// write program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem //

Algorithm

```
max(a,b)

return (a > b)? a : b

End


knapSack(W, wt, p, n)

  i, w, x[10]

 int K[n+1][W+1]

 for  i = 0 to n do

 for  w = 0 to W do

 if (i==0 || w==0) then

 Initialize K[i][w] = 0

 else if (wt[i-1] <= w) then

 K[i][w] = max(p[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])

 else

 K[i][w] = K[i-1][w]

 End for

 End for

 return K[n][W]

End

main function

 Read n,W

 for i = 0 to n do
 Read wt[i]
 for i = 0 to n do
Read p[i]
Call Function knapSack(W, wt, p, n)
 return
End
```

Program

```
#include<stdio.h>
int max(int a, int b)
 {
return (a > b)? a : b;
 }
int knapSack(int W, int wt[], int p[], int n)
 {
 int i, w, x[10];
 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(p[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 i, n, p[20], wt[20], W, x[10];

 printf("Enter number of objects:");

 scanf("%d", &n);
 printf("Enter the weight of objects:\n");

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

 {

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

 }

 printf("Enter the profits of objects:\n");

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

 {

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

 }

 printf("Enter size of knapsack:");

 scanf("%d", &W);

 printf("The optimal solution is %d", knapSack(W, wt, p, n));

return 0;
```

```
}
```

Input/Output

```
Enter number of objects: 4

Enter the weight of objects:

2 1 3 2

Enter the profits of objects:

12 10 20 15

Enter size of knapsack: 5

The optimal solution is 37
```

//Write program to implement the DFS and BFS algorithm for a graph//

Algorithm

```
int G[10][10],visited[10],n,v,i,j,queue[10],front=-1,rear=-1;

Main function

Read n

for i=0 to n do

for j=0 to n do

Read G[i][j]

for i=0 to n do

visited[i]=0

queue[i]=0

Call function DFS(0)

Read v

Call function bfs(v)

for i = 0 to n do

if (visited[i]) then

Print  i

End if

 End for

return 0

End

DFS(I)

Print i

visited[i]=1

for j=0 to n do
```

```
if(!visited[j]&&G[i][j]==1) then
Recursively call DFS(j)
End
 bfs(v)
for i = 1 to n do
if (G[v][i] && !visited[i]) then
queue[++rear] = i
if (front <= rear) then
visited[queue[front]] = 1
bfs(queue[front++])
End if
End
```

Program

```
#include<stdio.h>
void DFS(int);
void bfs(int);
int G[10][10],visited[10],n,v,i,j,queue[10],front=-1,rear=-1;
int main()
{
int i,j;
printf("Enter number of vertices:");
scanf("%d",&n);
//read the adjecency matrix
printf("\nEnter adjecency matrix of the graph:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
//visited is initialized to zero
for(i=0;i<n;i++)
visited[i]=0;
queue[i]=0;
printf("\nDFS algorithm\n");
DFS(0);
printf("\nEnter the starting vertex:\n");
scanf("%d",&v);
bfs(v);
printf("The node which are reachable using BFS algorithm are: \n");
for (i = 0; i <n; i++)
{
if (visited[i])
{
printf("%d\t", i);
}
}
return 0;
}
void DFS(int i)
{
int j;
```

```
printf("\n%d",i);

visited[i]=1;

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

if(!visited[j]&&G[i][j]==1)

DFS(j);

}

void bfs(int v)

{

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

if (G[v][i] && !visited[i])

queue[++rear] = i;

if (front <= rear)

{

visited[queue[front]] = 1;

bfs(queue[front++]);

}

}
```

Input/Output

```
Enter number of vertices:6

Enter adjecency matrix of the graph:

0 1 1 1 0 0

1 0 0 0 0 0

1 0 0 0 1 1

1 0 0 0 0 0

0 0 1 0 0 0

0 0 1 0 0 0

DFS algorithm

0

1

2

4

5

3

Enter the starting vertex:

0

The node which are reachable using BFS algorithm are:

0 1 2 3 4 5
```

// Write program to implement backtracking algorithm for solving problems like Nqueens//

Algorithm

```
Initialize count=0
 place(pos)
for i=1 to pos do
if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos)))) then
return 0
End for
return 1
End
 print_sol(n)
Increment count by one unit
Print count
for i=1 to n do
for j=1 to n do
if(a[i]==j) then
Print Q
else Print *
End for
End
queen(n)
Initialize k=1, a[k]=0
while(k!=0) do
a[k]=a[k]+1
```

```
while((a[k]<=n)&&!place(k)) do
Increment a[k] by one unit
if(a[k]<=n) then
if(k==n) then
Call function print_sol(n)
else
Increment k by one unit
Initialize a[k]=0
End else
End if
else decrement k by one unit
End while
End
 main finction
Read n
Call function queen(n)
Print count
End
```

Program

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

getch();

}
```

Input/Output

```
Enter the number of Queens

4

Solution #1:

* Q * *

* * * Q

Q * * *

* * Q *

Solution #2:

* * Q *

Q * * *

* * * Q

* Q * *

Total solutions=2
```

Input/Output

```
Enter the number of Queens

2

Total solutions=0
```

// Write c program to implement the backtracking algorithm for the sum of subsets problem//

Algorithm

```
subset(i,wt,total)
 return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)))
End
main function
{
 Initialize total=0;
 Read n
 Print n
 for i=0 to n do
  Read w[i]
  total+=w[i]
End for
 Read sum
 for i=0 to n do
  for j=0 to n-1 do
   if(w[j]>w[j+1]) then
    temp=w[j]
    w[j]=w[j+1]
    w[j+1]=temp
   End if
 Print n
 for i=0 to n do
```

```
  Print w[i]
 if((total<sum)) then
  Print “Subset construction is not possible"
 else
  for i=0 to n do
   Initialize inc[i]=0
  Call function sumset(-1,0,total)
End else
End
sumset(i,wt,total)
 if(subset(i,wt,total)) then
  if(wt==sum)
   for j=0 to i do
    if(inc[j]) then
     Print w[j]
  End if
  else
   inc[i+1]=TRUE
  Call function sumset(i+1,wt+w[i+1],total-w[i+1])
   inc[i+1]=FALSE
Call function sumset(i+1,wt,total-w[i+1])
  End else
 End if
End
```

Program

```
#include<stdio.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

int inc[50],w[50],sum,n;

int subset(int i,int wt,int total)

{

 return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}

void main()

{

 int i,j,n,temp,total=0;

 printf("Enter how many numbers:");

 scanf("%d",&n);

 printf("Enter %d numbers to th set:",n);

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

 {

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

  total+=w[i];

 }

 printf("Input the sum value to create sub set:");

 scanf("%d",&sum);

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

  for(j=0;j<n-1;j++)
```

```
   if(w[j]>w[j+1])
   {
    temp=w[j];
    w[j]=w[j+1];
    w[j+1]=temp;
   }
 printf("The given %d numbers in ascending order:",n);
 for(i=0;i<n;i++)
  printf("%d\t",w[i]);
 if((total<sum))
  printf("\nSubset construction is not possible");
 else
 {
  for(i=0;i<n;i++)
   inc[i]=0;
  printf("\nThe solution using backtracking is:");
  sumset(-1,0,total);
 }
 getch();
}
void sumset(int i,int wt,int total)
{
 int j;
 if(subset(i,wt,total))
 {
  if(wt==sum)
  {
   printf("\n{");
   for(j=0;j<=i;j++)
    if(inc[j])
     printf("%d\t",w[j]);
   printf("}\n");  }
  else
  {
   inc[i+1]=TRUE;
   sumset(i+1,wt+w[i+1],total-w[i+1]);
   inc[i+1]=FALSE;
sumset(i+1,wt,total-w[i+1]);
  } }}
```

Output

```
Enter how many numbers:6

Enter 6 numbers to the set:10  5  13  15  12  18

Input the sum value to create sub set:30

The given 6 numbers in ascending order:5  10  12  13  15  18

The solution using backtracking is:

{ 5  10  15 }

{ 5  12  13 }

{ 12  18 }
```