Remove duplicates from an unsorted linked list

Write a removeDuplicates() function which takes a list and deletes any duplicate nodes from the list. The list is not sorted.

For example if the linked list is 12->11->12->21->41->43->21 then removeDuplicates() should convert the list to 12->11->21->41->43.

METHOD 1 (Using two loops)
This is the simple way where two loops are used. Outer loop is used to pick the elements one by one and inner loop compares the picked element with rest of the elements.

Thanks to Gaurav Saxena for his help in writing this code.

```
/* Program to remove duplicates in an unsorted array */

#include<stdio.h>
#include<stdlib.h>

/* A linked list node */
struct node
{
 int data;
 struct node *next;
};

/* Function to remove duplicates from a unsorted linked list */
void removeDuplicates(struct node *start)
{
  struct node *ptr1, *ptr2, *dup;
  ptr1 = start;

  /* Pick elements one by one */
```


Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
  while(ptr1 != NULL && ptr1->next != NULL)
  {
     ptr2 = ptr1;

     /* Compare the picked element with rest of the elements */
     while(ptr2->next != NULL)
     {
       /* If duplicate then delete it */
       if(ptr1->data == ptr2->next->data)
       {
          /* sequence of steps is important here */
          dup = ptr2->next;
          ptr2->next = ptr2->next->next;
          free(dup);
       }
       else /* This is tricky */
       {
          ptr2 = ptr2->next;
       }
     }
     ptr1 = ptr1->next;
  }
}

/* UTILITY FUNCTIONS */
/* Function to push a node */
void push(struct node** head_ref, int new_data);

/* Function to print nodes in a given linked list */
void printList(struct node *node);

/* Druver program to test above function */
int main()
{
  struct node *start = NULL;

  /* The constructed linked list is:
   10->12->11->11->12->11->10*/
  push(&start, 10);
  push(&start, 11);
  push(&start, 12);
  push(&start, 11);
  push(&start, 11);
  push(&start, 12);
  push(&start, 10);

  printf("\n Linked list before removing duplicates ");
```

Geometric Algorithms

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
  printList(start);

  removeDuplicates(start);

  printf("\n Linked list after removing duplicates ");
  printList(start);

  getchar();
}

/* Function to push a node */
void push(struct node** head_ref, int new_data)
{
  /* allocate node */
  struct node* new_node =
            (struct node*) malloc(sizeof(struct node));

  /* put in the data  */
  new_node->data  = new_data;

  /* link the old list off the new node */
  new_node->next = (*head_ref);

  /* move the head to point to the new node */
  (*head_ref)    = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct node *node)
{
  while(node != NULL)
  {
    printf("%d ", node->data);
    node = node->next;
  }
}
```

Time Complexity: O(n^2)

METHOD 2 (Use Sorting)

In general, Merge Sort is the best suited sorting algorithm for sorting linked lists efficiently.

1) Sort the elements using Merge Sort. We will soon be writing a post about sorting a linked list. O(nLogn)

2) Remove duplicates in linear time using the algorithm for removing duplicates in sorted Linked

List. O(n)

Please note that this method doesn't preserve the original order of elements.

Time Complexity: O(nLogn)

METHOD 3 (Use Hashing)
We traverse the link list from head to end. For every newly encountered element, we check whether it is in the hash table: if yes, we remove it; otherwise we put it in the hash table.

Thanks to bearwang for suggesting this method.

Time Complexity: O(n) on average (assuming that hash table access time is O(1) on average).

Please write comments if you find any of the above explanations/algorithms incorrect, or a better ways to solve the same problem.

Related Tpoics:

- Given a linked list, reverse alternate nodes and append at the end

Recent Comments

- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List

Writing code in comment? Please use **ideone.com** and share the link here.