|  |  |
| --- | --- |
| Logo | LUKHDHIRJI ENGINEERING COLLEGE |
|  | *Information Technology Department* |

**Analysis & Design of Algorithms**

Subject Code:- 3150703

***Laboratory Manual***

*For*

***Semester – V***

***List of Experiments***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Experiment Title** | **Page**  **No.** | **Date** | **Grade** | **Signature** |
| 1. | Implementation and Time analysis of sorting algorithms: Bubble sort, Selection sort, Insertion sort, Merge sort and Quick sort. |  |  |  |  |
| 2. | Implementation and Time analysis of linear and binary search algorithm |  |  |  |  |
| 3. | Implementation of max – heap sort algorithm. |  |  |  |  |
| 4. | Implementation and Time analysis of factorial program  using iterative and recursive method. |  |  |  |  |
| 5. | Implementation of a knapsack problem using dynamic  programming |  |  |  |  |
| 6. | Implementation of chain matrix multiplication using  dynamic programming |  |  |  |  |
| 7. | Implementation of making a change problem using  dynamic programming. |  |  |  |  |
| 8. | Implementation of a knapsack problem using greedy  algorithm. |  |  |  |  |
| 9. | Implementation of Graph and Searching (DFS & BFS). |  |  |  |  |
| 10. | Implement LCS problem. |  |  |  |  |

Total Grade Points: / 110

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time analysis of sorting algorithms: Bubble sort, Selection sort, Insertion sort, Merge sort and Quick sort. | |
| *Sem. V* | EXPERIMENT NO. : **01** | DATE : |

# AIM:

## Implementation and Time analysis of sorting algorithms: Bubble sort, Selection sort, Insertion sort, Merge sort and Quick sort.

# THEORY:

***Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> void main()

{

int a[5]; int i,j,tmp; clrscr();

printf("enter elements:"); for(i=0;i<5;i++)

{

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

}

for(i=0;i<5;i++)

{

for(j=i+1;j<5;j++)

{

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

{

tmp=a[i]; a[i]=a[j]; a[j]=tmp

}

}

for(i=0;i<5;i++)

{

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

}

getch();

}

# OUTPUT:

enter element:

5

4

3

2

1

12345

**Signature: Grade : / 10**

|  |  |
| --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of Selection Sort: |

# AIM:

## Implementation and Time Analysis of Selection Sort:

# THEORY:

***Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.***

***The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> void main ()

{

int a[5];

int i,j,tmp,tmp1; clrscr();

printf("enter element:"); for(i=0;i<5;i++)

{

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

}

for(i=0;i<5;i++)

{

tmp=i; for(j=i+1;j<5;j++)

{

if(a[j]<a[tmp])

{

tmp=j;

}

}

tmp1=a[tmp];

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of Insertion Sort: | |
| *Sem. V* | EXPERIMENT NO. : **03** | DATE : |

# AIM:

## Implementation and Time Analysis of Insertion Sort:

.

# THEORY:

***This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> void main ()

{

int a[5]; int i,j,tmp; clrscr();

printf("enter element:"); for(i=0;i<5;i++)

{

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

}

for(i=1;i<5;i++)

{

tmp=a[i]; j=i-1;

while(j>=0 && a[j]>tmp)

{

a[j+1]=a[j]; j=j-1;

}

a[j+1]=tmp;

}

a[tmp]=a[i]; a[i]=tmp1;

}

for(i=0;i<5;i++)

{

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

}

getch();

}

# OUTPUT:

enter element:

5

4

3

2

1

12345

**Signature: Grade : / 10**

|  |  |
| --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of Merge Sort: |

# AIM:

## Implementation and Time Analysis of Merge Sort:

# THEORY:

***Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.***

***Merge sort first divides the array into equal halves and then combines them in a sorted manner.***

# PROGRAM:

* #include <stdio.h> #define max 10

int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };

int b[10];

void merging(int low, int mid, int high)

{

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)

{

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++) a[i] = b[i];

}

void sort(int low, int high)

{

int mid; if(low < high)

{

mid = (low + high) / 2; sort(low, mid); sort(mid+1, high); merging(low, mid, high);

}

else

{return;

}

}

void main()

{ int i;

printf("Elements before sorting\n"); for(i = 0; i <= max; i++)

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

}

sort(0, max);

printf("\nElements after sorting\n"); for(i = 0; i <= max; i++)

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

# OUTPUT:

Elements before sorting:

10 14 19 26 27 31 33 35 42 44 0

Elements after sorting:

0 10 14 19 26 27 31 33 35 42 44

**Signature: Grade : / 10**

|  |  |
| --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of quick Sort: |

# AIM:

## Implementation and Time Analysis of Quick Sort:

.

# THEORY:

***Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.***

# PROGRAM:

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

void quicksort(int [10],int,int); void main()

{

int x[20],size,i;

printf("Enter size of the array: "); scanf("%d",&size);

printf("Enter %d elements: ",size); for(i=0;i<size;i++)

scanf("%d",&x[i]); quicksort(x,0,size-1); printf("Sorted elements: "); for(i=0;i<size;i++)

printf(" %d",x[i]); getch();

}

void quicksort(int x[10],int first,int last)

{

int pivot,j,temp,i; if(first<last)

{

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; quicksort(x,first,j-1); quicksort(x,j+1,last);

}

}

### Output:

Enter size of the array: 5 Enter 5 elements: 5 4 3 2 1

Sorted elements: 1 2 3 4 5

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of linear search: | |
| *Sem. V* | EXPERIMENT NO. : **02** | DATE : |

# AIM:

Implementation and Time Analysis of Linear Search Algorithm:

# THEORY:

# *Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection*

# PROGRAM:

* #include<stdio.h> #include<conio.h> void main()

{ int a[5],s,i; clrscr();

printf("enter the numbers of elments in array\n"); for(i=0;i<5;i++)

scanf("%d",&a[i]); printf("enter a number to search\n"); scanf("%d",&s);

for(i=0;i<5;i++)

if(a[i]==s)

{ printf("%d is present at location %d\n",s,i+1); break;

}

if(i==5)

printf("%d isn't [resent in the aray\n",s);

getch();

}

# OUTPUT:

enter the numbers of elements in array 5

4

3

2

1

enter number to search 3

3 is present at location 3

**Signature: Grade : / 10**

|  |  |
| --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of Binary Search Algorithm: |

# AIM:

## Implementation and Time Analysis of Binary Search Algorithm:

# THEORY:

# *Binary search is a fast search algorithm with run-time complexity of Ο(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form*

# PROGRAM:

* #include<stdio.h> #include<conio.h> void main()

{ int a[5],i,f,l,m,s; clrscr();

printf("enter the 5 elements in array:\n"); for(i=0;i<5;i++) { scanf("%d",&a[i]); } printf("enter elment to search:"); scanf("%d",&s);

f=0; l=5-1;

m=(f+l)/2; while(f<=l)

if(a[m]<s) { f=m+1; }

else if(a[m]==s) { printf("%d found at location %d\n",s,m+1);

break; }

else { l=m-1; } m=(f+l)/2;

if(f>l) { printf("not found!! %d isnt present in thearray",s)

}

getch(); }

# OUTPUT:

enter the 5 elements in array: 5

4

3

2

1

enter element to search: 3 3 found ar location 3

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of Max-heap sort algorithm. | |
| *Sem. V* | EXPERIMENT NO. : **03** | DATE : |

# AIM:

## Implementation of Max-heap sort algorithm.

# THEORY:

***Heap Sort is a popular and efficient sorting algorithm in computer programming. Learning how to write the heap sort algorithm requires knowledge of two types of data structures - arrays and trees.***

***The initial set of numbers that we want to sort is stored in an array e.g. [10, 3, 76, 34, 23, 32] and after sorting, we get a sorted array [3,10,23,32,34,76]***

***Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called heap.***

# PROGRAM:

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

void heapify(int arr[],int n,int i)

{

int largest=i; int l=2\*i+1; int r=2\*i+2;

if(l<n && arr[l] > arr[largest]) largest=l;

if(r < n && arr[r] > arr[largest]) largest=r;

if(largest!=i)

{

int swap=arr[i]; arr[i]=arr[largest]; arr[largest]=swap; heapify(arr,n,largest);

}

}

void heapsort(int arr[],int n)

{

int i;

for(i=n/2-1;i>=0;i--)

heapify(arr,n,i); for(i=n-1;i>=0;i--)

{

int swap=arr[0]; arr[0]=arr[i]; arr[i]=swap;

Heapify(arr,i,0);

}

}

void printarray(int arr[],int n)

{

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

printf("\t%d",arr[i]);

}

void main()

{

clrscr();

int arr[]={12,11,13,5,6,7};

int n=sizeof(arr)/sizeof(arr[0]); heapsort(arr,n);

printf("Sorted array:"); printarray(arr,n); getch();

# }

# OUTPUT:

Sorted array: 5 6 7 11 12 13

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE:  **Implementation and Time Analysis of factorial program using iterative method:** | |
| *Sem. V* | EXPERIMENT NO. : **04** | DATE : |

# AIM:

Implementation and Time Analysis of factorial program using iterative method:

# THEORY:

***The factorial of a positive number n is given by:***

***factorial of n (n!) = 1\*2\*3\*4....n***

***The factorial of a negative number doesn't exist. And the factorial of 0 is 1.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> unsigned long fact(int n)

{

unsigned long fact=1; int i; for(i=1;i<=n;i++)

fact=fact\*i; return fact;

}

void main()

{

clrscr(); int n=5;

printf("factorial of %d:%lu",n,fact(n)); getch();

# }

# OUTPUT:

factorial of 5:120

**Signature: Grade : / 10**

|  |  |
| --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation and Time Analysis of factorial program using Recursive method: |

# AIM:

## Implementation and Time Analysis of factorial program using Recursive method:

# THEORY:

***The factorial of a positive number n is given by:***

***factorial of n (n!) = 1\*2\*3\*4....n***

***The factorial of a negative number doesn't exist. And the factorial of 0 is 1.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> int fact(int);

void main()

{

int n=5; clrscr();

printf("factorial of %d is %d",n,fact(n)); getch();

}

int fact(int n)

{

if(n==1) { return 1; } else

{ return(n\*fact(n-1)); }

}

# OUTPUT:

factorial of 5 is 120s s**ignature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of a knapsack problem using dynamic programming. | |
| *Sem. V* | EXPERIMENT NO. : **5** | DATE : |

# AIM:

## Implementation of a knapsack problem using dynamic programming.

# THEORY:

***In the supermarket there are n packages (n ≤ 100) the package i has weight W[i] ≤ 100 and value V[i] ≤ 100. A thief breaks into the supermarket, the thief cannot carry weight exceeding M (M ≤ 100). The problem to be solved here is: which packages the thief will take away to get the highest value?***

***Input:***

* ***Maximum weight M and the number of packages n.***
* ***Array of weight W[i] and corresponding value V[i].***

# PROGRAM:

* #include<stdio.h> #include<conio.h> int max(int a,int b)

{

return (a>b)?a:b;

}

int knapsack(int W,int wt[],int val[],int n)

{

if(n==0||W==0)

return 0; if(wt[n-1]>W)

return knapsack(W,wt,val,n-1);

else return max(val[n-1]+knapsack(W-wt[n-1],wt,val,n-1),knapsack(W,wt,val,n-1));

}

void main()

{

clrscr();

int val[]={60,100,120};

int wt[]={10,20,30};

int W=50;

int n=sizeof(val)/sizeof(val[0]); printf("\nvalue:%d",knapsack(W,wt,val,n)); getch();

}

# OUTPUT:

Value: 220

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of chain matrix multiplication using dynamic programming. | |
| *Sem. V* | EXPERIMENT NO. : **6** | DATE : |

# AIM:

## Implementation of chain matrix multiplication using dynamic programming.

# THEORY:

# *****Matrix chain multiplication***** *(or Matrix Chain Ordering Problem, MCOP) is an optimization problem that to find the most efficient way to multiply given sequence of matrices. The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved.*

# PROGRAM:

* #include<stdio.h>

#include<conio.h> #include<limits.h>

int matrixchainorder(int p[],int i,int j)

{

if(i==j)

int k;

return 0;

int min=INT\_MAX; int count; for(k=i;k<j;k++)

{

count=matrixchainorder(p,i,k)+matrixchainorder(p,k+1,j)+p[i-1]\*p[k]\*p[j]; if(count<min)

min=count;

}

return min;

}

void main()

{

clrscr();

int arr[]={1,2,3,4,3};

int n=sizeof(arr)/sizeof(arr[0]);

printf("Minimum number of multiplications is%d",matrixchainorder(arr,1,n1)); getch();

}

# OUTPUT:

Minimum number of multiplications is 30

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of making a change problem using dynamic programming. | |
| *Sem. V* | EXPERIMENT NO. : **7** | DATE : |

# AIM:

## Implementation of making a change problem using dynamic programming.

# THEORY:

***There are infinite number of coins of n different values. These values are given. Using these coins, you have to make change for Rs. Sum. Find the minimum number of coins required to change Rs. Sum.***

***Note that the problem is different from coin change problem. In coin change problem, we were asked to find out in how many ways the change can be made. In this problem, we are supposed to find out the minimum number of coins required to change***.

# PROGRAM:

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

int count(int s[],int m,int n)

{

if(n==0)

return 1;

if(n<0)

return 0;

if(m<=0&&n>=1)

return 0;

return count(s,m-1,n)+count(s,m,n-s[m-1]);

}

void main()

{

clrscr(); int i,j;

int arr[]={1,2,3};

int m=sizeof(arr)/sizeof(arr[0]); printf("%d",count(arr,m,4)); getch();

# }

# OUTPUT:

4

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of a knapsack problem using greedy algorithm. | |
| *Sem. V* | EXPERIMENT NO. : **8** | DATE : |

# AIM:

## Implementation of a knapsack problem using greedy algorithm.

# THEORY:

***Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.***

***The knapsack problem is in combinatorial optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.***

# PROGRAM:

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

void knapsack(int n,float weight[],float profit[],float capacity)

{

float x[20],tp=0; int i,j,u; u=capacity; for(i=0;i<n;i++)

{

x[i]=0.0;

}

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

{

if(weight[i]>u)

{

}

if(i<n)

{

}

else

{

}

break;

x[i]=1.0;

tp=tp+profit[i]; u=u-weight[i];

x[i]=u/weight[i];

}

tp=tp+(x[i]\*profit[i]);

printf("\nThe result vector is:"); for(i=0;i<n;i++)

{

printf("%f\t",x[i]);

}

printf("\nMaximum profit is:%f",tp);

}

void main()

{

clrscr();

float weight[20],profit[20],capacity; int num,i,j;

float ratio[20],tmp;

printf("Enter the no. of objects:"); scanf("%d",&num);

printf("\nEnter the wts and profits of each objects:"); for(i=0;i<num;i++)

{

scanf("%f %f",&weight[i],&profit[i]);

}

printf("\nEnter the capacity of knapsack:"); scanf("%f",&capacity); for(i=0;i<num;i++)

{

ratio[i]=profit[i]/weight[i];

}

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

{

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

{

if(ratio[i]<ratio[j])

{

tmp=ratio[j]; ratio[j]=ratio[i]; ratio[i]=tmp; tmp=weight[j]; weight[j]=weight[i]; weight[i]=tmp; tmp=profit[j]; profit[j]=profit[i]; profit[i]=tmp;

}

}

}

knapsack(num,weight,profit,capacity); getch();}

# OUTPUT:

Enter the no. of objects: 6

Enter the wts and profits of each objects:

3 10

5 20

5 15

2 4

5 3

2 6

Enter the capacity of knapsack: 20

The result vector is: 1.000000 1.000000 1.000000 1.000000 1.000000 0.600000

Maximum profit is: 56.799999

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE: Implementation of Graph and Searching (DFS and BFS). | |
| *Sem. V* | EXPERIMENT NO. : **9** | DATE : |

# AIM:

## Implementation of Graph and Searching (DFS and BFS).

# THEORY:

# *The objective of this article is to provide a basic introduction about graphs and the commonly used algorithms used for traversing the graph, BFS and DFS. Breadth First Search (BFS) and Depth First Search (DFS) are the two popular algorithms asked in most of the programming interviews. I was not able to find a simple, precise explanation for beginners on this topic.*

# PROGRAM:

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

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20]; int delete\_g();

void add(int item); void bfs(int s,int n); void dfs(int s,int n); void push(int item); int pop();

void main()

{

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

printf("enter the number vertices:"); scanf("%d",&n);

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

{

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

{

printf("enter 1 if %d has a node with %d else 0",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");

}

do

{

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

{

vis[i]=0;

}

printf("\nMENU"); printf("\n1. B.F.S");

printf("\n2. D.F.S"); printf("\n enter your choice"); scanf("%d",&ch);

printf("enter the source vertex:"); scanf("%d",&s);

switch(ch)

{

case 1:

case 2:

}

bfs(s,n); break;

dfs(s,n); break;

printf("DO U WANT TO CONTINUE(y/n)?");

scanf("%c",&dummy);

scanf("%c",&c);

}

while((c=='y')||(c=='y'));

getch();

}

void bfs(int s,int n)

{

int p,i; add(s);

vis[s]=1; p=delete\_g(); 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\_g(); 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)

{

}

else

{

printf("QUEUE FULL");

if(rear==-1)

{

q[++rear]=item;

front++;

}

else

{

q[++rear]=item;

front++;

}

}

}

int delete\_g()

{

int k; if((front>rear)||(front==-1))

{

}

else

{

}

}

return(0);

k=q[front++]; return(k);

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)

{

}

else

{

}

} printf("stack overflow");

stack[++top]=item;

printf("stack overflow");

stack[++top]=item;

printf("%d",k);

}

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

{

if(vis[i]==0)

{

dfs(i,n);

}

}

}

}

}

void push(int item)

{

if(top==19)

{

}

else

{

}

}

printf("stack overflow");

stack[++top]=item;

int pop()

{

Int k

If(top==-1)

{

Return(0);

}

Else

{

k=stack[top--];

return(k); }

}

# OUTPUT:

enter the number vertices:3

enter 1 if 1 has a node with 1 else 0 1

enter 1 if 1 has a node with 1 else 0 1

enter 1 if 1 has a node with 1 else 0 0

enter 1 if 1 has a node with 1 else 0 1

enter 1 if 1 has a node with 1 else 0 0

enter 1 if 1 has a node with 1 else 0 1

enter 1 if 1 has a node with 1 else 0 0

enter 1 if 1 has a node with 1 else 0 1

enter 1 if 1 has a node with 1 else 0 1 the adjacency matrix is:

110

101

011

MENU

1. B.F.S
2. D.F.S

enter your choice 1

enter the source vertex: 2

213DO YOU WANT TO CONTINUE(y/n)? y MENU

1. B.F.S
2. D.F.S

enter your choice 2

enter the source vertex: 2

213DO YOU WANT TO CONTINUE(y/n)?n

**Signature: Grade : / 10**

|  |  |  |
| --- | --- | --- |
| SUBJECT:  **Analysis & Design of Algorithms** | TITLE:  **Implement LCS Problem** | |
| *Sem. V* | EXPERIMENT NO. : **10** | DATE : |

# AIM:

# Implement LCS Problem

# THEORY:

***The longest common subsequence (LCS) problem is the problem of finding the longest subsequence that is present in given two sequences in the same order. i.e. find a longest sequence which can be obtained from the first original sequence by deleting some items, and from the second original sequence by deleting other items.***

# PROGRAM:

* #include<stdio.h> #include<conio.h> #include<string.h> int i,j,m,n,c[20][20];

char x[20],y[20],b[20][20];

void print(int i,int j)

{

if(i==0 || j==0)

return; if(b[i][j]=='c')

{

print(i-1,j-1); printf("%c",x[i-1]);

}

else if(b[i][j]=='u')

print(i-1,j);

}

Void lcs()

{

else

print(i,j-1);

m=strlen(x); n=strlen(y); for(i=0;i<=m;i++)

c[i][0]=0;

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

c[0][i]=0;

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

{

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

{

if(x[i-1]==y[j-1])

{

c[i][j]=c[i-1][j-1]+1;

b[i][j]='c';

}

else if(c[i-1][j]>=c[i][j-1])

{

c[i][j]=c[i-1][j];

b[i][j]='u';

}

}

}

}

void main()

{

clrscr();

printf("enter 1st sequence:"); scanf("%s",x);

printf("enter 2nd sequence:"); scanf("%s",y);

printf("\n the longest commin subsequence is:"); lcs();

print(m,n);

getch();

}

**OUTPUT**:

enter 1st sequence: ACFGHD enter 2nd sequence: ABFHD

the longest common subsequence is: AFHD

**Signature: Grade : / 10**