INDEX

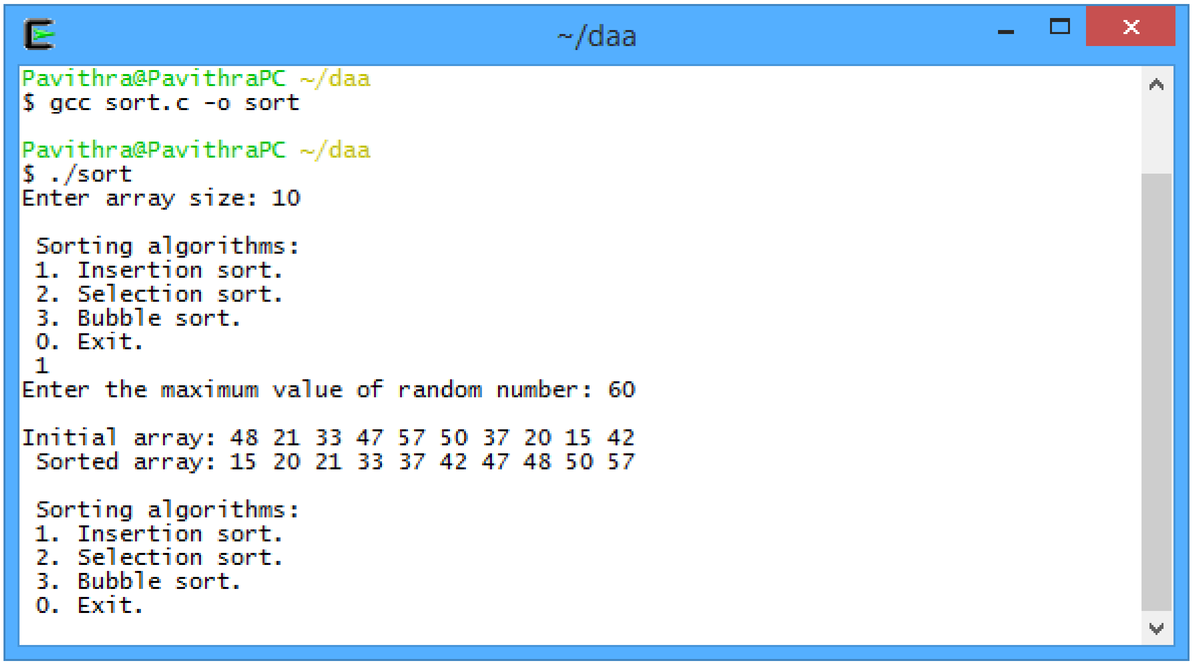
18	Knapsack		
19	Shortest distance from source to destination by using dynamic programing		
20	String matching by Naïve		
21	String matching by Knuth Morris Pratt		
22	String matching by automata		
23	String matching by Rabin Karp		
24	N Queen		
25	Hamiltonian path problem		

1,2,3. Insertion sort, Bubble sort and Selection sort.

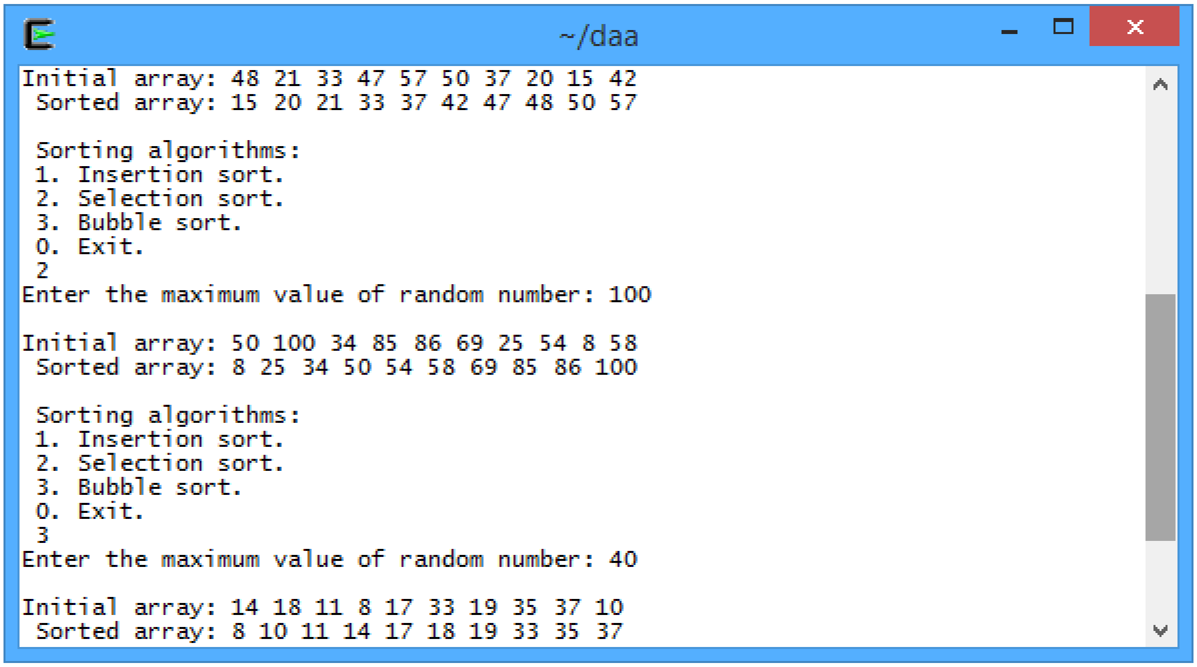
```
/*WAP to implement Insertion sort, Bubble sort and Insertion sort*/
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
//Swap two numbers
void swap(int *num1,int *num2)
{
	int temp;
	temp=*num1;
	*num1=*num2;
	*num2=temp;
}
//Initialize array
void initializeArray(int *arr,int size)
{
	int i,max,num;
	printf("Enter the maximum value of random number: ");
	scanf("%d",&max);
	srand(time(NULL));
	for ( i = 0 ; i <size ; i++ )
	{
		num = rand()%max+1;
		arr[i]=num;
	}

}
//Print an array
void printArray(int *arr,int size)
{
	int i;
	for(i=0; i<size;i++)
	{
		printf("%d ",arr[i]);
	}
}
//Implementation of insertion sort
void insertionSort(int *arr,int size)
{
	int i,j;
	for(i=1;i<size;i++)
	{
		j=i;
		while(j>0 && arr[j]<arr[j-1])
		{
			swap(&arr[j],&arr[j-1]);
			j--;
		}
	}
}
//Implementation of bubble sort
void bubbleSort(int *arr,int size)
{
	int i,j;
```

```
       for(i=1;i< size;i++)
    {
         for(j=0;j< size-1;j++)
              {
                    if(arr[j]>arr[j+1])
            {
               swap(&arr[j],&arr[j+1]);
            }
             }
       }
}
//Implementation of selection sort
void selectionSort(int *arr,int size)
{
       int i,minimum,j;
       for(i=0; i<size; i++)
       {
             minimum=i;
             for(j=i;j<size;j++)
             {
                    if(arr[minimum]>arr[j])
                    {
                          minimum=j;
                    }
             }
             swap(&arr[i],&arr[minimum]);
       }
}

void main()
{
       int *arr;
       int max,i,num,noOfElements,choice;
       printf("Enter array size: ");
       scanf("%d",&noOfElements);
       arr=(int *)malloc(noOfElements*sizeof(int));

       do
       {
             printf("\n Sorting algorithms: ");
             printf("\n 1. Insertion sort. ");
             printf("\n 2. Selection sort. ");
             printf("\n 3. Bubble sort. ");
             printf("\n 0. Exit.\n ");
             scanf("%d",&choice);
             switch(choice)
             {
                    case 1:      initializeArray(arr,noOfElements);
                                 printf("\nInitial array: ");
                                 printArray(arr,noOfElements);
                                 insertionSort(arr,noOfElements);
                                 printf("\n Sorted array: ");
                                 printArray(arr,noOfElements);
                                 printf("\n");
                                 break;
                    case 2:      initializeArray(arr,noOfElements);
                                 printf("\nInitial array: ");
                                 printArray(arr,noOfElements);
                                 selectionSort(arr,noOfElements);
                                 printf("\n Sorted array: ");
                                 printArray(arr,noOfElements);
                                 printf("\n");
                                 break;
                    case 3:      initializeArray(arr,noOfElements);
                                 printf("\nInitial array: ");
                                 printArray(arr,noOfElements);
                                 bubbleSort(arr,noOfElements);
                                 printf("\n Sorted array: ");
                                 printArray(arr,noOfElements);
                                 printf("\n");
                                 break;
                    case 0: printf("\n Exiting");
                                 break;
                    default:
                                 break;
              }

       }while(choice!=0 && choice <=3);


}
```

OUTPUT


~/daa

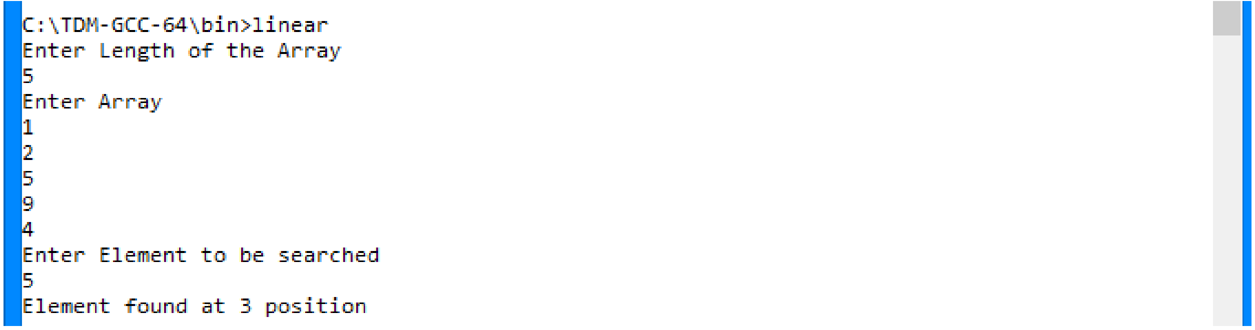
```
Pavithra@PavithraPC ~/daa
$ gcc sort.c -o sort

Pavithra@PavithraPC ~/daa
$ ./sort
Enter array size: 10

 Sorting algorithms:
 1. Insertion sort.
 2. Selection sort.
 3. Bubble sort.
 0. Exit.
 1
Enter the maximum value of random number: 60

Initial array: 48 21 33 47 57 50 37 20 15 42
 Sorted array: 15 20 21 33 37 42 47 48 50 57

 Sorting algorithms:
 1. Insertion sort.
 2. Selection sort.
 3. Bubble sort.
 0. Exit.
```


~/daa

```
Initial array: 48 21 33 47 57 50 37 20 15 42
 Sorted array: 15 20 21 33 37 42 47 48 50 57

 Sorting algorithms:
 1. Insertion sort.
 2. Selection sort.
 3. Bubble sort.
 0. Exit.
 2
Enter the maximum value of random number: 100

Initial array: 50 100 34 85 86 69 25 54 8 58
 Sorted array: 8 25 34 50 54 58 69 85 86 100

 Sorting algorithms:
 1. Insertion sort.
 2. Selection sort.
 3. Bubble sort.
 0. Exit.
 3
Enter the maximum value of random number: 40

Initial array: 14 18 11 8 17 33 19 35 37 10
 Sorted array: 8 10 11 14 17 18 19 33 35 37
```

4. Linear search

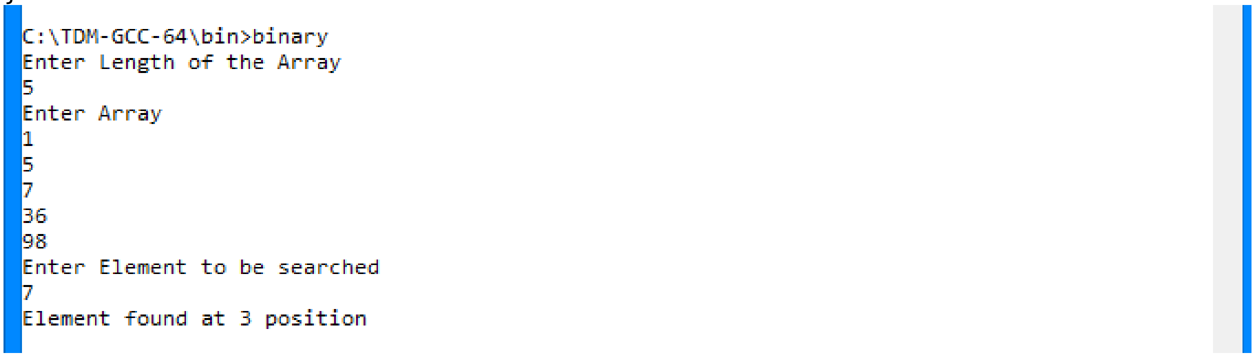
```
#include<stdio.h>
#define MAX 10

int Linear(int arr[],int n,int key)
{
 int i,pos=-1, j;
 for(i=0;i<n;i++)
 {
   if(arr[i]==key)
   {
       pos=i;
   }
 }
 return pos;
}
int main(void)
{
 int arr[MAX], n,i,key,pos;
 printf("Enter Length of the Array\n");
 scanf("%d",&n);
 printf("Enter Array \n");
 for(i=0;i<n;i++)
  scanf("%d",&arr[i]);
 printf("Enter Element to be searched\n");
 scanf("%d",&key);
 pos=Linear(arr,n,key);
 if(pos==-1)
      printf("Element not present in the list\n");
 else
      printf("Element found at %d position\n",pos+1);
}
```



```
C:\TDM-GCC-64\bin>linear
Enter Length of the Array
5
Enter Array
1
2
5
9
4
Enter Element to be searched
5
Element found at 3 position
```

5. Binary search

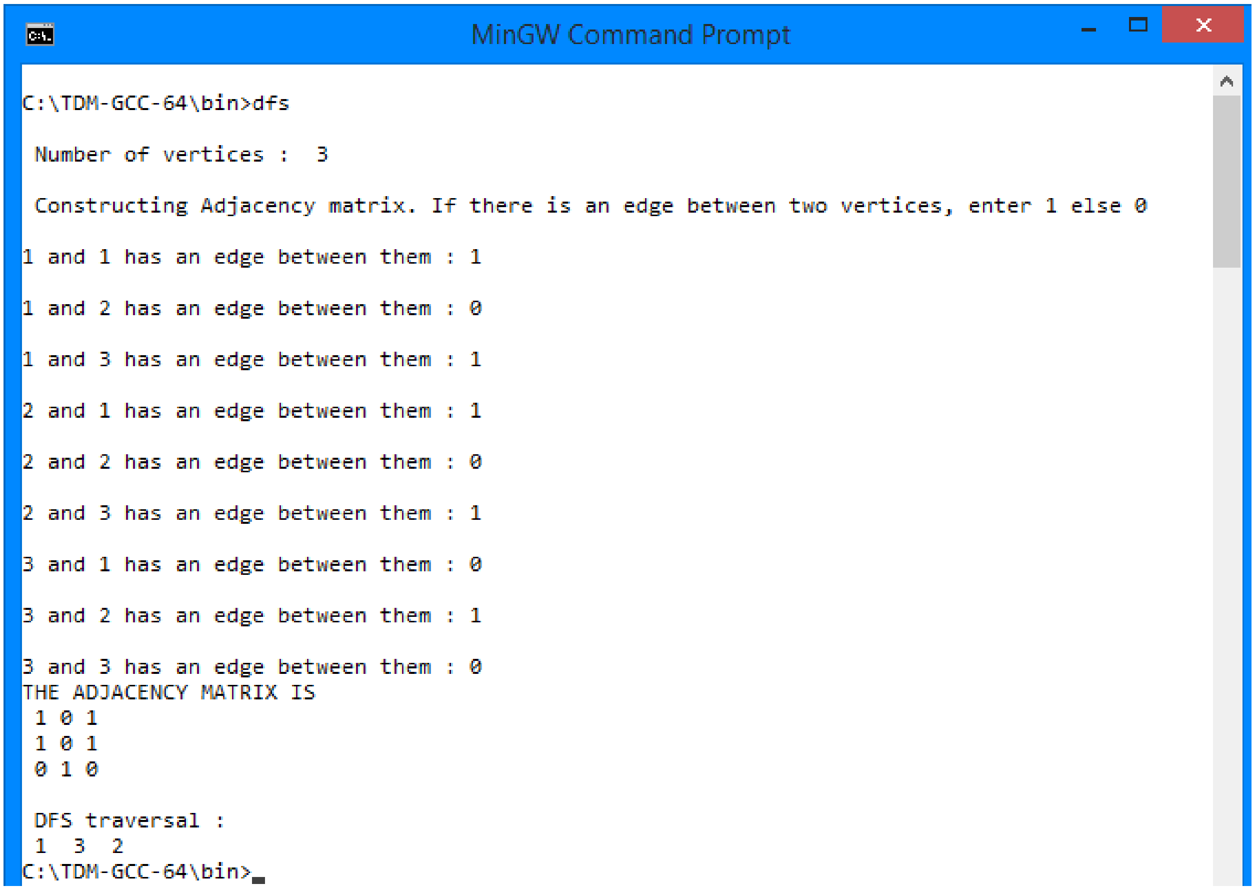
```
#include<stdio.h>
#define MAX 10

int Binary(int arr[],int n,int key)
{
 int i,pos=-1, HIGH=n-1, LOW=0, MID;
 for(i=0;i<n;i++)
 {      MID=(HIGH+LOW)/2;

       if(key==arr[MID])
       {
             pos=MID;
             break;
       }
       else if(key<=arr[MID])
             HIGH=MID-1;
       else
             LOW=MID+1;
 }
 return pos;
}

Int main()
{
 int arr[MAX], n,i,key,pos;
 printf("Enter Length of the Array\n");
 scanf("%d",&n);

 printf("Enter Array \n");
 for(i=0;i<n;i++)
  scanf("%d",&arr[i]);

 printf("Enter Element to be searched\n");
 scanf("%d",&key);

 pos=Binary(arr,n,key);

 if(pos==-1)
       printf("Element not present in the list\n");
 else
       printf("Element found at %d position\n",pos+1);
return 0;
}
```



```
C:\TDM-GCC-64\bin>binary
Enter Length of the Array
5
Enter Array
1
5
7
36
98
Enter Element to be searched
7
Element found at 3 position
```

6. DFS

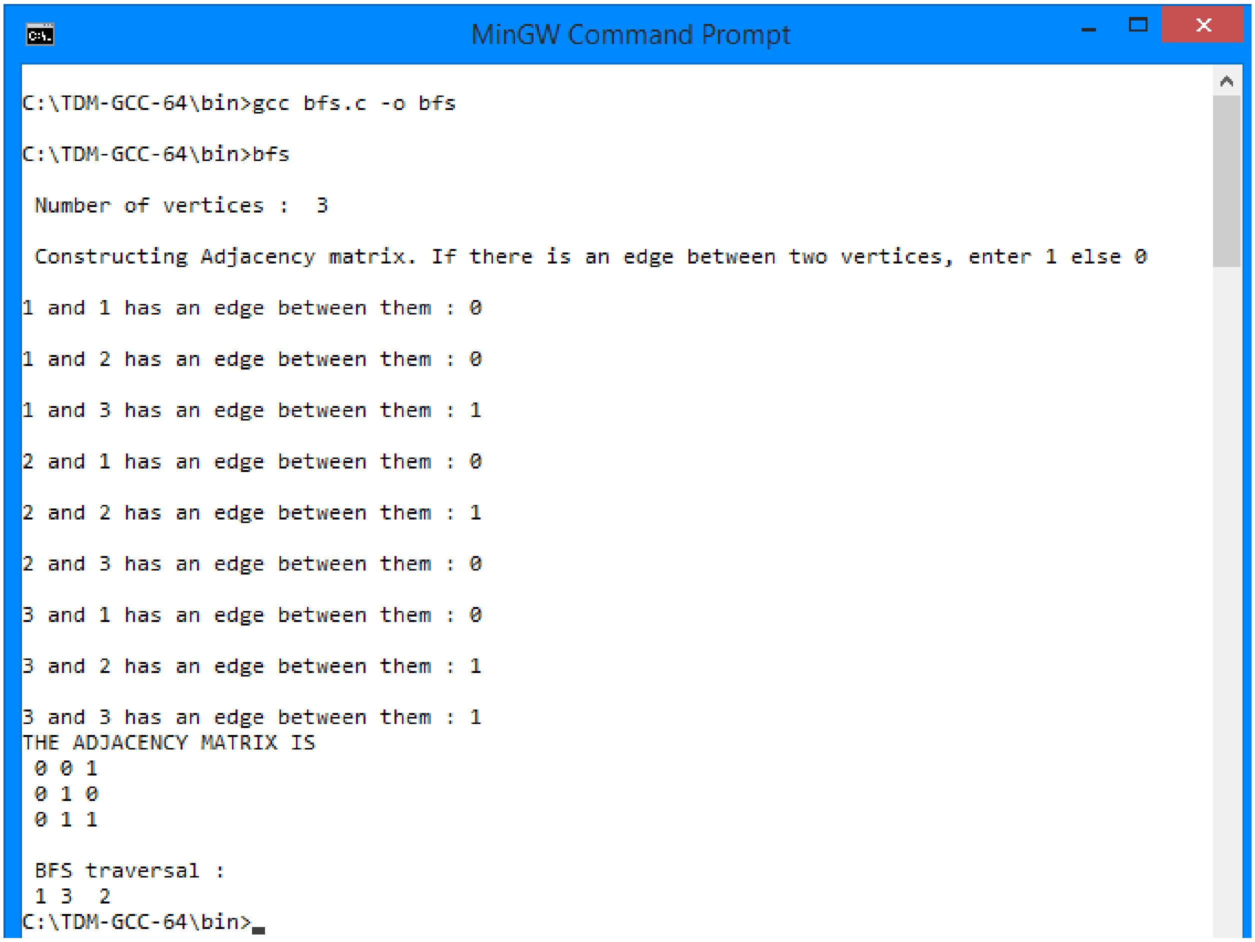
```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void push(int item);
int pop();

void dfs(int s, int n)
{
	int i, k;
	push(s);
	vis[s] = 1;
	k = pop();

	if (k != 0)
		printf(" %d ", k);

	while (k != 0)
	{
		for (i = 1; i <= n; i++)
		if ((a[k][i] != 0) && (vis[i] == 0))
		{
			push(i);
			vis[i] = 1;
		}

		k = pop();
		if (k != 0)
			printf(" %d ", k);
	}
	for (i = 1; i <= n; i++)
		if (vis[i] == 0)
			dfs(i, n);
}

void push(int item)
{
	if (top == 19)
		printf("Stack overflow ");
	else
		stack[++top] = item;
}
int pop()
{
	int k;
	if (top == -1)
		return (0);
	else
	{
		k = stack[top--];
		return (k);
	}
}

int main()
{
    int n,i,s,ch,j;
    char c,dummy;

    printf("\n Number of vertices :  ");
    scanf("%d",&n);

      printf("\n Constructing Adjacency matrix. If there is an edge between two
vertices, enter 1 else 0 \n");

    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("\n%d and %d has an edge between them : ",i,j);
            scanf("%d",&a[i][j]);
        }
    }

    printf("THE ADJACENCY MATRIX IS\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf(" %d",a[i][j]);
        }
        printf("\n");
    }

      printf("\n DFS traversal : \n");

      dfs(s,n);

      return 0;
}
```



```
C:\TDM-GCC-64\bin>dfs

 Number of vertices :  3

 Constructing Adjacency matrix. If there is an edge between two vertices, enter 1 else 0

1 and 1 has an edge between them : 1

1 and 2 has an edge between them : 0

1 and 3 has an edge between them : 1

2 and 1 has an edge between them : 1

2 and 2 has an edge between them : 0

2 and 3 has an edge between them : 1

3 and 1 has an edge between them : 0

3 and 2 has an edge between them : 1

3 and 3 has an edge between them : 0
THE ADJACENCY MATRIX IS
 1 0 1
 1 0 1
 0 1 0

 DFS traversal :
 1  3  2
C:\TDM-GCC-64\bin>
```

7. BFS

```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void push(int item);
int pop();

void bfs(int s, int n)
{
	int p, i;
	add(s);
	vis[s] = 1;
	p = delete();
	if (p != 0)
		printf(" %d", p);

	while (p != 0)
	{
		for (i = 1; i <= n; i++)
			if ((a[p][i] != 0) && (vis[i] == 0))
			{
				add(i);
				vis[i] = 1;
			}

		p = delete();

		if (p != 0)
			printf(" %d ", p);
	}

	for (i = 1; i <= n; i++)
		if (vis[i] == 0)
			bfs(i, n);
}


void add(int item)
{
	if (rear == 19)
		printf("QUEUE FULL");
	else
	{
		if (rear == -1)
		{
			q[++rear] = item;
			front++;
		}
		else
			q[++rear] = item;
	}
}

int delete()
{
       int k;
       if ((front > rear) || (front == -1))
             return (0);
       else
       {
             k = q[front++];
             return (k);
       }
}

int main()
{
    int n,i,s,ch,j;
    char c,dummy;

    printf("\n Number of vertices :  ");
    scanf("%d",&n);

       printf("\n Constructing Adjacency matrix. If there is an edge between two
vertices, enter 1 else 0 \n");

    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("\n%d and %d has an edge between them : ",i,j);
            scanf("%d",&a[i][j]);
        }
    }

    printf("THE ADJACENCY MATRIX IS\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf(" %d",a[i][j]);
        }
        printf("\n");
    }

       printf("\n BFS traversal : \n");

       bfs(s,n);

       return 0;
}
```

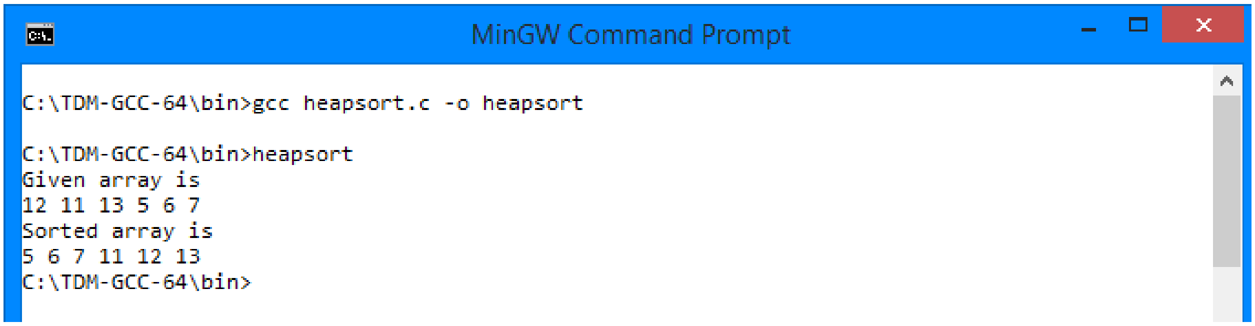


```
C:\TDM-GCC-64\bin>gcc bfs.c -o bfs

C:\TDM-GCC-64\bin>bfs

 Number of vertices :  3

 Constructing Adjacency matrix. If there is an edge between two vertices, enter 1 else 0

1 and 1 has an edge between them : 0

1 and 2 has an edge between them : 0

1 and 3 has an edge between them : 1

2 and 1 has an edge between them : 0

2 and 2 has an edge between them : 1

2 and 3 has an edge between them : 0

3 and 1 has an edge between them : 0

3 and 2 has an edge between them : 1

3 and 3 has an edge between them : 1
THE ADJACENCY MATRIX IS
 0 0 1
 0 1 0
 0 1 1

 BFS traversal :
 1 3  2
C:\TDM-GCC-64\bin>
```

8. Heap and heap sort

```
#include <stdio.h>
#include <stdlib.h>

typedef struct MaxHeap MaxHeap;

struct MaxHeap
{
    int size;
    int* array;
};

void swap(int* a, int* b)
{
       int temp;
       temp = *a;
       *a = *b;
       *b = temp;
}

void maxHeapify(MaxHeap* maxHeap, int index)
{
    int largest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    if ((left < maxHeap -> size) && (maxHeap -> array[left] > maxHeap ->
array[largest]))
        largest = left;

    if ((right < maxHeap -> size) && (maxHeap -> array[right] > maxHeap ->
array[largest]))
        largest = right;

    if (largest != index)
    {
        swap(&(maxHeap->array[largest]), &(maxHeap->array[index]));
        maxHeapify(maxHeap, largest);
    }
}

MaxHeap* createAndBuildHeap(int *array, int size)
{
    int index;
    MaxHeap* maxHeap = (MaxHeap*) malloc(sizeof(MaxHeap));

       maxHeap -> size = size;
    maxHeap -> array = array;

    for (index = ((maxHeap -> size) - 2) / 2; index >= 0; --index)
        maxHeapify(maxHeap, index);

       return maxHeap;
}
```

```
void heapSort(int* array, int size)
{
    MaxHeap* maxHeap = createAndBuildHeap(array, size);

    while (maxHeap -> size > 1)
    {
        swap(&(maxHeap -> array[0]), &(maxHeap -> array[maxHeap -> size - 1]));
        --(maxHeap -> size);

        maxHeapify(maxHeap, 0);
    }
}

void printArray(int* arr, int size)
{
    int index;
    for (index = 0; index < size; ++index)
        printf("%d ", arr[index]);
}

int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, size);

    heapSort(arr, size);

    printf("\nSorted array is \n");
    printArray(arr, size);
    return 0;
}
```

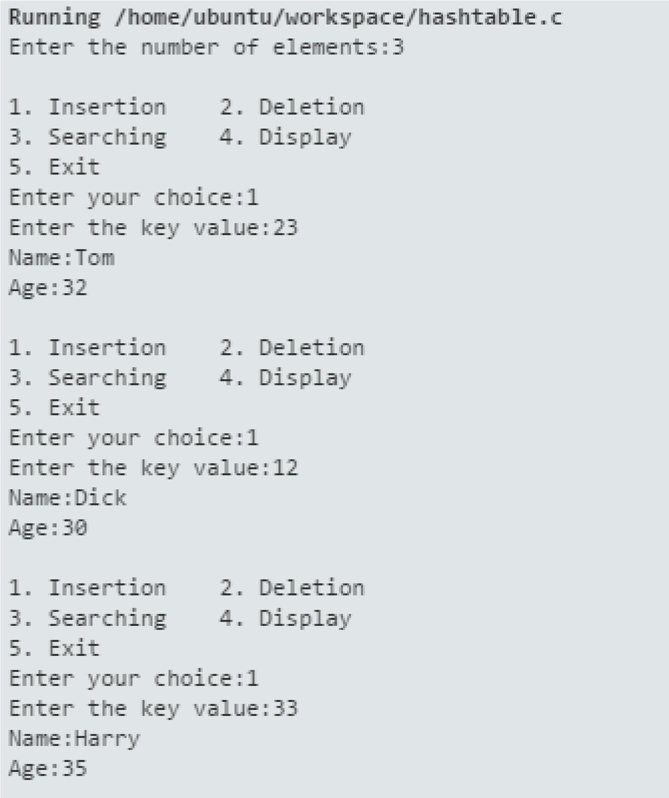


```
C:\TDM-GCC-64\bin>gcc heapsort.c -o heapsort

C:\TDM-GCC-64\bin>heapsort
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
C:\TDM-GCC-64\bin>
```

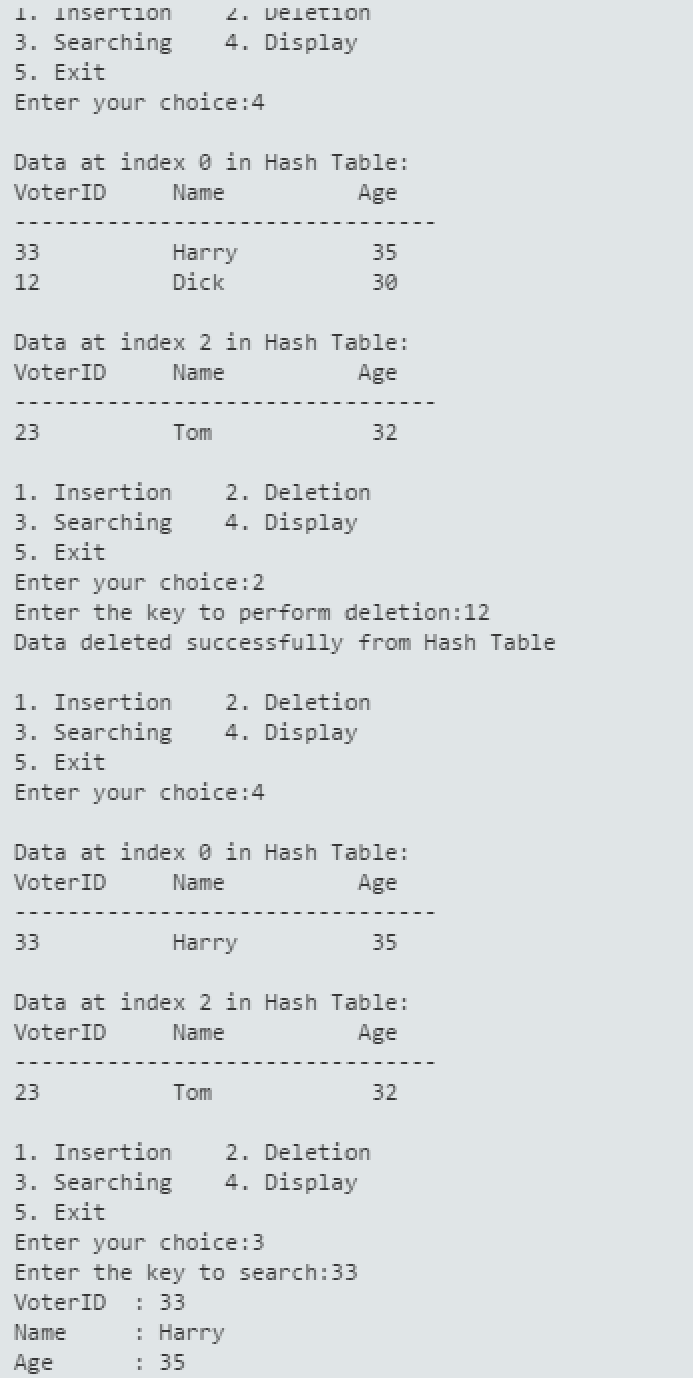
9. Hash table

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct hash *hashTable = NULL;
int eleCount = 0;
struct node {
    int key, age;
    char name[100];
    struct node *next;
};
struct hash {
    struct node *head;
    int count;
};
struct node * createNode(int key, char *name, int age) {
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->key = key;
    newnode->age = age;
    strcpy(newnode->name, name);
    newnode->next = NULL;
    return newnode;
}
void insertToHash(int key, char *name, int age) {
    int hashIndex = key % eleCount;
    struct node *newnode =  createNode(key, name, age);
    /* head of list for the bucket with index "hashIndex" */
    if (!hashTable[hashIndex].head) {
            hashTable[hashIndex].head = newnode;
            hashTable[hashIndex].count = 1;
            return;
    }
    /* adding new node to the list */
    newnode->next = (hashTable[hashIndex].head);
    /*
     * update the head of the list and no of
     * nodes in the current bucket
     */
    hashTable[hashIndex].head = newnode;
    hashTable[hashIndex].count++;
    return;
}
void deleteFromHash(int key) {
    /* find the bucket using hash index */
    int hashIndex = key % eleCount, flag = 0;
    struct node *temp, *myNode;
    /* get the list head from current bucket */
    myNode = hashTable[hashIndex].head;
    if (!myNode) {
            printf("Given data is not present in hash Table!!\n");
            return;   }
    temp = myNode;
    while (myNode != NULL) {
            /* delete the node with given key */
            if (myNode->key == key) {
                    flag = 1;
                    if (myNode == hashTable[hashIndex].head)
                            hashTable[hashIndex].head = myNode->next;
                    else
                            temp->next = myNode->next;
                    hashTable[hashIndex].count--;
                    free(myNode);
                    break;    }
            temp = myNode;
            myNode = myNode->next;
    }
    if (flag)
            printf("Data deleted successfully from Hash Table\n");
    else
            printf("Given data is not present in hash Table!!!!\n");
    return;
}
void searchInHash(int key) {
    int hashIndex = key % eleCount, flag = 0;
    struct node *myNode;
    myNode = hashTable[hashIndex].head;
    if (!myNode) {
            printf("Search element unavailable in hash table\n");
            return;
    }
   while (myNode != NULL) {
            if (myNode->key == key) {
                    printf("VoterID  : %d\n", myNode->key);
                    printf("Name     : %s\n", myNode->name);
                    printf("Age      : %d\n", myNode->age);
                    flag = 1;
                    break;   }
            myNode = myNode->next;
    }
    if (!flag)
            printf("Search element unavailable in hash table\n");
    return;
}
void display() {
    struct node *myNode;
    int i;
    for (i = 0; i < eleCount; i++) {
            if (hashTable[i].count == 0)
                    continue;
            myNode = hashTable[i].head;
            if (!myNode)
                    continue;
            printf("\nData at index %d in Hash Table:\n", i);
            printf("VoterID     Name          Age   \n");
            printf("--------------------------------\n");
            while (myNode != NULL) {
                    printf("%-12d", myNode->key);
                    printf("%-15s", myNode->name);
                    printf("%d\n", myNode->age);
                    myNode = myNode->next;
            }
    }
    return;
}
int main() {
    int n, ch, key, age;
    char name[100];
    printf("Enter the number of elements:");
    scanf("%d", &n);
    eleCount = n;
    /* create hash table with "n" no of buckets */
    hashTable = (struct hash *)calloc(n, sizeof (struct hash));
    while (1) {
            printf("\n1. Insertion\t2. Deletion\n");
            printf("3. Searching\t4. Display\n5. Exit\n");
            printf("Enter your choice:");
            scanf("%d", &ch);
            switch (ch) {
                    case 1:
                            printf("Enter the key value:");
                            scanf("%d", &key);
                            getchar();
                            printf("Name:");
                            fgets(name, 100, stdin);
                            name[strlen(name) - 1] = '\0';
                            printf("Age:");
                            scanf("%d", &age);
                            /*inserting new node to hash table */
                            insertToHash(key, name, age);
                            break;

                    case 2:
                            printf("Enter the key to perform deletion:");
                            scanf("%d", &key);
                            /* delete node with "key" from hash table */
                            deleteFromHash(key);
                            break;
                    case 3:
                            printf("Enter the key to search:");
                            scanf("%d", &key);
                            searchInHash(key);
                            break;
                    case 4:
                            display();
                            break;
                    case 5:
                            exit(0);
                    default:
                            printf("U have entered wrong option!!\n");
                            break;
            }
    }
    return 0;
```

```
}
```



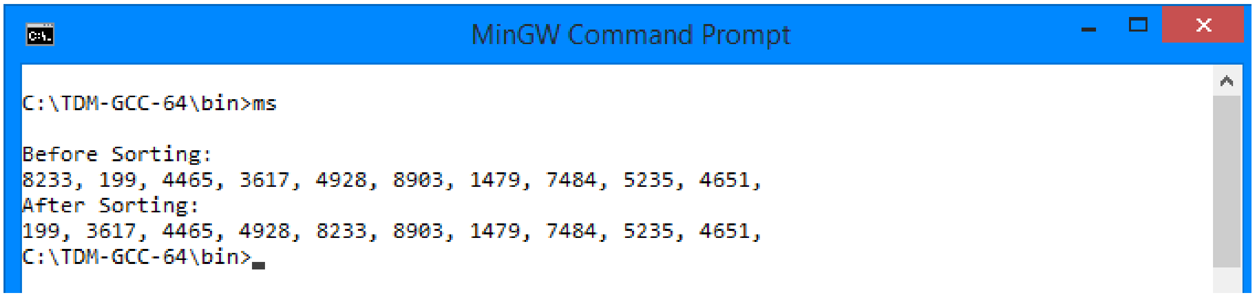
```
Running /home/ubuntu/workspace/hashtable.c
Enter the number of elements:3

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:23
Name:Tom
Age:32

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:12
Name:Dick
Age:30

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:1
Enter the key value:33
Name:Harry
Age:35
```



```
1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:4

Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
33          Harry          35
12          Dick           30

Data at index 2 in Hash Table:
VoterID     Name          Age
--------------------------------
23          Tom            32

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:2
Enter the key to perform deletion:12
Data deleted successfully from Hash Table

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:4

Data at index 0 in Hash Table:
VoterID     Name          Age
--------------------------------
33          Harry          35

Data at index 2 in Hash Table:
VoterID     Name          Age
--------------------------------
23          Tom            32

1. Insertion    2. Deletion
3. Searching    4. Display
5. Exit
Enter your choice:3
Enter the key to search:33
VoterID  : 33
Name     : Harry
Age      : 35
```

10. Merge sort

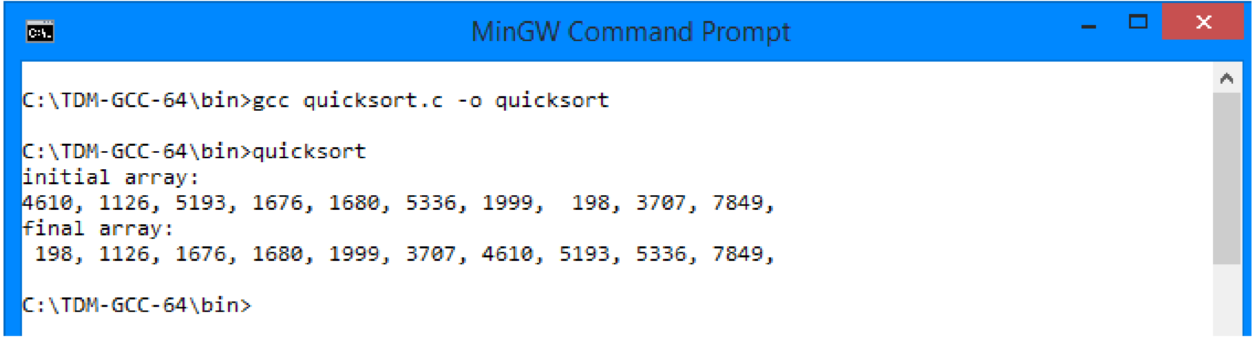
```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10
#define SEED 9999
void Merge(int[], int, int, int);
void DisplayArray(int[], int);

void MergeSort(int arr[], int startIndex, int endIndex)
{
        int mid;
        if(startIndex >= endIndex)
                return;
        mid = (endIndex + startIndex) / 2;
        MergeSort(arr, startIndex, mid);
        MergeSort(arr, mid+1, endIndex);
        Merge(arr, startIndex, mid, endIndex);
}

void Merge(int arr[], int startIndex, int mid, int endIndex)
{
        int i, j, k;
        int *workingArr = malloc(sizeof(int) * (endIndex - startIndex) + 1);
        for(i = startIndex, j = mid + 1, k = 0; i <= mid && j<= endIndex; k++)
        {
                if(arr[i] < arr[j])
                {
                        workingArr[k] = arr[i];
                        i++;
                }
                else
                {
                        workingArr[k] = arr[j];
                        j++;
                }
        }
        for( ; i <= mid; i++)
        {
                workingArr[k++] = arr[i];
        }
        for( ; j <= endIndex; j++)
        {
                workingArr[k++] = arr[j];
        }
        for(i = startIndex, k=0; i <= endIndex; i++, k++)
        {
                arr[i] = workingArr[k];
        }
}

void DisplayArray(int arr[], int arrLength)
{
        int i;
        for(i = 0; i < arrLength; i++)
        {
                printf("%d, ", arr[i]);
        }
}

int main()
{
        time_t t;
    srand((unsigned) time(&t));
        int arr[SIZE],i;
        for(i=0;i<SIZE;i++)
          arr[i]=rand()%SEED;

        printf("\nBefore Sorting:\n");
        DisplayArray(arr, SIZE);
        MergeSort(arr, 0, 4);
        printf("\nAfter Sorting:\n");
        DisplayArray(arr, SIZE);
}
```



```
C:\TDM-GCC-64\bin>ms

Before Sorting:
8233, 199, 4465, 3617, 4928, 8903, 1479, 7484, 5235, 4651,
After Sorting:
199, 3617, 4465, 4928, 8233, 8903, 1479, 7484, 5235, 4651,
C:\TDM-GCC-64\bin>
```

11. Quick sort

```
#include<stdio.h>
#include<stdlib.h>

#define SIZE 20
#define SEED 9999

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void print(int array[])
{
    int i;
    for(i=0;i<SIZE;i++)
        printf("%4d, ",array[i]);
    printf("\n");
}

int randpartition(int p, int r)
{
    int i;
    time_t t;
    srand((unsigned) time(&t));

    i=p+rand()%(r-p+1);

    return i;
}

int partition(int array[],int p, int r)
{
    int x= randpartition(p,r), i = p-1, j, pivot=array[x];
    swap(&array[x],&array[r]);
    for(j=p;j<r;j++)
    {
        if(array[j]<pivot)
        {
            i++;
            swap(&array[i],&array[j]);
        }
    }
    swap(&array[i+1],&array[r]);
    return i+1;
}

void quicksort(int array[],int p, int r)
{

    int q;
    if(p<r)
    {
        q= partition(array,p,r);
        quicksort(array,p, q-1);
        quicksort(array,q+1,r);
    }

}

int main()
{
    int array[SIZE] , i;

    time_t t;
    srand((unsigned) time(&t));
    for(i=0;i<SIZE;i++)
        array[i]=rand()%SEED;

    printf("initial array:\n");
    print(array);

    quicksort(array,0, SIZE-1);

    printf("final array:\n");
    print(array);

    return 0;
}
```



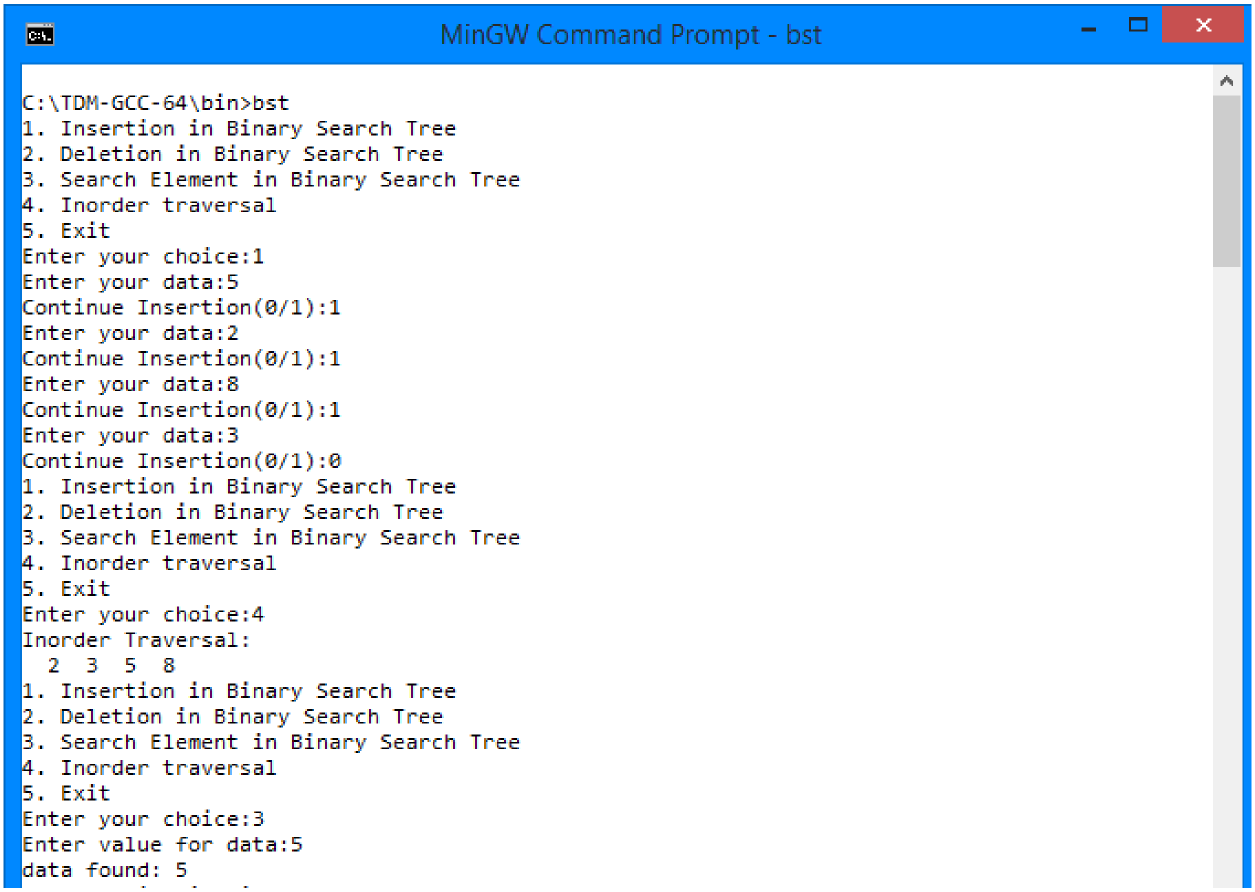
```
C:\TDM-GCC-64\bin>gcc quicksort.c -o quicksort

C:\TDM-GCC-64\bin>quicksort
initial array:
4610, 1126, 5193, 1676, 1680, 5336, 1999,  198, 3707, 7849,
final array:
 198, 1126, 1676, 1680, 1999, 3707, 4610, 5193, 5336, 7849,

C:\TDM-GCC-64\bin>
```

12. BST

```
#include <stdio.h>
#include <stdlib.h>

typedef struct treeNode treeNode;

struct treeNode
{
    int data;
    treeNode *left, *right;
};

treeNode *root = NULL;

treeNode* createNode(int data)
{
    treeNode *newNode;
    newNode = (treeNode *) malloc(sizeof (treeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return(newNode);
}


void insertion(treeNode **node, int data)
{
    if (*node == NULL)
        *node = createNode(data);
    else if (data < (*node)->data)
        insertion(&(*node)->left, data);
    else if (data > (*node)->data)
        insertion(&(*node)->right, data);
}


void deletion(treeNode **node, treeNode **parent, int data)
{
    treeNode *tmpNode, *tmpParent;

    if (*node == NULL)
            return;
    if ((*node)->data == data) {
            if (!(*node)->left && !(*node)->right)
            {
                if (parent)
                {
                    if ((*parent)->left == *node)
                        (*parent)->left = NULL;
                    else
                        (*parent)->right = NULL;

                    free(*node);
                }
```

```
                                else
                                        free(*node);
                        }
                        else if (!(*node)->right && (*node)->left)
                        {
                                tmpNode = *node;
                                (*parent)->right = (*node)->left;
                                free(tmpNode);
                                *node = (*parent)->right;
                        }
                        else if ((*node)->right && !(*node)->left)
                        {
                                tmpNode = *node;
                                (*parent)->left = (*node)->right;
                                free(tmpNode);
                                (*node) = (*parent)->left;
                        }
                        else if (!(*node)->right->left)
                        {
                                tmpNode = *node;
                                (*node)->right->left = (*node)->left;
                                (*parent)->left = (*node)->right;
                                free(tmpNode);
                                *node = (*parent)->left;
                        }
                        else
                        {
                                tmpNode = (*node)->right;
                                while (tmpNode->left)
                                {
                                                tmpParent = tmpNode;
                                                tmpNode = tmpNode->left;
                                }
                                tmpParent->left = tmpNode->right;
                                tmpNode->left = (*node)->left;
                                tmpNode->right =(*node)->right;
                                free(*node);
                                *node = tmpNode;
                        }
        }
        else if (data < (*node)->data)
                deletion(&(*node)->left, node, data);
        else if (data > (*node)->data)
                deletion(&(*node)->right, node, data);

}

void findElement(struct treeNode *node, int data)
{
        if (!node)
                        return;
        else if (data < node->data)
                findElement(node->left, data);
        else if (data > node->data)
                findElement(node->right, data);
```

```
    else
        printf("data found: %d\n", node->data);
    return;
}

void traverse(struct treeNode *node)
{
    if (node != NULL)
    {
        traverse(node->left);
        printf("%3d", node->data);
        traverse(node->right);
    }
    return;
}

int main() {

    int data, ch;

    while (1)
    {
            printf("1. Insertion in Binary Search Tree\n");
            printf("2. Deletion in Binary Search Tree\n");
            printf("3. Search Element in Binary Search Tree\n");
            printf("4. Inorder traversal\n5. Exit\n");
            printf("Enter your choice:");
            scanf("%d", &ch);

            switch (ch)
            {
                    case 1:
                            while (1)
                            {
                                printf("Enter your data:");
                                scanf("%d", &data);
                                insertion(&root, data);
                                printf("Continue
Insertion(0/1):");
                                scanf("%d", &ch);
                                if (!ch)
                                    break;
                            }
                            break;
                    case 2:
                            printf("Enter your data:");
                            scanf("%d", &data);
                            deletion(&root, NULL, data);
                            break;
                    case 3:
                            printf("Enter value for data:");
                            scanf("%d", &data);
                            findElement(root, data);
                            break;
                    case 4:
                            printf("Inorder Traversal:\n");
                            traverse(root);
                            printf("\n");
                            break;
                    case 5:
                            exit(0);
                    default:
                            printf("u've entered wrong option\n");
                            break;
            }
    }
    return 0;

}
```

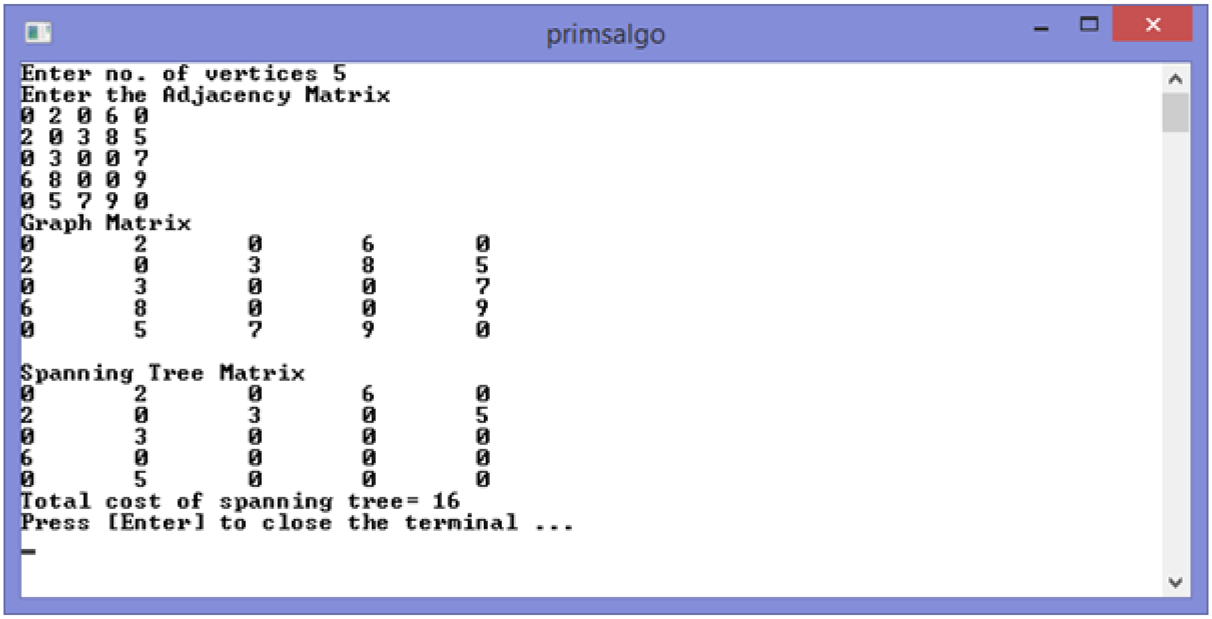


```
C:\TDM-GCC-64\bin>bst
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:1
Enter your data:5
Continue Insertion(0/1):1
Enter your data:2
Continue Insertion(0/1):1
Enter your data:8
Continue Insertion(0/1):1
Enter your data:3
Continue Insertion(0/1):0
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
  2  3  5  8
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:3
Enter value for data:5
data found: 5
```

13. Prims

```
#include<stdio.h>
#include<stdlib.h>

#define infinity 9999
#define MAX 20

void prims(int G[][MAX],int n);

int main()
{
	int i,j,total_cost;
	int G[MAX][MAX],n;
	printf("Enter no. of vertices ");
	scanf("%d",&n);
	printf("Enter the Adjacency Matrix ");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&G[i][j]);
	printf("Graph Matrix ");
	for(i=0;i<n;i++)
	{
		printf("\n");
		for(j=0;j<n;j++)
			printf("%d\t",G[i][j]);
	}
	prims(G,n);
	return 0;
}

void prims(int G[][MAX], int n)
{
	int cost[MAX][MAX];
	int spanning[MAX][MAX];
	int u,v,min_distance,distance[MAX],from[MAX];
	int visited[MAX],no_of_edges,i,min_cost,j;
	for(i=0;i<n;i++)
	for(j=0;j<n;j++)
	{
		if(G[i][j]==0)
			cost[i][j]=infinity;
		else
			cost[i][j]=G[i][j];
		spanning[i][j]=0;
	}
	distance[0]=0;
	visited[0]=1;
	for(i=1;i<n;i++)
	{
		distance[i]=cost[0][i];
		from[i]=0;
		visited[i]=0;
	}
	min_cost=0;
```

```
        no_of_edges=n-1;
        while(no_of_edges>0)
        {
                min_distance=infinity;
                for(i=1;i<n;i++)
                        if(visited[i]==0&&distance[i]<min_distance)
                        {
                                v=i;
                                min_distance=distance[i];
                        }
                u=from[v];
                spanning[u][v]=distance[v];
                spanning[v][u]=distance[v];
                no_of_edges--;
                visited[v]=1;
                for(i=1;i<n;i++)
                        if(visited[i]==0&&cost[i][v]<distance[i])
                        {
                                distance[i]=cost[i][v];
                                from[i]=v;
                        }
                min_cost=min_cost+cost[u][v];
        }
        printf("\n\nSpanning Tree Matrix ");
        for(i=0;i<n;i++)
        {
                printf("\n");
                for(j=0;j<n;j++)
                        printf("%d\t",spanning[i][j]);
        }
        printf("\nTotal cost of spanning tree= %d \n",min_cost);
}
```

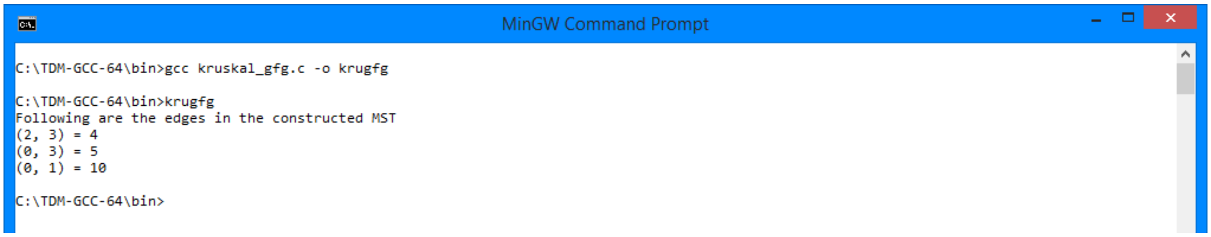


```
Enter no. of vertices 5
Enter the Adjacency Matrix
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Graph Matrix
0       2       0       6       0
2       0       3       8       5
0       3       0       0       7
6       8       0       0       9
0       5       7       9       0

Spanning Tree Matrix
0       2       0       6       0
2       0       3       0       5
0       3       0       0       0
6       0       0       0       0
0       5       0       0       0
Total cost of spanning tree= 16
Press [Enter] to close the terminal ...
_
```

14. Kruskal

```
#include <stdio.h>
#include <stdlib.h>

struct Edge
{
    int src, dest, weight;
};

struct Graph
{
    int V, E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

struct subset
{
    int parent;
    int rank;
};

int find(struct subset subsets[], int i)
{
if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
return subsets[i].parent;
}

void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

```
int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

void KruskalMST(struct Graph* graph)
{
    int v;
    int V = graph->V;
    struct Edge result[V];
    int e = 0;
    int i = 0;
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    struct subset *subsets =
        (struct subset*) malloc( V * sizeof(struct subset) );

    for (v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    while (e < V - 1)
    {
        struct Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }

    printf("Following are the edges in the constructed MST\n");
    for (i = 0; i < e; ++i)
        printf("(%d, %d) = %d\n", result[i].src, result[i].dest,
                                                   result[i].weight);
    return;
}

int main()
{
/*
         10
    0--------1
    |  \     |
   6|   5\   |15
    |      \ |
    2--------3
        4       */
    int V = 4;
    int E = 5;
    struct Graph* graph = createGraph(V, E);

    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}
```

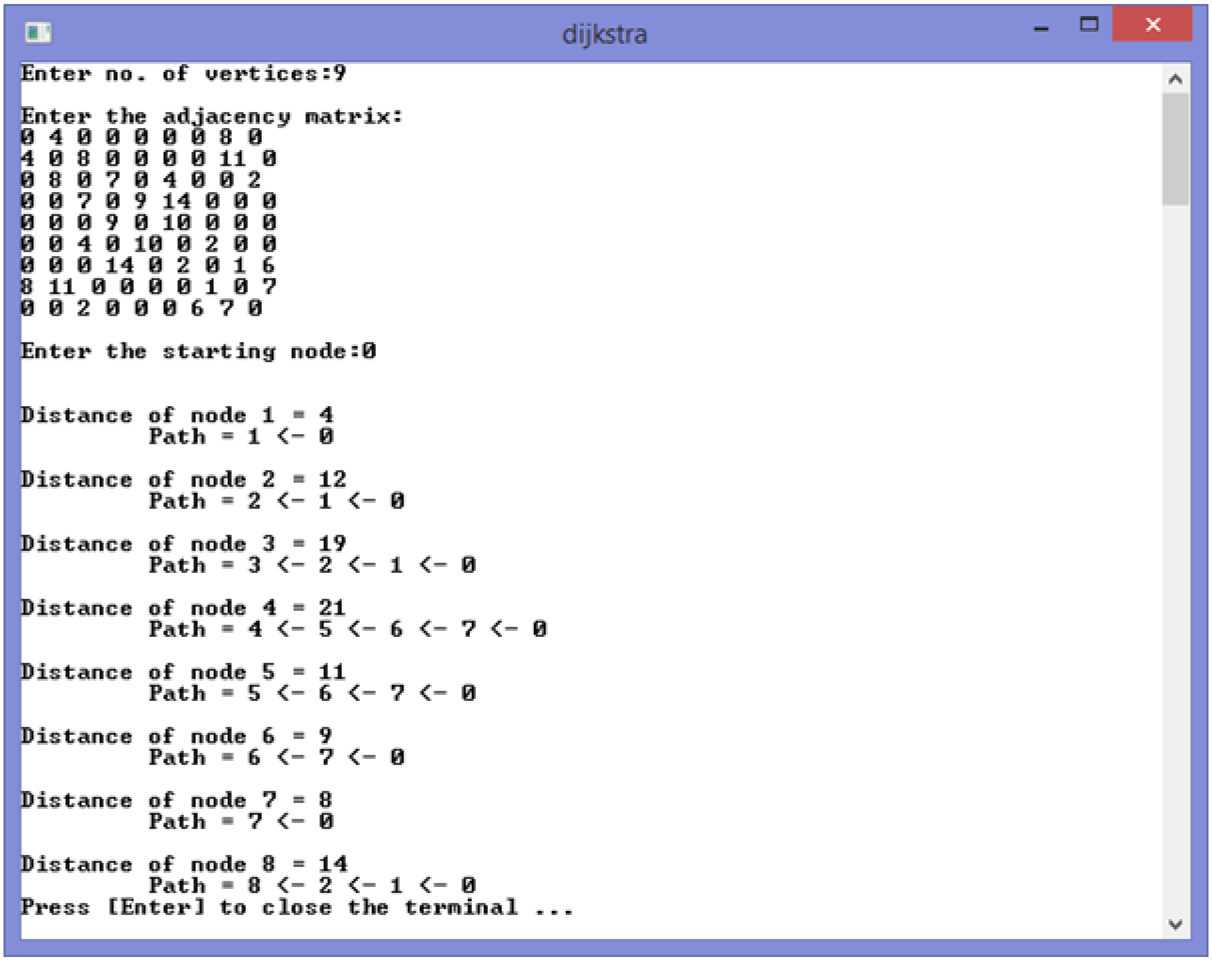


```
C:\TDM-GCC-64\bin>gcc kruskal_gfg.c -o krugfg

C:\TDM-GCC-64\bin>krugfg
Following are the edges in the constructed MST
(2, 3) = 4
(0, 3) = 5
(0, 1) = 10

C:\TDM-GCC-64\bin>
```

15. Dijikstras

```
#include<stdio.h>
#include<conio.h>

#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
	int G[MAX][MAX],i,j,n,u;
	printf("Enter no. of vertices:");
	scanf("%d",&n);
	printf("\nEnter the adjacency matrix:\n");
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&G[i][j]);
	printf("\nEnter the starting node:");
	scanf("%d",&u);
	dijkstra(G,n,u);
	return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{
	int cost[MAX][MAX],distance[MAX],pred[MAX];
	int visited[MAX],count,mindistance,nextnode,i,j;
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			if(G[i][j]==0)
				cost[i][j]=INFINITY;
			else
				cost[i][j]=G[i][j];
	for(i=0;i<n;i++)
	{
		distance[i]=cost[startnode][i];
		pred[i]=startnode;
		visited[i]=0;
	}
	distance[startnode]=0;
	visited[startnode]=1;
	count=1;
	while(count<n-1)
	{
		mindistance=INFINITY;
		for(i=0;i<n;i++)
			if(distance[i]<mindistance&&!visited[i])
			{
				mindistance=distance[i];
				nextnode=i;
			}
		visited[nextnode]=1;
		for(i=0;i<n;i++)
```

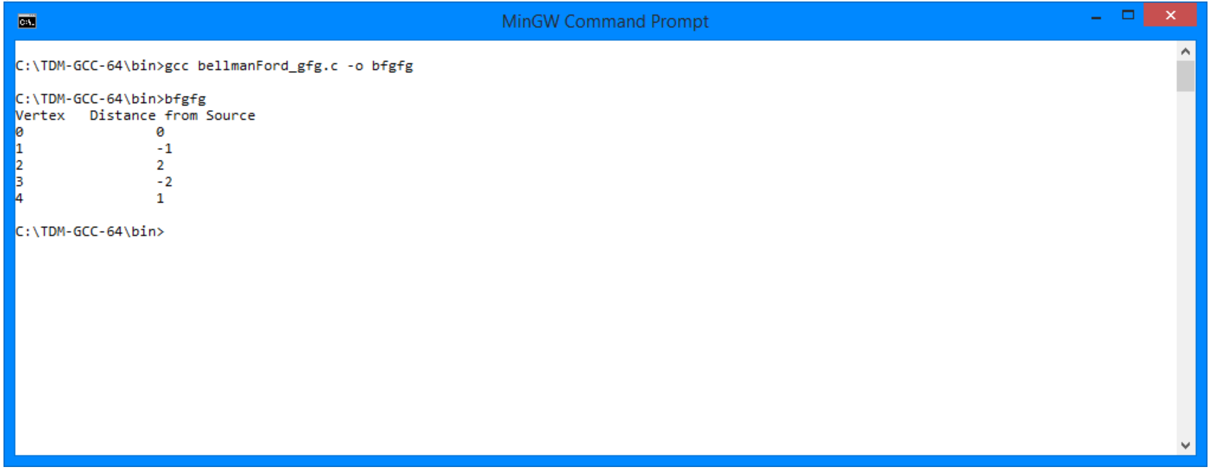
```
                        if(!visited[i])
                                if(mindistance+cost[nextnode][i]<distance[i])
                                {
                                                distance[i]=mindistance+cost[nextnode][i];
                                        pred[i]=nextnode;
                                }
                count++;
        }
        for(i=0;i<n;i++)
                if(i!=startnode)
                {
                        printf("\n\nDistance of node %d = %d",i,distance[i]);
                        printf("\n\t Path = %d",i);
                        j=i;
                        do
                        {
                                j=pred[j];
                                printf(" <- %d",j);
                        }while(j!=startnode);
                }
        printf("\n");
}
```



```
Enter no. of vertices:9

Enter the adjacency matrix:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 0 10 0 2 0 0
0 0 0 14 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

Enter the starting node:0


Distance of node 1 = 4
         Path = 1 <- 0

Distance of node 2 = 12
         Path = 2 <- 1 <- 0

Distance of node 3 = 19
         Path = 3 <- 2 <- 1 <- 0

Distance of node 4 = 21
         Path = 4 <- 5 <- 6 <- 7 <- 0

Distance of node 5 = 11
         Path = 5 <- 6 <- 7 <- 0

Distance of node 6 = 9
         Path = 6 <- 7 <- 0

Distance of node 7 = 8
         Path = 7 <- 0

Distance of node 8 = 14
         Path = 8 <- 2 <- 1 <- 0
Press [Enter] to close the terminal ...
```

16. Bellman-Ford

```
#include <stdio.h>
#include <stdlib.h>
struct Edge
{
    int src, dest, weight;
};

struct Graph
{
    int V, E;

    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

void printArr(int dist[], int n)
{
    int i;
    printf("Vertex   Distance from Source\n");
    for (i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

void BellmanFord(struct Graph* graph, int src)
{
    int i, j;
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    for (i = 0; i < V; i++)
        dist[i]   = INT_MAX;
    dist[src] = 0;

    for (i = 1; i <= V-1; i++)
    {
        for (j = 0; j < E; j++)
        {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    for (i = 0; i < E; i++)
    {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            printf("Graph contains negative weight cycle");
    }

    printArr(dist, V);

    return;
}

int main()
{
    int i,j;
    int V = 5;
    int E = 8;
    struct Graph* graph = createGraph(V, E);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;

    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;

    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;

    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);
    return 0;
}
```



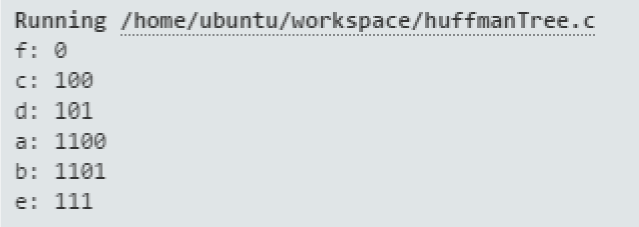
```
C:\TDM-GCC-64\bin>gcc bellmanFord_gfg.c -o bfgfg

C:\TDM-GCC-64\bin>bfgfg
Vertex   Distance from Source
0                0
1                -1
2                2
3                -2
4                1

C:\TDM-GCC-64\bin>
```

17. Huffman tree

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_TREE_HT 100

struct MinHeapNode
{
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

struct MinHeap
{
    unsigned size;
    unsigned capacity;
    struct MinHeapNode **array;
};

struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp =
          (struct MinHeapNode*) malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap = (struct MinHeap*) malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
MinHeapNode*));
    return minHeap;
}

void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size && minHeap->array[left]->freq < minHeap-
>array[smallest]->freq)
```

```
      smallest = left;

    if (right < minHeap->size && minHeap->array[right]->freq < minHeap-
>array[smallest]->freq)
      smallest = right;

    if (smallest != idx)
    {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1)/2]->freq)
    {
        minHeap->array[i] = minHeap->array[(i - 1)/2];
        i = (i - 1)/2;
    }
    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}
```

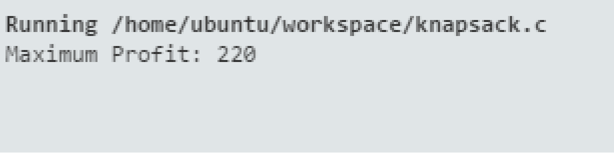
```
int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right) ;
}

struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
      int i;
    for (i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

    while (!isSizeOne(minHeap))
    {
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }

    return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* root, int arr[], int top)
{
    if (root->left)
    {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    if (root->right)
    {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    if (isLeaf(root))
    {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

void HuffmanCodes(char data[], int freq[], int size)
{
   struct MinHeapNode* root = buildHuffmanTree(data, freq, size);

   int arr[MAX_TREE_HT], top = 0;
   printCodes(root, arr, top);
}

int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr)/sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}
```



```
Running /home/ubuntu/workspace/huffmanTree.c
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

18. Knapsack

```
#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];

   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }

   return K[n][W];
}

int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int  W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("Maximum Profit: %d", knapSack(W, wt, val, n));
    return 0;
}
```

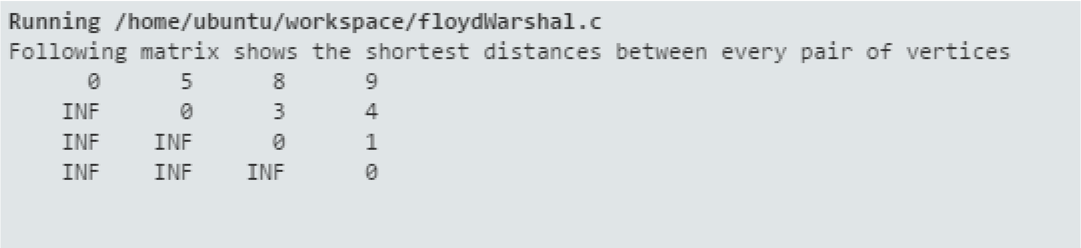


```
Running /home/ubuntu/workspace/knapsack.c
Maximum Profit: 220
```

19. Shortest distance from source to destination by using dynamic programing.

```
#include<stdio.h>

#define V 4

#define INF 99999

void printSolution(int dist[][V]);

void floydWarshell (int graph[][V])
{
    int dist[V][V], i, j, k;

    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    for (k = 0; k < V; k++)
    {
        for (i = 0; i < V; i++)
        {
            for (j = 0; j < V; j++)
            {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    printSolution(dist);
}

void printSolution(int dist[][V])
{
    int i, j;
    printf ("Following matrix shows the shortest distances"
            " between every pair of vertices \n");
    for (i = 0; i < V; i++)
    {
        for (j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    /* Let us create the following weighted graph
            10
```

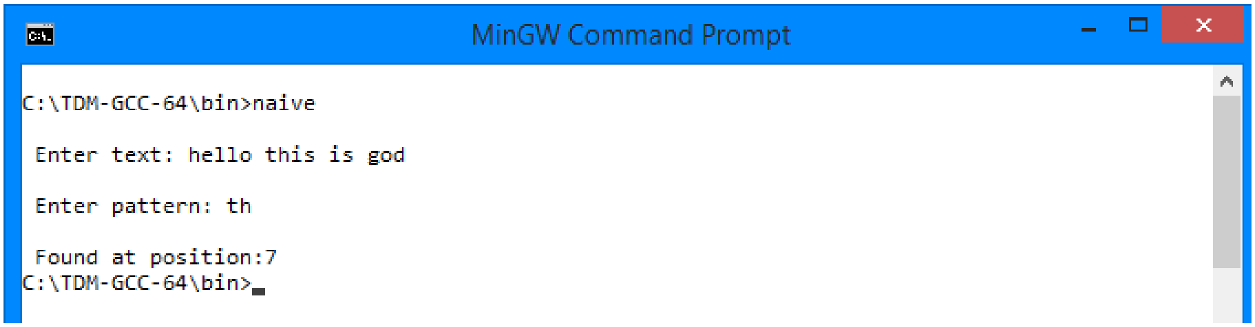
```
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           */
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                      };

    floydWarshell(graph);
    return 0;
}
```



```
Running /home/ubuntu/workspace/floydWarshal.c
Following matrix shows the shortest distances between every pair of vertices
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
```

20. Naïve String matching Algorithm

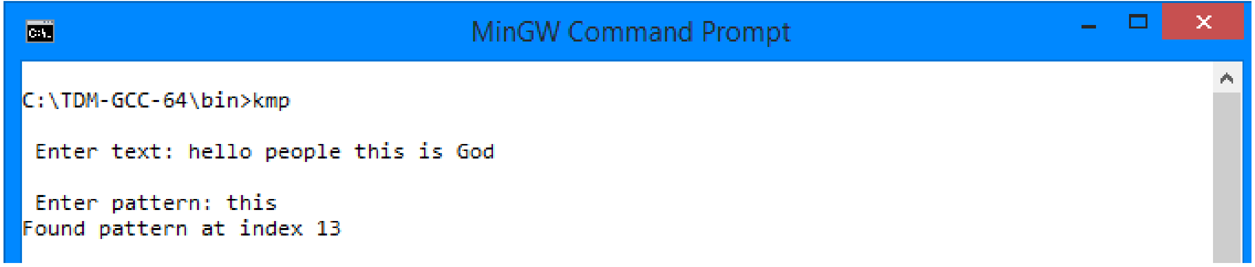
```
#include <stdio.h>
#include <string.h>

int naive_match(char text[], char pattern[])
{
  int i, j, k, text_length, pattern_length, position = -1;

  text_length    = strlen(text);
  pattern_length = strlen(pattern);

  if (pattern_length > text_length) {
    return -1;
  }

  for (i = 0; i <= text_length - pattern_length; i++) {
    position = k = i;

    for (j = 0; j < pattern_length; j++) {
      if (pattern[j] == text[k]) {
        k++;
      }
      else {
        break;
      }
    }
    if (j == pattern_length) {
      return position;
    }
  }

  return -1;
}

int main() {
  char text[100], pattern[100];
  int position;

  printf("\n Enter text: ");
  gets(text);

  printf("\n Enter pattern: ");
  gets(pattern);

  position = naive_match(text, pattern);

  if(position != -1) {
    printf("\n Found at position:%d", position + 1);
  }
  else {
    printf("\n Not found.");
  }
  return 0;
}
```



```
C:\TDM-GCC-64\bin>naive

 Enter text: hello this is god

 Enter pattern: th

 Found at position:7
C:\TDM-GCC-64\bin>
```

21. String matching by Knuth Morris Pratt

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void computePrefix(char pattern[], int M, int *lps);

int KMPSearch(char pattern[], char *text)
{
    int M = strlen(pattern);
    int N = strlen(text);


    int *lps = (int *)malloc(sizeof(int)*M);
    int j  = 0;
    computePrefix(pattern, M, lps);

    int i = 0;
    while (i < N)
    {
      if (pattern[j] == text[i])
      {
        j++;
        i++;
      }

      if (j == M)
      {
        printf("Found pattern at index %d \n", i-j);
        return 0;
      }

      else if (i < N && pattern[j] != text[i])
      {

        if (j != 0)
         j = lps[j-1];
        else
         i = i+1;
      }

    }
    free(lps);
return -1;
}

void computePrefix(char pattern[], int M, int *lps)
{
    int len = 0;
    int i;

    lps[0] = 0;
    i = 1;
    while (i < M)
    {
       if (pattern[i] == pattern[len])
       {
         len++;
         lps[i] = len;
         i++;
       }
       else
       {
         if (len != 0)
         {

           len = lps[len-1];


         }
         else
         {
           lps[i] = 0;
           i++;
         }
       }
    }
}

int main()
{
   char text[100], pattern[100];
  int position;

  printf("\n Enter text: ");
  gets(text);

  printf("\n Enter pattern: ");
  gets(pattern);


   position=KMPSearch(pattern, text);
   if(position==-1)
       printf("\n Not found");
   return 0;
}
```

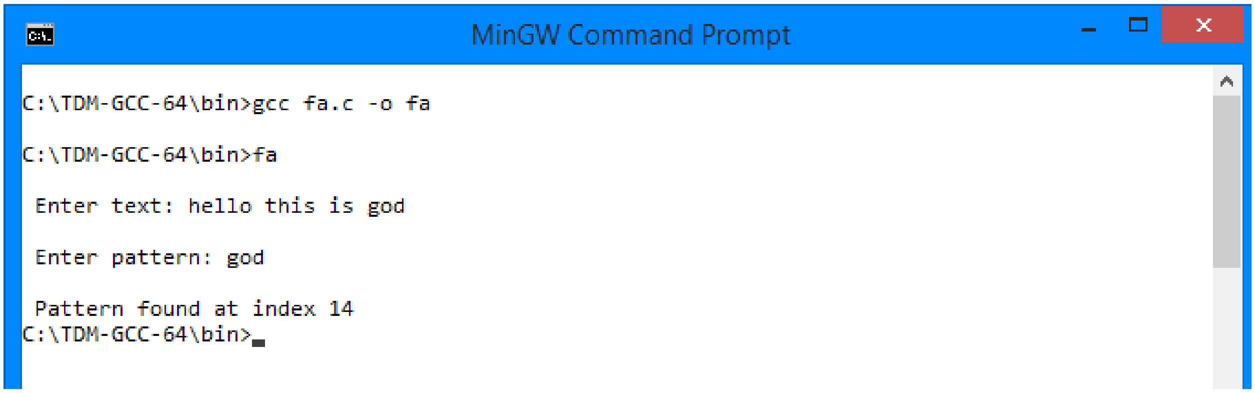


```
C:\TDM-GCC-64\bin>kmp

 Enter text: hello people this is God

 Enter pattern: this
Found pattern at index 13
```

22. String matching by automata

```
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

int getNextState(char *pattern, int M, int state, int x)
{
    if (state < M && x == pattern[state])
        return state+1;

    int ns, i;
    for (ns = state; ns > 0; ns--)
    {
        if(pattern[ns-1] == x)
        {
            for(i = 0; i < ns-1; i++)
            {
                if (pattern[i] != pattern[state-ns+1+i])
                    break;
            }
            if (i == ns-1)
                return ns;
        }
    }

    return 0;
}

void TransitionFunction(char *pat, int M, int Transition[][NO_OF_CHARS])
{
    int state, x;
    for (state = 0; state <= M; ++state)
        for (x = 0; x < NO_OF_CHARS; ++x)
           Transition[state][x] = getNextState(pat, M,  state, x);
}

void search(char *pattern, char *text)
{
    int M = strlen(pattern);
    int N = strlen(text);

    int Transition[M+1][NO_OF_CHARS];

    TransitionFunction(pattern, M, Transition);

    int i, state=0;
    for (i = 0; i < N; i++)
    {
       state = Transition[state][text[i]];
       if (state == M)
       {
         printf ("\n Pattern found at index %d", i-M+1);
       }
    }
}
```

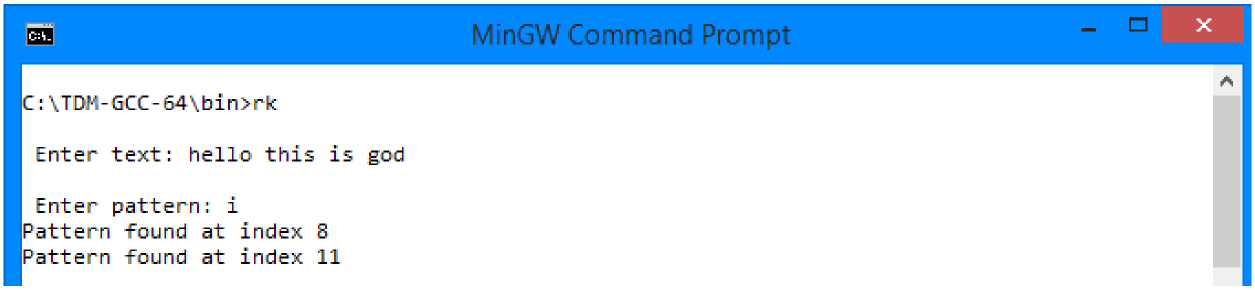
```
int main()
{
          char text[100], pattern[100];

         printf("\n Enter text: ");
         gets(text);

         printf("\n Enter pattern: ");
         gets(pattern);

       search(pattern, text);
   return 0;
}
```



```
C:\TDM-GCC-64\bin>gcc fa.c -o fa

C:\TDM-GCC-64\bin>fa

 Enter text: hello this is god

 Enter pattern: god

 Pattern found at index 14
C:\TDM-GCC-64\bin>
```

23. String matching by Rabin-Karp Algorithm

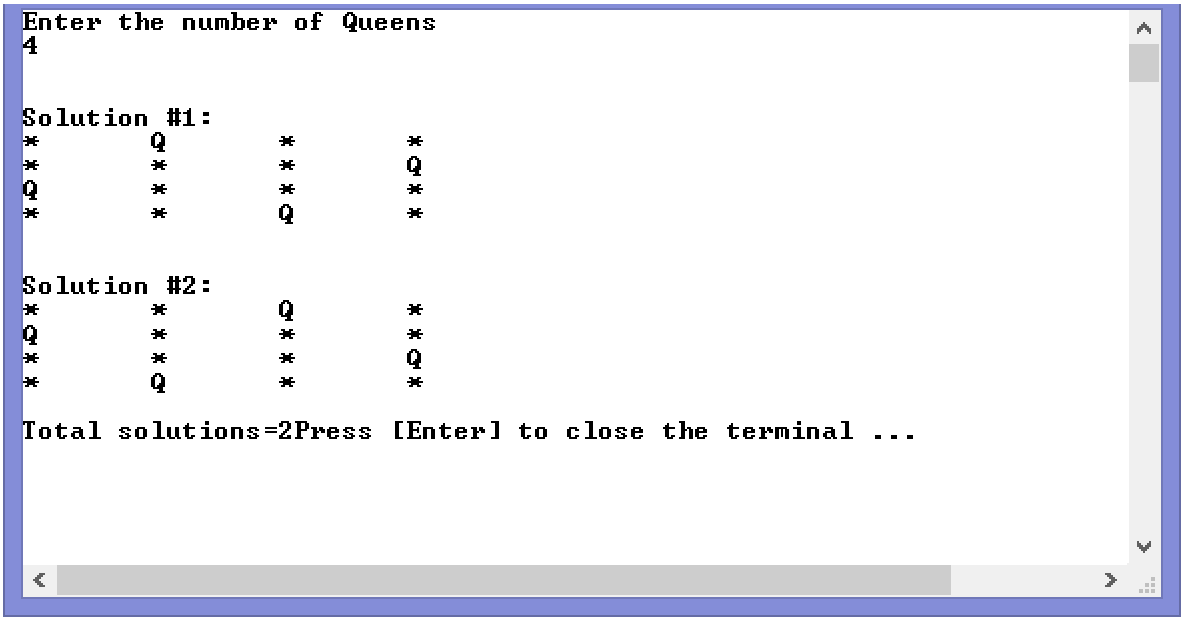
```
#include<stdio.h>
#include<string.h>
#define d 256

void search(char *pat, char *txt, int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;

    for (i = 0; i < M-1; i++)
        h = (h*d)%q;

    for (i = 0; i < M; i++)
    {
        p = (d*p + pat[i])%q;
        t = (d*t + txt[i])%q;
    }
    for (i = 0; i <= N - M; i++)
    {
        if ( p == t )
        {
            for (j = 0; j < M; j++)
            {
                if (txt[i+j] != pat[j])
                    break;
            }
            if (j == M)
            {
                printf("Pattern found at index %d \n", i);
            }
        }
        if ( i < N-M )
        {
            t = (d*(t - txt[i]*h) + txt[i+M])%q;

            if(t < 0)
              t = (t + q);
        }
    }
}

int main()
{
    char text[100], pattern[100];

        printf("\n Enter text: ");
        gets(text);

        printf("\n Enter pattern: ");
        gets(pattern);
    int q = 13;
    search(pattern, text, q);
    return 0;
}
```



```
C:\TDM-GCC-64\bin>rk

 Enter text: hello this is god

 Enter pattern: i
Pattern found at index 8
Pattern found at index 11
```

24. N Queen Problem using Backtracking

```
#include<stdio.h>
#include<math.h>

int a[30],count=0;
int place(int pos)
{
	int i;
	for(i=1;i<pos;i++)
	{
		if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
			return 0;
	}
	return 1;
}

void print_sol(int n)
{
	int i,j;
	count++;
	printf("\n\nSolution #%d:\n",count);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i]==j)
				printf("Q\t");
			else
				printf("*\t");
		}
		printf("\n");
	}
}

void queen(int n)
{
	int k=1;
	a[k]=0;
	while(k!=0)
	{
		a[k]=a[k]+1;
		while((a[k]<=n)&&!place(k))
			a[k]++;
		if(a[k]<=n)
		{
			if(k==n)
				print_sol(n);
			else
			{
				k++;
				a[k]=0;
			}
		}
		else
			k--;
	}
}

void main()
{
	int i,n;
	printf("Enter the number of Queens\n");
	scanf("%d",&n);
	queen(n);
	printf("\nTotal solutions=%d",count);
}
```

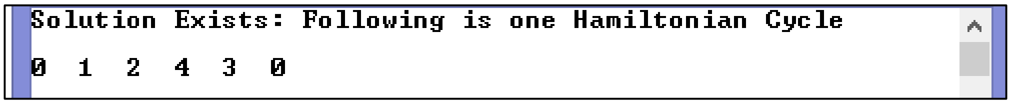


```
Enter the number of Queens
4


Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *


Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions=2Press [Enter] to close the terminal ...
```

25. Hamiltonian path problem

```
#include <stdio.h>
#include <stdlib.h>
#define V 5
#define false 0
#define true 1

void printSolution(int path[]);

int isSafe(int v, int graph[V][V], int path[], int pos)
{
	int i;
if (graph [path[pos-1]][v] == 0)
		return false;
	for (i = 0; i < pos; i++)
		if (path[i] == v)
			return false;
	return true;
}

int hamCycleUtil(int graph[V][V], int path[], int pos)
{
	int v;
	if (pos == V)
	{
		if (graph[ path[pos-1] ][ path[0] ] == 1)
			return true;
		else
			return false;
	}
	for (v = 1; v < V; v++)
	{
		if (isSafe(v, graph, path, pos))
		{
			path[pos] = v;
			if (hamCycleUtil (graph, path, pos+1) == true)
				return true;
			path[pos] = -1;
		}
	}
	return false;
}

int hamCycle(int graph[V][V])
{
	int *path = (int *)malloc(V*sizeof(int));
	int i;
	for (i = 0; i < V; i++)
		path[i] = -1;
	path[0] = 0;
	if (hamCycleUtil(graph, path, 1) == false)
	{
		printf("\nSolution does not exist");
		return false;
	}
```

```
        printSolution(path);
        return true;
}

void printSolution(int path[])
{
        int i;
        printf("Solution Exists:");
        printf(" Following is one Hamiltonian Cycle \n");
        for (i = 0; i < V; i++)
                printf("%d",path[i]);
        printf("%d",path[0]);
}

int main()
{
        int graph[V][V];
        int i,j;
        printf("\n Enter the adjacency matrix:");
        for(i=0;i<V;i++)
        {
                for(j=0;j<V;j++)
                {
                        scanf("%d",&graph[i][j]);
                }
        }
        printf("\n The adjacency matrix is: ");
        for(i=0;i<V;i++)
        {
                printf("\n");
                for(j=0;j<V;j++)
                {
                        printf("%d ",graph[i][j]);
                }
        }
        hamCycle(graph);

        return 0;
}
```



```
Solution Exists: Following is one Hamiltonian Cycle
0  1  2  4  3  0
```