DAA PROGRAMS

# 1) PRIMS ALGO:

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

#define V 100 // Maximum number of vertices (you can change this)

int i,j,v;

int minKey(int key[], int mstSet[], int n) {
   int min = INT_MAX, minIndex;

   for ( v = 0; v < n; v++)
      if (mstSet[v] == 0 && key[v] < min)
         min = key[v], minIndex = v;

   return minIndex;
}

void printMST(int parent[], int graph[V][V], int n) {
   printf("Edge \tWeight\n");
   for (i = 1; i < n; i++)
      printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
}

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

   for ( i = 0; i < n; i++) {
      key[i] = INT_MAX;
      mstSet[i] = 0;
   }

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

   int count = 0;

   for (count = 0; count < n - 1; count++) {
      int u = minKey(key, mstSet, n);
      mstSet[u] = 1;
```

```
        for ( v = 0; v < n; v++) {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph, n);
}

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

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

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

    primMST(graph, n);

    return 0;
}
```

**OUTPUT:**

**Enter the number of vertices: 4**
**Enter the adjacency matrix (4x4):**
**0 2 0 6**
**2 0 3 8**
**0 3 0 0**
**6 8 0 0**

## 2) DIJKSHTRA ALGO:

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

#define V 100  // Maximum number of vertices

Int i,j,v;

int minDistance(int dist[], int visited[], int n) {
   int min = INT_MAX, minIndex;

   for ( v = 0; v < n; v++)
      if (visited[v] == 0 && dist[v] <= min)
         min = dist[v], minIndex = v;

   return minIndex;
}

void dijkstra(int graph[V][V], int n, int src) {
   int dist[n];     // Shortest distance from src to each vertex
   int visited[n];  // Whether the vertex has been visited

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

   dist[src] = 0; // Distance to self is 0

   int count=0;

   for (count = 0; count < n - 1; count++) {
      int u = minDistance(dist, visited, n);
      visited[u] = 1;

      for ( v = 0; v < n; v++) {
         if (!visited[v] && graph[u][v] && dist[u] != INT_MAX &&
            dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
         }
      }
   }
```

```
    printf("Vertex \tDistance from Source %d\n", src);
    for ( i = 0; i < n; i++)
        printf("%d \t%d\n", i, dist[i]);
}

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

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

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

    printf("Enter the source vertex (0 to %d): ", n - 1);
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

OUTPUT:

**Enter the number of vertices: 5**
**Enter the adjacency matrix (5x5):**
**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 to 4): 0**

## 3) BFS TRAVERSAL:

```
#include <stdio.h>

int queue[100], front = 0, rear = 0;
int i,j,v;

void enqueue(int v) {
    queue[rear++] = v;
}

int dequeue() {
    return queue[front++];
}

void bfs(int n, int graph[100][100], int start) {
    int visited[100] = {0};
    enqueue(start);
    visited[start] = 1;

    while (front < rear) {
        int u = dequeue();
        printf("%d ", u);

        for (v = 0; v < n; v++) {
            if (graph[u][v] && !visited[v]) {
                enqueue(v);
                visited[v] = 1;
            }
        }
    }
}

int main() {
    int n, graph[100][100], start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

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

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

    printf("BFS traversal: ");
    bfs(n, graph, start);
    return 0;
}
```

OUTPUT:

**Enter number of vertices: 4**
**Enter adjacency matrix:**
**0 1 1 0**
**1 0 0 1**
**1 0 0 1**
**0 1 1 0**
**Enter starting vertex: 0**

**BFS traversal: 0 1 2 3**

## 4) CHECK IF GRAPH IS CONNECTED THROUGH DFS:

```
#include <stdio.h>

void dfs(int graph[100][100], int visited[100], int n, int u) {
    visited[u] = 1;
    int v;
    for ( v = 0; v < n; v++) {
        if (graph[u][v] && !visited[v]) {
            dfs(graph, visited, n, v);
        }
    }
}

int main() {
    int n, graph[100][100], visited[100] = {0};
    int start;
    int i,j;

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

    printf("Enter adjacency matrix:\n");
    for ( i = 0; i < n; i++)
        for ( j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);
    printf(“Enter the starting vertex:”);
    scanf(“%d”,&start);

    // Start DFS from starting vertex
   dfs(graph, visited, n, start);

    // Check if all vertices were visited
    int isConnected = 1;
    for ( i = 0; i < n; i++) {
        if (!visited[i]) {
            isConnected = 0;
            break;
        }
    }

    if (isConnected)
        printf("The graph is CONNECTED.\n");
```

```
    else
        printf("The graph is NOT CONNECTED.\n");

    return 0;
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter adjacency matrix:
0 1 0 1
1 0 1 1
0 1 0 1
1 1 1 0

The graph is CONNECTED.
```

## 5) TRANSITIVE CLOSURE (WARSHALL'S ALGO):

```
#include <stdio.h>

int main() {
    int n, i, j, k;
    int graph[100][100];

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

    printf("Enter adjacency matrix (0 or 1 only):\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    // Warshall's Algorithm
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (graph[i][j] == 0 && graph[i][k] && graph[k][j])
                    graph[i][j] = 1;
            }
        }
    }

    printf("\nTransitive Closure:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            printf("%d ", graph[i][j]);
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

**Enter number of vertices: 3**
**Enter adjacency matrix (0 or 1 only):** **Transitive Closure:**
**1 1 1** **0 1 0**
**1 1 1** **0 0 1**

# 6) ALL PAIRS SHORTEST PATH (FLOYD WARSHALL ALGO):

```
#include <stdio.h>
#define INF 99999

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

int main() {
   int n, i, j, k;
   int dist[100][100];

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

   printf("Enter the adjacency matrix (use 99999 for INF):\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &dist[i][j]);

   // Floyd–Warshall Algorithm
   for (k = 0; k < n; k++) {
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }

   printf("\nAll Pairs Shortest Path Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         if (dist[i][j] == INF)
            printf("INF ");
         else
            printf("%d ", dist[i][j]);
      }
      printf("\n");
```

```
    }

    return 0;
}
```

OUTPUT:

**Enter number of vertices: 4**
**Enter the adjacency matrix (use 99999 for INF):**
**0 5 99999 10**
**99999 0 3 99999**
**99999 99999 0 1**
**99999 99999 99999 0**

**All Pairs Shortest Path Matrix:**
**0 5 8 9**
**INF 0 3 4**
**INF INF 0 1**
**INF INF INF 0**

## 7) 0/1 knapsack or fractional knapsack using greedy:

```
#include <stdio.h>

struct Item {
    int value, weight, index;
};

int main() {
    int n, capacity;
    struct Item items[100];
    float ratio[100], used[100] = {0};
    int  i,j;

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

    for ( i = 0; i < n; i++) {
        printf("Item %d (value weight): ", i + 1);
        scanf("%d %d", &items[i].value, &items[i].weight);
        items[i].index = i + 1; // to remember original item number
        ratio[i] = (float)items[i].value / items[i].weight;
    }

    // Sort items by value/weight ratio (descending)
    for ( i = 0; i < n-1; i++)
        for ( j = i+1; j < n; j++)
            if (ratio[i] < ratio[j]) {
                struct Item temp = items[i];
                items[i] = items[j];
                items[j] = temp;
                float t = ratio[i];
                ratio[i] = ratio[j];
                ratio[j] = t;
            }

    float total = 0;
    for ( i = 0; i < n ; i++) {
        if (items[i].weight <= capacity) {
            total += items[i].value;
            used[i] = 1; // 100% used
            capacity -= items[i].weight;
```

```
        } else {
            used[i] = (float)capacity / items[i].weight; // fraction used
            total += ratio[i] * capacity;
            capacity = 0;
        }
    }

    printf("\nItems included in the knapsack:\n");
    for (i = 0; i < n; i++) {
        if (used[i] > 0) {
            if (used[i] == 1)
                printf("Item %d: FULL (value = %d, weight = %d)\n",
                    items[i].index, items[i].value, items[i].weight);
            else
                printf("Item %d: %.2f%% (value = %d, weight = %d)\n",
                    items[i].index, used[i] * 100,
                    items[i].value, items[i].weight);
        }
    }

    printf("\nMaximum value = %.2f\n", total);
    return 0;
}
```

OUTPUT:

**Enter number of items and capacity: 3 50**
**Item 1 (value weight): 60 10**
**Item 2 (value weight): 100 20**
**Item 3 (value weight): 120 30**

**Items included in the knapsack:**
**Item 1: FULL (value = 60, weight = 10)**
**Item 2: FULL (value = 100, weight = 20)**
**Item 3: 66.67% (value = 120, weight = 30)**

**Maximum value = 240.00**

## 8) 0/1 KNAPSACK USING DYNAMIC APPROACH:

```
#include <stdio.h>

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

int main() {
   int n, W;
   int val[100], wt[100];
   int i, w;

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

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

   printf("Enter value and weight of each item:\n");
   for (i = 0; i < n; i++) {
      printf("Item %d - value and weight: ", i + 1);
      scanf("%d %d", &val[i], &wt[i]);
   }

   int dp[101][101]; // dp[i][w] = max value using first i items and capacity w

   // Build DP table
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         if (i == 0 || w == 0)
            dp[i][w] = 0; // base case
         else if (wt[i - 1] <= w)
            dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
         else
            dp[i][w] = dp[i - 1][w];
      }
   }

   printf("\nMaximum value in knapsack = %d\n", dp[n][W]);

    printf("\nMaximum value in knapsack = %d\n", dp[n][W]);
```

```
    printf("items included in the knapsack:\n");
    i=n,w=W;
    while(i>0 && w>0){
        if(dp[i][w] != dp[i-1][w]){
            printf("item %d : value %d weight %d\n",i,val[i-1],wt[i-1]);
            w -= wt[i-1];
            }
        i--;
    }


    return 0;
}
```

OUTPUT:

**Enter number of items: 3**
**Enter the capacity of knapsack: 50**
**Enter value and weight of each item:**
**Item 1 - value and weight: 60 10**
**Item 2 - value and weight: 100 20**
**Item 3 - value and weight: 120 30**

**Maximum value in knapsack = 220**

## 9) N-QUEENS :

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

int board[10], count = 0;
int  i,j;

int isSafe(int row, int col) {

   for (i = 1; i < row; i++) {
      if (board[i] == col || abs(board[i] - col) == abs(i - row))
         return 0; // Same column or diagonal
   }
   return 1;
}

void solve(int row, int n) {
   int col;
   if (row > n) {
      count++;
      printf("Solution %d: \n", count);
      for ( i = 1; i <= n; i++) {
         for ( j = 1; j <= n; j++) {
            if (board[i] == j)
               printf("Q ");
            else
               printf(". ");
         }
         printf("\n");
      }
      printf("\n");

   } else {
      for (col = 1; col <= n; col++) {
         if (isSafe(row, col)) {
            board[row] = col;
            solve(row + 1, n); // Go to next row
         }
      }
   }
```

```
}

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

    solve(1, n);

    if (count == 0)
        printf("No solutions found!\n");
    else
        printf("Total solutions: %d\n", count);

    return 0;
}
```

OUTPUT:

**Enter number of queens: 4**
**Solution 1:**
**. Q . .**
**. . . Q**
**Q . . .**
**. . Q .**

**Solution 2:**
**. . Q .**
**Q . . .**
**. . . Q**
**. Q . .**

**Total solutions: 2**

## 10) HAMILTONIAN CYCLE:

```
#include <stdio.h>

int graph[20][20], path[20], n;
int i,j,v;

int isSafe(int v, int pos) {
    if (graph[path[pos - 1]][v] == 0)
   return 0;
    for ( i = 0; i < pos; i++)
        if (path[i] == v)
     return 0;
    return 1;
}

int solve(int pos) {
    if (pos == n)
   return graph[path[pos - 1]][path[0]];

    for (v = 1; v < n; v++) {
        if (isSafe(v, pos)) {
            path[pos] = v;
            if (solve(pos + 1))
                  return 1;
            path[pos] = -1;
        }
    }
    return 0;
}

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

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

    for ( i = 0; i < n; i++)
    path[i] = -1;
```

```
    path[0] = 0;

    if (solve(1)) {
        printf("Hamiltonian Cycle: ");
        for (int i = 0; i < n; i++){
         printf("%d ", path[i]);
        printf("%d\n", path[0]);
   }
    } else {
        printf("No Hamiltonian Cycle found\n");
    }

    return 0;
}
```

OUTPUT:

**Enter number of vertices: 4**
**Enter 4 x 4 adjacency matrix:**
**0 1 1 0**
**1 0 1 1**
**1 1 0 1**
**0 1 1 0**

**Hamiltonian Cycle: 0 1 2 3 0**

## 11) TRAVELLING SALES PERSON (DYNAMIC):

```
#include <stdio.h>
#define INF 99999

int n, graph[20][20];
int dp[20][1 << 15];
int parent[20][1 << 15]; // To store the path
int i, j;

int tsp(int pos, int mask) {
    if (mask == (1 << n) - 1)
        return graph[pos][0];

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

    int min = INF;
    int city;

    for (city = 0; city < n; city++) {
        if ((mask & (1 << city)) == 0) {
            int newCost = graph[pos][city] + tsp(city, mask | (1 << city));
            if (newCost < min) {
                min = newCost;
                parent[pos][mask] = city; // store next city in path
            }
        }
    }

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

void printPath() {
    int mask = 1, pos = 0;
    printf("Path: %d ", pos); // Start from city 0
    while (mask != (1 << n) - 1) {
        int nextCity = parent[pos][mask];
        printf("-> %d ", nextCity);
        mask |= (1 << nextCity);
        pos = nextCity;
    }
```

```
    printf("-> 0\n"); // return to start
}

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

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

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

    int cost = tsp(0, 1);
    printf("Minimum cost of TSP tour: %d\n", cost);
    printPath();

    return 0;
}
```

OUTPUT:

**Enter number of cities: 4**
**Enter cost matrix:**
**0 10 15 20**
**10 0 35 25**
**15 35 0 30**
**20 25 30 0**

**Minimum cost of TSP tour: 80**

**Path: 0 -> 1 -> 3 -> 2 -> 0**

# 12) OPTIMAL BST:

```
#include <stdio.h>
#define MAX 20
#define INF 99999

float cost[MAX][MAX], p[MAX];
int n;

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

void obst() {
    int i, j, r, l;
    float c;

    for (i = 0; i < n; i++)
        cost[i][i] = p[i];

    for (l = 2; l <= n; l++) {
        for (i = 0; i <= n - l; i++) {
            j = i + l - 1;
            cost[i][j] = INF;

            for (r = i; r <= j; r++) {
                c = ((r > i) ? cost[i][r - 1] : 0) +
                    ((r < j) ? cost[r + 1][j] : 0) +
                    sum(i, j);

                if (c < cost[i][j])
                    cost[i][j] = c;
            }
        }
    }
}

int main() {
    int i;
```

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

    printf("Enter probabilities:\n");
    for (i = 0; i < n; i++)
        scanf("%f", &p[i]);

    obst();

    printf("Minimum cost of OBST: %.2f\n", cost[0][n - 1]);
    return 0;
}
```

OUTPUT:

**Enter number of keys: 3**
**Enter probabilities:**
**0.2 0.5 0.3**

**Minimum cost of OBST: 1.40**