Applications of String Functions

Searching for a Sub-string in a String (KMP)

• Concatenate the substring to the beginning of the string, separated by a character that does not occur in either string.
• Run the prefix function on the concatenated string and count how many times the length of the substring appears.

Counting the Number of Prefixes in a String

• Run the prefix function on the string.
• For each prefix of length i: ans[pi[i-1]] += ans[i]
• Add one for each original (non-counted) prefix.

```
vll ans(n+1, 1); // Set to one for the actual prefix
for (int i = 0; i < n; i++) ans[pi[i]]++;
for (int i = n-1; i > 0; i--) ans[pi[i-1]] += ans[i];
```

The Number of Different Substrings in a String

• Start with an empty string.
• Add the next character of the original string and reverse.
• Run the prefix function to find the maximum length of substrings already found.
• New substrings added = |S| + 1 - maxPi, accumulated over all steps.

Compressing a String

• Given a string s of length n, we want the shortest compressed representation.
• After running the prefix function:
  – If n % (n - pi[n-1]) == 0, the string is compressible, with minimum size n - pi[n-1].
  – Otherwise, it is not compressible.

Creating Automation for String Matching

• After preprocessing a string with the prefix function, we can build an automaton to allow matching in O(m) per query (after O(n) preprocessing).
• This is useful for many repeated searches with the same pattern (KMP alone would take O(n+m) per query).
• Build a table aut[state][charNumber] where:
  – state = index in the pattern
  – charNumber = current character
• Depending on the character, move forwards or backwards to a new state.
  – Example: reading 'a' might move us from state 1 -> 2, while reading 'b' moves state 2->0.