T_{i,m-1}$ is an optimal BST for subproblem containing keys: $K_i < \ldots < K_{m-1}$
- $T_{m+1,j}$ is an optimal BST for subproblem containing keys: $K_{m+1} < \ldots < K_j$

$$\text{cost}(T_{ij}) = \text{cost}(T_{i,m-1}) + \text{cost}(T_{m+1,j}) + \sum_{h=i}^{j} p_h$$
recursive formula

\[ T_1 = \begin{cases} T_{1, n-1} & \text{if } n > 1 \\ \frac{n}{2} & \text{otherwise} \end{cases} \]

\[ C_n = \min_{1 \leq r \leq n} \left( C_{1, r-1} + C_{r+1, n} + \sum_{h=1}^{n} P_h \right) \]

\[ \text{cost}(T_1) = \text{cost}(T_{1, n-1}) + \text{cost}(T_{r+1, n}) + \sum_{h=1}^{n} P_h \]
Recursive Formulation

Note: We don’t know which root vertex leads to the minimum total cost. So, we need to try each vertex $m$, and choose the one with minimum total cost.

$c[i, j]$: cost of an optimal BST $T_{ij}$ for the subproblem $K_i < \ldots < K_j$

\[
c[i, j] = \begin{cases} 
0 & \text{if } i > j \\
\min_{i \leq r \leq j} \{c[i, r - 1] + c[r + 1, j] + P_{ij}\} & \text{otherwise}
\end{cases}
\]

where $P_{ij} = p_h$ for $h = i$.

$C_{ij} = \min_{i \leq k < j} \{C_{ik} + C_{k+1,j} + P_{ih} P_{kj}\}$
Bottom-up computation

\[
c[i, j] = \begin{cases} 
0 & \text{if } i > j \\
\min_{i \leq r \leq j} \{c[i, r - 1] + c[r + 1, j] + P_{ij}\} & \text{otherwise}
\end{cases}
\]

How to choose the order in which we process \(c[i, j]\) values?

Before computing \(c[i, j]\), we have to make sure that the values for \(c[i, r-1]\) and \(c[r+1,j]\) have been computed for all \(r\).
\[ c[i, j] = \begin{cases} 
0 & \text{if } i > j \\
\min_{i \leq r \leq j} \{c[i, r-1] + c[r+1, j] + P_{ij}\} & \text{otherwise}
\end{cases} \]
\[ c[i, j] = \begin{cases} 0 & \text{if } i > j \\ \min_{i \leq r \leq j} \{c[i, r-1] + c[r+1, j] + P_{ij}\} & \text{otherwise} \end{cases} \]

If the entries \( c[i,j] \) are computed in the shown order, then \( c[i,r-1] \) and \( c[r+1,j] \) values are guaranteed to be computed before \( c[i,j] \).
Computing the Optimal BST Cost

**COMPUTE-OPTIMAL-BST-COST** \((p, n)\)

\[
\text{for } i \leftarrow 1 \text{ to } n+1 \text{ do } \\
c[i, i-1] \leftarrow 0
\]

\[
\text{PS}[1] \leftarrow p[1] \quad \text{ // PS}[i]: \text{prefix}_\text{sum}(i): \text{Sum of all } p[j] \text{ values for } j \leq i
\]

\[
\text{for } i \leftarrow 2 \text{ to } n \text{ do } \\
\quad \text{PS}[i] \leftarrow p[i] + \text{PS}[i-1] \quad \text{ // compute the prefix sum}
\]

\[
\text{for } d \leftarrow 0 \text{ to } n-1 \text{ do } \\
\quad \text{for } i \leftarrow 1 \text{ to } n - d \text{ do } \\
\quad \quad j \leftarrow i + d \\
\quad \quad c[i, j] \leftarrow \infty
\]

\[
\text{for } r \leftarrow i \text{ to } j \text{ do } \\
\quad c[i, j] \leftarrow \min\{c[i, j], c[i, r-1] + c[r+1, j] + \text{PS}[j] - \text{PS}[i-1]\}
\]

\text{return } c[1, n]
Note on Prefix Sum

- We need \( P_{ij} \) values for each \( i, j \) (\( 1 \leq i \leq n \) and \( 1 \leq j \leq n \)),

  where:
  \[
  P_{ij} = \sum_{h=i}^{j} p_h
  \]

- If we compute the summation directly for every \((i, j)\) pair, the total runtime would be \( \Theta(n^3) \).

- Instead, we spend \( O(n) \) time in preprocessing to compute the prefix sum array \( PS \). Then we can compute each \( P_{ij} \) in \( O(1) \) time using \( PS \).
Note on Prefix Sum

In preprocessing, compute for each $i$:

$$PS[i] \text{: the sum of } p[j] \text{ values for } 1 \leq j \leq i$$

Then, we can compute $P_{ij}$ in $O(1)$ time as follows:

$$P_{ij} = PS[i] - PS[j-1]$$

**Example:**

```
1  2  3  4  5  6  7  8
p: 0.05 0.02 0.06 0.07 0.20 0.05 0.08 0.02
PS: 0.05 0.07 0.13 0.20 0.40 0.45 0.53 0.55
```

$$P_{27} = PS[7] - PS[1] = 0.53 - 0.05 = 0.48$$

$$P_{36} = PS[6] - PS[2] = 0.45 - 0.07 = 0.38$$