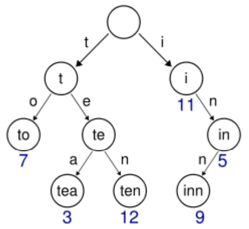
A Compact B-tree

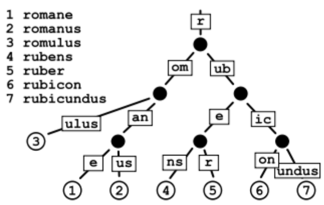
ACM SIGMOD, 2002
Peter Bumbulis, Ivan T.Bowman
Presented by Kyuseok Shim

Patricia Tree(trie)

- Specialized set data structure based on the trie that is used to store a set of strings
- In contrast with a regular trie, the edges of a patricia trie are labelled with sequences of characters rather than with single character



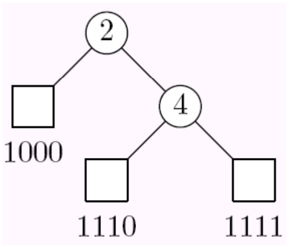
A trie for key {t,to,te,tea,ten,I,in,inn}



A patricia trie

Patricia Tree in Binary

- Leaf nodes contain keys
- Internal nodes contain offset, left-pointer and right-pointer
  - Offset : bit offset, all leaves descended from a node must agree on the first d-1 bits
  - Left-pointer : all leaves descended from the left child must have a '0' as the offset-th bit
  - Left-pointer : all leaves descended from the right child must have a '1' as the offset-th bit



A patricia trie for 1000,1110 and 1111

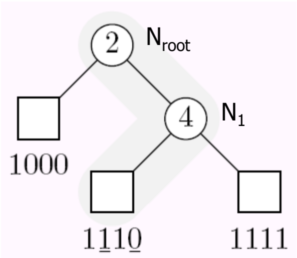
N(d,l,r)
d : offset
l,r : left, right child pointer

Observation

- The bit offsets in the nodes must increase as we follow a path from the root to a leaf
- The number of internal nodes in a patricia tree is always one less than the number of leaves
- A leaf **a** is to the left of a leaf **b** iff the key in the **a** is less than the key in **b**
- The offset of the first bit at which two keys differ is the bit offset found in their lowest common ancestor

Elementary Operations

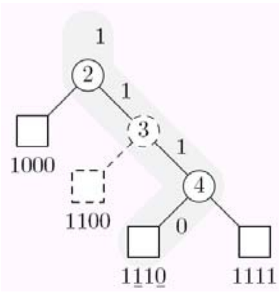
- Blind search
  - Given the key k_i stored in a leaf L doing a blind search from the root
  - For encountered node N(d,l,r) and search key k, if k[d] = 0, we continue on to left-child l, otherwise right-child r



Blind search for 1110
1. N_{root} : offset 2
   k's 2nd bit is 1, go right-child N_1
2. N_1 : offset 4
   k's 4th bit is 0, go left-child
3. Check the key stored in leaf node we found is the search key(1110 = 1110)

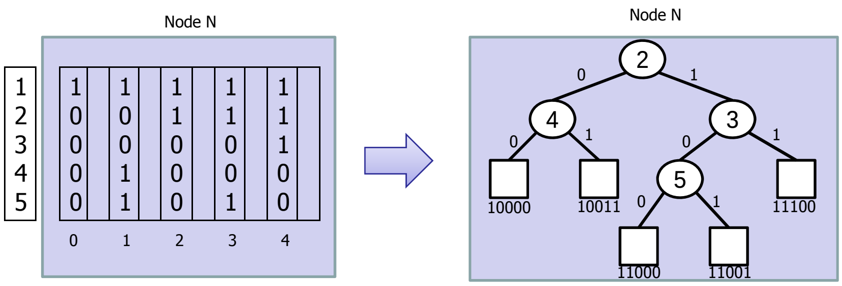
Elementary Operations

- Insert
  - To insert a value k, do a blind search and find a key k_j in a leaf node
  - K must differ from k_j at some bit position, let d be the first differing bit position
  - Encountered a node N(d',l',r') during the blind search with d'>d, then let the subtree s be the first such node, otherwise s be the leaf node found with blind search
  - If k[d] = 0, insert a new node N(d,L(k),s)
    If k[d] = 1, insert a new node N(d,s,L(k))



Compact B-Tree

- Basic structure : B-tree structure
- Patricia tree within each node



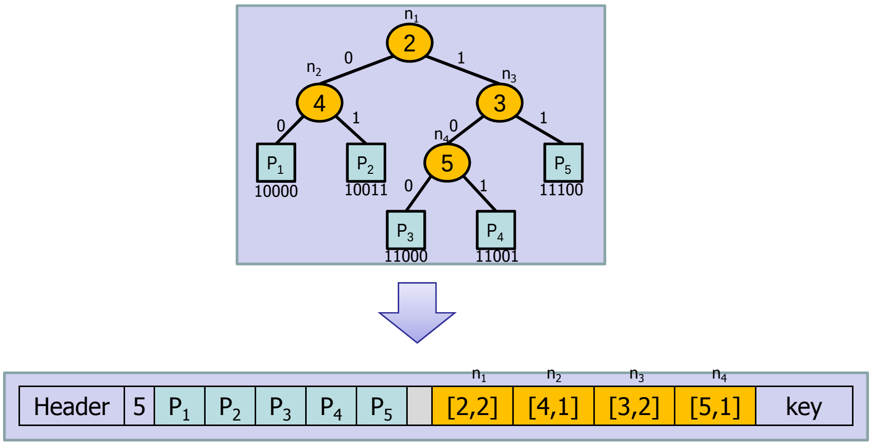
Node Structure in Compact B-Tree

- Header : node information
- N : total number of leave
- Pointers
- Node information (internal only)
  - D : bit offset
  - L : number of leaves in the left subtree
  - Pre-order traversal
- Key : value associated with the pointer to this page in the parent page

Header	N	$P_1 \ldots P_N$		$[D_1 L_1] \ldots [D_{N-1} L_{N-1}]$	key

Node Structure in Compact B-Tree

- Example



Node operations

- Blind search

```
B(K)
  i ← 1; j ← 1;  n ← N
  while n > 1 do
    if K[D_j] = 1 then
      i ← i + L_j; j ← j + L_j; n ← n - L_j
    else
      j ← j + 1; n ← L_j
    fi
  od
  return i
```

- i : index of leftmost leaf
  j : current visited node
  n : number of leaves it contains

Example of Blind Search

- Find key 11000

  $\mathcal{B}$(K)
    i ← 1; j ← 1; n ← N
    while n > 1 **do**
      if K[D_j] = 1 **then**
        i ← i + L_j; j ← j + L_j; n ←
      else
        j ← j + 1; n ← L_j
      fi
    od
    return i

K =11000

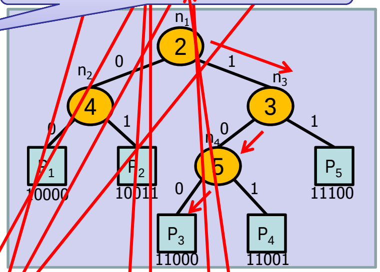
i = 3, j = 5, n = 1

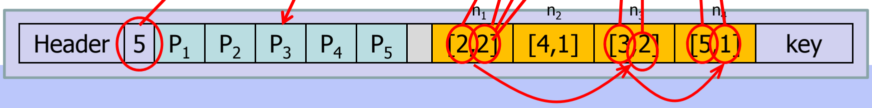
i = 3, j =4, n = 2, D_j = 5, K[D_j] = 0

i = 1+2 = 3, j = 1+2 = 3, n = 5-2 =3

i = 3, j = 4+1 = 5, n = 1

i = 3





Node operations

- Search

  $\mathcal{S}$(K)
    k ← $\mathcal{B}$(K)
    d,b ← $\mathcal{D}$(K,P_k)
    i ← 1; j ← 1; n ← N
    while n > 1 ∧ D_j ≤ d **do**
      if k ≥ L_j **then**
        k ← k – L_j; i ← i + L_j; j ← j + L_j; n ← n – L_j
      else
        j ← j + 1; n ← L_j
      fi
    od
    if b > 0 **then**
      return i + n
    else
      return i
    fi

- $\mathcal{D}$(K_1,K_2) : return offset d, and k_1[d]

Example of Search

- Find Key 10101

  $\mathcal{S}$(K)
  k ← $\mathcal{B}$(K)
  d,b ← $\mathcal{D}$(K,P_k)
  i ← 1; j ← 1; n ← N
  while n > 1 ∧ D_j ≤ d **do**
  if k ≥ L_j **then**
  k ← k − L_j; i ← i + L_j; j ← j + ...
  else
  j ← j + 1; n ← L_j
  fi
  od
  if b > 0 **then**
  return i + n
  else
  return i
  fi

K =11000, k = 1

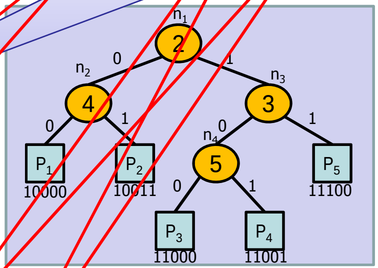
K =11000, k = 1, P_k = 10000, d = 3, b = 1

k = 1, d = 3, i = 1, j = 1, n = 5

k = 1, d = 3, i = 1, j = 2, n = 2, D_j = 4

k = 1, d = 3, i = 1, j = 1+1 = 2, n = 2, L_j = 1

k b = 1, i = 1, j = 2, n = 2
return 3



Header	5	P_1	P_2	P_3	P_4	P_5		[2,2]	[4,1]	[3,2]	[5,1]	key