Alignment vs. Pattern Matching

• global sequence alignment
  – input: \( n \geq 2 \) relatively short sequences
  – homology assumptions: homologous along entire length, colinear
  – goal: determine homologous positions

• pattern matching
  – input: \( n \geq 1 \) sequences (short or long)
  – homology assumptions: none
  – goal: find short exact/inexact substring (local) matches between or within input sequences
Why pattern matching?

- **applications:**
  - database search - short query, large DB
  - alignment of long sequences - exact global alignment no longer feasible
  - alignment of rearranged/duplicated sequences - no longer a global alignment task

- **key idea:** short local matches can *seed* longer global alignments

**Database Search**

- **highly-similar short match to query**
- **pattern matching**
- **global alignment of each candidate to query**
Global Alignment of Long Sequences

- find seed matches using pattern matching
- sparse dynamic programming with seed matches

Alignment of Rearranged Sequences

- whole-genome dot plot
- each dot represents a seed match between genome sequences
Suffix Trees

• substring problem:
  – given text $S$ of length $m$
  – preprocess $S$ in $O(m)$ time
  – such that, given query string $Q$ of length $n$, find occurrence (if any) of $Q$ in $S$ in $O(n)$ time
• suffix trees solve this problem, and others

History of Suffix Trees

• Weiner (1973): first linear time algorithm for suffix tree construction
  – “the algorithm of 1973” (Knuth)
• McCreight (1976): space efficient version of algorithm
• Ukkonen (1995): simple, elegant algorithm, with “online” property
Suffix Tree Definition

• a suffix tree $T$ for a string $S$ of length $m$ is a tree with the following properties:
  – *rooted* and *directed*
  – $m$ leaves, labeled 1 to $m$
  – each edge labeled by a substring of $S$
  – concatenation of edge labels on path from root to leaf $i$ is suffix $i$ of $S$ (we will denote this by $S_{i...m}$)
  – each internal non-root node has at least two children
  – edges out of a node must begin with different characters

Suffix Tree Example

• $S = \text{“banana$”}$
• add ‘$’ to end so that suffix tree exists (no suffix is a prefix of another suffix)
Solving the Substring Problem

• assume we have suffix tree $T$
• FindMatch($Q$, $T$):
  – follow (unique) path down from root of $T$
    according to characters in $Q$
  – if all of $Q$ is found to be a prefix of such a path
    return label of some leaf below this path
  – else, return no match found

Solving the Substring Problem

$Q = nan$

return 3

$Q = anab$

return no match found
Runtime of Substring Problem with Suffix Tree

- finite alphabet: $O(1)$ work at each node
- edges out of each node start with unique characters: unique path from root
- size of tree below end of path: $O(k)$, $k =$ number of suffixes starting with $Q$
- $O(n + k)$ time to report all $k$ matching substrings
- $O(n)$ to report just one with an additional trick

Naive Suffix Tree Building

- now we need a $O(m)$ time algorithm for building suffix trees
- naive algorithm is $O(m^2)$:
  - $T \leftarrow$ empty tree
  - for $i$ from 1 to $m$:
    - add suffix $S_{i..m}$ to $T$ by finding longest matching prefix of $S_{i..m}$ in $T$ and branching from there
    - each step is $O(m)$
O($m^2$) Suffix Tree Building

Ukkonen’s O($m$) Algorithm

- on-line algorithm
  - builds *implicit* suffix tree for each prefix of string $S$
  - implicit suffix tree of $S_{1...i}$ denoted $I_i$
  - builds $I_1$, then $I_2$ from $I_1$, ..., then $I_m$ from $I_{m-1}$

- basic algorithm is $O(m^3)$, but with a series of tricks, it is $O(m)$
Implicit Suffix Tree

- Suffix tree → implicit suffix tree
  - remove $ characters from labels
  - remove edges with empty labels
  - remove internal nodes with < 2 children

Ukkonen’s Algorithm Overview

construct $I_1$
for $i$ from 1 to $m - 1$:
    for $j$ from 1 to $i + 1$:
        • find end of path from root labeled $S_{j\ldots i}$
        • add character $S_{i+1}$ to the end of this path in the tree, if necessary
Suffix Extension Rule 1.

1. if path $S_{j...i}$ in tree ends at leaf, add character $S_{i+1}$ to end of label of edge into leaf

$$S_{j...i} = ...an$$  $$S_{j...i+1} = ...ana$$

Suffix Extension Rule 2.

2. if there are paths continuing from path $S_{j...i}$ in the tree, but none starting with $S_{i+1}$, then create a new leaf edge with label $S_{i+1}$ at the end of path $S_{j...i}$ (creating a new internal node if $S_{j...i}$ ends in the middle of an edge)

$$S_{j...i} = ...na$$  $$S_{j...i+1} = ...nay$$
Suffix Extension Rule 3.

3. if there are paths continuing from path $S_{j...i}$ in the tree, and one starts with $S_{i+1}$, then do nothing

$$S_{j...i} = \ldots na \quad S_{j...i+1} = \ldots nan$$

Conversion to Suffix Tree

- convert implicit suffix tree at end of algorithm into true suffix tree
- simply run algorithm for one more iteration with $\$$ final character
- traverse tree to label leaf edges with positions
Example

$I_1$  $I_2$  $I_3$  $I_4$

$I_5$  $I_6$

Example (Continued)
Key Idea 1: Leaves ⇒ Free Operations

- once a leaf always a leaf: when a leaf edge is created on phase $p$, label the edge with $(p,e)$
- $e$ is a global index that is updated in constant time on each phase

Key Idea 2: Existing Strings ⇒ Free Operations

- if suffix extension rule 3 applies to extension $j$, it will apply in all further extensions in phase; therefore end phase early

3. if there are paths continuing from path $S_{j...i}$ in the tree, and one starts with $S_{i+1}$, then do nothing

\[ S_{j...i} = \ldots \text{n} \text{a} \quad S_{j...i+1} = \ldots \text{n} \text{a} \text{n} \]
Ukkonen’s Algorithm with Implicit Free Operations

construct $I_1$
for $i$ from 1 to $m - 1$:
    for $j_L < j < j_R$:
        find end of path from root labeled $S_{j...i}$
        add character $S_{i+1}$ to the end of this path

- $j_L$ in iteration $i$ is the last leaf inserted in iteration $i-1$
- $j_R$ in iteration $i$ is the first index where $S_{j...i+1}$ is already in the tree

Explicit Operations

- for $j_L < j < j_R$, $j$ is made a leaf
- once a leaf, always a leaf

Figure from Aarhus Universitet course on String Algorithms
Ukkonen’s Algorithm

construct $I_i$
for $i$ from 1 to $m - 1$
for $j_L < j < j_R$
find end of path from root labeled $S_{j...i}$
add character $S_{i+1}$ to the end of this path

$2m$ of these can be done in constant time

Key Idea 3: Suffix Links

- how to find end of each suffix $S_{j...i}$?
- instead of searching down tree in $O(i-j+1)$ time, use suffix links and some tricks

- a suffix link is a pointer from an internal node $v$ to another node $s(v)$ where
  - $x$ is a character, $\alpha$ is a substring (possibly empty)
  - $v$ has path-label $x\alpha$
  - $s(v)$ has path-label $\alpha$
Edge-Label Compression

- to get run time down to $O(m)$ have to ensure that space is $O(m)$
- label edges with pair of indices into string rather than with explicit substring
- makes space requirement only $O(m)$

$S = \text{“banana$”}$

Final Runtime

- putting all of these tricks and implementation details together, Ukkonen’s algorithm runs in time $O(m)$
- more details found in (Ukkonen, 1995) or book by Dan Gusfield (Gusfield, 1997)