1st program Kruskal

```
#include<stdio.h>

int cost[10][10], n;

void kruskal() {
   int par[10];
   int a = 0, b = 0, u = 0, v = 0, min, mincost = 0, ne = 0;

   for(int i = 0; i < n; i++)
      par[i] = -1;

   printf("The minimum spanning tree edges are...\n");

   while(ne < n-1) {
      // Find the least cost edge
      min = 999;
      for(int i = 0; i < n; i++) {
         for(int j = 0; j < n; j++) {
            if(cost[i][j] < min) {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
         }
      }

      // Check if edge selected causes a cycle
      while(par[u] != -1)
         u = par[u];
```

```
        while(par[v] != -1)
            v = par[v];

        if(u != v) {
            printf("From vertex %d to vertex %d and the cost = %d\n", a, b, min);
            mincost += min;
            par[v] = u;
            ne++;
        }
        // Edge included in MST should not be considered for next iteration
        cost[a][b] = cost[b][a] = 999;
    }

    printf("Cost of MST = %d\n", mincost);
}

int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the cost matrix\n");

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    kruskal();
}
```

2nd program prims

```
#include <stdio.h>

int cost[10][10], n;

void prim() {
   int vt[10] = {0};
   int a = 0, b = 0, min, mincost = 0, ne = 0;

   // Start from the first vertex
   vt[0] = 1;

   printf("The edges of the minimum spanning tree are:\n");

   while (ne < n-1) {
      // Find the nearest neighbor
      min = 999;
      for (int i = 0; i < n; i++) {
         if (vt[i] == 1) {
            for (int j = 0; j < n; j++) {
               if (cost[i][j] < min && vt[j] == 0) {
                  min = cost[i][j];
                  a = i;
                  b = j;
               }
            }
         }
      }

      // Include nearest neighbor 'b' into MST
      printf("Edge from vertex %d to vertex %d and the cost %d\n", a, b, min);
```

```
        vt[b] = 1;
        ne++;
        mincost += min;
        cost[a][b] = cost[b][a] = 999;
    }

    printf("Minimum spanning tree cost is %d\n", mincost);
}

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

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

    prim();
}
```

3a floyds

```
#include <stdio.h>

int min(int a, int b) {
   return (a < b ? a : b);
}

void floyd(int D[][10], int n) {
   for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
}

int main() {
   int n, cost[10][10];
   printf("Enter number of vertices: ");
   scanf("%d", &n);

   printf("Enter the cost matrix\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);

   floyd(cost, n);

   printf("All pair shortest paths:\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
         printf("%d ", cost[i][j]);
      printf("\n");
```

```
    }

    return 0;
}
```

3B

```
#include <stdio.h>

void warshall(int A[][10], int n) {
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                A[i][j] = A[i][j] || (A[i][k] && A[k][j]);
}

int main() {
    int n, adj[10][10];
    printf("Enter 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", &adj[i][j]);

    warshall(adj, n);
```

```
    printf("Transitive closure of the given graph is:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%d ", adj[i][j]);
        printf("\n");
    }

    return 0;
}
```

4TH PROGRAM

```
#include<stdio.h>
int cost[10][10], n, dist[10];

int minm(int m, int n) {
   return ((m < n) ? m : n);
}

void dijkstra(int source) {
   int s[10] = {0};
   int min, w = 0;

   for (int i = 0; i < n; i++) {
      dist[i] = cost[source][i];
   }

   // Initialize dist from source to source as 0
   dist[source] = 0;

   // Mark source vertex - estimated for its shortest path
   s[source] = 1;

   for (int i = 0; i < n - 1; i++) {
      // Find the nearest neighbour vertex
      min = 999;
      for (int j = 0; j < n; j++) {
         if ((s[j] == 0) && (min > dist[j])) {
            min = dist[j];
            w = j;
         }
      }
```

```
      s[w] = 1;

      // Update the shortest path of neighbour of w
      for (int v = 0; v < n; v++) {
         if (s[v] == 0 && cost[w][v] != 999) {
            dist[v] = minm(dist[v], dist[w] + cost[w][v]);
         }
      }
   }
}

int main() {
   int source;

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

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

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

   dijkstra(source);

   printf("The shortest distances are:\n");
   for (int i = 0; i < n; i++) {
```

```
        printf("Cost from %d to %d is %d\n", source, i, dist[i]);
    }

    return 0;
}
```

5th program

```
#include <stdio.h>

int cost[10][10], n, colsum[10];

void cal_colsum() {
    for (int j = 0; j < n; j++) {
        colsum[j] = 0;
        for (int i = 0; i < n; i++) {
            colsum[j] += cost[i][j];
        }
    }
}

void source_removal() {
    int select[10] = {0};
    printf("Topological ordering is: ");

    for (int i = 0; i < n; i++) {
        // Calculate the outdegree for each vertex
        cal_colsum();

        int j;
        for (j = 0; j < n; j++) {
            if (select[j] == 0 && colsum[j] == 0) { // Source vertex
```

```
            break;
        }
    }

    if (j == n) { // No source vertex found, this implies a cycle
        printf("\nGraph has a cycle, topological sorting not possible.\n");
        return;
    }

    printf("%d ", j);
    select[j] = 1;

    // Remove source vertex j from cost matrix
    for (int k = 0; k < n; k++) {
        cost[j][k] = 0;
    }
  }
  printf("\n");
}

int main() {
  printf("Enter no. of vertices: ");
  scanf("%d", &n);

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

```
    source_removal();

    return 0;
}
```

6th program

```
#include <stdio.h>

int n, m, p[10], w[10];

int max(int a, int b) {
    return (a > b ? a : b);
}

void knapsack_DP() {
    int V[10][10], i, j;

    // Initialize the DP table
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            if (i == 0 || j == 0) {
                V[i][j] = 0;
            } else if (j < w[i]) {  // Weight of the item is larger than capacity
                V[i][j] = V[i - 1][j];
            } else {
                V[i][j] = max(V[i - 1][j], p[i] + V[i - 1][j - w[i]]);
            }
        }
    }

    // Print the DP table
```

```
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            printf("%d ", V[i][j]);
        }
        printf("\n");
    }

    // Tracking back the optimal solution vector
    printf("Items included are: ");
    int currentCapacity = m;
    for (i = n; i > 0 && currentCapacity > 0; i--) {
        if (V[i][currentCapacity] != V[i - 1][currentCapacity]) {
            printf("%d ", i);
            currentCapacity -= w[i];
        }
    }
    printf("\n");
}

int main() {
    int i;

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

    printf("Enter the weights of the items: ");
    for (i = 1; i <= n; i++) {
        scanf("%d", &w[i]);
    }

    printf("Enter the prices of the items: ");
```

```
    for (i = 1; i <= n; i++) {
        scanf("%d", &p[i]);
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &m);

    knapsack_DP();

    return 0;
}
```

7th program

```
#include<stdio.h>

int n, m, p[10], w[10];

void greedy_knapsack() {
    float max, profit = 0;
    int k = 0, i, j;

    printf("Items included: ");
    for (i = 0; i < n; i++) {
        max = 0;
        // Choose the item which has the highest price to weight ratio
        for (j = 0; j < n; j++) {
            if (((float)p[j]) / w[j] > max) {
                k = j;
                max = ((float)p[j]) / w[j];
            }
```

```
        }

        // kth element has the highest price to weight ratio
        if (w[k] <= m) {
            printf("%d ", k);
            m = m - w[k];
            profit = profit + p[k];
            p[k] = 0;
        } else {
            break; // Unable to fit item k into knapsack
        }
    }

    printf("\nDiscrete Knapsack profit = %f\n", profit);
    if (k < n && w[k] > 0) {
        printf("Continuous Knapsack also includes item %d with portion: %f\n", k, ((float)m / w[k]));
        profit = profit + ((float)m / w[k]) * p[k];
    }
    printf("Continuous Knapsack profit = %f\n", profit);
}

int main() {
    int i;

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

    printf("Enter the weights of n items: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);
```

```
    printf("Enter the prices of n items: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);

    printf("Enter the capacity of Knapsack: ");
    scanf("%d", &m);

    greedy_knapsack();

    return 0;
}
```