Longest Common Subsequence and Longest Increasing Subsequence

Longest common subsequence

- Subsequence is not a substring! Example: CALORIA is subsequence of CALIFORNIA but not substring.
- **LCS Problem.** Given two strings $S_1$ and $S_2$, find a *longest common subsequence (lcs)* of $S_1$ and $S_2$, i.e. a longest string $S$ that is subsequence of both $S_1$ and $S_2$.
- Example:
  
  $\text{FRDRTRGESRXDGWAQWSDF}$
  $\text{OJMTKLEBKMJHAOPKS}$
  Longest common subsequence is TEXAS
- **Theorem.** With a scoring scheme that scores 1 for each match and 0 for each mismatch or space, the matched characters in alignment of maximum value form a longest common subsequence.
  
  Thus, lcs can be computed in $O(mn)$ time.

Longest increasing sequence

- Let $P$ be a list of $n$ integers, not necessarily distinct. An *increasing subsequence* of $P$ is a subsequence of $P$ whose values strictly increase from left to right.
- Example: $P = (5, 3, 4, 9, 6, 2, 1, 8, 7, 10)$ then $(3, 4, 6, 8, 10)$ and $(5, 9, 10)$ are both increasing subsequences in $P$.
- **LIS Problem:** given $P$, find its longest increasing subsequence.
- A *decreasing subsequence* of $P$ is a subsequence of $P$ where the numbers are nonincreasing from left to right.
- Example: $(8, 5, 5, 3, 1, 1)$ is a decreasing subsequence of $(4, 8, 3, 9, 5, 2, 5, 3, 10, 1, 9, 1, 6)$.
- Note the asymmetry in definitions of increasing and decreasing subsequences.

Cover

- A *cover* of $P$ is a set of decreasing subsequences of $P$ that contain all the numbers of $P$.
- Example: $(5, 3, 2, 1), (4), (9, 6), (8, 7), (10)$ is a cover of $P = (5, 3, 4, 9, 6, 2, 1, 8, 7, 10)$.
- The *size* of a cover is the number of its subsequences.
- A smallest cover of $P$ is a cover with minimum size among all covers of $P$. 

Greedy cover

**GreedyCover(P)**

// input: a list P of n integers
// output: a cover C
C = {(P(1))} // C has one sequence with the first number
for i = 2 to n
    Let s_1, s_2, .. be the sequences stored in C.
    Find the first sequence s_j such that P(i)
    can be added to it. Add P(i) to s_j.
    If s_j is not found, then
    create new subsequence {P(i)}.

Example

\[ P = (5, 3, 4, 9, 6, 2, 1, 8, 7, 10). \]
\[ s_1 = (5) \]
\[ s_1 = (5, 3) \]
\[ s_1 = (5, 3), s_2 = (4) \]
\[ s_1 = (5, 3), s_2 = (4), s_3 = (9) \]
\[ s_1 = (5, 3), s_2 = (4), s_3 = (9, 6) \]
\[ s_1 = (5, 3, 2), s_2 = (4), s_3 = (9, 6) \]
\[ s_1 = (5, 3, 2, 1), s_2 = (4), s_3 = (9, 6) \]
\[ s_1 = (5, 3, 2, 1), s_2 = (4), s_3 = (9, 6), s_4 = (8) \]
\[ s_1 = (5, 3, 2, 1), s_2 = (4), s_3 = (9, 6), s_4 = (8, 7) \]
\[ s_1 = (5, 3, 2, 1), s_2 = (4), s_3 = (9, 6), s_4 = (8, 7), s_5 = (10) \]

Analysis

• Greedy cover can be found in \(O(n^2)\) time.

• To place \(i\)-th number we check \(k\) sequences stored in \(C\).
  \(k < i\), so time = 1 + 2 + .. + \(n = O(n^2)\).

Longest increasing subsequence

• **Theorem.** Let \(k\) be the size of the greedy cover of \(P\).
  There is an increasing subsequence of size \(k\).
  Any increasing subsequence of \(P\) has size \(\leq k\).

• For each number in \(s_i\) there is a smaller number in \(s_{i-1}\). Repeat this, starting with any number in \(s_k\).

• An increasing sequence cannot contain two numbers in one subsequence \(s_i\).
LIS($P$)
// input: a list $P$ of $n$ integers
// output: a longest increasing subsequence $I$
Let $C = \{s_1, s_2, ..., s_k\}$ be the greedy cover.
Set $I$ to the empty list.
$i = k$. Place any number $x$ from $s_k$ in $I$.
for $i = k - 1$ to 1 by $-1$
    find the first number $y$ in $s_i$ smaller than $x$
    place $y$ in front of $I$
$x = y$

Example

$P = (5, 3, 4, 9, 6, 2, 1, 8, 7, 10)$.
$s_1 = (5, 3, 2, 1), s_2 = (4), s_3 = (9, 6), s_4 = (8, 7), s_5 = (10)$
$I = (10)$
$I = (8, 10)$
$I = (6, 8, 10)$
$I = (4, 6, 8, 10)$
$I = (3, 4, 6, 8, 10)$

Analysis

- Longest increasing subsequence can be extracted from the greedy cover in $O(n)$ time.

Faster construction of greedy cover

- Let $n_i$ be the last element of $s_i$.
  Then $n_1, n_2, ..., n_k$ is increasing subsequence of $P$.
- A new number $P(i)$ either
  - creates new subsequence in $C$ or
  - replaces $n_j$ if $n_j \leq P(i) < n_{j+1}$.
- To locate $n_j$ we apply binary search.
- Runtime is $O(n \log n)$.

LCS reduces to LIS

- LCS is longest common subsequence.
- LIS is longest increasing subsequence.
- Input: String $S_1$ of size $n$ and string $S_2$ of size $m$ over alphabet $\Sigma$.
- Let $r(i)$ be the number of times $i$-th character of $S_1$ appears in $S_2$. Let $r = r(1) + r(2) + .. + r(m)$. 
• Replace $S_1(i)$ with numbers $n_1, n_2, ..., n_{r(i)}$ that are indices of $S_1(i)$ in $S_2$ in decreasing order.

• The length is $r \leq mn$.

Example and reduction

• $S_1 = \text{acabd}, S_2 = \text{caacba}$.

• $r(1) = 3, r(2) = 2, r(3) = 3, r(4) = 1, r(5) = 0, r = 9$.

• $a = (6, 3, 2), b = (5), c = (4, 1), d = ()$.

• $P = (6, 3, 2, 4, 1, 6, 3, 2, 5)$.

• **Theorem.** Every increasing subsequence in $P$ specifies an equal length common subsequence of $S_1$ and $S_2$ and vice versa. Thus LCS of $S_1$ and $S_2$ corresponds to LIS of $P$.

• $P = (6, 3, 2, 4, 1, 6, 3, 2, 5), S_1 = \text{acabd}, S_2 = \text{caacba}$.

• $P = (6, 3, 2, 4, 1, 6, 3, 2, 5), S_1 = \text{acabd}, S_2 = \text{caacba}$.

• **Theorem.** Longest common substring can be found in $O(r \log n)$ time.

• Usually $r$ is much less than $mn$.

• First $O(r \log n)$ algorithm was obtained by Hunt and Szymanski 1977.

• Relation between LCS and LIS found several times independently.

• Generalized to solve sparse dynamic programming.