Lecture 16

COL 351: Analysis and Design of Algorithms

Lecture 16

Prefix Suffix problem (Sub-problem for String Matching)

Given: A string $P = [p_1, \ldots, p_k]$ of size k.

Find: A Table of size k satisfying

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

i	1	2	3	4	5	6	7	8	9
P[i]	A	B	C	A	B	C	A	B	C
Table[i]									

Prefix Suffix problem — Lemma

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

Lemma 1: Suppose $L \geq 1$ satisfy that the L-length prefix and suffix of $P[1,\ i]$ are identical.

Then, the length of longest common prefix-suffix of $P[1,\ i]$ of size just smaller than L is "Table[L]".

Proof Sketch:

$$\overbrace{\underbrace{p_1\ \ p_2}_{}\ \ p_3\ \ p_4\ \ p_5\ \ p_6}^{\text{6-length prefix}}\ \ p_7\ \ \cdots\ \overbrace{p_{i-5}\ \ p_{i-4}\ \ p_{i-3}\ \ p_{i-2}\ \ \underbrace{p_{i-1}\ \ p_i}_{}}^{\text{6-length suffix}}$$

Prefix Suffix problem— Lemma

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

Lemma 1: Suppose $L \geq 1$ satisfy that the L-length prefix and suffix of $P[1,\ i]$ are identical.

Then, the length of longest common prefix-suffix of $P[1,\ i]$ of size just smaller than L is "Table[L]".

Proof Sketch:

6-length prefix 6-length suffix

$$\overbrace{\underbrace{p_1\ \ p_2}\ \ p_3\ \ p_4\ \ p_5\ \ p_6}^{\text{6-length prefix}}\ \ p_7\ \ \cdots\ \ \overbrace{p_{i-5}\ \ p_{i-4}\ \ p_{i-3}\ \ p_{i-2}\ \ \underbrace{p_{i-1}\ \ p_i}}^{\text{6-length suffix}}$$

So, Table[i] , Table[Table[i]] , Table[Table[Table[i]]] , …. is sequence of all common prefixes-suffixes of $P[1,i]$.

Prefix Suffix Algorithm— using Dynamic Programming

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

Table ← Array of size k;

Table[1], $L = 0$;

For ($i = 1$ to $k - 1$):

/* value of L is Table[i] */

While ($L > 0$ and P[i +1] $\neq$ P[L+1]): L = Table[L];

If (P[i +1] = P[L+1]): $L = L + 1$;

Table[i +1] = L;

Update L to length of largest common non-trivial Prefix-Suffix of P[1, i] that satisfy P[i +1] = P[L+1].

If no such L exists, then L is just 0.

Prefix Suffix Algorithm — using Dynamic Programming

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

Table ← Array of size k;

Table[1], $L = 0$;

For ($i = 1$ to $k - 1$):

    /* value of L is Table[i] */

    While ($L > 0$ and P[i +1] $\neq$ P[L+1]): L = Table[L];

    If (P[i +1] = P[L+1]): $L = L + 1$;

    Table[i +1] = L;

Analysis of Time-Complexity

Fact 1: L is incremented at most k times.

Fact 2: Throughout the algorithm L can decrease at most k times, so total number of iterations of While loop is at most k.

Therefore, time complexity is O(k).

Main Problem — String Matching

Given: String S = $[s_1, \ldots, s_n]$ and a pattern P = $[p_1, \ldots, p_k]$, represented as arrays of size n, k. (Here $k < n$).

Find: Does there exists a sub-string of S that is identical to P.

Examples:

P = "hash"
S = "cuckoo hashing is efficient"
Yes

P = "hash-table"
S = "cuckoo hashing is efficient"
No

String Matching problem— Lemma

Table[i] := <u>Length of longest (non-trivial) common prefix and suffix of P[$1, i$]</u>

Lemma 2: Suppose $i, L \geq 1$ satisfy that the L-length prefix of P is identical to L-length suffix of $S[1,\ i]$.

Then, the length of longest prefix of P that is also a suffix of $S[1,\ i]$ of size just smaller than L is "Table[L]".

Proof Sketch:

$$\overbrace{\underbrace{p_1 \;\; p_2} \;\; p_3 \;\; p_4 \;\; p_5 \;\; p_6}^{\text{6-length prefix}} \;\; p_7 \;\; \cdots \;\; p_k \qquad s_1 \;\; s_2 \;\; \cdots \;\; s_{i-6} \;\; \overbrace{s_{i-5} \;\; s_{i-4} \;\; s_{i-3} \;\; s_{i-2} \;\; \underbrace{s_{i-1} \;\; s_i}}^{\text{6-length suffix}}$$

Knuth–Morris–Pratt (KMP) algorithm

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

A[i] := Length of longest prefix of P that is also a suffix of S[1, i]

A ← Array of size $n + 1$ with $A[0] = 0$;

$L \leftarrow 0$;

For ($i = 0$ to $n - 1$):

Update L to largest integer satisfying

- L-length prefix of P = L-length suffix of $S[1,i]$
- S[i +1] = P[L+1].

If no such L exists, then L is just 0

/* value of L is A[i] */

While ($L > 0$ and S[i +1] $\neq$ P[L+1]): L = Table[L];

If (S[i +1] = P[L+1]): $L = L + 1$;

A[i +1] = L;

If ($L = k$) Return True;

Return False;

If no such L exists, then L is just 0.

Knuth–Morris–Pratt (KMP) algorithm

Table[i] := Length of longest (non-trivial) common prefix and suffix of P[1, i]

A[i] := Length of longest prefix of P that is also a suffix of S[1, i]

A ← Array of size $n + 1$ with $A[0] = 0$;

$L \leftarrow 0$;

For ($i = 0$ to $n - 1$):

/* value of L is A[i] */

While ($L > 0$ and S[i +1] $\neq$ P[L+1]): L = Table[L];

If (S[i +1] = P[L+1]): $L = L + 1$;

A[i +1] = L;

Analysis of Time-Complexity

Fact 1: L is incremented at most n times.

Fact 2: Throughout the algorithm L can decrease at most n times, so total number of iterations of While loop is at most n.

Therefore, time complexity is O(n).

If ($L = k$) Return True;

Return False;

Single Source Distances in graph with negative edge-weights

Single Source Distance Problem

Given: A directed weighted graph $G = (V, E)$ with possible negative edge weights, and a source s.

Output: Either a Shortest-path-tree rooted at s, or report that G contains a negative cycle reachable from s.

Examples: