DS list of practicals

1. Calculating floor and ceiling
2. Recursive factorial algorithm
3. Recursive power function
4. Recursive gcd calculation
5. Recursive linear search
6. Recursive calculation of power modulo m
7. Implementing Euclidean algorithm
8. Implementing extended Euclidean algorithm
9. Addition of two bit strings.
10. Division and remainder
11. Solving linear congruence.
12. Solving Chinese remainder problem
13. Join of two Boolean matrices
14. Meet of two Boolean matrices
15. Boolean product of two Boolean matrices
16. Testing properties of relations
17. Dijkstra's algorithm for shortest-path problem.

Calculating floor and ceiling:

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

void main(){
   double x, f, c;
   char q;
   do{
      printf("\n Enter a number: ");
      scanf("%lf", &x);
      f=floor(x);
      c=ceil(x);
      printf("\n The floor and ceiling of %lf is %lf and %lf respectively.", x, f, c);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   } while (q=='y');
}
```

Recursive factorial function:

```
#include <stdio.h>

long factorial(int n){
   if (n==0) return 1;
   else return n*factorial(n-1);
}
void main(){
   int n;
   long fact;
   char q;
   do{
      printf("\n Enter a nonnegative integer: ");
      scanf("%d", &n);
      fact=factorial(n);
      printf("\n The factorial of %d is %ld.", n, fact);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
    }
    while (q=='y');
 }
```

Recursive power function:

```
#include <stdio.h>

double power(float a, int n){
   if (n==0) return 1;
   else return a*power(a, n-1);
}
void main(){
   float a;
   int n;
   double pow;
   char q;
   do{
      printf("\n Enter a nonzero real number: ");
      scanf("%f", &a);
      printf("\n Enter a nonnegative integer: ");
      scanf("%d", &n);
      pow=power(a, n);
      printf("\n The value of %f raised to the power %d is %f.", a, n, pow);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   }
   while (q=='y');
}
```

Recursive gcd calculation:

```
#include <stdio.h>

int gcd(int a, int b){
   if (a==0) return b;
   else return gcd(b%a, a);
}
void main(){
   int a, b, g;
   char q;
   do{
      printf("\n Enter two nonnegative integers a and b: ");
      scanf("%d%d", &a, &b);
      g=gcd(a, b);
      printf("\n gcd(%d, %d)=%d", a, b, g);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   }
   while (q=='y');
}
```

① gcd(143, 227)=1

② gcd(24, 138)=6

③ gcd(1769, 2378)=29

④ gcd(272, 1479)=17

⑤ gcd(56, 72)=8

Recursive linear search:

```
#include <stdio.h>
#define max 20

int search(int a[], int i, int j, int x){
   if (a[i]==x) return i;
   else if (i==j) return 0;
   else return search(a, i+1, j, x);
}
void main(){
   int a[max], n, x, i=1, j, k, location;
   char q;
   printf("\n Enter the number of elements in the list: ");
   scanf("%d", &n);
   j=n;
   printf("\n Enter the elements of the list: ");
   for(k=1;k<=n;k++)
   scanf("%d", &a[k]);
   do{
        printf("\n Enter the element to search in the list: ");
        scanf("%d", &x);
        location = search(a, i, j, x);
        printf("\n The location of the element %d is at %d.", x, location);
        printf("\n Do you want to search another element?(y/n): ");
        scanf(" %c", &q);
   }
   while (q=='y');
}
```

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
2	4	3	9	12	7	11	1	15

Search $x = 9$: location = 4

Search $x = 5$: location = 0

Recursive calculation of power modulo m:

```
#include <stdio.h>

int mpower(int b, int n, int m){
   if (n==0) return 1;
   else if (n%2==0) return (mpower(b, n/2, m)*mpower(b, n/2, m)) % m;
   else return (((mpower(b, (n-1)/2, m)*mpower(b, (n-1)/2, m)) % m)*b%m)%m;
}
void main(){
   int b, n, m, mpow;
   char q;
   do{
      printf("\n To calculate b^n mod m");
      printf("\n Enter integers b, n and m:");
      scanf("%d%d%d", &b, &n, &m);
      mpow=mpower(b, n, m);
      printf("\n %d^%d mod %d=%d", b, n, m, mpow);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   }
   while (q=='y');
}
```

(1) $2^5 \bmod 11 = 10$

(2) $3^7 \bmod 4 = 3$

(3) $3^6 \bmod 5 = 4$

Implementing Euclidean algorithm:

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

void main(){
   int x, y, r;
   char q;
   do{
      printf("\n Enter first positive integer: ");
      scanf("%d", &x);
      printf("\n Enter second positive integer: ");
      scanf("%d", &y);
      while (y!=0){
         r=x%y;
         x=y;
         y=r;
      }
      printf("\n The gcd of these integers is %d.", x);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   } while (q=='y');
}
```

① gcd(143, 227) = 1

② gcd(24, 138) = 6

③ gcd(1769, 2378) = 29

④ gcd(272, 1479) = 17

⑤ gcd(56, 72) = 8

Implementing extended Euclidean algorithm:

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

void main(){
   int x, y;
   char b;
   do{
      printf("\n Enter first positive integer: ");
      scanf("%d", &x);
      printf("\n Enter second positive integer: ");
      scanf("%d", &y);
      int a=x, b=y;
      int s1=0, s2=1, t1=1, t2=0;
      int q, r, s, t;
      while (b!=0){
        q=a/b; r=a%b; s=s2-q*s1; t=t2-q*t1;
        a=b; b=r; s2=s1; s1=s; t2=t1; t1=t;
      }
      printf("\n The gcd of %d and %d is %d with coefficients %d and %d respectively.", x, y, a, s2,
t2);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &b);
   } while(b=='y');
}
```

(1) $\gcd(143, 227) = 1 = (-100) \times 143 + 63 \times 227$

(2) $\gcd(24, 138) = 6 = 6 \times 24 + (-1) \times 138$

(3) $\gcd(1769, 2378) = 29 = 39 \times 1769 + (-29) \times 2378$

(4) $\gcd(272, 1479) = 17 = (-38) \times 272 + 7 \times 1479$

(5) $\gcd(56, 72) = 8 = 4 \times 56 + (-3) \times 72$

Addition of two bit strings:

```
#include <stdio.h>
#include <math.h>
#define max 20

void main()
{
   int a[max], b[max], s[max], n;
   char q;
   do{
      int i, c, d;
      printf("\n Enter the length of bit strings ");
      scanf("%d", &n);
      printf("\nEnter first bit string, one bit at a time: ");
      for (i=n-1;i>=0;i--)
      scanf("%d", &a[i]);
      printf("\nEnter second bit string, one bit at a time: ");
      for (i=n-1;i>=0;i--)
      scanf("%d", &b[i]);
      c=0;
      for(i=0;i<=n-1;i++){
         d=floor((a[i]+b[i]+c)/2);
         s[i]=a[i]+b[i]+c-(2*d);
         c=d;
      }
      s[n]=c;
      printf("\n The sum of the bit strings is: ");
      for(i=n;i>=0;i--)
      printf("%d", s[i]);
      printf("\n Do you want to continue?(y/n):");
      scanf(" %c", &q);
   }
   while (q=='y');
}
```

(1)

```
  1 1 0 1 0 1
  1 0 1 1 1 0
-------------
1 1 0 0 0 1 1
```

(2)

```
  0 1 1 0 0
  1 1 1 0 1
-----------
1 0 1 0 0 1
```

Division and Remainder:

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

void main(){
   int a, d;
   char q, p;
   do{
      printf("\n Enter a divided: ");
      scanf("%d", &a);
      printf("\n Enter a positive divisor: ");
      scanf("%d", &d);
      int q=0;
      int r=abs(a);
      while (r>=d){
         r=r-d;
         q=q+1;
      }
      if (a<0){
         if (r>0) then{
            r=d-r;
            q=-(q+1);
         }
         elseif (r==0) then
         q=-q;
      }
      printf("\n The quotient and remainder are %d and %d respectively.", q, r);
      printf("\n Do you want to continue?(y/n): ");
      scanf("%c", &p);
   }
   while (p=='y');
}
```

Dividend:	11	7	22	-35	-57
Divisor:	2	9	11	7	4
quotient:	5	0	2	-5	-15
remainder:	1	7	0	0	3

Solving linear congruence:

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

int inverse(int a, int b){
   int s1=0, s2=1, t1=1, t2=0;
      int q, r, s, t;
      while (b!=0){
        q=a/b; r=a%b; s=s2-q*s1; t=t2-q*t1;
        a=b; b=r; s2=s1; s1=s; t2=t1; t1=t;
      }
   return s2;
}


void main(){
   int a, b, m, s, soln;
   char q;
   printf("\n Solving the linear congruence ax=b (mod m)\n");
   do{
      printf("\n Enter the integers a, b and m: ");
      scanf("%d%d%d", &a, &b, &m);
      s=inverse(a, m);
      printf("\n Solution of the given linear congruence is %d ", s*b);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   } while (q=='y');
}
```

Solving Chinese remainder theorem:

```
#include <stdio.h>
#include <math.h>
#define max 10

int inverse(int a, int b){
   int s1=0, s2=1, t1=1, t2=0;
      int q, r, s, t;
      while (b!=0){
        q=a/b; r=a%b; s=s2-q*s1; t=t2-q*t1;
        a=b; b=r; s2=s1; s1=s; t2=t1; t1=t;
      }
   return s2;
}

void main(){
   int n, i, M, a[max], m[max], s[max], Mk[max];
   char q;
   do{
      printf("\n Enter the number of equations: ");
      scanf("%d", &n);
      printf("\n Enter the integers a and m one line at a time: ");
      for(i=1;i<=n;i++)
         scanf("%d%d", &a[i], &m[i]);
      M=1;
      for(i=1;i<=n;i++)
         M=M*m[i];
      for(i=1;i<=n;i++)
         Mk[i]=M/m[i];
      for(i=1;i<=n;i++)
         s[i]=inverse(Mk[i], m[i]);
      int x=0;
      for(i=1;i<=n;i++)
         x=x+a[i]*Mk[i]*s[i];
      x=x % M;
      printf("\n The solution of the given system is %d+%dk where k is an integer.", x, M);
      printf("\n Do you want to continue?(y/n): ");
      scanf(" %c", &q);
   } while (q=='y');
}
```

(1) $x \equiv 1 \pmod 3$, $x \equiv 2 \pmod 5$, $x \equiv 3 \pmod 7$: 52

(2) $x \equiv 5 \pmod 6$, $x \equiv 4 \pmod{11}$, $x \equiv 3 \pmod{17}$: 785

(3) $x \equiv 1 \pmod 5$, $x \equiv 9 \pmod 6$, $x \equiv 1 \pmod 7$, $x \equiv 9 \pmod{11}$: 141

Join of two Boolean matrices:

```
#include <stdio.h>
#define rmax 10
#define cmax 10

void main(){
   int A[rmax][cmax], B[rmax][cmax];
   int r, c, i, j;
   printf("\n Enter the number of rows and columns of the matrices: ");
   scanf("%d%d", &r, &c);
   printf("\n Enter the elements of the first matrix A, one row at a time:\n");
   for(i=0;i<=r-1;i++)
      for(j=0;j<=c-1;j++)
         scanf("%d", &A[i][j]);
   printf("\n Enter the elements of the second matrix B, one row at a time:\n");
   for(i=0;i<=r-1;i++)
      for(j=0;j<=c-1;j++)
         scanf("%d", &B[i][j]);
   printf("\n The join of the matrices A and B is:\n");
   for(i=0;i<=r-1;i++){
      for(j=0;j<=c-1;j++)
         printf("%d ", A[i][j] || B[i][j]);
      printf("\n");
   }
}
```

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A \vee B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Meet of two Boolean matrices:

```
#include <stdio.h>
#define rmax 10
#define cmax 10

void main(){
   int A[rmax][cmax], B[rmax][cmax];
   int r, c, i, j;
   printf("\n Enter the number of rows and columns of the matrices: ");
   scanf("%d%d", &r, &c);
   printf("\n Enter the elements of the first matrix A, one row at a time:\n");
   for(i=0;i<=r-1;i++)
      for(j=0;j<=c-1;j++)
         scanf("%d", &A[i][j]);
   printf("\n Enter the elements of the second matrix B, one row at a time:\n");
   for(i=0;i<=r-1;i++)
      for(j=0;j<=c-1;j++)
         scanf("%d", &B[i][j]);
   printf("\n The meet of the matrices A and B is:\n");
   for(i=0;i<=r-1;i++){
      for(j=0;j<=c-1;j++)
         printf("%d ", A[i][j] && B[i][j]);
      printf("\n");
   }
}
```

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A \wedge B = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Boolean product of two Boolean matrices:

```
#include <stdio.h>
#define rmax 10
#define cmax 10

void main(){
   int A[rmax][cmax], B[rmax][cmax], C[rmax][cmax];
   int r1, c1, r2, c2, i, j, k;
   printf("\n Enter the number of rows and columns of the first matrix: ");
   scanf("%d%d", &r1, &c1);
   printf("\n Enter the number of rows and columns of the second matrix: ");
   scanf("%d%d", &r2, &c2);
   if (c1!=r2)
   printf("\n The dimensions of the matrices do not match.");
   else{
      printf("\n Enter the elements of the first matrix A, one row at a time:\n");
      for(i=0;i<=r1-1;i++)
         for(j=0;j<=c1-1;j++)
            scanf("%d", &A[i][j]);
      printf("\n Enter the elements of the second matrix B, one row at a time:\n");
      for(i=0;i<=r2-1;i++)
         for(j=0;j<=c2-1;j++)
            scanf("%d", &B[i][j]);
      printf("\n The join of the matrices A and B is:\n");
      for(i=0;i<=r1-1;i++){
         for(j=0;j<=c2-1;j++){
            C[i][j]=0;
            for(k=0;k<=r2-1;k++)
               C[i][j]=C[i][j] || (A[i][k] && B[k][j]);
            printf("%d ", C[i][j]);
         }
         printf("\n");
      }
   }
}
```

Boolean product

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$A \odot B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Testing properties of relations:

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

void main()
{
   int i, j, n, mat[max][max];
   printf("Enter the number of elements in set A: ");
   scanf("%d", &n);
   printf("\n Enter the matrix representation of relation");
   printf("\n one row at a time: \n");
   for(i=0;i<=n-1;i++)
      for(j=0;j<=n-1;j++)
         scanf("%d", &mat[i][j]);
   int key=1;
   for(i=0;i<=n-1;i++)
      key=key && mat[i][i];
   if (key==1)
      printf("\n The given relation is reflexive.");
   else
      printf("\n The given relation is not reflexive.");
   key=1;
   for(i=0;i<=n-1;i++)
      for(j=0;j<=n-1;j++)
         if (mat[i][j]!=mat[j][i]) key=0;
   if (key==1)
      printf("\n The given relation is symmetric.");
   else
      printf("\n The given relation is not symmetric.");
   key=1;
   for(i=0;i<=n-1;i++)
      for(j=0;j!=i && j<=n-1;j++)
         if ((mat[i][j] && mat[j][i])==1) key=0;
    if (key==1)
      printf("\n The given relation is antisymmetric.");
   else
      printf("\n The given relation is not antisymmetric.");
}
```

① $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ → $A=\{a,b,c\}$, $R=\{(a,a),(a,c),(b,b),(b,c),(c,a),(c,b),(c,c)\}$ → reflexive, symmetric, not antisymmetric.

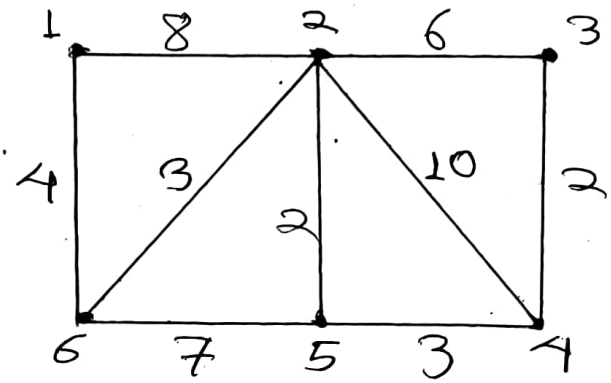
② $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ → $A=\{a,b,c,d\}$, $R=\{(a,a),(a,d),(b,c),(c,a),(c,b),(d,b),(d,d)\}$ → not reflexive, not symmetric, not antisymmetric.

③ $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ → $A=\{a,b,c\}$, $R=\{(a,a),(a,c),(b,b)\}$ → not reflexive, not symmetric, antisymmetric.

Dijkstra's algorithm for shortest path problem:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);
 void main()
{
   int G[MAX][MAX], i, j, n, u;
   printf("Enter number of vertices:");
   scanf("%d", &n);
   printf("\n Enter the cost adjacency matrix with entry 0 for nonadjacent vertices:\n");

   for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
         scanf("%d", &G[i][j]);

   printf("\n Enter the starting node:");
   scanf("%d", &u);
   dijkstra(G, n, u);
}
 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;

   //pred[] stores the predecessor of each node
   //count gives the number of nodes seen so far
   //create the cost matrix
   for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
         if(G[i][j]==0)
            cost[i][j]=INFINITY;
         else
            cost[i][j]=G[i][j];

   //initialize pred[], distance[] and visited[]
   for(i=1; 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;

      //nextnode gives the node at minimum distance
      for(i=1; i<=n; i++)
         if(distance[i]<mindistance && !visited[i])
```

```
        {
            mindistance=distance[i];
            nextnode=i;
        }

        //check if a better path exists through nextnode
        visited[nextnode]=1;
        for(i=1; i<=n; i++)
            if(!visited[i])
                if(mindistance+cost[nextnode][i]<distance[i])
                {
                    distance[i]=mindistance+cost[nextnode][i];
                    pred[i]=nextnode;
                }
        count++;
    }

    //print the path and distance of each node
    for(i=1; i<=n; i++)
        if(i!=startnode)
        {
            printf("\n Distance of node%d=%d", i, distance[i]);
            printf("\n Path=%d", i);

            j=i;
            do
            {
                j=pred[j];
                printf("<-%d", j);
            }while(j!=startnode);
        }
}
```



Start node = 1

	1	2	3	4	5	6
1	0	8	0	0	0	4
2	8	0	6	10	2	3
3	0	6	0	2	0	0
4	0	10	2	0	3	0
5	0	2	0	3	0	7
6	4	3	0	0	7	0

Path	Length
1→2 : 2←6←1	7
1→3 : 3←2←6←1	13
1→4 : 4←5←2←6←1	12
1→5 : 5←2←6←1	9
1→6 : 6←1	4