1 D4 - TEKKOM B

PRAKTIKUM 5

DOUBLY LINKED LIST

Nama	:	Septian Bagus Jumantoro
Kelas	:	1 – D4 Teknik Komputer B
NRP	:	3221600039
Dosen	:	Dr Bima Sena Bayu Dewantara S.ST, MT.
Mata Kuliah	:	Praktikum Pemrograman Dasar 2
Hari/Tgl. Praktikum	:	Rabu, 06 April 2022

1. Percobaan 1 – Menambahkan node pada doubly linked list

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

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

/* Given a reference (pointer to pointer) to the head of a
list and an int, inserts a new node on the front of the
list. */
void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. Make next of new node as head and previous as NULL
    */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* 5. move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Given a node as prev_node, insert a new node after the
* given node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL) {
        printf("the given previous node cannot be NULL");
        return;
    }

    /* 2. allocate new node */
    struct Node* new_node
```

```
        = (struct Node*)malloc(sizeof(struct Node));

    /* 3. put in the data */
    new_node->data = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. Make the next of prev_node as new_node */
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
        new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head
of a DLL and an int, appends a new node at the end */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
        make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
        node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
```

```
    last->next = new_node;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}

// This function prints contents of linked list starting
// from the given node
void printList(struct Node* node)
{
    struct Node* last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL) {
        printf(" %d ", node->data);
        last = node;
        node = node->next;
    }

    printf("\nTraversal in reverse direction \n");
    while (last != NULL) {
        printf(" %d ", last->data);
        last = last->prev;
    }
}

/* Driver program to test above functions*/
int main()
{
    system("cls");
    /* Start with the empty list */
    struct Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning. So linked list becomes
    // 7->6->NULL
    push(&head, 7);

    // Insert 1 at the beginning. So linked list becomes
    // 1->7->6->NULL
    push(&head, 1);

    // Insert 4 at the end. So linked list becomes
    // 1->7->6->4->NULL
    append(&head, 4);
```

```
    // Insert 8, after 7. So linked list becomes
    // 1->7->8->6->4->NULL

    printf("\nCreated DLL is: ");
    printList(head);

    insertAfter(head->next, 8);

    push(&head, 3);

    append(&head, 5);

    insertAfter(head->next, 9);

    printf("\nNew DLL is: ");
    printList(head);

    getchar();
    return 0;
}
```

Output

```
Created DLL is:
Traversal in forward direction
 1  7  6  4
Traversal in reverse direction
 4  6  7  1
New DLL is:
Traversal in forward direction
 3  1  9  7  8  6  4  5
Traversal in reverse direction
 5  4  6  8  7  9  1  3
```

Analisa

Pada praktikum tersebut memakai Doubly Linked List (DLL), yang berisi penunjuk tambahan, biasanya disebut penunjuk sebelumnya, bersama dengan penunjuk berikutnya dan data yang ada dalam daftar tertaut tunggal.

2. Percobaan 2 – Menghapus sebuah node pada doubly linked list

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

/* a node of the doubly linked list */
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

/* Function to delete a node in a Doubly Linked List.
head_ref --> pointer to head node pointer.
del --> pointer to node to be deleted. */
void deleteNode(struct Node** head_ref, struct Node* del)
{
    /* base case */
    if (*head_ref == NULL || del == NULL)
        return;

    /* If node to be deleted is head node */
    if (*head_ref == del)
        *head_ref = del->next;

    /* Change next only if node to be deleted is NOT the last node */
    if (del->next != NULL)
        del->next->prev = del->prev;

    /* Change prev only if node to be deleted is NOT the first node */
    if (del->prev != NULL)
        del->prev->next = del->next;

    /* Finally, free the memory occupied by del*/
    free(del);
    return;
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning of the Doubly Linked List */
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;

    /* since we are adding at the beginning,
```

```
    prev is always NULL */
    new_node->prev = NULL;

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

    /* change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

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

void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL) {
        printf("the given previous node cannot be NULL");
        return;
    }

    /* 2. allocate new node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 3. put in the data */
    new_node->data = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. Make the next of prev_node as new_node */
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
        new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head
of a DLL and an int, appends a new node at the end */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
```

```
        = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
        make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
        node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}

/* Function to print nodes in a given doubly linked list
This function is same as printList() of singly linked list */
void printList(struct Node* node)
{
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Driver program to test above functions*/
int main()
{
    system("cls");
    /* Start with the empty list */
    struct Node* head = NULL;
```

```
    /* Let us create the doubly linked list 10<->8<->4<->2 */
    push(&head, 2);
    push(&head, 4);
    push(&head, 8);
    push(&head, 10);

    printf("\nOriginal Linked list\n");
    printList(head);

    append(&head, 1);
    append(&head, 3);
    append(&head, 5);
    insertAfter(head->next, 11);
    insertAfter(head->next, 12);
    insertAfter(head->next, 13);

    printf("\nNew Linked list\n");
    printList(head);

    /* delete nodes from the doubly linked list */
    deleteNode(&head, head); /*delete first node*/
    deleteNode(&head, head->next); /*delete middle node*/
    deleteNode(&head, head->next); /*delete last node*/

    /* Modified linked list will be NULL<-8->NULL */
    printf("\nModified Linked list\n");
    printList(head);

    getchar();
}
```

Output

```
Original Linked list
10 8 4 2
New Linked list
10 8 13 12 11 4 2 1 3 5
Modified Linked list
8 11 4 2 1 3 5
```

Analisa

Pada praktikum tersebut dapat diketahui bahwa untuk:

- Kompleksitas Waktu: O(1).

Karena traversal dari linked list tidak diperlukan, maka kompleksitas waktu adalah konstan.

- Kompleksitas Ruang: O(1).

Karena tidak ada ruang tambahan yang diperlukan, maka kompleksitas ruang adalah konstan.

3. Percobaan 3 – Membalik urutan node pada doubly linked list

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

/* a node of the doubly linked list */
struct Node
{
    int data;
    struct Node *next;
    struct Node *prev;
};

/* Function to reverse a Doubly Linked List */
void reverse(struct Node **head_ref)
{
    struct Node *temp = NULL;
    struct Node *current = *head_ref;

    /* swap next and prev for all nodes of
    doubly linked list */
    while (current != NULL)
    {
    temp = current->prev;
    current->prev = current->next;
    current->next = temp;
    current = current->prev;
    }

    /* Before changing head, check for the cases like empty
        list and list with only one node */
    if(temp != NULL )
        *head_ref = temp->prev;
```

```
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at the beginning of the Doubly Linked List */
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;

    /* since we are adding at the beginning,
    prev is always NULL */
    new_node->prev = NULL;

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

    /* change prev of head node to new node */
    if((*head_ref) != NULL)
    (*head_ref)->prev = new_node ;

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

/* Function to print nodes in a given doubly linked list
This function is same as printList() of singly linked list */
void printList(struct Node *node)
{
    while(node!=NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Driver code*/
int main()
{
    system("cls");
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Let us create a sorted linked list to test the functions
    Created linked list will be 10->8->4->2 */
    push(&head, 2);
```

```
    push(&head, 4);
    push(&head, 8);
    push(&head, 10);

    printf("\n Original Linked list ");
    printList(head);

    /* Reverse doubly linked list */
    reverse(&head);

    printf("\n Reversed Linked list ");
    printList(head);

    getchar();
}
```

Output

```
Original Linked list 10 8 4 2
Reversed Linked list 2 4 8 10
```

Analisa

Pada praktikum tersebut terdapat traverse linked list satu kali dan menambahkan elemen ke tumpukan, lalu men-traverse-kannya sekali lagi secara keseluruhan untuk memperbarui semua elemen. Keseluruhannya membutuhkan waktu 2n, yang merupakan kompleksitas waktu dari O(n).

4. Percobaan 4 – Mengurutkan/Sorting doubly linked list

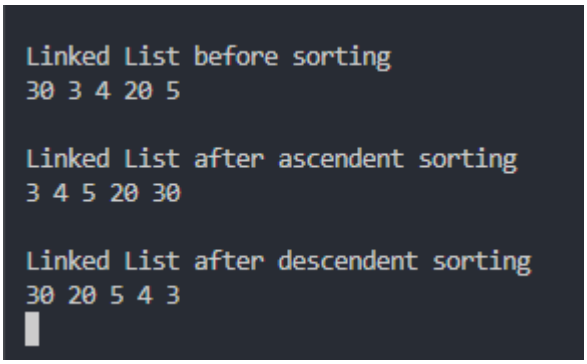
```
#include <stdio.h>
#include <stdlib.h>

/* a node of the doubly linked list */
struct Node
{
    int data;
    struct Node *next;
    struct Node *prev;
};

/* A utility function to swap two elements */
void swap ( int* a, int* b )
```

```
{ int t = *a; *a = *b; *b = t; }

// A utility function to find last node of linked list
struct Node *lastNode(struct Node *root)
{
    while (root && root->next)
        root = root->next;
    return root;
}

/* Considers last element as pivot, places the
pivot element at its correct position in sorted array,
and places all smaller (smaller than pivot) to left
of pivot and all greater elements to right of pivot */
struct Node* partition(struct Node *l, struct Node *h)
{
    // set pivot as h element
    int x = h->data;

    // similar to i = l-1 for array implementation
    struct Node *i = l->prev;

    // Similar to "for (int j = l; j <= h- 1; j++)"
    for (struct Node *j = l; j != h; j = j->next)
    {
        if (j->data <= x)
        {
            // Similar to i++ for array
            i = (i == NULL) ? l : i->next;

            swap(&(i->data), &(j->data));
        }
    }
    i = (i == NULL) ? l : i->next; // Similar to i++
    swap(&(i->data), &(h->data));
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(struct Node* l, struct Node *h)
{
    if (h != NULL && l != h && l != h->next)
    {
        struct Node *p = partition(l, h);
        _quickSort(l, p->prev);
        _quickSort(p->next, h);
    }
}
```

```
// The main function to sort a linked list.
// It mainly calls _quickSort()
void quickSort(struct Node *head)
{
    // Find last node
    struct Node *h = lastNode(head);

    // Call the recursive QuickSort
    _quickSort(head, h);
}

// A utility function to print contents of arr
void printList(struct Node *head)
{
    while (head)
    {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

void reverse(struct Node **head_ref)
{
    struct Node *temp = NULL;
    struct Node *current = *head_ref;

    /* swap next and prev for all nodes of
    doubly linked list */
    while (current != NULL)
    {
    temp = current->prev;
    current->prev = current->next;
    current->next = temp;
    current = current->prev;
    }

    /* Before changing head, check for the cases like empty
        list and list with only one node */
    if(temp != NULL )
        *head_ref = temp->prev;
}

/* Function to insert a node at the
beginning of the Doubly Linked List */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*)
            malloc(sizeof(struct Node)); /* allocate node */
```

```
    new_node->data = new_data;

    /* since we are adding at the beginning,
    prev is always NULL */
    new_node->prev = NULL;

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

    /* change prev of head node to new node */
    if ((*head_ref) != NULL) (*head_ref)->prev = new_node ;

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

// Driver Code
int main(int argc, char **argv)
{
    system("cls");
    struct Node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    printf("\nLinked List before sorting \n");
    printList(a);

    quickSort(a);

    printf("\nLinked List after ascendent sorting \n");
    printList(a);

    reverse(&a);

    printf("\nLinked List after descendent sorting \n");
    printList(a);

    getchar();

    return 0;
}
```

Output



Analisa

Pada praktikum tersebut, kompleksitas waktu implementasi di atas sama dengan kompleksitas waktu QuickSort() untuk array. Dibutuhkan O(n^2) waktu dalam kasus terburuk dan O(nLogn) dalam kasus rata-rata dan terbaik. Kasus terburuk terjadi ketika daftar tertaut sudah diurutkan. Quicksort dapat diimplementasikan untuk Linked List hanya jika kita dapat memilih titik tetap sebagai pivot (seperti elemen terakhir dalam implementasi di atas). QuickSort Acak tidak dapat diterapkan secara efisien untuk Daftar Tertaut dengan memilih pivot acak.