SUFFIX ARRAYS

1 2 3 4 5 6 7 8
ABA BAA BAABA

(8 5 6 3 1 7 4 2)

we can get \(O(n)\) time by using suffix trees!

\(O(n)\) time direct methods exist.

\(q: BA\)

naive querying

\(O(|q| \log n)\)

\(\downarrow\)

\(O(|q| + \log n)\)
1) Maintain \( M_{LR} \)
   - Start any comparison at \( M_{LR+1} \)
   
   \[ M_{LV} = \min(l, r) \]
   
   We found \( BA \).

2) Start at \( \max(l, r) \)

---

1 2 3 4 5 6 7 8

\[ q = BA \]

\[ l = \text{LCP}(q, L) = 0 \]

Longest common prefix length

\[ \gamma = \text{LCP}(q, K) = 2 \]

\[ M_{LR} = 0 \]

\[ M_{LV} = \min(l, r) \]

---

\[ q = ABC \]

\[ l = 2 = \text{LCP}(q, L) \]

\[ \gamma = 2 = \text{LCP}(q, K) \]

\[ M_{LR+1} = 2 + 1 = 3 \]

\[ O(|q| \cdot b \cdot \alpha) \]
The text on the page is as follows:

```
1 2 3 4 5 6 7 8
A B A B A A B A

q = BA
LCP(q, L)  LCP(q, R)
l = 0  \alpha = 2
LCP(L, M) = 0
LCP(M, R) = 2

Precomputed
Value

q matches with T

O(|q| + \log n)

Not only linear search

```

```

O(n + n-1) = O(n) \times O(1) = O(n)

```

The diagram shows a tree structure with nodes labeled with ordered pairs and branches indicating the LCP (Longest Common Prefix) values. The tree is used to illustrate the computation of LCP values for strings and to demonstrate the efficient computation of the LCP with a precomputed value.
In exact matching via Hashing

\[ l \leftarrow \text{match length} \]
\[ d \leftarrow \# \text{ mismatches allowed} \]

**LSH** : locality sensitive Hashing

\[ h_1: \text{ choose any } k \text{ position or random} \]
\[ \rightarrow \text{ hash key} \]

**DB:** H \[ \text{ACACACACACCC} \]

\[ q: \]
\[ \text{ATACATCA} \]
\[ \text{GTACACAC} \]

\[ h_1: (3, 4, 5) \]
\[ e.g. \ l(1, 2, 3) \]

**HT:**
\[ \text{ACAC} \rightarrow 4, 5 \]

\[ \text{Prob of a match between } q \text{ and DB for a single position} \]

**Answer:**
\[ \frac{DB(q)}{DB(\bar{q})} \]
\[ \left(1 - \frac{d}{l}\right) \]

**Prob of a match over k position is**
\[ \left(1 - \frac{d}{l}\right)^k \]
1) false positive: $DB[i] 
\rightarrow \text{db substring hence + a cell }
\checkmark
\text{the true distance } > d

\text{solution: just compare the query DB[i]}

2) false negatives:

there exists a match but they don't hash to the same cell?

\text{solution: we cannot eliminate this}

however, we can minimize this!

do m hash keys

$h_1(1, 2, 3) \rightarrow P(\text{if mismatch})$
$h_2(2, 3, 4)$
$h_3(\ldots)$
$h_m(\ldots)$

\text{Prob a false negative even after m hash functions:}

\begin{align*}
\left( 1 - \left( 1 - \frac{d}{\lambda} \right)^k \right)^m &< \varepsilon
\end{align*}
The Tree of Life: Evolutionary Tree

Phenotype

Genotype

1) Parsimony-based methods

Assumption: all positions are independent

Add up the cost across all positions
Given: search over binary trees

1) search over different topologies

\[(2n-3)!! \text{ rooted trees}\]

\[= \left( \frac{2n-1}{2} \right) (2n-5) (2n-7) \ldots (2)\]  

Infeasible to enumerate all trees

\[\Rightarrow\] sampling

e.g. MCMC

Markov Chain Monte Carlo

2) how to compare the size of a given tree
bottom up:

node: If LNR = \( A \)

then LNR is kept

else

keep LNR

top-down phase:

starting at the root for any node

if more than one choice

use parent's symbol

Dynamic programming

find the least cost tree.