Pattern matching part four
Pattern matching with at most $\hat{k}$ mismatches

Benjamin Sach

Pattern matching with few mismatches ($\hat{k}$-mismatch)

Input: A text string $T$ (length $n$), a pattern string $P$ (length $m$) and a positive integer $\hat{k}$

Output: the number of mismatches... unless it's more than $\hat{k}$

We could use the $O(n \sqrt{m} \log m)$ time algorithm for Hamming distance...
but when $\hat{k}$ is small we can do much better

Pattern matching with at most $\hat{k}$ mismatches

Goal: For every $i$, output

$\text{Ham}_\hat{k}(i) = \left\{ \begin{array}{ll} \text{Ham}(i) & \text{if Ham}(i) \leq \hat{k} \\ \text{X} & \text{if Ham}(i) > \hat{k} \end{array} \right.$

Output the number of mismatches... unless it's more than $\hat{k}$

(we interpret the output $\text{X}$ to mean "too many mismatches")

\[ \text{LCP} - \text{the Longest Common Prefix} \]

For any pair of locations $i$ in $T$ and $j$ in $P$, $\text{LCP}(i, j)$ is the largest $\ell$ such that

\[ T[i+1..i+\ell-1] = P[j+1..j+\ell-1] \]

it's the furthest you can go before hitting a mismatch

\[ \hat{k} \text{-mismatch using LCP queries} \]

Find the leftmost (at most) $\hat{k} + 1$ mismatches between $T[1..i+m-1]$ and $P$

We do this for each $i$ separately.

We can do this using (at most) $\hat{k} + 1$ LCP queries

Each query takes $O(1)$ time and finds a new mismatch
Build the suffix tree for $T$ and preprocess it for LCA (Lowest Common Ancestor) queries in $O(n)$ prep. time and space.

What is the LCA of the leaves representing suffixes $i$ and $j$?
It's the node representing the longest common prefix of $T[i \ldots n - 1]$ and $T[j \ldots n - 1]$.

**Single string LCP:** For any pair of locations $i, j$ in $T$, $LCP(i, j)$ is the largest $k$ such that $T[i \ldots i + k - 1] = T[j \ldots j + k - 1]$.
So we have \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time for the LCP problem on a single string.

We can extend this two strings \( (T' \text{ and } T) \) by first concatenating them together . . . (and proceeding as for a single string).

We also have \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time for the LCP problem on two strings. I.e. for any \( j \) in \( T \) and \( j \) in \( P \), LCP \( (i, j) \) is the largest \( i \) such that \( T[i \ldots i + \ell - 1] = P[j \ldots j + \ell - 1] \) (as we originally defined it).

### \( k \)-mismatch using LCP queries

![Suffix Tree Diagram](image)

**LCPs in Suffix Trees**

This is the suffix tree of this text.

LCA \((i, j)\)

Build the suffix tree for \( T \) and preprocess it for LCA (Lowest Common Ancestor) queries in \( O(n) \) prep. time and space.

What is the LCA of the leaves representing suffixes \( i \) and \( j \)?

It’s the node representing the longest common prefix of \( T[i \ldots i + n - 1] \) and \( T[j \ldots j + n - 1] \).

We store the root-to-node length at each internal node so we can recover the length, LCP \( (i, j) \) in \( O(1) \) time.

So we have \( O(n) \) space, \( O(n) \) prep. time and \( O(1) \) query time for the LCP problem on a single string.

### \( k \)-mismatch using frequent/infrequent symbols

**Definition:** A symbol is frequent if it occurs at least \( \sqrt{k} \) times in \( P \), and infrequent otherwise.

### Algorithm summary

**Stage 0:** Classify each symbol as frequent or infrequent.

**Stage 1:** Count all matches involving frequent symbols (using cross-correlations as in last lecture).

**Stage 2:** Count all matches involving infrequent symbols (as in last lecture).

### Case 1: There are fewer than \( 2\sqrt{k} \) frequent symbols in \( P \), \( \cdot O(n \log m) \) total time.

**Case 2:** There are at least \( 2\sqrt{k} \) frequent symbols

Pick any \( 2\sqrt{k} \) frequent symbols and for each symbol pick \( \sqrt{k} \) occurrences in \( P \).

This gives us \( 2k \) interesting pattern locations, denoted \( J \).

\( J = \{0, 2, 3, 4, 5, 7, 9, 10\} \)

This gives us \( 2k \) interesting pattern locations, denoted \( J \).

\( J = \{0, 2, 3, 4, 5, 7, 9, 10\} \)

\( k = 4 \)

Let \( d_k(i) \) be the number of \( j \in J \) such that \( P[j] = T[i + j] \)

i.e. the number of single character matches involving interesting pattern locations.

Fact: if \( d_k(i) < k \) then there are more than \( k \) mismatches i.e. \( H_{2k}(i) = X \)

because there are \( 2k \) interesting positions . . . and fewer than \( k \) of them match.

Fact: There are at most \( \sqrt{n \sqrt{k}} \) values of \( i \) with \( d_k(i) = k \).

This follows from a counting argument.
**Case 2: There are at least \(2\sqrt{k}\) frequent symbols**

Pick any \(2\sqrt{k}\) frequent symbols and for each symbol pick \(\sqrt{k}\) occurrences in \(P\).

This gives us \(2\sqrt{k}\) interesting pattern locations, denoted \(J\).

\[
T = \{0, 2, 3, 4, 5, 7, 9, 10\}, \quad k = 4
\]

Let \(d_k(i)\) be the number of \(j \in J\) such that \(P_j = T_j[i+\ldots]\)

i.e. the number of (single character) matches involving interesting pattern locations

**Fact** There are at most \(n\sqrt{k}\) values of \(i\) with \(d_k(i) > k\)

For any location \(i\), \(T[p] = P[j]\) if either 0 or \(\sqrt{k}\) distinct \(j \in J\).

This implies that \(\sum d_k(i) \leq \sum_{j \in J} \sum_{i} \text{Eq}(T[p], P[j]) < n\sqrt{k}\)

**Conclusion**

Input: A text string \(T\) (length \(n\)), a pattern string \(P\) (length \(m\)) and a positive integer \(k\):

\[
T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, \quad k = 2
\]

\[
P = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}
\]

**Goal:** For all \(i\), output,

\[
\text{Ham}_k(i) = \begin{cases} 
\text{Ham}(i) & \text{if } \text{Ham}(i) \leq k \\
X & \text{if } \text{Ham}(i) > k 
\end{cases}
\]

Output the number of mismatches... unless it’s more than \(k\)

we interpret the output \(X\) to mean “too many mismatches”

We saw two algorithms for this problem:

One algorithm takes \(O(nk)\) time

The other algorithm takes \(O(n\sqrt{k} \log m)\) time (improvable to \(O(n\sqrt{k} \log k)\) time)