String Matching

- detecting the occurrence of a particular substring (pattern) in another string (text)
- A straightforward Solution
- The Knuth-Morris-Pratt Algorithm
- The Boyer-Moore Algorithm

Straightforward solution

- Algorithm: Simple string matching
- Input: P and T, the pattern and text strings; m, the length of P. The pattern is assumed to be nonempty.
- Output: The return value is the index in T where a copy of P begins, or -1 if no match for P is found.

```
P: ABABC  ABABC  ABABC
   ↓↓↓↓↓  ↓  ↓↓↓↓↓  ↓↓↓↓↓
T: ABABBCCA  ABABBCCA  ABABBCCA
   ↑                      ↑
Successful match
```

Analysis

- Worst-case complexity is in $\Theta(mn)$
- Need to back up.
- Works quite well on average for natural language.

The Knuth-Morris-Pratt Algorithm

- Pattern Matching with Finite Automata
  - e.g. P = “AABC”

The Knuth-Morris-Pratt Flowchart

- Character labels are inside the nodes
- Each node has two arrows out to other nodes: success link, or fail link
- Next character is read only after a success link
- A special node, node 0, called “get next char” which read in next text character.
  - e.g. P = “ABACBC”
Construction of the KMP Flowchart

- Definition: Fail links
  - We define fail[k] as the largest r (with r < k) such that $p_1, p_2, \ldots, p_{r-1}$ matches $p_{k-r+1}, p_{k-r+2}, \ldots, p_{k-1}$. That is, the (r-1) character prefix of P is identical to the one (r-1) character substring ending at index k-1. Thus, the fail links are determined by repetition within P itself.

The Knuth-Morris-Pratt Scan Algorithm

- int kmpScan(char[] P, char[] T, int m, int[] fail)
  - int match, j, k;
  - match = -1;
  - j = 1; k = 1;
  - while(endText(T, j) == false)
    - if (k == m)
      - match = j - m;
      - break;
    - if (T[j] == P[k])
      - j++; k++;
    - else
      - // Follow fail arrow.
      - k = fail[k];
  - // Continue loop.
  - return match;

The Boyer-Moore Algorithm

- The new idea
  - first heuristic
    - e.g. scan from right to left, jump forward …
  - Find “must” in
    - If you wish to understand you must...

  - must
    - 1 1 1 1 1 1 1 1 1 1 1 2 1 1
    - If you wish to understand you must..

Algorithm: KMP flowchart construction

- Input: Pa string of characters; m, the length of P.
- Output: fail, the array of failure links, defined for indexes 1, ..., m. The array is passed in and the algorithm fills it.
- Step:
  - void kmpSetup(char[] P, int m, int[] fail)
    - int k, s
    - 1. fail[1] = 0;
    - 2. for (k = 2; k <= m; k++)
      - 3. s = fail[k-1];
      - 4. while (s > 1)
      - 5. if (P[k-1] == P[s])
      - 6. s = fail[s];
      - 7. fail[k] = s + 1;

Analysis

- KMP Flowchart Construction require 2m – 3 character comparisons in the worst case
- The scan algorithm requires 2n character comparisons in the worst case
- Overall: Worst case complexity is θ(n+m)

Algorithm: Computing Jumps for the Boyer-Morre Algorithm

- Input: Pattern string P: m the length of P, alphabet size alpha = |Σ|
- Output: Array charJump, defined on indexes 0, ..., alpha-1. The array is passed in and the algorithm fills it.
- void computeJumps(char[] P, int m, int alpha, int[] charJump)
  - char ch; int k;
  - for (ch = 0; ch < alpha; ch++)
    - charJump[ch] = m;
  - for (k = 1; k <= m; k++)
    - charJump[P[k]] = m - k;
If you wish to understand you must

• ...

...