```java
import java.util.*;

/**
 * File: Algorithm.java
 * Description:
 *      - The Investment Problem's Algorithm (Greedy Algorithm)
 *
 * Project:     Assignment 01 - ex2 Investment Problem
 * Class:       CS353
 *
 * @author Dr. Lei Wang [Code Template Author]
 * @author Kong Jimmy Vang [Completed Code Template]
 */
public class Algorithm {

    // solve investment problem
    /*
     * inputs:
     *   m - the total amount of money to be distributed to projects
     *   benefits - benefit function of each task
     * output:
     *   Money distribution on each project that can maximize the total benefit
     */
    public int[] investment(int m, int[][] benefits)
    {
        int[] res = new int[benefits.length];

        //**** Investment Algorithm ****//
        int n = benefits.length; // n = number of projects
        int size = m + n - 1;    // size of int array

        // Find all possible ints with 5 bits.
        ArrayList<Long> longs = new ArrayList<>();
        for (long i = 0; i < Long.SIZE * size; i++)
        {
            long bits = 0;
            for (long j = 0; j < size; j++)
            {
                bits += (i >> j) & 1;
            }

            if (bits == m)
            {
                longs.add(i);
            }
        }

        // Find the max benefit from each investment.
        int maxBenefit = Integer.MIN_VALUE;
        for (Long l : longs)
        {
            int[] investment = new int[benefits.length];
            int benefit = 0;

            // Calculate investment and set up investment array.
            int bits = 0;
            long bit = 0;
            int k = 0;
            for (int j = 0; j <= size; j++)
            {
                bit = (l >> j) & 1;

                if (bit == 1)
                {
                    bits++;
                }
                else
                {
                    if (k < benefits.length) {
                        investment[k++] = bits;
                        bits = 0;
                    }
                }
            }

            // Check if the current investment is better than the previous max benefit.
            for (int j = 0; j < investment.length; j++)
            {
                if (investment[j] < benefits[j].length) {
                    benefit += benefits[j][investment[j]];
                }
            }

            // Replace max benefit if the current investment provides better benefits.
            if (maxBenefit < benefit)
            {
                maxBenefit = benefit;
                res = investment;
            }
        }

        // Return the max benefit.
        return res;
    }
}

```