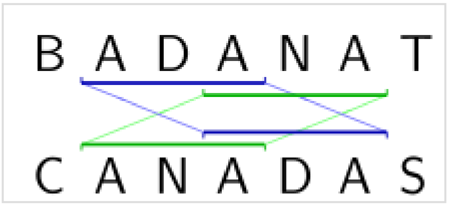
Wikipedia
The Free Encyclopedia

Longest common substring

In computer science, a **longest common substring** of two or more strings is a longest string that is a substring of all of them. There may be more than one longest common substring. Applications include data deduplication and plagiarism detection.

Examples



The strings "BADANAT" and "CANADAS" share the maximal-length substrings "ADA" and "ANA".

The picture shows two strings where the problem has multiple solutions. Although the substring occurrences always overlap, it is impossible to obtain a longer common substring by "uniting" them.

The strings "ABABC", "BABCA" and "ABCBA" have only one longest common substring, viz. "ABC" of length 3. Other common substrings are "A", "AB", "B", "BA", "BC" and "C".

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

Problem definition

Given two strings, S of length m and T of length n, find a longest string which is substring of both S and T.

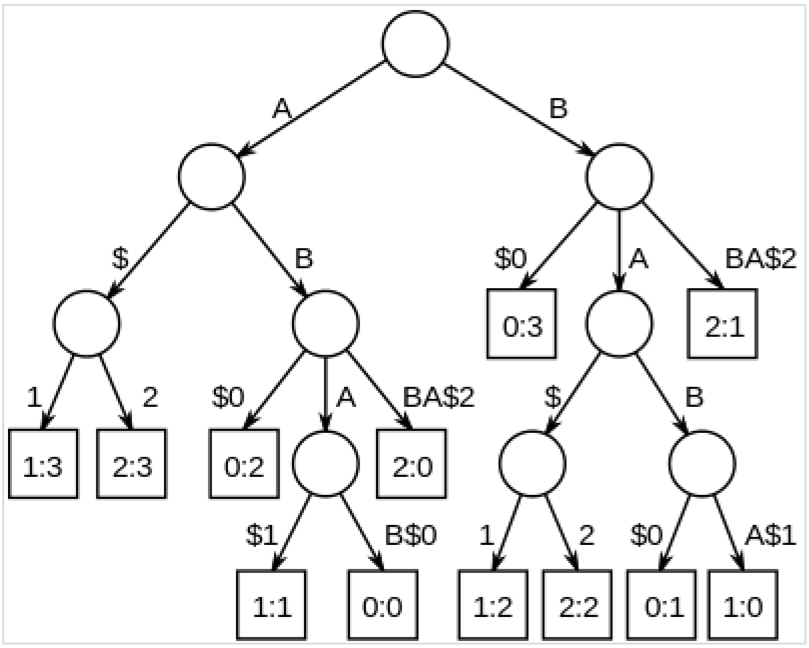
A generalization is the ***k*-common substring problem**. Given the set of strings $S=\{S_1,\ldots,S_K\}$, where $|S_i|=n_i$ and $\sum n_i=N$. Find for each $2\leq k\leq K$, a longest string which occurs as substring of at least k strings.

Algorithms

One can find the lengths and starting positions of the longest common substrings of S and T in $\Theta(n+m)$ time with the help of a generalized suffix tree. A faster algorithm can be achieved in the word RAM model of computation if the size σ of the input alphabet is in $2^{o\left(\sqrt{\log(n+m)}\right)}$. In particular, this algorithm runs in $O\left((n+m)\log\sigma/\sqrt{\log(n+m)}\right)$ time using $O\left((n+m)\log\sigma/\log(n+m)\right)$ space.[1] Solving the problem by dynamic programming costs $\Theta(nm)$. The solutions to the generalized problem take $\Theta(n_1+\cdots+n_K)$ space and $\Theta(n_1\cdots n_K)$ time with dynamic programming and take $\Theta(n_1+\cdots+n_K)$ time with a generalized suffix tree.

Suffix tree

The longest common substrings of a set of strings can be found by building a generalized 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.



Generalized suffix tree for the strings "ABAB", "BABA" and "ABBA", 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.[2]

Dynamic programming

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

```
function LongestCommonSubstring(S[1..r], T[1..n])
    L := array(1..r, 1..n)
    z := 0
    ret := {}

    for i := 1..r
        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 $O(nr)$ time. The array `L` stores the length of the longest common substring of the prefixes `S[1..i]` and `T[1..j]` which *end at position* `i` and `j`, respectively. 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 ($O(\min(r,n))$ instead of $O(nr)$)
  - The last and current row can be stored on the same 1D array by traversing the inner loop backwards
- 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

- Longest palindromic substring
- *n*-gram, all the possible substrings of length *n* that are contained in a string

References

1. Charalampopoulos, Panagiotis; Kociumaka, Tomasz; Pissis, Solon P.; Radoszewski, Jakub (Aug 2021). Mutzel, Petra; Pagh, Rasmus; Herman, Grzegorz (eds.). *Faster Algorithms for Longest Common Substring*. European Symposium on Algorithms. Leibniz International Proceedings in Informatics (LIPIcs). Vol. 204. Schloss Dagstuhl. doi:10.4230/LIPIcs.ESA.2021.30 (https://doi.org/10.4230%2FLIPIcs.ESA.2021.30). Here: Theorem 1, p.30:2.
2. Gusfield, Dan (1999) [1997]. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology* (https://books.google.com/books?id=Ofw5w1yuD8kC&q=%22longest+common+substring%22). USA: Cambridge University Press. ISBN 0-521-58519-8.

External links

- Dictionary of Algorithms and Data Structures: longest common substring (http://nist.gov/dads/HTML/longestCommonSubstring.html)
- Perl/XS implementation of the dynamic programming algorithm (http://metacpan.org/module/String::LCSS_XS)
- Perl/XS implementation of the suffix tree algorithm (http://metacpan.org/module/Tree::Suffix)
- Dynamic programming implementations in various languages on wikibooks
- working AS3 implementation of the dynamic programming algorithm (http://www.emanueleferonato.com/2010/12/01/solving-the-longest-common-substring-problem-with-as3/)
- Suffix Tree based C implementation of Longest common substring for two strings (http://www.geeksforgeeks.org/suffix-tree-application-5-longest-common-substring-2/)

-