lecture 4: Text algorithms

Plan
1. Pattern matching - Naive, KMP
2. Automata
3. Multiple pattern matching - Aho-Coraminick, suffix trees
4. Dictionary look-up: BST, hashing

1. Pattern matching

Definitions
- Alphabet, string, substring, matrix, prefix, occurrences

Problem 1: Given a string $T$ of length $n$ ("text"), and a string $P$ of length $m$ ("pattern"), find all occurrences of $P$ in $T$.

- Naive algorithm - $O(nm)$ time
- Knuth-Morris-Pratt algorithm - $O(m+n)$ time

Definition: Border - longest proper prefix of $P$ equal to suffix of $P$

Definition: Border array $B$ of $P$ - array of length $m$ s.t.

$B[i]$ is the length of the longest border of $P[1..i]$

Limit $B$ can be computed in $O(n)$ time.

Shift and skip rules:

- $j = i + B[i]$
- $T[p+1..x] = P[1..p+1]$

<table>
<thead>
<tr>
<th>$B$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>3</td>
<td>ab</td>
</tr>
<tr>
<td>4</td>
<td>ab</td>
</tr>
<tr>
<td>5</td>
<td>ab</td>
</tr>
</tbody>
</table>

Limit: No occurrence of $P$ can start between $j$ and $j + (i + 1 - B[i])$

Limit: For the position $j + (i + 1 - B[i])$, we don't need to compare the first $B[i]$ letters of $P$ with $T$.

The algorithm finds all occurrences of $P$ in $T$. The running time of the algorithm is $O(n+m)$.

- Two pointers: $i + j$, the letter of $T$ we are to compare with $P$. Their min increases by one at any step.
- $lm[i], lm[i+1]$ give correctness.
Properties of $B$

1. $B[1] = 0$
2. $b^i = \text{border of } P[1,k] \implies b^i$ is a border of $P[1,k-1]$, where $b^i$ is a portion of $b$ of length $1b1-1$


4. $b - \text{border of } S, \quad b^i - \text{border of } S^i$

Suppose that we have already computed $B[1], B[2], \ldots, B[k-1]$. We will now compute $B[k]$. If $P[k] = P[B[k-1]+1]$, we have $B[k] = B[k-1] + 1$ (Property 3). Let $P[k] \neq P[B[k-1]+1]$.


The algorithm computes $B[k]$ correctly.

We will show a stronger claim: the set of the borders of $P[1,k]$ is a subset of \{ $B[k], B[k]+1, \ldots, B[k]+n$ \}. From property 2, it follows that any border of $P[1,k]$ is obtained by appending a letter to some border of $P[1,k-1]$. Moreover, $B[k]$ is the $i$-th longest border of $P[1,k-1]$ by property 4.

The algorithm requires $O(n)$ time.

If $B[k] = B[k-1] + 1$, then I need $j$ steps. On the other side,

- $1 \leq \Sigma (B[k] - B[k-1]) \leq n$
- $-1 \leq \Sigma^+ (B[k] - B[k-1]) - \Sigma^- (B[k] - B[k-1]) \leq n$

$\Rightarrow \Sigma^- (B[k] - B[k-1]) = 0 \quad \Rightarrow \quad |B[k] - B[k-1]| \leq 2n$
A deterministic finite automaton is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \), where:
- \( Q \) = finite set of states
- \( \Sigma \) = alphabet
- \( \delta : Q \times \Sigma \to Q \) = transition function
- \( q_0 \) = start state
- \( F \subseteq Q \) = accept states.

An automaton reads a finite string \( S = a_1 \ldots a_n \), where \( a_i \in \Sigma \). A sequence of states \( q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n \), where \( q_{i+1} = \delta(q_i, a_{i+1}) \) is a run of the automaton on \( S \).

**KMP as an automaton**

**Problem 2** Given a string \( T \) of length \( n \), and a set of strings of total length \( m \) find all occurrences of the patterns in \( T \).

**Assumption** No pattern is a substring of another one.

\[ P = \{ abc, bca, ca, abc \} \]
Algorithm

We will maintain a pointer current_node. We initialize current_node with root. We also maintain the current position i in T. At each step we check whether there is a child u of current_node such that the edge (current_node, u) is labelled by T[i]. If it exists, we set current_node = u, i = i + 1. Otherwise, we set current_node = root. To do this, we set the failure link of current_node. If we arrive at a node labelled by a pattern, we report an occurrence.

The trie occupies O(n) space.

The algorithm takes O(n) time.

Averaged analysis: the depth of current_node can either increase by one (εn times), or decrease by one when taking a failure link.

The algorithm is correct. Similar to KMP.

Preprocessing trie and the failure links

To compute the trie, we simply add the strings in P to the trie one-by-one.

We first build all failure links for nodes of depth p, then 2p, ..., consider a node u. Consider the parent of u and the path of failure links outgoing from it. Let u be the first node on this path such that it has a child v and the letter on (v, v') is equal to the letter on (parent(v), v). Then (v, v') is a failure link.

We need O(n) time to build the failure links.

Consider one root-to-leaf path. To build the links for nodes in this path we need

\[ c \leq \ell_p(v_2) - \ell_p(v_1) + c \leq (p_1) - p_2 + c \leq (p_2) - p_3 + \cdots + \ell_p(v_{n+1}) - \ell_p(v_n) \leq 1, \]

so the same as for (KM).

\[ \ell_p(v_n) - \ell_p(v_1) \leq 1. \]
Removing the assumption

LM5 If the failure link path from current node
contains a node \( \emptyset \), then there is an occurrence
of \( p_i \) ending at the current position of \( T \).

Exit link: from \( p_i \) to the nearest node on the failure
link path that has a mark \( \textcircled{1} \).

Algorithm

Extra step: follow the path of exit links from
the current node and report occurrences.

Theorem A.1: correctly identifies all occurrences of
the patterns in \( T \). It requires \( O(m) \) space and \( O(a + m + 1) \) time.

Problem 3: Given a set of strings (dictionary), preprocess it into
a data structure to maintain the following queries: Given
a string \( S \), decide if it is in the dictionary.

\[ \text{Trie} \quad \text{Space} = O(m), \quad \text{Query time} = O(1 + |S|), \quad \text{update time linear} \]
Suffix trees

- Append $\varepsilon$
- Suffixes: $abc\varepsilon$, $bc\varepsilon$, $ca\varepsilon$, $ab\varepsilon$, $b\varepsilon$
- Trie on suffixes

Can be used to answer pattern matching in $O(1 + k)$ time

**[Ukkonen's algorithm]** Suffix tree can be built in $O(n)$ time

Using suffix trees to find the LCS of two strings.