1.Warshal’s Algorithm

```
#include <stdio.h>

#define MAX 10

void printMatrix(int matrix[MAX][MAX], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void warshall(int graph[MAX][MAX], int n) {
    int reach[MAX][MAX];
    int i, j, k;

    // Initialize the reachability matrix with the input graph's adjacency matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            reach[i][j] = graph[i][j];
        }
    }

    // Applying Warshall's algorithm
```

```
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                reach[i][j] = reach[i][j] || (reach[i][k] && reach[k][j]);
            }
        }
    }

    // Print the transitive closure
    printf("Transitive closure matrix:\n");
    printMatrix(reach, n);
}

int main() {
    int n;
    int graph[MAX][MAX];

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

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

```
    warshall(graph, n);


    return 0;
}
```

Output:

Enter the number of vertices: 4

Enter the adjacency matrix:

0 1 0 0

0 0 1 0

1 0 0 1

0 0 0 1

Transitive closure matrix:

1 1 1 1

1 1 1 1

1 1 1 1

0 0 0 1

2.Floyd's Algorithm

```
#include<stdio.h>
void floyds(int a[][10],int n)
{
 int i,j,k,min,d[10][10];
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
```

```
   d[i][j]=a[i][j];
  }
 }
 for(k=0;k<n;k++)
 {
  for(i=0;i<n;i++)
  {
   for(j=0;j<n;j++)
   {
    min=d[i][j]<(d[i][k]+d[k][j])?d[i][j]:(d[i][k]+d[k][j]);
    d[i][j]=min;
   }
  }
}
printf("Distance matrix\n");
    for(i = 0; i < n; i++) {
      for(j=0;j<n;j++)
        printf("%d\t", d[i][j]);
      printf("\n");
    }

}
void main()
{
 int n, b[10][10];
    int i, j;
```

```
    printf("Enter size\n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            scanf("%d", &b[i][j]);
        }
    }
    floyds(b, n);
}
```

Output:

Enter size

4

Enter the adjacency matrix

0 99 3 99

2 0 99 99

99 6 0 1

7 99 99 0

Distance matrix

0 9 3 4

2 0 5 6

8 6 0 1

7 16 10 0

3.Knapsack Algorithm

```
#include <stdio.h>

int v[10][10];

void knapsack(int n, int m, int w[], int p[]) {
    int i, j, max;
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            if (i == 0 || j == 0)
                v[i][j] = 0;
            else if (w[i - 1] > j)
                v[i][j] = v[i - 1][j];
            else {
                max = v[i - 1][j] > (v[i - 1][j - w[i - 1]] + p[i - 1]) ? v[i - 1][j] : (v[i -
1][j - w[i - 1]] + p[i - 1]);
                v[i][j] = max;
            }
        }
    }

    printf("Optimal solution\n");
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++)
            printf("%d\t", v[i][j]);
        printf("\n");
```

```
    }
}

int main() {
    int n, m, p[10], w[10];
    int i;

    printf("\nEnter the number of items: ");
    scanf("%d", &n);
    printf("\nEnter the capacity of the knapsack: ");
    scanf("%d", &m);
    printf("\nEnter profits:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);
    printf("\nEnter weights:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);

    knapsack(n, m, w, p);

    return 0;
}
```

Output:

Enter the number of items: 4

Enter the capacity of the knapsack: 5

Enter profits:

12 13 10 7

Enter weights:

2 3 4 6

Optimal solution

0	0	0	0	0	0
0	0	12	12	12	12
0	0	12	13	13	25
0	0	12	13	13	25
0	0	12	13	13	25