


Tek Bağlı Listelerde verilen bir değere sahip düğümü silme

Veriler: 

+ key değeri:


```

struct node *remove(struct node *head, int key){
    struct node *temp = head;
    if (head->data == key){
        head = head->next;
        free(temp);
    }
    else{
        while (temp->next->data != key){
            temp = temp->next;
        }
        struct node *temp2 = temp->next;
        temp->next = temp->next->next;
        free(temp2);
    }
    return head;
}

```

Diagram illustrating the removal process: A linked list with nodes 1, 2, 1, 3, NULL. The second node (2) is highlighted in yellow. A temporary pointer 'temp' points to the first node (1), and 'temp2' points to the second node (2). The second node is being removed, and the first node's next pointer is updated to point to the third node (1).

Örnek: Tek bağlı listenin elemanlarını tersten yazdıran bir fonksiyon yazınız.

Girdi: 

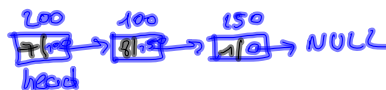
Çıktı: 3 1 8 7

```

void print_reverse(struct node *head){
    struct node *head2 = NULL;
    struct node *temp = head;
    while (temp != NULL){
        addhead(head2, temp->data);
        temp = temp->next;
    }
    print(head2);
}

void reverse_rec(struct node *head){
    if (head == NULL) { // base case
        return;
    }
    reverse_rec(head->next); // recursive case
    printf("%d ", head->data);
}

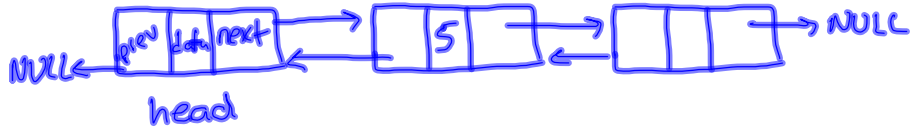
```



1 8 7

reverse_rec(200)
 ↓
 reverse_rec(100)
 ↓
 reverse_rec(150)
 ↓
 reverse_rec(0)

Çift Bağlı (Double Linked) Listeler



```

struct node {
    int data;
    struct *node next;
    struct *node prev;
}

```

```

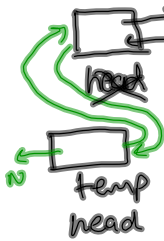
void insertAtFirst(int key) {

```

```

    if (head == NULL) {
        head = new node();
        head->data = key;
        head->next = NULL;
        head->prev = NULL;

```



```

    } else {
        struct node *temp = new node();
        temp->data = key;
        temp->next = head;
        temp->prev = NULL;
        head->prev = temp;
        head = temp;
    }
}

```

```

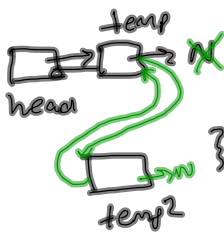
void insertAtEnd(int key) {

```

```

    if (head == NULL) {
        head = new node();
        head->data = key;
        head->next = NULL;
        head->prev = NULL;

```



```

    } else {
        struct node *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        struct node *temp2 = new node();
        temp2->data = key;
        temp2->next = NULL;
        temp2->prev = temp;
        temp->next = temp2;
    }
}

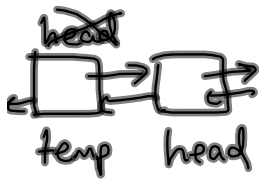
```

```

}

```

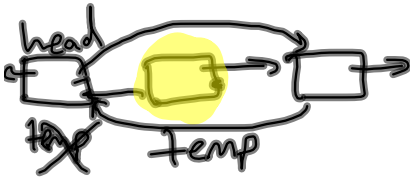
Örnek: Çift Bağlı Listelerde verilen bir değere sahip düğümü silme



```

void remove ( int key ) {
    struct node *temp = head;
    if ( head->data == key ) {
        head = head->next;
        head->prev = NULL;
        free ( temp );
    } else {

```



```

    while ( temp->data != key )
        temp = temp->next;
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    free ( temp );

```

```

    }

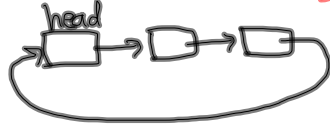
```

```

}

```

Dairesel Bağlı Listeler

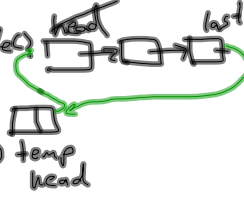


Başa Ekleme

```

void insertAtFront(int key){
    if (head == NULL){
        head = new node();
        head->data = key;
        head->next = head;
    }
    else{
        struct node *temp = new node();
        temp->data = key;
        struct node *last = head;
        while (last->next != head){
            last = last->next;
        }
        temp->next = head;
        last->next = temp;
        head = temp;
    }
}

```



Sona Ekleme

```

void insertAtEnd(int key){
    if (head == NULL){
        head = new node();
        head->data = key;
        head->next = head;
    }
    else{
        struct node *last = head;
        while (last->next != head){
            last = last->next;
        }
        last->next = new node();
        last->next->next = head;
    }
}

```



G.BL vs. TBL

Avantajları

- Her iki yönde gezilebilir
- Eklene daha kolaydır.

Dezavantajları

- Daha fazla yer kaplar
- 2 link olduğu için liste işlemleri daha yavaş
- Hata ihtimali yüksek

Soru: Çift bağlı dairesel bağlı listeler olur mu?

