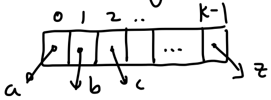
Tries and Digital Search Trees I

Tries: History

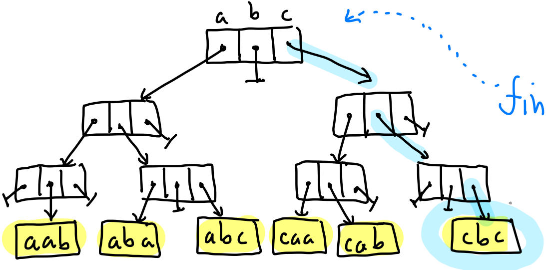
- de la Briandais (1959)
- Fredkin - "trie" from "retrieval"
- Pronounced like "try"

Node: Multiway of order k

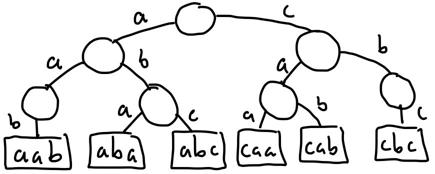


Example: $\Sigma = \{a=0, b=1, c=2\}$

Keys: {aab, aba, abc, caa, cab, cbc}

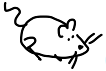


Same structure/Alt. Drawing



Digital Search:

- Keys are **strings** over some **alphabet** Σ
- Eg. $\Sigma = \{a, b, c, \dots\}$
  $\Sigma = \{0, 1\}$ Let $k = |\Sigma|$
- Assume chars coded as ints: $a=0, b=1, \dots z=k-1$



Analysis:

Search: ~ length of query string [O(1) time per node]

Space:

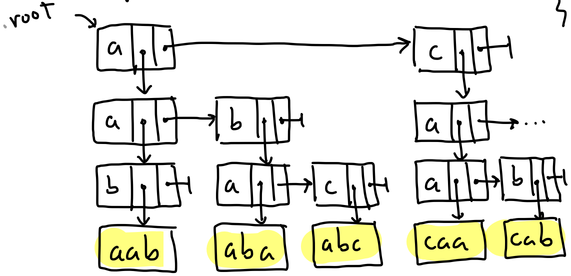
- No. of nodes ~ total no. of chars in all strings
- Space ~ k·(no. of nodes)

Large?

Analysis:

- **Space:** Smaller by factor k
- **Search Time:** Larger by factor of k

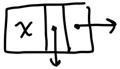
Example:



How to save space?

de la Briandais trees:

- Store 1 char. per node
-



≠ x ⇒ try next char in Σ

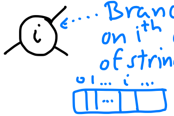
== x ⇒ advance to next character of search string

- First-child/next-sibling

Tries and Digital Search Trees II

Patricia Tries:

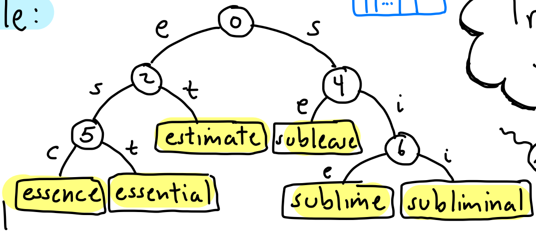
- Improves trie by compressing degenerate paths
- PATRICIA = Practical Alg. to Retrieve Info. Coded in Alpha...
- Late 1960's: Morrison + Gwehenberger
- Each node has index field, indicates which char to check next (Increase with depth)



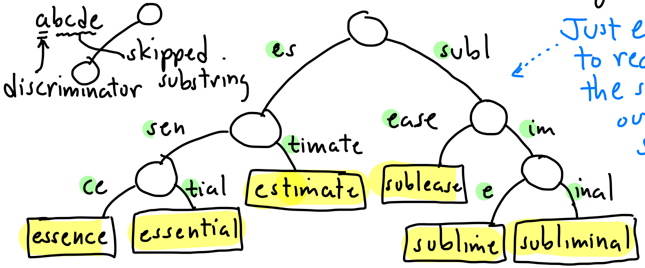
Branch based on ith char of string

Example:

0 1 2 3 ...
essence
essential
estimate
sublease
sublime
subliminal



Same data structure – Drawn differently

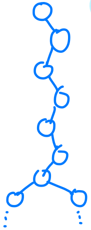


Just easier to read the strings out....

Same data struct.

Dealing with long Paths:

- To get both good space + query time efficiency, need to avoid long, degenerate paths.



- Path compression!

Example:

	ID
S_{10} : $	$
S_9 : a$	a$
S_8 : ma$	ma$
S_7 : ama$	ama$
S_6 : jama$	j

	ID
S_5 : ajam...	aj
S_4 : pajam...	paj
S_3 : apaja...	ap
S_2 : mapaj...	map
S_1 : amapaj...	amap
S_0 : pamapa...	pam

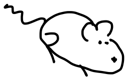
Example: S = pamapajama$

0 1 2 3 4 5 6 7 8 9 10

S_{10} - $
S_9 - a$
S_8 - ma$
S_7 - ama$
⋮

Def: Substring identifier for S_i is shortest prefix of S_i unique to this string

E.g. ID(S_1) = "amap"
ID(S_7) = "ama$"



Analysis:

- Query time: (Same as std trie) ~ search string length (may be less)
- Space:
  - No. nodes: ~ No. of strings (irresp. of length)
  - Total space: k·(No. of nodes) + (Storage for strings)

Suffix Trees:

- Given single large text S
- Substring queries: "How many occurrences of "tree" in CMSC 420 notes"

Notation: $S = a_0 a_1 a_2 \ldots a_{n-1}\$$

special terminal

- Suffix: $S_i = a_i a_{i+1} \ldots a_{n-1}\$$
- Q: What is minimum substring needed to identify suffix S_i?

Tries and Digital Search Trees III

Suffix Trees (cont.)

S - text string $|S| = n$

S_i = i^{th} suffix

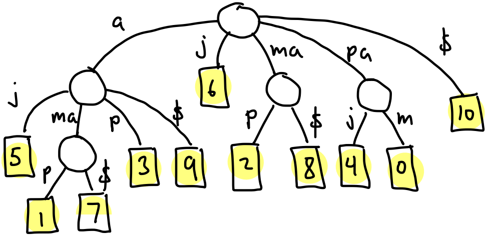
Substring ID = min substr. needed to identify S_i

A **suffix tree** is a Patricia trie of the n+1 substring identifiers

Example:

0 1 2 3 4 5 6 7 8 9 10

S = pamapajama$



E.g. $ID(S_1)$ = amap $ID(S_7)$ = ama$

Substring Queries:

How many occurrences of t in text?

- Search for target string t in trie
- if we end in internal node (or midway on edge) - return no. of extern. nodes in this subtree
- else (fall out at extern. node)
  - compare target with string
    - if matches - found 1 occurrence
    - else - no occurrences

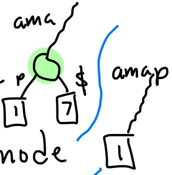
Example:

Search("ama") → End at intern node

Report: 2 occ's.

Search("amapaj") → End at extern node

Goto S_1 + verify



Analysis:

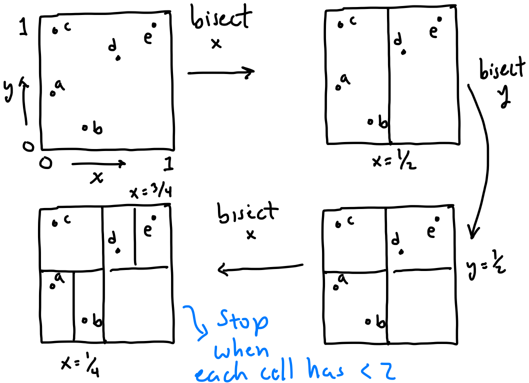
- **Space:** O(n) nodes
  O(n·k) total space
  ($k = |\Sigma| = O(1)$)
- **Search time:** ~ total length of target string
- **Construction time:**
  - O(n·k) [nontrivial]

PR k-d tree: Can be used for answering same queries as point kd-tree (orth. range, near. neigh)

Geometric Applications:

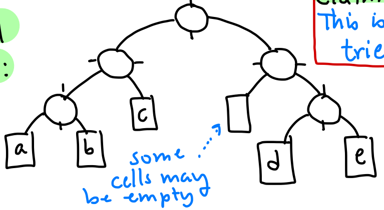
PR kd-Tree: kd-tree based on midpoint subdivision

Assume points lie in unit square



Stop when each cell has < 2 points

Final tree:



some cells may be empty

Claim:
This is a trie!

Tries and Digital Search Trees IV

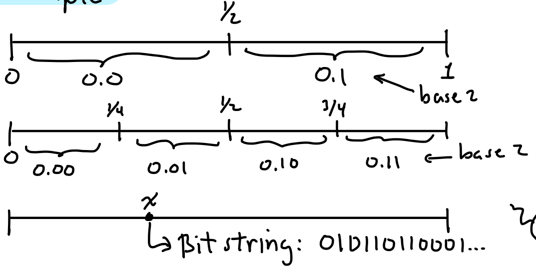
Binary Encoding:

- Assume our points are scaled to lie in unit square $0 \le x, y < 1$ (can always be done)
- Represent each coordinate as binary fraction:

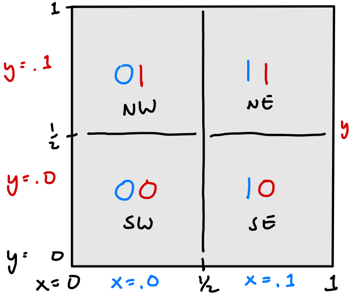
$$x = 0.a_1 a_2 a_3 \ldots \qquad a_i \in \{0,1\}$$

$$x = \sum a_i \cdot 1/2^i$$

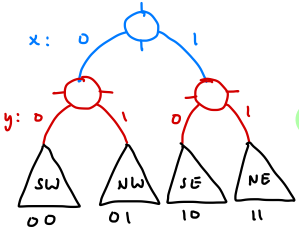
Example:



How do we extend to 2-D?



PR kd-tree



PR kd-Tree ≡ Trie ??

- Approach: Show how to map any point in $\mathbb{R}^2$ to bit string
- Store bit strings in a trie (alphabet $\Sigma = \{0,1\}$)
- Prove that this trie has same structure as kd-tree

Bit Interleaving:

Given a point $p = (x, y)$

$0 \le x, y < 1$

let: $x = 0.a_1 a_2 \ldots$ in binary

$y = 0.b_1 b_2 \ldots$

Define:

$$\phi(x,y) = a_1 b_1 a_2 b_2 a_3 b_3 \ldots$$

Called **Morton Code** of p

Further Remarks:

- Techniques for efficiently encoding, building, serializing, compressing... tries apply immediately to PR kd-tree
- Can generalize to any dimension

$$\left.\begin{array}{l} x = 0.a_1 a_2 \ldots \\ y = 0.b_1 b_2 \ldots \\ z = 0.c_1 c_2 \ldots \end{array}\right\} \phi = a_1 b_1 c_1 a_2 b_2 c_2 \ldots$$

Lemma: Given a pt set $P \subseteq \mathbb{R}^2$ (in unit square $[0,1)^2$) let $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$

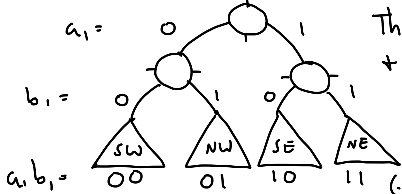
Let $\Phi(P) = \{\phi(p_1), \phi(p_2), \ldots, \phi(p_n)\}$ (n binary strings)

Then the PR kd-tree for P is equivalent to binary trie for $\Phi(P)$.

Proof: By induction on no. of bits

Let $x = 0.a_1 a_2 \ldots \quad y = 0.b_1 b_2 \ldots$

and consider just $\phi(x,y) = a_1 b_1 \ldots$



The PR kd-tree + binary trie assign pts to same subtrees.

(.... induction)