1. Program to implement set operations union, intersection, difference and cartesian product.

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

void unionSet(int set1[], int set2[], int m, int n) {
   int i, j = 0;
   while (i < m && j < n) {
      if(set1[i] < set2[j]) {
         cout<<set1[i]<<" ";
         i++;
      }
                  else if(set2[j] < set1[i]) {
         cout<<set2[j]<<" ";
         j++;
      }
                  else {
         cout<<set1[i]<<" ";
         i++;
         j++;
      }
   }
   while(i < m) {
      cout<<set1[i]<<" ";
      i++;
   }
   while(j < n) {
      cout<<set2[j]<<" ";
      j++;
   }
   cout<<endl;
}
```

```
void intersectionSet(int set1[], int set2[], int m, int n) {
   int i = 0, j = 0;
   while(i < m && j < n) {
      if(set1[i] < set2[j]) {
         i++;
      }
                  else if(set2[j] < set1[i]) {
         j++;
      }
                  else {
         cout<<set1[i]<<" ";
         i++;
         j++;
      }
   }
   cout<<endl;
}

void differenceSet(int set1[], int set2[], int m, int n) {
   int i = 0, j = 0;
   while(i < m && j < n) {
      if(set1[i] < set2[j]) {
         cout<<set1[i]<<" ";
         i++;
      }
                  else if(set2[j] < set1[i]) {
         j++;
      }
                  else {
         i++;
         j++;
      }
   }
   while(i < m) {
      cout<<set1[i]<<" ";
```

```
      i++;
   }
   cout<<endl;
}

void cartesianProduct(int set1[], int set2[], int m, int n) {
   for(int i = 0; i < m; i++) {
      for(int j = 0; j < n; j++) {
         cout<<"("<<set1[i]<<", "<<set2[j]<<") ";
      }
   }
   cout<<endl;
}

int main() {
   int m, n;

   cout<<"Enter the size of Set 1: ";
   cin>>m;
   int set1[m];
   cout<<"Enter the elements of Set 1: ";
   for(int i = 0; i < m; i++) {
         cin>>set1[i];
         }

         cout<<"\nEnter the size of Set 2: ";
         cin>>n;
         int set2[n];
         cout<<"Enter the elements of Set 2: ";
         for(int i = 0; i < n; i++) {
                  cin>>set2[i];
         }

         cout<<endl;
```

```
    cout<<"Union of sets: ";
    unionSet(set1, set2, m, n);
    cout<<endl;

    cout<<"Intersection of sets: ";
    intersectionSet(set1, set2, m, n);
    cout<<endl;

    cout<<"Difference of sets (Set1 - Set2): ";
    differenceSet(set1, set2, m, n);
    cout<<endl;

    cout<<"Cartesian Product of sets: ";
    cartesianProduct(set1, set2, m, n);
    cout<<endl;

    return 0;
}
```

2. Program to implement ceiling and floor function.

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

void ceilFunction(double n) {
        cout<<ceil(n)<<endl;
}

void floorFunction(double n) {
        cout<<floor(n)<<endl;
}
int main() {
        double num;
```

```
        cout<<"Enter a decimal place number: ";
        cin>>num;

        cout<<"\nCeiling of the given number is: ";
        ceilFunction(num);

        cout<<"Floor of the given number is: ";
        floorFunction(num);

        return 0;
}
```

3. Program to implement fuzzy set operations.

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

void Union(double s1[], double s2[], double result[], int n) {
   for (int i = 0; i < n; i++) {
      result[i] = max(s1[i], s2[i]);
   }
}

void Intersection(double s1[], double s2[], double result[], int n) {
   for (int i = 0; i < n; i++) {
      result[i] = min(s1[i], s2[i]);
   }
}

void Complement(double s1[], double result[], int n) {
        for(int i = 0; i < n; i++) {
                result[i] = 1 - s1[i];
        }
}
```

```
void Difference(double s1[], double s2[], double result[], int n) {
        for(int i = 0; i < n; i++) {
                result[i] = min(s1[i], 1-s2[i]);
        }
}

void print(double s[], int n) {
   for (int i = 0; i < n; i++) {
      cout << s[i] << " ";
   }
   cout << endl;
}

int main() {
   int num;

   cout<<"Enter the no. of elements of a Set: ";
   cin>>num;

   double a[num];
   double b[num];

//   double a[n] = {0.2, 0.4, 0.6, 0.8, 1.0};
//   double b[n] = {0.1, 0.3, 0.5, 0.7, 0.9};

   double unionResult[num];
   double intersectionResult[num];
   double complementResult[num];
   double differenceResult[num];

   cout<<"\nEnter the elements of set A: ";
   for(int i = 0; i < num; i++) {
        cin>>a[i];
        }
```

```
    cout<<"\nEnter the elements of Set B: ";
    for(int i = 0; i < num; i++) {
        cin>>b[i];
        }

    Union(a, b, unionResult, num);
    cout << "\nUnion of the sets: ";
    print(unionResult, num);

    Intersection(a, b, intersectionResult, num);
    cout << "Intersection of the sets: ";
    print(intersectionResult, num);

    Complement(a, complementResult, num);
    cout<<"Complement of the sets: ";
    print(complementResult, num);

    Difference(a, b, differenceResult, num);
    cout<<"Difference of the sets: ";
    print(differenceResult, num);

    return 0;
}
```

4. Program to implement Euclidean and Extended Euclidean.

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

// Function to find the GCD (Greatest Common Divisor) using Euclidean Algorithm
int gcd(int a, int b) {
    if (b == 0){
        return a;
        }
    return gcd(b, a % b);
```

```
}

// Function to find the extended GCD using the extended Euclidean algorithm
int extendedGCD(int a, int b, int &x, int &y) {
   if (a == 0) {
      x = 0;
      y = 1;
      return b;
   }

   int x1, y1;
   int gcd = extendedGCD(b % a, a, x1, y1);

   x = y1 - (b / a) * x1;
   y = x1;

   return gcd;
}

int main() {
   int a, b;
   cout << "Enter two numbers (a and b): ";
   cin >> a >> b;

   int gcd_result = gcd(a, b);

   int x, y;
   int extended_gcd_result = extendedGCD(a, b, x, y);

         cout << "GCD(" << a << ", " << b << ") = " << gcd_result <<endl;
   cout << "Extended GCD(" << a << ", " << b << ") = " << extended_gcd_result
<<endl;
   cout << "t = " << x << ", s = " << y <<endl;

   return 0;
```

```
}
```

5. Program to implement binary integer addition, multiplication and division.

```
#include<iostream>
#include<cmath>
using namespace std;
//const int len = 4;

void getInput(int arr[], int len) {

   cout << "Enter binary number: ";
   for (int i = len - 1; i >= 0; i--) {
      cin >> arr[i];
   }
}

void addBinary(int a[], int b[], int len) {
   int c = 0, d, sum[len + 1] = {0};
   for (int i = 0; i < len; i++) {
      d = floor((a[i] + b[i] + c) / 2);
      sum[i] = (a[i] + b[i] + c) - 2 * d;
      c = d;
   }
   sum[len] = c;
   cout << "\nSum is: ";
   for (int i = len; i >= 0; i--) {
      cout << sum[i];
   }
}

void multiplyBinary(int a[], int b[], int len) {
   int product[len * 2] = {0};
   for (int j = 0; j < len; j++) {
      if (b[j] == 1) {
```

```
            for (int k = 0; k < len; k++) {
               product[j + k] += a[k];
            }
         }
      }
      for (int i = 0; i < 2 * len - 1; i++) {
         product[i + 1] += product[i] / 2;
         product[i] %= 2;
      }
      cout << "\nProduct is: ";
      for (int i = 2 * len - 1; i >= 0; i--) {
         cout << product[i];
      }
}
void division() {
            int a, d, q, r;
            cout<<"\nEnter Dividend and Divisor: ";
            cin>>a>>d;
            q = 0;
            r = (abs(a));
            while (r >= d) {
                        r = r - d;
                        q = q + 1;
            }
            cout<<"Remainder (r): "<<r<<endl;
            cout<<"Quotient (q): "<<q<<endl;

            if( a < 0 && r > 0) {
                        r = d - r;
                        q = (-q + 1);
                        cout<<"Remainder (r): "<<r<<endl;
                        cout<<"Quotient (q):"<<q<<endl;
            }
}
```

```
int main() {
        int len;
        cout<<"Enter length of binary number: ";
        cin>>len;

   int a[len], b[len];
   getInput(a, len);
   getInput(b, len);

   addBinary(a, b, len);
   multiplyBinary(a, b, len);

   division();
   return 0;
}
```

6. Program to implement Boolean matrix operation join, product, and Boolean Product.

```
#include <iostream>
using namespace std;
const int N = 100;


void inputMatrix(int matrix[][N], int& r, int& c) {
   cout<<"Enter the number of rows: ";
   cin>>r;
   cout<<"Enter the number of columns: ";
   cin>>c;

   cout<<"Enter the Boolean matrix (0 or 1):\n";
   for(int i = 0; i < r; i++) {
      for(int j = 0; j < c; j++) {
         cin>>matrix[i][j];
      }
```

```
   }
}

void displayMatrix(int matrix[][N], int r, int c) {
   for(int i = 0; i < r; i++) {
      for(int j = 0; j < c; j++) {
         cout<<matrix[i][j] << " ";
      }
      cout<<endl;
   }
}

void joinMatrices(int m1[][N], int m2[][N], int result[][N], int r, int c) {
   for(int i = 0; i < r; i++) {
      for(int j = 0; j < c; j++) {
         result[i][j] = m1[i][j] || m2[i][j];
      }
   }
}

void productMatrices(int m1[][N], int m2[][N], int result[][N], int r1, int c1, int c2)
{
   for(int i = 0; i < r1; i++) {
      for(int j = 0; j < c2; j++) {
         result[i][j] = 0;
         for(int k = 0; k < c1; k++) {
            result[i][j] = result[i][j] || (m1[i][k] && m2[k][j]);
         }
      }
   }
}

void booleanProduct(int m1[][N], int m2[][N], int result[][N], int r, int c) {
   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
```

```
            result[i][j] = m1[i][j] && m2[i][j];
        }
    }
}


int main() {
    int matrix1[N][N];
    int matrix2[N][N];
    int result[N][N];
    int r1, c1, r2, c2;

    cout<<"Enter the first Boolean matrix:\n";
    inputMatrix(matrix1, r1, c1);

    cout<<"Enter the second Boolean matrix:\n";
    inputMatrix(matrix2, r2, c2);

    if(r1 != r2 || c1 != c2) {
        cout<<"Matrix dimensions do not match for operations. Exiting.\n";
        return 1;
    }

    cout<<"First Boolean matrix:\n";
    displayMatrix(matrix1, r1, c1);

    cout<<"Second Boolean matrix:\n";
    displayMatrix(matrix2, r2, c2);

    joinMatrices(matrix1, matrix2, result, r1, c1);
    cout << "Join of matrices:\n";
    displayMatrix(result, r1, c1);

    productMatrices(matrix1, matrix2, result, r1, c1, c2);
    cout << "Product of matrices:\n";
```

```
    displayMatrix(result, r1, c2);

    booleanProduct(matrix1, matrix2, result, r1, c1);
    cout << "Boolean Product of matrices:\n";
    displayMatrix(result, r1, c1);

    return 0;
}
```

7. Program to perform operations with large integers by breaking down them into set of small integers.

```
#include <iostream>
using namespace std;
const int m[3] = {95, 97, 98};
const int mod = m[0] * m[1] * m[2];

void findTuple(int a[3], int num) {
    for (int i = 0; i < 3; i++) {
        a[i] = num % m[i];
    }
}

void tuples(int a[3], int b[3], int s[3], int p[3]) {
    for (int i = 0; i < 3; i++) {
        s[i] = a[i] + b[i];
        p[i] = a[i] * b[i];
        if (s[i] > 99){
            s[i] %= m[i];
        }
        if (p[i] > 99) {
            p[i] %= m[i];
        }
    }
}
```

```
void calcInverse(int M[3], int x[3]) {
   for (int i = 0; i < 3; i++) {
      M[i] = mod / m[i];
      for (int k = 1; k < m[i]; k++) {
         if (((M[i] * k) % m[i]) == 1) {
            x[i] = k;
            break;
         }
      }
   }
}

void operate(int a[3], int b[3], int n1, int n2) {
   int s[3], p[3], M[3], x[3];

   tuples(a, b, s, p);
   calcInverse(M, x);

   int sum = 0, product = 0;
   for (int i = 0; i < 3; i++) {
      sum += (s[i] * M[i] * x[i]);
      product += (p[i] * M[i] * x[i]);
   }
   sum %= mod;
   product %= mod;

   cout<<"\nSum of "<<n1<<" and "<<n2<<" is: "<<sum<<endl;
   cout<<"Product of "<<n1<<" and "<<n2<<" is: "<<product<<endl;

}

int main() {
   int n1, n2;
   cout << "Enter two numbers: ";
```

```
        cin>>n1>>n2;

    int a[3], b[3];
    findTuple(a, n1);
    findTuple(b, n2);

    operate(a, b, n1, n2);

    return 0;
}
```

8. Program to generate truth tables of compound propositions.

```
#include<iostream>
using namespace std;
void getNegation() {
        int i;
        int a[] = {0, 1}, c[2];
        cout<<"A\t~A"<<endl;
        for(i = 0; i < 2; i++) {
                c[i] = !a[i];
                cout<<"....."<<endl;
                cout<<a[i]<<"\t"<<c[i]<<endl;
        }
}
void getConjunction() {
        int i;
        int a[] = {0, 0, 1, 1}, b[] = {0, 1, 0, 1}, c[4];
        cout<<"A\tB\tAnB"<<endl;
        for(i = 0; i < 4; i++) {
                c[i] = a[i] && b[i];
                cout<<"....."<<endl;
                cout<<a[i]<<"\t"<<b[i]<<"\t"<<c[i]<<endl;
        }
}
```

```
void getDisjunction() {
	int i;
	int a[] = {0, 0, 1, 1}, b[] = {0, 1, 0, 1}, c[4];
	cout<<"A\tB\tAuB"<<endl;
	for(i = 0; i < 4; i++) {
		c[i] = a[i] || b[i];
		cout<<"....................."<<endl;
		cout<<a[i]<<"\t"<<b[i]<<"\t"<<c[i]<<endl;
	}
}
void getImplication() {
	int i;
	int a[] = {0, 0, 1, 1}, b[] = {0, 1, 0, 1}, c[4];
	cout<<"A\tB\tA->B"<<endl;
	for(i = 0; i < 4; i++) {
		c[i] = (!a[i]) || (a[i] && b[i]);
		cout<<"....................."<<endl;
		cout<<a[i]<<"\t"<<b[i]<<"\t"<<c[i]<<endl;
	}
}
void getBiconditional() {
	int i;
	int a[] = {0, 0, 1, 1}, b[] = {0, 1, 0, 1}, c[4];
	cout<<"A\tB\tA<->B"<<endl;
	for(i = 0; i < 4; i++) {
		c[i] = ((!a[i]) || (a[i] && b[i])) && ((!b[i]) || (b[i] && a[i]));
		cout<<"....................."<<endl;
		cout<<a[i]<<"\t"<<b[i]<<"\t"<<c[i]<<endl;
	}
}

int main() {
	cout<<"Truth Table of Negation"<<endl;
	getNegation();
	cout<<"\nTruth Table of Conjunction"<<endl;
```

```
        getConjunction();
        cout<<"\nTruth Table of Disjunction"<<endl;
        getDisjunction();
        cout<<"\nTruth Table of Implication"<<endl;
        getImplication();
        cout<<"\nTruth Table of Biconditional"<<endl;
        getBiconditional();

        return 0;
}
```

9. Program to test validiy of arguments by using truth table.

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

int evaluate(int A, int B, const string& ch) {
  if (ch == "A && B") {
    return A && B;
  }
      else if (ch == "A || B") {
    return A || B;
  }
      else if (ch == "A && !B") {
    return A && !B;
  }
      else if (ch == "A || !B") {
    return A || !B;
  }
      else {
    cout<< "Invalid expression!" <<endl;
    return -1;
  }
}
```

```
int main() {
   int A, B;
   string expression;

   cout << "Enter the value of A (0 or 1): ";
   cin >> A;
   cout << "Enter the value of B (0 or 1): ";
   cin >> B;
   cout << "Enter a logical expression (e.g., A && B, A || B, A && !B, A || !B): ";
   cin.ignore();
   getline(cin, expression);

   int result = evaluate(A, B, expression);

   if (result == -1) {
      cout << "\nExpression is invalid." <<endl;
   }
         else {
      cout << "\nThe result of the expression is: " << result <<endl;
   }

   return 0;
}
```

10. Program to compute a^n, b^n mod m, linear search by using recursion.

```
#include <iostream>
using namespace std;
// Function to calculate (b^e) % m using recursion
int power(int b, int e, int m) {
   if (e == 0) {
      return 1;
   } else if (e % 2 == 0) {
      int p = power(b, e / 2, m);
```

```
        return (p * p) % m;
    } else {
        return (b * power(b, e - 1, m)) % m;
    }
}

int main() {
    int a, b, n, m;

    // Input values
    cout << "Enter the values of a, b, n, and m: ";
    cin >> a >> b >> n >> m;

    // Calculate a^n % m and b^n % m using the power function
    int result_a = power(a, n, m);
    int result_b = power(b, n, m);

    // Output the results
    cout << "\na^n % m = " << result_a << endl;
    cout << "b^n % m = " << result_b << endl;

    return 0;
}
```

11. Program to generate permutations and combinations

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

int factorial(int num) {
    if (num <= 1) return 1;
    return num * factorial(num - 1);
}

int permutations(int n, int r) {
```

```
    return factorial(n) / factorial(n - r);
}

int combinations(int n, int r) {
    return factorial(n) / (factorial(r) * factorial(n - r));
}

int main() {
    int n, r;
    cout<<"Enter the value of n: ";
    cin>>n;
    cout<<"Enter the value of r: ";
    cin>>r;

    if (n < 0 || r < 0 || r > n) {
        cout << "Invalid input. n and r should be non-negative, and r should be less
than or equal to n." << endl;
        return 1;
    }

    int nPr = permutations(n, r);
    int nCr = combinations(n, r);

    cout << "\nPermutations of " << n << " and " << r << " is: " << nPr << endl;
    cout << "Combinations of " << n << " and " << r << " is: " << nCr << endl;

    return 0;
}
```

12. Program to implement randomized algorithm.

```
#include <iostream>
using namespace std;
int main() {
    int a, c, m, x0, n;
```

```
    cout << "Enter the value of a, c, and m: ";
    cin >> a >> c >> m;
    cout<<"Enter the value of x[0]: ";
    cin>>x0;
    cout<<"Number of Random Number to generate: ";
    cin>>n;

    int x[n], i;
          x[0] = x0;

    for (i = 0; i < n; i++) {

       x[i + 1] = (a * x[i] + c) % m;

    }

    cout << "\nRandom Numbers are: ";
    for (i = 1; i <= n; i++) {
       cout << x[i] << "\t";
    }

    return 0;
}
```

13. Program for representing relations, testing its properties and testing equivalence.

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

int isReflexive(int r[][2], int n) {
   for (int i = 0; i < n; i++) {
      int found = 0;
      for (int j = 0; j < n; j++) {
         if (r[i][0] == r[j][0] && r[i][1] == r[j][1]) {
```

```
            found = 1;
            break;
        }
    }
    if (found == 0) {
        return 0;
    }
  }
  return 1;
}

int isSymmetric(int r[][2], int n) {
  for (int i = 0; i < n; i++) {
    int found = 0;
    for (int j = 0; j < n; j++) {
      if (r[i][0] == r[j][1] && r[i][1] == r[j][0]) {
        found = 1;
        break;
      }
    }
    if (found == 0) {
      return 0;
    }
  }
  return 1;
}

int isAntisymmetric(int r[][2], int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (i != j && r[i][0] == r[j][1] && r[i][1] == r[j][0]) {
        return 0;
      }
    }
  }
```

```
    return 1;
}

int isTransitive(int r[][2], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (r[i][1] == r[j][0]) {
                int found = 0;
                for (int k = 0; k < n; k++) {
                    if (r[i][0] == r[k][0] && r[j][1] == r[k][1]) {
                        found = 1;
                        break;
                    }
                }
                if (found == 0) {
                    return 0;
                }
            }
        }
    }
    return 1;
}

int isEquivalence(int r[][2], int n) {
    if (isReflexive(r, n) && isSymmetric(r, n) && isTransitive(r, n)) {
        return 1;
    }
    return 0;
}

int main() {
    int len;
    cout << "Enter the number of pairs in the relation: ";
    cin >> len;
```

```
    int relation[len][2];
    cout << "Enter the relation elements as a pair:" << endl;
    for (int i = 0; i < len; i++) {
       cin >> relation[i][0] >> relation[i][1];
    }

    cout<< "\nRelation:" << endl;
    cout<< " Reflexive: " << (isReflexive(relation, len) ? "1" : "0") << endl;
    cout<< " Symmetric: " << (isSymmetric(relation, len) ? "1" : "0") << endl;
    cout<< " Anti-Symmetric: " << (isAntisymmetric(relation, len) ? "1" : "0") <<
endl;
    cout<< " Transitive: " << (isTransitive(relation, len) ? "1" : "0") << endl;
    cout<< " Equivalence: " << (isEquivalence(relation, len) ? "1" : "0") << endl;
    cout<< "\n('1' means TRUE and '0' means FALSE.)";

    return 0;
}
```

14. Program to represent graphs.

```
// Program to represent graphs
#include<iostream>

using namespace std;

void graph(int v) {
        int i, j;
        char ch, a = 97, b = 97;
        int matrix[v][v];

        for(i = 0; i < v; i++) {
                for(j = 0; j < v; j++) {
                        cout<<"\nIs there edges between "<<a<<" and "<<b<<"?
(Enter 'y' for YES and 'n' for NO): ";
                        do {
```

```
                cin>>ch;
                if(ch == 'y') {
                        matrix[i][j] = 1;
                }
                else if(ch == 'n') {
                        matrix[i][j] = 0;
                }
                else {
                        cout<<"Invalid character!";
                        ch = 0;
                }

                b++;
            }
            while(ch == 0);
        }
        b = 97;
        a++;
    }

    cout<<endl;

    cout<<"Representing Graph in Matrix Form"<<endl;
    for(i = 0; i < v; i++) {
        for(j = 0; j < v; j++) {
            cout<<matrix[i][j]<<"\t";
        }
        cout<<endl;
    }
}

int main() {
    int vertices;

    cout<<"Enter the number of vertices: ";
```

```
        cin>>vertices;

        graph(vertices);

        return 0;
}
```

15. Program for finding shortest path (Dijkstra Algorithm).

```
#include <iostream>

using namespace std;

const int INFINITY = 999;

void setWeight(int** graph, int nvert) {
   char c1 = 'a', c2 = 'a';
   char ch;
   for (int i = 0; i < nvert; i++) {
      for (int j = 0; j < nvert; j++) {
         graph[i][j] = 99;
      }
   }
   for (int i = 0; i < nvert; i++) {
      for (int j = 0; j < nvert; j++) {
         if (graph[i][j] == 99) {
            if (i != j) {
               cout << "Is there an edge between " << c1 << " and " << c2 << "? (y/n):
";
               cin >> ch;
               if (ch == 'y') {
                  cout << "Enter the weight of the edge: ";
                  cin >> graph[i][j];
                  graph[j][i] = graph[i][j];
               } else {
```

```
                graph[i][j] = graph[j][i] = INFINITY;
            }
          } else {
            graph[i][j] = 0;
          }
        }
        c2++;
      }
      c2 = 'a';
      c1++;
    }
}

int minDistance(int* dist, bool* sptSet, int nvert) {
   int minDist = INFINITY;
   int minIndex = -1;

   for (int v = 0; v < nvert; v++) {
      if (!sptSet[v] && dist[v] < minDist) {
         minDist = dist[v];
         minIndex = v;
      }
   }

   return minIndex;
}

void printSolution(int* dist, int nvert) {
   cout << "Vertex   Distance from Source" << endl;
   for (int i = 0; i < nvert; i++) {
      cout << i << "       " << dist[i] << endl;
   }
}

void Dijkstra(int** graph, int nvert) {
```

```
    int* dist = new int[nvert];
    bool* sptSet = new bool[nvert];

    for (int i = 0; i < nvert; i++) {
        dist[i] = INFINITY;
        sptSet[i] = false;
    }

    dist[0] = 0;

    for (int count = 0; count < nvert - 1; count++) {
        int u = minDistance(dist, sptSet, nvert);
        sptSet[u] = true;

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

    printSolution(dist, nvert);

    delete[] dist;
    delete[] sptSet;
}

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

    int** graph = new int*[n];
    for (int i = 0; i < n; ++i) {
```

```
        graph[i] = new int[n];
    }

    setWeight(graph, n);
    Dijkstra(graph, n);

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

    return 0;
}
```

16. Program for generating minimum spanning trees (Kruskal's Algorithm).

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

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

int findParent(int parent[], int vertex) {
    if (parent[vertex] == -1)
        return vertex;
    return findParent(parent, parent[vertex]);
}

void unionSets(int parent[], int x, int y) {
    int xSet = findParent(parent, x);
    int ySet = findParent(parent, y);
    parent[xSet] = ySet;
}
```

```
void sortEdgesByWeight(Edge edges[], int E) {
   for (int i = 0; i < E - 1; i++) {
      for (int j = 0; j < E - i - 1; j++) {
         if (edges[j].weight > edges[j + 1].weight) {
            Edge temp = edges[j];
            edges[j] = edges[j + 1];
            edges[j + 1] = temp;
         }
      }
   }
}

void kruskalMST(Edge edges[], int V, int E) {
   Edge result[V - 1];
   int edgeCount = 0;

   sortEdgesByWeight(edges, E);

   int parent[V];
   for (int i = 0; i < V; i++)
      parent[i] = -1;

   for (int i = 0; i < E; i++) {
      int srcParent = findParent(parent, edges[i].src);
      int destParent = findParent(parent, edges[i].dest);

      if (srcParent != destParent) {
         result[edgeCount++] = edges[i];
         unionSets(parent, srcParent, destParent);

         if (edgeCount == V - 1)
            break;
      }
   }
```

```
    cout << "Minimum Spanning Tree:\n";
    for (int i = 0; i < V - 1; i++) {
        cout << result[i].src << " - " << result[i].dest << " : " << result[i].weight <<
endl;
    }
}

int main() {
    int V, E;
    cout << "Enter the number of vertices and edges: ";
    cin >> V >> E;

    Edge edges[E];
    for (int i = 0; i < E; i++) {
        cout << "Enter edge " << i + 1 << " (src dest weight): ";
        cin >> edges[i].src >> edges[i].dest >> edges[i].weight;
    }

    kruskalMST(edges, V, E);

    return 0;
}
```