EXPERIMENT :1 DATE:23-11-21

AIM: WRITE A PROGRAM TO MERGE AND SORT TWO ARRAY

PROGRAM:

#include<stdio.h>

void main()

{

int i,n,m,j,k,a[10],a2[20],temp=0,s,a3[20],index=0;

printf("enter the number of values:");

scanf("%d",&n);

printf("\nEnter the values of first array:\n");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

printf("The first sorted array is");

for(i=0;i<n;i++)

{

for(j=i+1;j<n;j++)

{

if(a[i]>a[j])

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

for(i=0;i<n;i++)

{

printf("%3d",a[i]);

}

temp=0;

printf("\nenter the number of values:");

scanf("%d",&m);

printf("\nenter the values of second array:");

for(i=0;i<m;i++)

{

scanf("%d",&a2[i]);

}

printf("the second sorted array is:\n");

for(i=0;i<m;i++)

{

for(j=i+1;j<m;j++)

{

if(a2[i]>a2[j])

{

temp=a2[i];

a2[i]=a2[j];

a2[j]=temp;

}

}

}

for(i=0;i<m;i++)

{

printf("%3d",a2[i]);

s=m+n;

}

printf("\nThe output of the two merged sorted array is:\n");

for(i=0;i<n;i++)

{

a3[i]=a[i];

}

k=n;

for(i=0;i<m;i++)

{

a3[k]=a2[i];

k++;

}

for(i=0;i<s;i++)

{

for(j=i+1;j<s;j++)

{

if(a3[i]>a3[j])

{

temp=a3[i];

a3[i]=a3[j];

a3[j]=temp;

}

}

}

for(i=0;i<s;i++)

{

printf("%3d",a3[i]);

}

}

Output:

enter the number of values:5

Enter the values of first array:

5

3

1

2

4

The first sorted array is 1 2 3 4 5

enter the number of values:5

enter the values of second array:9

6

0

7

8

the second sorted array is:

0 6 7 8 9

The output of the two merged sorted array is:

0 1 2 3 4 5 6 7 8 9

EXPERIMENT:2 DATE:29-11-21

AIM:Program to implement stack using linked list

#include<stdio.h>

#include<stdlib.h>

void push();

void display();

void pop();

void search();

struct node

{

int data;

struct node\*next;

};

struct node\*head;

void main()

{

int ch=0;

// printf("\n\*\*\*\*\*\*\*\*\*\*\*///stack using linked\*\*\*\*\*\*\*\*\*\*\*\*\\\\\ \n");

while(ch!=5)

{

printf("\n please select a choice from below?????????\n");

printf("\n 1.push \n 2. display\n 3. pop \n 4.search\n");

printf("\nEnter the choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

push();

break;

}

case 2:

{

display();

break;

}

case 3:

{

pop();

break;

}

case 4:

{

search();

break;

}

default:

{

printf("enter a valid choice");

break;

}

};

}

}

void push()

{

int e;

struct node \*ptr=(struct node\*)malloc(sizeof(struct node));

if(ptr==NULL)

{

printf("the value cannot be inserted");

}

else

{

printf("enter the value to be inserted");

scanf("%d",&e);

}

if(head==NULL)

{

ptr->data=e;

ptr->next=NULL;

head=ptr;

}

else

{

ptr->data=e;

ptr->next=head;

head=ptr;

}

printf("item pushed");

}

void display()

{

struct node \*ptr;

ptr=head;

if(ptr==NULL)

{

printf("the stack is empty");

}

else

{

printf("the stack is");

while(ptr!=NULL)

{

printf("%d-->",ptr->data);

ptr=ptr->next;

}

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->data;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

}

}

void search()

{

struct node\*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr==NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data==item)

{

printf("\n item found at location %d\n",i+1);

flag=0;

}

i++;

ptr = ptr->next;

}

if(flag==1)

{

printf("\nItem not found\n");

}

}

}

OUTPUT:

Enter the choice : 1

enter the value to be inserted4

item pushed

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 1

enter the value to be inserted6

item pushed

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 1

enter the value to be inserted7

item pushed

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 2

the stack is7-->6-->4-->

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 3

Item popped

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 2

the stack is6-->4-->

please select a choice from below?????????

1.push

2. display

3. pop

4.search

Enter the choice : 4

Enter item which you want to search?

6

item found at location 1

please select a choice from below?????????

1.push

2. display

3. pop

4.search

EXPERIMENT:3 DATE:30-11-21

WRITE A PROGRAM TO IMPLEMENT DOUBLE LINKED LIST

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node\*prev;

int data;

struct node\*next;

};

struct node\*head;

void insertatbegining();

void insertatend();

void insertatspecific();

void display();

void deleteatbegining();

void deleteatend();

void deleteatspecific();

void search();

void main()

{

int choice;

while(choice!=9)

{

printf("\n\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*\n");

printf("choose any of the option given below!!!!!!!!\n");

printf("\n1.insert at begining \n2. insert at end \n3.insert at specific position\n4.display the list \n5.delete from begining \n6.delete from end \n7.delete from specific position \n8.serach \n9.exit\n ");

printf("which one you choose????");

scanf("%d",&choice);

switch(choice)

{

case 1:

insertatbegining();

break;

case 2:

insertatend();

break;

case 3:

insertatspecific();

break;

case 4:

display();

break;

case 5:

deleteatbegining();

break;

case 6:

deleteatend();

break;

case 7:

deleteatspecific();

break;

case 8:

search();

break;

case 9:

exit(0);

break;

default:

printf("please enter a valid choice");

}

}

}

void insertatbegining()

{

int item;

struct node\*ptr;

ptr=(struct node\*)malloc(sizeof(struct node));

printf("enter the item to be inserted");

scanf("%d",&item);

if(head==NULL)

{

ptr->prev=NULL;

ptr->data=item;

ptr->next=NULL;

head=ptr;

}

else

{

ptr->prev=NULL;

ptr->data=item;

ptr->next=head;

head->prev=ptr;

head=ptr;

}

}

void insertatend()

{

int item;

struct node\*ptr,\*temp;

ptr=(struct node\*)malloc(sizeof(struct node));

printf("enter the item to be inserted");

scanf("%d",&item);

ptr->data=item;

if(head==NULL)

{

ptr->prev=NULL;

ptr->next=NULL;

head=ptr;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=ptr;

ptr->prev=temp;

ptr->next=NULL;

}

}

void insertatspecific()

{

int item,loc,i;

struct node\*ptr,\*temp;

ptr=(struct node\*)malloc(sizeof(struct node));

printf("enter the value to be inserted");

scanf("%d",&item);

printf("enter the position to be inserted");

scanf("%d",&loc);

ptr->data=item;

temp=head;

for(i=0;i<loc;i++)

{

temp=temp->next;

}

if(temp==NULL)

{

printf("there are less element than %d",loc);

}

else

{

ptr->next=temp->next;

ptr->prev=temp;

temp->next=ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

}

}

void display()

{

struct node\*temp;

temp=head;

while(temp!=NULL)

{

printf("%d-->",temp->data);

temp=temp->next;

}

}

void deleteatbegining()

{

struct node\*temp;

temp=head;

if(head==NULL)

{

printf("there is no element to delete");

}

else if(temp->next==NULL)

{

head=NULL;

free(temp);

}

else

{

temp->next->prev=NULL;

head=temp->next;

free(temp);

}

}

void deleteatend()

{

struct node \*temp;

temp=head;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

temp = head;

if(temp->next != NULL)

{

temp = temp -> next;

}

temp -> prev -> next = NULL;

free(temp);

printf("\nnode deleted\n");

}

}

void deleteatspecific()

{

struct node \*temp2, \*temp;

int val;

printf("\n Enter the position: ");

scanf("%d", &val);

temp = head;

while(temp -> data != val)

temp = temp -> next;

if(temp -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(temp -> next -> next == NULL)

{

temp ->next = NULL;

}

else

{

temp2 = temp -> next;

temp -> next = temp2 -> next;

temp2 -> next -> prev = temp;

free(temp2);

printf("\nnode deleted\n");

}

}

void search()

{

struct node \*temp;

int item,i=0,flag;

temp = head;

if(temp == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (temp!=NULL)

{

if(temp->data == item)

{

printf("\nitem found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

temp = temp -> next;

}

if(flag==1)

{

printf("\nItem not found\n");

}

}

}

OUTPUT:

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????1

enter the item to be inserted2

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????1

enter the item to be inserted3

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????2

enter the item to be inserted8

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????3

enter the value to be inserted0

enter the position to be inserted2

node inserted

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????4

3-->2-->8-->0-->

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????5

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????6

node deleted

\*\*\*\*\*\*\*\*\*main menu\*\*\*\*\*\*\*\*

choose any of the option given below!!!!!!!!

1.insert at begining

2. insert at end

3.insert at specific position

4.display the list

5.delete from begining

6.delete from end

7.delete from specific position

8.serach

9.exit

which one you choose????8

Enter item which you want to search?

2

item found at location 1

EXPERIMENT:4 DATE:14-12-21

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

void enqueue();

void display();

void dequeue();

void peek();

struct node

{

int data;

struct node \*next;

};

struct node \*front=-1;

struct node \*rear=-1;

void main()

{

int ch=0;

printf("\n\*\*\*\*\*\*\*\*\*\*\*///stack using linked\*\*\*\*\*\*\*\*\*\*\*\*\\\\\ \n");

while(ch!=6)

{

printf("\n please select a choice from below?????????\n");

printf("\n 1.enqueue \n 2. display\n 3. dequeue \n 4.peek \n5.exit \n");

printf("\nEnter the choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

{

enqueue();

break;

}

case 2:

{

display();

break;

}

case 3:

{

dequeue();

break;

}

case 4:

{

peek();

break;

}

case 5:

{

printf("exiting...");

exit(0);

break;

}

default:

{

printf("enter a valid choice");

break;

}

};

}

}

void enqueue()

{

int item;

struct node \*newnode;

newnode=(struct node \*)malloc(sizeof(struct node));

printf("enter the element to be inserted");

scanf("%d",&item);

newnode->data=item;

newnode->next=0;

if(rear==-1)

{

front=rear=newnode;

rear->next=front;

}

else

{

rear->next=newnode;

rear=newnode;

rear->next=front;

}

}

void dequeue()

{

struct node \*temp;

temp=front;

if((front==-1)&&(rear==-1))

{

printf("\nQueue is empty");

}

else if(front==rear)

{

front=rear=-1;

free(temp);

}

else

{

front=front->next;

rear->next=front;

free(temp);

}

}

void peek()

{

if((front==-1) &&(rear==-1))

{

printf("\nQueue is empty");

}

else

{

printf("\nThe front element is %d", front->data);

}

}

void display()

{

struct node \*temp;

temp=front;

printf("\n The elements in a Queue are : ");

if((front==-1) && (rear==-1))

{

printf("Queue is empty");

}

else

{

while(temp->next!=front)

{

printf("%d->", temp->data);

temp=temp->next;

}

printf("%d", temp->data);

}

}

OUTPUT:

\*\*\*\*\*\*\*\*\*\*\*///stack using linked\*\*\*\*\*\*\*\*\*\*\*\*\\

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 1

enter the element to be inserted1

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 1

enter the element to be inserted2

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 1

enter the element to be inserted3

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 2

The elements in a Queue are : 1->2->3

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 3

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 2

The elements in a Queue are : 2->3

please select a choice from below?????????

1.enqueue

2. display

3. dequeue

4.peek

5.exit

Enter the choice : 4

The front element is 2

EXPERIMENT:5 DATE:21-12-21

AIM: WRITE A PROGRAM TO IMPLEMENT BINARY SEARCH TREE

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

struct node\* insertion(struct node\*,int);

struct node\* deleteItem(struct node\*, int);

void inorder(struct node \*);

void inorder(struct node \*);

void inorder(struct node \*);

struct node\* minValueNode(struct node\*);

struct node\* deleteNode(struct node\*, int);

struct node{

int data;

struct node \*left;

struct node \*right;

};

struct node\* root= NULL;

struct node\* createNode(int data){

struct node\* temp= (struct node\*) malloc(sizeof(struct node));

temp->left= temp->right=NULL;

temp->data= data;

return temp;

}

void main(){

int size,choice;

printf("BST Operations\n");

while(1){

printf("\n\n");

printf("1. Insertion\n2. Deletion\n3. Traversal\n4. Search\n5. Exit\n\n");

printf("Enter the option number to perform operation : ");

scanf("%d",&choice);

fflush(stdin);

switch(choice){

case 1: {

int keyval;

printf("Enter the value to insert into the BST : ");

scanf("%d",&keyval);

if(root==NULL){

root= insertion(root, keyval);

}

else{

insertion(root,keyval);

}

break;

}

case 2: {

int keyval;

printf("Enter the value to delete from the BST : ");

scanf("%d",&keyval);

deleteNode(root,keyval);

break;

}

case 3: {

int dis\_choice, dis\_switch\_error=1;

while(dis\_switch\_error==1){

printf("\n1. In-Order Traversal.\n2. Pre-Order Traversal.\n3. Post-Order Traversal.");

printf("\n\nChoose type of traversal - ");

scanf("%d",&dis\_choice);

fflush(stdin);

dis\_switch\_error=0;

switch(dis\_choice){

case 1: {

inorder(root);

break;

}

case 2: {

preorder(root);

break;

}

case 3: {

postorder(root);

break;

}

default:{

printf("Invalid Operation !!!");

dis\_switch\_error=1;

}

}

}

break;

}

case 4: {

int search\_val;

printf("Enter the number that you want to search : ");

scanf("%d",&search\_val);

search(root, search\_val);

break;

}

case 5: {

exit(0);

break;

}

}

}

}

struct node\* minValueNode(struct node\* node){

struct node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* deleteNode(struct node\* root, int key){

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else {

if (root->left==NULL && root->right==NULL)

return NULL;

else if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

struct node\* temp = minValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

struct node\* insertion(struct node\* node, int item){

if(node==NULL){

return createNode(item);

}

else{

if(item < node->data){

node->left= insertion(node->left, item);

}

else if(item > node->data){

node->right= insertion(node->right, item);

}

return node;

}

}

void inorder(struct node \*temp){

if (temp != NULL) {

inorder(temp->left);

printf("%d \n", temp->data);

inorder(temp->right);

}

}

void postorder(struct node \*temp){

if (temp != NULL) {

postorder(temp->left);

postorder(temp->right);

printf("%d \n", temp->data);

}

}

void preorder(struct node \*temp){

if (temp != NULL) {

printf("%d \n", temp->data);

preorder(temp->left);

preorder(temp->right);

}

}

void search(struct node \*temp, int val){

if(temp==NULL){

printf("Element not found !! or List empty !!\n");

}

else if(val==temp->data){

printf("Element %d found\n", val);

}

else if(val > temp->data){

return search(temp->right, val);

}

else if(val < temp->data){

return search(temp->left, val);

}

}

OUTPUT:

BST Operations

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 1

Enter the value to insert into the BST : 2

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 1

Enter the value to insert into the BST : 3

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 1

Enter the value to insert into the BST : 4

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 2

Enter the value to delete from the BST : 4

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 4

Enter the number that you want to search : 1

Element not found !! or List empty !!

1. Insertion

2. Deletion

3. Traversal

4. Search

5. Exit

Enter the option number to perform operation : 4

Enter the number that you want to search : 2

Element 2 found

EXPERIMENT:6 DATE:04-01-22

AIM: PROGRAM TO IMPLEMENT SET AND ITS OPERATIONS

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int ch,n,v,i,flag=1;

int a1[10],a2[10];

printf("enter the number of bits:\n");

scanf("%d",&n);

printf("\nenter the bits:\n");

for(i=0;i<n;i++)

{

scanf("%d",&v);

if(v<0||v>1)

{

printf("please enter a valid bit:\n");

}

else

{

a1[i]=v;

}

}

printf("enter the second bits:\n");

for(i=0;i<n;i++)

{

scanf("%d",&v);

if(v<0||v>1)

{

printf("please enter a valid bit:\n");

}

else

{

a2[i]=v;

}

}

while(1)

{

printf("\n\n###########enter the choice!!!#########\n");

printf("\n 1.Union\n 2.Intersection\n 3.Difference\n 4.Equality\n 5.Exit\n");

printf("please select a choice!!\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\n Union of the set is: \n");

for(i=0;i<n;i++)

{

if((a1[i]||a2[i])==1)

{

printf("1");

}

else

{

printf("0");

}

}

break;

case 2:

printf("\n Intersection of the set is: \n");

for(i=0;i<n;i++)

{

if((a1[i]&&a2[i])==1)

{

printf("1");

}

else

{

printf("0");

}

}

break;

case 3:

printf("\n Difference of the set is: \n");

for(i=0;i<n;i++)

{

if(a1[i]==1&&a2[i]==0)

{

printf("1");

}

else

{

printf("0");

}

}

break;

case 4:

printf("\n Equality of the set is: \n");

for(i=0;i<n;i++)

{

if(a1[i]!=a2[i])

{

flag=0;

printf("False\_\_The both set are not same");

exit(0);

}

}

if(flag==1)

{

printf("True\_\_Two sets are same");

}

break;

case 5:

{

printf("exiting.......");

break;

}

default:

{

printf("please enter a valid case");

break;

}

};

}

}

OUTPUT:

enter the number of bits:

6

enter the bits:

1

1

0

1

1

0

enter the second bits:

1

0

1

1

0

0

###########enter the choice!!!#########

1.Union

2.Intersection

3.Difference

4.Equality

5.Exit

please select a choice!!

1

Union of the set is:

111110

###########enter the choice!!!#########

1.Union

2.Intersection

3.Difference

4.Equality

5.Exit

please select a choice!!

2

Intersection of the set is:

100100

###########enter the choice!!!#########

1.Union

2.Intersection

3.Difference

4.Equality

5.Exit

please select a choice!!

3

Difference of the set is:

010010

###########enter the choice!!!#########

1.Union

2.Intersection

3.Difference

4.Equality

5.Exit

please select a choice!!

4

Equality of the set is:

False\_\_The both set are not same

EXPERIMENT:7 DATE:11-01-22

AIM:PROGRAM TO IMPLEMENT DISJOINT SET

PROGRAM:

#include<stdio.h>

struct disjoint

{

int parent[10];

int rank[10];

int n;

}dis;

int main()

{

int n,x,y;

printf("How many elements ?\n");

scanf("%d",&dis.n);

makeSet();

int ch,choice;

while(1)

{

printf("\n\_\_\_MENU\_\n");

printf("1. Union \n2.Find\n3.Display\n 4.Exit\n");

printf("enter choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

Union();

break;

case 2:

printf("Enter elements to check if connected components");

scanf("%d %d",&x,&y);

if (find(x) == find(y))

{

printf("Connected components\n") ;

}

else

{

printf("Not onnected components \n") ;

}

break;

case 3:

displaySet();

break;

case 4:

printf("exiting....");

break;

default:

printf("please enter a valid case");

}

}

}

void makeSet()

{

int i;

for (i = 0; i < dis.n; i++)

{

dis.parent[i] = i;

dis.rank[i]=0;

}

}

void displaySet()

{

int i;

printf("\nParent Array\n");

for ( i = 0; i < dis.n; i++)

{

printf("%d ",dis.parent[i]);

}

printf("\nRank Array\n");

for (int i = 0; i < dis.n; i++)

{

printf("%d ",dis.rank[i]);

}

printf("\n");

}

int find(int x)

{

if (dis.parent[x] != x)

{

dis.parent[x] = find(dis.parent[x]);

}

return dis.parent[x];

}

void Union()

{

int xset=0,yset=0,x,y;

printf("\nEnter elements to perform union\n");

scanf("%d %d",&x,&y);

xset = find(x);

yset = find(y);

if (xset == yset)

return;

if (dis.rank[xset] < dis.rank[yset])

{

dis.parent[xset] = yset;

dis.rank[xset]=-1;

}

else if (dis.rank[xset] > dis.rank[yset])

{

dis.parent[yset] = xset;

dis.rank[yset]=-1;

}

else

{

dis.parent[yset] = xset;

dis.rank[xset] = dis.rank[xset] + 1;

dis.rank[yset]=-1;

}

}

OUTPUT:

How many elements ?

4

\_\_\_MENU\_

1. Union

2.Find

3.Display

4.Exit

enter choice

1

Enter elements to perform union

9

8

\_\_\_MENU\_

1. Union

2.Find

3.Display

4.Exit

enter choice

1

Enter elements to perform union

5

4

\_\_\_MENU\_

1. Union

2.Find

3.Display

4.Exit

enter choice

3

Parent Array

0 1 2 3

Rank Array

0 0 0 0

\_\_\_MENU\_

1. Union

2.Find

3.Display

4.Exit

enter choice

2

Enter elements to check if connected components5

4

Connected components

EXPERIMENT:8 DATE:18-01-22

AIM:PROGRAM TO IMPLEMENT BINOMIAL HEAP

PROGRAM:

#include<stdio.h>

#include<malloc.h>

struct node {

int n;

int degree;

struct node\* parent;

struct node\* child;

struct node\* sibling;

};

int main() {

int i, n, m, l;

struct node\* p;

struct node\* np;

char ch;

printf("\nENTER THE NUMBER OF ELEMENTS:");

scanf("%d", &n);

printf("\nENTER THE ELEMENTS:\n");

for (i = 1; i <= n; i++) {

scanf("%d", &m);

np = CREATE\_NODE(m);

H = bin\_HEAP\_INSERT(H, np);

}

DISPLAY(H);

do {

printf("\nMENU:-\n");

printf(

"\n1)INSERT AN ELEMENT\n2)EXTRACT THE MINIMUM KEY NODE\n3)DECREASE A NODE KEY\n 4)DELETE A NODE\n5)QUIT\n");

scanf("%d", &l);

switch (l) {

case 1:

do {

printf("\nENTER THE ELEMENT TO BE INSERTED:");

scanf("%d", &m);

p = CREATE\_NODE(m);

H = bin\_HEAP\_INSERT(H, p);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nINSERT MORE(y/n)= \n");

fflush(stdin);

scanf("%c", &ch);

} while (ch == 'Y' || ch == 'n');

break;

case 2:

do {

printf("\nEXTRACTING THE MINIMUM KEY NODE");

p = bin\_HEAP\_EXTRACT\_MIN(H);

if (p != NULL)

printf("\nTHE EXTRACTED NODE IS %d", p->n);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nEXTRACT MORE(y/Y)\n");

fflush(stdin);

scanf("%c", &ch);

} while (ch == 'Y' || ch == 'y');

break;

case 3:

do {

printf("\nENTER THE KEY OF THE NODE TO BE DECREASED:");

scanf("%d", &m);

printf("\nENTER THE NEW KEY : ");

scanf("%d", &l);

bin\_HEAP\_DECREASE\_KEY(H, m, l);

printf("\nNOW THE HEAP IS:\n");

DISPLAY(H);

printf("\nDECREASE MORE(y/Y)\n");

fflush(stdin);

scanf("%c", &ch);

} while (ch == 'Y' || ch == 'y');

break;

case 4:

do {

printf("\nENTER THE KEY TO BE DELETED: ");

scanf("%d", &m);

bin\_HEAP\_DELETE(H, m);

printf("\nDELETE MORE(y/n)\n");

fflush(stdin);

scanf("%c", &ch);

} while (ch == 'y' || ch == 'n');

break;

case 5:

printf("\nTHANK U SIR\n");

break;

default:

printf("\nINVALID ENTRY...TRY AGAIN....\n");

}

} while (l != 5);

}

struct node\* MAKE\_bin\_HEAP();

int bin\_LINK(struct node\*, struct node\*);

struct node\* CREATE\_NODE(int);

struct node\* bin\_HEAP\_UNION(struct node\*, struct node\*);

struct node\* bin\_HEAP\_INSERT(struct node\*, struct node\*);

struct node\* bin\_HEAP\_MERGE(struct node\*, struct node\*);

struct node\* bin\_HEAP\_EXTRACT\_MIN(struct node\*);

int REVERT\_LIST(struct node\*);

int DISPLAY(struct node\*);

struct node\* FIND\_NODE(struct node\*, int);

int bin\_HEAP\_DECREASE\_KEY(struct node\*, int, int);

int bin\_HEAP\_DELETE(struct node\*, int);

int count = 1;

struct node\* MAKE\_bin\_HEAP() {

struct node\* np;

np = NULL;

return np;

}

struct node \* H = NULL;

struct node \*Hr = NULL;

int bin\_LINK(struct node\* y, struct node\* z) {

y->parent = z;

y->sibling = z->child;

z->child = y;

z->degree = z->degree + 1;

}

struct node\* CREATE\_NODE(int k) {

struct node\* p;//new node;

p = (struct node\*) malloc(sizeof(struct node));

p->n = k;

return p;

}

struct node\* bin\_HEAP\_UNION(struct node\* H1, struct node\* H2) {

struct node\* prev\_x;

struct node\* next\_x;

struct node\* x;

struct node\* H = MAKE\_bin\_HEAP();

H = bin\_HEAP\_MERGE(H1, H2);

if (H == NULL)

return H;

prev\_x = NULL;

x = H;

next\_x = x->sibling;

while (next\_x != NULL) {

if ((x->degree != next\_x->degree) || ((next\_x->sibling != NULL)

&& (next\_x->sibling)->degree == x->degree)) {

prev\_x = x;

x = next\_x;

} else {

if (x->n <= next\_x->n) {

x->sibling = next\_x->sibling;

bin\_LINK(next\_x, x);

} else {

if (prev\_x == NULL)

H = next\_x;

else

prev\_x->sibling = next\_x;

bin\_LINK(x, next\_x);

x = next\_x;

}

}

next\_x = x->sibling;

}

return H;

}

struct node\* bin\_HEAP\_INSERT(struct node\* H, struct node\* x) {

struct node\* H1 = MAKE\_bin\_HEAP();

x->parent = NULL;

x->child = NULL;

x->sibling = NULL;

x->degree = 0;

H1 = x;

H = bin\_HEAP\_UNION(H, H1);

return H;

}

struct node\* bin\_HEAP\_MERGE(struct node\* H1, struct node\* H2) {

struct node\* H = MAKE\_bin\_HEAP();

struct node\* y;

struct node\* z;

struct node\* a;

struct node\* b;

y = H1;

z = H2;

if (y != NULL) {

if (z != NULL && y->degree <= z->degree)

H = y;

else if (z != NULL && y->degree > z->degree)

H = z;

else

H = y;

} else

H = z;

while (y != NULL && z != NULL) {

if (y->degree < z->degree) {

y = y->sibling;

} else if (y->degree == z->degree) {

a = y->sibling;

y->sibling = z;

y = a;

} else {

b = z->sibling;

z->sibling = y;

z = b;

}

}

return H;

}

int DISPLAY(struct node\* H) {

struct node\* p;

if (H == NULL) {

printf("\nHEAP EMPTY");

return 0;

}

printf("\nTHE ROOT NODES ARE:-\n");

p = H;

while (p != NULL) {

printf("%d", p->n);

if (p->sibling != NULL)

printf("-->");

p = p->sibling;

}

printf("\n");

}

struct node\* bin\_HEAP\_EXTRACT\_MIN(struct node\* H1) {

int min;

struct node\* t = NULL;

struct node\* x = H1;

struct node \*Hr;

struct node\* p;

Hr = NULL;

if (x == NULL) {

printf("\nNOTHING TO EXTRACT");

my number, [31-01-2022 22:22]

return x;

}

// int min=x->n;

p = x;

while (p->sibling != NULL) {

if ((p->sibling)->n < min) {

min = (p->sibling)->n;

t = p;

x = p->sibling;

}

p = p->sibling;

}

if (t == NULL && x->sibling == NULL)

H1 = NULL;

else if (t == NULL)

H1 = x->sibling;

else if (t->sibling == NULL)

t = NULL;

else

t->sibling = x->sibling;

if (x->child != NULL) {

REVERT\_LIST(x->child);

(x->child)->sibling = NULL;

}

H = bin\_HEAP\_UNION(H1, Hr);

return x;

}

int REVERT\_LIST(struct node\* y) {

if (y->sibling != NULL) {

REVERT\_LIST(y->sibling);

(y->sibling)->sibling = y;

} else {

Hr = y;

}

}

struct node\* FIND\_NODE(struct node\* H, int k) {

struct node\* x = H;

struct node\* p = NULL;

if (x->n == k) {

p = x;

return p;

}

if (x->child != NULL && p == NULL) {

p = FIND\_NODE(x->child, k);

}

if (x->sibling != NULL && p == NULL) {

p = FIND\_NODE(x->sibling, k);

}

return p;

}

int bin\_HEAP\_DECREASE\_KEY(struct node\* H, int i, int k) {

int temp;

struct node\* p;

struct node\* y;

struct node\* z;

p = FIND\_NODE(H, i);

if (p == NULL) {

printf("\nINVALID CHOICE OF KEY TO BE REDUCED");

return 0;

}

if (k > p->n) {

printf("\nSORY!THE NEW KEY IS GREATER THAN CURRENT ONE");

return 0;

}

p->n = k;

y = p;

z = p->parent;

while (z != NULL && y->n < z->n) {

temp = y->n;

y->n = z->n;

z->n = temp;

y = z;

z = z->parent;

}

printf("\nKEY REDUCED SUCCESSFULLY!");

}

int bin\_HEAP\_DELETE(struct node\* H, int k) {

struct node\* np;

if (H == NULL) {

printf("\nHEAP EMPTY");

return 0;

}

bin\_HEAP\_DECREASE\_KEY(H, k, -1000);

np = bin\_HEAP\_EXTRACT\_MIN(H);

if (np != NULL)

printf("\nNODE DELETED SUCCESSFULLY");

}