Pattern matching part four
Pattern matching with at most $k$ mismatches

Benjamin Sach
Pattern matching with mismatches

Input: A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{cccccccccccc}
 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
  a & b & c & d & a & b & a & a & d & a & c & a & a \\
\end{array}
\]

\[
\begin{array}{cccc}
 0 & 1 & 2 & 3 \\
\hline
  a & b & d & a \\
\end{array}
\]

Goal: For every alignment $i$, output

$\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches…*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

**Goal:** For every alignment $i$, output the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

The Hamming distance is the number of mismatches…

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$

$Ham(4) = 1$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th>$T$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$a$</th>
<th>$b$</th>
<th>$a$</th>
<th>$a$</th>
<th>$d$</th>
<th>$a$</th>
<th>$c$</th>
<th>$a$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$a$</td>
<td>$b$</td>
<td>$d$</td>
<td>$a$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

$Ham(5) = 4$

**Goal:** For every *alignment* $i$, output

$Ham(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches…*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T & a & b & c & d & a & b & a & a & d & a & c & a & a \\
\hline
P & a & b & d & a \\
\end{array}
\]

Ham(6) = 1

**Goal:** For every *alignment* $i$, output

\[
\text{Ham}(i), \text{ the Hamming distance between } P \text{ and } T[i \ldots i + m - 1]
\]

*The Hamming distance is the number of mismatches…*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

**Goal:** For every alignment $i$, output $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

The Hamming distance is the number of mismatches…

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

**Goal:** For every alignment $i$, output $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$$T = \text{a b c d a b a a d a c a a}$$

$$P = \text{a b d a}$$

this is alignment 8

\[\text{Ham}(8) = 3\]

**Goal:** For every alignment $i$, output

\[\text{Ham}(i), \text{the Hamming distance between } P \text{ and } T[i \ldots i + m - 1]\]

*The Hamming distance is the number of mismatches…*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

Input string $T$:  
```
0 1 2 3 4 5 6 7 8 9 10 11 12
T: a b c d a b a a d a c a a
```

Pattern string $P$:  
```
0 1 2 3 4 5 6 7 8 9 10 11 12
P: a b d a
```

This is alignment 8.  

**Goal:** For every alignment $i$, output

$\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches…*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$

Last lecture we saw two algorithms for this problem:
Pattern matching with mismatches

**Input:** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

\[
T = \begin{array}{ccccccc}
a & b & c & d & a & b & a \\
d & a & c & a & a \\
\end{array}
\]

\[
P = \begin{array}{cccc}
a & b & d & a \\
\end{array}
\]

**Goal:** For every *alignment* $i$, output $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

The Hamming distance is the number of mismatches…

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$

Last lecture we saw two algorithms for this problem:

One algorithm takes $O(n|\Sigma| \log m)$ time (where $|\Sigma|$ is the alphabet size)

The other algorithm takes $O(n\sqrt{m \log m})$ time (regardless of the alphabet size)
Pattern matching with few mismatches ($k$-mismatch)

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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”)*
Pattern matching with few mismatches ($k$-mismatch)

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

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”)*
Pattern matching with few mismatches ($k$-mismatch)

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

![Text and Pattern Strings]

**Goal:** For all $i$, output,

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

Output the number of mismatches... unless it's more than $k$

*(we interpret the output $X$ to mean “too many mismatches”)*
Pattern matching with few mismatches ($k$-mismatch)

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

$P$ | a | b | d | a |

$T$ has length $n$, $P$ has length $m$. $k = 2$

**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”)*
Pattern matching with few mismatches ($k$-mismatch)

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

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

$k = 2$

$T$

| a | b | c | d | a | b | a | a | d | a | a | a |

$P$

| a | b | d | a |

$Ham_k(6) = 1$

**Goal:** For all $i$, output,

$$Ham_k(i) = \begin{cases} 
Ham(i) & \text{if } Ham(i) \leq k \\
X & \text{if } Ham(i) > k 
\end{cases}$$

Output the number of mismatches... unless it's more than $k$

*(we interpret the output $X$ to mean “too many mismatches”)*
Pattern matching with few mismatches ($k$-mismatch)

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

**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")*
Pattern matching with few mismatches ($k$-mismatch)

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

![Text and Pattern Strings]

**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”)*
Pattern matching with few mismatches ($k$-mismatch)

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

- $T = a\ b\ c\ d\ a\ b\ a\ a\ d\ a\ a\ a$
- $P = a\ b\ d\ a$

**Goal:** For all $i$, output,

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

*Output the number of mismatches... unless it's more than $k$*

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

- We could use the $O(n\sqrt{m \log m})$ time algorithm for Hamming distance...
Pattern matching with few mismatches ($k$-mismatch)

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

**Goal:** For all $i$, output,

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

Output the number of mismatches... unless it's more than $k$

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

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

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

*it's the furthest you can go before hitting a mismatch*
For any pair of locations $i$ in $T$ and $j$ in $P$, $\text{LCP}(i, j)$ is the largest $\ell$ such that

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

*it's the furthest you can go before hitting a mismatch*
For any pair of locations $i$ in $T$ and $j$ in $P$, $\text{LCP}(i, j)$ is the largest $\ell$ such that

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

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

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

*it’s the furthest you can go before hitting a mismatch*
LCP - 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

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

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

\(\text{LCP}(i, j)\) returns 4
LCP - 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

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

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

$LCP(i, j)$ returns 0
LCP - 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

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

it's the furthest you can go before hitting a mismatch
LCP - the Longest Common Prefix

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

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

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

We can preprocess $P$ and $T$ for LCP queries in $O(n)$ time and $O(n)$ space
LCP - 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

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

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

We can preprocess $P$ and $T$ for LCP queries in $O(n)$ time and $O(n)$ space.

Each query then takes $O(1)$ time.
LCP - 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

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

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

We can preprocess \( P \) and \( T \) for LCP queries in \( O(n) \) time and \( O(n) \) space

Each query then takes \( O(1) \) time

we'll see how to do this later in this lecture
LCP - 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

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

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

We can preprocess $P$ and $T$ for LCP queries in $O(n)$ time and $O(n)$ space

Each query then takes $O(1)$ time

we'll see how to do this later in this lecture

First let's see how

we can use LCP queries to solve the $k$-mismatch problem...
$k$-mismatch using LCP queries

$T$

| a | b | c | b | a | b | a | b | c | a | b | a | b | a |

$P$

| b | c | b | c | a | a | b | a | a | b | a |

$n$
$\kappa$-mismatch using LCP queries

$T$

\[ a \ b \ c \ b \ a \ b \ a \ b \ c \ a \ b \ a \ b \ a \]

$P$

\[ b \ c \ b \ c \ a \ a \ b \ a \ a \ b \ a \]
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)
$k$-mismatch using LCP queries

Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$
(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries
Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

\textit{each query takes } \( O(1) \) \textit{time and finds a new mismatch}
Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

*each query takes \( O(1) \) time and finds a new mismatch*
$k$-mismatch using LCP queries

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

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
\( k \)-mismatch using LCP queries

Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

*each query takes \( O(1) \) time and finds a new mismatch*
Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) seperately)

We can do this using (at most) \( k + 1 \) LCP queries

\( \text{each query takes } O(1) \text{ time and finds a new mismatch} \)
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

each query takes $O(1)$ time and finds a new mismatch
Find the leftmost \((k + 1)\) mismatches between \(T[i \ldots i + m - 1] \) and \(P\)

(we do this for each \(i\) separately)

We can do this using (at most) \(k + 1\) LCP queries

each query takes \(O(1)\) time and finds a new mismatch
$k$-mismatch using LCP queries

Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

Each query takes \( O(1) \) time and finds a new mismatch
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
$k$-mismatch using LCP queries

Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

each query takes $O(1)$ time and finds a new mismatch
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
\( k \)-mismatch using LCP queries

Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

\textit{each query takes } \( O(1) \) \textit{time and finds a new mismatch}
Find the leftmost (at most) \( k + 1 \) mismatches between \( T[i \ldots i + m - 1] \) and \( P \) (we do this for each \( i \) separately)

We can do this using (at most) \( k + 1 \) LCP queries

*each query takes \( O(1) \) time and finds a new mismatch*
$k$-mismatch using LCP queries

Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$
(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

each query takes $O(1)$ time and finds a new mismatch
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
$k$-mismatch using LCP queries

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

(we do this for each $i$ seperately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
Find the leftmost (at most) $k+1$ mismatches between $T[i \ldots i+m-1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k+1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*
$k$-mismatch using LCP queries

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

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*

(or we reach the end of $P$)
$k$-mismatch using LCP queries

Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$ (we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

each query takes $O(1)$ time and finds a new mismatch
(or we reach the end of $P$)
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ seperately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*

(or we reach the end of $P$)

We can therefore calculate $\text{Ham}_k(i)$ for a single alignment $i$ in $O(k)$ time
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*

(or we reach the end of $P$)

We can therefore calculate $\text{Ham}_k(i)$ for a single alignment $i$ in $O(k)$ time

Overall this takes $O(nk)$ time (including the ‘preprocessing’ for LCP queries)
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m - 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*

(or we reach the end of $P$)

We can therefore calculate $\text{Ham}_k(i)$ for a single alignment $i$ in $O(k)$ time

Overall this takes $O(nk)$ time (including the ‘preprocessing’ for LCP queries)

*this is pretty good but we can do better*
Find the leftmost (at most) $k + 1$ mismatches between $T[i \ldots i + m − 1]$ and $P$

(we do this for each $i$ separately)

We can do this using (at most) $k + 1$ LCP queries

*each query takes $O(1)$ time and finds a new mismatch*

(or we reach the end of $P$)

We can therefore calculate $\text{Ham}_k(i)$ for a single alignment $i$ in $O(k)$ time

Overall this takes $O(nk)$ time (including the ‘preprocessing’ for LCP queries)

*this is pretty good but we can do better but first... how do we answer those LCP queries?
LCPs in Suffix Trees

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

This is the suffix tree of this text

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 \( j \)?
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 $j$?
LCPs in Suffix Trees

This is the suffix tree of this text

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 $j$?
LCPs in Suffix Trees

This is the suffix tree of this text

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 $j$?
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T[j \ldots n - 1]$
LCPs in Suffix Trees

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 $T$, $\text{LCP}_T(i, j)$ is the largest $\ell$ such that $T[i \ldots i + \ell - 1] = T[j \ldots j + \ell - 1]$.
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T$, $\text{LCP}_T(i, j)$ is the largest $\ell$ such that $T[i \ldots i + \ell - 1] = T[j \ldots j + \ell - 1]$
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T$, $\text{LCP}_T(i, j)$ is the largest $\ell$ such that $T[i \ldots i + \ell - 1] = T[j \ldots j + \ell - 1]$
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T[j\ldots n-1]$
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T[j \ldots n - 1]$

We store the root-to-node length at each internal node

so we can recover the length, $LCP_T(i, j)$ in $O(1)$ time
LCPs in Suffix Trees

This is the suffix tree of this text

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 $T[j \ldots n - 1]$

We store the root-to-node length at each internal node so we can recover the length, $LCP_T(i, j)$ in $O(1)$ time.
LCPs in Suffix Trees

This is the suffix tree of this text

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 \( T[j \ldots n − 1] \).

We store the root-to-node length at each internal node so we can recover the length, \( \text{LCP}_{T}(i, j) \) 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.
LCPs in Suffix Trees

This is the suffix tree of this text

So we have $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the LCP problem on a single string
LCPs in Suffix Trees

This is the suffix tree of this text

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 ($P$ and $T$) by first concatenating them together... (and proceeding as for a single string)
LCPs in Suffix Trees

This is the suffix tree of this text

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 ($P$ and $T$) by first concatenating them together...

(and proceeding as for a single string)

So we also have $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time for the LCP problem on two strings.
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 ($P$ and $T$) by first concatenating them together... (and proceeding as for a single string)

So 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 $i$ in $T$ and $j$ in $P$, $\text{LCP}(i, j)$ is the largest $\ell$ such that

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

(as we originally defined it)
\(k\)-mismatch using frequent/infrequent symbols

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

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

$m = 9$

$k = 4$

($\sqrt{k} = 2$)
\( k \)-mismatch using frequent/infrequent symbols

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

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

\[
P = \begin{array}{cccccccc}
a & b & b & a & c & a & d & b & d \\
\end{array}
\]

\[
k = 4 \\
(\sqrt{k} = 2)
\]
$k$-mismatch using frequent/infrequent symbols

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

$$ P = \begin{array}{cccccccc}
    a & b & b & a & c & a & d & b & d \\
\end{array} $$

$k = 4$

($\sqrt{k} = 2$)

$a$ is *frequent*
$k$-mismatch using frequent/infrequent symbols

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

\[ k = 4 \quad (\sqrt{k} = 2) \]

$P = \begin{array}{cccccc}
  a & b & b & a & c & a & d & b & d \\
\end{array}$

\[ a \text{ is frequent, } b \text{ is frequent} \]
Definition: A symbol is frequent if it occurs at least $\sqrt{k}$ times in $P$, and infrequent otherwise.

$k$-mismatch using frequent/infrequent symbols

\[
P = \begin{array}{cccccccc}
a & b & b & a & c & a & d & b & d \\
\end{array}
\]

\[
k = 4 \quad (\sqrt{k} = 2)
\]

$a$ is frequent, $b$ is frequent, $d$ is frequent.
\( k \)-mismatch using frequent/infrequent symbols

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

\[ P = \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
a & b & b & a & c & a & d & b & d
\end{array} \]

\( k = 4 \)  
(\( \sqrt{k} = 2 \))

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

$k = 4$ ($\sqrt{k} = 2$)

- $a$ is *frequent*, $b$ is *frequent*, $d$ is *frequent*
- $c$ is *infrequent*

*How many frequent symbols can there be?*
$k$-mismatch using frequent/infrequent symbols

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

$k = 4$ ($\sqrt{k} = 2$)

$a$ is frequent, $b$ is frequent, $d$ is frequent, $c$ is infrequent

*How many frequent symbols can there be?* **Lots!**
$k$-mismatch using frequent/infrequent symbols

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

- $k = 4$ ($\sqrt{k} = 2$)

$P$: 

```
0 1 2 3 4 5 6 7 8
```

- $a$ is frequent, $b$ is frequent, $d$ is frequent
- $c$ is infrequent

*How many frequent symbols can there be? Lots!* There could be $\frac{m}{\sqrt{k}}$ frequent symbols.
\(k\)-mismatch using frequent/infrequent symbols

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

\[
P = \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
a & b & b & a & c & a & d & b & d
\end{array}
\]

\(k = 4\) \((\sqrt{k} = 2)\)

- \(a\) is frequent
- \(b\) is frequent
- \(d\) is frequent
- \(c\) is infrequent

*How many frequent symbols can there be?* **Lots!** there could be \(\frac{m}{\sqrt{k}}\) frequent symbols

**Case 1:** There are fewer than \(2\sqrt{k}\) frequent symbols in \(P\).
Definition: A symbol is **frequent** if it occurs at least $\sqrt{k}$ times in $P$, and **infrequent** otherwise.

Case 1: There are fewer than $2\sqrt{k}$ frequent symbols in $P$.

Algorithm summary
\(k\)-mismatch using frequent/infrequent symbols

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

\[a \quad b \quad b \quad a \quad c \quad a \quad d \quad b \quad d\]

\[k = 4 \quad (\sqrt{k} = 2)\]

- \(a\) is *frequent*, \(b\) is *frequent*, \(d\) is *frequent*
- \(c\) is *infrequent*

*How many frequent symbols can there be?* **Lots!** there could be \(\frac{m}{\sqrt{k}}\) frequent symbols

**Case 1:** There are fewer than \(2\sqrt{k}\) frequent symbols in \(P\).

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

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

![Symbol Classification](image)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

\( k = 4 \) \((\sqrt{k} = 2)\)

- \( a \) is frequent, \( b \) is frequent, \( d \) is frequent
- \( c \) is infrequent

**How many frequent symbols can there be?** \textbf{Lots!} there could be \( \frac{m}{\sqrt{k}} \) frequent symbols

**Case 1:** There are fewer than \( 2\sqrt{k} \) frequent symbols in \( P \).

**Algorithm summary**

**Stage 0:** Classify each symbol as frequent or infrequent \(- O(m \log m) \) time

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

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

\[
P = \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{a} & \text{b} & \text{b} & \text{a} & \text{c} & \text{a} & \text{d} & \text{b} & \text{d}
\end{array}
\]

\( k = 4 \) 
\( (\sqrt{k} = 2) \)

\( a \) is frequent, \( b \) is frequent, \( d \) is frequent 
\( c \) is infrequent

*How many frequent symbols can there be?* **Lots!** there could be \( \frac{m}{\sqrt{k}} \) frequent symbols

**Case 1:** There are fewer than \( 2\sqrt{k} \) frequent symbols in \( P \).

**Algorithm summary**

- **Stage 0:** Classify each symbol as frequent or infrequent - \( O(m \log m) \) time
- **Stage 1:** Count all matches involving frequent symbols (using cross-correlations as in last lecture) - \( O(n\sqrt{k} \log m) \) time
- **Stage 2:** Count all matches involving infrequent symbols (as in last lecture)
**$k$-mismatch using frequent/infrequent symbols**

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

![Sequence P with frequent and infrequent symbols](image)

$k = 4$ \quad ($\sqrt{k} = 2$)

- $a$ is frequent, $b$ is frequent, $d$ is frequent
- $c$ is infrequent

*How many frequent symbols can there be?* **Lots!** there could be $\frac{m}{\sqrt{k}}$ frequent symbols

**Case 1:** There are fewer than $2\sqrt{k}$ frequent symbols in $P$.

**Algorithm summary**

**Stage 0:** Classify each symbol as frequent or infrequent \[- O(m \log m) \text{ time} \]

**Stage 1:** Count all matches involving frequent symbols (using cross-correlations as in last lecture) \[- O(n\sqrt{k} \log m) \text{ time} \]

**Stage 2:** Count all matches involving infrequent symbols (as in last lecture) \[- O(n\sqrt{k}) \text{ time} \]
$k$-mismatch using frequent/infrequent symbols

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

![Symbol occurrence](image)

$k = 4$  
($\sqrt{k} = 2$)

$P = \text{a b b a c a d b d}$

- $a$ is *frequent*, $b$ is *frequent*, $d$ is *frequent*
- $c$ is *infrequent*

**How many frequent symbols can there be?** Lots! There could be $\frac{m}{\sqrt{k}}$ frequent symbols.

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

**Algorithm summary**

- **Stage 0:** Classify each symbol as *frequent* or *infrequent* - $O(m \log m)$ time
- **Stage 1:** Count all matches involving *frequent* symbols (using cross-correlations as in last lecture) - $O(n\sqrt{k} \log m)$ time
- **Stage 2:** Count all matches involving *infrequent* symbols (as in last lecture) - $O(n\sqrt{k})$ 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 $J$.
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

\[
P = \begin{array}{cccccccccccccc}
  a & e & b & b & a & c & a & d & b & d & c & f & b & b \\
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{array}
\]

$k = 4$

$a$ is frequent, $b$ is frequent, $c$ is frequent, $d$ is frequent, $e$ and $f$ are infrequent
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

$a$ is frequent, $b$ is frequent, $c$ is frequent, $d$ is frequent
$e$ and $f$ are infrequent
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 $J$

$k = 4$

$P = \begin{array}{cccccccccccc}
a & e & b & b & a & c & a & d & b & d & c & f & b & b \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{array}$

$a$ is frequent, $b$ is frequent, $c$ is frequent, $d$ is frequent

e and $f$ are infrequent
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 $J$

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

$a$ is frequent, $b$ is frequent, $c$ is frequent, $d$ is frequent
$e$ and $f$ are infrequent

$k = 4$
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

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

$k = 4$

\[ P = \begin{array}{cccccccc}
    a & e & b & b & a & c & a & d & b & d & c & f & b & b \\
\end{array} \]
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 $J$

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

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
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurances in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

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

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
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 $J$

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

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
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 $J$

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

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. $\text{Ham}_k(i) = X$)
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 $J$

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

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. $\text{Ham}_k(i) = X$)

because there are $2k$ interesting positions... and fewer than $k$ of them match
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 $J$

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

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. $\text{Ham}_k(i) = X$) because there are $2k$ interesting positions... and fewer than $k$ of them match

**Fact** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$
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 $J$

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. $\text{Ham}_k(i) = X$) because there are $2k$ interesting positions... and fewer than $k$ of them match

**Fact** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$
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 $2k$ interesting pattern locations, denoted $J$

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

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 There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$
Case 2: There are at least $2\sqrt{k}$ frequent symbols. Let $d_k(i)$ be the number of $j \in J$ such that $P[j] = T[i+j]$. This gives us $2k$ interesting pattern locations, denoted $J = \{0, 2, 3, 4, 5, 7, 9, 10\}$.

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

Fact: There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$. For any location $i'$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$.

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.
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

\[ J = \{0, 2, 3, 4, 5, 7, 9, 10\} \]

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** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

For any location $i'$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

This implies that $\sum_i d_k(i) \leq \sum_{i'} \sum_{j \in J} \text{Eq}(T[i'], P[j]) \leq n\sqrt{k}$
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 $J$

\[ J = \{0, 2, 3, 4, 5, 7, 9, 10\} \]

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

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

For any location $i'$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

This implies that $\sum_i d_k(i) \leq \sum_{i'} \sum_{j \in J} \text{Eq}(T[i'], P[j]) \leq n\sqrt{k}$
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 $J$

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

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

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

For any location $i'$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

This implies that $\sum_i d_k(i) \leq \sum_{i'} \sum_{j \in J} \text{Eq}(T[i'], P[j]) \leq n\sqrt{k}$
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 $J$

$P = \text{[ae bb ac ad bd dc f b b]}$

$k = 4$

$T = \text{[ac cc a ab a bb a c fc dc e f f b b c e a e]}$

$i = 4$

$d_k(i) = 3$

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 There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

Assume that more than $n/\sqrt{k}$ values of $i$ have $d_k(i) \geq k$
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 $J$.

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

**Fact** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$.

Assume that more than $n/\sqrt{k}$ values of $i$ have $d_k(i) \geq k$.

So $\sum_i d_k(i) \geq \left( \frac{n}{\sqrt{k}} + 1 \right) \cdot k$. 

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

$k = 4$

$i = 4$

$d_k(i) = 3$
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 $J$

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

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** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

Assume that more than $n/\sqrt{k}$ values of $i$ have $d_k(i) \geq k$

So $\sum_i d_k(i) \geq \left(\frac{n}{\sqrt{k}} + 1\right) \cdot k > n\sqrt{k}$
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 $J$

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

Fact There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

Assume that more than $n/\sqrt{k}$ values of $i$ have $d_k(i) \geq k$

So $\sum_i d_k(i) \geq \left(\frac{n}{\sqrt{k}} + 1\right) \cdot k > n\sqrt{k}$

Contradiction!
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 $J$

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

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. $\text{Ham}_k(i) = X$) because there are $2k$ interesting positions... and fewer than $k$ of them match

Fact There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

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 \(2k\) *interesting* pattern locations, denoted \(J\)

\[
J = \{0, 2, 3, 4, 5, 7, 9, 10\}
\]

\(P\)

\[
\begin{array}{cccccccc}
  a & e & b & b & a & c & a & d & b & d & c & f & b & b \\
\end{array}
\]

\(k = 4\)

\(T\)

\[
\begin{array}{cccccccccccccccc}
  a & c & c & a & a & b & a & b & b & a & c & f & c & d & e & f & f & b & b & c & e & a & e \\
\end{array}
\]

\(i = 4\)

We can filter the text, using the \(d_k(i)\) values leaving only \(n/\sqrt{k}\) alignments to check

every other alignment has more than \(k\) mismatches
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 $J$

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

We can filter the text, using the $d_k(i)$ values leaving only $n/\sqrt{k}$ alignments to check every other alignment has more than $k$ mismatches

Check each of the remaining alignments using LCP queries in $O(k)$ time per alignment
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 $J$

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

We can filter the text, using the $d_k(i)$ values leaving only $n/\sqrt{k}$ alignments to check

every other alignment has more than $k$ mismatches

Check each of the remaining alignments using LCP queries in $O(k)$ time per alignment

This takes $n/\sqrt{k} \cdot O(k) = O(n\sqrt{k})$ 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 $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

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

We can filter the text, using the $d_k(i)$ values leaving only $n/\sqrt{k}$ alignments to check

every other alignment has more than $k$ mismatches

Check each of the remaining alignments using LCP queries in $O(k)$ time per alignment

This takes $n/\sqrt{k} \cdot O(k) = O(n\sqrt{k})$ total time.

How do we compute all the $d_k(i)$ values?
Computing all the $d_k(i)$ values

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 $J$

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

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

Let $d_k(i) = 0$ for all $i$.

For each text character $T[i']$, For each $j \in J$ such that $P[j] = T[i']$

increase $d_k(i' - j)$ by one
Computing all the $d_k(i)$ values

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 $J$

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

Let $d_k(i) = 0$ for all $i$.

For each text character $T[i']$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

(Store a list of $j$ values for each symbol)

For each $j \in J$ such that $P[j] = T[i']$

increase $d_k(i' - j)$ by one
Computing all the $d_k(i)$ values

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

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

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

Let $d_k(i) = 0$ for all $i$.

For each text character $T[i']$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

(store a list of $j$ values for each symbol)

For each $j \in J$ such that $P[j] = T[i']$

increase $d_k(i' - j)$ by one

This takes $O(n\sqrt{k})$ total time.
Pattern matching with k-mismatches: putting it all together

Algorithm summary
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $\mathbf{P}, \mathbf{T}$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $\mathbf{P}$ - $O(m \log m)$ time
Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of *frequent* symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

Count matches with frequent symbols using cross-correlations - $O(n\sqrt{k} \log m)$ time

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

- Count matches with frequent symbols using cross-correlations - $O(n\sqrt{k} \log m)$ time
- Count matches with infrequent symbols directly - $O(n\sqrt{k})$ time

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

Count matches with frequent symbols using cross-correlations - $O(n\sqrt{k} \log m)$ time

Count matches with infrequent symbols directly - $O(n\sqrt{k})$ time

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols

Filter the text, leaving $n/\sqrt{k}$ alignments - $O(n\sqrt{k})$ time
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

Count matches with frequent symbols using cross-correlations - $O(n\sqrt{k} \log m)$ time

Count matches with infrequent symbols directly - $O(n\sqrt{k})$ time

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols

Filter the text, leaving $\frac{n}{\sqrt{k}}$ alignments - $O(n\sqrt{k})$ time

Count mismatches at these alignments using LCP queries - $O(n\sqrt{k})$ time
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess $P, T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

Case 1: $P$ has at most $2\sqrt{k}$ frequent symbols

- Count matches with frequent symbols using cross-correlations - $O(n\sqrt{k} \log m)$ time
- Count matches with infrequent symbols directly - $O(n\sqrt{k})$ time

Case 2: $P$ has more than $2\sqrt{k}$ frequent symbols

- Filter the text, leaving $n/\sqrt{k}$ alignments - $O(n\sqrt{k})$ time
- Count mismatches at these alignments using LCP queries - $O(n\sqrt{k})$ time

*Overall, we obtain a time complexity of $O(n\sqrt{k} \log m)$.*
Pattern matching with k-mismatches: putting it all together

Algorithm summary

Preprocess \( P, T \) for LCP queries - \( O(n) \) time

Count the number of frequent symbols in \( P \) - \( O(m \log m) \) time

Case 1: \( P \) has at most \( 2\sqrt{k} \) frequent symbols

Count matches with frequent symbols using cross-correlations - \( O(n\sqrt{k} \log m) \) time

Count matches with infrequent symbols directly - \( O(n\sqrt{k}) \) time

Case 2: \( P \) has more than \( 2\sqrt{k} \) frequent symbols

Filter the text, leaving \( n/\sqrt{k} \) alignments - \( O(n\sqrt{k}) \) time

Count mismatches at these alignments using LCP queries - \( O(n\sqrt{k}) \) time

*Overall, we obtain a time complexity of \( O(n\sqrt{k} \log m) \).*

- This can be improved to \( O(n\sqrt{k} \log k) \)
Conclusion

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

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)