



Universitat
de les Illes Balears

TRABAJO DE FIN DE GRADO

DESARROLLO DE ALGORITMOS EFICIENTES

Marcos López Garau

Grado de ingeniería Electrónica, Industrial y Automática

Escuela Politécnica Superior

Año académico 2022-23



DESARROLLO DE ALGORITMOS EFICIENTES

Marcos López Garau

Trabajo de Fin de Grado

Escuela Politécnica Superior

Universidad de las Illes Balears

Año académico 2022-23

Palabras clave del trabajo:

Programación Informática, concurso, pensamiento computacional, coste computacional, competición, visual studio, C++, Java, Python, algoritmos.

Trabajo tutelado por: *Dr. José María Bades Rubio.*

Departamento de Matemáticas e informática.

Autorizo la Universidad a incluir este trabajo en el Repositorio Institucional para consultarlo en acceso abierto y difundirlo en línea, con finalidades exclusivamente académicas y de investigación

Autor/a		Tutor/a	
Sí	No	Sí	No
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>



AGRADECIMIENTOS

A mi familia por apoyarme en este camino y darme mi espacio para poder organizarme y acabar esta carrera de la mayor forma posible.

A mis amigos por estar allí siempre que los necesitaba y a mí mismo por ser capaz de llevar adelante toda la carrera sin ningún problema.



ÍNDICE

<i>CAPÍTULO 1: Introducción</i>	1
1.1 Motivación del Trabajo de Fin de Grado	1
1.2 Objetivo del Trabajo de Fin de Grado	4
1.3 Estructura del Trabajo de Fin de Grado	4
<i>CAPÍTULO 2: Programación competitiva</i>	5
2.1 ¿Qué es la programación competitiva?	5
2.2 Competiciones Internacionales	6
2.2.1 Competición Internacional Universitaria de Programación	6
2.2.1.1 Historia de la ICPC	6
2.2.1.2 Reglamento de la ICPC	7
2.2.2 Olimpiada Internacional de Informática	9
2.2.2.1 Historia de la IOI	10
2.2.2.2 Reglamento de la IOI	12
2.2.3 Meta Hacker Cup	13
2.2.3.1 Historia y reglamento de la Meta Hacker Cup	13
2.2.3.2 Estructura del concurso	14
2.2.3.3 Método de puntuación	15
2.3 Competiciones nacionales	15
2.3.1 Concurso universitario AdaByron	16
2.3.1.1 Historia de AdaByron	16
2.3.1.2 Reglamento de AdaByron	17
2.4 Zonas de entrenamiento	19
2.4.1 CodeForces	20
2.4.2 ¡Acepta el reto!	21
<i>CAPÍTULO 3: Coste computacional</i>	24
3.1 ¿Qué es el coste computacional?	24
3.1.1 Tipos de notación “big-O”	25
3.1.1.1 Notación $O(1)$	25
3.1.1.2 Notación $O(n)$	25
3.1.1.3 Notación $O(\log n)$	26



3.1.1.4 Notación $O(n \log n)$	26
3.1.1.5 Notación $O(n^2)$	26
3.1.1.6 Notación $O(2^n)$	26
3.1.1.7 Notación $O(n!)$	26
3.1.2 Algoritmos	26
3.1.2.1 Recursividad	27
3.1.2.2 Búsqueda binaria	28
3.1.2.3 MergeSort	28
3.1.2.5 Divide y vencerás	29
CAPÍTULO 4: Lista problemas realizados	30
4.1 Problemas de dificultad baja	30
4.1.1 Problema 116: ¡Hola mundo!	30
4.1.2 Problema 114: Último dígito del factorial	30
4.1.3 Problema 380: ¡Me caso!	31
4.1.4 Problema 245: Los Dalton	31
4.1.5 Problema 216: Goteras	31
4.1.6 Problema 117: La fiesta aburrida	31
4.1.7 Problema 184: El sueño de los concursantes	31
4.1.8 Problema 506: Tensión descompensada	32
4.1.9 Problema 512: Döner sospechoso	32
4.1.10 Problema 616: Pic,Poc,Pic...Pong!	32
4.1.11 Problema 217: ¿Qué lado de la calle?	32
4.1.12 Problema 397: ¿Es múltiplo de 3?	32
4.1.13 Problema 355: Gregorio XIII	33
4.1.14 Problema 219: La lotería de la peña Atlética	33
4.1.15 Problema 172: El pan en las bodas	33
4.1.16 Problema 112: Radares de tramo	33
4.2 Problemas de dificultad intermedia	34
4.2.1 Problema 100: Constante de Kaprekar	34
4.2.2 Problema 234: Carreras de coches	34
4.2.3 Problema 337: La abuela María	34
4.2.4 Problema 381: Alineación planetaria	34



4.2.5 Problema 382: Internet en el metro	35
4.2.6 Problema 383: El alcance de las historias	35
4.2.7 Problema 242: Erasmús	35
4.2.8 Problema 610: ¡No quiero irme señor Stark!	35
4.2.9 Problema 607: Minimizando el castigo	36
4.2.10 Problema 407: Rebotando en el parchís	36
4.2.11 Problema 325: Helados de cucurucho	36
4.2.12 Problema 405: Imprimiendo páginas sueltas	36
4.2.13 Problema 197: Mensaje interceptado	36
4.2.14 Problema 195: Saltos de trampolín	37
4.2.15 Problema 608: Peligro por hielo	37
4.2.16 Problema 604: Honor y distribución	37
4.2.17 Problema 108: De nuevo en el bar de Javier	37
4.2.18 Problema 354: Los niños primero	37
4.2.19 Problema 205: Números de Lychrel	38
4.3.20 Problema 109: Liga de pádel	38
4.3 Problemas de dificultad alta	38
4.3.1 Problema 295: Elévame	38
4.3.2 Problema 262: Ada, Babbage y Bernoulli	38
4.3.3 Problema 212: Operación asfalto	38
4.3.4 Problema 343: 7 de golpe	39
4.3.5 Problema 189: Embarque de un transatlántico	39
4.3.6 Problema 139: Números cubifinitos	39
4.3.7 Problema 230: Desórdenes temporales	40
4.3.8 Problema 386: Clúster de microondas	40
4.3.9 Problema 342: ¡No lo puedes saber!	40
4.3.10 Problema 324: Teorema del punto fijo	40
4.3.11 Problema 598: Comienza la temporada	41
4.3.12 Problema 319: La máquina calculadora	41
4.3.13 Problema 285: Las vacas pensantes	41
4.3.14 Problema 220: ¡Pasa la calculadora!	41



4.3.15 Problema 384: El juego de la linterna	42
4.3.16 Problema 326: La ardilla viajera.....	42
CAPÍTULO 5: Solución a los problemas realizados	42
5.1 Solución a los problemas de dificultad baja.....	43
5.1.1 Problema 116: ¡Hola mundo!.....	43
5.1.2 Problema 114: Último dígito del factorial.....	43
5.1.3 Problema 380: ¡Me caso!	43
5.1.4 Problema 245: Los Dalton	43
5.1.5 Problema 216: Goteras	44
5.1.6 Problema 117: La fiesta aburrida	44
5.1.7 Problema 184: El sueño de los concursantes.....	44
5.1.8 Problema 506: Tensión descompensada	44
5.1.9 Problema 512: Döner sospechoso	44
5.1.10 Problema 616: Pic,Poc,Pic...Pong!.....	45
5.1.11 Problema 217: ¿Qué lado de la calle?.....	45
5.1.12 Problema 397: ¿Es múltiplo de 3?.....	45
5.1.13 Problema 355: Gregorio XIII.....	45
5.1.14 Problema 219: La lotería de la peña Atlético	45
5.1.15 Problema 172: El pan en las bodas	45
5.1.16 Problema 112: Radares de tramo	46
5.2 Solución a los problemas de dificultad intermedia.....	46
5.2.1 Problema 100: Constante de Kaprekar	46
5.2.2 Problema 234: Carreras de coches	47
5.2.3 Problema 337: La abuela María	47
5.2.4 Problema 381: Alineación planetaria.....	47
5.2.5 Problema 382: Internet en el metro	47
5.2.6 Problema 383: El alcance de las historias	48
5.2.7 Problema 242: Erasmo	48
5.2.8 Problema 610: ¡No quiero irme señor Stark!.....	48
5.2.9 Problema 607: Minimizando el castigo	49
5.2.10 Problema 407: Rebotando en el parchís.....	49



5.2.11 Problema 325: Helados de cucurucho	49
5.2.12 Problema 405: Imprimiendo páginas sueltas.....	49
5.2.13 Problema 197: Mensaje interceptado	50
5.2.14 Problema 195: Saltos de trampolín.....	50
5.2.15 Problema 608: Peligro por hielo	50
5.2.16 Problema 604: Honor y distribución	50
5.2.17 Problema 108: De nuevo en el bar de Javier	50
5.2.18 Problema 354: Los niños primero	51
5.2.19 Problema 205: Números de Lychrel.....	51
5.3.20 Problema 109: Liga de pádel.....	51
5.3 Solución a los problemas de dificultad alta	52
5.3.1 Problema 295: Elévame	52
5.3.2 Problema 262: Ada, Babbage y Bernoulli.....	52
5.3.3 Problema 212: Operación asfalto	52
5.3.4 Problema 343: 7 de golpe	52
5.3.5 Problema 189: Embarque de un transatlántico	53
5.3.6 Problema 139: Números cubifinitos	53
5.3.7 Problema 230: Desórdenes temporales.....	53
5.3.8 Problema 386: Clúster de microondas.....	54
5.3.9 Problema 342: ¡No lo puedes saber!	54
5.3.10 Problema 324: Teorema del punto fijo	54
5.3.11 Problema 598: Comienza la temporada.....	55
5.3.12 Problema 319: La máquina calculadora	55
5.3.13 Problema 285: Las vacas pensantes.....	55
5.3.14 Problema 220: ¡Pasa la calculadora!	55
5.3.15 Problema 384: El juego de la linterna	56
5.3.16 Problema 326: La ardilla viajera.....	56
CAPÍTULO 6: Conclusión.....	58
BIBLIOGRAFÍA.....	60



ÍNDICE DE FIGURAS

Figura 1: Solución al primer ejemplo de tipo de algoritmos.	2
Figura 2: Solución al segundo ejemplo de tipo de algoritmos.....	3
Figura 3: Ganadores de la ICPC'22	7
Figura 4: Ceremonia IOI'22	10
Figura 5: Algunos de los finalistas de la Meta Hacker Cup 2018	15
Figura 6: Edición AdaByron 2022	17
Figura 7: Página principal CodeForces.....	20
Figura 8: Ranking CodeForces	21
Figura 9: Esquema general de un problema.....	22
Figura 10: Estadísticas de un problema.	23
Figura 11: Categorías y subcategorías de ¡Acepta el reto!	23
Figura 12: Diferentes valores de Big-O	25
Figura 13: Ejemplo recursividad.....	27
Figura 14: Ejemplo algoritmo búsqueda binaria.....	28
Figura 15: Ejemplo algoritmo MergeSort.....	29

ACRÓNIMOS

Pascal	Lenguaje de programación estructurado publicado en 1970.
Kotlin	Lenguaje de programación de alto nivel y tipado estático creado el 2016.
Ada	Lenguaje de programación estático y fuertemente tipado creado el 1983.
C	Lenguaje de programación estructurado creado a mediados de los años 70.
C++	Lenguaje de programación híbrido diseñado en 1979.
Java	Lenguaje de programación y plataforma informática desde 1995.
Python	Lenguaje de programación de alto nivel creado en 1991.
VS Code	Visual Studio Code. Editor de código fuente desde 2015.
ICPC	International Collegiate Programming Contest.
ACM	Association for Computing Machinery.
ACM-ICPC	Nombre dado a la ICPC cuando contaba con los auspicios de la ACM.
IBM	International Business Machines Corporation.
MIT	Massachusetts Institute of Technology.
SWERC	Southwestern Europe Regional Contest.
CE	Compilation error/Error de compilación.
RTE	Running Time Error/Error de tiempo de ejecución.
LTE	Limit Time Exceeded/Límite de tiempo excedido.
WA	Wrong Answer/Respuesta Incorrecta.
AC	Accepted/Aceptado.
PE	Presentation error/Error de presentación.
MLE	Memory limit exceeded/Límite de memoria superado.
OLE	Output limit exceeded/Límite de salida superado.
RF	Restricted function/Función restringida.
IQ	In queue/En cola.
IE	Internal error/Error interno.
IOI	International Olympiad in Informatics/Olimpiada Inter. de Informática.
UNESCO	Organización de las Naciones Unidas para la Educación, Ciencia y Cultura.
IFIP	Federación Internacional de Procesamiento de la Información.
OIE	Olimpiada Informática Española.



ACRÓNIMOS

UIB	Universidad de las Islas Baleares.
CUPCAM	Concurso Universitario de Programación de la Comunidad de Madrid.
USD	Dólar estadounidense.
UCM	Universidad Complutense de Madrid.

RESUMEN

El siguiente trabajo tiene como objetivo representar los distintos pasos a realizar a la hora de adentrarse en el mundo del desarrollo de algoritmos eficientes partiendo desde el nivel 0. Para ello hay que conocer la programación competitiva, una rama de la programación donde los problemas deben ser eficientes.

La programación competitiva consiste en un torneo entre programadores. Estos deben realizar distintos problemas que se ejecuten en un límite de tiempo, además de cumplir con un límite de memoria. Los problemas contienen una descripción y uno o varios ejemplos de datos de entrada y su respectiva salida.

Para poder hacer todo tipo de problemas de programación, primero se ha aprendido sobre el coste computacional y la algoritmia para hacer que estos sean eficientes. Después se han realizado más de cincuenta problemas para poner en práctica lo aprendido. Cada problema es un reto distinto y hay unos fáciles y otros más difíciles.

A medida que se hacían los problemas nos encontrábamos con algoritmos nuevos a incorporar, por tanto, siempre se tenía que investigar sobre ellos al tiempo que se probaban problemas nuevos.

A la hora de hacer los problemas se ha usado uno de los lenguajes más conocidos: el **C++**, pero se pueden usar lenguajes como **Java** o **Python**. Además, se utiliza el **VS Code** como editor de código fuente.

Para concluir se explica la importancia que tienen estos problemas independientemente de la competición. En todas las empresas de tecnología como Google o Amazon realizan pruebas de programación en sus entrevistas de trabajo. Así que, aunque no quieras apuntarte a una competición, deberías practicar este tipo de problemas si tu objetivo es trabajar en una empresa tecnológica.

CAPÍTULO 1: Introducción

En este capítulo se presenta la motivación básica sobre el tema de la programación competitiva, a la vez que se describen los objetivos del trabajo de fin de grado. Finalmente, se realizará un breve resumen de los capítulos que componen este trabajo.

1.1 Motivación del Trabajo de Fin de Grado

Como es bien sabido, desde los últimos años, el mundo ha cambiado drásticamente en varios aspectos como puede ser el económico o social. Uno de los cambios que más ha destacado por culpa de la revolución digital es el cambio tecnológico. Además, en estos últimos años ha incrementado mucho más con la revolución de la inteligencia artificial. En este cambio tecnológico, la informática toma un papel importante, donde la programación es uno de los procesos claves de esta revolución.

El proceso de programar consta de crear un conjunto de instrucciones que realizará una computadora. Existen varias ramas de programación dentro de la propia programación informática. Una de ellas es la programación competitiva, la cual trata de un torneo entre programadores. Los programadores deben realizar una serie de problemas, los cuales contienen una descripción, un ejemplo de datos de entrada y cual debería ser su salida o solución. Para ganar la competición, los problemas deben ser *lo más eficientes posibles*, es decir, se deben realizar de tal manera que sus algoritmos sean capaces de transformar los datos de entrada a su salida o solución en el menor tiempo posible, además de ocupar un espacio de memoria determinado.

Para competir en este tipo de torneos es necesario aprender las bases de la programación. Además, en la programación existen varios tipos de algoritmos. Estos son muy importantes ya que, si usamos uno u otro en un problema, conseguiremos tardar más, o menos, en ejecutarlo. Para aprender estos algoritmos primero debemos estudiar otros aspectos como el coste computacional, que se aprenderá a lo largo de este trabajo.

Este trabajo tiene como motivación adentrarse en el mundo de la programación competitiva partiendo desde un nivel básico. Desde este primer nivel se irá aprendiendo y practicando hasta conseguir realizar un mínimo de cincuenta problemas distintos. Cada uno de estos será realizado con uno de los lenguajes de programación más comunes: el C++.

Una muestra de la importancia de usar un algoritmo u otro a la hora de realizar un problema es la siguiente:

Se tiene una colección de varios números en orden, y a través de un programa se quiere encontrar un número en concreto dentro de esta colección. Para ello hay distintas opciones:

1. Se puede ir número por número, comparando con el que se quiere encontrar hasta que coincida.
2. Se puede ir primero a la mitad de esta colección y a través de varias interacciones encontrar el número de forma más rápida.

1:

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

int main(){

    int num[10]= {1,2,3,4,5,6,7,8,9,10};
    int time=0;
    int value=0;

    cout<<"valor a encontrar = ".cin>>value;

    for (int i=0; i<10; i++){
        time++;
        if(num[i]==value){
            cout<<time;
        }
    }

    return 0;
}
```

Figura 1: Solución al primer ejemplo de tipo de algoritmos.

En este ejemplo, el valor de “time” (el cual representaría de alguna forma el tiempo de ejecución del programa) para el peor escenario, el cual sería encontrar el valor 10, tarda **10 unidades de tiempo**. Este tipo de algoritmo se llama lineal **O(n)**. Se entrará en profundidad a lo largo del trabajo.

2:

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

int main(){

    int num [10]= {1,2,3,4,5,6,7,8,9,10};
    int time=0;
    int inf,sup,mitad,value;
    cout<<"valor a encontrar = ";<<cin>>value;
    inf=0;
    sup=10;

    while (inf<=sup) {
        mitad=(inf+sup)/2;
        time++;
        if(num[mitad]==value){
            cout<<time;
            break;
        }
        if(num[mitad]>value){
            sup=mitad;
        }
        if(num[mitad]<value){
            inf=mitad;
        }
    }

    return 0;
}
```

Figura 2: Solución al segundo ejemplo de tipo de algoritmos.

En este ejemplo, el valor de “time” para el peor escenario, nos tarda **4 unidades de tiempo**. Este tipo de algoritmo se llama **búsqueda binaria, $O(\log n)$** . Se entrará en profundidad a lo largo del trabajo.



1.2 Objetivo del Trabajo de Fin de Grado

El objetivo principal de este trabajo es adentrarse en el mundo de la programación competitiva: aprender sobre su historia y cómo funcionan los algoritmos para su posterior puesta en práctica en los problemas.

Para conseguir esta meta principal, se han marcado pequeñas tareas que a medida que se realicen se estará más cerca del objetivo.

Tarea 1: Informarse sobre la programación competitiva.

Saber la historia de esta modalidad de competición, además de que competiciones existen y qué información hay sobre estas.

Tarea 2: Aprender sobre coste computacional

Después de saber qué es la programación competitiva, se debe aprender sobre el coste computacional y la algoritmia antes de ponerse a realizar problemas. De estos términos interesa su historia, importancia y tipos.

Tarea 3: Problemas

Al acabar, se realizarán más de cincuenta problemas para poner en práctica lo aprendido. Estos se sacarán de las páginas donde se inspiran las competiciones. Los problemas se dividirán por dificultad.

1.3 Estructura del Trabajo de Fin de Grado

Este trabajo está dividido por capítulos, donde en cada uno se tratará una de las tareas anteriormente comentadas. Algunas tareas se realizan en uno o varios capítulos.

CAPÍTULO 1: Este capítulo sirve de introducción al tema y objetivos del trabajo de fin de grado.

CAPÍTULO 2: Se realiza la tarea 1. Se presenta toda la información sobre la programación competitiva.



CAPÍTULO 3: Se realiza la tarea 2. Muestra toda la información relevante sobre el coste computacional y la algoritmia para que posteriormente se puedan realizar los ejercicios.

CAPÍTULO 4: Presenta todos los problemas realizados divididos por su dificultad. Además de mostrar de qué trata cada uno. Se realiza la tarea 3.

CAPÍTULO 5: Explica cada problema por separado, indicando las bases de su procedimiento y cómo solucionarlo. Se realiza la tarea 3.

CAPÍTULO 6: Este último servirá como conclusión del trabajo de fin de grado.

CAPÍTULO 2: Programación competitiva

Para cualquier persona que escucha o lee por primera vez este término, no entiende muy bien a qué se refiere. Nunca se suelen ver las palabras “programación” y “competitiva” juntas, ya que es muy difícil saber quién programa mejor, debido a que existen multitud de lenguajes de programación y la dificultad de determinar qué programa es más funcional.

2.1 ¿Qué es la programación competitiva?

La programación competitiva es un deporte que consiste en que varios programadores compiten entre sí para resolver la mayor cantidad de problemas en un tiempo limitado. Este deporte se practica generalmente a través de internet o también de forma presencial en una red local. Los programadores que compiten en esta modalidad son llamados programadores deportivos.

El ganador será aquel que logre realizar más problemas en menos tiempo. Las reglas pueden variar según el concurso, pero generalmente los resultados se juzgan en función del número de preguntas resueltas, el tiempo usado en resolver estas preguntas, el tiempo que tarda en ejecutarse el programa y el tamaño del programa, entre otros factores.

Estas competiciones suelen ser entre equipos universitarios y de colegios. Los ganadores en sus competiciones regionales pasan a las internacionales, como en cualquier otro deporte. El nivel es distinto según la edad de los concursantes, no es igual una competición entre colegios que entre universidades.

Adentrarse en la programación competitiva es un factor que se debería considerar cualquier persona que desee iniciarse en el mundo de la programación, ya que le proporcionará experiencia y un amplio conocimiento de algoritmos y estructura de datos. Además, obtendrán conocimiento avanzado respecto a la resolución de problemas y generarán redes de contacto para un futuro. No es necesario que se compita en concursos de programación, pero sí es útil practicar los problemas y aprender los fundamentos de algoritmos para aplicar a puestos de trabajo en Google, Meta, Amazon, Apple, Microsoft o cualquier empresa tecnológica grande.

2.2 Competiciones Internacionales

Hoy en día existen varias competiciones a nivel internacional de distinto nivel e importancia. Al igual que cualquier otro deporte, este también cuenta con sus países estrellas que destacan en nivel sobre los demás. Rusia, China, Estados Unidos o Polonia son los países más fuertes en esta modalidad de programación. Además, al igual que los deportes de equipo, aparte de selecciones hay clubes, donde destacan las universidades ITMO de Rusia, Estatal de San Petersburgo de Rusia, Shanghai Jiao Tong de China o la Universidad de Varsovia de Polonia. También existen jugadores que destacan sobre los demás, donde actualmente el jugador estrella es el Bielorruso Gennady Korotkevich, un prodigio en este deporte, el cual empezó a programar a sus 7 años.

2.2.1 Competición Internacional Universitaria de Programación

La primera competición de esta modalidad se realizó en Texas A&M University en 1970. Esta pasó a ser una competición con varias rondas clasificatorias en 1977 y es conocida como la **ICPC** (International Collegiate Programming Contest), la competición más importante y prestigiosa de este deporte. Esta competición proporciona a sus estudiantes participantes la oportunidad de demostrar y mejorar sus habilidades de resolución de problemas de programación y trabajo en equipo.

2.2.1.1 Historia de la ICPC

Desde 1977 hasta 1989 participaban principalmente equipos provenientes de Estados Unidos y Canadá. Cada año ha ido aumentando entre un 10 y un 20 por ciento el número de equipos participantes. En la actualidad participan más de cincuenta mil estudiantes de más de tres mil universidades de todo el mundo.

La sede central está ubicada en la Universidad de Baylor, Texas. Desde el 1989 y hasta el 2017 estuvo patrocinada por la **ACM** y se denominó **ACM-ICPC**. La **ACM** (Asociación de Maquinaria Computacional) se fundó el 1947 en Nueva York y es una sociedad sin fines de lucro científica y educativa en el campo de la computación. Otro de sus principales

patrocinadores es **IBM** una de las empresas tecnológicas multinacionales con sede en Nueva York, se dedica a fabricar y comercializar software y hardware para computadoras.

Debido a la pandemia y al Covid-19, esta competición se detuvo unos años, ya que las finales mundiales son presenciales y por tanto van dos años atrasados. La última se realizó en Dhaka, Bangladesh el 2022 (donde participaron los finalistas regionales del año 2020/2021) y el ganador fue el **MIT** (Instituto de Tecnología de Massachusetts). Pretende volver la última semana de noviembre de este año 2023 planeando determinar los dos ganadores mundiales de los respectivos finalistas regionales de los años 2021/2022 y 2022/2023 simultáneamente en el mismo evento, pero con medallistas separados.



Figura 3: Ganadores de la ICPC'22

[2.2.1.2 Reglamento de la ICPC](#)

La **ICPC** es una competición por equipos constituidos por tres estudiantes de universidad los cuales llevan menos de 5 años estudiando. Aquellos estudiantes que ya hayan competido en dos finales mundiales o en cinco competiciones regionales no se les permite participar otra vez.

Existen competiciones locales, regionales y la final mundial. A la final pasarán los equipos mejor clasificados en las competiciones regionales, los cuales no pueden ser de la misma universidad. Para llegar a la final, hay que pasar varias fases clasificatorias:



1. Competiciones Locales:

Algunas universidades tienen una competición local para elegir a los miembros de su equipo. En otras optan por elegir a los estudiantes con mejor nota o los que están más interesados. Estas competiciones son opcionales y son organizadas por cada universidad como esta vea conveniente.

2. Competiciones Regionales:

Los equipos de cada universidad participan en esta competición contra equipos próximos geográficamente. Cada región enviará a la final un número concreto de equipos, de los cuales ninguno puede ser de la misma universidad.

Hay más de 30 regiones en todo el mundo las cuales algunas agrupan distintos países, otras solo un país y otra solo parte de un país.

La competición **SWERC** comprende los países de Europa situados en el suroeste, como son España, Portugal, Francia, Italia, Suiza y el oeste de Austria.

Finalmente se produce la final mundial, la cual se celebra presencialmente cada año en una ciudad distinta y participan todos los equipos ganadores de las competiciones regionales. En esta final participaran un máximo de 140 equipos (sujeto a variación).

Cada una de las anteriores competiciones siguen las siguientes reglas:

1. Miembros:

El equipo estará constituido por 3 deportistas más un entrenador, en ningún caso un competidor del equipo puede actuar como entrenador. Además, el equipo que avance para la final contendrá los mismos miembros con los que contaba cuando se clasificó. Si uno de ellos no quiere o no puede competir en las finales, el entrenador deberá notificarlo y el deportista será descalificado de otras competiciones del **ICPC** (puede apelar la descalificación).

2. Dispositivos informáticos:

Cada equipo dispondrá de un ordenador. No está permitido traer ningún otro dispositivo electrónico ni material impreso durante la competición.



3. Duración y número de problemas:

Los deportistas tendrán alrededor de 5 horas para resolver unos 8 o 15 problemas. En las competiciones regionales normalmente son 8 problemas y en la final mundial 12.

4. Lenguajes permitidos:

Según la normativa se pueden usar 5 lenguajes distintos: **C**, **C++**, **Java**, **Python** o **Kotlin**. La **ICPC** indica que los jueces habrán resuelto todos los problemas en al menos dos de los tres grupos de idiomas distintos (Java/Kotlin, C/C++ y Python). Se aconseja usar los lenguajes **C++** o **Java** ya que son dos de los lenguajes más utilizados.

5. Envíos:

Cada problema se puede enviar tantas veces como quieran los equipos hasta que el juez notifique que la solución es correcta. Los problemas pueden tener distintos tipos de errores: **CE**, **RTE**, **LTE**, **WA**. Para cada envío extra se suman 20 minutos al total de tiempo que lleva el equipo concursando.

6. Puntuación:

El equipo ganador es aquel que consigue resolver más problemas. Si existe un empate, la clasificación se calcula a partir del tiempo total que ha utilizado cada equipo para resolver los problemas. Si aun así empata, ganará el que haya tardado menos en realizar la última serie de problemas presentada.

2.2.2 Olimpiada Internacional de Informática

Otra de las competiciones más importantes a nivel internacional en el mundo de la programación es la **IOI**. Se ha celebrado en los cinco continentes y ha traído delegaciones de seis. Este evento es equivalente a los juegos olímpicos, el mayor evento deportivo internacional multidisciplinario.

2.2.2.1 Historia de la IOI

La primera vez que surgió el nombre de la **IOI** fue en la 24ª conferencia general de la **UNESCO** en París en octubre de 1987. En mayo de 1989, la **UNESCO** inició la primera Olimpiada Internacional de Informática y se celebró en Bulgaria. Son respaldadas por la **IFIP**.

Estas olimpiadas son celebradas anualmente y tienen como objetivo estimular el interés por la informática, además de servir como experiencia para los concursantes y que compartan intereses entre ellos. Cada año se organiza en un país distinto de entre los que participan y cada uno envía un grupo de cuatro concursantes y dos adultos acompañantes. El país donde se celebra es el que tendrá la responsabilidad de la organización.

Los estudiantes compiten individualmente. Las olimpiadas suelen durar una semana, pero la competición se realiza solamente en dos días, los días restantes se organizan distintos eventos educativos y de entretenimiento. Los ganadores de este evento corresponden al grupo de los mejores jóvenes informáticos del mundo.

Los meses más típicos en los que se celebra este evento son junio, julio y agosto, aunque se han realizado en meses de mayo y noviembre como las **IOI'91** en Grecia o las **IOI'97** en Sudáfrica respectivamente. Las próximas olimpiadas, las **IOI'23**, se celebrarán en Hungría entre el 28 de agosto y el 4 de septiembre.

Debido al COVID 19, las olimpiadas del 2020 y 2021 que en un principio se iban a realizar en Singapur, se llevaron a cabo como un concurso en línea. La **IOI'22** organizada por Indonesia fue un evento híbrido, un 25% de los concursantes participaron en línea.



Figura 4: Ceremonia IOI'22



Se realizan varias asambleas generales previamente y durante el evento para discutir las reglas y comentar la evolución de las olimpiadas.

Para que un país se convierta en el anfitrión de las **IOI**, debe pasar por distintos acontecimientos:

1. Carta de Intención:

Si un país quiere y es capaz de organizar las olimpiadas un año. Debe enviar una carta de intención al secretario donde quedarán redactados sus motivos y capacidades.

2. Anfitrión potencial:

Una vez el secretario ha recibido la carta y considera que es capaz, indica al país que es considerado un anfitrión potencial.

3. Candidato a anfitrión:

De todos los países que han presentado una carta de intención y se han considerado anfitriones potenciales, se nombra solo a uno. Cuando se haya elegido, el secretario enviará una carta de invitación a dicho país y este se convertirá en candidato a anfitrión de las olimpiadas.

4. Futuro anfitrión:

El candidato a anfitrión recibe el estatus de futuro anfitrión cuando la invitación es aceptada por la organización nacional.

5. Anfitrión actual:

Finalmente, el país se convierte en anfitrión actual en el momento que se realiza la olimpiada internacional de informática.

Al igual que la **ICPC**, se deben realizar competiciones regionales y nacionales antes de llegar a las olimpiadas internacionales. Para poder llegar a la olimpiada informática española (**OIE**) hay dos formas:



1. Olimpiada informática regional:

Se clasifican máximo dos concursantes de cada olimpiada regional. Una de ellas es realizada en la **UIB** de manera presencial en la escuela politécnica superior. La última se realizó el 11 de febrero del 2023.

2. Concurso clasificatorio abierto:

Un concurso en el que puede participar cualquier persona que cumpla las características de la **IOI**. Se hace online y su objetivo es aumentar el número de finalistas y conceder una segunda oportunidad de clasificación. Accederán a la olimpiada nacional los 10 mejores concursantes.

Si te has clasificado a través de las dos opciones, prevalecerá la olimpiada regional.

Finalmente, los cuatro mejores concursantes de la olimpiada nacional formarán parte de la selección nacional que representará a dicho país en la IOI.

2.2.2.2 Reglamento de la IOI

A diferencia de las **ICPC**, en las olimpiadas internacionales de informática los concursantes compiten individualmente. Aunque la finalidad es la misma: resolver una serie de problemas informáticos.

La competición durará dos días, cada uno de ellos vendrá seguido de un día de no competición, en el que se realizarán otras actividades más recreativas. La competición seguirá las siguientes reglas generales:

1. Miembros:

Los concursantes que participan deben ser estudiantes de secundaria, además de no ser mayor de veinte años el 1 de julio del año de la **IOI** en la que formará parte. Deben haber pasado por las distintas competiciones regionales y nacionales.

2. Dispositivos informáticos:

Cada concursante dispondrá de un ordenador, papel y boli. No está permitido traer ningún otro dispositivo electrónico ni material impreso durante la competición. Si el concursante necesita traducir la información, debe informarlo a la organización y se le traducirá a su idioma nativo sin ningún dato añadido.



3. Duración y número de problemas:

Cada uno de los dos días de competición constará de tres problemas que los deportistas deberán resolver en un máximo de 5 horas.

4. Lenguajes permitidos:

Antes del 2019 se podía usar tanto el lenguaje **C++** como el **Pascal**. A partir del 2019 en adelante, todos los problemas deben ser resueltos únicamente en **C++**.

5. Puntuación:

Los problemas se dividen en subtarear con distinta dificultad, donde la primera es más fácil que la última, y los puntos se otorgan cuando todas las pruebas para una subtarea son correctas. Los problemas tienen un límite específico de memoria y tiempo.

Los concursantes reciben medalla en función de su puntuación total relativa. Solo reciben medalla el 50% superior en cuanto a puntuación, donde 1/12 recibirá medalla de oro, 1/6 medalla de plata y 1/4 medalla de bronce.

2.2.3 Meta Hacker Cup

Además de las competiciones internacionales organizadas por países, muchas de las empresas multinacionales como Meta, realizan sus propios eventos de programación competitiva. Otras empresas como Google o Amazon también organizan eventos propios.

[2.2.3.1 Historia y reglamento de la Meta Hacker Cup](#)

La competencia se inició en el 2011 con el objetivo de identificar talentos para un futuro trabajo en Meta. Consiste en resolver un conjunto de problemas algorítmicos en un tiempo determinado.

Es una competición internacional anual abierta a todo el mundo que tenga, al menos, 18 años. Para participar se debe estar registrado en Meta y haber aceptado los términos y servicios. Los mejores concursantes recibirán premio en función de la posición en la calificación.



2.2.3.2 Estructura del concurso

El concurso está estructurado por cinco rondas:

1. Ronda de calificación:

Esta ronda está abierta durante cinco días y cualquiera puede participar. Si un participante resuelve al menos un problema correctamente, avanzará a la siguiente ronda.

2. Ronda 1:

Dura un día y, para avanzar de ronda, se deben obtener un mínimo de puntos. Estos cambian según el año, son anunciados antes de iniciar la ronda.

3. Ronda 2:

Tiene una duración de tres horas. Los 2000 mejores competidores recibirán una camiseta de la Meta Hacker Cup como premio, los 500 con la puntuación más alta pasarán de ronda. Si existe algún empate, este se romperá por el tiempo total de penalización.

4. Ronda 3:

Tiene la misma duración que la ronda anterior, tres horas. Los 200 mejores en esta ronda recibirán una camiseta de la Meta Hacker Cup con una insignia de “Top 200”. Para pasar a la ronda final debes estar entre los 25 mejores.

5. Ronda final:

La ronda final dura cuatro horas y el ganador será el finalista de la Meta Hacker Cup.

Los concursantes de la última ronda recibirán dinero (USD) como premio dependiendo de la posición final. El finalista recibirá 20.000\$, el segundo 10.000\$, el tercero 5.000\$, el cuarto 3.000\$ y el quinto 1.000\$. Finalmente, las posiciones restantes se reparten el premio por grupos: los jugadores que han quedado entre la sexta y décima posición recibirán 500\$, entre onceava y quinceava posición 300\$ y entre dieciseisava y vigésimo quinta posición 200\$.

2.2.3.3 Método de puntuación

En cada ronda habrá cuatro o más problemas para resolver, los cuales se pueden realizar en cualquier lenguaje de programación. Cada problema tiene un valor en puntos distintos.

Un problema se puede enviar tantas veces como uno considere, para corregir se tendrá en cuenta el último envío. Si su solución es correcta, recibirá los puntos del problema. Si su solución no es correcta no se le informará hasta que termine la ronda, momento en el cual se muestran los problemas incorrectos como tal.

Para cada problema solucionado correctamente se ganará tiempo de penalización. El tiempo de penalización es aquel tiempo que le tomó al concursante resolver el problema. Es utilizado a la hora de surgir un empate, donde el tiempo de penalización más bajo es el ganador.



Figura 5: Algunos de los finalistas de la Meta Hacker Cup 2018

2.3 Competiciones nacionales

Además de competiciones internacionales, en España también se realizan concursos de programación competitiva a nivel nacional. Una de ellas es la **OIE**, la cual forma parte de un proceso para llegar a la **IOI**. Existen otras competiciones independientes que también cabe destacar.

2.3.1 Concurso universitario AdaByron

AdaByron es un concurso universitario de programación bastante reciente, la primera edición se realizó en el año 2015. Surgió con el objetivo de fomentar la participación de los alumnos de las universidades públicas de Madrid en los concursos universitarios como la **SWERC**, ya que la representación madrileña había caído en picado.

Su nombre viene por la británica Augusta Ada Byron, una matemática y escritora considerada la primera programadora de computadoras del mundo. Propuso la primera pieza de código informático del mundo: un algoritmo que podría calcular la secuencia de números de Bernoulli.

2.3.1.1 Historia de AdaByron

Los impulsores de **AdaByron** son los que pertenecieron en la organización de algunas ediciones de la **CUPCAM** y de las ediciones de **SWERC** de los años 2009, 2010 y 2011.

La **CUPCAM** fue un concurso que nació en el año 2003 en la comunidad de Madrid con el objetivo de promover la actividad de programación en las universidades públicas. Se realizaron varias ediciones hasta la última en el año 2009. Después organizaron tres ediciones consecutivas de la **SWERC**.

En el año 2011 los promotores de **AdaByron** crearon un concurso llamado **ProgramaMe**, centrado en estudiantes de ciclos formativos. Hoy en día se realiza cada año a nivel nacional.

Finalmente, en el año 2015 se realiza la primera edición del concurso de **AdaByron**, con el objetivo de realizar un concurso interno que llame la atención a alumnos y los introduzca en el mundo de la programación competitiva, con la intención de, en un futuro, participar en las competiciones regionales. Esta primera edición fue interna de la facultad de informática de la Universidad Complutense de Madrid y se realizó durante la semana de la informática a toda prisa.

Esta primera edición fue tan exitosa que se llevaron a cabo tres ediciones más, esta vez abiertas a todas las universidades de la comunidad de Madrid. En el año 2019 el concurso se hace tan grande que sale de la comunidad de Madrid y varias universidades de otras comunidades se animaron a participar.

A causa del COVID 19, el concurso se tuvo que parar dos años para su próxima evolución. En el año 2021 se convirtió en un evento de dos niveles: una competición a nivel regional y otra a nivel nacional donde participaran los mejores clasificados de cada región. La primera final nacional se organizó en la facultad de informática de la Universidad Complutense de Madrid el 8 y 9 de julio del 2022.



Figura 6: Edición AdaByron 2022

2.3.1.2 Reglamento de AdaByron

Los estudiantes utilizan portales como **Acepta el reto** para entrenar. Estos portales contienen cientos de problemas, de algunos de los cuales se inspira **AdaByron** para realizar los suyos.

La final nacional es un evento presencial, aunque algunas competiciones regionales pueden realizarse de forma online. A partir del año 2017 existe un precio de inscripción, ya que hay que cubrir los gastos de organización, como los premios.

El reglamento de esta competición es muy parecido al de la **ICPC**. Puede variar según la edición, el reglamento de la última edición realizada el 7 y 8 de julio de 2023 fue el siguiente:



1. Miembros:

Los alumnos participan en equipos de 3 personas. La única condición para participar es estar cursando un grado de informática, ya sea orientado al software o al hardware. Puede participar tanto un estudiante de primer curso como otro de último curso. Cada equipo pertenecerá a una de las dos categorías de la competición según en base a las asignaturas cursadas o superadas de sus integrantes: categoría AB para alumnos de primero o segundo y categoría C para alumnos de tercero o superior.

2. Dispositivos informáticos:

Como el evento es en un entorno restringido, el equipo no necesita traer ningún dispositivo. Se usan los ordenadores de la propia organización, todos son el mismo modelo y tienen el mismo sistema operativo: **Linux**. Los estudiantes tendrán acceso a documentación de **Java**, manual de **C++** y librerías de **C**.

Además, podrán traer un dossier con información adicional de un máximo de 25 páginas. Se entregará una copia a la organización para su revisión, la cual podrá prohibir su uso si no cumple los requisitos.

3. Duración y número de problemas:

Los deportistas tendrán alrededor de 5 horas para resolver entre 8 y 12 problemas (puede variar el número de problemas según la edición). Estos no están ordenados por nivel de dificultad y son siempre aplicaciones de consola.

Su dificultad varía desde problemas fáciles que podrían solucionar alumnos de primer curso, como problemas más complejos que requieren técnicas solo al alcance de los alumnos de último curso.

4. Lenguajes permitidos:

Los participantes deben solucionar los problemas en **C**, **C++**, **Java** o **Python**. Cada problema puede ser solucionado por un lenguaje distinto.



5. Envíos:

Cada problema se puede enviar tantas veces como quieran y el juez indicará si el problema ha sido enviado correctamente o no. Los problemas pueden tener distintos tipos de errores: **AC**, **RTE**, **WA**, **LTE**, **CE**. Para cada envío incorrecto se le añade 20 minutos de penalización al equipo, el cual solo será añadido si dicho problema acaba siendo enviado correctamente.

Las soluciones enviadas se validan a través de un juez automático supervisado por jueces humanos. Este ejecuta el problema de manera automática contra un conjunto de casos de prueba no conocidos y establecidos anteriormente.

Las soluciones deben realizarse en un límite de tiempo y memoria. Aunque un problema que sea más eficiente que otro no quiere decir que vaya a tener más puntuación. Solo es necesario que se solucione por debajo del tiempo límite.

6. Puntuación:

Cada problema tiene distinta puntuación según su dificultad. El equipo ganador es aquel que haya conseguido más puntos, si existe algún empate se romperá con el tiempo tardado para solucionar los problemas.

Los problemas de este concurso se centran, principalmente, en cuestiones de algoritmia y estructura de datos, por ello se deben corregir probando muchos casos de entrada. La entrada de los programas será multicaso, por lo que la entrada estará compuesta de muchos casos que pondrán a prueba el código muchas veces.

Los premios varían según la edición, ya que depende del número de participantes y patrocinadores. En el año 2022 se entregó a cada uno de los integrantes del equipo ganador unas gafas de realidad virtual. Existen premios para el equipo que consiga el primer envío correcto del concurso y para aquel que consiga el ultimo envío correcto.

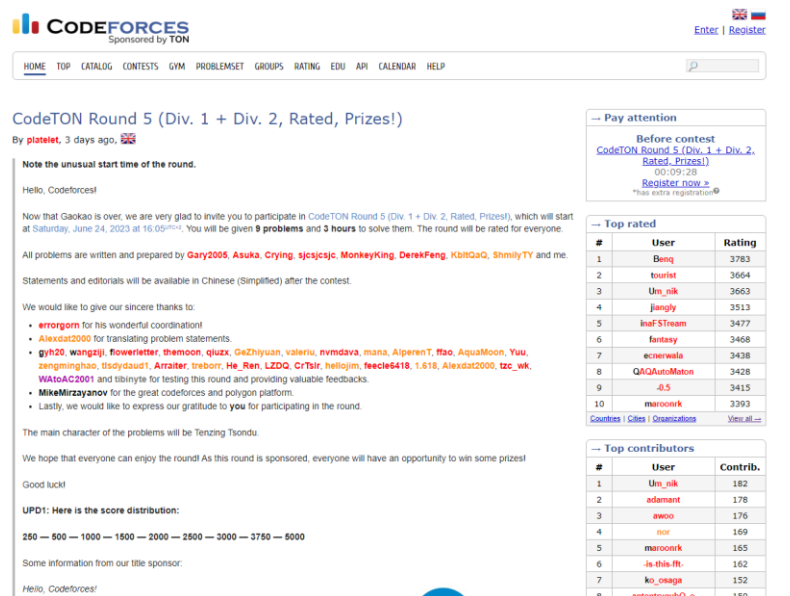
2.4 Zonas de entrenamiento

Todos los deportistas necesitan entrenar y los programadores deportivos no son una excepción. Existen diversas páginas donde uno puede encontrar multitud de problemas de distinta dificultad con los cuales puede practicar su destreza en la programación. Muchos de estos sirven de inspiración para posteriores problemas que se realicen en competiciones.

2.4.1 CodeForces

Una de las páginas web más famosas a nivel internacional es **CodeForces**. Fue lanzada el 10 de abril de 2009 y es patrocinada por la empresa rusa Telegram.

Es una red social de programadores que actúa como un blog donde recopila toda la información de prácticamente todas las competiciones y competidores a nivel mundial. Así como un calendario de los futuros eventos más importantes y diferentes publicaciones donde informa de novedades en el ámbito de la programación. También permite al usuario conectarse con otros de la plataforma y establecer una conversación.



The screenshot shows the CodeForces website with the announcement for CodeTON Round 5 (Div. 1 + Div. 2, Rated, Prizes!). The page includes a header with the CodeForces logo and navigation links. The main content area contains the announcement text, which mentions the round's start time, the number of problems (9), and the duration (3 hours). It also lists the authors of the problems and the sponsors. On the right side, there are two tables: 'Top rated' and 'Top contributors'.

#	User	Rating
1	Benq	3783
2	tourist	3664
3	Um_nik	3663
4	jiangly	3513
5	IsafStman	3477
6	fantasy	3468
7	ecnerwala	3438
8	QAQAutoMaton	3428
9	d.5	3415
10	maroonrk	3393

#	User	Contrib.
1	Um_nik	182
2	adamant	178
3	awoo	176
4	not	169
5	maroonrk	165
6	is this ft.	162
7	ko_osaga	152
8	antontrybasov	150

Figura 7: Página principal CodeForces

Otra faceta de esta página web son sus problemas de programación. Existen multitud de problemas que se pueden solucionar en varios lenguajes de programación. Los problemas constan de un enunciado donde se exponen las características de este, el tipo de entrada que tiene el problema y la salida que se espera recibir. Además, muestran 2 o más entradas de ejemplo y las respectivas salidas para poder poner a prueba el programa antes de enviarlo. Algunos problemas constan de notas adicionales que añaden información extra.

Estos problemas sirven para practicar para los distintos concursos, además de ser usados para el concurso de **CodeForces**. Este concurso es interno de la plataforma y sirve para crear una competición a baja escala entre los usuarios, cada problema resuelto proporciona una puntuación, creando un ranking de los mejores programadores de la página web.

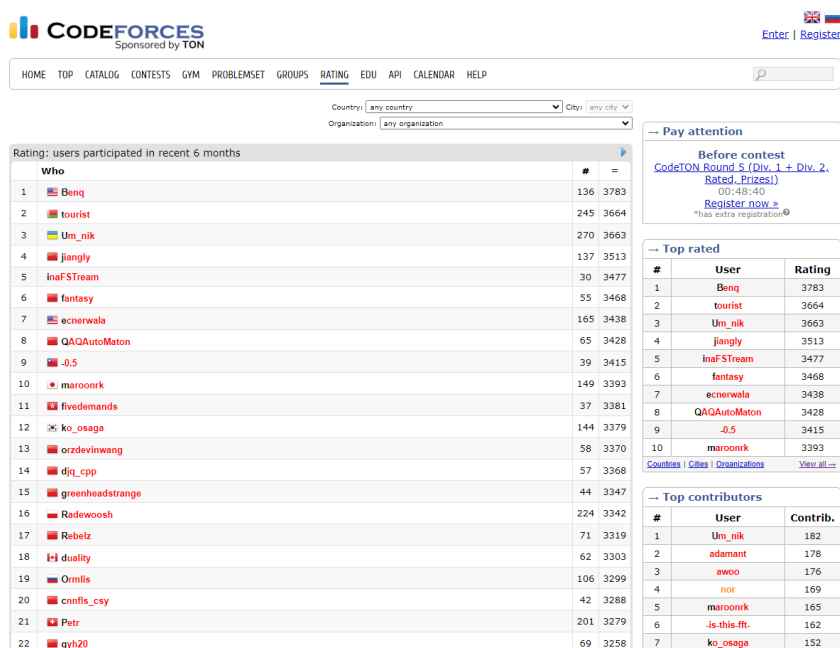


Figura 8: Ranking CodeForces

2.4.2 ¡Acepta el reto!

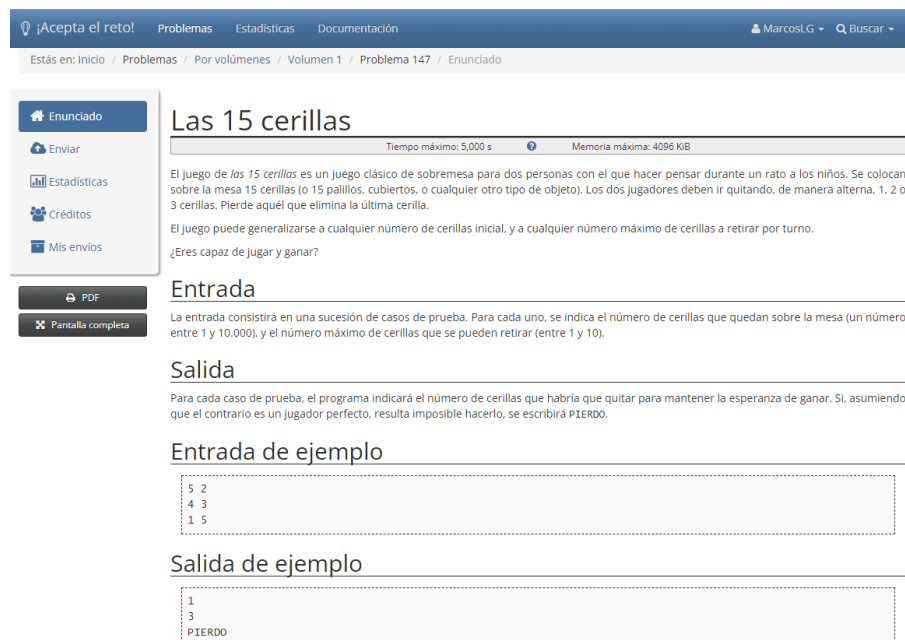
Otra página web muy conocida en España es “¡Acepta el reto!”, un repositorio de problemas de programación en español donde cualquier usuario de la plataforma tiene a disposición cientos de problemas que puede resolver y enviar a un juez para comprobar el resultado.

Nació en la facultad de informática de la **UCM** con el objetivo de fomentar la algoritmia y la programación competitiva entre los alumnos. Contiene todos los problemas aparecidos en **ProgramaMe**, un concurso de programación para ciclos formativos de nivel nacional ya comentado anteriormente.

El archivo de problemas va aumentando cada día y son aplicaciones de consola. Reciben datos en la entrada y envían el resultado por la salida, es decir, leen de teclado y escriben en pantalla.

Igual que en el concurso de **AdaByron**, el juez tiene a disposición un conjunto de casos de prueba desconocidos para el usuario. Cuando recibe un código a corregir, lo ejecuta con todos los casos de prueba y compara la respuesta recibida con la esperada. El juez emitirá un veredicto con el resultado. Puede tener diferentes respuestas: **AC**, **PE**, **WA**, **CE**, **RTE**, **LTE**, **MLE**, **OLE**, **RF**, **IQ** y **IE**.

Los problemas pueden ser solucionados por tres lenguajes de programación: **C++**, **C** y **Java**, además tienen un tiempo límite de ejecución y de memoria. Están formados por un enunciado explicativo, el tipo de entrada y la salida a esperar sumado a un ejemplo de entrada y de salida.



The screenshot shows the user interface for a programming problem titled "Las 15 cerillas". The top navigation bar includes links for "¡Acepta el reto!", "Problemas", "Estadísticas", and "Documentación", along with a user profile "MarcosL.G" and a search icon. The breadcrumb trail indicates the user is in "Estás en: Inicio / Problemas / Por volúmenes / Volumen 1 / Problema 147 / Enunciado".

On the left sidebar, there are buttons for "Enunciado" (selected), "Enviar", "Estadísticas", "Créditos", "Mis envíos", "PDF", and "Pantalla completa".

The main content area displays the problem title "Las 15 cerillas" with constraints: "Tiempo máximo: 5,000 s" and "Memoria máxima: 4096 KiB". The description explains the game of 15 matches, where two players take turns removing 1, 2, or 3 matches from a pile of 15. The player who removes the last match loses. The problem asks if the user can win given an initial number of matches and a maximum removal limit.

The "Entrada" (Input) section states that the input consists of a sequence of test cases, each with a number of matches (1 to 10,000) and a maximum removal limit (1 to 10).

The "Salida" (Output) section explains that for each test case, the program should output the number of matches to remove to maintain a winning chance, or "PIERDO" if it's impossible to win against a perfect opponent.

The "Entrada de ejemplo" (Example Input) shows a 3x2 grid of numbers:

```
5 2
4 3
1 5
```

The "Salida de ejemplo" (Example Output) shows a 3x1 grid of numbers and the word "PIERDO":

```
1
3
PIERDO
```

Figura 9: Esquema general de un problema

Cada usuario puede observar sus propias estadísticas y la de los demás usuarios, donde se puede acceder a información como: número de problemas resueltos, número de envíos de cada problema y que tiempo tarda en ejecutarse el programa.



Figura 10: Estadísticas de un problema.

Puedes buscar los problemas por volúmenes de 100 en 100 o por categorías, donde puedes acceder a problemas realizados en exámenes de alguna institución educativa o realizados en concursos de programación. También están organizados por tipo de algoritmo a usar.



Figura 11: Categorías y subcategorías de ¡Acepta el reto!

CAPÍTULO 3: Coste computacional

Después de conocer qué es la programación competitiva y los distintos torneos y zonas de entrenamiento existentes, es necesario entender ciertos conceptos importantes antes de ponerse a practicar problemas. Uno de los conceptos primordiales es el coste computacional, el cual está directamente relacionado con el tiempo de ejecución del problema, un aspecto a tener en cuenta en los torneos de programación.

3.1 ¿Qué es el coste computacional?

El coste computacional de un algoritmo nos indica la cantidad de tiempo y memoria que gasta antes, durante y después de su ejecución. Cuando un algoritmo es poco eficiente, quiere decir que tiene un coste computacional muy alto. Para averiguar el coste computacional de un algoritmo se usa la notación asintótica, la cual “proporciona el vocabulario base para discutir el diseño y análisis de los algoritmos” [1] y es lo suficientemente precisa como para hacer comparaciones entre distintos algoritmos de alto nivel que solucionan un problema, es decir, nos permite “diferenciar entre qué algoritmo es mejor o peor para realizar un tipo de tarea como ordenación de números o multiplicación de dos enteros, especialmente para entradas grandes” [2]. Se basa en el concepto de la notación “**big-O**”, el cual debe estar en el vocabulario de cualquier programador. Este concepto tiene como objetivo suprimir detalles secundarios como el tipo de lenguaje que se utiliza o el tipo de compilador.

La notación asintótica elimina los valores constantes y términos de bajo orden para reducir la ecuación primitiva correspondiente al tiempo de ejecución de un algoritmo en una genérica. Los términos de bajo orden se vuelven irrelevantes cuando las entradas son grandes y las constantes dependen del entorno de programación. Cuando no se quieren especificar detalles tiene más sentido no utilizar las constantes.

La notación “**big-O**” sirve para agrupar los algoritmos en función de sus tiempos de ejecución en el peor caso. Existen diferentes tipos de notación y su definición matemática es la siguiente:

$T(n) = O(f(n))$ sí y solo si la función $T(n)$ está siempre por debajo de la función determinada por la “**big-O**”.

$$T(n) \leq c * f(n) \quad \text{para todo } n \geq n_0$$

Donde c es un constante y n_0 el punto a partir del cual se cumple.

3.1.1 Tipos de notación “big-O”

Hay distintos tipos de notación “**big-O**”, cada una de ellas tiene un tiempo de ejecución distinto y siempre se intentará realizar un programa con la “**big-O**” más baja posible.

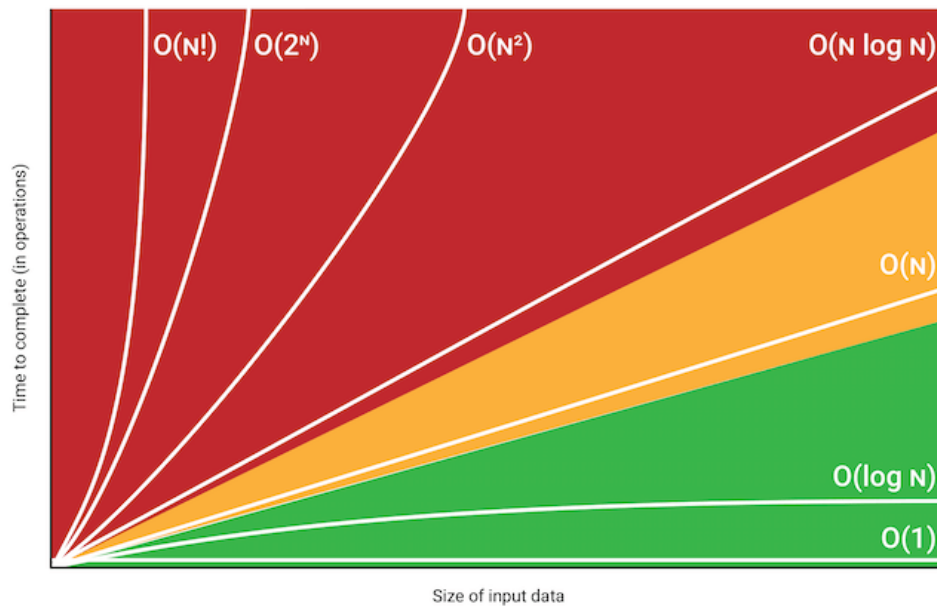


Figura 12: Diferentes valores de Big-O

El eje “**x**” del gráfico corresponde al tamaño de la entrada y el eje “**y**” al tiempo de ejecución del programa. Cuando la entrada empieza a ser muy grande, los algoritmos que tienen “**big-O**” cuadrática y superior (zona roja de la figura 12) tienen un tiempo de ejecución muy alto, por tanto, hay que intentar evitarlos.

3.1.1.1 Notación $O(1)$

Los programas más rápidos son aquellos con notación **$O(1)$** , es decir, siempre tienen el mismo tiempo de ejecución independientemente de la entrada. También son aquellos menos frecuentes, ya que es muy complicado realizar un problema que no dependa de la entrada.

3.1.1.2 Notación $O(n)$

Con este tipo de algoritmos, el tiempo de ejecución es directamente proporcional al tamaño de la entrada. Como se ha mostrado en la Figura 1.

3.1.1.3 Notación $O(\log n)$

Tiene un tiempo logarítmico. Cuando el tamaño de la entrada crece, el tiempo de ejecución tiende a estabilizarse. Se asocia con algoritmos que parten el problema en trozos para poder solucionarlo. Siempre que se realiza un algoritmo se intenta que tenga este tiempo de ejecución logarítmico, como por ejemplo ocurre en la **Búsqueda binaria**, Figura 2.

3.1.1.4 Notación $O(n \log n)$

Es parecido al anterior, pero se trata de funciones que trocean el problema por cada elemento, recomponiendo la información tras la ejecución de cada trozo. Aparece en algoritmos como **MergeSort** y **Divide y vencerás**.

3.1.1.5 Notación $O(n^2)$

El tiempo de ejecución crece exponencialmente por lo que será un algoritmo que evitar. Aunque para valores de entrada pequeños sea asumible, cuando la entrada aumente el tiempo de ejecución tenderá a ser muy elevado.

3.1.1.6 Notación $O(2^n)$

En condiciones normales no es muy típico realizar algoritmos con este tiempo de ejecución. Son algoritmos que duplican su coste con cada entrada añadida al procesamiento.

3.1.1.7 Notación $O(n!)$

El tipo de algoritmo a evitar. Cualquier algoritmo que siga esta notación es un algoritmo fallido, en cuanto la entrada crece un poco, este ya se considera computacionalmente inviable. Suele encontrarse en algoritmos que intentan solucionar un problema con fuerza bruta.

3.1.2 Algoritmos

Para evitar un tiempo de ejecución muy alto, existen distintos algoritmos y sistemas que son muy utilizados por los programadores. Algunos de ellos se deben aprender antes de realizar un problema de competición. La mayoría son usados para ordenar un array de números.

3.1.2.1 Recursividad

La recursividad es un proceso algorítmico para solucionar un problema. Un algoritmo recursivo es aquel que para solucionarse se llama a sí mismo hasta que se cumple una determinada condición. Se utilizan acciones repetitivas donde cada una se determina mediante el resultado anterior. Tiene dos partes: *el caso base*, el cual será el punto de ruptura y *el caso recursivo*.

Por ejemplo, si se quiere calcular un número elevado a otro, se puede realizar de forma recursiva, donde en cada paso se calcula el número elevado a la mitad (diferenciando si es par o impar) hasta llegar al caso base: cuando n sea menor o igual a 0, el resultado es uno. En cada iteración se devolverá el valor elevado multiplicado por el mismo (como lo elevamos a la mitad, si se multiplica por el mismo número se obtendrá el resultado esperado).

```
def potencia(b,n):  
    """ Precondición: n debe ser mayor o igual que cero.  
        Devuelve: b^n. """  
  
    # Caso base  
    if n <= 0:  
        return 1  
  
    # n par  
    if n % 2 == 0:  
        pot = potencia(b, n/2)  
        return pot * pot  
  
    # n impar  
    else:  
        pot = potencia(b, (n-1)/2)  
        return pot * pot * b
```

Figura 13: Ejemplo recursividad

Por cada entrada de longitud mayor o igual a 1, este algoritmo efectúa como mucho $6n\log_2 n + 6n$ iteraciones. Si se aplican las reglas de la notación asintótica: eliminar las constantes y los términos de bajo orden. Se obtiene un algoritmo de $O(n\log n)$.

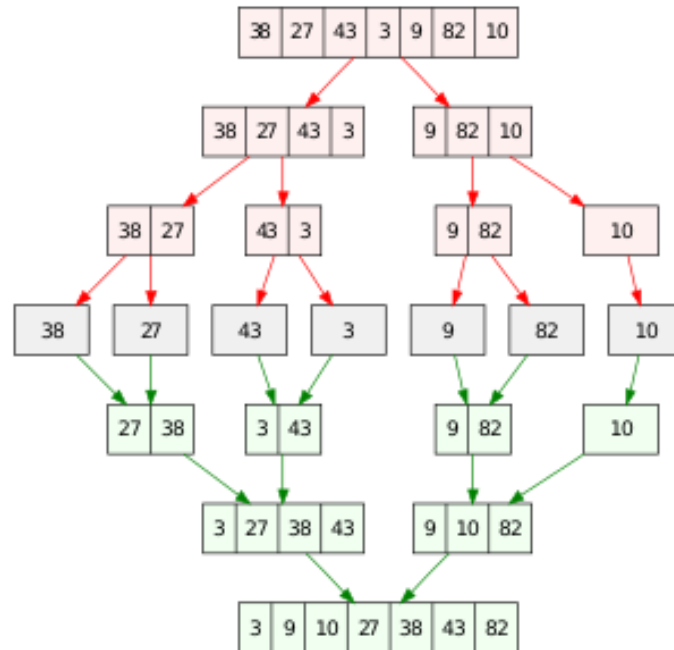


Figura 15: Ejemplo algoritmo MergeSort

3.1.2.5 Divide y vencerás

Divide y vencerás es un método muy utilizado en varios algoritmos ya que permite que estos sean mucho más eficientes al convertirlos en logarítmicos. Se resume en tres pasos básicos:

1. *Dividir* la entrada en subproblemas más pequeños
2. *Solucionar* los subproblemas de manera recursiva
3. *Combinar* las soluciones de los subproblemas en una única solución para el problema original

CAPÍTULO 4: Lista problemas realizados

Después de informarse sobre la programación competitiva y aprender los conceptos básicos de algoritmia y coste computacional, es hora de realizar problemas para poner en práctica todo lo aprendido. Se han realizado más de 50 problemas. Estos se han sacado de la página “**¡Acepta el reto!**” explicada anteriormente. Cada problema está formado por un enunciado que explica una historia para poner en contexto al programador, además de hacerlo más entretenido.

Se ha usado el lenguaje C++ ya que es un lenguaje muy utilizado en el ámbito de la programación y era uno de los que aceptada la página web. Además, se ha usado el **VS Code**; uno de los editores de código más populares.

En este apartado se indicarán qué problemas se han llevado a cabo ordenados por dificultad. Sin entrar en detalles de cómo se solucionan, pero si mostrando un resumen de su enunciado.

4.1 Problemas de dificultad baja

Para poner en práctica todo lo aprendido es importante ir paso a paso, por ello se han realizado varios problemas sencillos. Aunque los problemas no sean difíciles, han permitido aprender lo más básico, como el esquema inicial de un programa y cómo se escribe y lee, la salida y la entrada respectivamente.

4.1.1 Problema 116: ¡Hola mundo!

Este es el primer programa que realiza cualquier persona que se adentra por primera vez en la informática. Trata de mostrar por pantalla la frase “¡Hola mundo!” el número de veces que nos pidan.

4.1.2 Problema 114: Último dígito del factorial

Se pide calcular el factorial de un número, pero al ser entradas muy grandes, solo se devolverá el último dígito del resultado.

4.1.3 Problema 380: ¡Me caso!

Va de los gastos que se realizan en una boda y sobre la palabra “invitado”. La protagonista de la historia se queja de tener que dar dinero a los que se casan cuando era bien claro que es una invitada. Ahora que se casa ella entiende por qué y trata de no gastar mucho en su boda.

4.1.4 Problema 245: Los Dalton

Se explica la historia de los Dalton, unos personajes secundarios de la serie de comics “Lucky Luke” los cuales eran hermanos. Explica que estos personajes siempre aparecían en orden de altura y pregunta si la entrada que manda corresponde a los hermanos Dalton o no.

4.1.5 Problema 216: Goteras

Trata de unas goteras que han aparecido en el salón del lector. Estas caen cada segundo y pide cada cuanto hay que cambiar el cubo según el número de gotas que es capaz de retener.

4.1.6 Problema 117: La fiesta aburrida

Explica la historia de Tinín, una persona a la cual no le gustan nada las fiestas ni las celebraciones ni reuniones. Su novia consigue llevarle a una fiesta y pide si se le puede ayudar a saludar a todos los presentes.

En este problema aparece la librería “**sstream**”, la cual nos permite crear “strings” como “streams” y poder leer y separar la frase del “string” en palabras individuales.

4.1.7 Problema 184: El sueño de los concursantes

Este problema va de una persona que está en su primera programación competitiva. Esta ha dormido muy poco por los nervios y se pregunta si a los demás concursantes les ha pasado igual y cuánto habrán dormido.

4.1.8 Problema 506: Tensión descompensada

El protagonista explica lo preocupado que está de la hipertensión arterial, ya que él es hipertenso. Tan preocupado que tiene un tensiómetro en cada habitación para poder controlarse. El problema es que no quiere equivocarse al leer el resultado del tensiómetro y pide que se le ayude a saber si la lectura es correcta o no.

4.1.9 Problema 512: Döner sospechoso

Trata de una persona que se dirige a un establecimiento donde venden “Döner Kebabs”, este pide qué carnes componen uno en específico muy barato y el vendedor le indica que está hecho de mitad caballo y mitad conejo. Dice que para cada conejo pone un caballo. La persona prefiere irse al conocer que realiza tan mal las mezclas.

4.1.10 Problema 616: Pic,Poc,Pic...Pong!

Expone la historia de la primea videoconsola creada el 1972 y como dos meses después Atari saco su conocida máquina de arcade “Pong”. En este juego de dos jugadores, cada uno controlaba una “raqueta” que se encontraba en cada uno de los bordes laterales de la pantalla. Había una bola que iba de lado a lado, y se trataba de que no tocara los bordes laterales. Cuando un jugador tocaba la bola esta rebotaba hacia el lado contrario, igual que si tocaba el borde superior o inferior. Si la bola tocaba el borde de un jugador, era punto para el otro.

En cada acción la bola hacia un ruido. Se trata de saber qué jugador ha ganado en base a los ruidos que se han realizado.

4.1.11 Problema 217: ¿Qué lado de la calle?

Trata de saber qué lado de la calle es (izquierda o derecha) sabiendo el número del portal.

4.1.12 Problema 397: ¿Es múltiplo de 3?

Un profesor dijo que un número es múltiplo de 3 cuando la suma de sus dígitos lo es. Para poner en práctica a sus alumnos sobre lo que explicó, les manda unos ejercicios. Cada uno era un valor n con el que tenían que formar un gran número a partir de la concatenación de todos los números entre 1 y n para luego indicar si el resultado era múltiplo de 3.

4.1.13 Problema 355: Gregorio XIII

Explica la historia de los años bisiestos. En el año 325 se decidió que la Pascua se celebraría el domingo siguiente a la primera luna llena posterior al equinoccio de primavera, el 21 de marzo de ese año. En 1582, este equinoccio ocurrió el 11 de marzo debido al desfase de la tierra al dar una vuelta al sol. La tierra tardaba en girar 365 días, 5 horas, 48 minutos y 45'16 segundos mientras que, en el calendario hasta ese momento, el cual tenía un bisiesto cada cuatro, tardaba 365 días y 6 horas y media. Este desfase de 11 minutos se convirtió en 10 días a lo largo de los años.

Gregorio XIII decidió cambiar la regla de los años bisiestos: un año es bisiesto si es divisible por 4, salvo que sea divisible por 100, en cuyo caso deberá ser divisible también por 400. Con este cambio, los años pasaron a tener una longitud media de 365 días, 5 horas, 49 minutos y 12 segundos. El desfase es de menos de medio minuto, es decir, un día cada 3300 años. El problema trata de conocer si la entrada es un año bisiesto.

4.1.14 Problema 219: La lotería de la peña Atlético

El Atlético de Madrid ha tenido todas sus finales en Europa en años pares: 1974, 2010, 2012 y 2014. Los aficionados del Atlético y de la numerología no es de extrañar que compren números pares de la lotería. Piden un programa que les diga que décimos de la lotería pueden comprar.

4.1.15 Problema 172: El pan en las bodas

Expone la incertidumbre que tiene Jack Dawson a la hora de saber qué pan le pertenece en una mesa de una boda. Es importante no confundirse porque es posible que provoque que algún comensal se quede sin pan. Cuando se acerca a su mesa observa que ya hay gente que ha comido pan y pregunta si podrán comer todos o no.

4.1.16 Problema 112: Radares de tramo

Explica el funcionamiento de los radares de tramo. Consiste en colocar dos cámaras en dos puntos alejados de la carretera para poder saber cuánto tiempo ha tardado el coche en recorrer ese tramo. Si la velocidad media supera a la máxima querrá decir que en algún punto del tramo ha conducido por encima de la velocidad permitida. Se debe realizar un programa que calcule la velocidad del coche e indique si este recibirá multa por superar la velocidad o perderá puntos del carné si este la superaba en más de un 20%.

4.2 Problemas de dificultad intermedia.

Los problemas de dificultad intermedia son aquellos que, aunque no sigan siendo demasiados complicados en cuanto a programación, ya se necesita un buen nivel de matemáticas y cálculo.

4.2.1 Problema 100: Constante de Kaprekar

Cuenta la historia del matemático indio Dattaraya Ramchandra Kaprekar y cómo descubrió en 1949 una curiosa característica del número 6174. Es una constante a la que siempre se llega cuando se resta un número de cuatro cifras ordenado de forma ascendente, con el mismo número ordenado de forma descendente y se repite con el resultado conseguido. Es conocido como la constante de Kaprekar y el problema pide contar cuantas iteraciones son necesarias para llegar a esta constante.

4.2.2 Problema 234: Carreras de coches

Un niño quiere jugar con sus amigos a coches de carreras. Estos necesitan dos pilas que la suma de ellas tenga un voltaje superior al indicado en la entrada. Tiene la duda de cuantos coches podrá usar.

4.2.3 Problema 337: La abuela María

María es una abuela de 106 años. Tiene quince hijos y no es de extrañar que su sonrisa ya no es lo que era. De vez en cuando, a medida que van mellando los dientes, da la casualidad de que encajan a la perfección los de arriba con los de abajo. El problema trata de averiguar si sus dientes encajan.

4.2.4 Problema 381: Alineación planetaria

Cuando los planetas se alinean en el cielo, se realiza una reunión donde vienen familias y comen juntas. Siempre que se acaba se preguntan cuando volverán a reunirse. Sabiendo el tiempo de translación de cada planeta, se debe averiguar el número de días que tardarán en volver a alinearse.

Se utiliza el *algoritmo de Euclides*, el cual es un método antiguo y eficiente para calcular el máximo común divisor.

4.2.5 Problema 382: Internet en el metro

Un alcalde ha pedido a una empresa del servicio del metro que instale antenas wifi en los túneles con la finalidad de fomentar el transporte público. Para asegurarse de que todos los ciudadanos tendrán acceso a internet en todo el túnel le ha pedido a la empresa que le indique la localización y cobertura de cada una de las antenas. Con ello se debe averiguar si existe conexión Wifi a lo largo de todo el túnel o no.

4.2.6 Problema 383: El alcance de las historias

Trata de David, un niño al que le gusta mucho la colección de libros “Elige tu propia aventura”. En esta se puede leer la historia desde varios puntos iniciales. Como le gustan los números, mira de calcular el salto más alto que ha pegado entre páginas. A esto le llama el “alcance” de sus historias y pide ayuda para poder calcularlo.

4.2.7 Problema 242: Erasμός

Este problema va de estudiantes que se van de Erasmus. Estos siempre se llevan una baraja de cartas para jugar al “mus”, un juego de más de 200 años de antigüedad. Este juego es en parejas y siempre acaban haciendo competiciones por países. En esta ocasión quieren formar parejas heterogéneas y se preguntan cuántas parejas distintas pueden formarse.

4.2.8 Problema 610: ¡No quiero irme señor Stark!

Cuenta la historia de una de las últimas películas de Marvel sobre los vengadores: Vengadores Infinity War. En esta Thanos consigue las *6 gemas del infinito* y mata a la mitad de la población.

Hay algunas teorías sobre cómo elige Thanos quien muere. Una de ellas dice que coloca a toda la población mentalmente en un círculo y cada x número de personas, mata a la siguiente. Repitiendo este proceso una y otra vez hasta que solo queda la mitad.

En este círculo también está Spiderman y Stark y se quiere saber cuál de los dos muere por si se realizará el abrazo de despedida entre estos dos.

4.2.9 Problema 607: Minimizando el castigo

A un estudiante se le castigó teniendo que escribir 100 veces una frase. Este intento buscar un atajo y se le ocurrió que podría escribir la frase una sola vez, hacer una fotocopia, recortarla y pegarla justo debajo de la original. Repitiendo este proceso de copiar y pegar hasta tener la frase 100 veces. Quiere conocer el número de operaciones de fotocopia, recorte y pegado que tiene que realizar.

4.2.10 Problema 407: Rebotando en el parchís

Todo el mundo conoce el juego del parchís y lo frustrante que es no llegar a conseguir caer exactamente en la posición final. La ficha siempre acaba rebotando. Se quiere saber la posición final que tendrá la ficha después de realizar cierta acción.

4.2.11 Problema 325: Helados de cucurucho

Trata de Alba y Blanca y el problema que tienen con los helados de cucurucho. A una de ellas le gusta el helado de vainilla y a la otra de chocolate. Cuando se compran helados de dos bolas siempre hay un sabor que estará primero y nunca se deciden cual poner. Además, cuantas más bolas haya más combinaciones de sabores se pueden hacer. El problema va de conocer todas las posibles formas de helados de cucurucho que se pueden realizar con una cierta cantidad de bolas.

4.2.12 Problema 405: Imprimiendo páginas sueltas

Este problema va de indicar el número de páginas que se quieren imprimir de la forma más corta en el cuadro de diálogo.

4.2.13 Problema 197: Mensaje interceptado

Explica el método de codificación que tiene el agente 0069: primero transforma la frase reemplazando cada sucesión de caracteres consecutivos que no sean vocales por su imagen especular y seguidamente la transforma tomando el primer carácter y el último, el segundo y el penúltimo, así sucesivamente.

Un señor llamado Fon Noiman ha conseguido descifrar su método y está intentando crear un programa para ello.

4.2.14 Problema 195: Saltos de trampolín

Trata de programar el método de evaluación de los saltos de trampolín que lleva a cabo la Federación Internacional de Natación. Hay 7 jueces evaluando y cada uno pone una nota entre 0 y 10. Al final se quitan las dos notas más altas y las dos más bajas. El resultado es igual a sumar las que quedan y multiplicando por dos.

4.2.15 Problema 608: Peligro por hielo

En este problema se ha de ayudar a un conductor a crear un aparato que avise cuándo la temperatura desciende más allá de un límite, previniendo la posible existencia de hielo en la calzada. Se avisará cuando la temperatura baje de 4 grados si previamente había estado por encima de 6 grados.

4.2.16 Problema 604: Honor y distribución

Explica el funcionamiento del juego de cartas “bridge”. En este juego cada jugador empieza con 13 cartas y cada una tendrá una puntuación dependiendo del palo y el número. Además, se suman puntos según el número de palos repetidos que tenga. Se debe crear un programa que muestre la puntuación total de las cartas.

4.2.17 Problema 108: De nuevo en el bar de Javier

Javier está realizando un estudio para investigar con qué productos gana más dinero y con cuál gana menos en su bar. Además, le gustaría saber si la venta de comidas supera la media y para ello ha creado distintas categorías. Quiere realizar un programa que le muestre la categoría que ha recaudado más dinero, la que menos, y si el dinero medio conseguido en las comidas supera la media.

4.2.18 Problema 354: Los niños primero

La normativa de los circos establece que se deben realizar simulacros de evacuación cada cierto tiempo. El circo ha colocado un medidor de altura en la salida de emergencia para analizar los datos posteriormente y conocer el número de niños que había entre el público. Se debe realizar un programa que indique el mínimo número de niños que había en el circo según los datos del medidor de altura.

4.2.19 Problema 205: Números de Lychrel

Se trata de automatizar un proceso en el cual se coge cualquier número, se le da la vuelta y se suma a sí mismo, repitiéndolo hasta llegar a un número capicúa. Los números en los que se sospecha que no se puede llegar a él se les llama *números de Lychrel*.

4.3.20 Problema 109: Liga de pádel

Explica la organización de una liga de pádel, en la cual aún anotan los resultados a mano en un cuaderno, aunque hay más de 2000 parejas distintas. Al final de la temporada tienen tanto lío que no saben quién es el ganador y piden un programa que les ayude a aclararlo.

4.3 Problemas de dificultad alta

Estos problemas son más difíciles debido a que se necesitan mejores conocimientos de programación. En estos ya se empiezan a tratar conceptos más complicados y se utilizan algunos de los algoritmos explicados anteriormente.

4.3.1 Problema 295: Elévame

Pide realizar un programa que eleve un número a otro. Para ello hay que utilizar el concepto de *Recursividad*, además del algoritmo *Divide y vencerás* ya que lo pide para números grandes.

4.3.2 Problema 262: Ada, Babbage y Bernoulli

Explica la historia de la considerada primera programadora de la historia, Ada Byron. Ada creó un algoritmo dedicado a calcular los números de Bernoulli, una secuencia de números racionales que tienen conexiones muy interesantes con teoría de números. Trata de realizar este programa usando la fórmula “Faulhaber”: suma de los n primeros números elevados a una constante p .

4.3.3 Problema 212: Operación asfalto

Trata de las calles de “Eulerandia”, las cuales han decidió asfaltar. La asfaltadora solo puede pasar una vez por cada calle ya que es tan pesada que rompería la calle recién asfaltada. Se debe realizar un programa que calcule si la máquina es capaz de asfaltar todas las calles o no.

Se trata de un problema de *puentes de Königsberg*, un célebre problema matemático muy estudiado en las asignaturas de informática. Euler llegó a la conclusión de que los puntos intermedios del recorrido deben estar conectados a un número par de líneas, si más de tres puntos tienen un número de conexiones impar es imposible solucionar el problema.

4.3.4 Problema 343: 7 de golpe

Cuenta la hazaña que hizo un sastre en la adaptación de Walt Disney del cuento de “El sastrecillo valiente”. El sastre consigue matar a 7 moscas de un solo golpe con un matamoscas. Se pide realizar un programa que calcule en cuántas posiciones se puede poner el matamoscas para matar distintas moscas de un golpe.

Este es el primer problema donde aparece el concepto de matrices y cómo se deben usar en la programación. Además, se usa la imagen integral, una técnica usada para acelerar el cálculo de operaciones que incluyan la suma del valor de los píxeles de un área.

4.3.5 Problema 189: Embarque de un transatlántico

Expone el método que usan en un transatlántico para organizar las embarcaciones de los viajeros. Para ello embarcan a los ocupantes llamándolos por cubiertas. Cada vez que embarcan, los organizadores desean saber cuánta gente queda en el puerto esperando y también a qué cubierta va alguno de los pasajeros de la cola.

En este problema ya se debe tener en cuenta el coste computacional. Se podría realizar de forma sencilla obteniendo un coste de $O(n^2)$ pero el problema no lo aceptará porque superará el límite de tiempo de ejecución. Por tanto, hay que realizar un programa con coste $O(n)$.

4.3.6 Problema 139: Números cubifinitos

Un número es cubifinito cuando al elevar todos sus dígitos al cubo el resultado es 1 o bien un número cubifinito. Dado un número, hay que realizar un programa que calcule si es cubifinito.

Aparece el concepto de *set*, que es un contenedor donde se guarda una lista ordenada de objetos únicos y servirá para saber si el número resultante ya ha salido o no. Si ha salido querrá decir que no es cubifinito.

4.3.7 Problema 230: Desórdenes temporales

Trata de un futuro en el cual se consiguió viajar en el tiempo y cómo esto afectó a las edades de las personas. Si alguien de 20 años pasa 30 años en el antiguo Egipto, al volver tendrá 50 años, aunque para los demás tenga aún 20. Por tanto, se decidió tomar medidas y conocer cuánta gente está afectada y el nivel de desorden de cada una. El desorden temporal es el número de personas que siendo más viejas que uno mismo desde el punto de vista administrativo, han vivido menos días. Se debe crear un programa que calcule el desorden temporal de todos los habitantes de una población.

Este problema usa el algoritmo *MergeSort* para poder solucionarse. Uno de los algoritmos explicados anteriormente.

4.3.8 Problema 386: Clúster de microondas

Expone el problema que tienen varios estudiantes al llegar tarde a las clases debido a las largas colas que se forman en los microondas para calentar la comida. Se debe aumentar el número de microondas, pero se ha pedido un estudio para saber cuántos se necesitarían y para mantener el tiempo de espera dentro de unos márgenes razonables.

En este problema surge el concepto de *cola de prioridad*. Un tipo de estructura de datos en la que los elementos se encuentran en un orden indicado por una prioridad asociada a cada uno.

4.3.9 Problema 342: ¡No lo puedes saber!

Trata de realizar un programa capaz de saber si es posible adivinar un número simplemente indicando si el este es “menor” o “mayor o igual” al que se busca.

4.3.10 Problema 324: Teorema del punto fijo

Se debe realizar un problema capaz de calcular el número de veces que se ha de mover una taza de café para que cada molécula recupere su lugar original. Se basa en la idea del teorema del punto fijo que explicó Brouwer con la analogía de remover una taza de café: afirmaba que en todo momento hay un punto o molécula de la taza que no habrá cambiado de lugar.

En este tipo de problema se debe conocer muy bien el funcionamiento de los “arrays” y de la recursividad.

4.3.11 Problema 598: Comienza la temporada

Expone cómo organiza un equipo las camisetas para la nueva temporada. Cuenta que aprovechan las equipaciones de los chicos de la anterior temporada. Quieren calcular el mínimo número de equipaciones nuevas a comprar para que todos los alumnos puedan jugar, teniendo en cuenta que, si no hay suficientes de una talla, pueden usar una talla más.

Este problema usa mucho la función de mapas para seguir un registro de las tallas que se tienen y que se usan.

4.3.12 Problema 319: La máquina calculadora

Javier ha construido una máquina para su hijo Luis, el cual está aprendiendo a calcular. Esta máquina muestra un número y tiene 3 botones: sumar 1, multiplicar por 2 o dividir por 3. Le ha propuesto un reto, llegar de un número inicial a uno final con los mínimos pasos posibles.

4.3.13 Problema 285: Las vacas pensantes

Trata de calcular cuánto puede comer una vaca como máximo si le toca con una que siempre empieza a comer por el cubo de los extremos que tenga más cantidad.

Este problema es uno de los más completos, ya que aparece el concepto de recursividad, mapas e iteradores.

4.3.Problema 220: ¡Pasa la calculadora!

Explica el procedimiento de un juego para dos personas en el que se empieza con una calculadora recién encendida y cada jugador, de manera alterna, suma un número nuevo de un solo dígito al valor acumulado hasta el momento. El jugador que, tras sumar su número, llega a un resultado mayor o igual que 31 pierde.

Una de las reglas es que solo se puede utilizar los números situados en la misma fila o columna que el dígito marcado por su oponente, sin poder repetir número. Se debe realizar un problema que calcule quién es el ganador de una serie de números de entrada.

Este problema es uno de los más difíciles debido a que hay que entender muy bien el funcionamiento de las matrices y de la recursividad.

4.3.15 Problema 384: El juego de la linterna

Expone el juego que utiliza Jimmy con sus sobrinos. Los sobrinos se ponen uno al lado del otro con las luces apagadas y Jimmy apuntará con una linterna a 3 de ellos aleatoriamente, estos darán un paso al frente. Los niños se quedarán sin premio si al encender la luz la altura del primero es la más baja, la del segundo la más alta y la del tercero la mediana.

El problema más difícil debido a su complejidad matemática. Además, es necesario entender muy bien la recursividad y los *vectores*: una variable que permite almacenar una colección de objetos del mismo tipo.

4.3.16 Problema 326: La ardilla viajera

Cuenta la leyenda popular del geógrafo griego Estrabón, el cual dijo que la península ibérica era tan frondosa que una ardilla podía cruzarla saltando de árbol en árbol. El problema trata de calcular si esta hipótesis es cierta reduciendo la península por una cuadrilla de $m \times n$ y comprobando si la ardilla puede saltar por los árboles colocados dentro de esta cuadrilla. Se debe decir la posición del árbol que, cuando se cortó, provocó que la ardilla no pudiera realizar la hazaña o, si desde un primer momento, no podía.

El último problema y de los más complicado. Hay que entender muy bien las matrices que representan la cuadrilla y también aparece la recursividad.

CAPÍTULO 5: Solución a los problemas realizados

En el anterior capítulo se ha indicado los problemas que se han realizado y se ha mostrado de que trata cada uno de ellos. En este, se solucionarán todos los problemas en el mismo orden. Las soluciones no contendrán el código del problema, pero si el procedimiento para poder realizarlos y una idea general de su solución.

Los problemas pueden recibir por la entrada un número de casos, por tanto, se realizará el programa mientras ese número de casos sea mayor a cero. También se puede poner como condición que el programa funcione mientras la entrada sea un número mayor a cero. Finalmente hay casos en los que el programa debe funcionar mientras haya un número en la entrada. Cada uno de estos casos tiene una base distinta de programación.

5.1 Solución a los problemas de dificultad baja

Como se ha dicho anteriormente, los problemas de este punto no contienen algoritmos complicados. Son el primer paso para dominar las bases de la programación y entender bien el funcionamiento de las distintas variables. Tienen un mínimo de dificultad matemática.

5.1.1 Problema 116: ¡Hola mundo!

Simplemente hay que mostrar por pantalla la frase “¡Hola mundo!” el número de veces que indique la entrada, es decir, mientras el número de la entrada sea mayor a cero se mostrara por pantalla la frase. Este programa tiene un coste de $O(n)$ debido a que ira aumentando de manera proporcional a la entrada.

5.1.2 Problema 114: Último dígito del factorial

Durante un número de caso de prueba hay que mostrar el ultimo digito del factorial. El truco de este problema es pensar que cuando el número que entra es mayor a cinco, querrá decir que estamos multiplicando por 2 y 5, que es lo mismo que multiplicar por 10. Cuando un número es multiplicado por 10, su ultimo digito siempre es 0. Por tanto, simplemente hay que hacer 4 condiciones cuando el número es menor a 5 y, cuando el número es mayor a 5, el resultado será siempre 0. Este programa tiene un coste de $O(1)$ ya que no varía en función del tamaño de la entrada.

5.1.3 Problema 380: ¡Me caso!

Mientras la entrada no sea 0 se irá sumando los gastos de cada caso en módulo 10^9 . Cuando se sumen todos se mostrará por pantalla el resultado. *Módulo* = *gasto*%1000000000. También tiene un coste de $O(n)$.

5.1.4 Problema 245: Los Dalton

El truco es crear dos variables “long long” para si las alturas son decrecientes o crecientes. Cada una se incrementa por separado cuando el siguiente hermano es mayor (se suma uno a crecientes) o es menor (se suma uno a decrecientes). Cuando una de las dos sea igual al número de hermanos menos 1, querrá decir que son los hermanos Dalton y se mostrará por pantalla “DALTON”, si no, se mostrará “DESCONOCIDOS”.

5.1.5 Problema 216: Goteras

La entrada son distintos casos de prueba que representan segundos. Estos segundos se pasan a horas, minutos y segundos. Para pasar de segundos a minutos se divide entre 60, donde el resultado son los minutos y el resto los segundos. Se debe tener en cuenta al mostrar por pantalla “hh:mm:ss” que si uno de los valores es menor a 10, se pondrá un 0 delante del número, es decir, si las horas son menores a 10, se mostrará por pantalla “0h:mm:ss”.

5.1.6 Problema 117: La fiesta aburrida

Se separa la entrada mediante el “sstream” por los espacios, es decir, si la entrada es “Soy Lotario”, se obtendrá “Soy” y “Lotario” por separado. Después se mostrará por pantalla “Hola, Lotario”. Se repetirá el proceso durante el número de casos indicados: $O(n)$.

5.1.7 Problema 184: El sueño de los concursantes

Se leerá el momento que se fueron a dormir y el momento que se despertaron. Si las horas están entre 0 y 10 se les sumará 24. Después se pasará todo a minutos, multiplicando las horas por 60, y se restará el minuto al que se despertaron y al que se fueron a dormir, obteniendo los minutos que se ha dormido. Luego, igual que el problema Goteras, se pasa a “hh:mm” y se mostrará por pantalla.

5.1.8 Problema 506: Tensión descompensada

Si el primer valor de la entrada es mayor o igual al segundo, se mostrará por pantalla “BIEN”, si no “MAL”.

5.1.9 Problema 512: Döner sospechoso

La entrada consta de dos valores que representan la cantidad de conejo y de caballo respectivamente. Para calcular el porcentaje de conejo dentro de la mezcla se realizará una regla de 3:

$$\frac{cant_conejo * 100}{total} = \%_{conejo}$$

5.1.10 Problema 616: Pic,Poc,Pic...Pong!

La entrada consta de los ruidos que realiza la bola. Se usarán dos booleanos que representarán el lado derecho e izquierdo. El derecho empezará en “true” y el izquierdo en “false”, ya que la bola empieza a ir a la derecha. Si se lee “PIC” quiere decir que la bola rebota y se intercambiarán los booleanos. Si se lee “PONG”, se sumará punto al contador del booleano que se encuentra en “false”.

5.1.11 Problema 217: ¿Qué lado de la calle?

Si el número leído es par, se muestra por pantalla “DERECHA”, si no, “IZQUIERDA”.

5.1.12 Problema 397: ¿Es múltiplo de 3?

El truco es ver si el número leído, cuando se divide por 3, el resto es 0 o 2. Si es así, querrá decir que es múltiplo de 3 y se mostrará “SI”, en caso contrario se mostrará “NO”.

5.1.13 Problema 355: Gregorio XIII

Durante un número de casos se leerán distintas fechas. Si esta fecha no es divisible por 4, se pondrá un booleano a “false”. Si es divisible por 4 pero no por 100 el booleano será “true”. Si es divisible por 4, por 100 y por 400 el booleano también será “true”, si no es divisible por 400 será “false”. Si el booleano es “false”, se muestra por pantalla 28. Si es “true” se mostrará 29.

5.1.14 Problema 219: La lotería de la peña Atlética

Si el número de la entrada es par se sumará uno al contador de décimos a comprar. Al acabarse la entrada se mostrará por pantalla el número total de décimos que pueden comprar.

5.1.15 Problema 172: El pan en las bodas

Para lograr hacer el programa lo más eficiente posible se realiza de la siguiente manera: Si uno de la mesa come por la derecha y otro por la izquierda siempre habrá alguien que se quedará sin comer pan. Para lograrlo se usan variables que tendrán valor 1 si comen por la derecha o por la izquierda. Si las dos valen 1 se muestra por pantalla “ALGUNO NO COME”, si solo uno de las dos tiene valor 1, se muestra “TODOS COMEN”.

5.1.16 Problema 112: Radares de tramo

Se calcula la velocidad del coche mediante la ecuación $v = \frac{\text{espacio}}{\text{tiempo}}$ y esta se pasará a kilómetros por hora multiplicándola por 3,6. Si la velocidad supera en más de un 20% a la máxima se mostrará por pantalla “PUNTOS” si solo la supera, “MULTA” y si es menor a la máxima permitida se mostrará “OK”. Si alguno de los valores de la entrada que indica distancia, tiempo o velocidad máxima es menor a 0 se muestra “ERROR”.

5.2 Solución a los problemas de dificultad intermedia.

La base de estos problemas es la misma que en los anteriores, pero en estos ya se deben realizar más procesos para poder llegar a la solución.

5.2.1 Problema 100: Constante de Kaprekar

Este problema consta de 4 pasos:

1. Transformar los números enteros a “strings”

Para transformarlo se usa la librería “sstream”.

2. Ordenar el “string” de números de menor a mayor y de mayor a menor por separado

Para ordenar el “string” se usa la función *sort*, que permite ordenar un array indicándole el inicio y el fin de este.

3. Restar los números

4. Repetir el paso 2 y 3 mientras el número que resulte no sea igual a 6174 (Kaprekar)

En cada iteración se sumará uno al contador y al final se mostrar por pantalla este contador.

5.2.2 Problema 234: Carreras de coches

La entrada consta de diversos números que indican la carga de cada pila. El primer paso será ordenar estos números de menor a mayor mediante la función *sort*. Después se recorrerá el array con dos índices que apuntarán al inicio y al final de este. Si la suma de los dos índices es mayor al voltaje necesario se sumará uno al contador de coches y se moverán los dos índices. Si no es mayor, se moverá el índice del inicio. Cuando los dos índices apunten al mismo número se acabará el proceso y se mostrará por pantalla el valor del contador.

5.2.3 Problema 337: La abuela María

Se inicializa un booleano a “true”. La entrada consta de dos “arrays” de enteros que indican las alturas de los dientes superiores e inferiores respectivamente. Se calculará la diferencia que hay entre los dos dientes superiores e inferiores consecutivos. Si la diferencia es distinta, se pondrá el booleano a “false”. Después de mirar todos los dientes, si el booleano es “false” quiere decir que no encajan y se mostrará por pantalla “NO”, en caso contrario se mostrará “SI”.

5.2.4 Problema 381: Alineación planetaria

Se debe realizar el **mínimo común múltiplo** de todos los tiempos de translación de los planetas. Este será igual a $\frac{a*b}{mcd(a,b)}$ donde ‘a’ y ‘b’ son tiempos de translación de dos planetas distintos. Como el “mcm” se debe acumular, la ecuación es igual a la siguiente:

$$\frac{mcm*b}{mcd(mcm,b)}$$

El máximo común divisor se calcula mediante el *algoritmo de Euclides*, un algoritmo recursivo que tiene como *caso base* devolver ‘a’ cuando ‘b’ sea 0 y, como *caso recursivo*, calcular el máximo común divisor de b con el resto de dividir ‘a’ por ‘b’.

Al final se mostrará el mínimo común múltiplo resultante. Este algoritmo tiene coste **O(logn)** debido a que es recursivo.

5.2.5 Problema 382: Internet en el metro

Este problema trata de intervalos. Cada antena tiene un rango de cobertura distinto. Cuando se mira el intervalo de la siguiente antena se comprueba si estas dos intersecan, si es así, se mira si el rango aumenta por arriba o por abajo. Al final se obtendrá un rango final, el cual debe estar entre 0 y la longitud total del túnel. Si el rango es el que se busca se mostrará por pantalla “SI”, en caso contrario “NO”.

5.2.6 Problema 383: El alcance de las historias

Se inicializa un valor máximo como el primer número de la entrada y un valor de salto a -300000. Cada vez que se lee un valor, se calcula el salto con el valor máximo: $salto = valor_{máximo} - número_{leído}$. Si el salto es mayor al que se tiene como salto máximo, este se actualiza y, si el valor leído es mayor al valor máximo, este también se actualizará. Al final se mostrará por pantalla el salto máximo

5.2.7 Problema 242: Erasμός

La entrada son varios números que indican la cantidad de personas de cada país, se debe tener en cuenta que, si solo hay un país, hay que leer la entrada y mostrar por pantalla un 0. Este problema se puede solucionar de dos formas distintas:

1. De derecha a izquierda: inicializar una variable *suma* al último número de la entrada. De manera recurrente ir sumando la suma con número siguiente de la lista, $suma = suma + número$.
2. Sumar todo el array de enteros sobre dos (todas las parejas posibles): $suma = \binom{suma}{2}$. Al resultado se le restan las parejas que se pueden hacer del mismo país. Mostrar por pantalla el resultado.

5.2.8 Problema 610: ¡No quiero irme señor Stark!

Realizar una cola de números. Durante k personas, coger el último valor de la cola, eliminarlo de ella y volverlo a meter en la cola por encima (rotamos las personas). Luego coger el último valor de la cola y, si coincide con la posición en la que estaba Spiderman o Stark, se pone a “true” el booleano correspondiente al mismo (inicialmente en “false”). Repetir el proceso mientras el tamaño de la cola sea mayor a la mitad de la población.

Al finalizar, si Spiderman muere y Stark sobrevive se mostrará por pantalla “¡No quiero irme señor Stark!”. Si muere Stark y sobrevive Spiderman se mostrará “¡No quiero irme, Peter!” y, si no sobrevive ninguno de los dos, saldrá por pantalla “No hay abrazo”.

5.2.9 Problema 607: Minimizando el castigo

El truco de este programa es utilizar la función *pow*, la cual sirve para elevar un número a otro. Se irá elevando 2 a “x” (un número entre 0 y 34) y, si el resultado es mayor o igual al número de veces que debe escribir la frase del castigo, se mostrará por pantalla el valor de “x”.

5.2.10 Problema 407: Rebotando en el parchís

El primer paso es sumar a la posición en la que estaba la ficha el valor de la tirada que se lee por teclado. Si el resultado es mayor a la posición máxima, se le debe restar a la posición máxima el valor del resultado menos la posición máxima. Así dará la posición final después de rebotar. Por pantalla se mostrará el valor del resultado.

5.2.11 Problema 325: Helados de cucurucho

Inicializar un array con valores 0. A partir del valor de chocolate leído por teclado se meterán “1s” al array. Crear otro array de dos posiciones, en la primera tendrá una C y en la segunda una V. Crear un segundo array igual que el primero, pero donde haya 0 se meterá una V y donde haya 1, una C. Se mostrará el array por pantalla cada vez que se pueda permutar una C con una V usando la función *next_permutation*.

5.2.12 Problema 405: Imprimiendo páginas sueltas

El primer paso de este problema es inicializar un booleano a “false” y un entero auxiliar al primer número leído. Después se leerá el segundo número y se repetirán los siguientes pasos mientras haya números para leer:

1. Si el número auxiliar es el anterior al número leído se pone a “true” el booleano.
2. Si no se cumple la condición anterior, se mirará si el booleano está a “true” o “false”.
3. Si el booleano es “true”, se muestra el rango, que será igual al primer número leído más el auxiliar. Además, ahora el primer número leído será igual al segundo y se pondrá el booleano a “false”.
4. Si el booleano es “false”, se muestra el rango, que será igual al número auxiliar. Además, ahora el primer número leído será igual al segundo.
5. Se iguala el número auxiliar al segundo número
6. Si el número leído es 0 se corta el bucle.

5.2.13 Problema 197: Mensaje interceptado

Se pasará del mensaje leído por teclado a su forma original con ayuda de “strings” auxiliares. Primero se meterá la primera letra en un “string”, la segunda en otro, la tercera en el primer “string”, así sucesivamente. Al acabar se sumarán los dos. El segundo paso será recorrer el “string” resultante guardando a la vez lo que se lee y, cuando aparece una vocal (en minúsculas), se gira lo leído añadiéndole la vocal. Al final se obtendrá el mensaje escrito correctamente.

5.2.14 Problema 195: Saltos de trampolín

Se leen los 7 valores y se ordenan de menor a mayor con la función *sort*. Se toman solo los valores 2,3 y 4, sumándolos y multiplicándolos por 2. Se muestra el valor resultante.

5.2.15 Problema 608: Peligro por hielo

Inicializar un booleano a “true”. La primera condición determinará si el número es menor o igual a 4 y si el booleano está a true, sumando uno al contador de avisos y poniendo el booleano a “false” si esta se cumple. Se creará otra condición que pondrá el booleano a “true” si el número es mayor a 6. Finalmente se mostrará el número total de avisos.

5.2.16 Problema 604: Honor y distribución

Se creará un contador para cada palo de carta y “strings” para el número y la letra de las cartas. Se lee el número y el palo, se suma uno al palo correspondiente y a la puntuación total que corresponde cada uno. Al acabar se mira el número de palos que se tiene de cada uno, sumando los respectivos puntos. Luego se muestra por pantalla la puntuación total.

5.2.17 Problema 108: De nuevo en el bar de Javier

Mientras haya entrada para leer se deben realizar distintas tareas:

1. Si la letra que se lee no es una N, sumamos la venta de esta letra en un registro (mapa) y se suma 1 al contador de categorías.
2. Si la letra es una A, se suma 1 al contador de comidas.
3. Cuando la letra sea una N, se recorrerá todo el mapa buscando los valores máximos y mínimos, los cuales son variables inicializadas al primer valor.
4. Si se encuentra un valor mayor al máximo o menor al mínimo, estos se actualizarán.

5. Si se encuentra un valor igual al máximo o al mínimo, se pondrá un booleano a “true”.
6. Al acabar se mirará si el valor mínimo y máximo son iguales, si es así se pondrán los booleanos a “true”.
7. Si el booleano de máximo es “true”, mostrará por pantalla “empate”. Si no, mostrara la categoría del valor máximo.
8. Si el booleano de mínimo es “true”, mostrará por pantalla “empate”. Si no, mostrara la categoría del valor mínimo.
9. Si la media de dinero ganado en las comidas supera a la media del dinero total, se mostrará por pantalla un “SI”, en caso contrario un “NO”.

5.2.18 Problema 354: Los niños primero

Se crearán dos variables, una equivalente a la altura máxima de las personas (*max*) y otra equivalente a la altura máxima de los niños (*alt_max*). Si la altura leída es mayor a la máxima se actualiza, si es menor o igual a la altura máxima se igualará *alt_max* al valor de *max*. Luego se mirarán otra vez las alturas, si es menor a *alt_max* se sumará uno al contador de niños. Finalmente se mostrará ese contador final por pantalla.

5.2.19 Problema 205: Números de Lychrel

Se crea una variable total que se inicializa con el número leído. A este total se le suma el número invertido. Para invertir un número se toma ayuda de una variable auxiliar, la cual empieza con el valor 0 y se le va sumando el resto de dividir el número por 10, después se repite con el número dividido por 10. El resultado se repite hasta encontrar un número capicúa, que se mostrará por pantalla. Si el número llega a ser mayor a 1000000000 se mostrará por pantalla “Lychrel?”.

5.3.20 Problema 109: Liga de pádel

Mientras la categoría que se lee por teclado no sea “FIN” y el nombre del primer equipo no sea “FIN”, se lee el nombre del primer equipo, del segundo equipo y los sets ganados de cada uno. Se les suma los puntos correspondientes al equipo que haya ganado más sets y, mediante un mapa de registro, se guarda esta información.

Al acabar de leer la entrada, se recorrerá el mapa buscando el equipo con una puntuación más alta, Si dos de ellos tienen la misma puntuación se pondrá a “true” un booleano. Al finalizar, si el booleano estaba en “true” se muestra por pantalla “EMPATE”, si no, el nombre del equipo ganador.

Para calcular el número de partidos que no se han jugado, se suman el número de equipos y se multiplica por el mismo número restándole 1.

5.3 Solución a los problemas de dificultad alta

La solución de estos problemas ya implica conocimientos técnicos de programación, como pueden ser los algoritmos explicados en el *CAPÍTULO 3*.

5.3.1 Problema 295: Elévame

Se debe utilizar una función recurrente de la cual el *caso base* será: si n es 0, devolver 1 (cualquier número elevado a 0 es 1). El *caso recurrente* dependerá de si el número al que elevamos es divisible por dos. Si lo es, se hará la recurrencia elevando a la mitad y devolviendo el valor multiplicado por sí mismo. Si es impar se hará la recurrencia elevando a un número menos y devolviendo el número multiplicado por el anterior. Todo se hará en modulo 31543 como indica el problema.

5.3.2 Problema 262: Ada, Babbage y Bernoulli

Se usará la misma base que el problema anterior, simplemente se deberán ir sumando los resultados de cada número elevado. Se mostrará el resultado final por pantalla.

5.3.3 Problema 212: Operación asfalto

Cada vez que se lea un número, se sumará uno a la cantidad de intersecciones del mismo. Si al final hay más de 3 vértices con un número de intersecciones impar se mostrará por pantalla “NO”, en caso contrario se mostrará “SI”.

Este problema no es complicado de programar, pero sí que hay que conocer el concepto de *puentes de Königsberg*.

5.3.4 Problema 343: 7 de golpe

La idea principal es crear una matriz de 1001 por 1001 dispuesta de forma horizontal (una columna y una fila de más formada por todo 0). Se leerá la matriz de entrada y por cada “X” que se encuentre se sumará uno al contador de moscas.

En la matriz del programa, cada celda contendrá el número de moscas que hay desde la celda inicial hasta la actual.

Para saber cuántas moscas mueren en la cuadrícula del tamaño del matamoscas es tan fácil como coger el número contenido en la celda de abajo a la derecha del matamoscas, restarle las celdas superior derecha e inferior izquierda de una cuadrícula igual al tamaño del matamoscas más uno y sumarle la celda superior izquierda de la misma. Se repetirá el proceso durante toda la matriz, moviendo el matamoscas. Al final se mostrará el número de posiciones en las que mueren 0 moscas, 1 moscas, etc. Hasta un total de 7 moscas de golpe.

5.3.5 Problema 189: Embarque de un transatlántico

Se debe utilizar un mapa para llevar un registro de las cubiertas que están ocupadas (si tiene valor 1 está ocupada), además de crear un array con el número de cubierta de cada persona. La entrada indica la cubierta en la que se hospeda cada pasajero. A medida que se lee la entrada se actualizan los registros.

Si la acción es un “EMBARQUE”, se mirará si esa cubierta está ocupada o no, si lo está, se pondrá a 0 (ya no estará ocupada) y se mirará cuantas personas van a esa cubierta. Por cada persona que embarque se sumará uno al desplazamiento y se quitará uno al número de personas restantes. Si no está, se debe actualizar el array de cubiertas mediante el desplazamiento. Finalmente se mostrará el número de pasajeros que han embarcado. Si la acción es una “CONSULTA”, se mostrará la cubierta en la que está esa persona.

5.3.6 Problema 139: Números cubifinitos

Se calcula el total de todos los dígitos que forman el número al cubo mediante la función *pow*. Si el número resultante es 1, se muestra el total más “cubifinito”. Si no, se debe llevar un registro de los números que van saliendo mediante la variable *set*. Si el número ya ha salido se debe mostrar el total más “no cubifinito”, sin embargo, si el número no ha salido se debe meter en el registro y repetir el proceso hasta que ocurra alguno de los dos casos anteriores.

5.3.7 Problema 230: Desórdenes temporales

Se crean dos “arrays” iguales a la entrada y se les hace el algoritmo *MergeSort*. En este se ordenarán las edades en partes, primero la primera mitad y después la segunda (de manera recurrente). Al final se realizará el algoritmo *Merge* para juntar las dos partes ordenadas. Mientras se juntan las partes se irán calculando el número de desórdenes temporales. Para ello se le sumará la diferencia de posiciones que está un número de la edad más pequeña. Al final se mostrará por pantalla el total de desórdenes temporales.

5.3.8 Problema 386: Clúster de microondas

En este problema se debe crear una cola de prioridad del tiempo que tarda cada microondas. Se crean dos “arrays” con el momento de llegada de los alumnos y el tiempo que tardad cada uno. Mientras el valor superior de la lista y el inferior sea mayor a 1 se deben hacer distintas comprobaciones: Si se está en la primera mitad (inferior más superior entre 2) de la lista de microondas, en la cola de prioridad se mete la suma de los dos “arrays”. Si se está en la segunda mitad, se mirará si el primer valor de la cola menos el instante de llegada es mayor al tiempo máximo de espera. Si es mayor se pondrá un booleano a “false”, si no, se debe mirar si el primer valor de la cola es menor al tiempo de llegada. Si se cumple la condición, se saca el valor de la cola y se mete el nuevo, si no, se saca el valor de la cola y se le mete el mismo valor sumándole el tiempo de llegada.

Si el booleano es “false” se repetirá lo anterior sustituyendo el valor inferior por la mitad, en caso contrario, se sustituirá el valor superior por la mitad. Cuando ya no se cumpla la primera condición, se mostrará por pantalla el valor de la mitad correspondiente, la cual indicara el número de microondas necesarios.

5.3.9 Problema 342: ¡No lo puedes saber!

Inicializar una variable al valor inicial y otra al valor final más uno. Si el valor leído por teclado está entre los dos valores anteriores, se deberá mirar si este es menor al que se busca o mayor o igual, actualizando los valores inicial y final. Si al acabar, el valor inicial es igual al número anterior al final, querrá decir que lo puede saber y se mostrará por pantalla “LO SABE”, si no, se mostrará “NO LO SABE”.

5.3.10 Problema 324: Teorema del punto fijo

Se tiene un array de enteros donde el valor de la posición “x” es la posición a la dentro del mismo array. Se debe ir moviendo dentro del array a las posiciones que marca y se pondrá valor 0 en las que se ha pasado, sumando uno al contador de pasos. Cuando se lea un valor 0 querrá decir que ya se ha dado la vuelta. Se repetirá hasta que todos los valores del array sean 0 (se irán guardando el número de pasos realizados en cada caso). Se mostrará por pantalla el **mínimo común múltiplo** de todos los pasos mediante la función explicada anteriormente en el Problema 381: Alineación planetaria.

5.3.11 Problema 598: Comienza la temporada

Se crea un mapa con el registro del número de camisetas que se necesitan de una talla y el número de camisetas que ya disponen de esa talla. Durante todas las tallas, se cogerá el mínimo entre la cantidad de camisetas que se necesitan de esa talla y de las que hay y se restará al total. Si aún se necesitan camisetas de esa talla se hace lo mismo con las camisetas de una talla superior y, si aún se necesitan, se sumarán al contador de camisetas que se deben comprar. Al final se mostrarán por pantalla el número de camisetas que hay que comprar.

5.3.12 Problema 319: La máquina calculadora

Crear una pila añadiendo el valor inicial de la entrada y un valor nulo para simbolizar la siguiente fila. En bucle se irá sacando el primer valor de la cola, si es el valor nulo, se vuelve a meter al final de esta. Si se saca otro valor hay que sumarle uno, multiplicarle 2 y dividirlo 3 de manera separada. Si alguno de los anteriores resultados da el número que se busca, se termina, si no, se mira si ha salido. Si no ha salido se mete en la lista de números que han salido y se meten al final de la cola. Se repite hasta encontrar el valor que se busca.

5.3.13 Problema 285: Las vacas pensantes

Se debe crear un mapa para tener un registro de los casos que van ocurriendo. Se creará una función recurrente en el que *el caso base* será: si ya se ha pasado por este caso, devolver el máximo de comida que se ha comido. *La recurrencia* será comiendo por la izquierda o comiendo por la derecha de la fila de cubos, indicando como será la fila de cubos si come la vaca Devoradora primero, la cual siempre come el cubo más lleno de los extremos. Al final se mostrará el máximo de comida que puede comer la vaca.

5.3.14 Problema 220: ¡Pasa la calculadora!

Se debe inicializar la calculadora con qué botones puede pulsar según el dígito que le llega y crear una matriz de todos los posibles valores que pueden llegar.

Se crearán dos funciones recurrentes entre ellas para cada persona que juega. Cuando juega la persona 1, se mira si el número que le ha llegado es mayor a 31, en cuyo caso devolverá “true” porque ha ganado. Si la combinación de números que le han llegado ya se ha calculado, devolverá “true” o “false” si con esta combinación gana. Le mandará al segundo jugador las 4 combinaciones posibles de botones que puede pulsar a partir de los que le han llegado. Si el segundo jugador pierde, guardará la combinación como ganadora, si no, como perdedora.

El jugador 2 le mandará al jugador uno las 4 posibles combinaciones que puede hacer con lo que le ha llegado.

Al final se mostrará por pantalla si con la combinación de la entrada “GANA” o “PIERDE”.

5.3.15 Problema 384: El juego de la linterna

Se basa en crear un registro de intervalos. Cada vez que entra un número se mira si está entre el intervalo de los dos números anteriores. Puede haber varios casos:

1. El nuevo número no está en el intervalo, pero sí que hace que este intervalo crezca, por tanto, hay que actualizarlo
2. El nuevo número no está en el intervalo ni hace que este crezca, pero, con el siguiente número, crea un intervalo nuevo.
3. El nuevo número está entre alguno de los intervalos, en este caso siempre fallará.
4. Acaban todos los números y ninguno está entre los intervalos (que se han ido actualizando a lo largo de la entrada). En este caso siempre habrá premio.

5.3.16 Problema 326: La ardilla viajera

Se debe crear una matriz donde se irán colocando los árboles uno por uno hasta que, cuando se ponga uno, éste haga posible llegar del inicio al final. Cada vez que se coloca un árbol se pone su casilla a uno y se busca alrededor de él en una distancia igual al salto que puede hacer la ardilla. Si hay un 1 en alguna celda quiere decir que hay un árbol y, por tanto, se puede saltar de uno a otro. Si no hay 1, se pone un valor de 2 a cada celda alrededor del árbol (significa que este árbol es accesible desde otro). Se repite el proceso con cada árbol hasta que uno haga posible llegar del inicio al final saltando por cada uno. Este último será la posición del árbol que, si se tala, no se podrá llegar al final. Se mostrará por pantalla su posición.

Si la ardilla nunca pudo llegar al final (se han colocado todos los árboles y ninguno ha hecho posible llegar saltando al final), se mostrará por pantalla “NUNCA SE PUDO”.



CAPÍTULO 6: Conclusión

Este trabajo de fin de grado es muy útil para aquellas personas interesadas en la programación competitiva o en el mundo de la programación. Explica las bases necesarias que se deben aprender y contiene más de 50 problemas con los pasos para poder solucionarlos y así ser capaces de practicar.

La programación competitiva es un mundo muy extenso, aunque no tengas interés en competir, si quieres convertirte en un ingeniero de software es importante conocer muchos algoritmos, los cuales se usan en la programación competitiva.

Además, las empresas multinacionales del mundo de la informática como puede ser Google o Facebook utilizan problemas propios de una competición de programación en sus entrevistas de trabajo. Por tanto, es muy útil si tu objetivo es acceder a una de estas empresas.

Para mí uno de los objetivos es entrar en Google trabajando como programador, por tanto, me será muy útil tomar el mundo de la programación competitiva como base para impulsarme en la ingeniería de software.

BIBLIOGRAFÍA

- [1], [2] Tim Roughgarden, “*Algorithms Illuminated*”. Pág. 36.
- [3] Medium - ¿Qué es la programación competitiva? <https://medium.com/elemental-school/qu%C3%A9-es-programaci%C3%B3n-competitiva-y-por-qu%C3%A9-deber%C3%ADa-comenzar-a-aprenderla-66f44dff53f3>
- [4] Wikipedia – Competición internacional de programación. https://es.wikipedia.org/wiki/Competici%C3%B3n_Internacional_Universitaria_de_Programaci%C3%B3n
- [5] ICPC web – About ICPC. <https://icpc.global/regionals/abouticpc>
World Finals Rules. <https://icpc.global/worldfinals/rules>
- [6] SWERC web – About SWERC. <https://swerc.eu/2022/about/>
- [7] Wikipedia – IBM. <https://es.wikipedia.org/wiki/IBM>
Association for Computing Machinery. https://es.wikipedia.org/wiki/Association_for_Computing_Machinery
- [8] IOI web – History of the IOI. <https://ioi.te.lv/history.shtml>
Rules of the IOI. <https://ioi.te.lv/rules/regulations18.pdf>
- [9] Wikipedia – IOI. https://en.wikipedia.org/wiki/International_Olympiad_in_Informatics
- [10] Olimpiada informática española – Sobre la IOE. <https://olimpiada-informatica.org/oie>. <https://olimpiada-informatica.org/portada>



BIBLIOGRAFÍA

- [11] AdaByron web – Historia de AdaByron. <https://ada-byron.es/historia.php#:~:text=La%20idea%20de%20los%20concursos,1977%20impulsado%20por%20la%20ACM>.
I edición AdaByron. <https://ada-byron.es/2015/index.php>
¿Qué es AdaByron? <https://ada-byron.es/quees.php>
Preguntas frecuentes. <https://ada-byron.es/faq.php>
Los problemas de AdaByron. <https://ada-byron.es/syllabus.php>
Premios edición 2022. <https://ada-byron.es/2022/nac/premios.php>

- [12] Wikipedia – Facebook Hacker Cup.
https://es.wikipedia.org/wiki/Facebook_Hacker_Cup

- [13] Facebook web – Términos de servicio Facebook Hacker Cup.
<https://www.facebook.com/codingcompetitions/hacker-cup/terms>
About Meta Hacker Cup.
<https://www.facebook.com/codingcompetitions/hacker-cup>

- [14] CodeForces – Página principal. <https://codeforces.com/>

- [15] ¡Acepta el reto! – Página principal. <https://acceptaelreto.com/>