(14) BINARY TREE USING LINKED LIST

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    struct node * lc;
    int data;
    struct node * rc;
};
struct node * root, * rcptr, * lcptr, * new, * parent, * ptr, * ptr1, * ptr0, * ptr2;
int item, key, top, top1, flag;
struct node * stack[100];
void creation(struct node * ptr, int item) {
    int option, newl;
    if (ptr != NULL) {
        ptr -> data = item;
        printf("Does the node %d has left subtree <1/0>\n", item);
        scanf("%d", & option);
        if (option == 1) {
            lcptr = malloc(sizeof(struct node));
            ptr -> lc = lcptr;
            printf("Enter the item to be inserted\n");
            scanf("%d", & newl);
            creation(lcptr, newl);
        } else
            ptr -> lc = NULL;
        printf("Does the node %d has left subtree <1/0>\n", item);
        scanf("%d", & option);
        if (option == 1) {
            rcptr = malloc(sizeof(struct node));
            ptr -> rc = rcptr;
            printf("Enter the item to be inserted\n");
            scanf("%d", & newl);
            creation(rcptr, newl);
        } else
            ptr -> rc = NULL;
    }
}
void push(struct node * ptr) {
    top = top + 1;
    stack[top] = ptr;
}
struct node * pop() {
    if (top != -1) {
        ptr = stack[top];
        top = top - 1;
        return ptr;
    }
}
struct node * search_link(struct node * ptr, int key) {
    struct node * ptr3, * ptr4;
    push(ptr);
    while (top != -1) {
        ptr = pop();
        if (ptr != NULL) {
            ptr3 = ptr -> lc;
            ptr4 = ptr -> rc;
            if (ptr -> data == key)
                ptr2 = ptr;
            if (ptr3 != NULL)
```

```
                push(ptr4);
            if (ptr != NULL)
                push(ptr3);
        }
    }
    if (ptr2 -> data != key)
        return NULL;
    else
        return ptr2;
}
void insertion(int key, int item) {
    int option;
    ptr = search_link(ptr0, key);
    if (ptr == NULL)
        printf("Search unsucessful\n\t");
    else {
        if (ptr -> lc == NULL || ptr -> rc == NULL) {
            printf("\nleft or right child<1,0>");
            scanf("%d", & option);
            if (option == 1) {
                if (ptr -> lc == NULL) {
                    new = malloc(sizeof(struct node));
                    new -> data = item;
                    new -> lc = new -> rc = NULL;
                    ptr -> lc = new;
                } else
                    printf("\nInsertion not possible as left child");
            } else {
                if (ptr -> rc == NULL) {
                    new = malloc(sizeof(struct node));
                    new -> data = item;
                    new -> lc = new -> rc = NULL;
                    ptr -> rc = new;
                } else
                    printf("\nInsertion not possible as right child");
            }
        } else
            printf("\nThe key node already has child");
    }
}
void inorder(struct node * ptr1) {
    if (ptr1 != NULL) {
        inorder(ptr1 -> lc);
        printf("%d\t", ptr1 -> data);
        inorder(ptr1 -> rc);
    }
}
void preorder(struct node * ptr1) {
    if (ptr1 != NULL) {
        printf("%d\t", ptr1 -> data);
        preorder(ptr1 -> lc);
        preorder(ptr1 -> rc);
    }
}
void postorder(struct node * ptr1) {
    if (ptr1 != NULL) {
        postorder(ptr1 -> lc);
        postorder(ptr1 -> rc);
        printf("%d\t", ptr1 -> data);
    }
```

```
}
struct node * search_parent(struct node * ptr, int item) {
    struct node * ptr3, * ptr4;
    top = -1;
    flag = 0;
    push(ptr);
    while (ptr -> data != item) {
        ptr = pop();
        if (ptr != NULL) {
            ptr3 = ptr -> lc;
            ptr4 = ptr -> rc;
            if (ptr -> data == item) {
                flag = 1;
                break;
            }
            if (ptr3 != NULL) {
                parent = ptr;
                push(ptr3);
            }
            if (ptr4 != NULL) {
                parent = ptr;
                push(ptr4);
            }
        }
    }
    if (flag == 0)
        return NULL;
    else
        return parent;
}
void deletion(int item) {
    struct node * c;
    ptr = root;
    if (ptr == NULL)
        printf("\nTree is empty\n\t");
    else {
        if (ptr -> rc == NULL && ptr -> lc == NULL) {
            root -> data = 0;
            ptr0 = ptr1 = NULL;
            return;
        }
        parent = search_parent(ptr, item);
        if (parent == NULL)
            printf("Parent node not found\n");
        else {
            if (parent -> lc != NULL) {
                c = parent -> lc;
                if (c -> data == item) {
                    parent -> lc = NULL;
                    c -> data = 0;
                    c -> lc = NULL;
                    c -> rc = NULL;
                    free(c);
                }
            }
            if (parent -> rc != NULL) {
                c = parent -> rc;
                if (c -> data == item) {
                    parent -> rc = NULL;
                    c -> data = 0;
```

```
                c -> lc = NULL;
                c -> rc = NULL;
                free(c);
            }
        }
    }
  }
}
void main() {
  int choice, k = 0;
  root = malloc(sizeof(struct node));
  root -> lc = NULL;
  root -> rc = NULL;
  printf("\n1.Creation\n2.Insertion\n3.Deletion\n4.inorder\n5.Postorder\n6.Preorder\n");
  while (k == 0) {
    printf("choice : ");
    scanf("%d", & choice);
    switch (choice) {
    case 1:
      printf("Enter the item to be inserted at root node\n");
      scanf("%d", & item);
      creation(root, item);
      ptr1 = root;
      ptr0 = root;
      break;
    case 2:
      if (ptr1 != NULL) {
        printf("Enter the key node after which new node to be inserted\n");
        scanf("%d", & key);
        printf("\nEnter the item to be inserted\n\t");
        scanf("%d", & item);
        insertion(key, item);
      } else
        printf("Create tree first then press this option");
      break;
    case 3:
      if (ptr1 != NULL) {
        printf("\nEnter the leaf node to be deleted");
        scanf("%d", & item);
        deletion(item);
      } else
        printf("\nNo tree created\n\t");
      break;
    case 4:
      inorder(ptr1);
      break;
    case 5:
      postorder(ptr1);
      break;
    case 6:
      preorder(ptr1);
      break;
    default:
      exit(0);
    }
  }
}
```

```
1.Creation
2.Insertion
3.Deletion
4.inorder
5.Postorder
6.Preorder
choice : 1
Enter the item to be inserted at root node
6
Does the node 6 has left subtree <1/0>
1
Enter the item to be inserted
7
Does the node 7 has left subtree <1/0>
1
Enter the item to be inserted
2
Does the node 2 has left subtree <1/0>
0
Does the node 2 has left subtree <1/0>
0
Does the node 7 has left subtree <1/0>
1
Enter the item to be inserted
9
Does the node 9 has left subtree <1/0>
0
Does the node 9 has left subtree <1/0>
0
Does the node 6 has left subtree <1/0>
1
Enter the item to be inserted
5
Does the node 5 has left subtree <1/0>
1
Enter the item to be inserted
11
Does the node 11 has left subtree <1/0>
0
Does the node 11 has left subtree <1/0>
0
Does the node 5 has left subtree <1/0>
1
Enter the item to be inserted
13
Does the node 13 has left subtree <1/0>
0
Does the node 13 has left subtree <1/0>
0
```

```
Enter your choice : 4
Inorder Traversal: 30 35 45 46 50 55 59 60 77
Enter your choice : 5
Preorder Traversal: 50 45 30 35 46 60 55 59 77
Enter your choice : 6
Postorder Traversal: 35 30 46 45 59 55 77 60 50
Enter your choice : 2
Enter the item to be inserted : 80
Enter your choice : 4
Inorder Traversal: 30 35 45 46 50 55 59 60 77 80
Enter your choice : 5
Preorder Traversal: 50 45 30 35 46 60 55 59 77 80
Enter your choice : 6
Postorder Traversal: 35 30 46 45 59 55 80 77 60 50
Enter your choice : 3
Enter the value of node to be deleted : 46
Enter your choice : 4
Inorder Traversal: 30 35 45 50 55 59 60 77 80
Enter your choice : 5
Preorder Traversal: 50 45 30 35 60 55 59 77 80
Enter your choice : 6
Postorder Traversal: 35 30 45 59 55 80 77 60 50
Enter your choice : 7
Exit

--------------------------------
Process exited after 181 seconds with return value 7
Press any key to continue . . .
```

(15) GRAPH USING ADJACENCY MATRIX BFS AND DFS

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define visited 2
void graph_traversal_D();
void graph_traversal_B();
void DFS(int vertex);
void BFS(int vertex);
void make_graph();
void push(int vertex);
int pop();
int isEmpty_D();
void enqueue(int vertex);
int dequeue();
int isEmpty_B();
int top = -1, front = -1, rear = -1, vertices, ;
int stack[MAX], queue[MAX], adjacent_matrix[MAX][MAX], vertex_status[MAX];;
void main() {
   int choice;
   printf("GRAPH TRAVERSAL USING ADJASCENCY MATRIX\n");
   do {
      printf("\n 1:Depth First Search\n 2:Breadth First Search\n 3:Exit\n");
      printf("Enter your choice : ");
      scanf("%d", & choice);
      switch (choice) {
```

```c
        case 1:
            printf("\n\tDFS TRAVERSAL\n\n");
            make_graph();
            graph_traversal_D();
            break;
        case 2:
            printf("\n\tBFS TRAVERSAL\n\n");
            make_graph();
            graph_traversal_B();
            break;
        case 3:
            break;
        default:
            printf("\nInvalid choice\n");
        }
    } while (choice != 3);
}
void graph_traversal_D() {
    int vertex;
    for (vertex = 0; vertex < vertices; vertex++) {
        vertex_status[vertex] = initial;
    }
    printf("Enter Starting Vertex for DFS:\t");
    scanf("%d", & vertex);
    DFS(vertex);
    printf("\n");
}
void DFS(int vertex) {
    int count;
    push(vertex);
    while (!isEmpty_D()) {
        vertex = pop();
        if (vertex_status[vertex] == initial) {
            printf("%3d", vertex);
            vertex_status[vertex] = visited;
        }
        for (count = vertices - 1; count >= 0; count--) {
            if (adjacent_matrix[vertex][count] == 1 && vertex_status[count] == initial) {
                push(count);
            }
        }
    }
}
void push(int vertex) {
    if (top == (MAX - 1)) {
        printf("Stack Overflow\n");
        return;
    }
    top = top + 1;
    stack[top] = vertex;
}
int pop() {
    int vertex;
    if (top == -1) {
        printf("Stack Underflow\n");
        exit(1);
    } else {
        vertex = stack[top];
        top = top - 1;
        return vertex;
```

```
    }
}
int isEmpty_D() {
    if (top == -1) {
        return 1;
    } else {
        return 0;
    }
}
void graph_traversal_B() {
    int vertex;
    for (vertex = 0; vertex < vertices; vertex++) {
        vertex_status[vertex] = initial;
    }
    printf("Enter Starting Vertex for BFS:\t");
    scanf("%d", & vertex);
    BFS(vertex);
    printf("\n");
}
void BFS(int vertex) {
    int count;
    enqueue(vertex);
    while (!isEmpty_B()) {
        vertex = dequeue();
        if (vertex_status[vertex] == initial) {
            printf("%3d", vertex);
            vertex_status[vertex] = visited;
        }
        for (count = vertices - 1; count >= 0; count--) {
            if (adjacent_matrix[vertex][count] == 1 && vertex_status[count] == initial) {
                enqueue(count);
            }
        }
    }
}
void enqueue(int vertex) {
    if (rear == (MAX - 1))
        printf("Queue Overflow\n");
    else {
        if (front == -1) {
            front = rear = 0;
            queue[rear] = vertex;
        } else
            queue[++rear] = vertex;
    }
}
int dequeue() {
    int vertex;
    if (front == -1) {
        printf("Queue Underflow\n");
        exit(1);
    } else {
        vertex = queue[front];
        if (front == rear) {
            front = rear = -1;
        } else front++;
        return vertex;
    }
}
int isEmpty_B() {
```

```
    if (front == -1) {
        return 1;
    } else {
        return 0;
    }
}
void make_graph() {
        int count, maximum_edges, origin_vertex,
        destination_vertex;
        printf("Enter total number of
            vertices: \t "); scanf(" % d ", &vertices);
            maximum_edges = vertices * (vertices - 1);
            for (count = 0; count < maximum_edges; count++) {
                printf("Enter Edge [%d] Co-ordinates [-1 -1] to Quit\n", count +
                    1);
                printf("Enter Origin Vertex Point:\t");
                scanf("%d", &
                    origin_vertex);
                printf("Enter Destination Vertex Point:\t");
                scanf("%d", & destination_vertex);
                if ((origin_vertex == -1) && (destination_vertex == -1)) {
                    break;
                }
                if (origin_vertex >= vertices || destination_vertex >= vertices || origin_vertex <
                    0 || destination_vertex < 0) {
                    printf("\tEdge Co - ordinates are Invalid\n");
                    count--;
                } else {
                    adjacent_matrix[origin_vertex][destination_vertex] = 1;
                }
            }
        }
}
```

```
Enter Edge [4] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:       2
Enter Destination Vertex Point: 1
Enter Edge [5] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:       3
Enter Destination Vertex Point: 4
Enter Edge [6] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:       4
Enter Destination Vertex Point: 0
Enter Edge [7] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:       -1
Enter Destination Vertex Point: -1
Enter Starting Vertex for BFS:   1
  1   0   3   2   4

 1:Depth First Search
 2:Breadth First Search
 3:Exit
Enter your choice : 3
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ gcc graph_matri
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out
GRAPH TRAVERSAL USING ADJASCENCY MATRIX

 1:Depth First Search
 2:Breadth First Search
 3:Exit
Enter your choice : 1

        DFS TRAVERSAL

Enter total number of vertices: 5
Enter Edge [1] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      0
Enter Destination Vertex Point: 2
Enter Edge [2] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      0
Enter Destination Vertex Point: 3
Enter Edge [3] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      1
Enter Destination Vertex Point: 0
Enter Edge [4] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      2
Enter Destination Vertex Point: 1
Enter Edge [5] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      3
Enter Destination Vertex Point: 4
Enter Edge [6] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      4
Enter Destination Vertex Point: 0
Enter Edge [7] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      -1
Enter Destination Vertex Point: -1
Enter Starting Vertex for DFS:  1
  1  0  2  3  4

 1:Depth First Search
 2:Breadth First Search
 3:Exit
Enter your choice : 2

        BFS TRAVERSAL

Enter total number of vertices: 5
Enter Edge [1] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      0
Enter Destination Vertex Point: 2
Enter Edge [2] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      0
Enter Destination Vertex Point: 3
Enter Edge [3] Co-ordinates [-1 -1] to Quit
Enter Origin Vertex Point:      1
Enter Destination Vertex Point: 0
```

16) QUICK SORT, HEAP SORT, MERGE SORT

```
#include<stdio.h>
int a[20], b[20], h[20], length, heapsize;
void exchange(int * a, int * largest);
void display(int b[]);
void heapify(int a[], int i);
void buildheap(int a[]);
void heapsort(int a[]);
void quick(int x[], int first, int last);
int partition(int x[], int first, int last);
int mergesort(int a[], int p, int r);
int merge(int a[], int p, int q, int s);
void quick(int x[], int first, int last) {
    int pivot;
    if (first < last) {
        pivot = partition(x, first, last);
        quick(x, first, pivot - 1);
        quick(x, pivot + 1, last);
    }
}
int partition(int x[], int first, int last) {
    int pivot, temp, i, j;
    pivot = first;
    i = first;
    j = last;
    while (i < j) {
        while ((x[i] <= x[pivot]) && (i < last))
            i++;
        while (x[j] > x[pivot])
            j--;
        if (i < j) {
            temp = x[i];
            x[i] = x[j];
            x[j] = temp;
        }
    }
    temp = x[pivot];
    x[pivot] = x[j];
    x[j] = temp;
    return j;
}
int mergesort(int a[], int p, int r) {
    int q;
    if (p < r) {
        q = (p + r) / 2;
        mergesort(a, p, q);
        mergesort(a, q + 1, r);
        merge(a, p, q, r);
    }
    return 0;
}
int merge(int a[], int p, int q, int s) {
    int n1, n2, l[10], r[10], i, j, k;
    n1 = q - p + 1;
    n2 = s - q;
    for (i = 0; i < n1; i++)
        l[i] = a[p + i];
    for (j = 0; j < n2; j++)
        r[j] = a[q + j + 1];
    l[i] = 9999;
```

```
    r[j] = 9999;
    i = 0;
    j = 0;
    for (k = p; k <= s; k++) {
        if (l[i] <= r[j]) {
            a[k] = l[i];
            i = i + 1;
        } else {
            a[k] = r[j];
            j = j + 1;
        }
    }
    return 0;
}
void exchange(int * a, int * largest) {
    int temp;
    temp = * a;
    * a = * largest;
    * largest = temp;
}
void display(int b[]) {
    int i;
    for (i = 1; i <= length; i++)
        printf("\t%d", b[i]);
}

void heapify(int a[], int i) {
    int left, right, largest;
    left = 2 * i;
    right = 2 * i + 1;
    if (left <= heapsize && a[left] > a[i])
        largest = left;
    else
        largest = i;
    if (right <= heapsize && a[right] > a[largest])
        largest = right;
    if (largest != i) {
        exchange( & a[i], & a[largest]);
        heapify(a, largest);
    }
}

void buildheap(int a[]) {
    int i, j, x;
    i = 1;
    while (i <= length) {
        x = a[i];
        h[i] = x;
        j = i;
        while (j > 1) {
            if (h[j] > h[j / 2]) {
                exchange( & h[j], & h[j / 2]);
                j = j / 2;
            } else
                j = 1;
        }
        i = i + 1;
    }
}
```

```
void heapsort(int a[]) {
    int i;
    buildheap(a);
    printf("\nArray after creating heap\n");
    display(h);
    for (i = length; i >= 2; i--) {
        exchange( & h[1], & h[i]);
        heapsize = heapsize - 1;
        heapify(h, 1);
    }
}

void main() {
    int i, j, n, c, ch;
    do {
        printf("\nEnter your choice:\n\t1.Quicksort\n\t2.Mergesort.\n\t3.Heapsort\n\t4.Exit\n");
        scanf("%d", & ch);
        switch (ch) {
        case 1:
            printf("\nEnter the no. of elements in the array:\t");
            scanf("%d", & n);
            printf("\nEnter the elements into the array:\n");
            for (i = 1; i <= n; i++)
                scanf("%d", & a[i]);
            quick(a, 1, n);
            printf("\nThe sorted array using quicksort is:\n");
            for (i = 1; i <= n; i++)
                printf("%d\t", a[i]);
            break;
        case 2:
            printf("\nEnter the no of elements in the array:\t");
            scanf("%d", & n);
            printf("\nEnter the elements into the array:\n");
            for (i = 0; i < n; i++)
                scanf("%d", & b[i]);
            mergesort(b, 0, n - 1);
            printf("\nThe sorted array using mergesort is:\n");
            for (i = 0; i < n; i++)
                printf("%d\t", b[i]);
            break;
        case 3:
            printf("\nEnter the no. of elements in the array:\t");
            scanf("%d", & n);
            printf("\nEnter the elements into the array:\n");
            for (i = 1; i <= n; i++)
                scanf("%d", & a[i]);
            length = n;
            heapsize = length;
            heapsort(a);
            printf("\nThe sorted array using heapsort is:\n")
            display(h);
            break;
        case 4:
            break;
        default:
            printf("ERROR! Invalid Choice.., Try Again!!!\n");
            break;
        }
    } while (ch != 4);
}
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ gcc heap_quick_merge_sort.c
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out

Enter your choice:
        1.Quicksort
        2.Mergesort.
        3.Heapsort
        4.Exit
1

Enter the no. of elements in the array: 7

Enter the elements into the array:
11 34 56 54 89 100 43

The sorted array using quicksort is:
11      34      43      54      56      89      100
Enter your choice:
        1.Quicksort
        2.Mergesort.
        3.Heapsort
        4.Exit
2

Enter the no of elements in the array:  7

Enter the elements into the array:
11 34 56 54 89 100 43

The sorted array using mergesort is:
11      34      43      54      56      89      100
Enter your choice:
        1.Quicksort
        2.Mergesort.
        3.Heapsort
        4.Exit
3

Enter the no. of elements in the array: 7

Enter the elements into the array:
11 34 56 54 89 100 43

Array after creating heap
        100     56      89      11      54      34      43
The sorted array using heapsort is:
        11      34      43      54      56      89      100
Enter your choice:
        1.Quicksort
        2.Mergesort.
        3.Heapsort
        4.Exit
4
hp@hp-HP-Laptop-15s-du0xxx:~$
```

(17)GRAPH USING ADJACENCY LIST

BFS

```
#include<stdio.h>
#include<stdlib.h>
struct node1 {
    int data;
    struct node1 * next;
}* visit = NULL;
struct node {
    int data;
    struct node * link;
    struct node1 * alink;
}* gptr = NULL;
int front = -1, rear = -1;
int que[10];
void enqueue(int item) {
    if (rear == 10)
        printf("Queue is full");
    else {
        if (front == -1) {
            front = rear = 0;
            que[rear] = item;
        } else
            que[++rear] = item;
    }
}
int dequeue() {
    if (front == -1)
        return -1;
    else {
        int x = que[front];
        if (front == rear) {
            front = rear = -1;
        } else front++;
        return (x);
    }
}
int searchVisit(int vert) {
    struct node1 * p;
    p = visit;
    if (p == NULL)
        return 0;
    else
        while ((p != NULL) && (p -> data != vert)) {
            p = p -> next;
        }
    if (p != NULL)
        return 1;
    elsereturn 0;
}
void addVisit(int vert) {
    struct node1 * p, * new;
    p = visit;
    if (p == NULL) {
        new = malloc(sizeof(struct node1));
        new -> data = vert;
        new -> next = NULL;
        visit = new;
    } else {
```

```
        while (p -> next != NULL)
            p = p -> next;
        new = malloc(sizeof(struct node1));
        new -> data = vert;
        new -> next = NULL;
        p -> next = new;
    }
}
void bfs(struct node * gptr, int s) {
    struct node * temp;
    struct node1 * temp1;
    int u = s;
    enqueue(u);
    while (front != -1) {
        u = dequeue();
        if (searchVisit(u) == 0) {
            printf("%3d", u);
            addVisit(u);
            temp = gptr;
            while ((temp != NULL) && (temp -> data != u))
                temp = temp -> link;
            temp1 = temp -> alink;
            while (temp1 != NULL) {
                enqueue(temp1 -> data);
                temp1 = temp1 -> next;
            }
        }
    }
}
void display(struct node * g) {
    struct node * ptr;
    struct node1 * ptr1;
    ptr = g;
    while (ptr != NULL) {
        printf("\nNode : %d", ptr -> data);
        ptr1 = ptr -> alink;
        while (ptr1 != NULL) {
            printf("..%3d..", ptr1 -> data);
            ptr1 = ptr1 -> next;
        }
        ptr = ptr -> link;
    }
}
void main() {
    int m, n, opt, yes;
    struct node * temp, * temp1;
    struct node1 * ptr, * ptr1, * ptr2;
    int start;
    printf("\nHow many nodes?");
    scanf("%d", & n);
    for (int i = 1; i <= n; i++) {
        temp = malloc(sizeof(struct node));
        printf("Enter the vertex in the graph:");
        scanf("%d", & temp -> data);
        if (gptr == NULL) {
            gptr = temp;
            temp1 = temp;
        } else {
            temp1 -> link = temp;
            temp1 = temp;
```

```
        }
        temp -> link = NULL;
        printf("\n Any edges starting from vertex [1/0] ");
        scanf("%d", & yes);
        if (yes == 1) {
            printf("\n Enter the nodes with which this node shares an edge :");
            ptr1 = NULL;
            do {
                ptr = malloc(sizeof(struct node1));
                printf("\nEnter the vertex:");
                scanf("%d", & ptr -> data);
                if (ptr1 == NULL) {
                    ptr1 = ptr;
                    ptr2 = ptr;
                } else {
                    ptr2 -> next = ptr;
                    ptr2 = ptr;
                }
                printf("\nAnymore adjascent nodes??[1/0] :");
                scanf("%d", & opt);
            } while (opt == 1);
            ptr2 -> next = NULL;
            temp -> alink = ptr1;
        } else
            temp -> alink = NULL;
    }
    printf("\n Enter the start vertex :");
    scanf("%d", & start);
    bfs(gptr, start);
}
```

DFS

```
#include<stdio.h>
#include<stdlib.h>
struct node1 {
    int data;
    struct node1 * next;
}* visit = NULL;

struct node {
    int data;
    struct node * link;
    struct node1 * alink;
}* gptr = NULL;
int top = -1;
int stack[10];

void push(int item) {
    if (top == 10) {
        printf("Stack is full");
    } else {
        stack[++top] = item;
    }
}

int pop() {
    if (top == -1) {
        return -1;
    } else {
```

```
        return (stack[top--]);
    }
}
int searchVisit(int vert) {
    struct node1 * p;
    p = visit;
    if (p == NULL)
        return 0;
    else
        while ((p != NULL) && (p -> data != vert)) {
            p = p -> next;
        }
    if (p != NULL)
        return 1;
    else
        return 0;
}

void addVisit(int vert) {
    struct node1 * p, * new;
    p = visit;
    if (p == NULL) {
        new = malloc(sizeof(struct node1));
        new -> data = vert;
        new -> next = NULL;
        visit = new;
    } else {
        while (p -> next != NULL)
            p = p -> next;
        new = malloc(sizeof(struct node1));
        new -> data = vert;
        new -> next = NULL;
        p -> next = new;
    }
}

void dfs(struct node * gptr, int s) {
    struct node * temp;
    struct node1 * temp1;
    int u = s;
    push(u);
    while (top != -1) {
        u = pop();
        if (searchVisit(u) == 0) {
            printf("%3d", u);
            addVisit(u);
            temp = gptr;
            while ((temp != NULL) && (temp -> data != u))
                temp = temp -> link;
            temp1 = temp -> alink;
            while (temp1 != NULL) {
                push(temp1 -> data);
                temp1 = temp1 -> next;
            }
        }
    }
}

void display(struct node * g) {
    struct node * ptr;
```

```
    struct node1 * ptr1;
    ptr = g;
    while (ptr != NULL) {
        printf("\nNode : %d", ptr -> data);
        ptr1 = ptr -> alink;
        while (ptr1 != NULL) {
            printf("..%3d..", ptr1 -> data);
            ptr1 = ptr1 -> next;
        }
        ptr = ptr -> link;
    }
}
void main() {
    int m, n, opt, yes;
    struct node * temp, * temp1;
    struct node1 * ptr, * ptr1, * ptr2;
    int start;
    printf("\nHow many nodes?");
    scanf("%d", & n);
    for (int i = 1; i <= n; i++) {
        temp = malloc(sizeof(struct node));
        printf("Enter the vertex in the graph:");
        scanf("%d", & temp -> data);
        if (gptr == NULL) {
            gptr = temp;
            temp1 = temp;
        } else {
            temp1 -> link = temp;
            temp1 = temp;
        }
        temp -> link = NULL;
        printf("\n Any edges starting from vertex[1/0] ");
        scanf("%d", & yes);
        if (yes == 1) {
            printf("\n Enter the nodes with which this node shares an edge :");
            ptr1 = NULL;
            do {
                ptr = malloc(sizeof(struct node1));
                printf("\nEnter the vertex:");
                scanf("%d", & ptr -> data);
                if (ptr1 == NULL) {
                    ptr1 = ptr;
                    ptr2 = ptr;
                } else {
                    ptr2 -> next = ptr;
                    ptr2 = ptr;
                }
                printf("\nAnymore adjascent nodes??[1/0] :");
                scanf("%d", & opt);
            } while (opt == 1);
            ptr2 -> next = NULL;
            temp -> alink = ptr1;
        } else
            temp -> alink = NULL;
    }
    printf("\n Enter the start vertex :");
    scanf("%d", & start);
    dfs(gptr, start);
}
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ gcc graphdfs.c
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out

How many nodes?5
Enter the vertex in the graph:0

 Any edges starting from vertex[1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:1

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:1

 Any edges starting from vertex[1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:2

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:2

 Any edges starting from vertex[1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:4

Anymore adjascent nodes??[1/0] :1

Enter the vertex:3

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:4

 Any edges starting from vertex[1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:2

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:3

 Any edges starting from vertex[1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:0

Anymore adjascent nodes??[1/0] :0

 Enter the start vertex :0
  0  1  2  3  4hp@hp-HP-Laptop-15s-du0xxx:~$
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ gcc graphbfs.c
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out

How many nodes?5
Enter the vertex in the graph:0

 Any edges starting from vertex [1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:1

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:1

 Any edges starting from vertex [1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:2

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:2

 Any edges starting from vertex [1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:4

Anymore adjascent nodes??[1/0] :1

Enter the vertex:3

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:4

 Any edges starting from vertex [1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:2

Anymore adjascent nodes??[1/0] :0
Enter the vertex in the graph:3

 Any edges starting from vertex [1/0] 1

 Enter the nodes with which this node shares an edge :
Enter the vertex:0

Anymore adjascent nodes??[1/0] :0

 Enter the start vertex :0
  0  1  2  4  3hp@hp-HP-Laptop-15s-du0xxx:~$
```

(18) HASH TABLE USING CHAINING MEHOD

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
struct node {
    int data;
    struct node * next;
};
struct node * head[TABLE_SIZE] = {
    NULL
}, * c;
void insert() {
    int i, key;
    printf("Enter a value to insert into hash table :");
    scanf("%d", & key);
    i = key % TABLE_SIZE;
    struct node * newnode = (struct node * ) malloc(sizeof(struct node));
    newnode -> data = key;
    newnode -> next = NULL;
    if (head[i] == NULL)
        head[i] = newnode;
    else {
        c = head[i];
        while (c -> next != NULL) {
            c = c -> next;
        }
        c -> next = newnode;
    }
}
void search() {
    int key, index;
    printf("Enter the element to be searched :");
    scanf("%d", & key);
    index = key % TABLE_SIZE;
    if (head[index] == NULL)
        printf("Search element not found\n");
    else {
        for (c = head[index]; c != NULL; c = c -> next) {
            if (c -> data == key) {
                printf("search element found\n");
                break;
            }
        }
        if (c == NULL)
            printf("Search element not found\n");
    }
}
void display() {
    int i;
    for (i = 0; i < TABLE_SIZE; i++) {
        printf("\nEntries at index %d\n", i);
        if (head[i] == NULL) {
            printf("No Hash Entry");
        } else {
            for (c = head[i]; c != NULL; c = c -> next) printf("%d->", c -> data);
        }
    }
}
void main() {
    int i, choice;
```

```
    printf("\nIMPLEMENTATION OF HASH TABLE USING CHAINING METHOD FOR COLLISION
RESOLUTION\n");
    printf("1:Insert\n 2:Display\n 3:Search\n 4:Exit \n");
    while (choice != 4) {
        switch (choice) {
        case 1:
            insert();
            break;
        case 2:
            display();
            break;
        case 3:
            search();
            break;
        case 4:
            break;
        }
        printf("Enter your choice :");
        scanf("%d", & choice);
    }
}
```

```
File 4:31

File_no         File_size       Block_no        Block_size      Fragment
1               22              2               25              3
2               10              3               12              2
3               42              6               45              3
4               31              5               35              4
Enter your choice       3

Enter the number of blocks:7
Enter the number of files:4

Enter the size of the blocks:-
Block 1:60
Block 2:25
Block 3:12
Block 4:20
Block 5:35
Block 6:45
Block 7:40
Enter the size of the files:-
File 1:22
File 2:10
File 3:42
File 4:31

File_no         File_size       Block_no        Block_size      Fragment
1               22              1               60              38
2               10              6               45              35
3               42              0               -449664976              0
4               31              7               40              9
Enter your choice       4
hp@hp-HP-Laptop-15s-du0xxx:~$
```

(19) HASH TABLE USING LINEAR PROBING COLLISION RES

```
#include<stdio.h>
int H[10];
void Insert(int key);
void Display();
void Search(int key);
void Delete(int key);
void main() {
    int x, key, i;
    for (i = 0; i < 10; i++)
        H[i] = -1;
    printf("Choices are\n1)Insertion\n2)Display\n3)Search\n4)Deletion\n5)Exit\n");
    while (x != 5) {
        printf("Enter your choice : ");
        scanf("%d", & x);
        switch (x) {
        case 1:
            printf("Enter the element to be inserted : ");
            scanf("%d", & key);
            Insert(key);
            break;
        case 2:
            printf("Hash Table\n");
            Display();
            break;
        case 3:
            printf("Enter the key to be searched : ");
            scanf("%d", & key);
            Search(key);
            break;
        case 4:
            printf("Enter the key to be deleted : ");
            scanf("%d", & key);
            Delete(key);
            break;
        case 5:
            printf("Exit\n");
            break;
        default:
            printf("Invelid choice\n");
            break;
        }
    }
}
void Insert(int key) {
    int i, j;
    i = key % 10;
    if (H[i] == -1) {
        H[i] = key;
    } else {
        for (j = (i + 1) % 10; j != i; j = (j + 1) % 10) {
            if (H[j] == -1) {
                H[j] = key;
                break;
            }
        }
        if (j == i) {
            printf("The table is overflow\n");
        }
    }
```

```
}
void Display() {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d ", i);
        if (H[i] != -1)
            printf("%d", H[i]);
        printf("\n");
    }
}
void Search(int key) {
    int i, j, insert;
    i = key % 10;
    if (H[i] == key) {
        printf("Key is found at the index %d\n", i);
        return;
    } else {
        j = (i + 1) % 10;
        while (j != i) {
            if (H[j] == key) {
                printf("Key is found at the index %d\n", j);
                return;
            } else {
                j = (j + 1) % 10;
            }
        }
        if (j == i) {
            printf("Key value does not exist\n");
            printf("Do you want to insert this key(Give option 1/0) ?");
            scanf("%d", & insert);
            if (insert == 1) {
                Insert(key);
            }
        }
    }
}
void Delete(int key) {
    int i, j;
    i = key % 10;
    if (H[i] == key) {
        H[i] = -1;
        return;
    } else {
        j = i + 1;
        while (j != i) {
            if (H[j] == -1) {
                printf("Key does not exist : No deletion\n");
                return;
            } else if (H[j] == key) {
                H[j] = -1;
                return;
            } else {
                j = (j + 1) % 10;
            }
        }
        if (j == i) {
            printf("Key does not exist : No deletion\n");
        }
    }
}
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out
Choices are
1)Insertion
2)Display
3)Search
4)Deletion
5)Exit
Enter your choice : 1
Enter the element to be inserted : 10
Enter your choice : 1
Enter the element to be inserted : 16
Enter your choice : 1
Enter the element to be inserted : 11
Enter your choice : 1
Enter the element to be inserted : 1
Enter your choice : 1
Enter the element to be inserted : 3
Enter your choice : 1
Enter the element to be inserted : 4
Enter your choice : 1
Enter the element to be inserted : 23
Enter your choice : 1
Enter the element to be inserted : 15
Enter your choice : 2
Hash Table
0 10
1 11
2 1
3 3
4 4
5 23
6 16
7 15
8
9
Enter your choice : 3
Enter the key to be searched : 11
Key is found at the index 1
Enter your choice : 4
Enter the key to be deleted : 1
Enter your choice : 2
Hash Table
0 10
1 11
2
3 3
4 4
5 23
6 16
7 15
8
9
Enter your choice : 4
Enter the key to be deleted :
```

(20) DYNAMIC MEMORY ALLOCATION AND DEALLOCATION

```
#include<stdio.h>
#define max 25
void FirstFit();
void BestFit();
void WorstFit();
void makeallo() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];
    printf("\nEnter the number of blocks:");
    scanf("%d", & nb);
    printf("Enter the number of files:");
    scanf("%d", & nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++) {
        printf("Block %d:", i);
        scanf("%d", & b[i]);
    }
    printf("Enter the size of the files:-\n");
    for (i = 1; i <= nf; i++) {
        printf("File %d:", i);
        scanf("%d", & f[i]);
    }
}

void FirstFit() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];
    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0) {
                    ff[i] = j;
                    break;
                }
            }
        }
        frag[i] = temp;
        bf[ff[i]] = 1;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void BestFit() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
    static int bf[max], ff[max];
    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0)
                    if (lowest > temp) {
                        ff[i] = j;
                        lowest = temp;
                    }
            }
        }
```

```
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile_no \tFile_size \tBlock_no \tBlock_size \tFragment");
    for (i = 1; i <= nf && ff[i] != 0; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void WorstFit() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
    static int bf[max], ff[max];
    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) //if bf[j] is not allocated
            {
                temp = b[j] - f[i];
                if (temp >= 0)
                    if (highest < temp) {
                        ff[i] = j;
                        highest = temp;
                    }
            }
        }
        frag[i] = highest;
        bf[ff[i]] = 1;
        highest = 0;
    }
    printf("\nFile_no \tFile_size \tBlock_no \tBlock_size \tFragment");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void main() {
    int choice;
    printf("\nVariable Sized Dynamic Memory Allocation \n");
    printf("\nChoices:\n 1:FIRST FIT\n 2:BEST FIT\n 3:WORST FIT\n 4:EXIT\n\n");
    do {
        printf("\nEnter your choice\t");
        scanf("%d", & choice);
        switch (choice) {
        case 1:
            makeallo();
            FirstFit();
            break;
        case 2:
            makeallo();
            BestFit();
            break;
        case 3:
            makeallo();
            WorstFit();
            break;
        case 4:
            break;
        default:
            printf("\nInvalid Choice\n");
        }
    } while (choice != 4);
}
```

```
hp@hp-HP-Laptop-15s-du0xxx:~$ gcc VariableBlockAllo.c
hp@hp-HP-Laptop-15s-du0xxx:~$ ./a.out

Variable Sized Dynamic Memory Allocation

Choices:
 1:FIRST FIT
 2:BEST FIT
 3:WORST FIT
 4:EXIT


Enter your choice       1

Enter the number of blocks:7
Enter the number of files:4

Enter the size of the blocks:-
Block 1:60
Block 2:25
Block 3:12
Block 4:20
Block 5:35
Block 6:45
Block 7:40
Enter the size of the files:-
File 1:22
File 2:10
File 3:42
File 4:31

File_no:        File_size :     Block_no:       Block_size:     Fragment
1               22              1               60              38
2               10              2               25              15
3               42              6               45              3
4               31              5               35              4
Enter your choice       2

Enter the number of blocks:7
Enter the number of files:4

Enter the size of the blocks:-
Block 1:60
Block 2:25
Block 3:12
Block 4:20
Block 5:35
Block 6:45
Block 7:40
Enter the size of the files:-
File 1:22
File 2:10
File 3:42
```