Data Structures

How to Represent Sets?

- Finite dynamic sets, to be more precise.
- Operations on these sets, such as search, insert, delete, minimum, maximum, successor, predecessor.
- If only insert, delete, and test membership, then such a dynamic set is called a *dictionary*.
- Best way to implement a set depends on the needed operations.

Examples of Dynamic Sets

- Heaps.
- Stacks, queues, linked lists.
- Hash tables.
- Binary search trees.
- Red-black trees (a particular binary search tree that is balanced).
- In general they use pointers.

Stacks and Queues

- Specify which element the **Delete** operation removes:
  - stacks = LIFO (last-in, first-out)
  - queues = FIFO (fist-in, first-out)
- **Insert** called **push** or **enqueue**.
- **Delete** called **pop** or **dequeue**.
- Can be implemented with an array.
- Operations in O(1).

Stack Operations

- **Stack_empty(S):** // test emptiness
  return top(S) == 0 // index of last element

- **Push(S,x):**
  top(S)++
  S[top(S)]:=x

- **Pop(S):**
  if Stack_empty(S) **then error** "underflow"
  else
      top(S)--
      return S[top(S)+1]

Queue Operations

- **Enqueue(Q,x):**

  Q[tail(Q)]:=x

  if tail(Q) == length(Q) **then** tail(Q):=1

  else tail(Q)++

- **Dequeue(Q):**

  x:=Q[head(Q)]

  if head(Q) == length(Q) **then** head(Q):=1

  else head(Q)++

Linked Lists

- Linear structure, order given by pointers.
- Singly linked & doubly linked lists.
- List:
  - head (+ tail)
  - elements of the list (key + next + previous)
- **List_search(L,k):** *// O(n)*

x:=head(L)

while x != NIL and key(x) != k **do** x:=next(x)

return x

Linked Lists

- **List_insert(L,x):**

  next(x):=head(L)

  if head(L)!=NIL **then** prev(head(L)):=x

  head(L):=x

  prev(x):=NIL

- **List_delete(L,x):**

  if prev(x)!=NIL then next(prev(x)):=next(x)

  else head(L):=next(x)

  if next(x)!=NIL then prev(next(x)):=prev(x)

- Running time in *O(1)*.

Linked Lists with Sentinels

- Sentinel: special element to avoid tests.
  - next(nil)=head(L), prev(nil)=tail(L)
  - empty list: next(nil)=prev(nil)=nil
- **List_delete(L,x):**

  next(prev(x)):=next(x)

  prev(next(x)):=prev(x)
- **List_search(L,x):**

  x:=next(nil(L))

  while x!=nil(L) and key(x)!=k **do** x:=next(x)

  return x // can be nil element (sentinel)

Linked Lists with Sentinels

- **List_insert(L,x):**

  next(x):=next(nil(L))

  prev(next(nil(L))):=x

  next(nil(L)):=x

  prev(x):=nil(L)

- Note:
  - *O(1)* gain in speed, may be useful in tight loops
  - sentinels consume memory, bad if many small lists

Coding with Arrays

- If you have no pointers, it is possible to use arrays and indices.
- Memory management:
  - one list of *used* element,
  - one list of *free* element.

Rooted Trees

- Trees represented by linked data structures.
- Binary trees.
- Trees with unbounded branching.
- Best representation depends on the application.