Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
فارسی
Français
Polski
Русский
Tiếng Việt
Edit links

Longest common substring problem

From Wikipedia, the free encyclopedia

Not to be confused with longest common subsequence problem.

In computer science, the **longest common substring problem** is to find the longest string (or strings) that is a substring (or are substrings) of two or more strings.

Example [edit]

The longest common substring of the strings "ABABC", "BABCA" and "ABCBA" is string "ABC" of length 3. Other common substrings are "A", "AB", "B", "BA", "BC" and "C".

```
ABABC
  | | |
 BABCA
  | | |
  ABCBA
```

Problem definition [edit]

Given two strings, S of length m and T of length n, find the longest strings which are substrings of both S and T.

A generalisation is the **k-common substring problem**. Given the set of strings $S = \{S_1, ..., S_K\}$, where $|S_i| = n_i$ and $\Sigma n_i = N$. Find for each $2 \leq k \leq K$, the longest strings which occur as substrings of at least k strings.

Algorithms [edit]

One can find the lengths and starting positions of the longest common substrings of S and T in $\Theta(n+m)$ with the help of a generalised suffix tree. Finding them by dynamic programming costs $\Theta(nm)$. The solutions to the generalised problem take $\Theta(n_1 + ... + n_K)$ space and $\Theta(n_1 \cdot ... \cdot n_K)$ time with dynamic programming and take $\Theta(N * K)$ time with generalized suffix tree.

Suffix tree [edit]

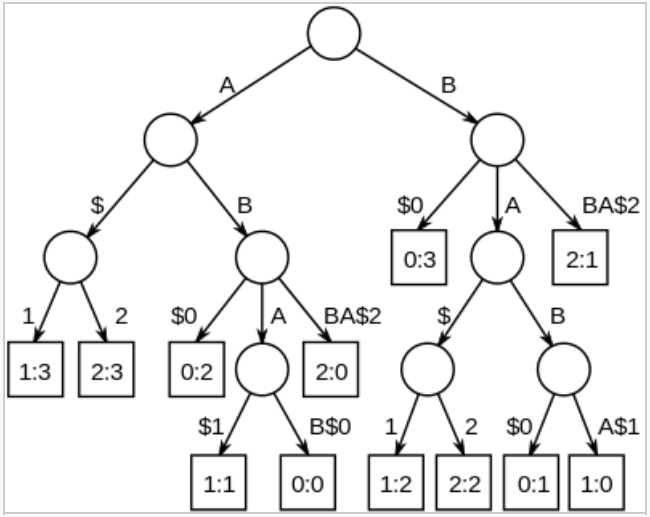
The longest common substrings of a set of strings can be found by building a generalised suffix tree for the strings, and then finding the deepest internal nodes which have leaf nodes from all the strings in the subtree below it. The figure on the right is the suffix tree for the strings "ABAB", "BABA" and "ABBA", padded with unique string terminators, to become "ABAB$0", "BABA$1" and "ABBA$2". The nodes representing "A", "B", "AB" and "BA" all have descendant leaves from all of the strings, numbered 0, 1 and 2.

Building the suffix tree takes $\Theta(N)$ time (if the size of the alphabet is constant). If the tree is traversed from the bottom up with a bit vector telling which strings are seen below each node, the k-common substring problem can be solved in $\Theta(NK)$ time. If the suffix tree is prepared for constant time lowest common ancestor retrieval, it

can be solved in $\Theta(N)$ time.[1]

Dynamic programming [edit]

First find the longest common suffix for all pairs of prefixes of the strings. The longest common suffix is



Generalised suffix tree for the strings "ABAB", "BABA" and "ABBA", numbered 0, 1 and 2.

$$LCSuff\left(S_{1..p}, T_{1..q}\right) = \begin{cases} LCSuff\left(S_{1..p-1}, T_{1..q-1}\right) + 1 & \text{if } S[p] = T[q] \\ 0 & \text{otherwise.} \end{cases}$$

For the example strings "ABAB" and "BABA":

		A	B	A	B
	0	0	0	0	0
B	0	0	1	0	1
A	0	1	0	2	0
B	0	0	2	0	3
A	0	1	0	3	0

The maximal of these longest common suffixes of possible prefixes must be the longest common substrings of *S* and *T*. These are shown on diagonals, in red, in the table. For this example, the longest common substrings are "BAB" and "ABA".

$$LCSubstr\left(S, T\right) = \max_{1 \le i \le m, 1 \le j \le n} LCSuff\left(S_{1..i}, T_{1..j}\right)$$

This can be extended to more than two strings by adding more dimensions to the table.

Pseudocode [edit]

The following pseudocode finds the set of longest common substrings between two strings with dynamic programming:

```
function LCSubstr(S[1..m], T[1..n])
    L := array(1..m, 1..n)
    z := 0
    ret := {}
    for i := 1..m
        for j := 1..n
            if S[i] == T[j]
                if i == 1 or j == 1
                    L[i,j] := 1
                else
                    L[i,j] := L[i-1,j-1] + 1
                if L[i,j] > z
                    z := L[i,j]
                    ret := {S[i-z+1..i]}
                else
                if L[i,j] == z
                    ret := ret ∪ {S[i-z+1..i]}
            else
                L[i,j] := 0
    return ret
```

This algorithm runs in $O(nm)$ time. The variable `z` is used to hold the length of the longest common substring found so far. The set `ret` is used to hold the set of strings which are of length `z`. The set `ret` can be saved efficiently by just storing the index `i`, which is the last character of the longest common substring (of size z) instead of `S[i-z+1..i]`. Thus all the longest common substrings would be, for each i in `ret`, `S[(ret[i]-z)..(ret[i])]`.

The following tricks can be used to reduce the memory usage of an implementation:

- Keep only the last and current row of the DP table to save memory ($O(\min(m,n))$ instead of $O(nm)$)
- Store only non-zero values in the rows. This can be done using hash tables instead of arrays. This is useful for large alphabets.

See also [edit]

- Longest palindromic substring

References [edit]

1. ^ Gusfield, Dan (1999) [1997]. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press. ISBN 0-521-58519-8.

External links [edit]

- Dictionary of Algorithms and Data Structures: longest common substring
- Perl/XS implementation of the dynamic programming algorithm
- Perl/XS implementation of the suffix tree algorithm
- Dynamic programming implementations in various languages on wikibooks
- working AS3 implementation of the dynamic programming algorithm
- Suffix Tree based C implementation of Longest common substring for two strings

The Wikibook *Algorithm implementation* has a page on the topic of: ***Longest common substring***

Categories: Problems on strings | Dynamic programming