PAGE EDGE

DATE : / /

Sl.No.	Date	Title	Signature Date
1	21/12/23	Swapping two variables Dynamic memory allocation -on Stack Implementation.	21/12/23
2.	28/12/23	Infix to Postfix. Queue Implementation circular queue implemen -tation.	18/1/24 11/1/24
3.	11/1/24	Implemenatation of linked list Leetcode - Minstack's	18/1/24
4.	18/1/24	Implemenatation of deletion operation in Singly linked list	18/1/24.
5	25/1/24	Linked list operation Stack & Queue in Linked list	25/1/24
6.	1/02/24	Doubly linked list	1/2/24
7.	15/2/24	Binary search tree Leetcode - 2.	15/2/24
8.	22/2/24	BFS, DFS (Leetcode 3&4)	22/2/24.
9.	29/2/24	Hashing (Hackerrank)	29/2/24

PAGE EDGE

DATE: / /

21/12/23 Week-9.

1) Write a c program with a function to swap two numbers using pointers:

```
#include<stdio.h>
int main()
{
int a;
int b;
int temp;
int *ptr1, *ptr2;
printf("Enter value of a:");
scanf("%d", &a);
printf("Enter value of b:");
scanf("%d", &b);
printf("Before swapping two numbers is a=%d and b=%d");
ptr1 = &a;
ptr2 = &b;
temp = *ptr1;
*ptr1 = *ptr2;
*ptr2 = temp;
printf("After swapping two numbers is a=%d
        and b=%d", a, b);
return 0;
}
```

output:

```
Enter value of a: 5
Enter value of b: 3
Before swapping two numbers.
After swapping two numbers is a=3 and
                        b=5
```

9) #include<stdio.h>
#include<stdlib.h>
int main()
{
int *ptr;
int n, i;
n=5
printf("Enter number of elements: %d \n", n);
~~ptr=(int*) calloc(n, sizeof(int));~~
ptr=(int*)malloc(n* sizeof(int));
if(ptr==NULL){
printf("Memory not allocated. \n");
exit(0);
}
else{
printf("Memory successfully allocated
using malloc. \n");
for(i=0; i<n; ++i){
ptr[i]=i+1;
}
printf("The elements of the array are:");
for(i=0; i<n; ++i){
printf("%d,", ptr[i]);
}
ptr=(int*)calloc(n, sizeof(int));
if(ptr==NULL)
{
printf("Memory not allocated. \n");
exit(0);
}
else{
printf("Memory successfully allocated using

```
calloc.\n");
  for(i=0; i<n; ++i){
    ptr[i]= i+1;
  }
printf("The elements of the array are:");
  for(i=0; i<n; ++i){
    printf("%d,", ptr[i]);
  }
n=10;
printf("\n\n Enter the new size of the array:
        %d\n", n);
  ptr=(int*) realloc(ptr, n* sizeof(int));
printf("Memory successfully re-allocated
  using realloc.\n");
  for(i=5; i<n; ++i){
    ptr[i]= i+1;
  }
printf("\n\nThe elements of the array are:");
  for(i=0; i<n; ++i){
    printf("%d,", ptr[i]);
  }
  free(ptr);
}
return 0;
}
```

Output:

Enter number of elements: 5
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5
Memory successfully allocated using calloc.

The elements of the array are: 1, 2, 3, 4, 5,
Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

3)
```
#include<stdio.h>
int stack[SIZE];
int top=-1;
void push(int element);
void pop();
void display();
int main(){
    int choice,element;
do{
  printf("\nStack operations:\n");
  printf("1. Push\n");
  printf("2. Pop\n");
  printf("3. Display\n");
  printf("4. Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&choice);

switch(choice){
    case1:
      printf("Enter the element to push:");
      scanf("%d",&element);
      push(element);
      break;
   case2:
      pop();
```

```
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Exiting the program.\n");
        break;
    default:
        printf("Invalid choose. Please enter a valid
    option.\n");
    }
    while(choice != 4);
    return 0;
}
void push(int element)
{ if(top==SIZE-1){
    printf("Stack overflow! cannot push element.\n");
} else {
    top++;
    stack[top]=element;
    printf("%d pushed onto the stack.\n",element);
}
}
void pop() {
    if(top==-1){
        printf("Stack underflow! cannot pop
            element.\n");
    } else {
        printf("%d poped from the stack.\n",
            stack[top]);
        top--;
    }
```

```
}
void display(){
    if(top==-1){
        printf("stack is empty. Nothing to
display.\n"); }
    else{
        printf("Elements in the stack:\n");
        for(int i=0; i<=top; i++){
            printf("%d", stack[i]);
        }
        printf("\n");
    }
}
```

output:

```
Stack operations:
1. Push
2. POP
3. display
4. exit.
Enter your choice: 1
Enter the element to push: 34
34 pushed onto the stack

Stack operations:
1. Push
2. POP
3. display
4. exit
Enter your choice: 1
Enter the element to push: 9
```

9 Pushed onto the stack

Stack operations:
1. Push
2. Pop
3. display
4. Exit
Enter your choice: 3
Elements in the stack:
34

Stack operations:
1. push
2. pop
3. display
4. exit.
Enter your choice: 2
9 poped from the stack.
Stack operations:
1. push
2. Pop
3. display
4. exit
Enter your choice: 4
Exiting the program.

Ex. 1
21/2/23

28/12/23 Week - 3

Lab program 2

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply), / (divide) and ^ (power)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 100

char stack[MAX];
char infix[MAX];
char postfix[MAX];
int top=-1;

void push(char);
char pop();
int isEmpty();
void InToPost();
void print();
int precedence(char);

int main()
{
  printf("Enter infix expression: ");
  gets(infix);
  InToPost();
  print();
```

Executed
18/1/24

```
return 0;
}
void InToPost()
{
  int i, j=0;
  char symbol, next;
  for(i=0; i<strlen(infix); i++)
  {
    symbol = infix[i];
    switch(symbol)
    {
    case '(':
        push(symbol);
        break;
    case ')':
      while((next=pop())!='(')
          postfix[j++]=next;
          break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
      while(!isEmpty() && precedence(stack
        [top]) >= precedence(symbol))
          postfix[j++]=pop();
      push(symbol);
      break;
    default:
      postfix[j++]= symbol;
  }
}
```

```
return 0;
}
void InToPost()
{
  int i, j=0;
  char symbol, next;
  for(i=0; i<strlen(infix); i++)
  {
    symbol = infix[i];
    switch(symbol)
    {
    case '(':
        push(symbol);
        break;
    case ')':
        while((next=pop())!='(')
            postfix[j++]=next;
            break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
        while(!isempty() && precedence(stack
        [top])>=precedence(symbol))
            postfix[j++]=pop();
        push(symbol);
        break;
    default:
        postfix[j++]= symbol;
    }
}
```

```
while (!isEmpty())
        postfix[j++] = pop();
postfix[j] = '\0';
}

int precedence(char symbol)
{
   switch(symbol)
    {
     case '^':
          return 3;
     case '/':
     case '*':
          return 2;
     case '+':
     case '-':
          return 1;
     default:
          return 0;
    }
}

void print()
{
 int i=0;
 printf("The equivalent postfix expression is:");
 while(postfix[i])
  {
     printf("%c", postfix[i++]);
  }
  printf("\n");
}
```

```
void push(char c)
{
  if(top==MAX-1)
  {
     printf("stack overflow");
     return;
  }
  top++;
  stack[top]=c;
}
char pop()
{
  char c;
  if(top==-1)
  {
     printf("stack overflow");
     exit(1);
  }
  c=stack[top];
  top=top-1;
  return c;
}
int isempty()
{
  if(top==-1)
     return 1;
  else
     return 0;
}
```

Output:

```
Enter infix expression: ((a+b)-(c*d))
The equivalent postfix expression is:
                abt cd* -
```

2) Postfix evaluation:

```
#include<stdio.h>
int stack[20];
int top=-1;
void push(int a)
{
   stack[++top]=a;
}
int pop()
{
  return stack[top--];
}
int main()
{
  char exp[20];
  char *e;
  int n1,n2,n3,num;
  printf("Enter the expression:");
  scanf("%s",exp);
  e=exp;
  while(*e!='\0')
  {
    if(isdigit(*e))
     {
       num=*e-48;
```

```
push(num);
}
else
{
n1=pop();
n2=pop();
switch(*e)
{
case '+':
{
n3=n1+n2;
break;
}
case '-':
{
n3=n2-n1;
break;
}
case '*':
{
n3=n1*n2;
break;
}
case '/':
{
n3=n2/n1;
break;
}
}
push(n3);
}
e++;
```

```
}
printf("\n the result of expression %s =
%d \n\n", exp, pop());
return 0;
}
```

output:

```
Enter the expression : 23*5+
The result of expression 23*5+ = 11.
```

3a)

```
#include<stdio.h>
#define MAX 50
void insert();
void delete();
void display();
int queue_array[MAX];
int rear=-1;
int front=-1;
main()
{
  int choice;
  while(1)
  {
  printf("1. insert element\n");
  printf("2. delete element\n");
  printf("3. display elements:\n");
  printf("4. quit\n");
  printf("Enter your choice:");
  scanf("%d", &choice);
  switch(choice)
  {
```

```
case 1:
  insert();
  break;
case 2:
  delete();
  break;
case 3:
   display();
   break;
case 4:
   exit(1);
   default
    printf("wrong choice \n");
  }
 }
}
void insert()
{
  int add_item;
  if (rear==MAX-1)
  printf("Queue overflow \n");
 .. else {
    if (front==-1)
     front=0;
    printf("Insert the element:");
    scanf("%d", &add_item);
    rear=rear+1;
    queue_array[rear]=add_item;
  }
}
void delete()
{
```

18/11/24

```
if (front==-1 || front>rear)
{
  printf("queue underflow \n");
  return;
}
else
{
  printf("element deleted from queue is:
        %d \n", queue_array[front]);
  front=front+1;
}
}
void display()
{
  int i;
  if (front==-1)
    printf("queue is empty \n");
  else
    {
    printf("queue is: \n");
    for(i=front; i<=rear; i++)
      printf("%d", queue_array[i]);
    printf("\n");
    }
}
```

output

1. Insert element
2. Delete element
3. Display element
4. Quit.

- Enter your choice: 1
  Insert element : 23
- Enter your choice: 1
  Insert element: 24
- Enter your choice: 3
  Queue is 23, 24
- Enter your choice: 2
  element deleted from queue is: 23
- Enter your choice: 4.

11/1/24 Week - 4

3b) Circular Queue

```
#include<stdio.h>
#include<stdlib.h>
# define size 50
int q[size];
int rear=-1;
int front=-1;
int ISFull()
{
  if(front==(rear+1)%size)
  {
    return 0;
  }
  else
  {
    return -1;
  }
}
int IsEmpty()
{
  if(front==-1 && rear==-1)
  {
    return 0;
  }
  else
  {
    return -1;
  }
}
```

```
void Enqueue(int x)
{
  int item;
  if(IsFull()==0)
  {
    printf("Queue Overflow \n");
    return;
  }
  else
  {
   if(IsEmpty()==0)
   {
     front=0;
     rear=0;
   }
  else
   {
   rear=(rear+1)%size;
   }
   q[rear]=x;
  }
}
int Dequeue()
{
  int x;
  if(IsEmpty()==0)
  {
    printf("Queue underflow \n");
  }
  else
  {
    if(front==rear)
```

```
{
  x=Q[front];
  front=-1;
  rear=-1;
}
else
{
  x=Q[front];
  front=(front+1)%size;
}
return x;
}
}
void display()
{
  int i;
  if(IsEmpty()==0)
  {
    printf("Queue is empty\n");
  }
  else
  {
    printf("Queue elements:\n");
    for(i=front; i!=rear; i=(i+1)%size)
    {
      printf("%d\n", Q[i]);
    }
    printf("%d\n", Q[i]);
  }
}
```

PAGE EDGE

DATE : / /

```
Void main()
{
  int choice, x, b;
  while(1)
  {
    printf("1. Enqueue, 2. Dequeue, 3. Display,
            4. Exit \n");
    printf("Enter your choice: \n");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
        printf("Enter the number to be inserted
                into the queue \n");
        scanf("%d", &x);
        Enqueue(x);
        break;
      case 2:
        b = Dequeue();
        printf("%d was removed from the queue \n",
                b);
        break;
      case 3:
        Display();
        break;
      case 4:
        exit(1);
      default:
        printf("Invalid input \n");
    }
  }
}
```

Output:

1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
Enter your choice: 1
Enter the number to be inserted into the queue: 23
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
Enter your choice: 1
Enter the number to be inserted into the queue: 24
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
Enter your choice: 3
Queue element: 23, 24
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
Enter your choice: 4

singly linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node * next;
};
void display();
void insert_begin();
void insert_end();
void insert_pos();
struct node *head=NULL;
void display()
{
  printf("Elements are :\n");
  struct node *ptr;
  if(head==NULL)
  {
    printf("List is empty\n");
    return;
  }
else
{
  ptr=head;
  while(ptr!=NULL)
  {
    printf("%d \n",ptr->data);
    ptr=ptr->next;
  }
}
```

```
}
void insert_begin()
{
  struct node *temp;
  temp=(struct node *) malloc(sizeof(struct node));
  printf("Enter the value to be inserted \n");
  scanf("%d", &temp->data);
  temp->next =NULL;
  if (head ==NULL)
    head= temp;
  else
  {
   temp-> next =head;
    head=temp;
  }
}
void insert_end()
{
  Struct node *temp, *ptr;
  temp=(Struct node*) malloc(sizeof(struct node));
  printf("Enter the value to be inserted \n");
  scanf("%d", &temp->data);
  temp0 ->next = NULL;
  if(head==NULL)
  {
    head=temp;
  }
  else
  {
    ptr=head;
    while (ptr->next !=NULL)
    {
```

```
    ptr = ptr->next;
    }
    ptr->next = temp;
  }
}
void insert_pos()
{
  int pos, i;
  struct node *temp, *ptr;
    printf("enter the position");
  scanf("%d", &pos);
  temp=(struct node*) malloc(sizeof(struct node));
  printf("enter the value to be inserted \n");
  scanf("%d", &temp->data);
  temp->next = NULL;
  if(pos == 0)
  {
    temp->next = head;
    head = temp;
  }
else
{
  for(i=0, ptr=head; i<pos-1; i++)
  {
    ptr = ptr->next;
  }
  temp->next = ptr->next;
  ptr->next = temp;
  }
}
void main()
{
```

```
int choice;
while(1)
{
printf("\n1. to insert at the beginning \n
        2. to insert at the end \n
        3. to insert at the position \n
        4. to display \n 5. exit \n");
printf("enter your choice : \n");
scanf("%d", &choice);
switch(choice)
{
case 1:
    insert_begin();
    break;
case 2:
    insert_end();
    break;
case 3:
    insert_pos();
    break;
case 4:
    display();
    break;
case 5:
    exit(0);
    break;
default:
    printf("Invalid choice \n choice \n");
    break;
}
}}
```

Output:

1. to insert at the beginning
2. to insert at the end
3. to insert at any position enter
4. to display
5. to exit
Enter your choice: 1
Enter the value to be inserted 23.

1. to insert at the beginning
2. to insert at the end
3. to insert at any position enter
4. to display
5. to exit
Enter your choice: 2.
Enter the value to be inserted 13

1. to insert at the beginning
2. to insert at the end
3. to insert at any position enter
4. to display
5. to exit
Enter to your choice: 3.
Enter the position: 1
Enter the valued to be inserted
25

1. to insert at the beginning
2. to insert at the end
3. to insert at any position enter
4. to display

5. to exit.
Enter your choice: 4
23, 25, 13

18/1/24 Week-5

Singly linked list.

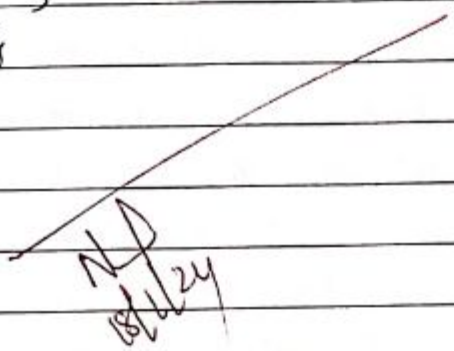
```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();
struct node *head=NULL;
void display()
{
  printf("Elements are:\n");
  struct node *ptr;
  if(head==NULL)
  {
    printf("List is empty\n");
    return;
  }
else
```

```
{
  ptr=head;
  while(ptr!=NULL)
  {
   printf("%d \n", ptr->data);
   ptr= ptr -> next;
  }
}
void insert()
{
 struct node *temp, *ptr;
 temp=(struct node *) malloc(sizeof(struct node));
 printf("Enter the value to be inserted \n");
 scanf("%d", &temp->data);
 temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
 }
 else
 {
 ptr=head;
 while(ptr->next!=NULL)
 {
   ptr=ptr->next;
 }
  ptr->next=temp;
 }
}
```

```
void delete_begin()
{
  struct node *ptr;
  if(head == NULL)
  {
    printf("List is empty \n");
    return;
  }
  else
  {
    ptr=head;
    head = head ->next;
    printf("the element deleted from the list -%d \n",
        ptr -> data);
    free(ptr);
  }
}
void delete_end()
{
  struct node *ptr, *temp;
  if(head==NULL)
  {
    printf("List is empty \n");
    return;
  }
  else
  {
    ptr=head;
    while(ptr ->next !=NULL)
    {
      temp=ptr;
      ptr = ptr -> next;
```

```
    }
    temp->next = NULL;
    printf("The element deleted - %d", ptr->data);
    free(ptr);
    }
}
void delete_pos()
{
  int i, pos;
  struct node *ptr, *temp;
  if (head == NULL)
  {
    printf("List is empty \n");
    return;
  }
  else
  {
    printf("Enter the position to be deleted: \n");
    scanf("%d", &pos);
    if (pos == 0)
    {
      ptr = head;
      head = head->next;
      printf("The item deleted from the list is
              %d", ptr->data);
      free(ptr);
    }
  else
    {
    ptr = head;
      for(i=0; i<pos; i++)
      {
```

```
temp = ptr;
ptr = ptr -> next;
if (ptr == NULL)
{
printf("position not found \n");
return;
}
}
temp -> next = ptr -> next;
printf("The value deleted is - %d", ptr -> data);
free(ptr);
}
}
}
```



```
void main()
{
int choice;
while(1)
{
printf("\n 1. to insert \n 2. to display \n
3. to delete beginning \n 4. to delete at
end \n 5. delete at a given position \n
6. to exit \n");
printf("Enter your choice \n");
scanf("%d", &choice);
switch(choice)
{
case 1:
```

```
        insert();
        break;
    case 2:
        display();
        break;
    case 3:
        delete_begin();
        break;
    case 4:
        delete_end();
        break;
    case 5:
        delete_pos();
        break;
    case 6:
        exit(0);
        break;
    default:
        printf("Enter the correct choice \n");
        break;
    }
  }
}
```

output

1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. delete at position
6. to exit.

PAGE EDGE

DATE: / /

Enter your choice: 1
Enter the value to be inserted 12.
1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. to delete at position
6. to exit.
Enter your choice: 1
Enter the value to be inserted 13.
1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. delete at position
6. to exit.
Enter your choice: 1
Enter the value to be inserted 14.
1. to insert.
2. to display
3. to delete beginning
4. to delete at end
5. delete at position
6. to exit.
Enter your choice: 2.
Elements are: 12, 13, 14, 15.
1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. delete at position
6. exit.

Enter your choice: 3
The element deleted from list - 12
1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. delete at a given ~~given~~ position
6. to exit.
Enter your choice: 4
The element deleted from list - 15.
1. to insert
2. to display
3. to delete beginning
4. to delete at end
5. delete at a given position
6. to exit
Enter your choice: 5
Enter your position: 0
The element deleted from list - 13.

Leetcode 1

```
typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;
}
MinStack;
MinStack* minStackCreate()
{
    MinStack *st=(MinStack*)malloc(sizeof
                    (MinStack));
    if(st==NULL)
    {
        printf("memory allocation failed");
        exit(0);
    }
    st->size=5;
    st->top=-1;
    st->s=(int*)malloc(st->size*sizeof(int));
    st->minstack=(int*)malloc(st->size*
                    sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
    return st;
}
```

16/1/24

```
void minstackPush(MinStack* obj, int val) {
  if (obj->top==obj->size-1)
  {
  printf("stack is overflow");
  }
  else{
  obj->topt++;
  obj->s[obj->top]=val;
  if(obj->top==0 || val<obj->minstack
          [obj->top-1]){
     obj->minstack[obj->top]=val;
     }
  else{
     obj->minstack[obj->top]=obj->minstack
          [obj->top-1];
     }
  }
}
void minStackPop(MinStack* obj)
{
  int value;
  if(obj->top==-1)
  {
  printf("underflow");
  }
  else
  {
  value=obj->s[obj->top];
  obj->top--;
  printf("%d is popped\n", value);
  }
}
```

```
int minStackTop(MinStack* obj)
{
    int value = -1;
    if(obj->top == -1)
    {
        printf("underflow");
        exit(0);
    }
else
    {
    value = obj->s[obj->top];
    return value;
    }
}
int minstackgetmin(MinStack* obj) {
    if(obj->top == -1)
    {
    printf("underflow");
    exit(0);
    }
else
    {
    return obj->minstack[obj->top];
    }
}
void minStackFree(MinStack* obj) {
    free(obj->s);
    free(obj->minStack);
    free(obj);
}
```

output

["MinStack", "push", "push", "push", "getmin", "pop", "top", "getmin"]
[[], [-2], [0], [-3], [], [], [], []]
-3 is poped

output.
[null, null, null, null, -3, null, 0, -2]

expected.
[null, null, null, null, -3, null, 0, -2]

25/1/24 Week-6

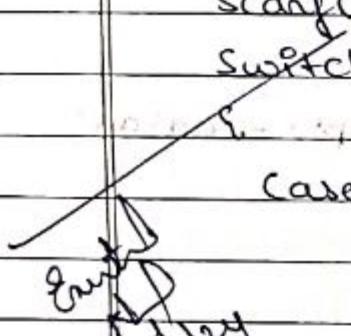
2)a Stack Queue

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *next;
};
struct node *head=NULL;

void enqueue()
{
  struct node *temp, *ptr;
  temp=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter value to be inserted \n");
  scanf("%d",&temp->data);
  temp->next=NULL;
  if(head==NULL)
  {
    head=temp;
    return;
  }
  else
  {
    struct node *ptr;
    ptr=head;
    while(ptr->next!=NULL)
    {
      ptr=ptr->next;
    }
    ptr->next=temp;
```

```
    }
}
void dequeue()
{
  struct node *ptr;
  if (head==NULL)
  {
    printf("queue is empty \n");
    return;
  }
  else
  {
    ptr=head;
    head=ptr->next;
    printf("value dequeued =%d", ptr->data);
    free(ptr);
  }
}
void display()
{
  if(head==NULL)
  {
    printf("queue is empty \n");
    return;
  }
  else
  {
    struct node *ptr;
    ptr=head;
    while(ptr!=NULL)
    {
      printf("%d \n", ptr->data);
```

```
        ptr=ptr->next;
    }
  }
}
void main()
{
  int choice;
  while(1)
  {
    printf("\n 1. to enqueue \n 2. to dequeue \n
          3. display \n 4. exit \n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
          enqueue();
          break;
      case 2:
          dequeue();
          break;
      case 3:
          display();
          break;
      case 4:
          exit(0);
          break;
    }
  }
}
```



output:

PAGE EDGE

DATE: / /

1. to enqueue
2. to dequeue
3. to display
4. exit
Enter your choice: 1
Enter value to be inserted: 12
1. to enqueue
2. to dequeue
3. to display
4. exit
Enter your choice: 1
Enter value to be inserted: 13.
1. to enqueue
2. to dequeue
3. to display
4. exit
Enter your choice: 1, Enter value to be inserted: 14
1. to enqueue
2. to dequeue
3. to display
4. exit
Enter your choice: 3
12 13 14
1. to enqueue
2. to dequeue
3. to display
4. exit
Enter your choice: 2
Entered value dequeue = 12.

Queue stack

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
 int data;
 struct node *next;
};
struct node *head=NULL;
void push()
{
  struct node *temp;
 temp=(struct node*) malloc(sizeof(struct node));
 printf("Enter the data to be pushed \n");
 scanf("%d", &temp->data);
 temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
   return;
 }
 else {
   temp->next=head;
   head=temp;
 }
}
void pop()
{
  struct node *ptr;
  ptr=head;
  if(head==NULL)
```

```
{
printf("Stack is empty item can't be popped \n");
return;
}
else {
ptr = head;
head = ptr->next;
printf("Element popped from the stack is
        - %d", ptr->data);
free(ptr);
}
}
void display()
{
if(head == NULL)
{
printf("Stack is empty \n");
return;
}
else {
struct node *ptr;
ptr = head;
while(ptr != NULL)
{
printf("%d \n", ptr->data);
ptr = ptr->next;
}
}
}
void main()
{
int choice;
```

```
while(1)
{
printf("\n 1. to push \n 2. to pop \n 3. to
      display \n 4. to exit \n");
scanf("%d", &choice);
switch(choice)
{
case 1:
      push();
      break;
case 2:
      pop();
      break;
case 3:
     display();
      break;
case 4:
     exit(0);
      break;
  }
 }
}
```

Output

```
1. to push
2. to pop
3. to display
4. exit
Enter your choice: 1
Enter the data to be inserted: 12.
1. to push
```

2. to pop
3. to display
4. exit
Enter your choice: 1
Enter the data to be inserted: 13.
1. to push
2. to pop
3. to display
4. exit
Enter your choice: 1
Enter the data to be inserted: 14
1. to push
2. to pop
3. to display
4. exit.
Enter your choice: 3.
12 13 14. 14 13 12
1. to push
2. to pop
3. to display
4. exit
Enter your choice: 2.
1. 14

1)
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *next;
} *first=NULL, *second=NULL;

void display(struct node *s)
{
  while(s!=NULL)
  {
    printf("%d \t", s->data);
    s=s->next;
  }
}
void create1(int a[], int n)
{
  struct node *last, *t;
  first=(struct node*)malloc(sizeof(struct nod
  first->data=a[0];
  first->next=NULL;
  last=first;
  for(int i=1; i<n; i++)
  {
    t=(struct node*)malloc(sizeof(struct node)
    t->data=a[i];
    t->next=NULL;
    last->next=t;
    last=t;
  }
}
```

PAGE EDGE

DATE : / /

```
void create2(int a[], int n)
{
  struct node *last, *t;
  second=(struct node*)malloc(sizeof(struct node));
  second->data=a[0];
  second->next=NULL;
  last=second;
  for(int i=1; i<n; i++)
  {
    t=(struct node*)malloc(sizeof(struct node));
    t->data=a[i];
    t->next=NULL;
    last->next=t;
    last=t;
  }
}.
void reverse(struct node *p)
{
  struct node *q, *r;
  p=first;
  q=NULL;
  r=NULL;
  while(p!=NULL)
  {
    r=q;
    q=p;
    p=p->next;
    q->next=r;
  }
  first=q;
}
Struct node* concatat(struct node *p, struct
```

```
    node *q)
{
  struct node *x;
  if(p==NULL)
  {
    p=q;
    return p;
  }
  if(q==NULL)
  {
    return q;
  }
  x=p;
  while(x->next!=NULL)
    x=x->next;
    x->next=q;
    return p;
}
void sort(struct node *p)
{
  struct node *i, *j;
  int temp;
  for(i=p; i->next!=NULL; i=i->next)
  {
    if(i->data>j->data)
    {
      temp=i->data;
      i->data=j->data;
      j->data=temp;
    }
  }
}
```

```
}
void main()
{
  int a[10], b[10], n1, n2;
  printf("enter n:");
  scanf("%d", &n1);
  printf("enter the values:");
  for(int i=0; i<n1; i++)
  {
    scanf("%d", &a[i]);
  }
  struct node *s;
 s=(struct node*)malloc(sizeof(struct node));
  create1(a, n1);
  sort(first);
  printf("sorted list");
  display(first);
  reverse(first);
  printf("\n reversed list");
  display(first);
  printf("\n enter n:");
  scanf("%d", &n2);
  printf("enter the values");
  for(int i=0; i<n2; i++)
  {
    scanf("%d", &b[i]);
  }
  create2(b, n2);
  display(second); printf("concat:");
  s=concatat(first, second);
  display(s);
}
```

Output

```
Enter n: 3
Enter the values:
7 3 8
Sorted list:
3 7 8
Reversed list:
8 7 3
enter n: 3
Enter the values:
6 5 3

6 5 3
Concat: 8 7 3 6 5 3
```

Entered Queue
Stack, Queue using SLL
Reverse, concatenation
SLL Reverse sort

25/11/24

PAGE EDGE

DATE : / /

1/02/24

Week-7

1) Doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
Struct node
{
  int data;
  Struct node *prev;
  Struct node *next;
};
Struct node *S1=NULL;

Struct node *insert_begin(struct node *start)
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct
                    node));
   printf("Enter the value to be inserted \n");
   scanf("%d", &temp->data);
   temp->next=NULL;
   temp-> prev = NULL;
   if(start == NULL)
   {
      start = temp;
   }
   else
   {
     temp -> next = start;
     start -> prev = temp;
     Start = temp;
```

```
    }
    return start;
}

Struct node *insert_end(struct node *start)
{
    Struct node *temp;
    temp=(struct node*)malloc(sizeof(struct
                    node));
    printf("Enter the value to be inserted \n");
    scanf("%d",&temp->data);
    temp->next=NULL;
    temp->prev=NULL;
    if(start==NULL)
    {
        start=temp;
    }
    else
    {
        struct node *ptr;
        ptr=start;
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
        temp->prev=ptr;
    }
    return start;
}
```

```
struct node* delvalue (struct node *start,
        int val)
{
  struct node *ptr=start;
  int value=val;
  while (ptr!=NULL)
  {
    if(ptr->data==value)
    {
      if(ptr->prev!=NULL)
      {
        ptr->prev->next=ptr->next;
      }
    if(ptr->next!=NULL)
      {
        ptr->next->prev=ptr->prev;
      }
    if(ptr==start)
    {
      start=ptr->next;
      }
    free(ptr);
    printf("value %d deleted \n", value);
    return start;
    }
    ptr=ptr->next;
  }
printf("value %d not found \n", value);
return start;
}.
```

```
void display(struct node *start)
{
  struct node *ptr = start;
  if(start == NULL)
  {
    printf("\n List is empty \n");
  }
  else
  {
    while(ptr != NULL)
    {
      printf("%d \n", ptr->data);
      ptr = ptr->next;
    }
  }
}
int main()
{
  int choice;
  while(1)
  {
    printf("\n 1. to add in beginning \n
          2. to add at end \n
          3. to display \n 4. to delete \n
          5. exit \n");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
        s1 = insert_begin(s1);
        break;
      case 2:
```

```
        sl=Insert-end(sl);
        break;
    case 3:
        display(sl);
        break;
    case 4:
        printf("Enter the value to delete:");
        int val;
        scanf("%d",&val);
        sl=delevalue(sl,val);
        break;
    case 5:
        exit(0);
    default:
        printf("wrong choice");
    }
}
}
```

PAGE EDGE

DATE: / /

Leetcode 2.

```
struct ListNode* reverseBetween(struct
listnode* ~~bed~~ head, int left, int right).
{
    if(head == NULL || left == right)
    {
        return head;
    }

struct listnode* dummy=(struct listnode*)
    malloc sizeof(struct listnode));
dummy->next=head;
struct listnode* prev=dummy;

for(int i=1; i<left; ++i)
 {
    prev=prev->next;
 }

struct listnode* current = prev->next;
struct listnode* next =NULL;
struct listnode* tail = current;

for(int i= left; i<=right; ++i)
{
struct listnode* temp=current->next;
current->next=next;
next = current;
current= temp;
}
```

```
prev->next = next;
tail->next = current;

struct ListNode* result = dummy->next;
free(dummy);
return result;
}
```

output

head = [1,2,3,4,5]
left = 2
right = 4
output = [1,4,3,2,5]

PAGE EDGE

DATE: / /

15/02/24

Week-8

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int value;
  struct node *left;
  struct node *right;
}node;

struct node *create()
{
  struct node *temp;
  temp=(struct node*) malloc(sizeof(struct
                    node));
  printf("Enter data:");
  scanf("%d", &temp->value);
  temp->left=temp->right=NULL;
  return temp;
}
void insert(struct node *root, struct node
     *temp)
{
  if(temp->value<root->value)
  {
    if(root->left!=NULL)
        insert(root->left, temp);
    else
        root->left=temp;
  }
  if(temp->value>root->value)
```

```
{
    if (root->right != NULL)
        insert(root->right, temp);
    else
        root->right = temp;
}
}
void inorder(struct node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        printf("%d \t", root->value);
        inorder(root->right);
    }
}
void postorder(struct node *root)
{
    if(root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d \t", root->value);
    }
}
void preorder(struct node *root)
{
    if(root != NULL)
    {
        printf("%d \t", root->value);
        preorder(root->left);
        preorder(root->right);
```

```
-}
}
int main()
{
  int choice;
  struct node *root=NULL;
  while(1)
  {
    printf("\n 1.Insert the element \n
          2.Inorder traversal \n 3.Preorder
          traversal \n 4.Post order traversal
          \n 5.exit");
    printf("Enter choice \n");
    scanf("%d", &choice);
    switch(choice)
    {
    case 1:
      if(root==NULL)
        root=create();
      else
        insert(root, create());
      break;
    case 2:
      printf("\n Inorder traversal \n");
      inorder(root);
      break;
    case 3:
      printf("\n Preorder traversal \n");
      preorder(root);
      break;
    case 4:
      printf("\n Postorder traversal \n");
```

```
        postorder(root);
        break;
    case 5:
        exit(0);
    default:
        printf("Invalid choice");
        break;
    }
}

return 0;
}
```

output

```
1. Insert the element
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. exit
Enter choice : 1
Enter data : 50

Enter choice : 1
Enter data : 60

Enter choice : 1
Enter data : 70

Enter choice : 1
Enter data : 90

Enter choice : 1
```

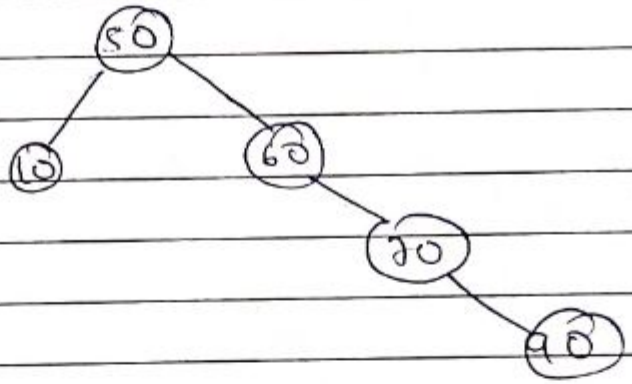
15/2/24

Enter data: 10

Enter choice: 2.
Inorder traversal: 10, 50, 60, 70, 90
Preorder traversal: 50.

Enter choice: 3
Preorder traversal: 50, 10, 60, 70, 90.

Enter choice: 4
Postorder traversal: 10, 90, 70, 60, 50.



Inorder.

10 → 50 → 60 → 70 → 90.

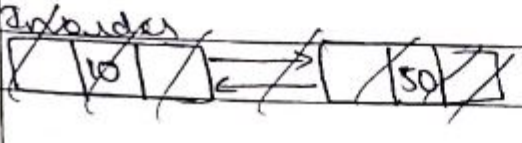
Preorder.

50 → 10 → 60 → 70 → 90.

Postorder

10 → 90 → 70 → 60 → 50.

Inorder



22/2/24

Week-9

BFS

```
#include<stdio.h>
#define MAX_VERTICES 10
int n, i, f, visited[MAX_VERTICES],
 queue[MAX_VERTICES], front=0, rear=0;
int adj[MAX_VERTICES][MAX_VERTICES];

void bfs(int v)
{
   visited[v]=1;
   queue[rear++]=v;
   while(front<rear)
      int current=queue[front++];
      printf("%d\t", current);
      for(int i=0; i<n; i++)
         {
            if(adj[current][i] && !visited[i])
               {
                  visited[i]=1;
                  queue[rear++]=i;
               }
         }
}
int main()
{
   int v;
   printf("enter the number of vertices:");
   scanf("%d", &n);
```

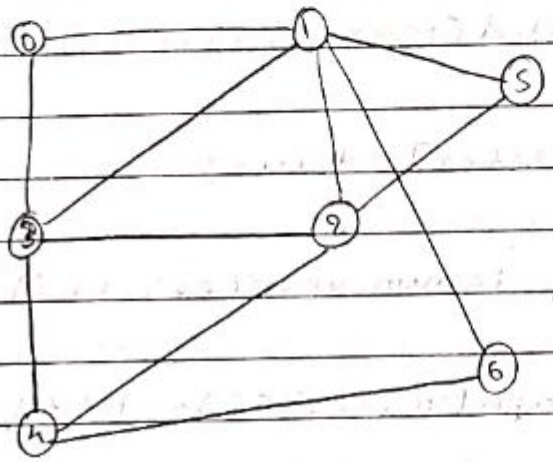
```
for(i=0; i<n; i++)
{
    visited[i]=0;
}
printf("Enter graph data in matrix
        form:\n");
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
        scanf("%d", &adj[i][j]);

printf("Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
for(i=0; i<n; i++)
{
    if(!visited[i])
    {
        printf("\n BFS is not possible.
                Not all nodes are
                reachable.\n");
        return 0;
    }
}
return 0;
}
```

Output

```
Enter the number of vertices: 7
Enter graph data in matrix form:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
```

0 1 0 1 1 1 0.
1 1 1 0 1 0 0.
0 0 1 1 0 0 1
0 1 1 0 0 0 0.
0 1 0 0 1 0 0.
Enter the starting vertex: 0.
0, 1, 3, 2, 5, 6, 4.



DFS

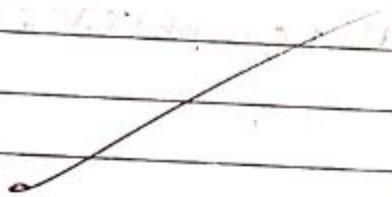
```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 10

void dfs(int graph[MAX_VERTICES]
        [MAX_VERTICES], int num_vertices,
    bool visited[MAX_VERTICES], int vertex)
{
    visited[vertex] = true;
    int i;
    for(i=0; i<num_vertices; ++i)
    {
        if(graph[vertex][i]==1 && !visited[i])
        {
            dfs(graph, num_vertices, visited, i);
        }
    }
}
bool is_connected(int graph[MAX_VERTICES]
        [MAX_VERTICES], int num_vertices)
{
    bot bool visited[MAX_VERTICES]={false};
    dfs(graph, num_vertices, visited, 0);
    int i;
    for(i=0; i<num_vertices; ++i)
    {
        if(!visited[i])
        {
            return false;
        }
```

```
    }
    return true;
}
int main()
{
    int num_vertices;
    printf("Enter the number of vertices:");
    scanf("%d", &num_vertices);
    int i, j;
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<num_vertices; ++i)
    {
        for(j=0; j<num_vertices; ++j)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    if(is_connected(graph, num_vertices))
    {
        printf("The graph is connected.\n");
    }
    else {
        printf("The graph is not connected.\n");
    }
    return 0;
}
```

output

Enter the number of vertices: 7
Enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
The graph is connected.



Leet code - 3 (Split Linked List)

```
typedef struct ListNode lnode;
int get_len(lnode *head)
{
    int n=0;
    while(head)
    {
        n++;
        head=head->next;
    }
    return n;
}
Struct ListNode** splitListToParts
  (struct ListNode* head, int k, int*
                returnSize)
{
   int n= get_len(head), elems, i, j;
   *returnSize=k;
   lnode **list=(lnode**) calloc(k, sizeof(lnode*)),
              *t=head;
   if(n>k)
   {
      for(i=0; i<k; i++)
      {
         elems= i<n%k?n/k+1:n/k;
         j=0;
         list[i]=head;
         t=head;
         while(j++<elems)
         {
            t=head;
```

```
        head=head->next;
        }
        t->next=NULL;
      }
    }
  else
  {
    for(i=0;i<n;i++)
    {
      list[i]=head;
      head=head->next;
      list[i]->next=NULL;
    }
  }
  return list;
  }
```

Output

case 1:

head = [1, 2, 3]

k = 5

case 2:

head = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k = 3

Sol:

22/2/24.

Leetcode-61. (Rotate List).

```
int getLength(struct ListNode* head)
{
    if(head==NULL)
        return 0;
    return 1+getLength(head->next);
}
struct ListNode* rotateRight(struct
        ListNode* head, int k)
{
    if(head==NULL || k==0)
    return head;
int length = getLength(head);
if(length==1)
    return head;
for(int i=0; i<k%length; i++)
{
    struct ListNode *p=head;
    while(p->next->next!=NULL)
    {
        p=p->next;
    }
struct ListNode *a=(struct ListNode*)
                malloc(sizeof(struct ListNode));
a->val=p->next->val;
a->next=head;
head=a;
p->next=NULL;
}
return head;
}
```

output

case 1:

head = [1, 2, 3, 4, 5]

k = 2.

[4, 5, 1, 2, 3]

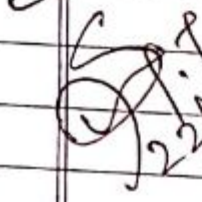
case 2:

head = [0, 1, 2]

k = 4

output

~~[0, 0, 1: [2, 0, 1].~~



29/02/24

Week-10

Hashing

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_EMPLOYEES 10
#define HASH_TABLE_SIZE 10
struct Employee
{
    int key;
};
int hashFunction(int key);
void insertEmployee(struct Employee employees[],
        int hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main()
{
  struct Employee employees[MAX_EMPLOYEES];
  int hashTable[HASH_TABLE_SIZE]={0};
  int n, m, i;
  printf("Enter the number of employees:");
  scanf("%d",&n);
  printf("Enter employee records:\n");
  for(i=0; i<n; ++i)
  {
     printf("Employee %d:\n",i+1);
     printf("Enter 4-digit key:");
     scanf("%d",&employee[i].key);
   insertEmployee(employees, hashTable,
              employees[i]);
```

```
    }
    printf("\n Hash Table:\n");
    displayHashTable(hashTable);
    return 0;
}
int hashFunction(int key)
{
    return key % HASH_TABLE_SIZE;
}
void insertEmployee(struct Employee employees[],
    int hashTable[], struct Employee emp)
{
    int index = hashFunction(emp.key);
    while(hashTable[index] != 0)
    {
        index = (index+1) % HASH_TABLE_SIZE;
    }
    hashTable[index] = emp.key;
}
void displayHashTable(int hashTable[])
{
    int i;
    for(i=0; i<HASH_TABLE_SIZE; ++i)
    {
        printf("%d->", i);
        if(hashTable[i] == 0)
        {
            printf("Empty\n");
        }
        else {
            printf("%d\n", hashTable[i]);
        }
```

}

}

output

Enter the number of employees: 10.
Enter the employee records:
Employee 1:
Enter 4-digit key: 1234
Employee 2:
Enter 4-digit key: 3456
Employee 3:
Enter 4-digit key: 2635
Employee 4:
Enter 4-digit key: 2789
Employee 5:
Enter 4-digit key: 1254
Employee 6:
Enter 4 digit key: 8970
Employee 7:
Enter 4 digit key: 4578
Employee 8:
Enter 4 digit key: 3241
Employee 9:
Enter 4 digit key: 9875
Employee 10:
Enter 4 digit key: 2456.

Hash Table:

0 → 8970
1 → 3241
2 → 9875
3 → 2456
4 → 1234
5 → 2675
6 → 3456
7 → 1254
8 → 4578
9 → 2789

29/2/24

HackerRank

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int id;
  int depth;
  struct node *left, *right;
};
void inorder(struct node* tree)
{
   if(tree==NULL)
      return;
   inorder(tree->left);
   printf("%d", tree->id);
   inorder((tree->right));
   }
int main(void)
{
   int no_of_nodes, i=0;
   int l, r, max_depth, t;
   struct node* temp=NULL;
   scanf("%d", &no_of_nodes);
   struct node* tree=(struct node*) calloc
         (no_of_nodes, sizeof(struct node));
   tree[0].depth=1;
   while(i< no_of_nodes)
   {
      tree[i].id=i+1;
      scanf("%d %d", &l, &r);
      if(l==-1)
```

```
        tree[i].left = NULL;
    else
      {
        tree[i].left = &tree[i-1];
        tree[i].left->depth = tree[i].depth + 1;
        max_depth = tree[i].left->depth;
      }
    if(r==-1)
      tree[i].right = NULL;
    else
      {
        tree[i].right = &tree[r-1];
        tree[i].right->depth = tree[i].depth + 1;
        max_depth = tree[i].right->depth;
      }
    p++;
  }
scanf("%d", &p);
while(i--)
{
  scanf("%d", &l);
  r = 1;
  while(l <= max_depth)
  {
    for(k=0; k<no_of_nodes; ++k)
    {
      if(tree[k].depth == l)
      {
        temp = tree[k].left;
        tree[k].left = tree[k].right;
        tree[k].right = temp;
      }
```

```
    }
    l = l + r;
    }
    inorder(tree);
    printf("\n");
}
```

output:

Test case 0.

Input

```
3
2 3
-1 -1
-1 -1
2
1
1
```

output

```
3 1 2
2 1 3.
```

Test case 1

Input.

```
17
2 3
4 5
6 -1
-1 7
8 9
10 11
12 13
-1 14
-1 -1
15 -1
16 17
-1 -1
-1 -1
-1 -1
-1 -1
-1 -1
2
```

2f-t-uncated-f

output: 14 8 5 9 2 4 13 7 12 1 3 10 15 6
9 5 14 8 2 1 3 7 12 4 1 3 17 11

29/2/24