Recursividad III (2015-2016)

A - Tarea

Problema A

```
using System;

namespace RecursividadIII
{
    class A
    {
        static bool Tarea(int n, int[] array, int k)
        {
            if(k > n) return true;

            for(int i = 0; i + k + 1 < 2 * n; i++)
            {
                if(array[i] == 0 && array[i + k + 1] == 0)
                {
                    array[i] = array[i + k + 1] = k;
                    if(Tarea(n, array, k+1)) return true;
                    array[i] = array[i + k + 1] = 0;
                }
            }
            return false;
        }

        static void Main()
        {
            int N = int.Parse(Console.ReadLine());
            int[] array = new int[2*N];

            if(Tarea(N, array, 1))
            {
                for(int i = 0; i < 2 * N; i++)
                {
                    if(i > 0) Console.Write(" ");
                    Console.Write(array[i]);
                }
            }
            else
                Console.Write("NO ES POSIBLE");
        }
    }
}
```

B - Problema de espacio

Este problema es directamente la misma idea del Problema de la Mochila

Problema B

```
using System;

namespace RecursividadIII
{
    public class B
    {
        public static int MayorUtilidad(int n, int[] t, int[] u)
        {
            return MayorUtilidad(n, t, u, 0, 0, 0);
        }

        public static int MayorUtilidad(int n, int[] t, int[] u,
                                         int idx, int tAct, int uAct)
        {
            if(idx == t.Length)
                return uAct;

            int copiar = 0;
            int noCopiar = 0;

            if(tAct + t[idx] <= n)
                copiar = MayorUtilidad(n, t, u, idx + 1, tAct + t[idx], uAct + u[idx]);

            noCopiar = MayorUtilidad(n, t, u, idx + 1, tAct, uAct);

            return Math.Max(copiar, noCopiar);
        }

        public static void Main()
        {
            string[] linea = Console.ReadLine().Split();
            int n = int.Parse(linea[0]);
            int m = int.Parse(linea[1]);

            int[] t = new int[m];
            int[] u = new int[m];

            for(int i = 0; i < m; i++)
            {
                linea = Console.ReadLine().Split();
                t[i] = int.Parse(linea[0]);
                u[i] = int.Parse(linea[1]);
            }

            Console.WriteLine(MayorUtilidad(n, t, u));
        }
    }
}
```

C - Partición Balanceada I

Para cada número se decide si se pone en el conjunto A o en el B y se escoge la variante que tiene menos factor de balance

Problema C

```
using System;

namespace RecursividadIII
{
    public class C
    {
        public static int MinFactorBalance(int[] n)
        {
            bool[] marcas = new bool[n.Length];
            return MinFactorBalance(n, marcas, 0);
        }

        public static int MinFactorBalance(int[] n, bool[] marcas, int idx)
        {
            if(idx == n.Length)
                return FactorBalance(n, marcas);

            marcas[idx] = true;
            int a = MinFactorBalance(n, marcas, idx + 1);
            marcas[idx] = false;
            int b = MinFactorBalance(n , marcas, idx +1);

            return Math.Min(a, b);
        }

        public static int FactorBalance(int[] n, bool[] marcas)
        {
            int sa = 0;
            int sb = 0;
            for(int i = 0; i < n.Length; i++)
            {
                if(marcas[i]) sa += n[i];
                else sb += n[i];
            }

            return Math.Abs(sa - sb);
        }
        public static void Main()
        {
            string[] linea = Console.ReadLine().Split();
            int[] n = new int[linea.Length];

            for(int i = 0; i < n.Length; i++)
                n[i] = int.Parse(linea[i]);

            Console.WriteLine(MinFactorBalance(n));
        }
    }
}
```

D - Partición Balanceada II

Misma idea que el ejercicio anterior solo que la cantidad de elementos que puede ir en A es una cantidad determinada.

Problema D

```
using System;

namespace RecursividadIII
{
    public class D
    {
        public static int MinFactorBalanceII(int[] n)
        {
            bool[] marcas = new bool[n.Length];
            return MinFactorBalanceII(n, marcas, 0, 0);
        }

        public static int MinFactorBalanceII(int[] n, bool[] marcas, int idx, int cant)
        {
            if(cant == n.Length / 2)
                return FactorBalance(n, marcas);

            if(idx == n.Length)
                return int.MaxValue;

            int a = int.MaxValue;
            if(cant + 1 <= n.Length / 2)
            {
                marcas[idx] = true;
                a = MinFactorBalanceII(n, marcas, idx + 1, cant + 1);
            }
            marcas[idx] = false;
            int b = MinFactorBalanceII(n , marcas, idx + 1, cant);

            return Math.Min(a, b);
        }

        public static int FactorBalance(int[] n, bool[] marcas)
        {
            int sa = 0;
            int sb = 0;
            for(int i = 0; i < n.Length; i++)
            {
                if(marcas[i]) sa += n[i];
                else sb += n[i];
            }

            return Math.Abs(sa - sb);
        }
        public static void Main()
        {
            string[] linea = Console.ReadLine().Split();
            int[] n = new int[linea.Length];

            for(int i = 0; i < n.Length; i++)
                n[i] = int.Parse(linea[i]);

            Console.WriteLine(MinFactorBalanceII(n));
        }
    }
}
```

E - Cambio de Monedas II

Problema E

```
using System;

namespace RecursividadIII
{
    public class E
    {
        public static int[] CicloGanancia(int m, double[,] t)
        {
            int[] ciclo = new int[m + 1];

            for (int i = 0; i <= m; i++) ciclo[i] = -1;

            if (CicloGanancia(t, new bool[m], ciclo, 0))
                return ciclo;

            return null;
        }

        public static bool CicloGanancia(double[,] t, bool[] usado,
                                         int[] ciclo, int idxC)
        {
            if (idxC > 1)
            {
                ciclo[idxC] = ciclo[0];
                if (Cambiar(t, ciclo, idxC + 1) > 1)
                    return true;
                ciclo[idxC] = -1;
            }

            for (int i = 0; i < t.GetLength(0); i++)
            {
                if (!usado[i])
                {
                    usado[i] = true;
                    ciclo[idxC] = i;
                    if (CicloGanancia(t, usado, ciclo, idxC + 1))
                        return true;
                    usado[i] = false;
                    ciclo[idxC] = -1;
                }
            }

            return false;
        }

        public static double Cambiar(double[,] t, int[] ciclo, int length)
        {
            double dinero = 1;

            for (int i = 0; i < length - 1; i++)
                dinero *= t[ciclo[i], ciclo[i + 1]];

            return dinero;
        }

        public static void Main()
        {
            int m = int.Parse(Console.ReadLine());

            double[,] t = new double[m, m];

            for (int i = 0; i < m; i++)
            {
                string[] linea = Console.ReadLine().Split();

                for (int j = 0; j < m; j++)
                    t[i, j] = double.Parse(linea[j]);
            }

            int[] ciclo = CicloGanancia(m, t);
```

```
            if (ciclo == null)
                Console.WriteLine("NO HAY GANANCIA");
            else
            {
                for (int i = 0; ciclo[i] != -1; i++)
                {
                    if (i > 0) Console.Write(" ");
                    Console.Write(ciclo[i]);
                }
            }
        }
    }
}
```

F - Salto de Caballo

Problema F

```
using System;

namespace RecursividadIII
{
    public class F
    {
        static int[] df = { 2, 1, -1, -2, -2, -1, 1, 2 };
        static int[] dc = { 1, 2, 2, 1, -1, -2, -2, -1 };

        static int[,] SaltoCaballo(int n, int f, int c)
        {
            int[,] ans = new int[n, n];
            bool[,] visitado = new bool[n, n];
            visitado[f, c] = true;
            if (SaltoCaballo(n, f, c, ans, visitado, 1)) return ans;
            return null;
        }

        static bool SaltoCaballo(int n, int f, int c, int[,] res,
                                 bool[,] visitado, int cur)
        {
            if (cur == n * n) return true;
            for (int i = 0; i < 8; i++)
            {
                int nf = f + df[i];
                int nc = c + dc[i];
                if (EsValida(n, nf, nc) && !visitado[nf, nc])
                {
                    visitado[nf, nc] = true;
                    res[nf, nc] = cur;
                    if (SaltoCaballo(n, nf, nc, res, visitado, cur + 1)) return true;
                    visitado[nf, nc] = false;
                    res[nf, nc] = -1;
                }
            }
            return false;
        }

        static bool EsValida(int n, int f, int c)
        {
            return f >= 0 && c >= 0 && c < n && f < n;
        }

        public static void Main()
        {
            string[] linea = Console.ReadLine().Split();
            int n = int.Parse(linea[0]);
            int f = int.Parse(linea[1]);
            int c = int.Parse(linea[2]);

            int[,] res = SaltoCaballo(n, f, c);

            if (res == null) Console.WriteLine("NO ES POSIBLE");
            else
            {
                for (int i = 0; i < n; i++)
                {
                    if (i > 0) Console.WriteLine();
                    for (int j = 0; j < n; j++)
                    {
                        if (j > 0) Console.Write(" ");
                        Console.Write(res[i, j]);
                    }
                }
            }

        }
    }
}
```