Algorithms in bioinformatics (CSI 5126)[1]

Marcel Turcotte
(turcotte@site.uottawa.ca)

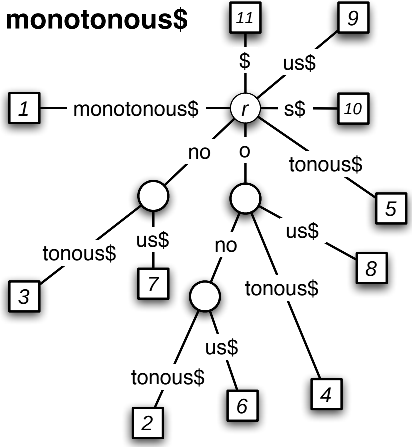
School of Information Technology and Engineering
University of Ottawa
Canada

October 2, 2009

[1] Please don't print these lecture notes unless you really need to!

Plan

- String algorithms
  - Applications of suffix trees (ST)
  - Generalized suffix trees
  - Lowest common ancestor (LCA)
  - Applications of LCA + ST


monotonous$
11
9
$
us$
1
monotonous$
r
s$
10
no
o
tonous$
5
us$
tonous$
us$
no
8
3
7
tonous$
us$
tonous$
6
4
2

Summary

- A suffix tree can be built in linear time and space
- Suffix trees were developed to determine if a string P occurs in a text T in time proportional to $|P|$ (after pre-processing, i.e. building the tree)
- **Indeed, P is a substring of T iff P is a prefix of a suffix of T**
- To locate P, it suffice to follow a unique path from the root of the tree up to a node, explicit or implicit, that corresponds to the end of the pattern. This takes time proportional to the length of the pattern
- **Nowadays, suffix tree based algorithms have been developed to solve a large array of problems for which no efficient algorithm was known. This lecture presents some of them**

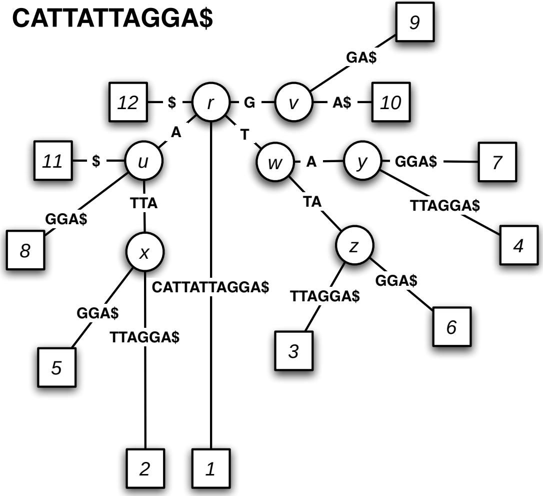
Longest repeated substring

- Let (i, j) denote the **substring** of S starting at i and ending at j, i.e. $S[i..j]$.
- A **repeat** is a pair $((i, j), (i', j'))$ such that $i < i'$ and $S[i..j] = S[i'..j']$.
- The **longest repeated substring** is the pair $((i, j), (i', j'))$ such that the length of the substring is maximum.
- The longest repeated substring of *abracadabra* is *abra*.

Naive algorithm

Imagine an algorithm to find the longest repeated substring without using a suffix tree.
What is its time complexity?

Suffix tree based algorithm

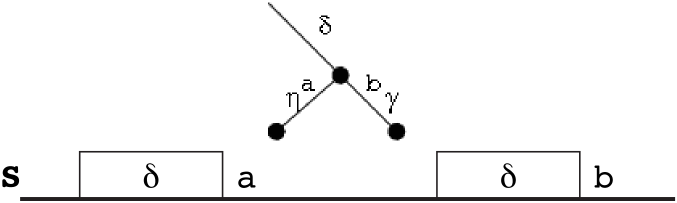


Suffix tree based algorithm (cont.)

Outline a suffix tree based algorithm for finding repeats? What characterizes a repeat?

Suffix tree based algorithm (cont.)

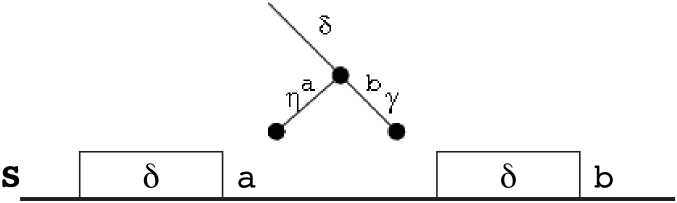
Let's define a **branching node**, sometimes called **fork**, as a node having two or more children.



The **path-label** of a node is the concatenation of all the edge labels along the path from the root to the node.

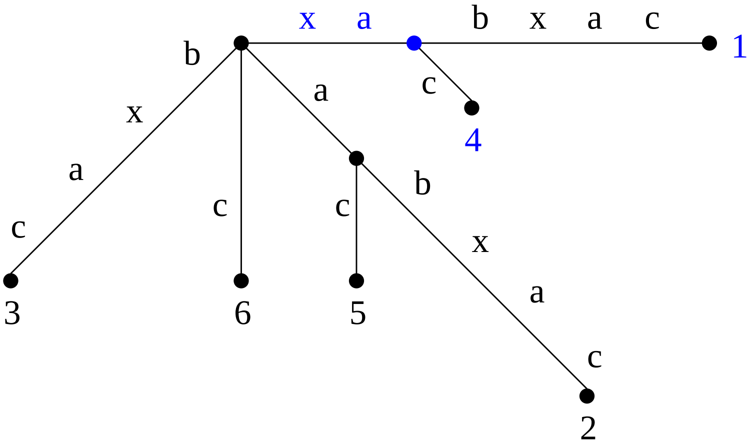
Suffix tree based algorithm (cont.)

Let's define a **branching node**, sometimes called **fork**, as a node having two or more children.



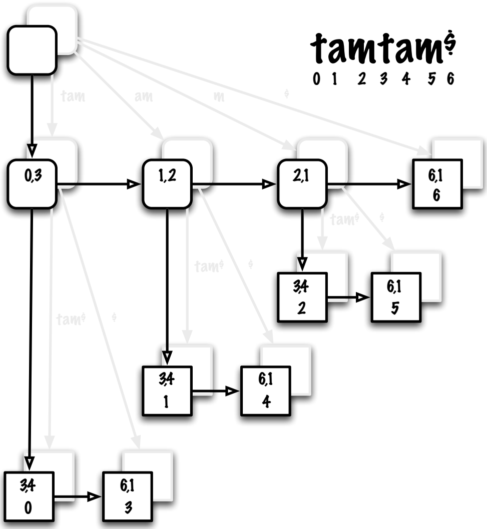
It suffice to traverse the tree and find a node **1)** which is a fork node and **2)** which has the longest path-label.
Finding the longest repeated substring takes $\mathcal{O}(|T|)$.

Suffix tree based algorithm (cont.)



1	2	3	4	5	6
x	a	b	x	a	c

Suffix tree based algorithm (cont.)



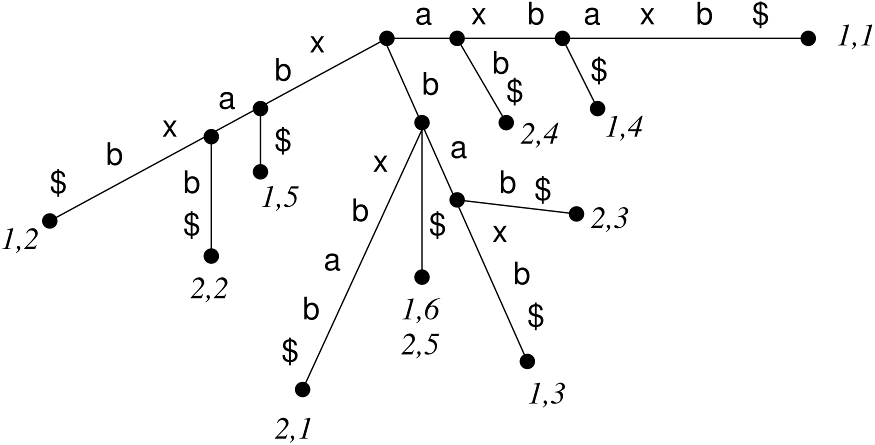
Suffix tree based algorithm (cont.)

```
public class Annotation implements Info {

    private int pathLength;
    private Info next;

    Annotation( int pathLength ) { this.pathLength = pathLength; }

    public Info getNextInfo() { return next; }

    public void setNextInfo( Info next ) { this.next = next; }

    public int getPathLength() { return pathLength; }

    public static void addPathLength( SuffixTree tree ) {
        InternalNode root = (InternalNode) tree.getRoot();
        if ( root != null )
            addPathLength( 0, (NodeInterface) root.getFirstChild() );
    }
    private static void addPathLength( int prefix, NodeInterface node ) {
        if ( node == null )
            return;
        int pathLength = prefix + node.getLength();
        node.setInfo( new Annotation( pathLength ) );
        if ( node instanceof InternalNode )
            addPathLength( pathLength, (NodeInterface) ( (InternalNode) node ).getFirstChild());
        addPathLength( prefix, (NodeInterface) node.getRightSybling() );
    }
}
```

Generalized suffix tree

- To find the **longest common substring** of a set of strings, we need to introduce the concept of generalized suffix tree.
- A generalized suffix tree represents all the suffixes of a set of strings $\{S_1, S_2, \ldots, S_K\}$.
- In the suffix tree for a single sequence, leaves are labeled with the starting position of the suffix within the string.
- In a generalized suffix tree, the leaves are labeled with a tuple, with a first index indicating the string this suffix belongs to, $1..k$, and the second index indicating the starting position.
- Because some of the k strings might have a common suffix, some leaves might contain more than one tuple.
- Alternatively, a unique terminator can be appended to each string so that a leaf designates a unique suffix.

Generalized suffix tree: an example



$S_1 = axbaxb$ and $S_2 = bxbab$

Substring problem

Definition. A set of strings, or database, is known in advanced and fixed.
After spending a linear amount of time pre-processing the input database, the algorithm will be presented a collection of strings and for each string the algorithm should be able to tell if the string is present in one or more strings from the input.

Substring problem

DNA identification. The U.S. army sequences a portion of the DNA of each member of its personnel. The sequence is selected so that **1)** it is easy to retrieve that exact sequence and **2)** it is unique to each individual.

In the case of a severe casualty, this particular DNA sequence can be used to identify uniquely a person.

Solution. A generalized suffix tree is built that contains all the input sequences. This takes time proportional to the sum of the lengths. To identify a person takes time proportional to length of the sequence identifier. The solution would also work if the sequence identifier can only be partially identified (in extreme cases).

Longest Common Substring

Finding the **longest common substring** of a set of strings is a recurring problem, and one which has many applications in bioinformatics.

In 1970, Donald Knuth conjectured that it would be impossible to find a linear time algorithm to solve this problem.

The longest common substring of $S_1 = axbaxb$ and $S_2 = bxbab$, is xba.

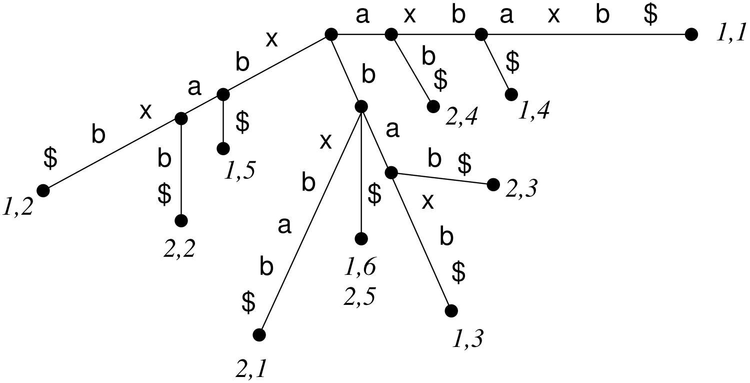
This problem can be elegantly solve in $\mathcal{O}(|S_1| + |S_2|)$ using generalized suffix trees. How?

Longest Common Substring (cont.)

Let's consider the case of two sequences, the generalization to k strings is trivial,

1. Construct a generalized suffix tree for S_1 and S_2;
2. In linear time, traverse the tree and label each node with (1), (2) or (1,2) if the subtree underneath the node contains only leaves from the first string, only leaves from the second string or a mixture of the two; (hint: use a bottom-up traversal)
3. In linear time, find the node such that 1) it's labeled (1,2) and 2) it has the longest path-label.

Longest common substring



⇒ The node with prefix *xba* is the deepest node (longest path label) that has descendants in both strings.

DNA contamination problem

- A host organism can be used to store foreign DNA molecules.
- **Clone library**. A foreign DNA segment can be inserted in a host organism in a way that makes it easy to retrieve the segment for later uses.
- The host will be selected for its ability to rapidly replicate, yeast for example, and therefore to make an endless number of copies of the original information.

DNA contamination problem

- It sometimes occur that the retrieved segments are contaminated with DNA from the host.
- The DNA contamination problem consists in finding all the substrings that are common to the host, S_1, and the segment, S_2, and are at least l nucleotides long.

DNA contamination problem

Solution: build a generalized suffix tree for S_1 and S_2. Traverse the tree and annotate all the nodes whose subtree contains leaves from both sequences; this takes a linear amount of time. Traverse the tree and for each node annotated with 1 and 2, such that the string length of the path is greater than l, print the string and locations, the traversal of the tree takes a linear amount of time.

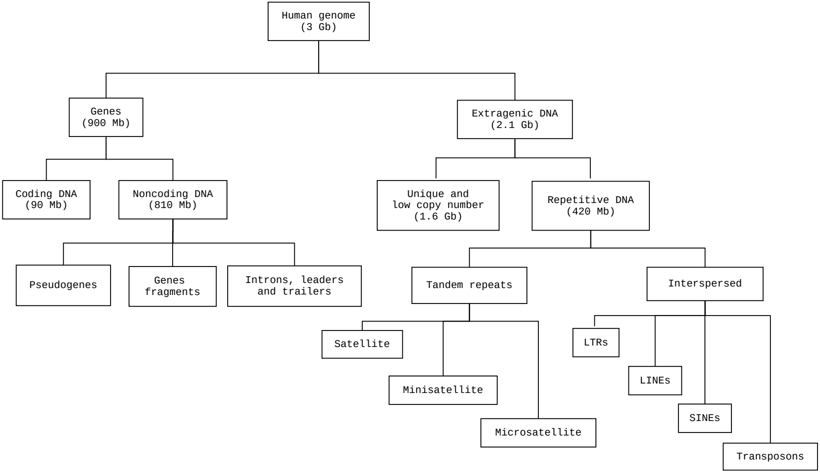
Sequence repeats

Repetitive sequences constitute a large fraction of the genomes. Transposable elements represent:

- 35.0–50% of the *Homo sapiens* (Human genome)
- 50.0% *Zea mays* (maize, corn)
- 15.0% *Drosophila melanogaster* (fruit fly)
- 2.0% *Arabidopsis thaliana* (a flowering plant)
- 1.8% *Caenorhabditis elegans* (a nematode, round worm)
- 3.1% *Saccharomyces cerevisiae* (baker's yeast)

⇒ Certain repeats have been related to diseases, regulation and molecular evolution.

Sequence repeats (cont.)



Sequence repeats — classification

Satellites: located near the centromeres or telomeres, up to one million bp long.

Microsatellite: 2 to 5 bp, 100 copies, found at the end of the eukaryotic chromosomes (telomeres), in humans hundreds of copies of TTAGGG.

Minisatellite: up to 25 bp, 30 to 2,000 copies

Sequence repeats — classification (cont.)

transposable elements: sequences that have the ability to move from one location of the genome to another, play an important role in evolution, they are classified according to their mechanism of transposition.

- class I: RNA mediated.
  - long terminal repeat (LTR): retrotransposons (related to retroviruses),
    - SINES: short interspersed nuclear elements, 80-300 bp, one particular

Sequence repeats — classification (cont.)

family is called Alu, in the human genome, there are 1.2 million copies (10%), (other sources say 300,000 copies, i.e ca. 5% of the genome),

Sequence repeats — classification (cont.)

LINES: long interspersed nuclear elements, 6-800 Kbp, one particular family is called LINE1, the human genome genome contains 593,000

Sequence repeats — classification (cont.)

copies
(14.6%).

class II: DNA mediated. Human genome has ca. 200,000 copies of elements of this type.

class III: has features of class I and class II,

MITES miniature inverted repeat transposable elements, 400 bp, discovered in flowering plants, frequently associated with regulatory regions of genes.

Sequence repeats

```
>ALU   Human ALU interspersed repetitive sequence - a consensus.
ggccgggcgcggtggctcacgcctgtaatcccagcactttgggaggccgaggcgggaggatcacttgagc
ccaggagttcgagaccagcctgggcaacatagtgaaaccccgtctctacaaaaaatacaaaaattagccg
ggcgtggtggcgcgcgcctgtagtcccagctactcgggaggctgaggcaggaggatcgcttgagcccggg
aggtcgaggctgcagtgagccgtgatcgcgccactgcactccagcctgggcgacagagcgagaccctgtc
tcaaaaaaaa
```

The Alu itself is constituted of repeats of length aprox. 40.
Often flanked by a tandem repeat, length 7-10, such that the left and right sequence are complementary palindromes.
$300,000+$ nearly, but not identical, copies dispersed throughout the genome.

Finding all repetitive structures

- For a given string of length n, there are $\Theta(n^2)$ substrings (one of length n, two of length $n-1$, three of length $n-2$ ... n substrings of length 1). **There are therefore $\Theta(n^4)$ possible pairs** — 8.1×10^{37} possible pairs in the case of the human genome!

We must carefully define what pairs are interesting otherwise too many results will be returned to the user!

Finding all repetitive structures

Definition. A **maximal pair** (or **maximal repeat pair**) is a pair of identical substrings α and β that cannot be extended either to the left or to the right without causing a mismatch, in other words, the character to the immediate left of α is different than the one to the immediate left of β, and similarly to the right, the characters immediately following α and β are different.

$\Rightarrow$ A maximal pair will be denoted (p_α, p_β, n') where p_α and p_β are the starting positions and n' their length. The set of all the maximal pairs of S will be noted $\mathcal{R}(S)$.

Maximal pairs

$$xyz\underline{bcd}eee\underline{bcd}xyz\underline{bcd}$$

The first and second occurrences of *bcd* form a maximal pair, $(4, 10, 3)$, the second and third occurrences form a maximal pair, $(10, 16, 3)$, but not occurrences one and three.

Are the two occurrences of *xyzbcd* forming a maximal pair?

To ensure that suffixes and prefixes can participate to maximal pairs a terminator is added at both ends.

$$\$\underline{xyzbcd}eeebcd\underline{xyzbcd}\$$$

Our definition does not prevent overlapping substrings, and this is fine.

Where to find maximal pairs?

Construct a suffix tree for S.

Repeats are found at internal nodes, so let $\mathcal{V}$ be the current internal node under consideration. Let α denote its path-label.

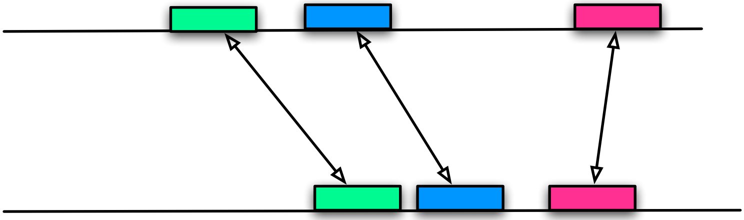
What next?

Let's take care of the right hand side. How can you make sure that α cannot be extended on the right? Select pairs of suffixes such that each of the two elements of the pair is from a distinct child of $\mathcal{V}$.

How would you take care of the left hand side? For every pair of suffixes i, j, $S[i] - 1 \neq S[j] - 1$.

Still many possible pairs of suffixes! Let's consider a more constrained problem.

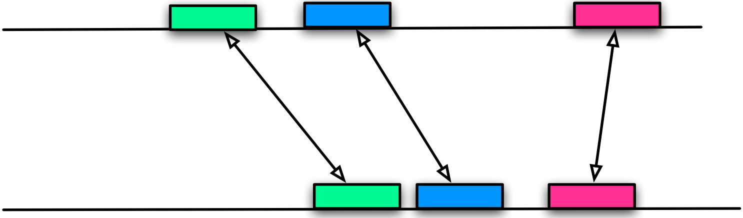
MUM — Maximum Unique Pairs



Algorithms to compare biological sequences (to be presented later) run in quadratic time and space. In the case of complete genomic sequences this is not feasible.

To circumvent this limitation, algorithms have been developed that first find a set of mums that are used as a starting point, anchors, for further processing by conventional sequence alignment techniques.

MUM — Maximum Unique Pairs



Given 2 sequences S_1 and $S_2 \in \mathcal{A}^*$ and $l > 0$, a *maximal unique match* is a string u such that:

- $|u| \geq l$
- u occurs exactly once in S_1 and exactly once in S_2
- $\forall\ a \in \mathcal{A}$, nor au or ua occurs simultaneously in S_1 and S_2.

MUM — Maximum Unique Pairs

ACAAGTCTTCTATCAGACTCCAGAAAAGTATCAGAGAGCAATGAA

CCACACTGCCTACCAGGTGTATCAGACCCACAAGTCCTTCTTAGA

MUM — Maximum Unique Pairs

ACAAGTCTTCTATCAGACTCCAGAAAAGTATCAGAGAGCAATGAA

CCACACTGCCTACCAGGTGTATCAGACCCACAAGTCCTTCTTAGA

MUM — Maximum Unique Pairs

ACAAGTCTTCTATCAGACTCCAGAAAAGTATCAGAGAGCAATGAA

CCACACTGCCTACCAGGTGTATCAGACCCACAAGTCCTTCTTAGA

MUM — Maximum Unique Pairs

ACAAGTCTTCTATCAGACTCCAGAAAAGTATCAGAGAGCAATGAA

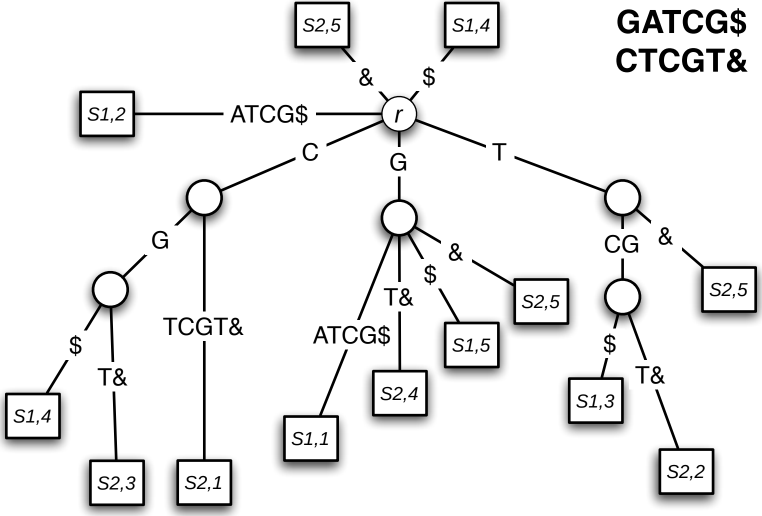
CCACACTGCCTACCAGGTGTATCAGACCCACAAGTCCTTCTTAGA

Where to look for MUMs?

- Construct a generalized suffix tree for S_1 and S_2
- Repeats and common substrings are found at internal nodes, look for an internal node that has children in S_1 and S_2, let's call it $\mathcal{V}$
- Can $\mathcal{V}$ have more than 2 children? No, this would mean that u occurs more than once in one or both input strings
- Is it possible that there are internal nodes along one of the paths from $\mathcal{V}$ to a leaf? No, again it would mean that u occurs more than once in one of both input strings
- **So, we have that $\mathcal{V}$ has to be an internal node that has exactly 2 children that are leaves**

Where to look for MUMs?

- Is it enough? No. Is it possible that u is embedded in a longer motif? In other words, that u is not maximal. u can certainly not be extended to the right. But how about the left? Yes, it is quite possible that u is in fact part of a larger motif, say au
- How, to check for that?
- The leaves beneath $\mathcal{V}$ contains the starting positions of the string u in S_1 and S_2, therefore it suffice to compare $S_1[i-1]$ and $S_2[j-1]$
- Time and space complexity?


S2,5
S1,4
&
$
S1,2
ATCG$
r
C
G
T
G
TCGT&
&
$
T&
ATCG$
CG
&
S2,5
S1,5
S2,5
$
T&
S1,4
S2,3
S2,1
S1,1
S2,4
$
T&
S1,3
S2,2

GATCG$
CTCGT&

```
all_mums( node v )

  if v is a leaf
     return

  if # children is 2
    if left child is a leaf and right child is a leaf
       set u to the path label of the path
       if char to the left of u in S1 differs from the char to
           the left of u in S2 and the path is long enough then
           display mum information
     else
        all_mums( left child )
        all_mums( right child )
 else
    for each child of v
        all_mums( child )
```

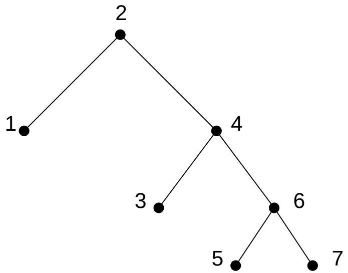
The subsequent steps of a complete algorithm for the alignment of two genomic sequences involve:

Finding the longest sequence of MUMs occurring in the same order the two sequences.

Apply an alignment algorithm (to be presented later) on the pairs of regions in between two MUMs.

Lowest Common Ancestor

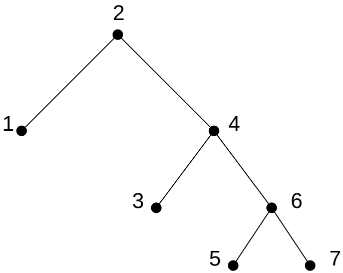
Definition. The lowest common ancestor (**lca**) of any two nodes of a rooted tree is the deepest node which is an ancestor[2] of both nodes.



The lca of 5 and 7 is 6, the lca of 1 and 3 is 2, and so on.

[2] A node *u* is an **ancestor** of a node *v* if *u* is a node that occurs on the unique path from the root to *v*.

Lowest Common Ancestor



How would you find the lowest common ancestor? What is the time complexity?

Lowest Common Ancestor in $\mathcal{O}(3n)$ time

Using two stacks S_i and S_j.

- Starting at node i, visit all the parents nodes until reaching the root of the tree, each visited node is pushed onto S_i
- Repeat the same operations starting at node j, this time, each visited node is pushed onto S_j
- Whilst the top nodes are identical, pop(S_i) and pop(S_j)
- The last identical node is the lowest common ancestor

Lowest Common Ancestor Problem (Overview)

Given an input tree with n nodes. Let's assume that $n < 4,294,967,296$ nodes.

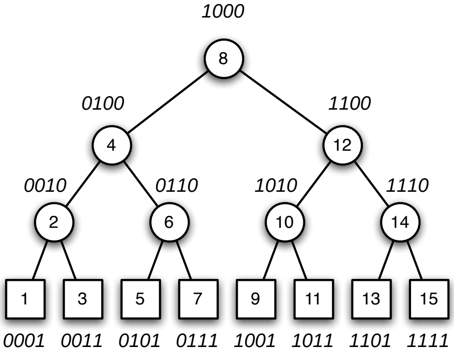
In the unit-cost RAM model, $\mathcal{O}(\log n)$ bits can be **read**, **written** or **used as an address** in constant time. Words of 32 bits.

(Although not necessary) Let's also make the following assumptions.

1. $\mathcal{O}(\log n)$ bits can be **compared**, **added**, **subtracted**, **multiplied**, or **divided** in constant time.
2. **bit-level operations** on $\mathcal{O}(\log n)$ bits numbers can be performed in constant time, including **AND**, **OR**, **XOR**, left or right shift by up to $\mathcal{O}(\log n)$ bits, creating masks of 1s, and finding the position of the left-most or right-most 1.

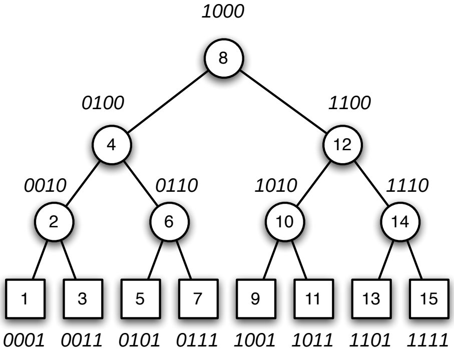
It can be shown, but we will not, that **after a linear amount of time pre-processing the input tree**, linear w.r.t. the number of nodes, **the lca of any two nodes can be found in constant time**! See (Gusfield 1997) §8.

LCA Algorithm: Overview



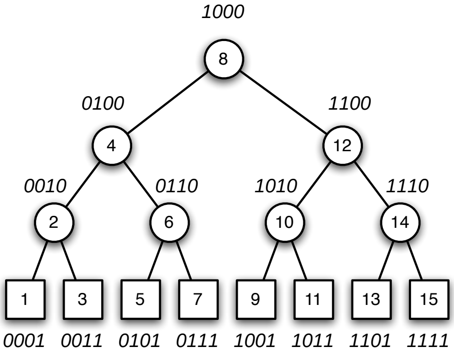
For this overview of the **lca** algorithm, let's consider the case of a complete rooted binary tree. This tree has p leaves and n nodes, where $n = 2p - 1$.

LCA Algorithm: Overview



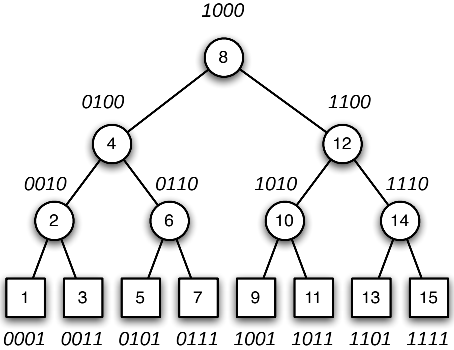
Furthermore, consider the **in order** (Left-Root-Right) labelling of the tree and its interpretation as binary numbers. How much does it cost to label this tree? $\mathcal{O}(n)$ time. This is the pre-processing step/time.

LCA Algorithm: Overview



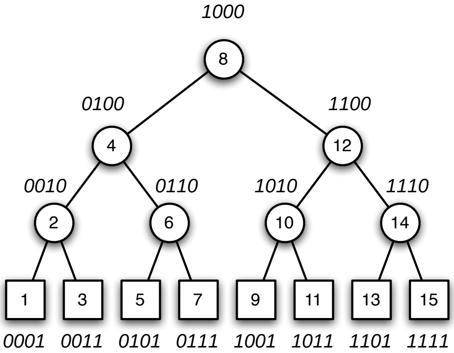
The number of edges on any path from the root to any leaf is $d = \log_2 p$. Let's now interpret the numbers (labels) as $d + 1$ bit **path numbers**, i.e. starting from the left hand side of the number, each bit represents a direction, $0 =$ left, $1 =$ right.

LCA Algorithm: Overview



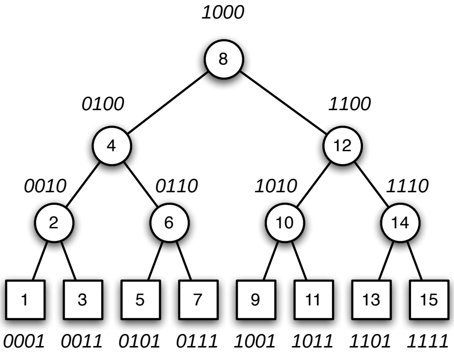
The structure of a path number is as follows, for a node *v* at level *i*, the left-most *i* bits are the **path bits**, followed by 1, which is a separator, and the remaining bits are 0s, i.e. (**path bits, 1, 0s**).

LCA Algorithm: Overview



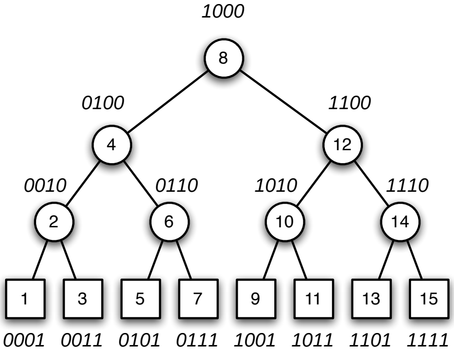
Any two nodes that have a common ancestor at level k are labeled with path numbers such that the first k bits are identical. Consider the nodes 5 and 7, 3 and 6, or 7 and 9.

LCA Algorithm: Overview



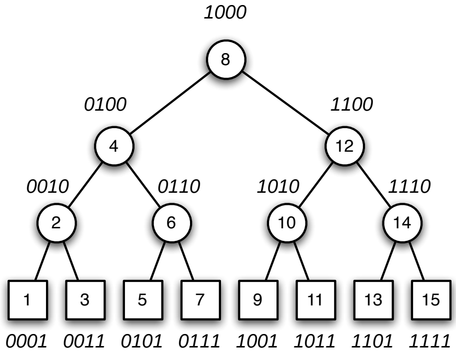
Given two nodes u and v, what property of $\text{XOR}_{u,v}$ would be particularly interesting here? $\text{lca}(9,11) = \text{XOR}_{1001,1011} = 0010$.

LCA Algorithm: Overview



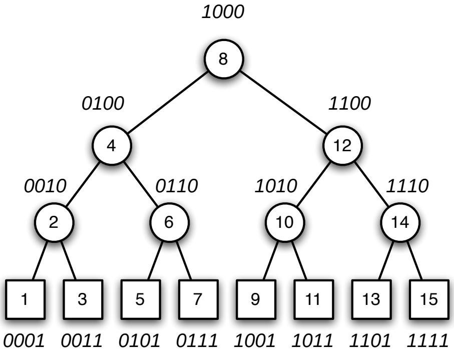
Given two nodes u and v, since the left-most k bits are identical, the left-most k bits of the XOR of the two path numbers, denoted $\text{XOR}_{u,v}$, will all be 0s. The left-most 1 of $\text{XOR}_{u,v}$, occurs a position $k+1$ (from the left).

LCA Algorithm: Overview



Given two nodes u and v, the path number of $\text{lca}(u, v)$ is obtained by calculating $\text{XOR}_{u,v}$, finding the left-most 1, let k be the position of the left-most 1, shift u right $d + 1 - k$ positions, set the right-most bit to 1, shift u left $d + 1 - k$ positions, thus inserting 0s.

LCA Algorithm: Overview



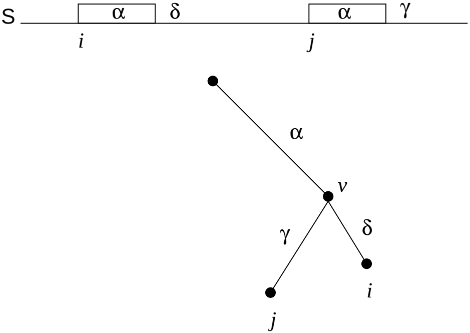
$\text{lca}(9, 14) = \text{XOR}_{1001,1110} = 0111$, $k = 2$, shift 1001 right by $d + 1 - k = 3 + 1 - 2 = 2$ positions, result is 10, set right most bit to 1, result is 11, shift 11 $d + 1 - k = 3 + 1 - 2 = 2$ to the left, padding with 0s, result is 1100.

LCA Algorithm: Overview

- Pre-processing (labelling) requires $\mathcal{O}(n)$ time.
- **lca(u, v)** requires a fixed number of bit-level operations, each of which can be performed in constant time.
- The idea behind the general **lca** algorithm is to conceptually map the nodes of a complete binary tree, labeled with path numbers, onto the nodes of the input tree, in such a way that the result of an lca query on the complete binary can be used to answer a query on the input tree.

LCA and suffix trees

What does **lca** mean in the context of suffix trees?



LCA and suffix trees

- **lca(i,j)** returns the deepest node which is a common ancestor of both *i* and *j*. The path from the root to that node spells the longest common prefix of the suffixes *i* and *j*
- **Therefore, the longest common prefix of any two suffixes can be found in constant time!** Once the tree has been pre-processed, which takes a linear amount of time.

Longest Common Extension

Definition. The *longest common extension* problem is as follows. One is given two strings, S_1 and S_2, after a preprocessing phase the user should be able find the longest common substring starting at position i in sequence 1 and j in sequence 2.

Longest Common Extension (cont.)

To solve this problem, first build a generalized suffix tree for S_1 and S_2. Then process the tree so that **lca** queries can be answered for that tree, this will take a linear amount of time, and label the tree to record the string-depth of every node, this also will take linear time.

The length of the longest common extension starting at positions i and j is the string-depth recorded at the node designated by lca(i, j).

Longest Common Extension (cont.)

Is it useful?

k-mismatch problem

Definition. Given a pattern P, a text T and a number of mismatches, k, fixed in advance and independent of $|P|$ and $|T|$. A k-**mismatch** of P against T is a substring of T that matches P with at most k mismatches (errors), in other words, there are at least $|P| - k$ matches.

The k-mismatch problem consists in finding all k-mismatches.

k-mismatch problem

Given T = Taumatawhakatangihangakoauauotamateapokaiwhenuakitanatahu is a hill south of Waipukurau, New Zealand and P = auauatamateapakaiwhemua, find all 3-mismatches.

LONGEST PLACE NAME IN THE WORLD
Taumatawhakatangihangakoauauotamateaturipukakapikimaungahoronukupokaiwhenuakitanatahu

k-mismatch check

Is there a 3-mismatch occurrence of P at position 25 of T?

```
T = Taumatawhakatangihangakoauauotamateapokaiwhenuakitanata

                        P = auauatamateapakaiwhemua
```

k-mismatch check

Checking for k-mismatch starting from position i in T.

1. Set j to 1, i' to i and *count* to 0;
2. Compute $l = \mathrm{lce}((P, j), (T, i'))$;
3. If $j + l = n + 1$ then a k-mismatch of P in T occurs at position i, stop.
4. If *count* $\leq k$ then increment *count* by one, set j to $j + l + 1$ and i' to $i' + l + 1$, go to 2.
5. a k-mismatch of P does not occur in T at position i.

⇒ A similar algorithm exists for matching with wild cards.

Exact matching with wild cards

A protein motif called the *Zinc finger*[3]:

c..c.............h..h

the "." symbol matches any character.
The following (regular) expression matches 45 words in **/usr/share/dict/words**:

i...ement

including: imbuement, implement, inclement, increment, induement and inurement.

[3] PROSITE is a collection of known protein motifs (www.expasy.ch/prosite)

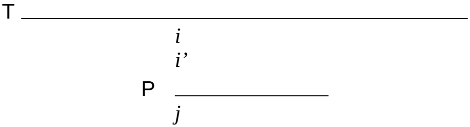
Exact String Matching with Wild Cards

Finding a match starting from position i in T.

1. Set j to 1 and i' to i.
2. Compute $l = \mathrm{lce}(j, i')$, where j is a starting position in P and i' a position in T.
3. If $j + l = n + 1$ then P occurs in T at position i, stop.
4. If $P(j + l)$ or $T(i' + l)$ is a wild card, set j to $j + l + 1$ and i' to $i' + l + 1$, go to 2.
5. P does not occur in T at position i.

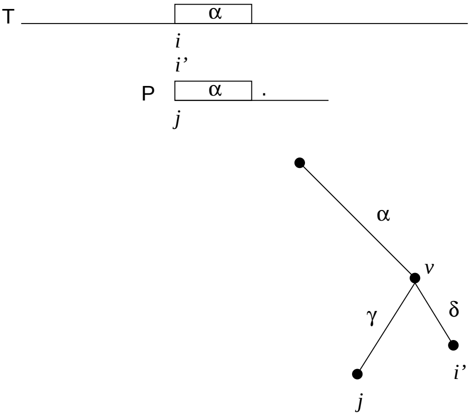
$\Rightarrow$ How much space is needed? The algorithm takes $\mathcal{O}(k)$ for a fix k. How much time to find all occurrences?

Exact String Matching with Wild Cards (cont.)



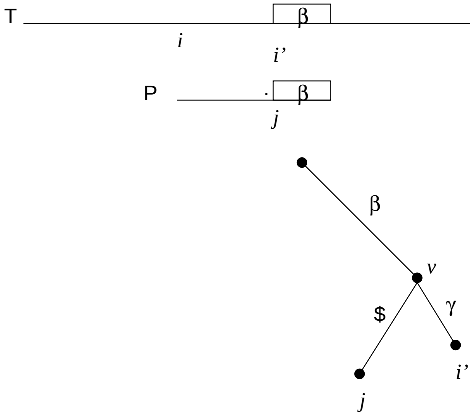
⇒ At the start of the algorithm.

Exact String Matching with Wild Cards (cont.)



⇒ Since the longest common extension is immediately followed by a wild card, the algorithm is allowed to continue.

Exact String Matching with Wild Cards (cont.)



⇒ The end of P has been reached.

k-differences

A similar algorithm can be obtained for finding all the substrings of T that no more than k differences away from P.

A difference is an insertion, a deletion or a substitution.

The resulting algorithm requires $\mathcal{O}(3^k|T|)$ time.

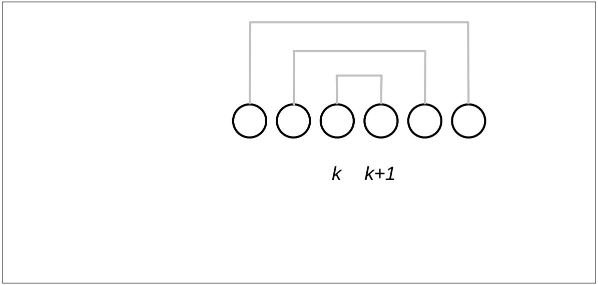
See Alaru page 6-3.

Maximum Palindromes

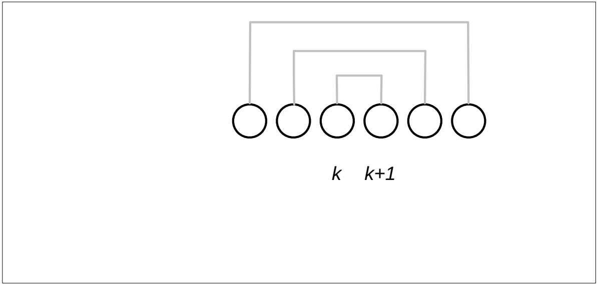
Definition. A *maximal palindrome of radius d* is a substring S' of S such that $S' = \alpha\alpha^r$, $|\alpha| = d$, and for any $d' > d$ S' is not a palindrome.

According to the above definition the length of S' is even.

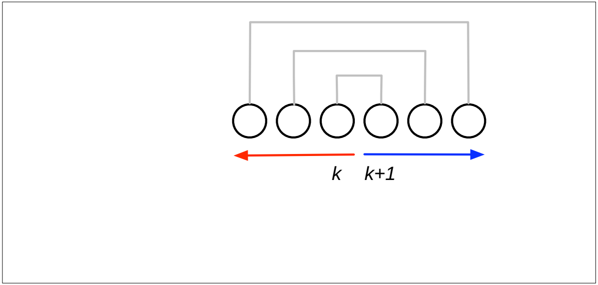
Maximum Palindromes



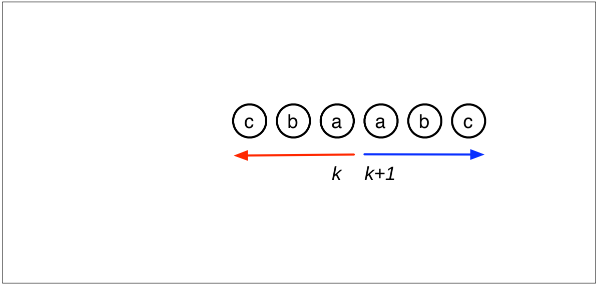
Maximum Palindromes



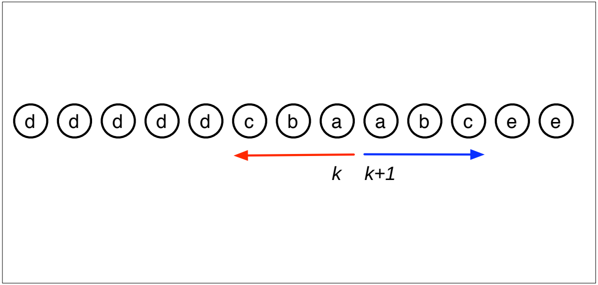
Maximum Palindromes



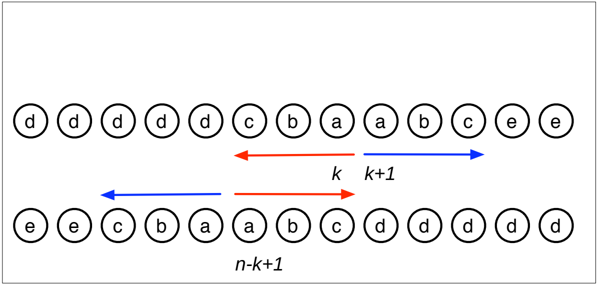
Maximum Palindromes



Maximum Palindromes



Maximum Palindromes



Finding All Maximal Palindromes

1. Create a generalized suffix tree for S and S^r, process the tree so that **lce** queries can be answered in constant time. Creating S^r, the generalized suffix tree and process necessary for **lce** takes $\mathcal{O}(|S|)$.
2. For q from 1 to $|S|-1$; $k = \mathrm{lce}(q+1, |S|-q+1)$ is the radius of the longest palindrome centered at q.

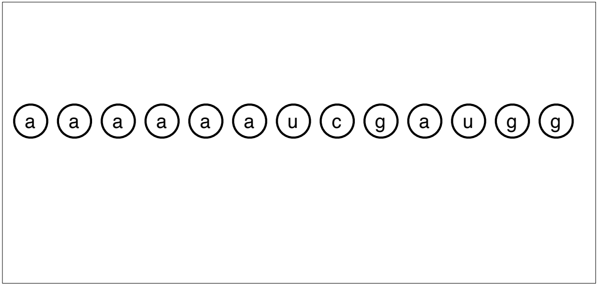
Each iteration takes $\mathcal{O}(1)$ time.

$\Rightarrow$ where **lce**, is the "longest common extension".

Finding Maximal Palindromes

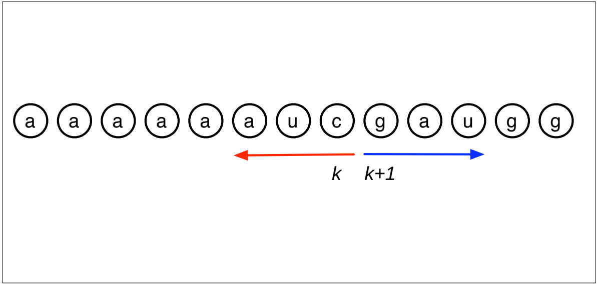
Hum . . . is it biologically relevant?
Probably not.

Finding Maximal Palindromes



The above string is biologically relevant, why?

Finding Maximal Palindromes



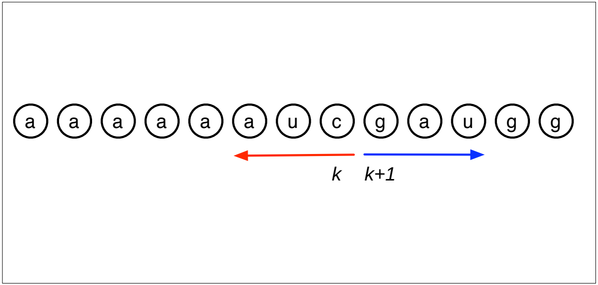
It contains a biological palindrome, a string that reads the same when **reversed** and **complemented**, where the following rules are used to obtain the complement, A is the complement of U (and vice versa), and G is the complement of C (and vice versa).

Finding Maximal Palindromes

$\mathbf{a}$	$\mathbf{a}^c$	$\mathbf{a}^{c^c}$
A	U	A
C	G	C
G	C	G
U	A	U

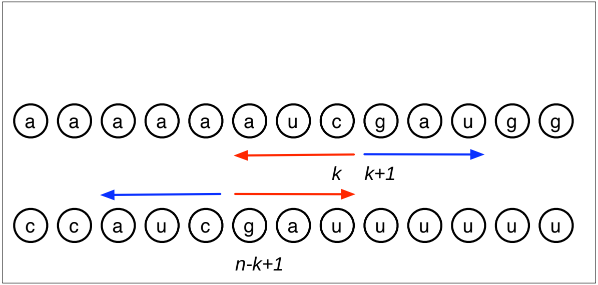
where $\mathbf{a}^c$ denotes the complement

Finding Maximal Palindromes



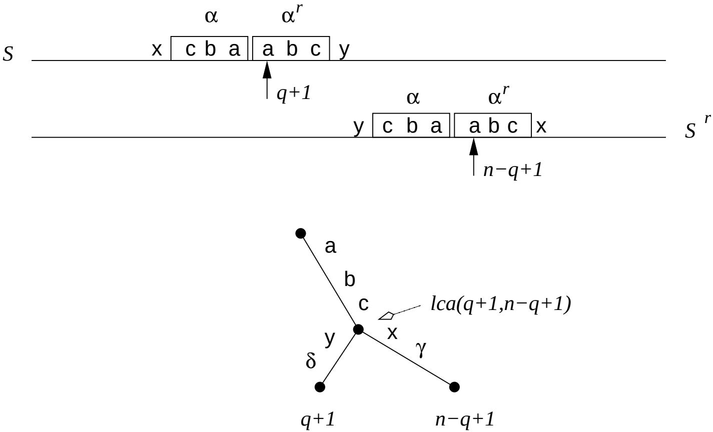
How to find the maximal biological palindromes.

Finding Maximal Palindromes



Given and input sequence S compute its reverse complement, S^{rc}, for every k, compute **lce**$(S, k+1, S^{rc}, n-k+1)$.

Finding Maximal Palindromes



⇒ Similarly for complemented palindromes or for palindromes separated by a bounded distance k.

Seed

Seed: bio.site.uottawa.ca/software/seed
Mohammad Anwar, Truong Nguyen and Marcel Turcotte (2006) Identification of consensus RNA secondary structures using suffix arrays. BMC Bioinformatics, 7:244.

The Nobel Prize in Chemistry 2006

In 2006, Fire and Mello received the Nobel prize in Medicine for their discovery of **RNA interference**, which is a cellular process by which the expression of a specific gene is inhibited — we say that the gene has been silenced.

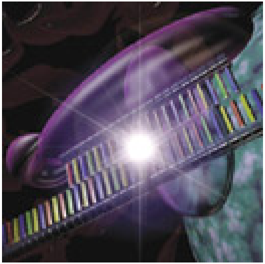
Andrew Z. Fire
Stanford University School of Medicine
Stanford, CA, USA

Craig C. Mello
University of Massachusetts Medical School
Worcester, MA, USA

RNA interference

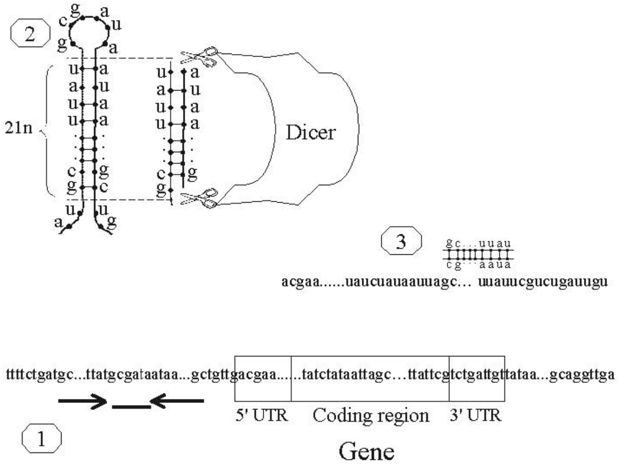
RNA interference (RNAi) is a mechanism in molecular biology where the presence of certain fragments of double-stranded ribonucleic acid (dsRNA) interferes with the expression of a particular gene which shares a homologous sequence with the dsRNA.

Wikipedia



See www.nature.com/focus/rnai/animations/

RNA interference

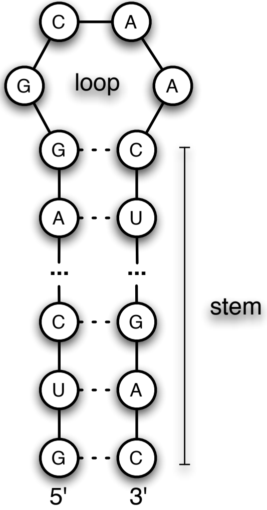


2
c g a u g a
u a u u
a u a a
21n
c g
g c
u u
a g
u a u u
a u a a
c g
g
Dicer
3
gc…uuau
cg…aaua
acgaa......uaucuauaauuagc… uuauucgucugauugu
ttttctgatgc...ttatgcgataataa...gctgttgacgaa......tatctataattagc…ttattcgtctgattgttataa...gcaggttga
1
5' UTR
Coding region
3' UTR
Gene

Final examination 2006

RNA silencing involves an RNAi element, which consists of stem-loop secondary structure, where the stem (a double stranded region) is 20 to 25 nucleotides long and the loop is at least 4 nucleotides long but no more than k. Moreover, one of the two strands of the stem is the reverse complement of a portion of the gene that it silences. RNAi elements are encoded by the genome.

Outline an algorithm in pseudo-code that finds all the RNAi elements of a given genome. Specifically, it finds stem-loops structures, with the above characteristics, such that one of their two strands is the complement of an existing gene. Assume that the location of all the protein-coding genes is known. Make sure to describe the necessary data structures and how they are initialized.



Tandem Repeats

Definition. A *tandem repeat* is a string α such that $\alpha = \beta\beta$, where β is a substring.
Finding tandem repeats in $\mathcal{O}(n^2)$:

```
for i to n-1 do
  for j from i+1 to n do
    l = length of the longest common extention of i,j
    if i+l >= j then
        a tandem repeat of length 2(j-i+1) starts at i
```

k-mismatch tandem repeat

Definition. A *k-mismatch tandem repeat* is a substring that becomes a tandem repeat after k or fewer characters are changed. Outline an algorithm which finds k-mismatch tandem repeats. What's the complexity of your algorithm?

$\Rightarrow$ (Landau & Schmidt 1993) presents an algorithm running $\mathcal{O}(kn\log(\frac{n}{k}) + z)$, where z is the number of tandem repeats.

Software using suffix trees/arrays

REPuter: bibiserv.techfak.uni-bielefeld.de/reputer
S. Kurtz, J. V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, R. Giegerich: REPuter: The Manifold Applications of Repeat Analysis on a Genomic Scale. Nucleic Acids Res., 29(22):4633-4642, 2001.

VMATCH: www.vmatch.de

MUMMER: mummer.sourceforge.net
A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White, and S.L. Salzberg (1999) Alignment of Whole Genomes. Nucleic Acids Research, 27:11 (1999), 2369-2376.

Applications

Excerpts from www.vmatch.de.

- Detecting unique substrings in large collection of DNA sequences that are used as signatures allowing for rapid and accurate diagnostics to identify pathogen bacteria and viruses;
- Computing a non-redundant set from a large collection of protein sequences from Zea-Maize;
- Finding sequence contamination errors in Arabidopsis thaliana;
- Mapping clustered sequences to large genomes;
- Pattern searches in plant sequences;
- Computing repeats in complete genomes.

References

Gusfield D. (1997) Algorithms on strings, trees, and sequences. Cambridge Press.

Landau G. M. and Schmidt J.P. (1993) An algorithm for approximate tandem repeats. *Proc. 4th Symp. on Combinatorial Pattern Matching*. Springer LNCS 684, pages 120–133.

Please don't print these lecture notes unless you really need to!