Sets (II)

Hwansoo Han
Hashing

- List, array for directory
  - INSERT, DELETE, MEMBER require $O(N)$ for $N$ elements

- Hashing is widely used for directory
  - Operations require constant time on the average
  - In the worst case, could take $O(N)$
Open vs. Closed Hashing

- Open (or external) hashing
  - Use potentially unlimited space
  - No limit on the size of set

- Closed (or internal) hashing
  - Use a fixed space
  - Limit on the size of set
Open Hashing

- Main idea
  - Partition (possibly infinite) elements into a finite number of classes – e.g. $B$ classes numbered 0, 1, ..., $B-1$

- Hash function, $h(x)$
  - $x$ is called key
  - The value of $h(x)$ is called hash value
  - If $h(x) = i$, $x$ belongs to $i_{th}$ class

- Maintain $B$ buckets – 0, 1, ..., $B-1$
  - Hash value is one of the bucket numbers
  - An array, bucket table keeps the headers for $B$ lists
Open Hashing – data organization

The diagram illustrates the data organization in open hashing. Each bucket contains a list of elements, and the buckets are indexed from 0 to B-1. The diagram shows how elements are distributed across the buckets.
Open Hashing – data organization

- If buckets are roughly equal in size
  - The list for each bucket will be short
  - Average bucket will have $N/B$ members

- If we can estimate $N$, choose $B$ to be as large
  - Each bucket will have one or two members
  - Operations will take, on average, small constant time

- Hash function $h$ should distribute fairly evenly among the buckets – i.e. random value independent of $x$
Open Hashing – example $h(x)$

- Hash function for string
  - Add the encoding of each character – $\text{ord}(c)$

```plaintext
function h ( x: nametype ) : 0..B – 1 ;
    var
        i, sum: integer
    begin
        sum: = 0;
        for i: 1 to 10 do
            sum: = sum + ord(x[i]);
        h : = sum  mod  B
    end { h }
```
Open Hashing - operations

- **MEMBER** \((x, A)\) – \(x\): element, \(A\): bucket table
  - Search the bucket list headed by \(A[h(x)]\)

- **INSERT** \((x, A)\)
  - If not MEMBER \((x, A)\), insert \(x\) to the bucket list headed by \(A[h(x)]\)

- **DELETE** \((x, A)\)
  - Search the bucket list headed by \(A[h(x)]\)
  - If found, delete \(x\) from the list
Closed Hashing

- Keep the elements in the bucket table itself
  - No use of separate lists as in open hashing

- Only one element can be placed in any bucket
  - What if two elements, \( x \) and \( y \), are needed to be placed in the same bucket – i.e. \( h(x) = h(y) \)
  - Use rehash strategy
    - Find an alternative place for one element, if they have conflicting hash values by using a sequence of rehash functions, \( h_1(x), h_2(x), ... \) in order
Closed Hashing - example

- Suppose $B = 8$ and hash values for keys, $a, b, c, d$
  - $h(a) = 3$, $h(b) = 0$, $h(c) = 4$, $h(d) = 3$

- Rehash strategy
  - Linear hashing
  - $h_i(x) = (h(x) + i) \mod B$
Closed Hashing - operations

- **MEMBER** \((x, A)\) – \(x\): element, \(A\): bucket table
  - Examine \(h(x), h_1(x), h_2(x), \ldots\) until an empty bucket found
  - If found before an empty bucket, \(x\) belongs to \(A\)
  - If found empty bucket, \(x\) does not belong to \(A\)

- If deletions are performed, empty bucket does not guarantee \(x\) is not in somewhere else
  - When delete, insert a constant called `deleted` into a bucket
  - Need to distinguish between `deleted` and `empty`
  - `Deleted` can be treated as available space when inserting
Priority Queues

- An ADT based on set with operations
  - INSERT
  -DELETEMIN
  -MAKENULL

- Priority among elements
  - $p(a)$ for each element $a$, produces a real number, more generally a member of some linearly ordered set
  - DELETEMIN returns some element of smallest priority
Priority Queue Implementation

- Can use implementations for set
  - Linked list (sorted or unsorted)
    - Sorted list makes easy to find minimum priority element
    - Unsorted list makes easy to insert
    - One of INSERT and DELETEMIN will take $O(N)$
  - Bit-vector
    - Sequential search of bit vectors to find minimum

- But hash table cannot be used
  - No convenient way to find minimum
Priority Queue Implementation

- Partially ordered trees for priority queues
  - INSERT and DELETEMIN in $O(\log n)$ steps

- Main idea of partially ordered trees
  - Organize elements of the priority queue in a binary tree
  - Make the binary tree as balanced as possible
  - Make the priority of node $v$ in no greater than the priorities of $v$’s children
Partially Ordered Tree - example

- Leaves at the lowest level are filled from the left
Partially Ordered Tree Operations

- **DELETEMIN**
  - Root has the element with minimum priority
  - Thus, remove the root and take the rightmost element at the lowest level and temporarily put it at the root
  - Push down this element as far down as it will go, by exchanging it with the one of its children having the smaller priority – *percolation* step
  - The element goes down either to a leaf or to a node where its priority is no larger than either of its children
Partially Ordered Trees - DELETEMIN

- DELETEMIN (P)
Partially Ordered Trees - DELETEMIN
**Partially Ordered Trees - INSERT**

- **INSERT**
  - Place the element to insert as far left as possible at the lowest level – if the current level is all filled, start a new level.
  - If the new element has priority lower than its parent, exchange it with its parent.
  - Repeat the exchange, until it reaches either at the root or at the position where its priority is larger than its parent.
Partially Ordered Trees - INSERT

- INSERT (4, P)

```
3
  
5
  
6 8
  
10 18 9
```

```
3
  
5
  
6 8
  
10 18 9 4
```
Partially Ordered Trees - INSERT
Partially Ordered Trees - Array Implementation

- Heap
  - Array representation for binary trees, balanced as much as possible, and have leaves at the lowest level pushed to the left
  - For \( n \) nodes, use the first \( n \) elements of array \( A[1..N] \)
  - \( A[1] \) holds the root
  - For a node \( A[i] \), left child, if exists, is at \( A[2i] \) and right child, if exists, is at \( A[2i+1] \)
  - For a node \( A[i] \), its parent is at \( A[i \div 2] \) \( (i > 1) \)
Partially Ordered Trees in Heap


$$A[1..n] = 3, 5, 9, 6, 8, 9, 10, 10, 18, 9$$