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

Dealing with long Paths:
- To get both good space and query time efficiency, need to avoid long degenerate paths
- Path compression

Example: $S_0: \text{ajam...}$
Example: $S_1: \text{apaj...}$
Example: $S_2: \text{apmaj...}$
Example: $S_3: \text{apmaj...}$
Example: $S_4: \text{pajam...}$
Example: $S_5: \text{pajam...}$

Tries and Digital Search Trees II

Branch based on the char of string

Example:
- essence
- estimate
- sublease
- sublime
- subliminal

Same data structure - Drawn differently

Just easier to read the strings out... same data structure

Analysis:
- Query time: (Same as std trie) $n$ search string length (may be less)
- Space:
  - No. nodes: $\approx$ No. of strings (irresp. of length)
  - Total space: $K \cdot (\text{No. of nodes}) + (\text{Storage for strings})$

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

Notation: $S = a_{-1}a_0a_1...a_{n-1}$
- Suffix: $S_i = a_1a_2...a_{n-1}$
- Q: What is minimum substring needed to identify suffix $S_i$?

Def: Substring identifier for $S_i$:
- $S_i$ is shortest prefix of $S_i$ unique to this string
- $S_i: \text{ama...}$
- $S_i: \text{ama...}$

Eq: $\text{ID}(S_i) = \text{ama...}$

Example: $S: \text{pamapajama...}$

Essential estimate sublease sublime subliminal

Essence

$S_0: \text{ajam...}$
$S_1: \text{apaj...}$
$S_2: \text{apmaj...}$
$S_3: \text{apmaj...}$
$S_4: \text{pajam...}$
$S_5: \text{pajam...}$
Example: $S = \text{puma pajama}$

Suffix Trees (cont.)
- $S$ - text string $|S| = n$
- $S_i$ - $i^{th}$ suffix
- $S_i'$ - $i^{th}$ suffix

Substring ID = min substr. needed to identify $S_i'$
A suffix tree is a Patricia trie of the $n+1$ substring identifiers

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

Example:
- Search("ama") → End at internal node
- Report: 2 occ.
  - Search("amapaj") → End at external node
- Go to $S_i$, + verify

Tries and Digital Search Trees III

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

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

Geometric Applications:
- PR k-d-Tree: k-d-tree based on midpoint subdivision
- Assume points lie in unit square

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:
  \(x = 0.a_1a_2a_3\ldots a_i, a_i \in \{0,1\}\)
  \(x = \sum a_i \cdot 2^{-i}\)

Example:
\[
\begin{array}{c|c}
0,0 & x \\
0,1 & y \\
1 & z \\
\end{array}
\]

Bit string: 01011011000...

0.3725 = 0.110110...

How do we extend to 2-D?

PR kd-tree

Tries and Digital Search Trees IV

Further Remarks:
- Techniques for efficiently encoding, building, serializing, compressing...
- Can generalize to any dimension:
  \(x = a_1a_2\ldots a_i\)
  \(y = b_1b_2\ldots b_i\)
  \(z = c_1c_2\ldots c_i\)

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 \(\Phi(E) = \{\phi(p_1), \phi(p_2), \ldots, \phi(p_n)\}\) (in binary strings)
Then the PR kd-tree for \(P\) is equivalent to binary trie for \(\Phi(E)\)

Proof:
By induction on no. of bits
Let \(x = 0.a_1a_2\ldots a_i, y = 0.b_1b_2\ldots b_i\)
and consider just \(\phi(x, y) = a_1b_1a_2b_2\ldots\)

Define:
\[\phi(x, y) = a_1b_1a_2b_2\ldots\]
Called Morton Code of \(p\)

Call the PR kd-tree + binary trie assign pts to same subtrees
(* induction)
Deallocation Models:

Explicit: (C, C++)
- programmer deletes
- may result in leaks, if not careful

Implicit: (Java, Python)
- runtime system deletes
- Garbage collection
- Slower runtime
- Better memory compaction

Explicit Allocation/Deallocation
- Heap memory is split into blocks whenever requests made
- Available blocks:
  - Merged when contiguous
  - stored in available block list

What happens when you do
- new (Java)
- malloc/free (C)
- new/delete (C++)

Runtime System Mem. Mgr.
- Stack - local vars, recursion
- Heap - for "new" objects
  - Don't confuse with heap data structure/heaps!

Block Structure:

Allocated:
- inUse
- prevInUse

Available:
- prevInUse

Guide:
- prevInUse: 1 if prev. contig. block is allocated
- prev/next: links in avail. list
- size/size2: total block size (includes headers)

How to select from available blocks?
- First-fit: Take first block from avail. list that is large enough
- Best-fit: Find closest fit from avail. list

Surprise:
- First-fit is usually better
  - faster + avoids small fragments

Memory Management

Fragmentation:
- Results from repeated allocation + deallocation
  (Swiss-cheese effect)

External: Caused by pattern of alloc/dealloc

Internal: Induced by mem. manage. policies (not user)

Printervene
Some C-style pointer notation

void* - pointer to generic word of memory
Let p be of type void*:
  p+10 - 10 words beyond p
  *(p+10) - contents of this
Let p point to head of block:
  p.inUse, p.prevInUse, p.size
  - we omit bit manipulation
  *(p+p.size-1) - references last word in this block

Example: Alloc b=59

Allocation:
  malloc(b)
  - Search avail. list for block of size $b' \geq b+1$
  - If $b'$ close to $b$: alloc entire block (unlink from avail list)
  - Else: split block

Deallocation:
  If prev+next contiguous blocks are allocated → add this to avail
  Else - merge with either/both to make max. avail block

Example:

```
(module)

allocate (int b) {
  b+1 // add +1 for header
  p = search avail list for block size \geq b
  if ( p == null ) Error - Out of mem!
  if ( p.size - b \leq TOO_SMALL )
    unlink p from avail list
    q = p
  else .... (continued)
}
```

Memory Management II
Announcements - 5/4

- HW 4: On the handouts page
  - Due Thu, May 11 at start of class

- New policy: Can drop lowest HW score

- Final Exam
  - Mon, May 15 4-6 pm
  - Room: IRB 0324

- Prog Assign 3: Deadline extended to May 17, 11:59 pm