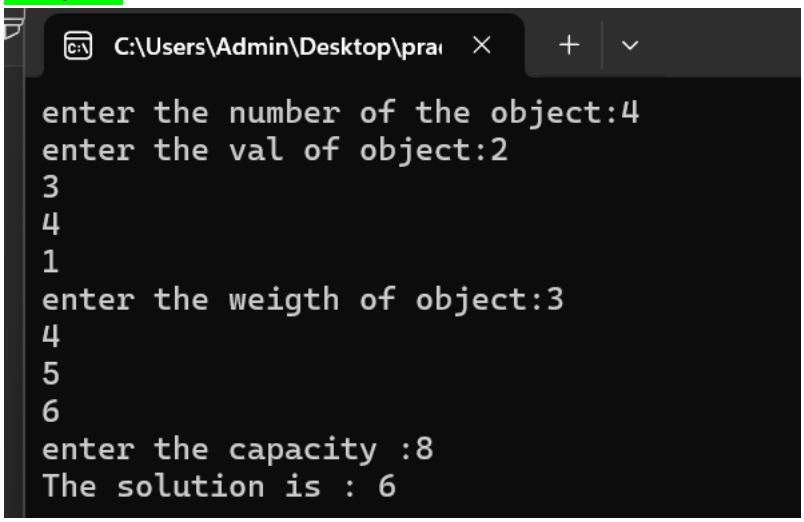
1. Knapsack using dynamic programming

Program:

```
#include<stdio.h>
int max(int a, int b) {
 if(a>b){
 return a;
 } else {
 return b;
 }
}
int knapsack(int W, int wt[], int val[], int n) {
 int i, w;
 int knap[n+1][W+1];
 for (i = 0; i <= n; i++) {
 for (w = 0; w <= W; w++) {
 if (i==0 || w==0)
 knap[i][w] = 0;
 else if (wt[i-1] <= w)
 knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
 else
 knap[i][w] = knap[i-1][w];
 }
 }
 return knap[n][W];
}
int main() {
 int val[10],i,n;
 int wt[20];
 int W;
 printf("enter the number of the object:");
 scanf("%d",&n);
 printf("enter the val of object:");
 for(i=0;i<n;i++)
 {
 scanf("%d",&val[i]);
 }
```

```
 printf("enter the weigth of object:");
 for(i=0;i<n;i++)
 {
 scanf("%d",&wt[i]);
 }
 printf("enter the capacity :");
 scanf("%d",&W);
 printf("The solution is : %d", knapsack(W, wt, val, n));
 return 0;
}
```

Output:



2. Using Dynamic programming concept to find out Optimal binary search tree.

Program:

```
#include <stdio.h>
int sum(int freq[], int low, int high)
{
 int sum = 0;
 for (int k = low; k <= high; k++)
{
 sum += freq[k];
 }
 return sum;
}
int minCostBST(int keys[], int freq[], int n)
{
 int cost[n][n];
```

```
 for (int i = 0; i < n; i++)
{
 cost[i][i] = freq[i];
 }
 for (int length = 2; length <= n; length++)
{
 for (int i = 0; i <= n - length + 1; i++)
{
 int j = i + length - 1;
 cost[i][j] =999;
 for (int r = i; r <= j; r++)
{
 int c = 0;
 if (r > i)
{
 c += cost[i][r - 1];
 }
 if (r < j)
{
 c += cost[r + 1][j];
 }
 c += sum(freq, i, j);
 if (c < cost[i][j])
{
 cost[i][j] = c;
 }
 }
 }
 }
 return cost[0][n - 1];
}
int main()
{
 int keys[10], freq[10];
 int n,i;
 printf("enter the no, of nodes:");
 scanf("%d",&n);
```

```
 printf("enter the keys:");
 for(i=0;i<n;i++)
 {
 scanf("%d",&keys[i]);
}
printf("enter the freq of node:");
for(i=0;i<n;i++)
{
scanf("%d",&freq[i]);
}
 int minCost = minCostBST(keys, freq, n);
 printf("Minimum cost of optimal binary search tree: %d\n", minCost);
 return 0;
}
```

Output:

```
enter the no, of nodes:3
enter the keys:10
20
30
enter the freq of node:4
5
6
Minimum cost of optimal binary search tree: 25
```

3.Using Dynamic programming techniques to find binomial coefficient of a given number

Program:

```
#include <stdio.h>
int bin_table(int val) {
 for (int i = 0; i <= val; i++)
 {
 printf("%2d", i);
 int num = 1;
 for (int j = 0; j <= i; j++)
 {
 if (i != 0 && j != 0)
{
 num = num * (i - j + 1) / j;
 }
 printf("%4d", num);
 }
 printf(" \n");
 }
}
int main() {
 int value;
 printf("enter the value:");
 scanf("%d",&value);
 bin_table(value);
 return 0;
}
```

Output:

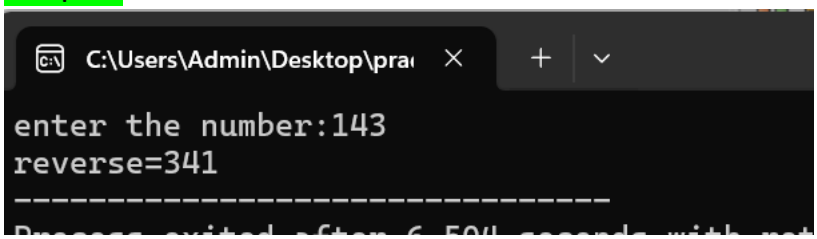
```
enter the value:5
 0   1
 1   1   1
 2   1   2   1
 3   1   3   3   1
 4   1   4   6   4   1
 5   1   5  10  10   5   1
```

4.Write a program to find the reverse of a given number using recursive.

Program:

```
#include<stdio.h>
int rev(int n,int b)
{ int d,sum=0;
if(n==0)
{
return b;
}
else
{
 return rev(n/10,b*10+n%10);
}
}
int main()
{
int n,result;
printf("enter the number:");
scanf("%d",&n);
result=rev(n,0);
printf("reverse=%d",result);
return 0;
}
```

Output:

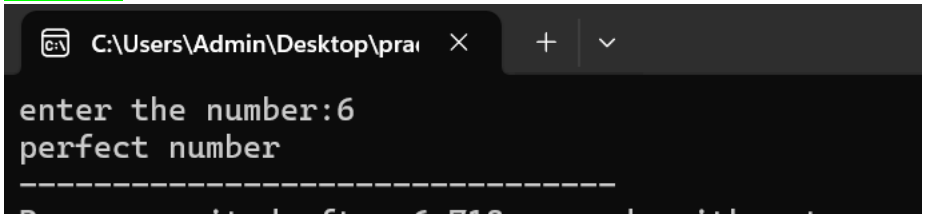


5. Write a program to find the perfect number.

Program:

```
#include<stdio.h>
int main()
{
int n,sum=0,i,temp;
printf("enter the number:");
scanf("%d",&n);
temp=n;
for(i=1;i<n;i++)
{
if(n%i==0)
{
sum+=i;
}
}
if(sum==temp)
{
printf("perfect number");
}
else
{
printf("not perfect number");
}
}
```

Output:



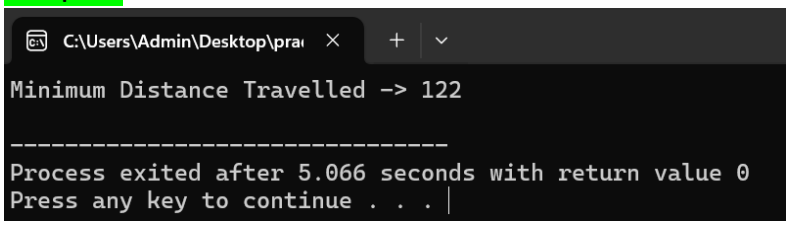
6. Write a program to perform a travelling salesman problem using dynamic programming

Program:

```
#include <stdio.h>
#include <limits.h>
#define MAX 9999
int n = 4;
int distan[20][20] = {
 {0, 22, 26, 30},
 {30, 0, 45, 35},
 {25, 45, 0, 60},
 {30, 35, 40, 0}};
int DP[32][8];
int TSP(int mark, int position) {
 int completed_visit = (1 << n) - 1;
 if (mark == completed_visit) {
 return distan[position][0];
 }
 if (DP[mark][position] != -1) {
 return DP[mark][position];
 }
 int answer = MAX;
 for (int city = 0; city < n; city++) {
 if ((mark & (1 << city)) == 0) {
 int newAnswer = distan[position][city] + TSP(mark | (1 << city), city);
 answer = (answer < newAnswer) ? answer : newAnswer;
 }
 }
 return DP[mark][position] = answer;
}
int main() {
 for (int i = 0; i < (1 << n); i++) {
 for (int j = 0; j < n; j++) {
 DP[i][j] = -1;
 }
 }
```

```
 printf("Minimum Distance Travelled -> %d\n", TSP(1, 0));
 return 0;
}
```

Output:

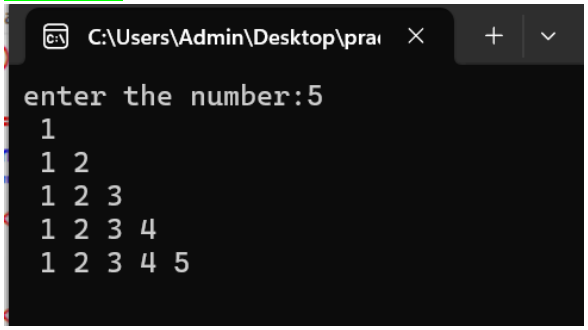


7. Write a program for the given pattern using recursion n=4

Program:

```
#include<stdio.h>
int main()
{
int i,j,k=0,n;
printf("enter the number:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
k=0;
for(j=1;j<=i;j++)
{
k=k+1;
printf(" %d",k);
}
printf("\n");
}
}
```

Output:



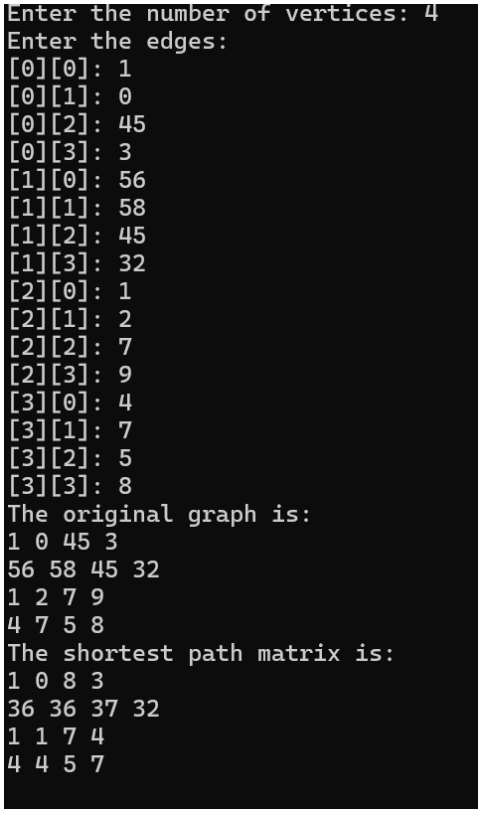
8. Write a program to perform Floyd's algorithm

Program:

```
#include <stdio.h>
#include <stdlib.h>
void floydWarshall(int **graph, int n)
{
 int i, j, k;
 for (k = 0; k < n; k++)
 {
 for (i = 0; i < n; i++)
 {
 for (j = 0; j < n; j++)
 {
 if (graph[i][j] > graph[i][k] + graph[k][j])
 graph[i][j] = graph[i][k] + graph[k][j];
 }
 }
 }
}
int main(void)
{
 int n, i, j;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 int **graph = (int **)malloc((long unsigned) n * sizeof(int *));
 for (i = 0; i < n; i++)
 {
 graph[i] = (int *)malloc((long unsigned) n * sizeof(int));
 }
 for (i = 0; i < n; i++)
 {
 for (j = 0; j < n; j++)
 {
 if (i == j)
 graph[i][j] = 0;
 else
```

```
graph[i][j] = 100;
}
}
printf("Enter the edges: \n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("[%d][%d]: ", i, j);
scanf("%d", &graph[i][j]);
}
}
printf("The original graph is:\n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("%d ", graph[i][j]);
}
printf("\n");
}
floydWarshall(graph, n);
printf("The shortest path matrix is:\n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("%d ", graph[i][j]);
}
printf("\n");
}
return 0;
}
```

Output:



9. Write a program for pascal triangle.

Program:

```
#include<stdio.h>
void displaypascaltri(int n){
	int triangle[10][10];
	for(int i=0;i<n;i++){
		triangle[0][i]=1;
	}
	for(int i=1;i<n;i++){
		for(int j=0;j<=i;j++){
			if(j==0||j==i){

			triangle[i][j]=1;
		}
		else{
			triangle[i][j]=triangle[i-1][j-1]+triangle[i-1][j];
		}
		}
	}
	for(int i=0;i<n;i++){
```

```
                for(int j=0;j<n-i-1;j++){
                        printf("  ");

                }
                for(int j=0;j<=i;j++){
                        printf("%4d",triangle[i][j]);

                }
                printf("\n");
        }
}
int main(){
        int n=5;
        displaypascaltri(n);
        return 0;

}
```

Output: