# VAN EMDE BOAS TREE

## INTRODUCTION

A **van Emde Boas tree** (or **van Emde Boas priority queue**), also known as a **vEB tree**, is a tree data structure which implements an associative array with *m*-bit integer keys. It performs all operations in O(log *m*) time. Notice that *m* is the *size* of the keys — therefore O(log *m*) is O(log log *n*) in a full tree, exponentially better than a self-balancing binary search tree. They also have good space efficiency when they contain a large number of elements, as discussed below. They were invented by a team led by Peter van Emde Boas in 1977.

## WORKING

For the sake of simplicity, let *log2 m = k* for some integer *k*. Define $M=2^m$. A vEB tree *T* over the universe {0,...,*M*-1} has a root node that stores an array *T.children* of length $M^{1/2}$. *T.children[i]* is a pointer to a vEB tree that is responsible for the values $\{iM^{1/2},...,(i+1)M^{1/2-1}\}$. Additionally, *T* stores two values *T.min* and *T.max* as well as an auxiliary vEB tree *T.aux*. Data is stored in a vEB tree as follows: The smallest value currently in the tree is stored in *T.min* and largest value is stored in *T.max*.

These two values are not stored anywhere else in the vEB tree. If *T* is empty then we use the convention that *T.max=-1* and *T.min=M*. Any other value *x* is stored in the subtree *T.children[i]* where **formula** the auxiliary tree *T.aux* keeps track of which children are non-empty, so *T.aux* contains the value *j* if and only if *T.children[j]* is non-empty.

APPLICATIONS

**1)** As an example, of you have a linear layout of stores on some line and want to find the closest store to some particular customer, using a vEB-tree could make the search exponentially faster than the (already fast) BST.

**2)** Used in Network routers

Insert

The call *Insert(T, x)* that inserts a value *x* into a vEB tree T operates as follows:

- If *T* is empty then we set *T.min = T.max = x* and we are done.
- Otherwise, if *x<T.min* then we insert *T.min* into the subtree *i* responsible for *T.min* and then set *T.min = x*.
- If *T.children[i]* was previously empty, then we also insert *i* into *T.aux*
- Otherwise, if *x>T.max* then we insert *T.max* into the subtree *i* responsible for *T.max* and then set *T.max = x*.
- If *T.children[i]* was previously empty, then we also insert *i* into *T.aux*
- Otherwise, *T.min< x < T.max* so we insert *x* into the subtree *i* responsible for *x*.
- If T.children[i] was previously empty, then we also insert *i* into *T.aux*.

Delete

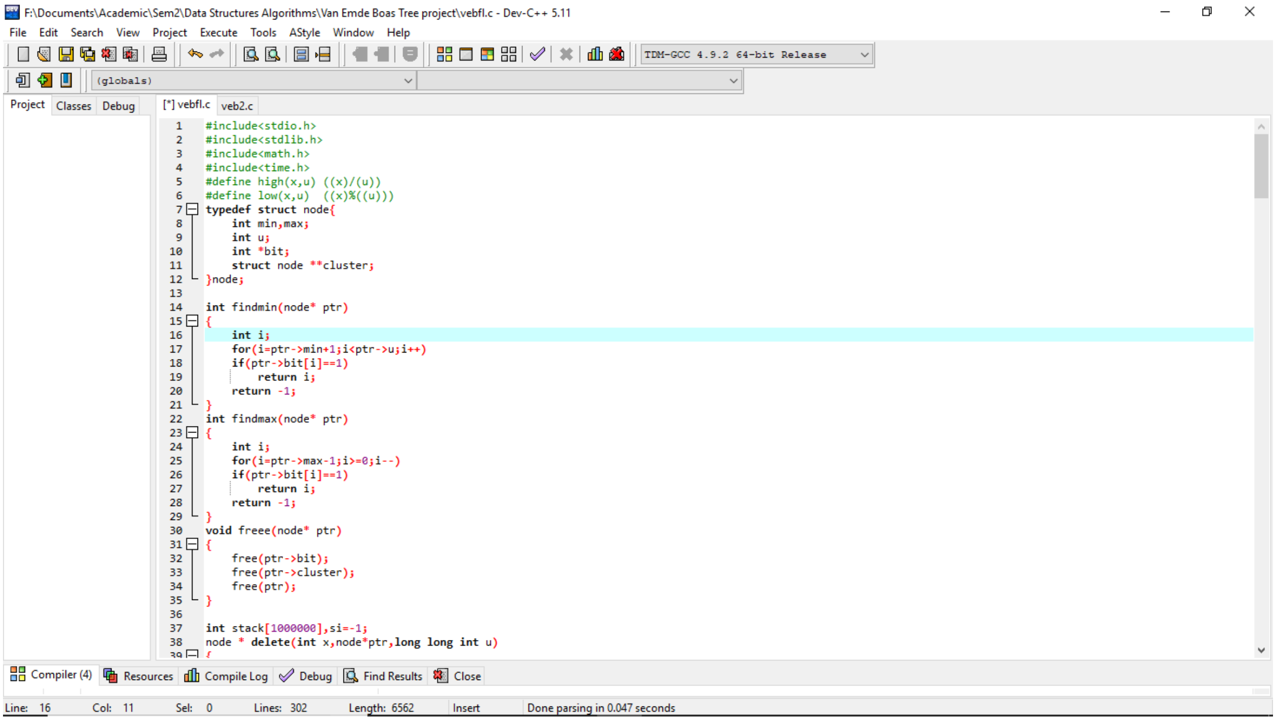
Deletion from vEB trees is the trickiest of the operations. The call *Delete(T, x)* that deletes a value *x* from a vEB tree T operates as follows:

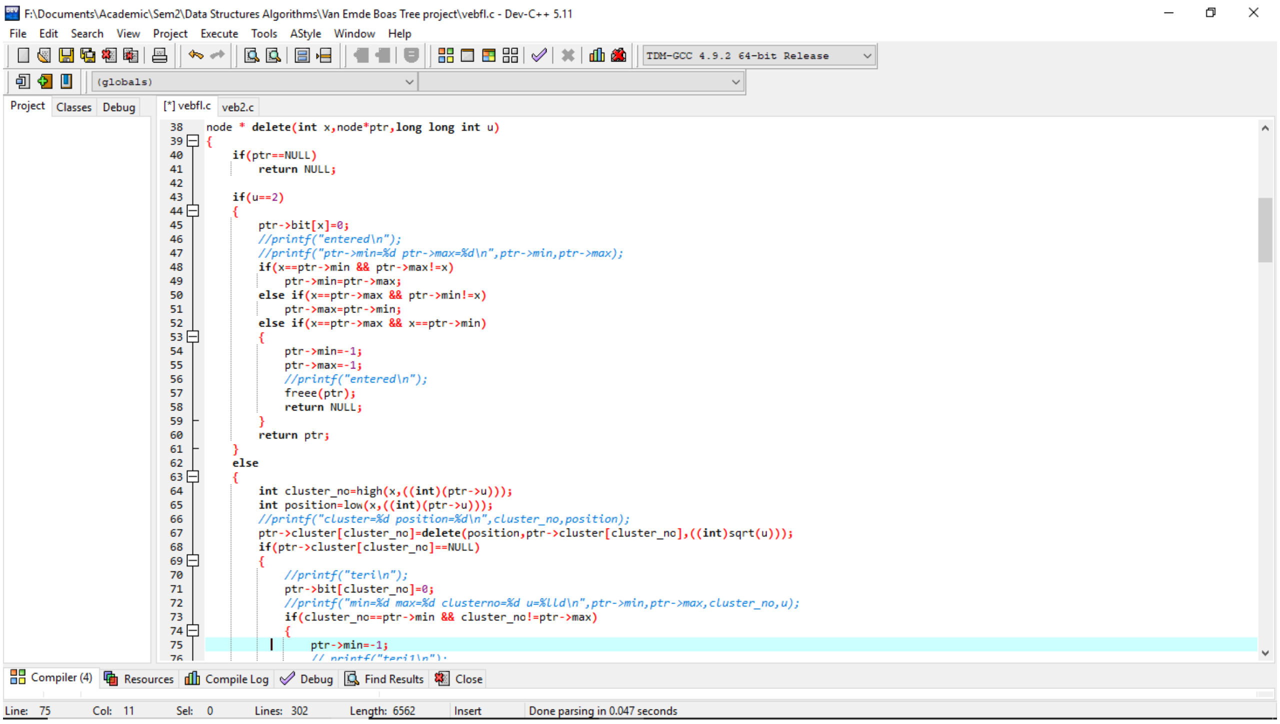
- If *T.min = T.max = x* then *x* is the only element stored in the tree and we set *T.min = M* and *T.max = -1* to indicate that the tree is empty.

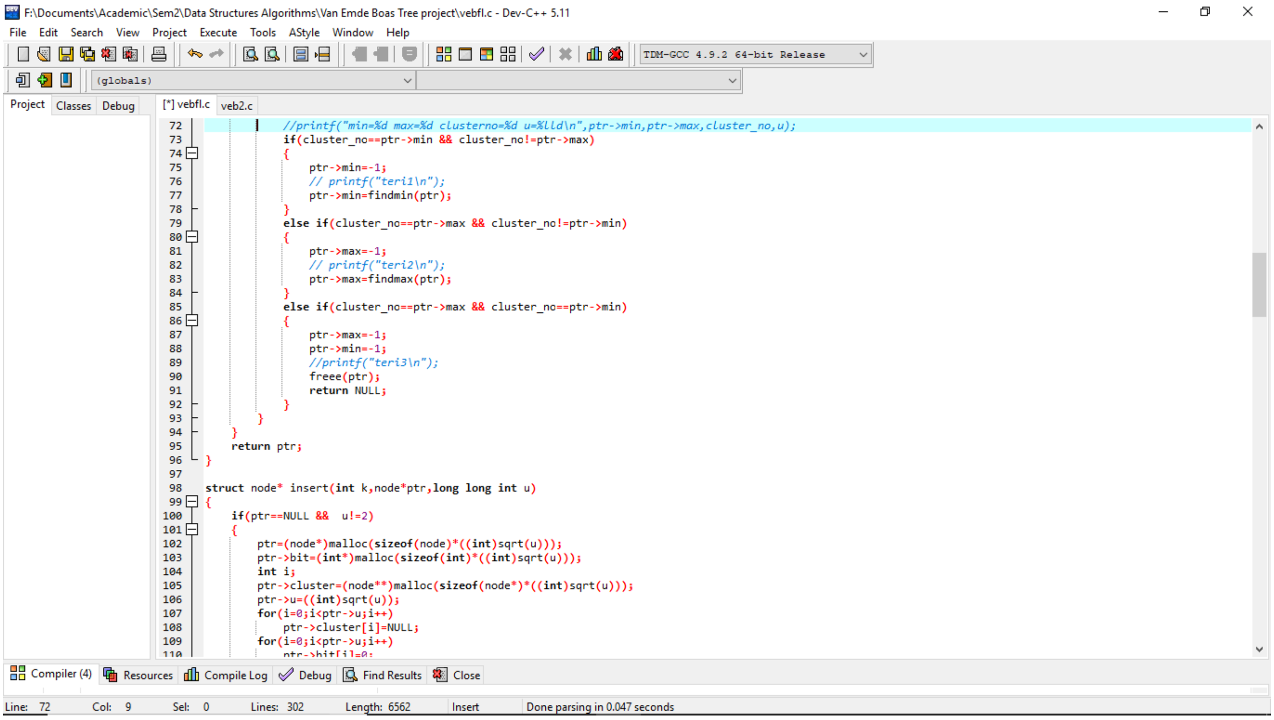
- Otherwise, if *x = T.min* then we need to find the second-smallest value *y* in the vEB tree, delete it from its current location, and set *T.min=y*. The second-smallest value *y* is either *T.max* or *T.children[T.aux.min].min*, so it can be found in 'O'(1) time. In the latter case we delete *y* from the subtree that contains it. Similarly, if *x = T.max* then we need to find the second-largest value *y* in the vEB tree, delete it from its current location, and set *T.max=y*. The second-largest value *y* is either *T.min* or *T.children[T.aux.max].max*, so it can be found in 'O'(1) time.

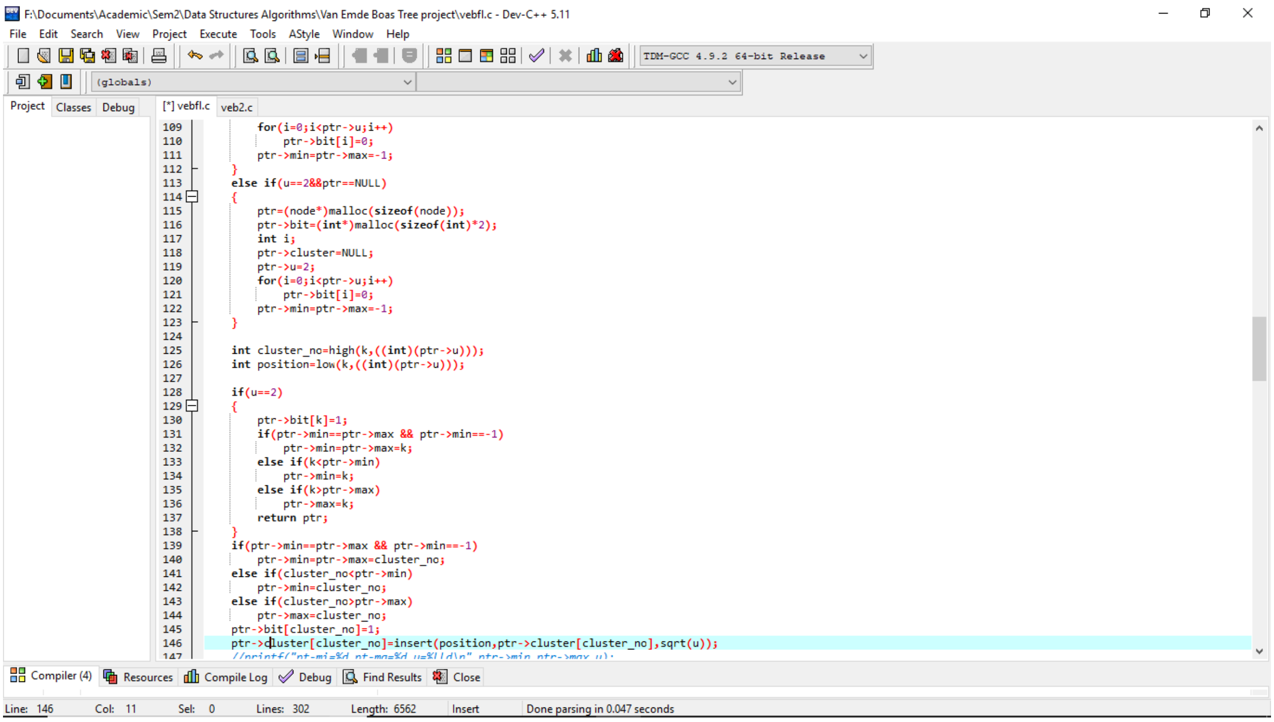
- In the latter case, we delete *y* from the subtree that contains it. Otherwise, we have the typical case where *x≠T.min* and *x≠T.max*. In this case we delete x from the subtree *T.children[i]* that contains *x*. In any of the above cases, if we delete the last element *x* or *y* from any subtree *T.children[i]* then we also delete *I* from *T.aux.*

CODE


F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
[*] vebfl.c veb2.c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define high(x,u) ((x)/(u))
#define low(x,u) ((x)%((u)))
typedef struct node{
int min,max;
int u;
int *bit;
struct node **cluster;
}node;
int findmin(node* ptr)
{
int i;
for(i=ptr->min+1;i<ptr->u;i++)
if(ptr->bit[i]==1)
return i;
return -1;
}
int findmax(node* ptr)
{
int i;
for(i=ptr->max-1;i>=0;i--)
if(ptr->bit[i]==1)
return i;
return -1;
}
void freee(node* ptr)
{
free(ptr->bit);
free(ptr->cluster);
free(ptr);
}
int stack[1000000],si=-1;
node * delete(int x,node*ptr,long long int u)
Compiler (4) Resources Compile Log Debug Find Results Close
Line: 16 Col: 11 Sel: 0 Lines: 302 Length: 6562 Insert Done parsing in 0.047 seconds


F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
[*] vebfl.c veb2.c
node * delete(int x,node*ptr,long long int u)
{
if(ptr==NULL)
return NULL;
if(u==2)
{
ptr->bit[x]=0;
//printf("entered\n");
//printf("ptr->min=%d ptr->max=%d\n",ptr->min,ptr->max);
if(x==ptr->min && ptr->max!=x)
ptr->min=ptr->max;
else if(x==ptr->max && ptr->min!=x)
ptr->max=ptr->min;
else if(x==ptr->max && x==ptr->min)
{
ptr->min=-1;
ptr->max=-1;
//printf("entered\n");
freee(ptr);
return NULL;
}
return ptr;
}
else
{
int cluster_no=high(x,((int)(ptr->u)));
int position=low(x,((int)(ptr->u)));
//printf("cluster=%d position=%d\n",cluster_no,position);
ptr->cluster[cluster_no]=delete(position,ptr->cluster[cluster_no],((int)sqrt(u)));
if(ptr->cluster[cluster_no]==NULL)
{
//printf("teri\n");
ptr->bit[cluster_no]=0;
//printf("min=%d max=%d clusterno=%d u=%lld\n",ptr->min,ptr->max,cluster_no,u);
if(cluster_no==ptr->min && cluster_no!=ptr->max)
{
ptr->min=-1;
Compiler (4) Resources Compile Log Debug Find Results Close
Line: 75 Col: 11 Sel: 0 Lines: 302 Length: 6562 Insert Done parsing in 0.047 seconds


F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11
//printf("min=%d max=%d clusterno=%d u=%lld\n",ptr->min,ptr->max,cluster_no,u);
if(cluster_no==ptr->min && cluster_no!=ptr->max)
{
ptr->min=-1;
// printf("teri1\n");
ptr->min=findmin(ptr);
}
else if(cluster_no==ptr->max && cluster_no!=ptr->min)
{
ptr->max=-1;
// printf("teri2\n");
ptr->max=findmax(ptr);
}
else if(cluster_no==ptr->max && cluster_no==ptr->min)
{
ptr->max=-1;
ptr->min=-1;
//printf("teri3\n");
freee(ptr);
return NULL;
}
}
}
return ptr;
}
struct node* insert(int k,node*ptr,long long int u)
{
if(ptr==NULL && u!=2)
{
ptr=(node*)malloc(sizeof(node)*((int)sqrt(u)));
ptr->bit=(int*)malloc(sizeof(int)*((int)sqrt(u)));
int i;
ptr->cluster=(node**)malloc(sizeof(node*)*((int)sqrt(u)));
ptr->u=((int)sqrt(u));
for(i=0;i<ptr->u;i++)
ptr->cluster[i]=NULL;
for(i=0;i<ptr->u;i++)


F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11
for(i=0;i<ptr->u;i++)
ptr->bit[i]=0;
ptr->min=ptr->max=-1;
}
else if(u==2&&ptr==NULL)
{
ptr=(node*)malloc(sizeof(node));
ptr->bit=(int*)malloc(sizeof(int)*2);
int i;
ptr->cluster=NULL;
ptr->u=2;
for(i=0;i<ptr->u;i++)
ptr->bit[i]=0;
ptr->min=ptr->max=-1;
}
int cluster_no=high(k,((int)(ptr->u)));
int position=low(k,((int)(ptr->u)));
if(u==2)
{
ptr->bit[k]=1;
if(ptr->min==ptr->max && ptr->min==-1)
ptr->min=ptr->max=k;
else if(k<ptr->min)
ptr->min=k;
else if(k>ptr->max)
ptr->max=k;
return ptr;
}
if(ptr->min==ptr->max && ptr->min==-1)
ptr->min=ptr->max=cluster_no;
else if(cluster_no<ptr->min)
ptr->min=cluster_no;
else if(cluster_no>ptr->max)
ptr->max=cluster_no;
ptr->bit[cluster_no]=1;
ptr->cluster[cluster_no]=insert(position,ptr->cluster[cluster_no],sqrt(u));

F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11

[*] vebfl.c | veb2.c

```c
    ptr->cluster[cluster_no]=insert(position,ptr->cluster[cluster_no],sqrt(u));
    //printf("pt-mi=%d pt-ma=%d u=%lld\n",ptr->min,ptr->max,u);
    return ptr;
}


int findnext(node*ptr,int u)
{
    if(u==2)
        return ptr->min;

    int val=findnext(ptr->cluster[ptr->min],((int)sqrt(u)));
    return (ptr->min*ptr->u + val);
}

int member(node *ptr,int x,long long int u)
{
    if(ptr==NULL)
        return 0;
    if(u==2)
    {
        if(ptr->bit[x]==0)
            return 0;
        else
            return 1;
    }
    int cluster_no=high(x,((int)(ptr->u)));
    int position=low(x,((int)(ptr->u))),i;

    return member(ptr->cluster[cluster_no],position,(int)sqrt(u));
}

int succ(node*ptr,int x,int u)
{
    if(ptr==NULL)
        return -1;
    if(u==2)
    {
        //printf("ptr->max=%d ptr->min=%d x=%d\n",ptr->max,ptr->min,x);
```

Line: 183 Col: 6 Sel: 0 Lines: 302 Length: 6562 Insert Done parsing in 0.047 seconds

F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.c - Dev-C++ 5.11

[*] vebfl.c | veb2.c

```c
        if(x<ptr->max)
            return ptr->max;
        return -1;
    }

    int cluster_no=high(x,((int)(ptr->u)));
    int position=low(x,((int)(ptr->u))),i;
    //printf("cluster_no=%d position=%d x=%d u=%d ptr->u=%d\n",cluster_no,position,x,u,ptr->u);
    //printf("ptr->max=%d ptr->min=%d x=%d\n",ptr->max,ptr->min,x);

    if(ptr->min==-1)
        return -1;
    if(cluster_no>ptr->max&&ptr->max!=-1)
        return -1;
    if(cluster_no<ptr->min)
    {
        int val=findnext(ptr->cluster[ptr->min],ptr->u);
        return ptr->min*ptr->u+val;
    }
    //printf("again\n");
    int r=succ(ptr->cluster[cluster_no],position,(int)sqrt(u));
    if(r==-1)
    {
        for(i=cluster_no+1;i<ptr->u;i++)
            if(ptr->bit[i]==1)
                break;
        if(i==ptr->u)
            return -1;
        else
        {
            int val=findnext(ptr->cluster[i],ptr->u);
            return i*ptr->u+val;
        }
    }
    else
        return cluster_no*ptr->u + r;
}

void print(node* ptr,long long int u,long long int no)
```

Line: 222 Col: 1 Sel: 0 Lines: 302 Length: 6562 Insert Done parsing in 0.047 seconds

```c
void print(node* ptr,long long int u,long long int no)
{
    if(ptr==NULL)
        return;
    if(ptr!=NULL && u==2)
    {
        //printf("ptr->bit[0] : %d\n",ptr->bit[0]);
        fflush(stdout);
        if(ptr->bit[0]==1)
        {
            stack[++si]=no;
            //printf("%lld ",no);
        }
        if(ptr->bit[1]==1)
        {
            stack[++si]=no+1;
            //printf("%lld ",no+1);
        }
        return;
    }
    int i;
    for(i=0;i<ptr->u;i++)
    {
        long long int tem=ptr->u*i;
        long long int to=tem+no ;
        print(ptr->cluster[i],(int)sqrt(u),to);
    }
}

int main()
{
    int n,i,j,k,m,l,c,x;
    node *head=NULL;
    long long int temp=65536;
    long long int univ=temp*temp;
    printf("**WELCOME**\n\nEnter as:\n'I'->Insert\n'S'->Search\n'D'->Delete\n'P'->Print\n'Q'->Quit\n\n");
    while((c=getchar())!='Q')
    {
```

```c
    printf("**WELCOME**\n\nEnter as:\n'I'->Insert\n'S'->Search\n'D'->Delete\n'P'->Print\n'Q'->Quit\n\n");
    while((c=getchar())!='Q')
    {
        if(c=='I')
        {
            scanf("%d",&x);
            head=insert(x,head,univ);
        }
        else if(c=='S')
        {
            scanf("%d",&x);
            if(member(head,x,univ)==1)
                printf("Found\n");
            else
                printf("Not Found\n");
        }
        else if(c=='P')
        {
            si=-1;
            print(head,univ,0);
            //if(member(head,37,univ))
            //printf("Found");
            //else printf("Not Found");
            for(i=0;i<=si;i++)
            {
                if(i!=si)
                    printf("%d ",stack[i]);
                else
                    printf("%d",stack[si]);
            }
            printf("\n");
        }
        else if(c=='D')
        {
            scanf("%d",&x);
            head=delete(x,head,univ);
            //printf("\n");
        }
        else if(c=='N')
```

# OUTPUT


F:\Documents\Academic\Sem2\Data Structures Algorithms\Van Emde Boas Tree project\vebfl.exe
Enter as:
'I'->Insert
'S'->Search
'D'->Delete
'P'->Print
'Q'->Quit
I
3
I
7
I
1
I
4
I
5
P
1 3 4 5 7
S
2
Not Found
S
4
Found
D
5
P
1 3 4 7
D
3
D
7
P
1 4
I
2
I
0
P
0 1 2 4