7. Write a program to implement Matrix Chain Multiplication

```
#include <iostream>
#include <vector>

#include <climits>

using namespace std;

int matrixChainMultiplication(vector<int>& dims) {
   int n = dims.size(); vector<vector<int>>
   dp(n, vector<int>(n, 0));

   for (int length = 2; length < n; ++length) {
      for (int i = 1; i < n - length + 1; ++i) {
         int j = i + length - 1;
         dp[i][j] = INT_MAX; for
         (int k = i; k < j; ++k) {
            int cost = dp[i][k] + dp[k + 1][j] + dims[i - 1] * dims[k] * dims[j];
            dp[i][j] = min(dp[i][j], cost);
         }
      }
   }

   return dp[1][n - 1];
}

int main() {
   int n; cout << "Enter the number of
   matrices: "; cin >> n; vector<int>
   dimensions(n + 1);

   cout << "Enter the dimensions of the matrices: \n";
   for (int i = 0; i <= n; ++i) {
      cin >> dimensions[i];
   }

   int minCost = matrixChainMultiplication(dimensions); cout <<
   "Minimum number of multiplications is: " << minCost << endl;

   return 0;
}
```

Input :-

```
Enter the number of matrices: 4
Enter the dimensions of the matrices:
40 20 30 10 30
```

Output :- Minimum number of multiplications is: 26000

8. Implement Knapsack Problem using Greedy Solution

```
#include <iostream>
#include <vector>

#include <algorithm>

using namespace std;

struct Item {
   double weight, value;
};

bool compare(const Item &a, const Item &b) {
   return (a.value / a.weight) > (b.value / b.weight);
}

double fractionalKnapsack(vector<Item> &items, double maxWeight) {
   sort(items.begin(), items.end(),
   compare); double totalValue = 0.0; for
   (const auto &item : items) { if
   (maxWeight >= item.weight) {
   totalValue += item.value; maxWeight -=
   item.weight;
      } else { totalValue += item.value * (maxWeight /
         item.weight); break;
      } } return
   totalValue;
}

int main() {
   int n; double
   maxWeight;

   cout << "Enter the number of items: ";
   cin >> n;

   vector<Item> items(n);
   for (int i = 0; i < n; ++i) {
      cout << "Enter weight and value for item " << i + 1 << ": ";
      cin >> items[i].weight >> items[i].value;
   }

   cout << "Enter the maximum weight of the knapsack: ";
   cin >> maxWeight;

   double maxValue = fractionalKnapsack(items, maxWeight); cout << "Maximum value in
   knapsack = " << fixed << setprecision(2) << maxValue << endl;
```

```
    return 0;
}
```

Input :-

```
Enter the number of items: 3
Enter weight and value for item 1: 10 60
Enter weight and value for item 2: 20 100
Enter weight and value for item 3: 30 120
Enter the maximum weight of the knapsack: 50
```

Output : Maximum value in knapsack

= 240.00

9. Find Minimum Spanning Tree using Kruskal’s Algorithm

```
#include <iostream>
#include <vector>

#include <algorithm>

using namespace std;

// Structure to represent an edge
struct Edge {
   int u, v, weight;
};

// Find function for union-find int
findParent(int node, vector<int>& parent) {
   if (node == parent[node]) return node; return
   parent[node] = findParent(parent[node], parent);
}

// Union function for union-find void unionNodes(int u, int v,
vector<int>& parent, vector<int>& rank) {
   u = findParent(u, parent);
   v = findParent(v, parent);
   if (rank[u] < rank[v]) {
      parent[u] = v;
   } else if (rank[u] > rank[v]) {
      parent[v] = u;
   } else {
      parent[v] = u;
      rank[u]++;
   }
}

// Kruskal's Algorithm to find MST void
kruskalMST(int n, vector<Edge>& edges) {
   sort(edges.begin(), edges.end(), [](Edge a, Edge b) {
      return a.weight < b.weight;
   });

   vector<int> parent(n);
   vector<int> rank(n, 0);
   for (int i = 0; i < n; i++)
   { parent[i] = i; }

   vector<Edge> mst;
   int totalWeight = 0;

   for (auto edge : edges) { if (findParent(edge.u, parent) !=
      findParent(edge.v, parent)) {
           mst.push_back(edge); totalWeight +=
           edge.weight; unionNodes(edge.u, edge.v,
           parent, rank); } }

   cout << "Edges in the Minimum Spanning Tree:\n";
   for (auto edge : mst) {
      cout << edge.u << " - " << edge.v << " : " << edge.weight << endl;
   } cout << "Total weight of the Minimum Spanning Tree: " << totalWeight <<
   endl;
}

int main() {
   int n, e; cout << "Enter the number of
   vertices: "; cin >> n; cout << "Enter the
   number of edges: "; cin >> e;

   vector<Edge> edges(e); cout << "Enter
   the edges (u v weight):\n"; for (int i = 0;
   i < e; i++) {
      cin >> edges[i].u >> edges[i].v >> edges[i].weight;
   }

   kruskalMST(n, edges);
   return 0;
}
```

Input :-

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
```

Output :-

```
Edges in the Minimum Spanning Tree:
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10
Total weight of the Minimum Spanning Tree: 19
```

```
#include
#include
```

10. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm

```
          <iostream>
          <vector>

#include <climits>

using namespace std;

int findMinVertex(vector<int>& key, vector<bool>& mstSet, int V) {
   int minIndex = -1, minValue =
   INT_MAX; for (int i = 0; i < V; i++) { if
   (!mstSet[i] && key[i] < minValue) {
         minValue = key[i];
         minIndex = i;
      } } return
   minIndex;
}

void primsAlgorithm(vector<vector<int>>& graph, int V) {
   vector<int> parent(V, -1), key(V,
   INT_MAX); vector<bool> mstSet(V, false);
   key[0] = 0;

   for (int count = 0; count < V - 1; count++) {
      int u = findMinVertex(key, mstSet, V);
      mstSet[u] = true;

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

   cout << "Edge   Weight\n";
   int totalCost = 0; for (int i
   = 1; i < V; i++) {
      cout << parent[i] << " - " << i << "           " << graph[i][parent[i]] << endl;
      totalCost += graph[i][parent[i]];
   } cout << "Minimum Cost: " << totalCost <<
   endl;
}

int main() {
   int V; cout << "Enter the number of
   vertices: "; cin >> V;
```

```
    vector<vector<int>> graph(V, vector<int>(V));
    cout << "Enter the adjacency matrix (use 0 for no
    edge):\n"; for (int i = 0; i < V; i++) for (int j = 0; j < V;
    j++) cin >> graph[i][j]; primsAlgorithm(graph, V);

    return 0;
}
```

Input :-

```
Enter the number of vertices: 5
Enter the adjacency matrix (use 0 for no edge):
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
```

Output :-

```
Edge    Weight

0 - 1                                                          5

1 - 2                                                         17

0 - 3                                                         17


1 - 4   5

Minimum Cost: 16
```

```
#include
#include
```

11. Implement, the 0/1 Knapsack problem using Dynamic Programming method.

```
             <iostream>
             <vector>

#include <algorithm>

using namespace std;

int knapsack(int W, vector<int>& weights, vector<int>& values, int n) {
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

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

    return dp[n][W];
}

int main() {
    int n, W;

    cout << "Enter the number of items: ";
    cin >> n;

    cout << "Enter the capacity of the knapsack: ";
    cin >> W;

    vector<int> weights(n), values(n);

    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; ++i) cin >> weights[i];

    cout << "Enter the values of the items: "; for (int i = 0; i < n;

    ++i) cin >> values[i]; int maxValue = knapsack(W, weights,

    values, n); cout << "Maximum value in knapsack: " <<

    maxValue << endl;

    return 0;
}
```

Input :-

```
Enter the number of items: 4
Enter the capacity of the knapsack: 50
Enter the weights of the items: 10 20 30 40
Enter the values of the items: 60 100 120 150
```

Output : Maximum value in

knapsack: 220

```
#include
#include
```

12. Write programs to implement All-Pairs Shortest Paths problem using Floyd's algorithm

```
       <iostream>
       <climits>
using namespace std;

void floydWarshall(int graph[][4], int V) {
  int dist[V][V];

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

  for (int k = 0; k < V; k++) {
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
        if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][j] > dist[i][k] + dist[k][j]) {
          dist[i][j] = dist[i][k] + dist[k][j];
        }
      }
    }
  }

  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      if (dist[i][j] == INT_MAX) {
        cout << "INF ";
      } else { cout << dist[i][j]
        << " ";
      } } cout
    << endl;
  }
}

int main() {
  int graph[4][4] = {
    {0, 3, INT_MAX, 7},
    {8, 0, 2, INT_MAX},
    {5, INT_MAX, 0, 1},
    {2, INT_MAX, INT_MAX, 0}
  };
  floydWarshall(graph, 4);
  return 0;
}
```

Input :-

```
4
0 3 INF 7
8 0 2 INF
5 INF 0 1
2 INF INF 0
```

Output :-

```
0 3 5 6
8 0 2 3
5 8 0 1
2 5 7 0
```

```
#include
#include
```

13.Write programs to implement Travelling Sales Person problem using Dynamic programming

```
        <iostream>
        <vector>
#include <climits>
using namespace std;

int tsp(int mask, int pos, vector<vector<int>>& dist, vector<vector<int>>& dp, int n) {
    if(mask == (1 << n) - 1) {
        return dist[pos][0];
    } if(dp[mask][pos] != -
    1) {
        return dp[mask][pos];
    } int ans = INT_MAX; for(int
    city = 0; city < n; city++) {
        if((mask & (1 << city)) == 0) {
            int newAns = dist[pos][city] + tsp(mask | (1 << city), city, dist, dp, n);
            ans = min(ans, newAns);
        } } dp[mask][pos]
= ans; return ans; }

int main() {
    int n; cin
    >> n;
    vector<vector<int>> dist(n, vector<int>(n));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> dist[i][j];
        } } vector<vector<int>> dp(1 << n,
    vector<int>(n, -1)); cout << tsp(1, 0, dist, dp, n) <<
    endl; return 0;
}
```

Input :-

```
4
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
```

Output :-

```
80
```

14. Implement N Queen Problem using Backtracking

```
#include <iostream>
#include <vector>
using namespace std;

bool isSafe(vector<vector<int>>& board, 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; i < N && j >= 0; i++, j--)
      if (board[i][j]) return false;

   return true;
}

bool solveNQueen(vector<vector<int>>& board, 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
         (solveNQueen(board, col + 1, N))
            return true;
         board[i][col] = 0;
      } }
return false; }

void printBoard(vector<vector<int>>& board, int N) {
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         cout << board[i][j] << " ";
      } cout <<
      endl;
   }
}

int main() {
   int N; cin
   >> N;
   vector<vector<int>> board(N, vector<int>(N, 0));

   if (solveNQueen(board, 0, N)) {
      printBoard(board, N);
#include
#include

    } else { cout << "Solution does not exist"
        << endl;
    }
    ret
    urn
    0;
}
```

Input :-

```
4
```

Output :-

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

15. Design and implement to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle

```
#include <iostream>
#include <vector>
using namespace std;

class HamiltonianCycle
{ public: int n;
  vector<vector<int>> graph;

  HamiltonianCycle(int n) { this->n =
    n; graph.resize(n, vector<int>(n,
    0));
  }

  bool isSafe(int v, vector<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; }

  bool hamiltonianCycleUtil(vector<int>& path, int pos) {
    if (pos == n) { if (graph[path[pos -
      1]][path[0]] == 1)
        return true;
      return false;
    }

    for (int v = 1; v < n; v++) {
      if (isSafe(v, path, pos)) {
        path[pos] = v; if
        (hamiltonianCycleUtil(path, pos + 1))
          return true;
        path[pos] = -1;
      } }
    return
    false;
  }

  void printCycle(const vector<int>& path) {
    for (int i = 0; i < n; i++)
      cout << path[i] << " ";
    cout << path[0] << endl;
  } void

  findHamiltonianCycles() {
    vector<int> path(n, -1); path[0]

    = 0;


        if (hamiltonianCycleUtil(path, 1)) {
            printCycle(path);
        } else { cout << "No Hamiltonian Cycle found"
            << endl;
        }
    }
};

int main() {
    int n, e; cout << "Enter the number of
    vertices: "; cin >> n;
    HamiltonianCycle hc(n);

    cout << "Enter the number of edges: "; cin >> e;
    cout << "Enter the edges (pair of vertices): " <<
    endl; for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        hc.graph[u][v] = 1;
    hc.graph[v][u] = 1; }
    hc.findHamiltonianCycles()
    ;

    return 0;
}
```

Input :-

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edges (pair of vertices):
0 1
1 2
2 3
3 0
```

Output :-

```
0 1 2 3 0
```

16. Design and implement to find a subset of a given set S = {Sl, S2,……,Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

```
#include <iostream>
#include <vector>
using namespace std;

bool findSubset(vector<int>& S, int n, int d, vector<int>& result) {
   if (d == 0) return true;
   if (n == 0) return false;

   if (S[n-1] <= d) {
      result.push_back(S[n-1]); if (findSubset(S, n-1, d -
      S[n-1], result)) return true; result.pop_back();
   }
   return findSubset(S, n-1, d,
   result);
}

int main() {
   int n, d;
   cin >> n >> d;
   vector<int> S(n), result;

   for (int i = 0; i < n; i++) {
      cin >> S[i];
   }

   if (findSubset(S, n, d, result)) {
      cout << "Subset found: {"; for (int i
      = 0; i < result.size(); i++) { cout <<
      result[i];
         if (i != result.size() - 1) cout << ", ";
      } cout << "}" <<
      endl;
   } else { cout << "No subset found"
      << endl;
   }

   return 0;
}
```

Input :-

```
5 9

1 2 5 6 8
```

Output :-

```
Subset found: {1, 2, 6}
```