B+ Tree and Hashing
• B+ Tree Properties
• B+ Tree Searching
• B+ Tree Insertion
• B+ Tree Deletion
• Static Hashing
• Extendable Hashing
• Questions in pass papers
– Balanced Tree
  • Same height for paths from root to leaf
  • Given a search-key K, nearly same access time for different K values

– B+ Tree is constructed by parameter \( n \)
  • Each Node (except root) has \( \lceil n/2 \rceil \) to \( n \) pointers
  • Each Node (except root) has \( \lceil n/2 \rceil - 1 \) to \( n-1 \) search-key values
• Search keys are sorted in order
  – $K_1 < K_2 < \ldots < K_{n-1}$

• Non-leaf Node
  – Each key-search values in subtree $S_i$ pointed by $P_i < K_i$, $\geq K_{i-1}$
    Key values in $S_1 < K_1$
    $K_1 \leq$ Key values in $S_2 < K_2$

• Leaf Node
  – $P_i$ points record or bucket with search key value $K_i$
  – $P_n$ points to the neighbor leaf node
• Given a search-value \( k \)
  
  – Start from the root, look for the largest search-key value \( (K_i) \) in the node \( \leq k \)
  
  – Follow pointer \( P_{i+1} \) to next level, until reach a leaf node

  \[
  \begin{array}{c|c|c|c}
  K_1 & K_2 & \ldots & K_i & K_{i+1} & K_{n-1} \\
  P_1 & P_2 & P_3 & P_{i+1} & P_{n-1} & P_n \\
  \end{array}
  \]

  – If \( k \) is found to be equal to \( K_i \) in the leaf, follow \( P_1 \) to search the record or bucket

  \[
  \begin{array}{c|c|c}
  K_i & K_{i+1} \\
  P_1 & k = K_i \\
  \end{array}
  \]
**Overflow**

- When number of search-key values exceed n-1

**Leaf Node**

- Split into two nodes:
  - 1\textsuperscript{st} node contains \(\lceil (n-1)/2 \rceil\) values
  - 2\textsuperscript{nd} node contains remaining values
  - Copy the smallest search-key value of the 2\textsuperscript{nd} node to parent node
• **Overflow**
  – When number of search-key values exceed \( n-1 \)

– **Non-Leaf Node**
  • Split into two nodes:
    – 1\(^{st}\) node contains \( \lceil n/2 \rceil - 1 \) values
    – Move the smallest of the remaining values, together with pointer, to the parent
    – 2\(^{nd}\) node contains the remaining values
Example 1: Construct a B⁺ tree for (1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42) with n=4.
• 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42
• 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42
Example 2: n=3, insert 4 into the following B+Tree
• **Underflow**
  
  - When number of search-key values < $\left\lceil \frac{n}{2} \right\rceil - 1$

- **Leaf Node**
  
  • Redistribute to sibling
    
    - Right node not less than left node
    
    - Replace the between-value in parent by their smallest value of the right node
  
  • Merge (contain too few entries)
    
    - Move all values, pointers to left node
    
    - Remove the between-value in parent

- **Delete 10**
  
  - Initial B+ Tree
  - Redistribute to sibling
  - Merge
  - Final B+ Tree
- Non-Leaf Node

- Redistribute to sibling
  - Through parent
  - Right node not less than left node
- Merge (contain too few entries)
  - Bring down parent
  - Move all values, pointers to left node
  - Delete the right node, and pointers in parent

Delete 10

```
9 10
```

```
13 18
```

```
9 10 14 15 16
```

```
14 18
```

```
9 13 15 16 18
```

```
14 18
```

```
9 13 18
```

```
14 22
```

```
9 13 14 16
```

```
18 22
```

```
9 13 14 16
```
• Example 3: n=3, delete 3
Example 4: Delete 28, 31, 21, 25, 19
• **Example 4: Delete 28, 31, 21, 25, 19**
A hash function $h$ maps a search-key value $K$ to an address of a bucket.

Commonly used hash function $\text{hash value mod } n_B$ where $n_B$ is the no. of buckets.

E.g. $h(\text{Brighton}) = (2+18+9+7+8+20+15+14) \mod 10 = 93 \mod 10 = 3$

No. of buckets = 10
- Hash function returns $b$ bits
- Only the prefix $i$ bits are used to hash the item
- There are $2^i$ entries in the bucket address table
- Let $i_j$ be the length of the common hash prefix for data bucket $j$, there is $2^{(i-i_j)}$ entries in bucket address table points to $j$
• Splitting (Case 1 $i_j=i$)
  – Only one entry in bucket address table points to data bucket $j$
  – $i++$; split data bucket $j$ to $j, z$; $i_j=i_z=i$; rehash all items previously in $j$;
Splitting (Case 2 $i_j < i$)
- More than one entry in bucket address table point to data bucket $j$
- Split data bucket $j$ to $j$, $z$; $i_j = i_z = i_j + 1$; Adjust the pointers previously point to $j$ to $j$ and $z$; rehash all items previously in $j$;
Example 5: Suppose the hash function is \( h(x) = x \mod 8 \) and each bucket can hold at most two records. Show the extendable hash structure after inserting 1, 4, 5, 7, 8, 2, 20.
inserting 1, 4, 5, 7, 8, 2, 20
Suppose the hash function $h(x) = x \mod 8$, each bucket can hold at most 2 records.

Show the structure after inserting “20”