Searching a list of strings

Maxime Crochemore

King's College London

Maxime.Crochemore@kcl.ac.uk
http://www.dcs.kcl.ac.uk/staff/mac/

Searching problem

⋆ **Input**

- a list L of n strings of Σ^* stored in increasing lexicographic order in a table: $L_0 \leq L_1 \leq \cdots \leq L_{n-1}$
- a string $x \in \Sigma^*$ of length m.

⋆ **Simple problem**

find

- either i, $-1 < i < n$, with $x = L_i$ if x occurs in L,
- or d and f, $-1 \leq d < f \leq n$, that satisfy $d + 1 = f$ and $L_d < x < L_f$ otherwise.

⋆ **Interval**

find d and f, $-1 \leq d < f \leq n$, with:
$d < i < f$ if and only if x prefix of L_i.

Example

- ⋆ **List** L

  $L_0 =$ `a a a b a a`

  $L_1 =$ `a a a b b`

  $L_2 =$ `a a b b b b`

  $L_3 =$ `a b`

  $L_4 =$ `b a a a`

  $L_5 =$ `b b`

- ⋆ **Search**

  $x =$ `a a a b b` $\longrightarrow 1$

  $x =$ `a a b a` $\longrightarrow (1, 2)$

- ⋆ **Interval**

  $x =$ `a a` $\longrightarrow (-1, 3)$

Searching algorithm

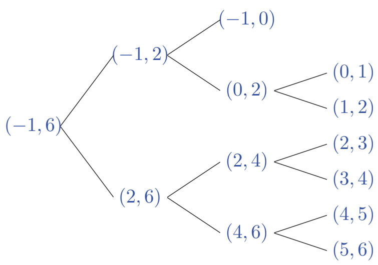
```
SIMPLE-SEARCH(L, n, x, m)
d ← −1
f ← n
while d + 1 < f do                    ▷ Invariant: L_d < x < L_f
      i ← ⌊(d + f)/2⌋
      ℓ ← |lcp(x, L_i)|
      if ℓ = m and ℓ = |L_i| then
            return i
      elseif (ℓ = |L_i|) or (ℓ ≠ m and L_i[ℓ] < x[ℓ]) then
            d ← i
      else f ← i
return (d, f)
```

Complexity

- **Running time**
  $O(m \times \log n)$
- **Worst case**
  - list $L = (\mathtt{a}^{m-1}\mathtt{b}, \mathtt{a}^{m-1}\mathtt{c}, \mathtt{a}^{m-1}\mathtt{d}, \ldots)$
  - string $x = \mathtt{a}^m$
- **Additional space**
  constant

Binary search tree

- **Nodes**
  $n+1$ external nodes $(-1,0), (0,1), (1,2), \ldots, (n-1,n)$
  n internal nodes in the form (d,f) with $d+1 < f$
  children of (d,f): $(d, \lfloor (d+f)/2 \rfloor)$ and $(\lfloor (d+f)/2 \rfloor, f)$
  root: $(-1,n)$
- **Size**
  $2n+1$ nodes for a list of n strings
- **Example for** $n = 6$



Search using LCP's

- **Aim**
  reduce the running time to $O(m + \log n)$
- **LCP**, *longest common prefix*
  $lcp(L_d, L_f)$ known for any pair (d, f) considered in the binary search
- **Additional space**: $O(n)$ integers for the $2n+1$ LCP's associated with nodes of the binary search tree
- **Algorithm** based on properties arising in three cases (plus symmetric cases)
- **Variables**
  $ld = |lcp(x, L_d)|$, $lf = |lcp(x, L_f)|$, $i = \lfloor (d + f)/2 \rfloor$
  maintained during execution

Case one

- **Hypotheses**
  $L_d < x < L_f$ and $ld \leq |lcp(L_i, L_f)| < lf$
- **Example**

L_d	`a a a c a`	x	`a a b b b a a`
	`a a a c b a`		
L_i	`a a b b a b a`		
	`a a b b a b b`		
L_f	`a a b b b a b`	x	`a a b b b a a`

- **Conclusion**
  $L_i < x < L_f$ and $|lcp(x, L_i)| = |lcp(L_i, L_f)|$

Case two

- **Hypotheses**

  $L_d < x < L_f$ and $ld \leq lf < |lcp(L_i, L_f)|$

- **Example**

L_d	`a a a c a`	x	`a a b a c b`
	`a a a c b a`		
L_i	`a a b b a b a`		
	`a a b b a b b`		
L_f	`a a b b b a b`	x	`a a b a c b`

- **Conclusion**

  $L_d < x < L_i$ and $|lcp(x, L_i)| = |lcp(x, L_f)|$

Case three

- **Hypotheses**

  $L_d < x < L_f$ and $ld \leq lf = |lcp(L_i, L_f)|$

- **Example**

L_d	`a a a c a`	x	`a a b b a b`
	`a a a c b a`		
L_i	`a a b b a b a`		
	`a a b b a b b`		
L_f	`a a b b b a b`	x	`a a b b a b`

- **Conclusion**

  compare x and L_i from position lf

Improved searching algorithm

Search(L, n, Lcp, x, m)
$(d, ld) \leftarrow (-1, 0)$
$(f, lf) \leftarrow (n, 0)$
while $d + 1 < f$ **do** ▷ Invariant : $L_d < x < L_f$
    $i \leftarrow \lfloor (d + f)/2 \rfloor$
    if $ld \leq Lcp(i, f) < lf$ **then**
        $(d, ld) \leftarrow (i, Lcp(i, f))$
    elseif $ld \leq lf < Lcp(i, f)$ **then**
        $f \leftarrow i$
    elseif $lf \leq Lcp(d, i) < ld$ **then**
        $(f, lf) \leftarrow (i, Lcp(d, i))$
    elseif $lf < ld < Lcp(d, i)$ **then**
        $d \leftarrow i$
    else $\ell \leftarrow \max\{ld, lf\}$
        $\ell \leftarrow \ell + |lcp(x[\ell \,.\,.\, m - 1], L_i[\ell \,.\,.\, |L_i| - 1])|$
        if $\ell = m$ and $\ell = |L_i|$ **then**
            return i
        elseif $(\ell = |L_i|)$ or $(\ell \neq m$ and $L_i[\ell] < x[\ell])$ **then**
            $(d, ld) \leftarrow (i, \ell)$
        else $(f, lf) \leftarrow (i, \ell)$
return (d, f)

Complexity

Proposition 1 *Algorithm* Search *finds a string x of length m in a sorted list of n strings in time $O(m + \log n)$.*
It makes no more than $m + \lceil \log(n + 1) \rceil$ comparisons of letters.
It requires $O(n)$ extra space.

Sketch of the proof

Number of letter comparisons:

- each positive comparison strictly increases ℓ, yielding no more than m such comparisons.
- each negative comparison leads to divide by two the value of $f - d$, producing no more than $\lceil \log(n + 1) \rceil$ such comparisons.

Lcp can be implemented to run in constant time after preprocessing.

Interval

```
▷ next line replaces line 15 of SEARCH
if ℓ = m then
    ▷ next lines replace line 16 of SEARCH
    e ← i
    while d + 1 < e do
        j ← ⌊(d + e)/2⌋
        if Lcp(j, e) < m then
            d ← j
        else e ← j
    if Lcp(d, e) ≥ m then
        d ← max{d − 1, −1}
    e ← i
    while e + 1 < f do
        j ← ⌊(e + f)/2⌋
        if Lcp(e, j) < m then
            f ← j
        else e ← j
    if Lcp(e, f) ≥ m then
        f ← min{f + 1, n}
    return (d, f)
```


Preprocessing the list

Let $||L|| = \Sigma_{i=0}^{n-1} |L_i|$.

- **Sorting**
  repetitive application of bin sorting: time $O(||L||)$
- **Computing LCP's** of L_{f-1} and L_f, $0 \leq f \leq n$
  straight algorithm: time $O(||L||)$
- **Computing other LCP's**
  based on next lemma

Lemma 1 *Let $L_0 \leq L_1 \leq \ldots \leq L_{n-1}$. Let d, i and f, $-1 < d < i < f < n$. Then $|lcp(L_d, L_f)| = \min\{|lcp(L_d, L_i)|, |lcp(L_i, L_f)|\}$.*

Proposition 2 *Preprocessing L, sorting and computing LCP's, takes $O(||L||)$ time.*