Honors Algorithms
G22.3520-001 Fall 2006

Lecture 8
Read: CLRS 15
Dynamic Programming

A way of speeding up certain recursive algorithms
Idea: don’t compute the same thing twice!

Example: Subset Sum

Given: positive integers $a_1, \ldots, a_n$, and a “target” value $t$

Question: is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = t$?

Predicate: $P(\langle a_1, \ldots, a_n \rangle, t)$
\[ P(\langle a_1, \ldots, a_n \rangle, t) \iff \]
\[
\begin{cases}
(n = 0) \land (t = 0) \\
\lor \\
(n > 0) \land (a_n \leq t) \land P(\langle a_1, \ldots, a_{n-1} \rangle, t - a_n) \\
\lor \\
(n > 0) \land P(\langle a_1, \ldots, a_{n-1} \rangle, t)
\end{cases}
\]

Recursion Tree:

\[
\begin{array}{c}
\text{height} = n \\
\text{size of tree} = 2^n
\end{array}
\]
Running time is exponential in $n$

Observe: the number of *unique* problem instances is at most $(n + 1)(t + 1)$, which is much smaller (assuming $t$ is not too large)

Idea: maintain a table of results $T[i, s]$, where $i = 0 \ldots n$ and $s = 0 \ldots t$

Initially, $T[i, s] = \perp$ for all $i, s$

As recursion proceeds, $T[i, s]$ is set to $P(\langle a_1, \ldots, a_i \rangle, s)$

If a sub-problem has already been solved, fetch value from table, and skip recursion

Total time $= O(nt)$

We can also modify algorithm to output the set $S$
The “subproblem graph”

Iterative implementation: evaluate top to bottom / left to right
Example: Longest Common Subsequence

$X = \langle x_1, \ldots, x_m \rangle$

$Z = \langle z_1, \ldots, z_k \rangle$

$Z$ is a subsequence of $X$ is there exist indices $i_1 < i_2 < \cdots < i_k$ such that $x_{i_j} = z_j$ for $j = 1 \ldots k$

$X = \langle a, b, c, b, d, a, b \rangle$

$Z = \langle b, c, d, b \rangle$

Given sequences $X$ and $Y$, we say $Z$ is a common subsequence of $X$ and $Y$ if $Z$ is a subsequence of $X$ and $Z$ is a subsequence of $Y$
Example: \( X = \langle a, b, c, b, d, a, b \rangle, \ Y = \langle b, d, c, a, b, a \rangle \)
\( Z = \langle b, c, b, a \rangle \)

\[
X = \langle a, b, c, b, d, a, b \rangle
\]

\[
Y = \langle b, d, c, a, b, a \rangle
\]

Problem: given \( X \) and \( Y \), find a Longest Common Subsequence (LCS) of \( X \) and \( Y \)

Notation: for \( X = \langle x_1, \ldots, x_m \rangle \) and \( i = 0 \ldots m \), we define \( [X]_i := \langle x_1, \ldots, x_i \rangle \)
Key observation: Let $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$, and let $Z = \langle z_1, \ldots, z_k \rangle$ be an LCS of $X$ and $Y$

1. $x_m = y_n \implies z_k = x_m$ and $[Z]_{k-1}$ is an LCS of $[X]_{m-1}$ and $[Y]_{n-1}$

2. $(x_m \neq y_n) \land (z_k \neq x_m) \implies Z$ is an LCS of $[X]_{m-1}$ and $Y$

3. $(x_m \neq y_n) \land (z_k \neq y_n) \implies Z$ is an LCS of $X$ and $[Y]_{n-1}$

---

$X = < a, b, c, b, d, a, b>$ $X = < a, b, c, b, d, a, b>$

$Y = < b, d, c, a, b, a, b, c, b>$ $Y = < b, d, c, a, b, a>$
Recursive algorithm $LCS(X, Y)$:

if $m = 0$ or $n = 0$ then
    return empty sequence
else if $x_m = y_n$ then
    return $LCS([X]_{m-1}, [Y]_{n-1}) \| x_m$
else
    $Z_1 \leftarrow LCS([X]_{m-1}, Y)$
    $Z_2 \leftarrow LCS(X, [Y]_{n-1})$
    if $\text{length}(Z_1) > \text{length}(Z_2)$ then
        return $Z_1$
    else
        return $Z_2$
Correctness: observation and induction on $m + n$

- Assume algorithm is correct on all inputs of length $< m + n$
- Consider an LCS $Z$ for the input
- Use induction hypothesis and observation to show that the algorithm constructs a solution of the same length as $Z$

There are only $(m + 1)(n + 1)$ distinct subproblems: $([X]_i, [Y]_j)$, for $i = 0 \ldots m$ and $j = 0 \ldots n$

Implement using a table — running time $= O(mn)$
The subproblem graph

Iterative implementation: evaluate top to bottom / left to right
Example: Optimum Weighted Trees

Definitions

Let $a_1 < a_2 < \cdots < a_n$

Set $a_0 := -\infty$ and $a_{n+1} := +\infty$

An weight assignment is a tuple

$$(\beta_0, \alpha_1, \beta_1, \ldots, \alpha_n, \beta_n)$$

of real numbers

Interpretation: for $i = 1 \ldots n$, $\alpha_i$ may be the probability of accessing $a_i$, and for $j = 0 \ldots n$, $\beta_j$ may be the probability of accessing an element in the interval $b_j := (a_j, a_{j+1})$
Let $T$ be a binary search tree for $a_1, \ldots , a_n$

- internal nodes labeled by items $a_i$
- leaves labeled by intervals $b_j$
- in-order traversal of tree yields $b_0, a_1, b_1, \ldots , a_n, b_n$

For a node $v$, let $d(v)$ denote its depth in the tree

Define the cost of $T$:

$$C := \sum_{i=1}^{n} \alpha_i (1 + d(a_i)) + \sum_{j=0}^{n} \beta_j d(b_j)$$

If the weights are probabilities, $C$ represents the expected number of comparisons performed to look up a random element
Example:

\[
C = 2 \frac{1}{6} + 2 \frac{1}{24} + 3 \frac{1}{8} + 1 \frac{1}{8} + 2 \frac{1}{8} + 2 \frac{5}{12} = 2
\]

**Goal:** given weights, construct a binary search tree that minimizes \( C \)
Consider a tree $T$ with root $a_k$

The left subtree $T_L$ contains $b_0, a_1, b_1, \ldots, a_{k-1}, b_{k-1}$

Let $C_L$ be the cost of $T_L$

The right subtree $T_R$ contains $b_k, a_{k+1}, b_{k+1}, \ldots, a_n, b_n$

Let $C_R$ be the cost of $T_R$

Let $S := \beta_0 + \alpha_1 + \beta_1 + \cdots + \alpha_n + \beta_n$

Key observation:

$$C = S + C_L + C_R$$
For $1 \leq k \leq \ell \leq n$, let

$$S(k, \ell) := \beta_{k-1} + \alpha_k + \beta_k + \cdots + \alpha_\ell + \beta_\ell$$

and define $C(k, \ell)$ to be the optimum cost for $b_{k-1}, \alpha_k, b_k, \ldots, \alpha_\ell, b_\ell$

We have

$$S(k, \ell) = S(k, \ell - 1) + \alpha_\ell + \beta_\ell$$

$$C(k, \ell) = S(k, \ell) + \min_{k \leq m \leq \ell} \left( C(k, m - 1) + C(m + 1, \ell) \right)$$

where

$$C(k, k - 1) := C(\ell + 1, \ell) := 0$$

and

$$S(k, k - 1) := \beta_{k-1}$$
The subproblem graph:

\[ \# \text{nodes} = O(n^2), \ # \text{edges per node} = O(n) \quad \Rightarrow \quad \text{running time} = O(n^3) \]
General dynamic programming strategies:

- Formulate a recursive solution
- Identify subproblems and dependencies
- Analyze structure of subproblem graph
- Iterative solution: determine a convenient "topological order" for the nodes in the graph
- Running time analysis: (usually) equals the size of the subproblem graph (# nodes + # edges)
Some subtleties:

- Must choose subproblems so that:
  - the correct/optimal solution can be reconstructed
    * for Subset Sum and Optimum Weighted Trees, this was easy
    * for LCS, some care was necessary to ensure the optimal solution was not missed
  - the number of distinct subproblems does not explode exponentially
Example: variations on “tiling”

We are given “target” \( t \in \{0, 1\}^* \), and several “tiles” \( s_1, \ldots, s_k \in \{0, 1\}^* \)

Can we \( t \) be obtained by concatenating some of the tiles?

Variations:

- repetitions, given order (easy)
- repetitions, any order (easy)
- no repetitions, given order (easy)
- no repetitions, any order (???)