CSC 212
Data Structures and Abstractions
Fall 2015

Lecture 03: Linked Lists
How to Shrink arrays?

halve the array when array is one-half full?

or

halve the array when array is one-quarter full?
int a[] = {3, 4, 7, 1, 6, 9};
int a[] = {3, 4, 7, 1, 6, 9};
int a[] = \{3, 4, 7, 1, 6, 9\};

Cost of inserting a new element at the beginning?
Linked Lists

Collection of sequential data stored at non-contiguous nodes
Linked Lists

Collection of sequential data stored at non-contiguous nodes.

Nodes are connected by links:
- Every node has a pointer to the next node.
Linked Lists

Collection of sequential data stored at non-contiguous nodes

Nodes are connected by links
every node has a pointer to the next node

Can grow and shrink dynamically
Singly Linked List

5 -> 7 -> 2 -> 6

NULL pointer
class List {

private:
    Node *head;
    Node *tail;

public:
    List();
    ~List();

    void insert_end(int);
};
class Node {

    private:
        int data;
        Node *nxt;

    public:
        Node(int d);
        ~Node();

        friend class List;

};
Basic Methods

**InsertKey**
- front, end, by index, by value

**DeleteKey**
- front, end, by index, by value

**GetKey/Search**
- by value, by index

**Traverse**
- visit all nodes (e.g. print)

**Destroy**

When implementing methods for linked lists, always draw the state of the linked list.
Insert at End

\[
p = \text{create\_node}(\text{key})
\]

\[
\text{if (head == NULL)}
\quad \text{head} = \text{tail} = p
\]

\[
\text{else}
\quad \text{tail}\rightarrow\text{nxt} = p
\quad \text{tail} = p
\]
p = create_node(key)

if (head == NULL)
    head = tail = p
else
    tail->nxt = p
    tail = p

O(1)
void List::insert_end(int d) {
    if (head) {
        tail->nxt = new Node(d);
        tail = tail->nxt;
    }
    else {
        head = tail = new Node(d);
    }
}
Insert at Front

\[ p = \text{create\_node} \]

\[
\text{if} \ (\text{head} == \text{NULL}) \\
\quad \text{head} = \text{tail} = p \\
\text{else} \\
\quad p->\text{nxt} = \text{head} \\
\quad \text{head} = p
\]
p = create_node

if (head == NULL)
    head = tail = p
else
    p->nxt = head
    head = p

O(1)
// inserting key at position k
// starting at 0

if size < k
    throw exception
else if k == 0
    call insert at front
else if k == size
    call insert at end
else
    q = head
    for i=0, i<k-1, i++
        q = q->nxt
    p = create_node(key)
    p->nxt = q->nxt
    q->nxt = p
// inserting key at position k
// starting at 0
if size < k
  throw exception
else if k == 0
  call insert at front
else if k == size
  call insert at end
else
  q = head
  for i=0, i<k-1, i++
    q = q->nxt
  p = create_node(key)
  p->nxt = q->nxt
  q->nxt = p
Insert by Value

Useful when making ordered insertions

pseudo code?
Insert by Value

Useful when making ordered insertions

pseudo code?  \( O(n) \)