CS301

Data Structure and Algorithms

Lecture 6: Linked list - delete node and copy list

Pandav Patel
Assistant Professor

Computer Engineering Department
Dharmsinh Desai University
Nadiad, Gujarat, India

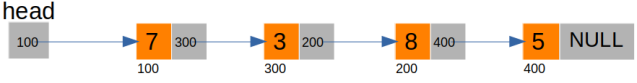
Objective

- Learn to delete a node from the linked list
- Learn to make a copy of the list

Overview

Delete a node whose address is given by X

- cases
  - Empty list
  - Node with address X is NOT present in the non-empty list
  - Node with address X is present in the list
    - Node to be deleted is the first node
    - There is only one node in the list and that is being deleted
    - Node to be deleted is the last node
    - Node to be deleted is some node in the middle
- Think of deleting Nodes with addresses 200, 400, 100, 300 in this sequence - MIDDLE, LAST, FIRST, and ONLY node scenarios



Delete a node whose address is given by X (cont...)

- steps
  1. Check if list is empty
  2. Search for a node with address X and keep track of predecessor while searching
  3. If node is not found then return with appropriate message
  4. Delete a node with address X

Algorithm: DELETE_BY_ADDRESS(X, HEAD)
Delete a node whose address is X
HEAD: Pointer to the first node of the linked list
CURRENT: Temporary node pointer for traversal
PRED: Temporary node pointer for predecessor
Algorithm should be called as
HEAD ←DELETE_BY_ADDRESS(X, HEAD)

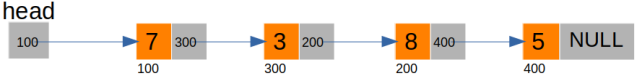
1. [Is list empty?]
   If HEAD = NULL then
      Write("List is empty")
      return(HEAD)
2. [Search for a node with address X]
   CURRENT ←HEAD
   While CURRENT ≠ X and
   LINK(CURRENT) ≠ NULL do
      PRED ←CURRENT
      CURRENT ←LINK(CURRENT)
3. [Return if node with address X is not found]
   If CURRENT ≠ X then
      Write("Node with address X not found")
      return(HEAD)
4. [Delete a node with address X]
   If CURRENT = HEAD then // If its first node
      HEAD ←LINK(CURRENT)
   else
      LINK(PRED) ←LINK(CURRENT)
   FREE(CURRENT) // Free the deleted node
   return(HEAD)

Delete a node whose address is given by X (cont...)

- Check if step 4 works fine for all cases when a node with address X is present
  - X is the address of the FIRST node
  - X is the address of the ONLY node (list with single node)
  - X is the address of the LAST node
  - X is the address of the MIDDLE node
- What is condition in step 3 was
  - If LINK(CURRENT) = NULL do
    - Will it work fine if node to be deleted is the last node?

Delete a node whose INFO is set to X

- cases
  - Empty list
  - Node whose INFO is set to X is NOT present in the non-empty list
  - Node whose INFO is set to X is present in the list
    - Node to be deleted is the first node
    - There is only one node in the list and that is being deleted
    - Node to be deleted is the last node
    - Node to be deleted is some node in the middle
- Think of deleting Nodes with INFO 8, 5, 7, 3 in this sequence - MIDDLE, LAST, FIRST, and ONLY node scenarios



Delete a node whose INFO is set to X (cont...)

- steps
  1. Check if list is empty
  2. Search for a node whose INFO is set to X and keep track of predecessor while searching
  3. If node is not found then return with appropriate message
  4. Delete a node whose INFO is set to X

Algorithm: DELETE_BY_INFO(X, HEAD)
Delete a node whose INFO is set to X
HEAD: Pointer to the first node of the linked list
CURRENT: Temporary node pointer for traversal
PRED: Temporary node pointer for predecessor
Algorithm should be called as
HEAD ←DELETE_BY_INFO(X, HEAD)

1. [Is list empty?]
   If HEAD = NULL then
      Write("List is empty")
      return(HEAD)
2. [Search for a node whose INFO is set to X]
   CURRENT ←HEAD
   While INFO(CURRENT) ≠ X and
   LINK(CURRENT) ≠ NULL do
      PRED ←CURRENT
      CURRENT ←LINK(CURRENT)
3. [Return if node with X as INFO is not found]
   If INFO(CURRENT) ≠ X then
      Write("Node with X as INFO not found")
      return(HEAD)
4. [Delete a node whose INFO is set to X]
   If CURRENT = HEAD then // If its first node
      HEAD ←LINK(CURRENT)
   else
      LINK(PRED) ←LINK(CURRENT)
   FREE(CURRENT) // Free the deleted node
   return(HEAD)

Delete a node whose INFO is set to X (cont...)

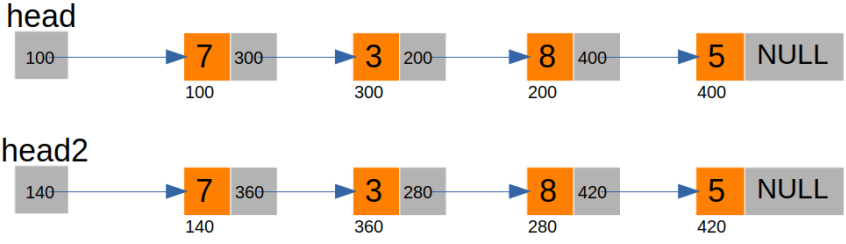
- Check if step 4 works fine for all cases when a node whose INFO is set to X is present and is the
  - FIRST node
  - ONLY node (list with single node)
  - LAST node
  - MIDDLE node
- What is condition in step 3 was
  - If LINK(CURRENT) = NULL do
    - Will it work fine if node to be deleted is the last node?

Try it yourself

- Write an algorithm to delete
  - first node in the list
  - last node in the list

Copy entire linked list

- cases
  - Empty list
  - List with only one node
  - List with more than one node



Copy entire linked list (cont...)

- steps
  1. return NULL if list is empty
  2. Make clone of the first node and set address of the clone as head of the new copy of the linked list
  3. Traverse original list (second node onwards) and make clones of each node and attach these clones to the last node in the new copy of the linked list

Algorithm: COPY(HEAD)
Copy a list whose first node is pointed by HEAD
HEAD2: Pointer to first node of new copy of list
CURRENT: Temporary node pointer for traversal
LAST: Pointer tracking last node in copy of list
Algorithm should be called as
HEAD_OF_COPY ←COPY(HEAD)

1. [Is list empty?]
   If HEAD = NULL then
      Write("List is empty")
      return(HEAD)
2. [Copy first node]
   HEAD2 ←Create a new node
   If HEAD2 = NULL then
      Write("New node not created")
      return(NULL)
   else
      INFO(HEAD2) ←INFO(HEAD)
      LINK(HEAD2) ←NULL
3. [Copy rest of the nodes]
   LAST ←HEAD2
   CURRENT ←LINK(HEAD)
   While CURRENT ≠ NULL do
      LINK(LAST) ←Create new node
      If LINK(LAST) = NULL then
         Write("New node not created")
         return(NULL)
      LAST ←LINK(LAST)
      INFO(LAST) ←INFO(CURRENT)
      LINK(LAST) ←NULL
      CURRENT ←LINK(CURRENT)
   return

Copy entire linked list (cont...)

- If calling algorithm gets NULL in return, what can be the possible reasons for it?
  - Original list was empty
  - Problem while copying list
- Is there memory leak?
  - How can it be fixed?
    - First, try to fix it on your own
    - If you can not fix it then check NEXT slide for solution
- Check if first and last node are being copied correctly
- Check if it will work for a list with single node

Algorithm: COPY(HEAD)
Copy a list whose first node is pointed by HEAD
HEAD2: Pointer to first node of new copy of list
CURRENT: Temporary node pointer for traversal
LAST: Pointer tracking last node in copy of list
TEMP: Temporary pointer to the node
Algorithm should be called as
HEAD_OF_COPY ←COPY(HEAD)

```
1. [ Is list empty? ]
     If HEAD = NULL then
          Write("List is empty")
          return(HEAD)
2. [ Copy first node ]
     HEAD2 ←Create a new node
     If HEAD2 = NULL then
          Write("New node not created")
          return(NULL)
     else
          INFO(HEAD2) ←INFO(HEAD)
          LINK(HEAD2) ←NULL
3. [ Copy rest of the nodes ]
     LAST ←HEAD2
     CURRENT ←LINK(HEAD)
     While CURRENT ≠ NULL do
          LINK(LAST) ←Create new node
          If LINK(LAST) = NULL then
               Write("New node not created")
               While HEAD2 ≠ NULL do
                    TEMP ←HEAD2
                    HEAD2 ←LINK(HEAD2)
                    FREE(TEMP)
               return(NULL)
          LAST ←LINK(LAST)
          INFO(LAST) ←INFO(CURRENT)
          LINK(LAST) ←NULL
          CURRENT ←LINK(CURRENT)
     return
```

Try it yourself

- Write an algorithm to
  - copy only FIRST 10 nodes from the linked list
    - If list has less than or equal to 10 nodes then copy all the nodes
  - copy only LAST 10 nodes from the linked list
    - Do not copy anything if list has less than 10 nodes
  - HINT: You can use For loop in above algorithms

Practice problems (From: Tremblay and Sorenson)

- Write an algorithm to
  - find number of nodes in the linked list
  - change INFO of k^{th} node in the linked list to value given by Y
  - insert value (INFO) given by Y to the immediate left of the k^{th} node in the linked list
  - append one linked list to the end of the another linked list
  - split linked list into two (Address of the node from where to split will be given)
  - delete all nodes with value (INFO) greater than X and less than Y in an ORDERED linked list
  - delete all nodes with value (INFO) greater than X and less than Y in an UNORDERED linked list

Think

- Can there exist two list with different heads which end up sharing nodes? (Start with different node(s) and end with same node(s))
- Can there exist two list with same head which end with different nodes? (Start with same node(s) and end with different node(s))
- How to delete a node whose predecessor's address is X. If first node is to be deleted, X would be NULL.
  - Can you write an algorithm for this?
    - Will it be easier or harder than deleting a node with address X? Why?