UNIVERSITY COLLEGE OF ENGINEERING
(AUTONOMOUS)
OSMANIA UNIVERSITY, HYDERABAD, TELANGANA STATE, INDIA
A University Accredited by NAAC with A+ Grade
Category -I Graded Autonomy by UGC
ESTD. 1917
ESTD. 1929

Design & Analysis of Algorithms Record

Name	:	
Roll No.	:	**1005227330**
Class	:	**BE(CSE)**
Semester	:	**IInd Year, IV Sem**
Academic Year	:	**2023-24**
Department	:	**Computer Science**

CERTIFICATE

This is to certify that the programming assignments and projects submitted by ____________________ with Roll No. 1005227330___, pursuing B.E. in Computer Science and Engineering, during IInd Year, IV Sem for the subject "Design & Analysis of Algorithms", are genuine and represent their individual work.

Submitted on ____________________

Faculty Member

INDEX

S. No.	Name of the Program	Page No.
1	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.	1-2
2	Write a program to find the shortest path in graph using Dijkstra’s algorithm.	3-4
3	Write a program that implements N Queen's problem using backtracking algorithm.	5-6
4	Write a program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.	7-8
5	Write a program to implement dynamic programming algorithm to solve all pairs shortest path problem.	9-10
6	Write a program to solve fractional knapsack problem using Greedy algorithm.	11-12
7	Write a program to solve 0/1 knapsack problem using Dynamic programming algorithm.	13-14
8	Write a program to solve 0/1 knapsack problem using Backtracking algorithm.	15-16
9	Write a program to solve 0/1 knapsack problem using Branch and bound algorithm.	17-18
10	Write a program that uses dynamic programming algorithm to solve the optimal binary search tree.	19-20
11	Write a program for solving traveling sales persons problem using Dynamic programming algorithm.	21-22
12	Write a program for solving traveling sales persons problem using the back tracking algorithm.	23-24
13	Write a program for solving traveling sales persons problem using Branch and Bound.	25-27
14	Write a program to obtain the Topological ordering of vertices in a given digraph using DFS.	28-30
15	Write a program to obtain the Topological ordering of vertices in a given digraph using Kahn’s algorithm.	31-33
16	Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.	34-35
17	Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method.	36-37
18	Write a program to check whether a given graph is connected or not using DFS method.	38-39
19	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	40-42

1) Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define MAX 100

int minKey(int key[], bool mstSet[], int V) {
  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++)
    if (mstSet[v] == false && key[v] < min)
      min = key[v], min_index = v;

  return min_index;
}

void printMST(int parent[], int graph[MAX][MAX], int V) {
  int totalCost = 0;
  printf("Edge \tWeight\n");
  for (int i = 1; i < V; i++) {
    printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
    totalCost += graph[i][parent[i]];
  }
  printf("Total cost of MST: %d\n", totalCost);
}

void primMST(int graph[MAX][MAX], int V) {
  int parent[MAX];
  int key[MAX];
  bool mstSet[MAX];

  for (int i = 0; i < V; i++)
    key[i] = INT_MAX, mstSet[i] = false;

  key[0] = 0;
  parent[0] = -1;

  for (int count = 0; count < V - 1; count++) {
    int u = minKey(key, mstSet, V);

    mstSet[u] = true;

    for (int v = 0; v < V; v++)
      if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
```

```
            parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph, V);
}

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

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

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

    primMST(graph, V);

    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix of the graph:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Edge    Weight
0 - 1     2
1 - 2     3
0 - 3     6
1 - 4     5
Total cost of MST: 16
```

2) Write a program to find the shortest path in graph using Dijkstra's algorithm.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define MAX 100

int minDistance(int dist[], bool sptSet[], int V) {
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;

   return min_index;
}

void printSolution(int dist[], int V) {
   printf("Vertex \t Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[MAX][MAX], int V, int src) {
   int dist[MAX];
   bool sptSet[MAX];

   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;

   dist[src] = 0;

   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(dist, sptSet, V);

      sptSet[u] = true;

      for (int v = 0; v < V; v++)
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
            && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
   }

   printSolution(dist, V);
}
```

```
int main() {
   int V, src;
   int graph[MAX][MAX];

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

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

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

   dijkstra(graph, V, src);

   return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix of the graph:
0 10 0 0 5
0 0 1 0 2
0 0 0 4 0
7 0 6 0 0
0 3 9 2 0
Enter the source vertex: 0
Vertex   Distance from Source
0              0
1              8
2              9
3              7
4              5
```

3) Write a program that implements N Queen's problem using backtracking algorithm.

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

#define MAX 20

void printSolution(int board[MAX][MAX], int N) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      if (board[i][j])
        printf(" Q ");
      else
        printf(" . ");
    }
    printf("\n");
  }
}

bool isSafe(int board[MAX][MAX], int row, int col, int N) {
  for (int i = 0; i < col; i++)
    if (board[row][i])
      return false;

  for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
    if (board[i][j])
      return false;

  for (int i = row, j = col; j >= 0 && i < N; i++, j--)
    if (board[i][j])
      return false;

  return true;
}

bool solveNQUtil(int board[MAX][MAX], int col, int N) {
  if (col >= N)
    return true;

  for (int i = 0; i < N; i++) {
    if (isSafe(board, i, col, N)) {
      board[i][col] = 1;

      if (solveNQUtil(board, col + 1, N))
        return true;

      board[i][col] = 0;
    }
```

```
    }
    return false;
}

bool solveNQ(int N) {
    int board[MAX][MAX] = {0};

    if (!solveNQUtil(board, 0, N)) {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board, N);
    return true;
}

int main() {
    int N;
    printf("Enter the number of queens: ");
    scanf("%d", &N);
    solveNQ(N);
    return 0;
}
```

Output:

```
Enter the number of queens: 4
 . Q . .
 . . . Q
 Q . . .
 . . Q .
```

4) Write a program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

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

#define MAX 20

void printSolution(int path[], int n) {
  for (int i = 0; i < n; i++)
    printf(" %d ", path[i]);
  printf(" %d\n", path[0]);
}

bool isSafe(int v, int graph[MAX][MAX], int path[], int pos) {
  if (graph[path[pos - 1]][v] == 0)
    return false;

  for (int i = 0; i < pos; i++)
    if (path[i] == v)
      return false;

  return true;
}

void hamCycleUtil(int graph[MAX][MAX], int path[], int pos, int n) {
  if (pos == n) {
    if (graph[path[pos - 1]][path[0]] == 1) {
      printSolution(path, n);
    }
    return;
  }

  for (int v = 1; v < n; v++) {
    if (isSafe(v, graph, path, pos)) {
      path[pos] = v;
      hamCycleUtil(graph, path, pos + 1, n);
      path[pos] = -1;
    }
  }
}

void hamCycle(int graph[MAX][MAX], int n) {
  int path[MAX];
  for (int i = 0; i < n; i++)
    path[i] = -1;

  path[0] = 0;
```

```
    hamCycleUtil(graph, path, 1, n);
}

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

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

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

    printf("Hamiltonian Cycles:\n");
    hamCycle(graph, n);
    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the adjacency matrix of the graph:
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 1 1 0
Hamiltonian Cycles:
 0  1  2  4  3  0
 0  3  4  2  1  0
```

5) Write a program to implement dynamic programming algorithm to solve all pairs shortest path problem.

```
#include <stdio.h>

#define MAX 20
#define INF 99999

void printSolution(int dist[MAX][MAX], int n) {
    printf("Shortest distances between every pair of vertices:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

void floydWarshall(int graph[MAX][MAX], int n) {
    int dist[MAX][MAX];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            dist[i][j] = graph[i][j];

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    printSolution(dist, n);
}

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

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

    printf("Enter the adjacency matrix of the graph (use %d to represent infinity):\n", INF);
```

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

    floydWarshall(graph, n);
    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the adjacency matrix of the graph (use 99999 to represent infinity):
0 3 99999 7
8 0 2 99999
5 99999 0 1
2 99999 99999 0
Shortest distances between every pair of vertices:
   0    3    5    7
   8    0    2    3
   5    8    0    1
   2    5    7    0
```

6) Write a program to solve fractional knapsack problem using Greedy algorithm.

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

struct Item {
    int value, weight;
    float ratio;
};

int compare(const void *a, const void *b) {
    struct Item *item1 = (struct Item *)a;
    struct Item *item2 = (struct Item *)b;
    if (item1->ratio < item2->ratio) return 1;
    if (item1->ratio > item2->ratio) return -1;
    return 0;
}

void fractionalKnapsack(int W, struct Item items[], int n) {

    qsort(items, n, sizeof(struct Item), compare);

    int curWeight = 0;
    float finalValue = 0.0;
    printf("Included items:\n");
    printf("Value  Weight  Fraction\n");

    for (int i = 0; i < n; i++) {
        if (curWeight + items[i].weight <= W) {
            curWeight += items[i].weight;
            finalValue += items[i].value;
            printf("%5d  %6d  1.0000\n", items[i].value, items[i].weight);
        } else {
            int remaining = W - curWeight;
            finalValue += items[i].value * ((float)remaining / items[i].weight);
            printf("%5d  %6d  %.4f\n", items[i].value, items[i].weight, (float)remaining / items[i].weight);
            break;
        }
    }

    printf("\nMaximum value in Knapsack = %.2f\n", finalValue);
}

int main() {
    int n, W;

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

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

    struct Item *items = (struct Item *)malloc(n * sizeof(struct Item));

    for (int i = 0; i < n; i++) {
        printf("Enter value and weight for item %d: ", i + 1);
        scanf("%d %d", &items[i].value, &items[i].weight);
        items[i].ratio = (float)items[i].value / items[i].weight;
    }

    fractionalKnapsack(W, items, n);

    free(items);

    return 0;
}
```

Output:

```
Enter the number of items: 4
Enter the capacity of knapsack: 50
Enter value and weight for item 1: 60 10
Enter value and weight for item 2: 100 20
Enter value and weight for item 3: 120 30
Enter value and weight for item 4: 200 40
Included items:
Value  Weight  Fraction
   60      10  1.0000
  100      20  1.0000
  200      40  0.5000

Maximum value in Knapsack = 260.00
```

7) Write a program to solve 0/1 knapsack problem using Dynamic programming algorithm.

```
#include <stdio.h>

#define MAX 100

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

void knapsack(int W, int weights[], int values[], int n) {
    int dp[MAX][MAX];

    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    int maxValue = dp[n][W];
    printf("Maximum value that can be put in knapsack = %d\n", maxValue);

    int w = W;
    printf("Items included in the knapsack:\n");
    for (int i = n; i > 0 && maxValue > 0; i--) {
        if (maxValue == dp[i - 1][w])
            continue;
        else {
            printf("Item %d (Value = %d, Weight = %d)\n", i, values[i - 1], weights[i - 1]);
            maxValue -= values[i - 1];
            w -= weights[i - 1];
        }
    }
}

int main() {
    int n, W;
    int weights[MAX], values[MAX];

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

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

   printf("Enter the value and weight of each item:\n");
   for (int i = 0; i < n; i++) {
      printf("Item %d:\n", i + 1);
      printf("Value = ");
      scanf("%d", &values[i]);
      printf("Weight = ");
      scanf("%d", &weights[i]);
   }

   knapsack(W, weights, values, n);

   return 0;
}
```

Output:

```
Enter the number of items: 4
Enter the capacity of knapsack: 50
Enter value and weight for item 1: 60 10
Enter value and weight for item 2: 100 20
Enter value and weight for item 3: 120 30
Enter value and weight for item 4: 200 40
Included items:
Value  Weight  Fraction
  60      10  1.0000
 100      20  1.0000
 200      40  0.5000

Maximum value in Knapsack = 260.00
```

8) Write a program to solve 0/1 knapsack problem using Backtracking algorithm.

```
#include <stdio.h>

#define MAX_ITEMS 100

int n, W;
int weights[MAX_ITEMS], values[MAX_ITEMS];
int max_value = 0;
int best_set[MAX_ITEMS];
int current_set[MAX_ITEMS];

void knapsack(int i, int current_weight, int current_value) {
  if (i == n) {
    if (current_value > max_value) {
      max_value = current_value;
      for (int j = 0; j < n; j++) {
        best_set[j] = current_set[j];
      }
    }
    return;
  }

  current_set[i] = 0;
  knapsack(i + 1, current_weight, current_value);

  if (current_weight + weights[i] <= W) {
    current_set[i] = 1;
    knapsack(i + 1, current_weight + weights[i], current_value + values[i]);
  }
}

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

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

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

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

```
    }

    knapsack(0, 0, 0);

    printf("The maximum value is: %d\n", max_value);
    printf("The included items are:\n");
    for (int i = 0; i < n; i++) {
        if (best_set[i]) {
            printf("Item %d (Weight: %d, Value: %d)\n", i + 1, weights[i], values[i]);
        }
    }

    return 0;
}
```

Output:

```
Enter the number of items: 4
Enter the maximum capacity of the knapsack: 5
Enter the weights of the items:
2 3 4 5
Enter the values of the items:
3 4 5 6
The maximum value is: 7
The included items are:
Item 1 (Weight: 2, Value: 3)
Item 2 (Weight: 3, Value: 4)
```

9) Write a program to solve 0/1 knapsack problem using Branch and bound algorithm.

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

#define MAX_ITEMS 10

struct Item {
   int weight;
   int profit;
   float ratio;
};

int numItems, capacity;
struct Item items[MAX_ITEMS];
int maxProfit = 0;
int finalSet[MAX_ITEMS];

void knapsack(int currentWeight, int currentProfit, int level, int include[]) {
   if (level == numItems) {
      if (currentWeight <= capacity && currentProfit > maxProfit) {
         maxProfit = currentProfit;
         for (int i = 0; i < numItems; i++) {
            finalSet[i] = include[i];
         }
      }
      return;
   }

   float upperBound = currentProfit;
   int remainingWeight = capacity - currentWeight;
   int i = level;
   while (i < numItems && remainingWeight >= items[i].weight) {
      remainingWeight -= items[i].weight;
      upperBound += items[i].profit;
      i++;
   }
   if (i < numItems) {
      upperBound += (float)remainingWeight / items[i].weight * items[i].profit;
   }

   if (upperBound <= maxProfit) {
      return;
   }

   include[level] = 1;
```

```
   knapsack(currentWeight + items[level].weight, currentProfit + items[level].profit, level + 1,
include);

   include[level] = 0;
   knapsack(currentWeight, currentProfit, level + 1, include);
}

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

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

   printf("Enter weight and profit of each item:\n");
   for (int i = 0; i < numItems; i++) {
      printf("Item %d: ", i + 1);
      scanf("%d %d", &items[i].weight, &items[i].profit);
      items[i].ratio = (float)items[i].profit / items[i].weight;
   }

   int include[MAX_ITEMS] = {0};
   knapsack(0, 0, 0, include);

   printf("Maximum profit: %d\n", maxProfit);
   printf("Selected items (1 for selected, 0 for not selected):\n");
   for (int i = 0; i < numItems; i++) {
      printf("%d ", finalSet[i]);
   }
   printf("\n");

   return 0;
}
```

Output:

```
Enter the number of items: 3
Enter the capacity of knapsack: 50
Enter weight and profit of each item:
Item 1: 10 60
Item 2: 20 100
Item 3: 30 120
Maximum profit: 220
Selected items (1 for selected, 0 for not selected):
0 1 1
```

10) Write a program that uses dynamic programming algorithm to solve the optimal binary search tree.

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

int sum(int freq[], int i, int j);

int optCost(int freq[], int i, int j) {
   if (j < i)
      return 0;
   if (j == i)
      return freq[i];

   int fsum = sum(freq, i, j);
   int min = INT_MAX;

   for (int r = i; r <= j; ++r) {
      int cost = optCost(freq, i, r-1) + optCost(freq, r+1, j);
      if (cost < min)
         min = cost;
   }

   return min + fsum;
}

int optimalSearchTree(int keys[], int freq[], int n) {
   return optCost(freq, 0, n-1);
}

int sum(int freq[], int i, int j) {
   int s = 0;
   for (int k = i; k <= j; k++)
     s += freq[k];
   return s;
}

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

   int keys[n], freq[n];

   printf("Enter keys:\n");
   for (int i = 0; i < n; i++)
```

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

    printf("Enter frequencies:\n");
    for (int i = 0; i < n; i++)
        scanf("%d", &freq[i]);

    printf("Cost of Optimal BST is %d\n", optimalSearchTree(keys, freq, n));
    return 0;
}
```

Output:

```
Enter the number of keys: 3
Enter keys:
10
12
20
Enter frequencies:
34
8
50
Cost of Optimal BST is 142
```

11) Write a program for solving traveling sales persons problem using Dynamic programming algorithm.

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

#define MAX_CITIES 10

int n;
int dist[MAX_CITIES][MAX_CITIES];
int dp[MAX_CITIES][1 << MAX_CITIES];
int path[MAX_CITIES][1 << MAX_CITIES];

int tsp(int mask, int pos) {

  if (mask == (1 << n) - 1) {
    return dist[pos][0];
  }

  if (dp[pos][mask] != -1) {
    return dp[pos][mask];
  }

  int ans = INT_MAX;

  for (int city = 0; city < n; city++) {
    if (!(mask & (1 << city))) {
      int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
      if (newAns < ans) {
        ans = newAns;
        path[pos][mask] = city;
      }
    }
  }

  return dp[pos][mask] = ans;
}

void printPath() {
  int curr = 0, mask = 1;
  printf("Path: 1 -> ");
  for (int i = 0; i < n - 1; i++) {
    int next = path[curr][mask];
    printf("%d -> ", next + 1);
    mask |= (1 << next);
    curr = next;
  }
```

```
    printf("1\n");
}

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

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


    for (int i = 0; i < n; i++) {
        for (int j = 0; j < (1 << n); j++) {
            dp[i][j] = -1;
            path[i][j] = -1;
        }
    }

    int ans = tsp(1, 0);

    printf("Minimum cost of visiting all cities: %d\n", ans);
    printPath();

    return 0;
}
```

Output:

```
Enter the number of cities: 4
Enter the distance matrix (4 x 4):
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Minimum cost of visiting all cities: 80
Path: 1 -> 2 -> 4 -> 3 -> 1
```

12) Write a program for solving traveling sales persons problem using the back tracking algorithm.

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

#define MAX_CITIES 10

int n;
int dist[MAX_CITIES][MAX_CITIES];
int visited[MAX_CITIES];
int bestPath[MAX_CITIES];
int minCost = INT_MAX;

void tsp(int currentCity, int visitedCount, int cost, int path[]) {

  if (visitedCount == n) {

    cost += dist[currentCity][0];

    if (cost < minCost) {
      minCost = cost;
      for (int i = 0; i < n; i++) {
        bestPath[i] = path[i];
      }
    }
    return;
  }

  for (int nextCity = 0; nextCity < n; nextCity++) {
    if (!visited[nextCity]) {

      visited[nextCity] = 1;

      path[visitedCount] = nextCity;
      int newCost = cost + dist[currentCity][nextCity];

      if (newCost < minCost) {
        tsp(nextCity, visitedCount + 1, newCost, path);
      }

      visited[nextCity] = 0;
    }
  }
}

void printPath() {
  printf("Path: ");
```

```
    for (int i = 0; i < n; i++) {
        printf("%d -> ", bestPath[i] + 1);
    }
    printf("1\n");
}

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

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

    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }

    visited[0] = 1;
    int path[MAX_CITIES];
    path[0] = 0;

    tsp(0, 1, 0, path);

    printf("Minimum cost of visiting all cities: %d\n", minCost);
    printPath();

    return 0;
}
```

Output:

```
Enter the number of cities: 4
Enter the distance matrix (4 x 4):
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Minimum cost of visiting all cities: 80
Path: 1 -> 2 -> 4 -> 3 -> 1
```

13) Write a program for solving traveling sales persons problem using Branch and Bound.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define MAXCITIES 10

int numCities;
int graph[MAXCITIES][MAXCITIES];
int visited[MAXCITIES];
int minCost = INT_MAX;
int finalPath[MAXCITIES + 1];

void copyToFinal(int currPath[], int finalPath[]) {
  for (int i = 0; i < numCities; i++) {
    finalPath[i] = currPath[i];
  }
  finalPath[numCities] = currPath[0];
}

int firstMin(int i) {
  int min = INT_MAX;
  for (int k = 0; k < numCities; k++) {
    if (graph[i][k] < min && i != k) {
      min = graph[i][k];
    }
  }
  return min;
}

int secondMin(int i) {
  int first = INT_MAX, second = INT_MAX;
  for (int j = 0; j < numCities; j++) {
    if (i == j) continue;
    if (graph[i][j] <= first) {
      second = first;
      first = graph[i][j];
    } else if (graph[i][j] <= second && graph[i][j] != first) {
      second = graph[i][j];
    }
  }
  return second;
}

void tsp(int currBound, int currWeight, int level, int currPath[]) {
  if (level == numCities) {
```

```
        if (graph[currPath[level - 1]][currPath[0]] > 0) {
            int currCost = currWeight + graph[currPath[level - 1]][currPath[0]];
            if (currCost < minCost) {
                copyToFinal(currPath, finalPath);
                minCost = currCost;
            }
        }
        return;
    }

    for (int i = 0; i < numCities; i++) {
        if (graph[currPath[level - 1]][i] > 0 && visited[i] == 0) {
            int temp = currBound;
            currWeight += graph[currPath[level - 1]][i];

            if (level == 1) {
                currBound -= ((firstMin(currPath[level - 1]) + firstMin(i)) / 2);
            } else {
                currBound -= ((secondMin(currPath[level - 1]) + firstMin(i)) / 2);
            }

            if (currBound + currWeight < minCost) {
                currPath[level] = i;
                visited[i] = 1;
                tsp(currBound, currWeight, level + 1, currPath);
            }

            currWeight -= graph[currPath[level - 1]][i];
            currBound = temp;

            for (int j = 0; j < numCities; j++) {
                visited[j] = 0;
            }
            for (int j = 0; j <= level - 1; j++) {
                visited[currPath[j]] = 1;
            }
        }
    }
}

void printPath(int path[]) {
    printf("Path taken: ");
    for (int i = 0; i <= numCities; i++) {
        printf("%d ", path[i] + 1);
    }
    printf("\n");
}
```

```
int main() {
    printf("Enter the number of cities: ");
    scanf("%d", &numCities);

    printf("Enter the cost matrix (enter 0 for same city and INT_MAX for unreachable):\n");
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j < numCities; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    int currPath[MAXCITIES + 1];
    int currBound = 0;
    for (int i = 0; i < numCities; i++) {
        currPath[i] = -1;
        visited[i] = 0;
        currBound += (firstMin(i) + secondMin(i));
    }

    currBound = (currBound % 2 == 0) ? currBound / 2 + 1 : currBound / 2;
    visited[0] = 1;
    currPath[0] = 0;
    tsp(currBound, 0, 1, currPath);

    printf("Minimum cost of traversal is: %d\n", minCost);
    printPath(finalPath);

    return 0;
}
```

<u>Output:</u>

```
Enter the number of cities: 4
Enter the cost matrix (enter 0 for same city and INT_MAX for unreachable):
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0

Minimum cost of traversal is: 80
Path taken: 1 2 4 3 1
```

14) Write a program to obtain the Topological ordering of vertices in a given digraph using DFS.

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

#define MAX_VERTICES 100

struct Node {
  int vertex;
  struct Node* next;
};

struct Graph {
  int numVertices;
  struct Node** adjLists;
  int* visited;
};

struct Node* createNode(int v) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

struct Graph* createGraph(int vertices) {
  struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
  graph->visited = (int*)malloc(vertices * sizeof(int));

  for (int i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct Node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;
}

void topologicalSortDFS(struct Graph* graph, int v, int visited[], struct Node** stack) {
  visited[v] = 1;
```

```
    struct Node* adjList = graph->adjLists[v];
    while (adjList != NULL) {
        int connectedVertex = adjList->vertex;
        if (!visited[connectedVertex]) {
            topologicalSortDFS(graph, connectedVertex, visited, stack);
        }
        adjList = adjList->next;
    }

    struct Node* newNode = createNode(v);
    newNode->next = *stack;
    *stack = newNode;
}

void printStack(struct Node* stack) {
    while (stack != NULL) {
        printf("%d ", stack->vertex);
        stack = stack->next;
    }
    printf("\n");
}

void topologicalSortUsingDFS(struct Graph* graph) {
    struct Node* stack = NULL;
    int* visited = (int*)malloc(graph->numVertices * sizeof(int));

    for (int i = 0; i < graph->numVertices; i++) {
        visited[i] = 0;
    }

    for (int i = 0; i < graph->numVertices; i++) {
        if (visited[i] == 0) {
            topologicalSortDFS(graph, i, visited, &stack);
        }
    }

    printf("Topological Sorting using DFS: ");
    printStack(stack);
}

int main() {
    int vertices, edges, src, dest;

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

    struct Graph* graph = createGraph(vertices);
```

```
    printf("Enter the number of edges: ");
    scanf("%d", &edges);

    for (int i = 0; i < edges; i++) {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    topologicalSortUsingDFS(graph);

    return 0;
}
```

Output:

```
Enter the number of vertices: 6
Enter the number of edges: 6
Enter edge 1 (source destination): 5 2
Enter edge 2 (source destination): 5 0
Enter edge 3 (source destination): 4 0
Enter edge 4 (source destination): 4 1
Enter edge 5 (source destination): 2 3
Enter edge 6 (source destination): 3 1
Topological Sorting using DFS: 5 4 2 3 1 0
```

15) Write a program to obtain the Topological ordering of vertices in a given digraph using Kahn's algorithm.

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

#define MAX_VERTICES 100

struct Node {
  int vertex;
  struct Node* next;
};

struct Graph {
  int numVertices;
  int* inDegree;
  struct Node** adjLists;
};

struct Node* createNode(int v) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

struct Graph* createGraph(int vertices) {
  struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = (struct Node**)malloc(vertices * sizeof(struct Node*));
  graph->inDegree = (int*)malloc(vertices * sizeof(int));

  for (int i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->inDegree[i] = 0;
  }
  return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
  struct Node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  graph->inDegree[dest]++;
}
```

```c
void topologicalSortUsingKahn(struct Graph* graph) {
    int vertices = graph->numVertices;
    int* inDegree = graph->inDegree;
    struct Node** adjLists = graph->adjLists;

    int* topoOrder = (int*)malloc(vertices * sizeof(int));
    int topoIndex = 0;

    int* queue = (int*)malloc(vertices * sizeof(int));
    int front = 0, rear = 0;

    for (int i = 0; i < vertices; i++) {
        if (inDegree[i] == 0) {
            queue[rear++] = i;
        }
    }

    while (front != rear) {
        int u = queue[front++];
        topoOrder[topoIndex++] = u;

        struct Node* temp = adjLists[u];
        while (temp != NULL) {
            int v = temp->vertex;
            if (--inDegree[v] == 0) {
                queue[rear++] = v;
            }
            temp = temp->next;
        }
    }

    if (topoIndex != vertices) {
        printf("Topological sorting not possible (graph has cycle)\n");
    } else {
        printf("Topological Sorting using Kahn's Algorithm: ");
        for (int i = 0; i < vertices; i++) {
            printf("%d ", topoOrder[i]);
        }
        printf("\n");
    }

    free(queue);
    free(topoOrder);
}

int main() {
    int vertices, edges, src, dest;
```

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

    struct Graph* graph = createGraph(vertices);

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    topologicalSortUsingKahn(graph);

    return 0;
}
```

Output:

```
Enter the number of vertices: 6
Enter the number of edges: 6
Enter edge 1 (source destination): 5 2
Enter edge 2 (source destination): 5 0
Enter edge 3 (source destination): 4 0
Enter edge 4 (source destination): 4 1
Enter edge 5 (source destination): 2 3
Enter edge 6 (source destination): 3 1
Topological Sorting using Kahn's Algorithm: 4 5 0 2 3 1
```

16) Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.

```
#include <stdio.h>

#define MAX_VERTICES 100

void printMatrix(int n, int matrix[MAX_VERTICES][MAX_VERTICES]) {
   printf("Transitive Closure Matrix:\n");
   printf("   ");
   for (int i = 0; i < n; i++) {
      printf("%d ", i);
   }
   printf("\n");

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

void transitiveClosure(int n, int graph[MAX_VERTICES][MAX_VERTICES]) {
   int closure[MAX_VERTICES][MAX_VERTICES];

   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         closure[i][j] = graph[i][j];
      }
   }

   for (int k = 0; k < n; k++) {
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (closure[i][j] == 0 && closure[i][k] && closure[k][j]) {
               closure[i][j] = 1;
            }
         }
      }
   }

   printMatrix(n, closure);
}

int main() {
   int n, edges, src, dest;
```

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

    int graph[MAX_VERTICES][MAX_VERTICES];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            graph[i][j] = 0;
        }
    }

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        graph[src][dest] = 1;
    }

    transitiveClosure(n, graph);

    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edge 1 (source destination): 0 1
Enter edge 2 (source destination): 1 2
Enter edge 3 (source destination): 1 3
Enter edge 4 (source destination): 2 0
Transitive Closure Matrix:
   0 1 2 3
0| 1 1 1 1
1| 1 1 1 1
2| 1 1 1 1
3| 0 0 0 0
```

17) Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method.

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

#define MAX 100

int adj[MAX][MAX], visited[MAX], n;

void bfs(int start) {
   int queue[MAX], front = -1, rear = -1;
   int i;

   printf("Nodes reachable from node %d: ", start);
   queue[++rear] = start;
   visited[start] = 1;

   while (front != rear) {
      start = queue[++front];
      printf("%d ", start);

      for (i = 0; i < n; i++) {
         if (adj[start][i] == 1 && visited[i] == 0) {
            queue[++rear] = i;
            visited[i] = 1;
         }
      }
   }
   printf("\n");
}

int main() {
   int i, j, start;

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

   printf("Enter adjacency matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &adj[i][j]);

   printf("Enter starting node: ");
   scanf("%d", &start);

   for (i = 0; i < n; i++)
      visited[i] = 0;
```

```
    bfs(start);

    return 0;
}
```

Output:

```
Enter number of nodes: 4
Enter adjacency matrix:
0 1 1 0
0 0 1 1
0 0 0 1
0 0 0 0
Enter starting node: 0
Nodes reachable from node 0: 0 1 2 3
```

18) Write a program to check whether a given graph is connected or not using DFS method.

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

#define MAX 100

int adj[MAX][MAX], visited[MAX], n;

void dfs(int v) {
  visited[v] = 1;
  for (int i = 0; i < n; i++) {
    if (adj[v][i] == 1 && visited[i] == 0) {
      dfs(i);
    }
  }
}

int isConnected() {
  for (int i = 0; i < n; i++) {
    visited[i] = 0;
  }

  dfs(0);

  for (int i = 0; i < n; i++) {
    if (visited[i] == 0) {
      return 0;
    }
  }
  return 1;
}

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

  printf("Enter adjacency matrix (0/1):\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      scanf("%d", &adj[i][j]);
    }
  }

  if (n <= 0) {
    printf("Invalid number of nodes.\n");
    return 1;
  }
```

```c
    if (isConnected()) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

Output:

```
Enter number of nodes: 4
Enter adjacency matrix (0/1):
0 1 1 0
0 0 1 1
1 0 0 1
0 0 0 0
The graph is connected.
```

19) Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

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

struct Edge {
  int src, dest, weight;
};

struct Graph {
  int V, E;
  struct Edge* edge;
};

struct Subset {
  int parent;
  int rank;
};

struct Graph* createGraph(int V, int E) {
  struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
  graph->V = V;
  graph->E = E;
  graph->edge = (struct Edge*) malloc(E * sizeof(struct Edge));
  return graph;
}

int find(struct Subset subsets[], int i) {
  if (subsets[i].parent != i)
    subsets[i].parent = find(subsets, subsets[i].parent);
  return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
  int rootX = find(subsets, x);
  int rootY = find(subsets, y);

  if (subsets[rootX].rank < subsets[rootY].rank)
    subsets[rootX].parent = rootY;
  else if (subsets[rootX].rank > subsets[rootY].rank)
    subsets[rootY].parent = rootX;
  else {
    subsets[rootY].parent = rootX;
    subsets[rootX].rank++;
  }
}
```

```
int compareEdges(const void* a, const void* b) {
  struct Edge* edgeA = (struct Edge*) a;
  struct Edge* edgeB = (struct Edge*) b;
  return edgeA->weight - edgeB->weight;
}

void KruskalMST(struct Graph* graph) {
  int V = graph->V;
  struct Edge resultMST[V];
  int e = 0;
  int i = 0;

  qsort(graph->edge, graph->E, sizeof(graph->edge[0]), compareEdges);

  struct Subset* subsets = (struct Subset*) malloc(V * sizeof(struct Subset));

  for (int v = 0; v < V; v++) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
  }

  while (e < V - 1 && i < graph->E) {
    struct Edge nextEdge = graph->edge[i++];

    int x = find(subsets, nextEdge.src);
    int y = find(subsets, nextEdge.dest);

    if (x != y) {
      resultMST[e++] = nextEdge;
      Union(subsets, x, y);
    }
  }

  printf("Edges in Minimum Cost Spanning Tree:\n");
  int totalWeight = 0;
  for (i = 0; i < e; ++i) {
    printf("%d -- %d  Weight: %d\n", resultMST[i].src, resultMST[i].dest, resultMST[i].weight);
    totalWeight += resultMST[i].weight;
  }
  printf("Total Weight of MST: %d\n", totalWeight);

  free(subsets);
}

int main() {
  int V, E;
  int src, dest, weight;
```

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

    printf("Enter the number of edges: ");
    scanf("%d", &E);

    struct Graph* graph = createGraph(V, E);

    for (int i = 0; i < E; ++i) {
        printf("Enter edge %d (source destination weight): ", i + 1);
        scanf("%d %d %d", &src, &dest, &weight);
        graph->edge[i].src = src;
        graph->edge[i].dest = dest;
        graph->edge[i].weight = weight;
    }

    KruskalMST(graph);

    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter edge 1 (source destination weight): 0 1 10
Enter edge 2 (source destination weight): 0 2 6
Enter edge 3 (source destination weight): 0 3 5
Enter edge 4 (source destination weight): 1 3 15
Enter edge 5 (source destination weight): 2 3 4
Edges in Minimum Cost Spanning Tree:
2 -- 3  Weight: 4
0 -- 3  Weight: 5
0 -- 1  Weight: 10
Total Weight of MST: 19
```