**Digital Search:**
- Keys are strings over some alphabet $\Sigma$
- Eg. $\Sigma = \{a, b, c, \ldots\}$
- $\Sigma' = \{0, 1\}$
- Assume chars coded as ints: $a = 0$, $b = 1$, $\ldots$, $z = k - 1$

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

**Example:**
- $\Sigma' = \{a = 0, b = 1, c = 2\}$
- Keys: $\{aab, aba, abc, caa, cab, cbc\}$

**Tries and Digital Search Trees I**

**Example:**
- Find("cbc")

**Analysis:**
- Search: $\sim$ length of query string $[O(1) \text{ time per node}]$

**Same structure/Alt. Drawing**
- No. of nodes $\sim$ total no. of characters in all strings
- Space $\sim k \cdot (\text{no. of nodes})$

**How to save space?**
- de la Briandais trees:
  - Store 1 char. per node
  - $x \rightarrow \neq \Rightarrow \text{try next char in } \Sigma'$
  - $x \Rightarrow \text{advance to next character of search string}$
  - First-child/next-sibling
Patricia Tries:
- Improves trie by compressing degenerate paths
- PATRICIA = Practical Alg. to Retrieve Info. Coded in Alpha...
- Late 1960’s: Morrison & Guehenberger
- Each node has index field, indicates which char to check next (Increase with depth)

Example:
- essence: essential, estimate sublease, sublime subliminal
  - Same data structure - Drawn differently

Dealing with long Paths:
- To get both good spaces & query time efficiency, need to avoid long, degenerate paths.
- Path compression!

Example:
- $S_0 = \text{aj}\
  - $S_1 = \text{paj}\
  - $S_2 = \text{apaj}\
  - $S_3 = \text{mapaj}\
  - $S_4 = \text{pamapa}\

Tries and Digital Search Trees II

Example: $S = \text{pamapajama}$
- Def: Substring identifier for $S_i$ is shortest prefix of $S_i$ unique to this string
- $S_i$ = mach
  - Eg. $\text{ID}(S_i) = \text{"ama"}$
  - $\text{ID}(S_i) = \text{"ama"}$

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

Notation: $S = a_0a_1a_2...a_{n-1}$
- Suffix: $S_i = a_ia_{i+1}...a_{n-1}$
- Q: What is minimum substring needed to identify suffix $S_i$?
**Example:** \( S = \text{pamapajama} \)

**Suffix Trees (cont.)**
- \( S \) - text string, \( |S| = n \)
- \( S_i^j \) - \( i \text{th} \) suffix
- Substring ID = \text{min substr. needed to identify } S_i^j

A suffix tree is a Patricia trie of the \( n+1 \) substring identifiers.

**Example:**
- E.g. \( \text{ID}(S_1) = \text{amap} \)
- \( \text{ID}(S_7) = \text{ama} \)

**Substrings Queries:**
- How many occurrences of \( t \) in \( S \)?
  - 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

**Suffix Trees**
- \( S \) - text string, \( |S| = n \)
- \( S_i^j \) - \( i \text{th} \) suffix
- Substring ID = \text{min substr. needed to identify } S_i^j

A suffix tree is a Patricia trie of the \( n+1 \) substring identifiers.

**Tries and Digital Search Trees III**

**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

**Analysis:**
- Space: \( O(n) \) nodes
- \( O(n \cdot k) \) total space
  - \( k = |S| = o(1) \)
- Search time: \( n \) total length of target string
- Construction time:
  - \( O(n \cdot k) \) [nontrivial]
  - \( O(n \cdot k) \) [nontrivial]

**Final tree:**
- **Claim:** This is a trie!
Binary Encoding:
- Assume our points are scaled to lie in unit square $0 \leq x, y < 1$ (can always be done).
- Represent each coordinate as binary fraction:
  $x = 0.a_1 a_2 a_3 \ldots, a_i \in \{0, 1\}$
  $y = \sum a_i \cdot \frac{1}{2^i}$

Example:

<table>
<thead>
<tr>
<th>$x$</th>
<th>0.0</th>
<th>0.1</th>
<th>$\frac{1}{2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.1</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.1</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>

How do we extend to 2-D?

**PR kd-Tree $\equiv$ Trie?**

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

**Tries and Digital Search Trees IV**

Further Remarks:
- Techniques for efficiently encoding, building, serializing, compressing...
- Can generalize to any dimension $x = 0.a_1 a_2 a_3 \ldots$
  $y = 0.b_1 b_2 b_3 \ldots$

**Lemma:** Given a point 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 $\Phi(P) = \{\phi(p_1), \phi(p_2), \ldots, \phi(p_n)\}$ (in binary strings).
Then the PR kd-tree for $P$ is equivalent to binary trie for $\Phi(P)$.

**Bit Interleaving:**

Given a point $p = (x, y)$ with $0 \leq x, y < 1$.
Let $x = 0.a_1 a_2 a_3 \ldots$ in binary.
$y = 0.b_1 b_2 b_3 \ldots$.
Define:
$\phi(x, y) = a_1, b_1 a_2 b_2 a_3 b_3 \ldots$.

**Proof:** By induction on no. of bits.
Let $x = 0.a_1 a_2 \ldots, y = 0.b_1 b_2 \ldots$.
and consider just $\phi(x, y) = a_1, b_1 a_2 b_2 a_3 b_3 \ldots$.

*The PR kd-tree + binary trie assign pts to same subtrees (... induction)