Lab assignment - 4

Name: srujitha Devineni
ID: AP21110010086

Implementation of Priority Queue using Linked List

OBJECTIVE :-

By the end of this assignment, we will be able to
I) use the concepts of push,pop,peek operations using queue.
II) Can Implement Priority Queue using Linked List.

PROBLEM STATEMENT :-

Implement Priority Queue using Linked List.

- Implementation of priority Queue data by using a linked list is more efficient as compared to arrays.
- The linked list provides you with the facility of Dynamic memory allocation.
- The memory is not wasted as memory, not in use can be freed, using free(); method

---

ALGORITHM :-

Start
Step 1-> Declare a struct node
  Declare data, priority
  Declare a struct node* next
Step 2-> In function Node* newNode(int d, int p)
  Set Node* temp = (Node*)malloc(sizeof(Node))
  Set temp->data = d
  Set temp->priority = p
  Set temp->next = NULL
  Return temp
Step 3-> In function int peek(Node** head)
  return (*head)->data
Step 4-> In function void pop(Node** head)
  Set Node* temp = *head
  Set (*head) = (*head)->next
  free(temp)
Step 5-> In function push(Node** head, int d, int p)
  Set Node* start = (*head)
  Set Node* temp = newNode(d, p)

If (*head)->priority > p then,
Set temp->next = *head
Set (*head) = temp
Else
Loop While start->next != NULL && start->next->priority < p
Set start = start->next
Set temp->next = start->next
Set start->next = temp
Step 6-> In function int isEmpty(Node** head)
Return (*head) == NULL
Step 7-> In function int main()
Set Node* pq = newNode(7, 1)
Call function push(&pq, 1, 2)
Call function push(&pq, 3, 3)
Call function push(&pq, 2, 0)
Loop While (!isEmpty(&pq))
Print the results obtained from peek(&pq)
Call function pop(&pq)
Stop

PROGRAM

File Name Saved as:- Priority Queue using Linked List

Input Used:- 10 20 30 40

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

struct Node
{
        int data;
        int priority;   // lower value means higher priority
        struct Node* next;
};

// Function to Create A New Node

Node* create(int data, int priorityVal)
{
   Node* temp = (Node*) malloc(sizeof(Node));
   temp->data = data;
   temp->priority = priorityVal;
   temp->next = NULL;
```

```
    return temp;
}

// Return the value at head

int peek(Node** head)
{
    return (*head)->data;
}

// Function to push in accodance with priority

void push(Node** head, int data, int priorityVal)
{
    Node* curr = (*head);

    // Create new Node

    Node* temp = create(data, priorityVal);

    // If incoming node has lower priority value
    // than the current head. This incoming node
    // is inserted before head
    // Note: Lower priority value means higher priority

    if ((*head)->priority > priorityVal)
{

        // new node inserted before head

        temp->next = *head;
        (*head) = temp;
    }
    else
 {

        // else we traverse the list to find
        // correct position to insert the incoming node

        while (curr->next != NULL &&
            curr->next->priority < priorityVal) { curr = curr->next;
        }

        // incoming node inserted here
        // either at req. position or end of the list
        temp->next = curr->next;
        curr->next = temp;
```

```
    }
}

// Here we remove the element with highest priotity
// highest priority will be at the head itself

void pop(Node** head)
{
    Node* temp = *head;
    (*head) = (*head)->next;
    printf("(%d, priority: %d) popped\n",temp->data,temp->priority);
    free(temp);
}

// Function to check is list is empty

int isEmpty(Node** node)
{
    return (*node) == NULL;
}

void display(struct Node* node){
    printf("Priority Queue: ");
    // as linked list will end when Node is Null
    while(node!=NULL)
    {
        printf("%d ",node->data);
        node = node->next;
    }
    printf("\n");
}
int main()
{
    Node* pq = create(10, 1);
            push(&pq, 30, 3);
            push(&pq, 20, 2);
            push(&pq, 40, 4);

            display(pq);

            pop(&pq);
            pop(&pq);

            display(pq);

            push(&pq, 15, 2);
            // if two items have same priority then
            // they are served in their order of entry
```

```
        push(&pq, 20, 2);
        push(&pq, 50, 5);
        push(&pq, 5, 1);

        display(pq);

    return 0;
}
```

Output

Priority Queue: 10 20 30 40
(10, priority: 1) popped
(20, priority: 2) popped
Priority Queue: 30 40
Priority Queue: 5 15 20 30 40 50

PROBLEMS FACED.

Implementing a queue and linked list separetley is a hard task for me but combining both and doing was hard at starting but later it was easy ,from this i learnt more about queue and linked list.

CONCLUSION

A) By doing this assignment I got to learn about push,pop,peek operations using queue..
B) Learned how to Implement Priority Queue using Linked List .
C)This program benefited me to learn a newthing and improve old concepts(i.equeue,linkedlist).

Polynomial addition using linked list

OBJECTIVE :-

By the end of this assignment, we will be able to
I) Can learn how to use structures and recursion .
II) Can Implement Polynomial addition using Linked List.

PROBLEM STATEMENT :-

Polynomial addition using Linked List.

We store each polynomial as a singly linked list where each node stores the exponent and coefficient in the data part and a reference to the next node. Their sum is then stored in another linked list.

ALGORITHM :-

Step 1: Start.
Step 2: Define user defined datatype node consisting of int coefficient and exponent
Step 3: Defining create function
struct node* create(struct node* head, int co, int exp)
Check if (head == Null)
temp <- GetNode (node)
temp.co<- CO
temp. exp<- exp
temp. next<- Null
Set head<- temp
otherwise
Set temp <- head
while(temp. next != Null) do
temp<-temp.next
flag<- GetNode (node)
Flag.co<-co
flag. exp<- exp
flag.next<- Null
temp. next <- flag
return head
Step 4: Defining Addition function
struct node* polyAdd (struct node *pl, struct node *p2, struct node *sum)

Set poly1<- p1
Set poly2<- p2
Set result<- Null

```
check if (poly1 != Null AND poly2 == Null)
Set sum <- poly1
Return sum
else if (poly== Null AND poly2 != Null)
Set sum <- poly2
Return sum
while (poly1 != Null AND poly2 != Null) do
Check if (sum == Null)
sum<-GetNode(node)
Set result<- SUM
Otherwise
result.next<- GetNode (node)
Set result<-result.next
Check if (poly1.exp> poly2. exp)
Setresult.co<= poly1.Co
Set result.exp<- pOly1.exp

Set poly1 <- poly1.next
Check if (poly1.exp< poly2. exp)
Set result.co<- poly2.co
Set result.exp<- poly2. exp
Set poly2<- poly2. next
Otherwise
Set result.co<- polv1.co + poly2.co
Set result.exp<- polyl.exp
Set poly1<- poly1.next
Set poly2<- poly2.next

while(poly1 !=Null) do
Set result.next<- GetNode (node)
Set result<- result. next
Set result.co<- poly1.c0
Set result.exp<- poly1. exp
Set poly1<- poly1.next
while(polv2 # Null) do
Set result.next<- GetNode(node)
Set result<-result.next
Set result.co<- poly2.co
Set result. exp<- poly2.exp
Set poly2 <- poly2.next
Set result.next<-Null
Return Sum
```

Step 5: Defining Display function

```
void display(struct node* head)
Set temp <- head
while(temp Null) do
Display temp.co, temp.exp
```

Set temp <- temp.next
Step 6: Defining Main function
Set p1<- Null
Set p2 <- Null
Set sum<-Null
Set flag<-1
while(flag== 1) do
Display"1: Create Polynomial 1"
Display"2: Create Polynomial 2"
Display"3: Perform Addition"
Display"4: Exit"
Read choice
switch(choice)
Case 1:
Display"Enter coefficient for polynomial 1"
Read co
Display"Enter exponent for polynomial 1"
Read exp
Call create(p1, co, exp)
Case 2:
Display"Enter coefficient for polynomial 2"
Read co
Display"Enter exponent for polynomial 2"
Read exp
Call create (p2, co, exp)
Case 3:
Set sum<-call polyAdd (pl, p2, sum)
Call display(sum)
Case 4:
Set flag<-0
End switch
Step 7: Stop

PROGRAM

File Name Saved as:- Priority Queue using Linked List

Input Used:-1,5,4,1,2,2,1,3,1,2,4,2,2,7,1,3,4

```
#include<stdio.h>
#include<stdlib.h>
//polynomial node structure
struct node
{
```

```
int co, exp;
struct node* next;
};
//create a polynomial
struct node* create(struct node* head, int co, int exp)
{
struct node *temp, *flag;
//if polynomial empty. make the node the head node
if (head == NULL)
{
temp = (struct node*) malloc (sizeof(struct node)) ;
temp->co = co;
temp->exp = exp;
temp->next = NULL;
head = temp;
}
else
{
//else go to the last node and append
   temp = head;
   while(temp->next != NULL)
   temp = temp->next;
   flag = (struct node *)malloc(sizeof (struct node)) ;
   flag->co = co;
   flag->exp = exp;
   flag->next = NULL;
   temp->next = flag;
}
   return head;
}
//add two polynomial
struct node* polyAdd(struct node *p1,struct node *p2,struct node *sum)
{
   //copy the two polynomial and initialize variable res to store the sum
   struct node *poly1=p1,*poly2=p2,*res;
   //if polynomial 2 is null,set polynomial 1 as the sum
   if (poly1 != NULL && poly2== NULL)
   {
   sum = poly1;
      return sum;
   }
   //if polynomial 1 is null, set polynomial 2 as the sum
else if (poly1 == NULL && poly2 != NULL)
{
sum= poly2;
return sum;
}
//if both polynomials are non-empty
```

```
while(poly1 != NULL && poly2 != NULL)
{
//if the sum is empty, initialize sum with a node structure
//and set res equal to sum
if (sum == NULL)
{
sum = (struct node *)malloc(sizeof(struct node)) ;
res= sum;
}
   //add a new node structure at the end of res to store sum
else
{
res->next = (struct node *)malloc(sizeof (struct node)); ;
res = res->next;
}
//if exponent of current node of polynomial 1 is greater than th
//add it to the sum
if (poly1->exp > poly2->exp)
{
   res->co = poly1->co;
   res->exp = poly1->exp;
   poly1 = poly1->next;
}
   //if exponent of current node of polynomial 2 is greater than th
   //add it to the sum
   else if (poly1->exp < poly2->exp)
   {
   res->co = poly2->co;
   res->exp = poly2->exp;
   poly2 = poly2->next;
   }
   //if exponent of current node of polynomial 1 is equal to that
   //add the sum of their co-efficient to the sum
   else if (poly1->exp == poly2->exp)
   {
   res->co = poly1->co + poly2->co;
   res->exp = poly1->exp;
   poly1 = poly1->next;
   poly2 = poly2->next;
   }
}
   //if polynomial 1 is non-empty add the remaining nodes to the sun
   while (poly1 != NULL)
   {
      res->next = (struct node *)malloc(sizeof(struct node)) ; ;
      res = res->next;
      res->co = poly1->co;
      res->exp = poly1->exp;
```

```
        poly1 = poly1->next;
        }
        //if polynomial 2 is non-empty add the remaining nodes to the sun
        while(poly2 != NULL)

        {
        res->next = (struct node *)malloc(sizeof (struct node));
        res = res->next;
        res->co = poly2->co;
        res->exp = poly2->exp;
        poly2 = poly2->next;
        }
        //set pointer of last node to null
        res->next = NULL;
        //return the head node of the sum
        return sum;
        }
        //display polynomial
void display(struct node* head)
{
struct node *temp=head;
while(temp != NULL)
{
printf("d^d+", temp->co, temp->exp) ;
temp=temp->next;
}
printf("\n");
}
int main()
{
//to store polynomial 1, polynomial 2 and the sum
struct node *p1 = NULL, *p2 = NULL, *sum = NULL;
int ch, co, exp;
int loop = 1;
while(loop)
{
printf("1. Add to Polynomial 1\n") ;
printf("2. Add to Polynomial 1\n") ;
   printf("3. Perform Addition\n") ;
printf("4. Exit\n") ;
scanf("%d", &ch) ;
switch(ch)
{
case 1:
printf ("Enter co-efficient\n");
scanf("%d", &co);
printf("Enter exponent\n") ;
scanf("%d", &exp) ;
```

```
p1 = create (p1, co, exp) ;
break;
case 2:
printf ("Enter co-efficient\n");
scanf("%d", &co) ;
printf( "Enter exponent\n");
scanf("%d", &exp) ;
p2 = create (p2, co, exp) ;
break;
case 3:
sum=polyAdd(p1, p2, sum) ;
printf("\nPolynomial 1\n") ;
display(p1);
printf("\nPolynomial 2\n" ) ;
display(p2);
printf("\nSum:\n");
display(sum);
break;
case 4:
loop = 0;
break;
default: printf("Wrong Choice! Re-enter\n");
break;
}
}
};
```

Output

```
1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
1
Enter co-efficient
5
Enter exponent
4

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
1
Enter co-efficient
2
Enter exponent
```

2

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
1
Enter co-efficient
3
Enter exponent
1

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
2
Enter co-efficient
4
Enter exponent
2

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
2
Enter co-efficient
7
Enter exponent
1

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition
4. Exit
3
Polynomial 1
5^ 4+2 ^ 2+3 ^ 1 +
Polynomial
4 ^ 2+7 ^ 1 +
Sum:
5 ^ 4 +6 ^ 2 +10 ^ 1 +

1. Enter Polynomial 1
2. Enter Polynomial 2
3. Perform Addition

4. Exit

4

PROBLEMS FACED.

This program seems to be hard,implementing and visulazing it seems difficult .

CONCLUSION

A.)Can learn how to use structures and recursion .
B.)Can Implement Polynomial addition using Linked List.

Sparse matrix operations using linked list

OBJECTIVE :-

By the end of this assignment, we will be able to
1.)Sparse matrix operations using linked list

PROBLEM STATEMENT :-

Sparse matrix operations using linked list ,In linked list representation, we use a linked list to represent a sparse matrix. Each node of the linked list has four fields.
These four fields are defined as:
Row: Row keeps row index of a non-zero element
Column: Column keeps column index of a non-zero element
Value: non zero element located at (row,column) index
Next node: Next node, stores the address of the next node

ALGORITHM :-

1.)Start
2.)Test a matrix to be sparse
3.)Declare and initialize a two-dimensional array intArray[][]
4.)Loop through the array and count the number of zeroes present in the given array and store in the variable count.
5.)Calculate the size of the array by multiplying the number of rows with many columns of the array.
6.)If the count is greater than size/2, given matrix is the sparse matrix. That means, most of the elements of the array are zeroes,
7.)Else, the matrix is not a sparse matrix.
8.)Stop

PROGRAM

File Name Saved as:- Sparse matrix operations using linked list
Input Used:-1,2,4

```
#include<stdio.h>
#include<stdlib.h>
#define R 4
#define C 5
```

```
// Node to represent row - list
struct row_list
{
    int row_number;
    struct row_list *link_down;
    struct value_list *link_right;
};

// Node to represent triples
struct value_list
{
    int column_index;
    int value;
    struct value_list *next;
};

// Function to create node for non - zero elements
void create_value_node(int data, int j, struct row_list **z)
{
    struct value_list *temp, *d;

    // Create new node dynamically
    temp = (struct value_list*)malloc(sizeof(struct value_list));
    temp->column_index = j+1;
    temp->value = data;
    temp->next = NULL;

    // Connect with row list
    if ((*z)->link_right==NULL)
        (*z)->link_right = temp;
    else
    {
        // d points to data list node
        d = (*z)->link_right;
        while(d->next != NULL)
            d = d->next;
        d->next = temp;
    }
}

// Function to create row list
void create_row_list(struct row_list **start, int row,
                    int column, int Sparse_Matrix[R][C])
{
    // For every row, node is created
    for (int i = 0; i < row; i++)
    {
```

```
        struct row_list *z, *r;

        // Create new node dynamically
        z = (struct row_list*)malloc(sizeof(struct row_list));
        z->row_number = i+1;
        z->link_down = NULL;
        z->link_right = NULL;
        if (i==0)
            *start = z;
        else
        {
            r = *start;
            while (r->link_down != NULL)
                r = r->link_down;
            r->link_down = z;
        }

        // Firstiy node for row is created,
        // and then traversing is done in that row
        for (int j = 0; j < 5; j++)
        {
            if (Sparse_Matrix[i][j] != 0)
            {
                create_value_node(Sparse_Matrix[i][j], j, &z);
            }
        }
    }
}

//Function display data of LIL
void print_LIL(struct row_list *start)
{
    struct row_list *r;
    struct value_list *z;
    r = start;

    // Traversing row list
    while (r != NULL)
    {
        if (r->link_right != NULL)
        {
            printf("row=%d \n", r->row_number);
            z = r->link_right;

            // Traversing data list
            while (z != NULL)
            {
                printf("column=%d value=%d \n",
```

```
                z->column_index, z->value);
            z = z->next;
        }
      }
      r = r->link_down;
   }
}

//Driver of the program
int main()
{
   // Assume 4x5 sparse matrix
   int Sparse_Matrix[R][C] =
   {
      {0 , 0 , 3 , 0 , 4 },
      {0 , 0 , 5 , 7 , 0 },
      {0 , 0 , 0 , 0 , 0 },
      {0 , 2 , 6 , 0 , 0 }
   };

   // Start with the empty List of lists
   struct row_list* start = NULL;

   //Function creating List of Lists
   create_row_list(&start, R, C, Sparse_Matrix);

   // Display data of List of lists
   print_LIL(start);
   return 0;
}
```

Output

```
row=1
column=3 value=3
column=5 value=4
row=2
column=3 value=5
column=4 value=7
row=4
column=2 value=2
column=3 value=6
```

PROBLEMS FACED.

visualizing and doing this problem is hard but doing many times it was easy .

CONCLUSION

By doing this assignment I got to learn about
1.)Sparse matrix operations using linked list