CAOS seminar series, February 3, 2005

Dynamic range reporting in one dimension on a RAM

(to appear at STOC 2005)

Rasmus Pagh

Dept. of Theoretical Computer Science, ITU

Joint work with Chr. W. Mortensen and Mihai Patrascu

Outline

- Range reporting
- RAM model
- Van Emde Boas-like solution
- New data structure:
  - Solution using suboptimal space
  - Reducing space
- Open problems

Dynamic range reporting in 1-D

- Maintain a set S of points (numbers) along a line under insertion and deletion of points
- Answer *FindAny* queries: Given x,y return an element from $S \cap [x;y]$, or report than none exists.
- Once a point has been found, further points in [x;y] can be retrieved in constant time per point.

RAM model

- Models the capabilities of a real computer:
  - Numbers are really bit strings, and we can manipulate these bit strings, use them to address memory cells, etc.
  - Every step of a computation, and every memory access counts as 1 time unit.
- Contrast e.g. with the comparison model, where membership searches take $\Omega(\log n)$ time. $O(1)$ time solutions are known on a RAM.

Approach 1: Predecessor search

- Find predecessor of y in S.
- Elements of S in binary search tree:
  - O(log n) time for *FindAny(x,y)*
  - O(log n) time for updates.
- Optimal in comparison-model.

Can the features of the RAM model be used to improve on this?

van Emde Boas - basic idea

(1975)

- Consider integers as *bit strings* of length w.
- The integer $s \in S$ that has the *longest common prefix* with x is either the *predecessor* or *successor* of x.
- Search for length of lcp(x,s) by binary search in [0;w] - log(w) steps.
- Each prefix of a key in S is stored in a hash table. If there is a unique key x having prefix p, we associate x with p.

van Emde Boas - example

- Search for x=10001101:
  - Lookup(1000): Nonunique prefix.
  - Lookup(100011): Not a prefix.
  - Lookup(10001): Unique prefix of 10001010.
- Insert x=10001101:
  - Look up every prefix and change:
    Not a prefix → Unique prefix of x.
    Unique prefix → Nonunique prefix.

van Emde Boas - analysis

- Predecessor search: O(log w) time.
- Insertion: O(log w) time.
- Space: O(nw) words.

- Space saving trick (Willard 1983):
  - Use vEB structure only for every Θ(w)th element of S (in sorted order)
  - Associate with every element of vEB a search tree of Θ(w) elements from S.
  - Improves space to O(n) words.

Limits to predecessor search

- It is known that $\Omega(\log w / \log \log w)$ time is needed to answer precesessor queries, using polynomial space.
- But *FindAny*(x,y) is different from predecessor search:
  - We know both endpoints.
  - We are happy with *any* point in $S \cap [x;y]$.
- Useful fact: All points in $S \cap [x;y]$ will have lcp(x,y) as a prefix.

Approach 2: LCP search

Miltersen et al. (1995)

- Store every prefix p of some element in S in a hash table along with:
  - The largest element a in S with prefix p0.
  - The smallest element b in S with prefix p1.
- *FindAny*(x,y):
  - Look up lcp(x,y) and retrieve (if ∃) a and b.
  - If S∩[x;y] is nonempty, a or b is in [x;y].
- Constant time search!
- Space later improved to O(n) words.
  (Alstrup, Brodal, and Rauhe, 2001)

New result: Fast and dynamic

- *FindAny*(x,y):
  - Choose your own time bound t in the range O(1) to O(log log w).
  - Update time becomes $O(w^{-2^t}+\log w)$.
  - Space O(n).
- I will concentrate on the end of the trade-off with:
  - *FindAny* in time O(log log w), and
  - Updates in time O(log w)
  - ... and not go into details on space usage.

Tries

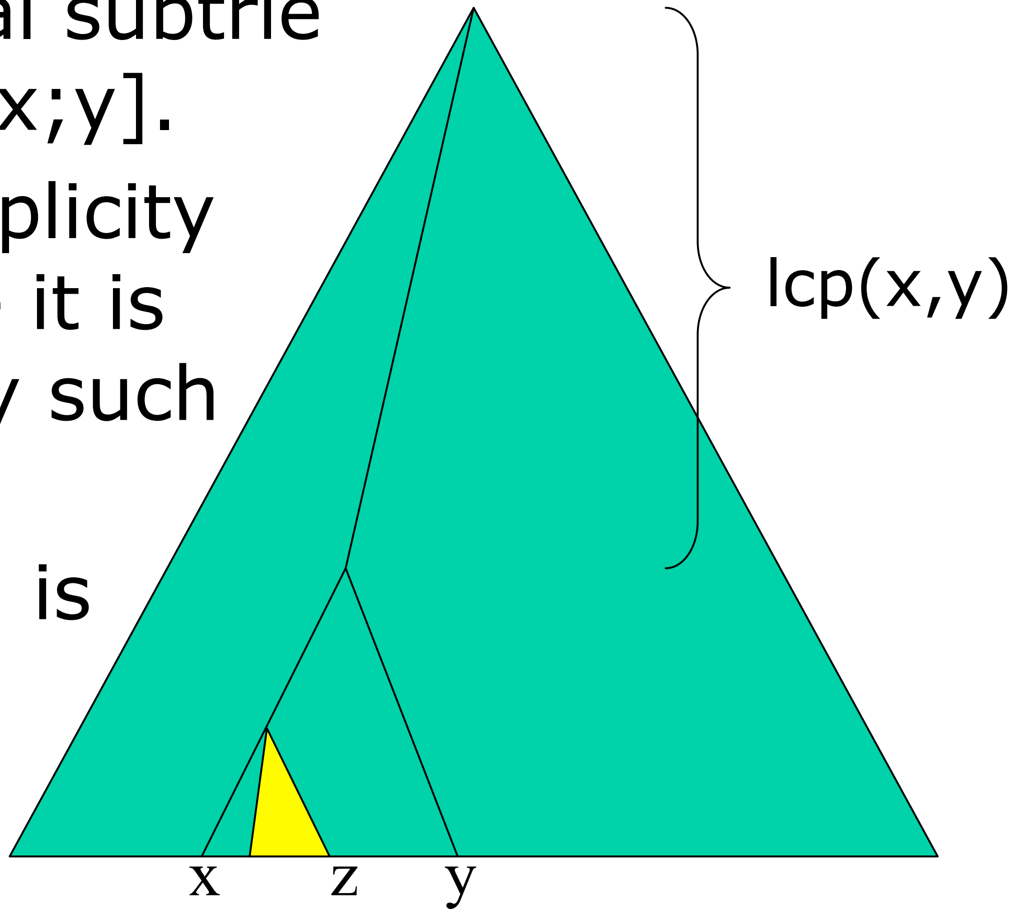
- A *trie* for a set of strings S is a tree with
  - labeled edges, where
  - the labels of the root-to-leaf paths form (by concatenation) the strings in S.
- We will consider:
  - The binary trie, where labels are in {0,1}, and more generally:
  - The trie of order t, with labels from $\{0,1\}^{2^t}$, for t=0,1,..,log w.
  - In the trie of order t we view elements of S as strings of length $w/2^t$.

Searching tries

- van Emde Boas search:
  - Look up node in trie of order log(w)-1,
  - look up node in trie of order log(w)-2,
  - ...
  - look up node in trie of order 0.
- Our search idea:
  - Do binary search on the tries to find the one ”suitable” for the search.
  - Number of steps becomes log log w.
  - Updates take constant time per trie.

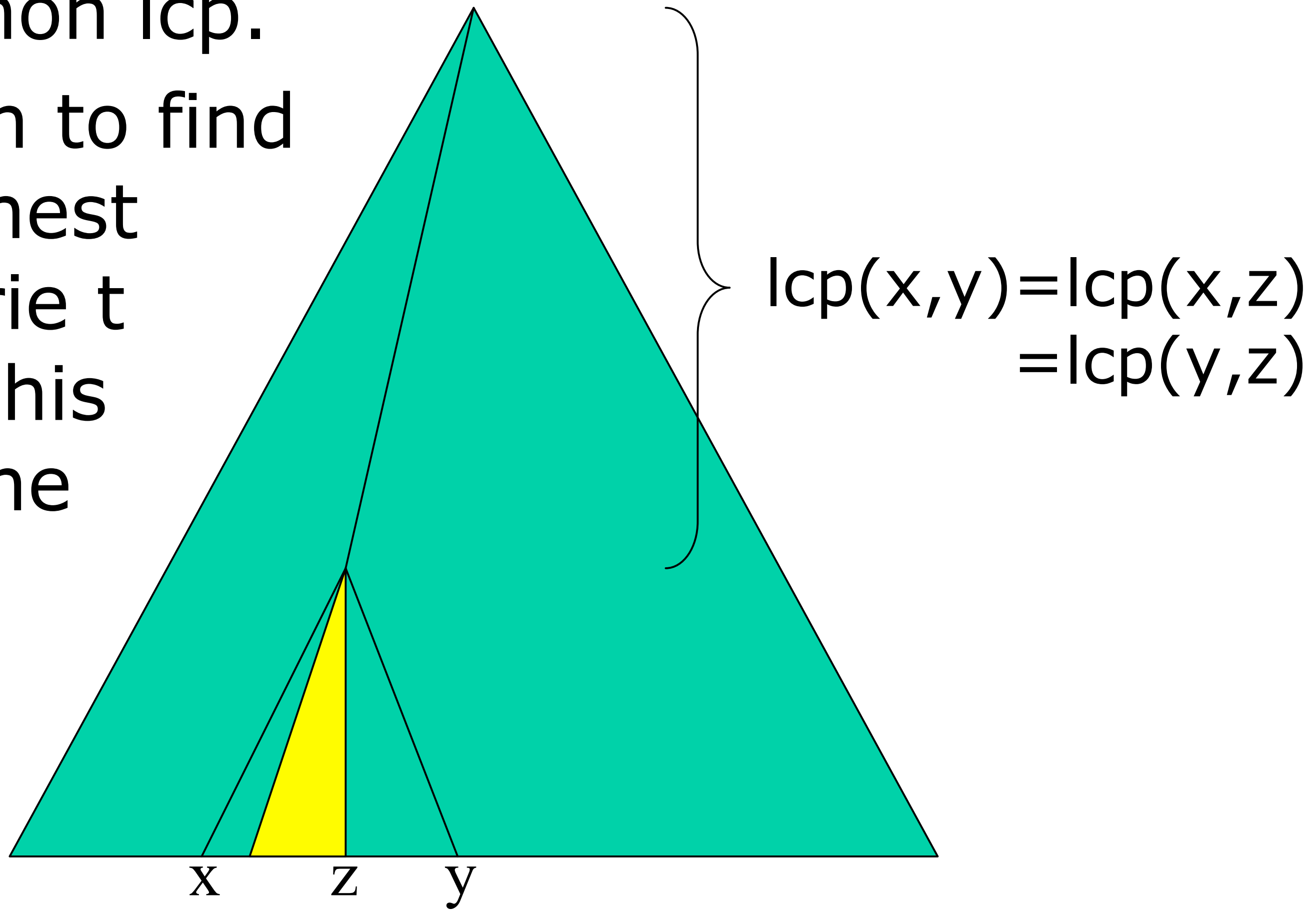
Example trie of order 0

- Assume z is an extreme element of a maximal subtrie inside [x;y].
- For simplicity assume it is the only such subtrie.
- lcp(x,y) is a prefix of z.



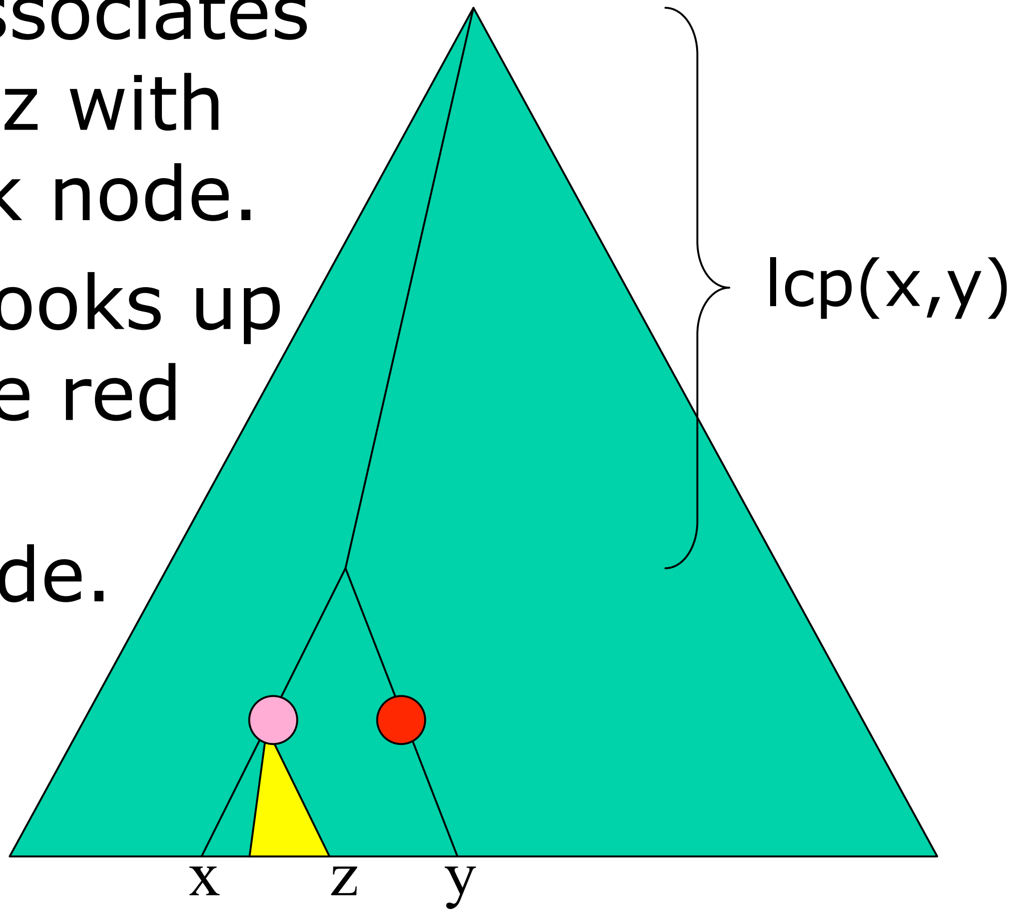
Example higher order trie

- In some higher order trie, {x,y,z} have a common lcp.
- We wish to find the highest order trie t where this is *not* the case.



Answering the query

- In example, the data structure for the trie t associates info on z with the pink node.
- Query looks up both the red and the pink node.



Finding the right trie

- Tries of order > t:
  - The node where x and y branch is also a branching node of that trie.
- Tries of order ≤ t:
  - The node where x and y branch is not a branching node of that trie.
- All tries store their branching nodes in a hash table (at most n per trie).

Dynamic updates - sketch

- For insertion of an element we:
  - Find its position in the 0th order trie, using vEB search, in O(log w) time.
  - Adjust at most one extreme point in each trie in O(1) time.
  - Create at most one new branching node in each trie in O(1) time.
- Deletions are symmetric to insertions.

Reducing the space

- **Ingredient 1:**
  ”Compressed pointers” of O(log w) bits enough to represent most nodes in the tries (Alstrup et al. ’01).
- **Ingredient 2:**
  Dynamic perfect hashing using less space than the set of keys hashed.

Conclusion and open questions

- Presented new dynamic range reporting data structure with very fast queries.
- Application: String prefix search
  - ”Find a string with prefix x”
- Are the bounds optimal?
- From a practical point of view, the query time is a small constant (log log w<4 in practical situations).
- Better than vEB and search trees in practice?