Practical-8

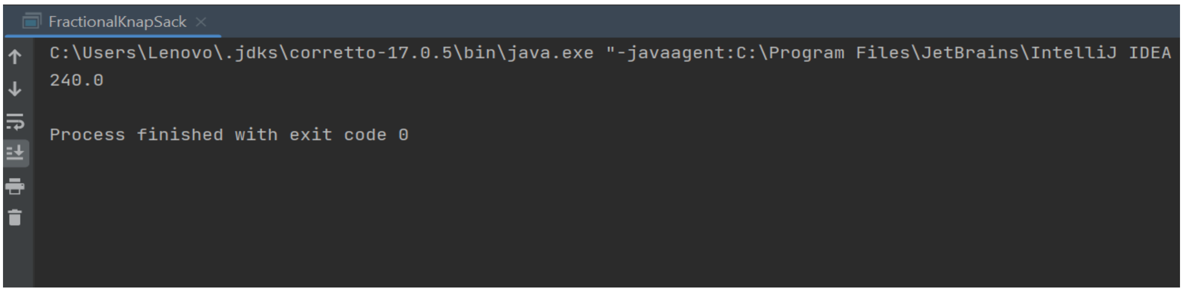
Aim: Write a program to implement the knapsack problem using greedy algorithm.

Code:

```
  import java.io.*;

  import java.util.Arrays;

  import java.util.Comparator;

  class FractionalKnapSack {

      private static double getMaxValue(ItemValue[] arr, int capacity) {

              Arrays.sort(arr, new Comparator<ItemValue>() {

                public int compare(ItemValue item1,

                       ItemValue item2)

               {

              double cpr = new Double((double)item1.value

                                                               / (double)item1.weight);

          double cpr2= new Double((double)item2.value

                                                               / (double)item2.weight);

              if (cpr1 < cpr2)

               return 1;

          else

               return -1;

    }

});

double totalValue = 0d;

for (ItemValue i : arr) {

    int curWt = (int)i.weight;

    int curVal = (int)i.value;

    if (capacity - curWt >= 0) {

        capacity = capacity - curWt;

        totalValue += curVal;

    }

    else {
```

```
                        double fraction = ((double)capacity / (double)curWt);
                        totalValue += (curVal * fraction);
                        capacity=  (int)(capacity - (curWt * fraction));
                        break;
            }
        }
        return totalValue;
    }
    static class ItemValue {
        int value, weight;
        public ItemValue(int val, int wt)
        {
            this.weight = wt;
            this.value = val;
        }
    }
    public static void main(String[] args) {
        ItemValue[] arr = { new ItemValue(60, 10),
        new ItemValue(100, 20),
        new ItemValue(120, 30) };
        int capacity = 50;
        double maxValue = getMaxValue(arr, capacity);
        System.out.println(maxValue);
    }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
240.0

Process finished with exit code 0
```

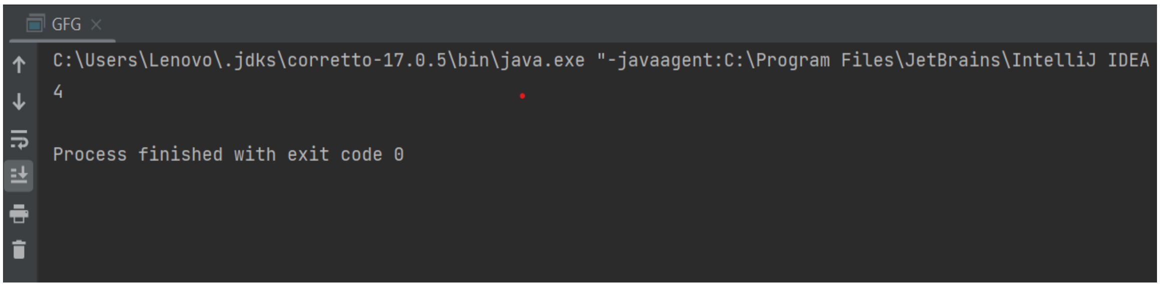
Practical-9

Aim: Write a program to implement making change problem using dynamic programming.

Code:

```
import java.util.*;
class GFG {
 static int count(int coins[], int n, int sum)
 {

        // If sum is 0 then there is 1 solution
        // (do not include any coin)
        if (sum == 0)
               return 1;

        // If sum is less than 0 then no
        // solution exists
        if (sum < 0)
               return 0;

        // If there are no coins and sum
        // is greater than 0, then no
        // solution exist
        if (n <= 0)
               return 0;

        // count is sum of solutions (i)
        // including coins[n-1] (ii) excluding coins[n-1]
        return count(coins, n - 1, sum)+ count(coins, n, sum - coins[n - 1]);
 }
 public static void main(String args[])
 {
        int coins[] = { 1, 2, 3 };
        int n = coins.length;
        System.out.println(count(coins, n, 4));
 }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
4

Process finished with exit code 0
```

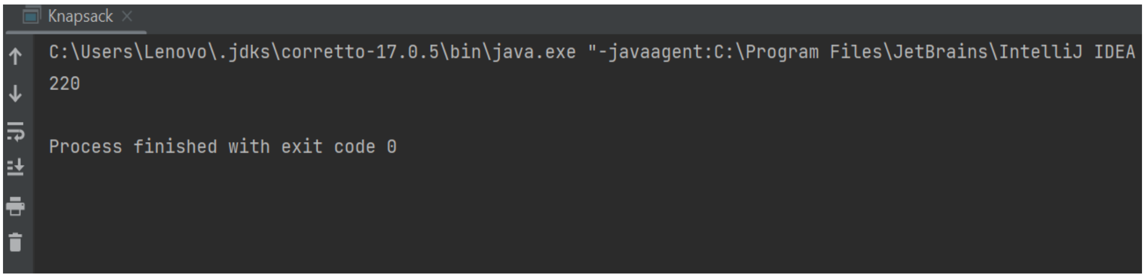
Practical-10

Aim: Write a program to implement the knapsack problem using dynamic programming

Code:

```
class Knapsack {
    static int max(int a, int b)
    {
        return (a > b) ? a : b;
    }
    static int knapSack(int W, int wt[], int val[], int n)
    {
        int i, w;
        int K[][] = new int[n + 1][W + 1];
        for (i = 0; i <= n; i++)
        {
            for (w = 0; w <= W; w++)
            {
                if (i == 0 || w == 0)
                    K[i][w] = 0;
                else if (wt[i - 1] <= w)
                    K[i][w]
                            = max(val[i - 1]
                                    + K[i - 1][w - wt[i - 1]],
                            K[i - 1][w]);
                else
                    K[i][w] = K[i - 1][w];
            }
        }
        return K[n][W];
    }
    public static void main(String args[])
    {
        int val[] = new int[] { 60, 100, 120 };
        int wt[] = new int[] { 10, 20, 30 };
        int W = 50;
        int n = val.length;
        System.out.println(knapSack(W, wt, val, n));
    }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
220

Process finished with exit code 0
```

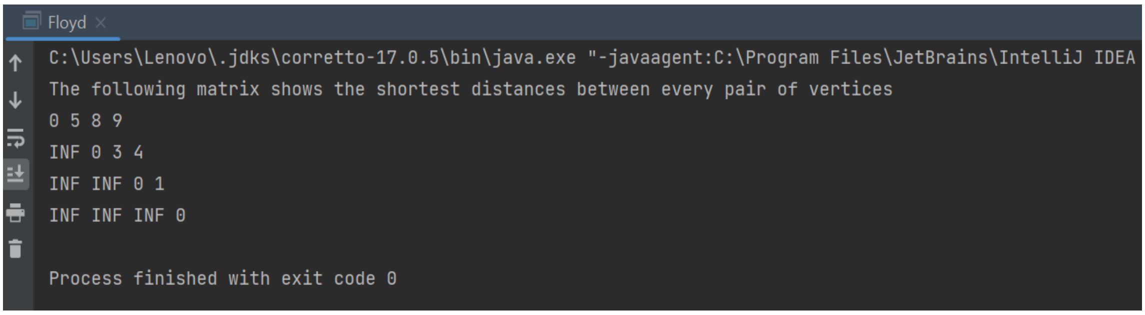
Practical-11

Aim: Write a program to implement Floyd’s algorithm for finding shortest path using dynamic programming.

Code:

```
import java.io.*;
import java.lang.*;
import java.util.*;
class Floyd {
    final static int INF = 99999, V = 4;
    void floydWarshall(int graph[][])
    {
        int dist[][] = new int[V][V];
        int i, j, k;
        for (i = 0; i < V; i++)
            for (j = 0; j < V; j++)
                dist[i][j] = graph[i][j];

        for (k = 0; k < V; k++) {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++) {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++) {
                    // If vertex k is on the shortest path
                    // from i to j, then update the value of
                    // dist[i][j]
                    if (dist[i][k] + dist[k][j]
                            < dist[i][j])
                        dist[i][j]
                                = dist[i][k] + dist[k][j];
                }
            }
        }
        printSolution(dist);
    }
    void printSolution(int dist[][])
    {
        System.out.println(
                "The following matrix shows the shortest "
                        + "distances between every pair of vertices");
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                if (dist[i][j] == INF)
                    System.out.print("INF ");
                else
                    System.out.print(dist[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args)
    {
        int graph[][] = { { 0, 5, INF, 10 },
                { INF, 0, 3, INF },
                { INF, INF, 0, 1 },
                { INF, INF, INF, 0 } };
        Floyd a = new Floyd();
        a.floydWarshall(graph);
    }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
The following matrix shows the shortest distances between every pair of vertices
0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0

Process finished with exit code 0
```

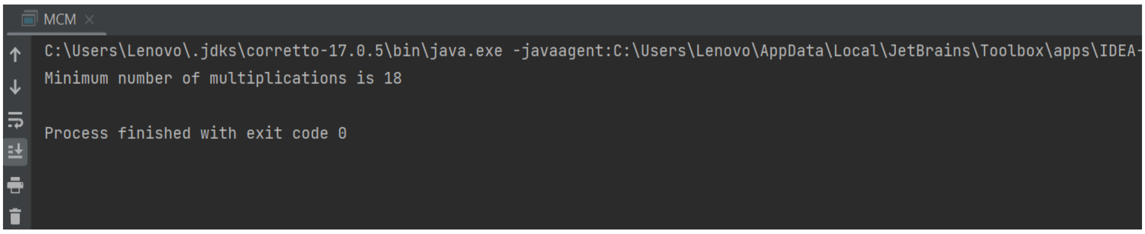
Practical-12

Aim: Write a program to implement chained matrix multiplication using dynamic programming.

Code:

```
    class MCM {
    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int p[], int n)
    {
        int m[][] = new int[n][n];
        int i, j, k, L, q;
        for (i = 1; i < n; i++)
            m[i][i] = 0;
        // L is chain length.
        for (L = 2; L < n; L++) {
            for (i = 1; i < n - L + 1; i++) {
                j = i + L - 1;
                if (j == n)
                    continue;
                m[i][j] = Integer.MAX_VALUE;
                for (k = i; k <= j - 1; k++) {
                    // q = cost/scalar multiplications
                    q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                    if (q < m[i][j])
                        m[i][j] = q;
                }
            }
        }
        return m[1][n - 1];
    }
    public static void main(String args[])
    {
        int arr[] = new int[] { 1, 2, 3, 4 };
        int size = arr.length;

        System.out.println("Minimum number of multiplications is "
                + MatrixChainOrder(arr, size));
    }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe -javaagent:C:\Users\Lenovo\AppData\Local\JetBrains\Toolbox\apps\IDEA-
Minimum number of multiplications is 18

Process finished with exit code 0
```

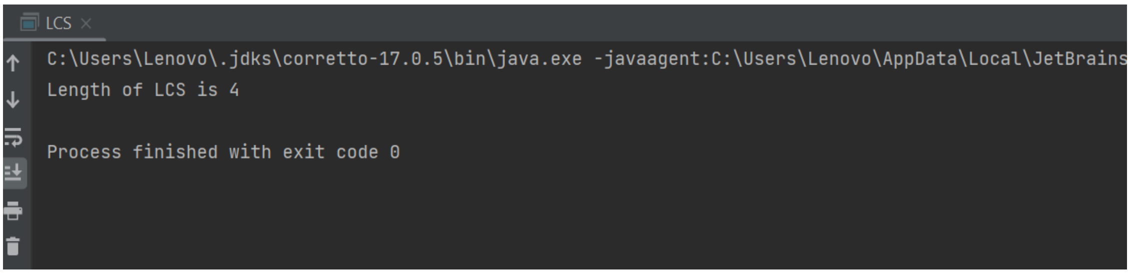
Practical-13

Aim: Write a program to implement longest common subsequence using dynamic programming.

Code:

```
 public class LCS
{
          int lcs(char[] X, char[] Y, int m, int n){
                    int L[][] = new int[m + 1][n + 1];
                    /* Following steps build L[m+1][n+1] in bottom up fashion. Note
                    that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
                    for (int i = 0; i <= m; i++) {
                              for (int j = 0; j <= n; j++) {
                                        if (i == 0 || j == 0)
                                                  L[i][j] = 0;
                                        else if (X[i - 1] == Y[j - 1])
                                                   L[i][j] = L[i - 1][j - 1] + 1;
                                         else
                                                   L[i][j] = max(L[i - 1][j], L[i][j - 1]);
                              }
                    }
                     return L[m][n];
          }
          int max(int a, int b)
          {
                     return (a > b) ? a : b;
          }
  public static void main(String[] args) {
           LCS lcs = new LCS();
           String s1 = "AGGTAB";
          String s2 = "GXTXAYB";
          char[] X = s1.toCharArray();
          char[] Y = s2.toCharArray();
          int m = X.length;
          int n = Y.length;
          System.out.println("Length of LCS is"+ " " + lcs.lcs(X, Y, m, n));
  }
}
```

Output:



```
C:\Users\Lenovo\.jdks\corretto-17.0.5\bin\java.exe -javaagent:C:\Users\Lenovo\AppData\Local\JetBrains
Length of LCS is 4

Process finished with exit code 0
```