**SINGLE LINKED LIST**

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

struct Node

{

int val;

Node \*next;

};

Node \*Head=NULL;

void insert\_At\_Begin(int val)

{

Node \*new\_node = new Node();

new\_node->val=val;

new\_node->next=NULL;

new\_node->next=Head;

Head=new\_node;

}

void insert\_At\_Pos(int val, int pos)

{

if(pos==0)

{

insert\_At\_Begin(val);

}

else

{

Node \*new\_node = new Node();

new\_node->val=val;

new\_node->next=NULL;

Node \*temp1=Head;

for(int i=0; i<pos-1; i++)

{

temp1=temp1->next;

}

Node \*temp2=temp1->next;

temp1->next=new\_node;

new\_node->next=temp2;

}

}

void Reverse\_Linked\_List()

{

Node \*current=Head;

Node \*next=NULL;

Node \*prev=NULL;

while(current!=NULL)

{

next=current->next;

current->next=prev;

prev=current;

current=next;

}

Head=prev;

}

void Display()

{

Node \*temp=Head;

while(temp!=NULL)

{

cout<<temp->val<<" ";

temp=temp->next;

}

cout<<endl;

}

int main()

{

insert\_At\_Begin(5);

Display();

insert\_At\_Begin(6);

Display();

insert\_At\_Pos(7,1);

Display();

insert\_At\_Pos(10,2);

Display();

Reverse\_Linked\_List();

Display();

return 0;

}

**DOUBLE LINK LIST**

struct Node{

int val;

Node \*prev;

Node \*next;

};

Node \*Head=NULL;

Node \*Tail=NULL;

Node\* GetNewNode(int val)

{

Node \*temp = new Node();

temp->val=val;

temp->prev=NULL;

temp->next=NULL;

return temp;

}

void InsertNode(int val,int pos)

{

Node \*New\_Node = GetNewNode(val);

if(Head==NULL)

{

Head=New\_Node;

return;

}

else

{

if(pos==0)

{

Node \*temp;

temp=Head;

Head->prev=New\_Node;

New\_Node->next=temp;

Head=New\_Node;

}

else

{

Node \*temp1=Head,\*temp2;

for(int i=1;i<=pos-1;i++)

{

temp1=temp1->next;

}

temp2=temp1->next;

temp1->next=New\_Node;

New\_Node->prev=temp1;

temp2->prev=New\_Node;

New\_Node->next=temp2;

}

}

}

void Display()

{

Node \*temp=Head;

while(temp!=NULL)

{

cout<<temp->val<<" ";

temp=temp->next;

}

cout<<endl;

}

int main()

{

InsertNode(5,0);

InsertNode(25,0);

InsertNode(35,0);

InsertNode(51,1);

InsertNode(55,2);

Display();

return 0;

}

**HEAP**

void Heapify(int ara[], int n, int parent)

{

int left=parent\*2;

int right=(parent\*2)+1;

int largest=parent;

if(left<=n && ara[left]>ara[largest]) // (left<=n) ei shorto vule jabo… tai likhe rakha uchit

{

largest=left;

}

if(right<=n && ara[right]>ara[largest])

{

largest=right;

}

if(parent!=largest)

{

swap(ara[parent],ara[largest]);

Heapify(ara, n, largest);

}

}

void Build\_MaxHeap(int ara[],int n)

{

for(int i=n/2;i>=1;i--)

{

Heapify(ara, n, i);

}

}

void HeapSort(int ara[], int n)

{

Build\_MaxHeap(ara,n);

for(int i=n;i>1;i--)

{

swap(ara[i],ara[1]);

Heapify(ara, i-1, 1);

}

}

int main()

{

int ara[]={0,10,3,5,1,2,11,13};

HeapSort(ara,7);

for(int i=1;i<=7;i++)

{

cout<<ara[i]<<” ”;

}

return 0;

}

**Binary Search**

int Binary\_Search(int ara[], int n, int x)

{

int low=0;

int high=n-1;

while(low<high)

{

int mid=(low+high)/2;

if(ara[mid]==x)

{

return mid;

}

else if(x>ara[mid])

{

low=mid+1;

}

else if(x<ara[mid])

{

high=mid;

}

}

return -1;

}

Single Linked List:

struct Node{

int val;

Node \*next;

};

Node \*Head=NULL;

void Insert\_Node\_At\_begin(int val)

{

Node \*temp = new Node();

temp->val=val;

temp->next=NULL;

temp->next=Head;

Head=temp;

return;

}

void Insert\_At\_Position(int val, int pos)

{

if(pos==0)

{

void Insert\_Node\_At\_begin(val);

}

else

{

Node \*new\_node=new Node();

new\_node>val=val;

new\_node>next=NULL;

Node \*temp1=Head,\*temp2;

for(int i=1;i<=pos-1;i++)

{

temp1=temp1->next;

}

temp2=temp1->next;

temp1->next=new\_node;

new\_node->next=temp1;

}

}

Double Linked List:

struct Node{

int val;

Node \*next;

Node \*prev;

};

Node \*Head=NULL;

Node \*Tail=NULL;

Node\* getNewNode(int val)

{

Node \*new\_node = new Node();

new\_node->val=val;

new\_node->next=NULL;

new\_node->prev=NULL;

}

void Insert\_at(int val, int pos)

{

Node \*new\_node=getNewNode(val);

if(pos==0)

{

Head=new\_node;

}

else

{

Node \*temp1=Head;

Node \*temp2;

for(int i=1;i<=pos-1;i++)

{

temp1=temp1->next;

}

temp2=temp1->next;

temp1->next=new\_node;

new\_node->prev=temp1;

temp2->prev=new\_node;

new\_node->next=temp2;

}

}

Binary Search

int Binary\_Search(int ara[],int n,int x)

{

int low=0;

int high=n-1;

while(low<high)

{

int mid=(low+high)/2;

if(ara[mid]==x)

{

return mid;

}

else if(x>ara[mid])

{

low=mid+1;

}

else

{

high=mid;

}

}

return -1;

}

Heap

void MaxHeapify(int ara[],int n, int parent)

{

int left=parent\*2;

int right=parent\*2+1;

int largest=parent;

if(left<=n && ara[left]>ara[largest])

{

largest=left;

}

if(right<=n && ara[right]>ara[largest])

{

largest=right;

}

if(parent!=largest)

{

swap(ara[parent],ara[largest]);

MaxHeapify(ara,n, largest);

}

}

void Build\_Max\_Heap(int ara[],int n)

{

for(int i=n/2;i>=1;i--)

{

MaxHeapify(ara,n, i);

}

}

void HeapSort(int ara[],int n)

{

Build\_Max\_Heap(ara,n);

for(int i=n;i>=1;i--)

{

swap(ara[i],ara[1]);

MaxHeapify(ara,i-1, 1);

}

}

BFS

#include<stdio.h>

#include<bits/stdc++.h>

using namespace std;

int color[100];

int parent[100];

map<int,vector<int> > Map;

void BFS(int start)

{

memset(color,-1,sizeof(color));

parent[start]=0;

color[start]=1;

queue<int>q;

q.push(start);

while(!q.empty())

{

int current=q.front();

q.pop();

for(\_\_typeof(Map[current].begin())it=Map[current].begin(); it!=Map[current].end(); it++)

{

if(color[\*it]==-1)

{

parent[\*it]=current;

color[\*it]=color[current]+1;

q.push(\*it);

}

}

}

return;

}

void PrintPath(int x)

{

if(parent[x]==0)

{

cout<<x<<" ";

return;

}

PrintPath(parent[x]);

cout<<x<<" ";

}

int main()

{

int edge;

cin>>edge;

for(int i=1; i<=edge; i++)

{

int a,b;

cin>>a>>b;

Map[a].push\_back(b);

Map[b].push\_back(a);

}

BFS(1);

cout<<endl;

PrintPath(5);

}

BCD

int BCD(int a, int b)

{

if(a%b==0)

{

return b;

}

return BCD(b,a/b);

}

Fibonacci

int Fibo(int n)

{

if(n==0)

{

return 0;

}

if(n==1)

{

return 1;

}

return Fibo(n-1)+Fibo(n-2);

}

Factorial

int Fact(int n)

{

if(n==1)

{

return 1;

}

return n\*Fact(n-1);

}