Tries (radix searching)

Branching factor of trie equals the number of alternative symbols (radix of alphabet) that can occur in each position of the key.

e.g., FZR pg. 526: build a trie to store the keys: able, abrahams, adams, anderson, andrews, baird

This is an example of a radix-26 trie

What about a radix-2 trie? If we let choices be $0, 1$, we can make search decisions on a bit by bit basis.

How do we represent such a trie?

We could use a tree, but would suffer many seeks if the trie was too large to hold in memory.

solution: collapse the trie into a flat array of contiguous records forming a directory of hash addresses and pointers to corresponding buckets. First, we must turn the radix-2 trie into a complete binary tree:

How do we handle overflow?

goal of extendible hashing: increase address space in response to overflow rather than create long overflow sequences.

Example: Bucket **A** overflows. Since only 1 bit (of 2) are used to address bucket **A**, split it, keeping 00 in **A** and 01 in **D**, i.e., use the full 2 bits:

This growing and shrinking of the address space as needed satisfies our criteria for extendible hashing.

Do we need a hash function at all? Why not use the trie on the bits of the key rather than on the bits of $h(key)$?

Dynamic Hashing

2 hash functions:

1. original address space
2. one for descending the trie

Example:

If directory node, then stop and search bucket.
If internal node, then $h_2(key)$ and search trie using bits of $h_2(key)$. For example, could use $h_2(key)$ to provide a seed for a random number generator that will generate a bit sequence to search trie.

Differences between dynamic hashing and radix searching:

- dynamic hashing allows for slower growth. Why?
does not double the size of the address space.

Linear Hashing

- get rid of the directory
- use progressively more bits of address (like extendible hashing)
- **linear** extension of address space

Hash functions in linear hashing:

let: $h_d =$ hash function of current address depth
$h_{d+1} =$ hash function of expansion buckets
$p =$ ptr to next bucket to split

if $h_d(key) \geq p$ then
  addr $= h_d(key)$
else
  addr $= h_{d+1}(key)$