***ASSIGNMENT-11***

1. ***WAP to implement AVL trees.***

#include<stdio.h>

#include<stdlib.h>

struct avlnode{

int info;

struct avlnode \*rchild,\*lchild;

};

typedef struct avlnode AVLNODE;

AVLNODE \*insert(AVLNODE \*root,int data);

AVLNODE \*deleteNode(AVLNODE \*root,int data);

int height(AVLNODE \*root);

int balance(AVLNODE \*root);

AVLNODE \*minvalnode(AVLNODE \*root);

AVLNODE \*leftrotate(AVLNODE \*root);

AVLNODE \*rightrotate(AVLNODE \*root);

void predisplay(AVLNODE \*root);

int main()

{

AVLNODE \*root = NULL;

printf("Enter your choice\n");

printf("1.Insert an element in AVL tree\n");

printf("2.Delete an element in AVL tree\n");

printf("3.Display inorder traversal\n");

printf("4.Exit\n);

while(1)

{

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter element you want to insert : ");

scanf("%d",&ch);

root = insert(root,ch);

break;

case 2:

printf("Enter element you want to delete : ");

scanf("%d",&ch);

root = deleteNode(root,ch);

break;

case 3:

predisplay(root);

printf("\n");

break;

case 4:

exit(1);

}

}

return 0;

}

AVLNODE \*insert(AVLNODE \*root,int data)

{

if(root==NULL)

{

root = (AVLNODE \*)malloc(sizeof(AVLNODE));

root->lchild = root->rchild = NULL;

root->info = data;

}

else if(root->info < data)

root->rchild = insert(root->rchild,data);

else

root->lchild = insert(root->lchild,data);

int a = balance(root);

if(a>1 && data < root->lchild->info)

return rightrotate(root);

if(a>1 && data > root->lchild->info)

{

root->lchild = leftrotate(root->lchild);

return rightrotate(root);

}

if(a<-1 && data < root->rchild->info)

{

root->rchild = rightrotate(root->rchild);

return leftrotate(root);

}

if(a<-1 && data > root->rchild->info)

return leftrotate(root);

return root;

}

AVLNODE \*deleteNode(AVLNODE \*root,int data)

{

if(root==NULL)

return root;

else if(root->info < data)

root->rchild = deleteNode(root->rchild,data);

else if(root->info > data)

root->lchild = deleteNode(root->lchild,data);

else

{

if(root->lchild==NULL)

{

AVLNODE \*tmp = root->rchild;

free(root);

return tmp;

}

if(root->rchild==NULL)

{

AVLNODE \*tmp = root->lchild;

free(root);

return tmp;

}

else

{

AVLNODE \*tmp = minvalnode(root->rchild);

root->info = tmp->info;

root->rchild = deleteNode(root->rchild,root->info);

}

}

int a =balance(root);

if(a>1 && balance(root->lchild)>=0)

return rightrotate(root);

if(a>1 && balance(root->lchild)<0)

{

AVLNODE \*tmp = leftrotate(root->lchild);

return rightrotate(root);

}

if(a<-1 && balance(root->rchild)>=0)

{

AVLNODE \*tmp = rightrotate(root->rchild);

return leftrotate(root);

}

if(a<-1 && balance(root->rchild)<0)

return leftrotate(root);

return root;

}

int height(AVLNODE \*root)

{

if(root==NULL)

return 0;

int a = 1+height(root->lchild);

int b = 1+height(root->rchild);

if(a>b)

return a;

return b;

}

int balance(AVLNODE \*root)

{

return height(root->lchild)-height(root->rchild);

}

AVLNODE \*leftrotate(AVLNODE \*root)

{

AVLNODE \*p = root->rchild;

AVLNODE \*q = p->lchild;

p->lchild = root;

root->rchild = q;

return p;

}

AVLNODE \*rightrotate(AVLNODE \*root)

{

AVLNODE \*p = root->lchild;

AVLNODE \*q = p->rchild;

p->rchild = root;

root->lchild = q;

return p;

}

void predisplay(AVLNODE \*root)

{

if(root!=NULL)

{

printf("%d ",root->info);

predisplay(root->lchild);

predisplay(root->rchild);

}

}

AVLNODE \*minvalnode(AVLNODE \*root)

{

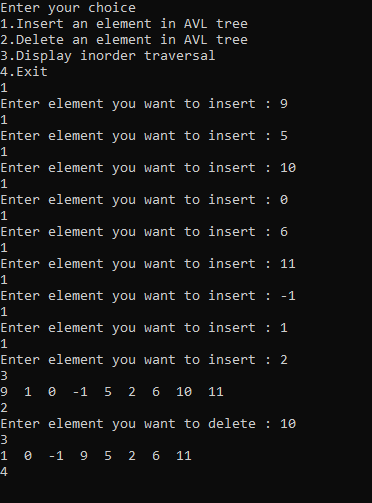
if(root->lchild==NULL)

return root;

return minvalnode(root->lchild);

}

***OUTPUT :***



***2. WAP to implement Hashing.***

*Linear probing* ***:***

#include<stdio.h>

#include<stdlib.h>

#define SIZE 10

int hashf(int key);

int search(int hash[],int key);

void insert(int hash[],int key);

void del(int hash[],int key);

void display(int hash[]);

int main()

{

int hash[SIZE];

int i;

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

hash[i]=-1;

printf("Enter your choice what you want to do\n");

printf("1.Search for a key in hash table\n");

printf("2.Insert a key in hash table\n");

printf("3.delete a key from hash table\n");

printf("4.Display hash table\n");

printf("5.Exit\n");

while(1)

{

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter key you want to search : ");

scanf("%d",&ch);

if(search(hash,ch))

printf("Key is there in hash table\n");

else

printf("Key is not there\n");

break;

case 2:

printf("Enter key you want to insert : ");

scanf("%d",&ch);

insert(hash,ch);

break;

case 3:

printf("Enter key you want to delete : ");

scanf("%d",&ch);

del(hash,ch);

break;

case 4:

display(hash);

break;

case 5:

exit(1);

}

}

return 0;

}

int hashf(int key)

{

return key%SIZE;

}

int search(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i)]==-1)

return 0;

else if(hash[hashf(key+i)]==key)

return 1;

}

return 0;

}

void insert(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i)]==-1 || hash[hashf(key+i)]==-2)

{

hash[hashf(key+i)]=key;

break;

}

}

}

void del(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i)]==-1)

break;

else if(hash[hashf(key+i)]==key)

{

hash[hashf(key+i)]=-2;

break;

}

}

}

void display(int hash[])

{

int i;

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

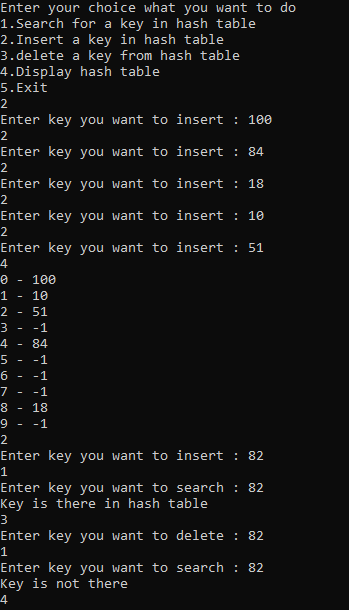
{

printf("%d - %d\n",i,hash[i]);

}

}

***OUTPUT :***



*Quadratic probing* ***:***

#include<stdio.h>

#include<stdlib.h>

#define SIZE 10

int hashf(int key);

int search(int hash[],int key);

void insert(int hash[],int key);

void del(int hash[],int key);

void display(int hash[]);

int main()

{

int hash[SIZE];

int i;

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

hash[i]=-1;

printf("Enter your choice what you want to do\n");

printf("1.Search for a key in hash table\n");

printf("2.Insert a key in hash table\n");

printf("3.delete a key from hash table\n");

printf("4.Display hash table\n");

printf("5.Exit\n");

while(1)

{

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter key you want to search : ");

scanf("%d",&ch);

if(search(hash,ch))

printf("Key is there in hash table\n");

else

printf("Key is not there\n");

break;

case 2:

printf("Enter key you want to insert : ");

scanf("%d",&ch);

insert(hash,ch);

break;

case 3:

printf("Enter key you want to delete : ");

scanf("%d",&ch);

del(hash,ch);

break;

case 4:

display(hash);

break;

case 5:

exit(1);

}

}

return 0;

}

int hashf(int key)

{

return key%SIZE;

}

int search(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i\*i)]==-1)

return 0;

else if(hash[hashf(key+i\*i)]==key)

return 1;

}

return 0;

}

void insert(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i\*i)]==-1 || hash[hashf(key+i\*i)]==-2)

{

hash[hashf(key+i\*i)]=key;

break;

}

}

}

void del(int hash[],int key)

{

int i;

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

{

if(hash[hashf(key+i\*i)]==-1)

break;

else if(hash[hashf(key+i\*i)]==key)

{

hash[hashf(key+i\*i)]=-2;

break;

}

}

}

void display(int hash[])

{

int i;

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

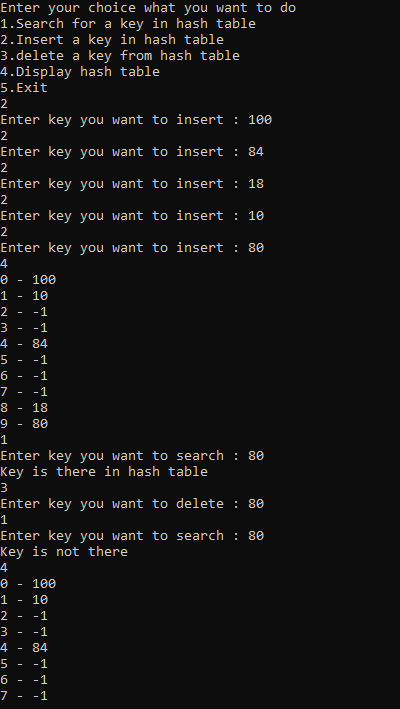
{

printf("%d - %d\n",i,hash[i]);

}

}

***OUTPUT :***





*Double Hashing* ***:***

#include<stdio.h>

#include<stdlib.h>

#define SIZE 10

int hashf(int key);

int hashf2(int key);

int search(int hash[],int key);

void insert(int hash[],int key);

void del(int hash[],int key);

void display(int hash[]);

int main()

{

int hash[SIZE];

int i;

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

hash[i]=-1;

printf("Enter your choice what you want to do\n");

printf("1.Search for a key in hash table\n");

printf("2.Insert a key in hash table\n");

printf("3.delete a key from hash table\n");

printf("4.Display hash table\n");

printf("5.Exit\n");

while(1)

{

int ch;

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter key you want to search : ");

scanf("%d",&ch);

if(search(hash,ch))

printf("Key is there in hash table\n");

else

printf("Key is not there\n");

break;

case 2:

printf("Enter key you want to insert : ");

scanf("%d",&ch);

insert(hash,ch);

break;

case 3:

printf("Enter key you want to delete : ");

scanf("%d",&ch);

del(hash,ch);

break;

case 4:

display(hash);

break;

case 5:

exit(1);

}

}

return 0;

}

int hashf(int key)

{

return key%SIZE;

}

int hashf2(int key)

{

return (7-key%7);

}

int search(int hash[],int key)

{

int i;

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

{

if(hash[(hashf(key)+i\*hashf2(key))%SIZE]==-1)

return 0;

else if(hash[(hashf(key)+i\*hashf2(key))%SIZE]==key)

return 1;

}

return 0;

}

void insert(int hash[],int key)

{

int i;

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

{

if(hash[(hashf(key)+i\*hashf2(key))%SIZE]==-1 || hash[(hashf(key)+i\*hashf2(key))%SIZE]==-2)

{

hash[(hashf(key)+i\*hashf2(key))%SIZE]=key;

break;

}

}

}

void del(int hash[],int key)

{

int i;

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

{

if(hash[(hashf(key)+i\*hashf2(key))%SIZE]==-1)

break;

else if(hash[(hashf(key)+i\*hashf2(key))%SIZE]==key)

{

hash[(hashf(key)+i\*hashf2(key))%SIZE]=-2;

break;

}

}

}

void display(int hash[])

{

int i;

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

{

printf("%d - %d\n",i,hash[i]);

}

}

***OUTPUT :***

