UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

PROGRAMACIÓN PARA COMPETICIÓN AVANZADO

MANUAL DE INICIO EN OMEGAUP

Responsable de Asignatura
BEJAR GONZALES VICTOR HUGO

Estudiante
WILLIAM ALAVE COLQUE

Puno, Perú

08/01/2026

Índice

1 AUTENTIFICACIÓN Y ACCESO AL SISTEMA

El ingreso a la plataforma **omegaUp** representa el punto de partida para cualquier competidor de alto rendimiento. Este proceso garantiza la persistencia de los avances y la participación en eventos oficiales.

1.1 Portal de Inicio

Para comenzar diríjase a la dirección `https://omegaup.com/`. En la interfaz principal deberá localizar el control de acceso para iniciar su sesión de trabajo.

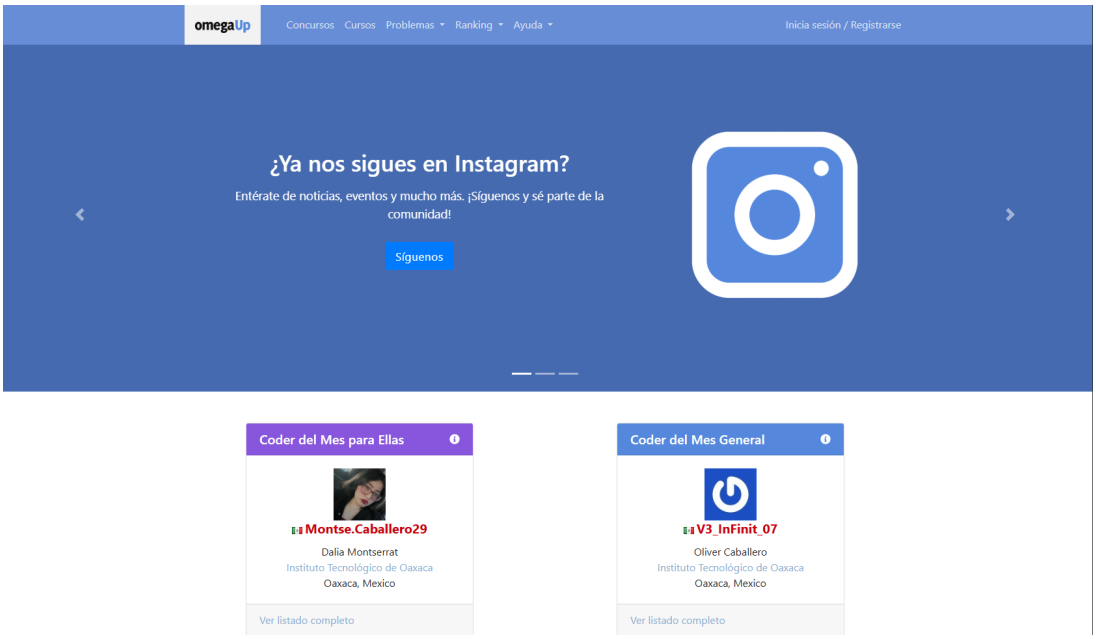


Figura 1: Interfaz global de acceso al ecosistema omegaUp.

1.2 Creación y Autentificación de Cuentas

El sistema permite la validación mediante diversos métodos para facilitar la integración del usuario. Es posible utilizar credenciales locales o servicios de terceros.

- **Registro Manual:** Proceso estándar mediante el ingreso de un nombre de usuario y correo electrónico.
- **OAuth Google:** Método acelerado que vincula la cuenta institucional directamente con el perfil de programador.

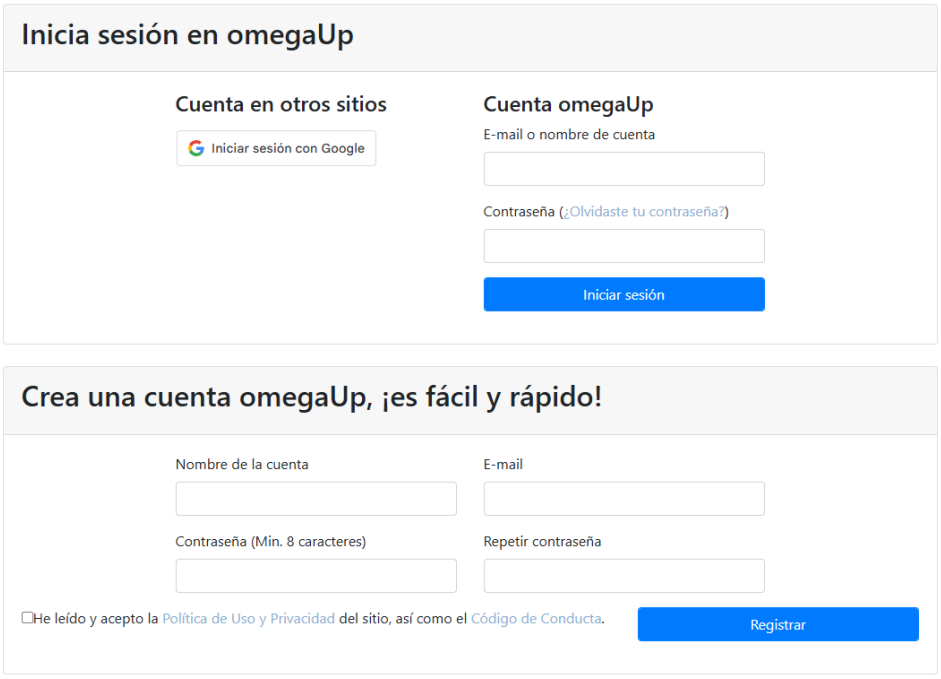


Figura 2: Formulario detallado para la gestión de nuevas cuentas y acceso social.

2 NAVEGACIÓN Y GESTIÓN DE PROBLEMAS

La arquitectura de **omegaUp** se fundamenta en un repositorio masivo de retos técnicos. La navegación efectiva por estos menús es vital para localizar el material de estudio adecuado según el avance del sílabo.

2.1 Menú Principal y Exploración

Para iniciar la búsqueda de retos debe interactuar con el menú superior de la plataforma. Al desplegar la opción **Problemas** se presentan tres rutas principales de acceso:

- **Explorar Problemas:** Dirige al usuario a una interfaz visual basada en colecciones y categorías temáticas.
- **Lista de Problemas:** Muestra el catálogo global en formato de tabla con filtros avanzados de búsqueda.
- **Ver Últimos Envíos:** Permite monitorear la actividad de la comunidad en tiempo real para observar qué retos están siendo resueltos actualmente.

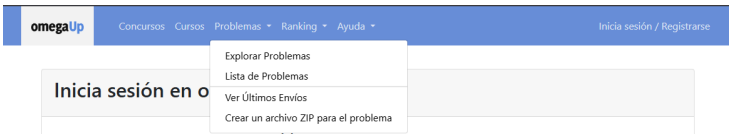


Figura 3: Desglose de opciones del menú de navegación superior.

2.2 Colecciones por Nivel de Competencia

La sección de colecciones es el núcleo educativo del sistema. Aquí los problemas se encuentran pre-clasificados para guiar al estudiante de forma progresiva. Dentro de esta interfaz se distinguen bloques claramente diferenciados:

- **Nivel Básico Karel:** Bloque diseñado para desarrollar el pensamiento lógico-matemático mediante instrucciones espaciales. Contiene aproximadamente **73 problemas** fundamentales.
- **Nivel Intermedio:** Se enfoca en matemáticas aplicadas a la programación y estructuras de datos elementales con un total de **131 problemas** técnicos.
- **Nivel Avanzado:** Es el entorno principal para la asignatura de **Programación para Competición Avanzado**. Este bloque agrupa retos de alta complejidad algorítmica y optimización extrema.

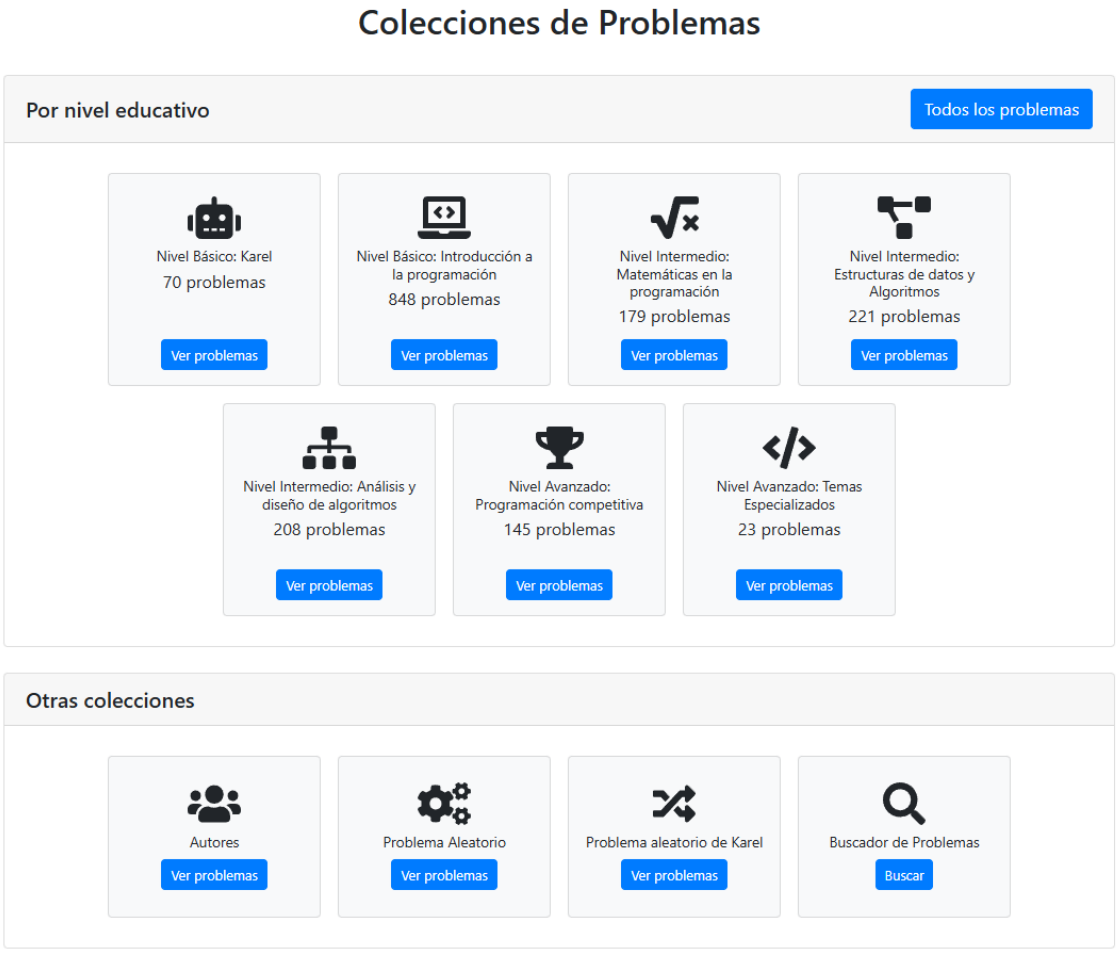


Figura 4: Distribución de los pilares formativos por niveles de dificultad.

2.3 Análisis del Listado Maestro de Problemas

Al acceder a la lista completa se presenta una tabla técnica de alta densidad informativa. Cada fila representa un reto y contiene las siguientes columnas de datos:

- **ID y Título:** Identificador único del problema y su nombre oficial.
- **Nivel y Etiquetas:** Muestra el nivel asignado y los temas específicos como **Ciclos, Arreglos o Búsqueda Binaria**.
- **Calidad y Dificultad:** Indicadores basados en la retroalimentación de la comunidad y el análisis del juez.
- **Ratio:** Porcentaje de éxito que indica la relación entre envíos totales y soluciones aceptadas.

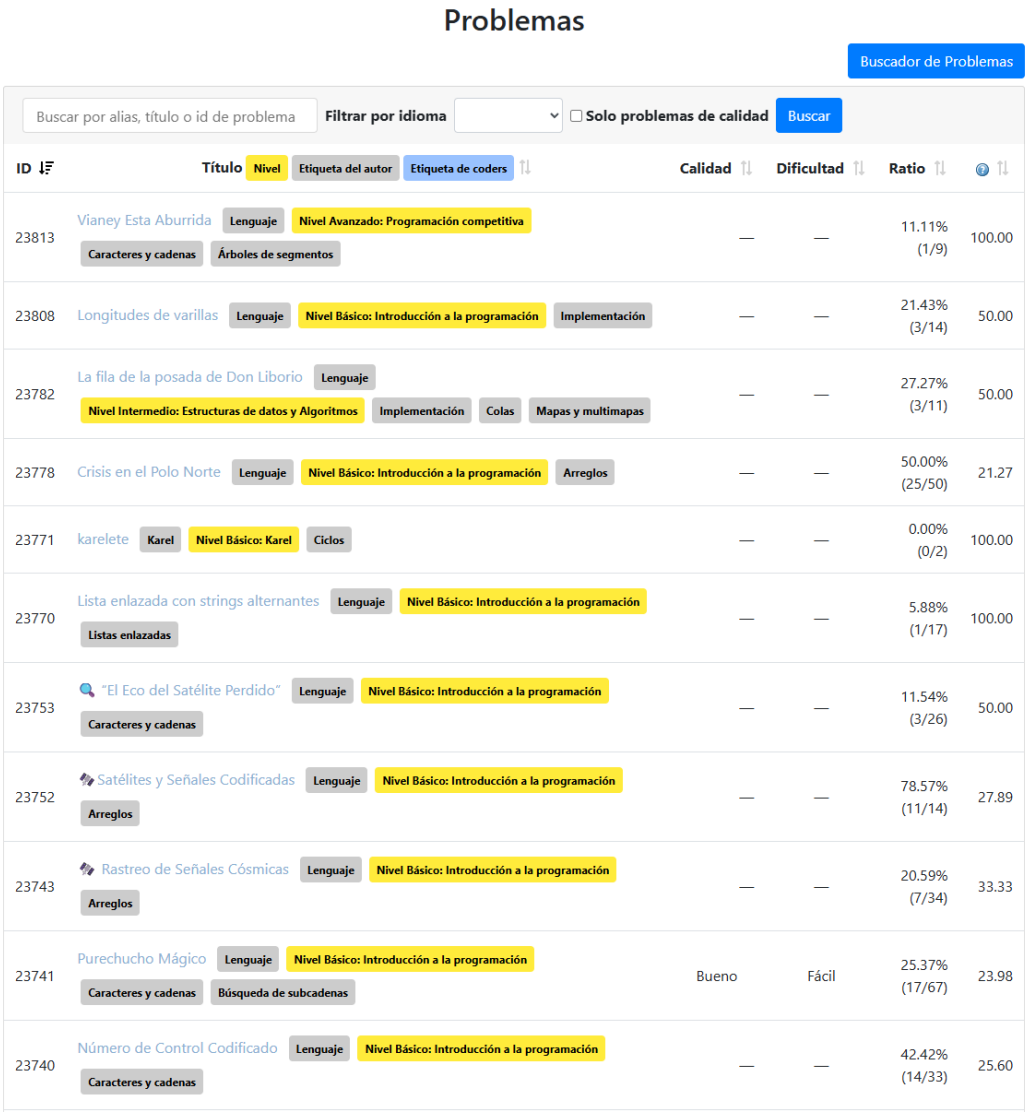


Figura 5: Detalle de la tabla de problemas y métricas de resolución comunitaria.

2.4 Interfaz de Filtros Laterales

En la parte izquierda de la lista de problemas se encuentra el panel de filtrado dinámico. Este panel permite segmentar los retos mediante los siguientes criterios:

- **Etiquetas Temáticas:** Permite buscar problemas que requieran algoritmos específicos como **Recursión o Toma de decisiones**.
- **Selector de Dificultad:** Filtra los resultados para mostrar únicamente retos de nivel **Fácil, Medio o Difícil**.
- **Filtro de Calidad:** Asegura que el estudiante trabaje únicamente con problemas que poseen descripciones claras y casos de prueba consistentes.

2.5 Exploración Detallada por Niveles

A continuación se detallan las vistas específicas que el estudiante encontrará al navegar por las colecciones principales del sistema.

Figura 6: Interfaz detallada del Nivel Básico enfocado en lógica con Karel.

Figura 7: Entorno de problemas de Nivel Intermedio con enfoque matemático.

2.6 Gestión de Concursos Actuales

La pestaña de **Concursos** es el área destinada a la evaluación bajo presión de tiempo. Dentro de este módulo el usuario puede observar los eventos próximos, actuales y pasados.

1. **Identificación del Concurso:** El sistema muestra el nombre del evento y el organismo que lo coordina.
2. **Cronograma:** Se detalla la duración exacta y el tiempo restante para el cierre de la competencia.
3. **Botón de Registro:** Una vez seleccionado el concurso es imperativo hacer clic en el botón **Ver detalles** para confirmar la participación y acceder a los problemas exclusivos del evento.

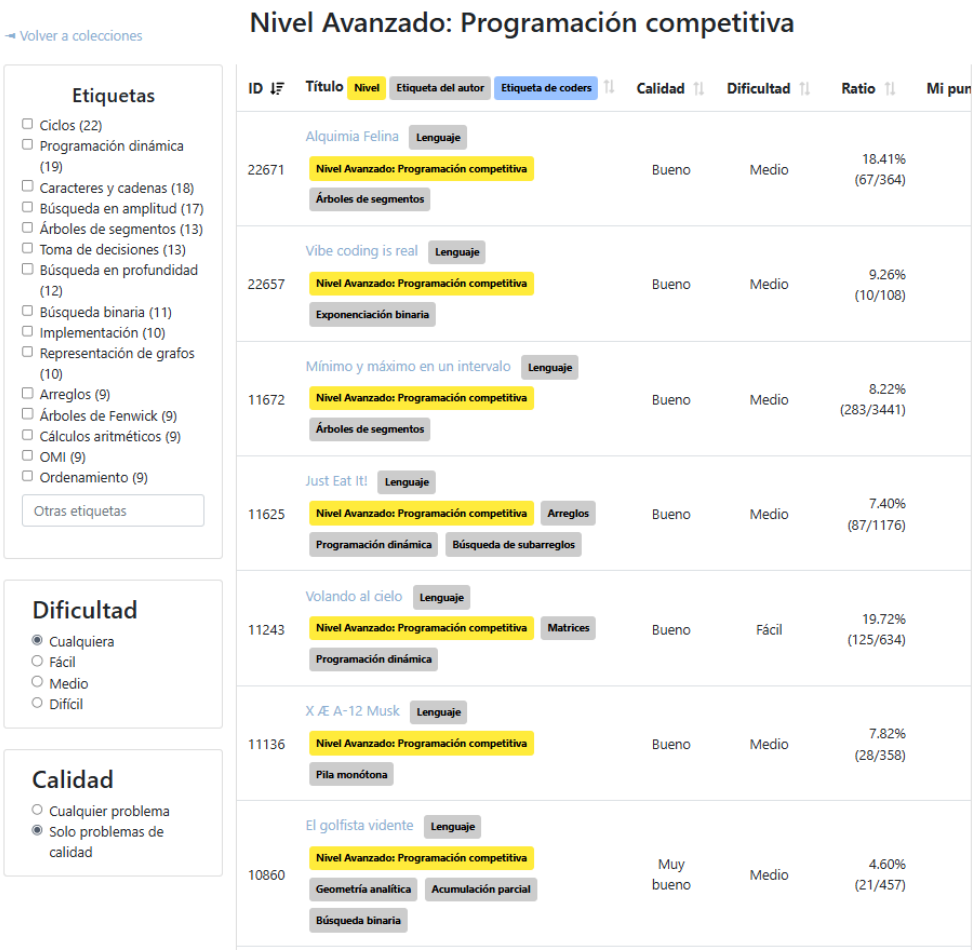


Figura 8: Repositorio de problemas avanzados para entrenamiento de élite.

3 ANÁLISIS DE LA INTERFAZ DEL PROBLEMA

La comprensión precisa de la interfaz de un problema es determinante para el éxito en la competición. Cada elemento visual proporciona información crítica sobre las restricciones y los requisitos de formato que el juez automático validará.

3.1 Anatomía Técnica del Reto

Al seleccionar un problema se despliega un entorno estructurado que guía al programador a través de las especificaciones del desafío. Los componentes principales son los siguientes:

- **Título y Autor:** Identifica el nombre del reto y el creador del mismo. Esto permite rastrear el origen del problema en competencias previas.
- **Límites de Ejecución:** Localizados usualmente en la parte superior derecha o bajo el título. Definen el **Tiempo Límite** en milisegundos y el **Límite de Memoria** en megabytes. Ignorar estos valores resulta en fallos de tipo TLE o MLE.
- **Descripción del Problema:** Texto narrativo que establece el contexto y la lógica del algoritmo que se debe implementar.

Figura 9: Listado de eventos y concursos disponibles.

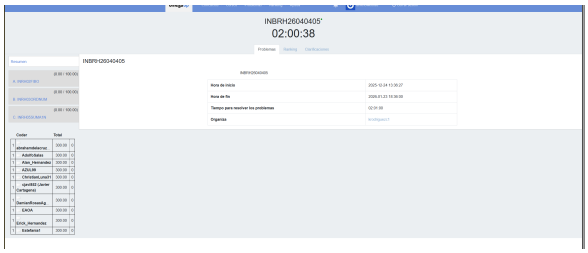


Figura 10: Panel de acceso y confirmación a la arena oficial.

- **Especificación de Entrada:** Detalla exactamente cómo recibirá los datos el programa. Indica el número de casos, los rangos de los valores y el orden de lectura.
- **Especificación de Salida:** Define el formato estricto de la respuesta. Cualquier carácter adicional o falta de espacios resultará en un veredicto de respuesta incorrecta.

3.2 Casos de Ejemplo y Validación Inicial

Al final de la descripción se presentan tablas con **Ejemplos de Entrada** y **Ejemplos de Salida**. Estos casos son fundamentales para validar la lógica del código antes de realizar un envío oficial al juez.

- **Validación de Formato:** Permite verificar si el programa lee los datos correctamente y si imprime la respuesta con los espacios y saltos de línea exigidos.
- **Prueba de Escritorio:** El estudiante debe ser capaz de replicar manualmente el resultado del ejemplo siguiendo la lógica de su algoritmo.

3.3 Interfaz de Competición en Tiempo Real

Cuando el acceso se realiza a través de un concurso oficial la interfaz se modifica para ofrecer herramientas de monitorización de la competencia. El entorno se vuelve más minimalista para priorizar la resolución rápida.

- **Cronómetro de Competencia:** Muestra el tiempo restante del concurso en formato de cuenta regresiva. Es vital para la gestión del tiempo entre problemas fáciles y difíciles.
- **Panel de Problemas del Evento:** Lista de forma compacta todos los retos asignados al concurso permitiendo la alternancia rápida entre ellos.
- **Puntuación Actual:** Indica el valor acumulado según la correctitud de los envíos realizados durante el tiempo activo.

3.4 Sistema de Etiquetas y Metadatos

En la parte lateral o inferior el sistema muestra etiquetas que categorizan el problema. Para un programador de nivel avanzado estas etiquetas son pistas sobre las estructuras de datos necesarias como **Punteros, Pilas, Colas o Grafos**.

4 IMPLEMENTACIÓN Y ENVÍO DE SOLUCIONES

El proceso de envío es el puente entre el desarrollo lógico y la validación automatizada. En **Programación para Competición Avanzado** cualquier error en el protocolo de envío puede resultar en una descalificación del problema incluso si la lógica es correcta.

4.1 Reglas de Codificación y Flujo de Datos

El juez automático de **omegaUp** interactúa exclusivamente con el flujo de datos estándar. Es imperativo seguir estas normas técnicas antes de realizar cualquier remisión de código:

- **Entrada Estándar - stdin:** El programa debe leer los datos tal como se presentan en el problema. No se deben incluir archivos externos ni rutas de lectura locales.
- **Salida Estándar - stdout:** La respuesta debe imprimirse directamente. Está estrictamente prohibido imprimir mensajes informativos como **Ingrese un número** o **El resultado es**. Cualquier carácter extra en la salida provocará un veredicto de respuesta incorrecta.
- **Librerías Permitidas:** Se recomienda utilizar librerías estándar del lenguaje elegido para evitar errores de vinculación en el servidor del juez.

4.2 Procedimiento de Envío en Modo Práctica

Cuando se trabaja en colecciones de problemas de entrenamiento el sistema despliega una ventana emergente de carga. Para completar el envío de forma exitosa debe realizar las siguientes acciones:

1. Localice el botón **Enviar solución** en la parte superior derecha de la interfaz del problema.
2. En el cuadro de diálogo resultante seleccione el lenguaje de programación. Es vital elegir la versión correcta del compilador como **C++20 u 11** para asegurar la compatibilidad de las funciones.
3. Pegue el código fuente en el área de texto o utilice la opción de cargar archivo para subir el documento desde su ordenador.

4.3 Gestión de Envíos en Entorno de Competición

Dentro de un concurso oficial la interfaz de envío se integra de forma directa en el panel de control para agilizar la respuesta del competidor. Este módulo es más compacto y requiere una atención especial a la selección del lenguaje.

- **Selector de Lenguaje:** Presenta un menú desplegable con todos los compiladores soportados por el evento. El estudiante debe verificar que el lenguaje seleccionado coincida con la sintaxis del código pegado.
- **Área de Código Integrada:** Permite realizar ajustes rápidos de último minuto antes de confirmar el envío oficial.
- **Confirmación de Envío:** Al presionar el botón de envío el código es encolado inmediatamente en los servidores de evaluación. No es posible cancelar un envío una vez que ha entrado en la fase de calificación.

4.4 Sincronización de Envíos Simultáneos

En problemas de alta complejidad es posible realizar múltiples envíos de un mismo reto para intentar corregir fallos. El sistema mantendrá un registro histórico de cada intento permitiendo al competidor visualizar qué versión del código obtuvo el mejor rendimiento en términos de tiempo y memoria.

5 EVALUACIÓN Y CALIFICACIÓN AUTOMATIZADA

Una vez que el código fuente es remitido a los servidores de **omegaUp** comienza la fase de evaluación. El sistema utiliza un motor de calificación que somete al programa a una batería de pruebas de estrés para validar su integridad y eficiencia bajo condiciones extremas.

5.1 Interpretación de la Tabla de Envíos

Tras procesar la solución el sistema devuelve una fila informativa en la sección de **Envíos**. Esta tabla es la principal fuente de retroalimentación para el competidor y se compone de los siguientes indicadores técnicos:

- **Porcentaje:** Indica el grado de acierto del algoritmo. Un valor de **100.00 %** en color verde significa que el código resolvió todos los casos de prueba satisfactoriamente.
- **Ejecución y Salida:** Muestra el estado del proceso. El estado **Terminada** con salida **Correcta** confirma que el programa finalizó sin errores de sistema y produjo la respuesta esperada.
- **Memoria y Tiempo:** Son métricas críticas de eficiencia. El tiempo se mide en segundos y la memoria en megabytes. Estos valores determinan la posición del competidor en el ranking de mejores envíos.

5.2 Glosario de Veredictos del Juez

Es fundamental que el estudiante de **Programación para Competición Avanzada** sea capaz de diagnosticar fallos a partir de las siglas de veredicto proporcionadas por el juez:

- **AC - Accepted:** Código correcto que cumple con todos los requisitos del problema.
- **WA - Wrong Answer:** El código es lógicamente erróneo para ciertos casos de prueba.
- **TLE - Time Limit Exceeded:** El algoritmo posee una complejidad temporal ineficiente para los límites establecidos.
- **MLE - Memory Limit Exceeded:** El programa utiliza estructuras de datos que desbordan la memoria asignada.
- **CE - Compilation Error:** El código fuente presenta errores sintácticos que impiden la generación del binario.

5.3 Análisis de Opiniones y Dificultad Percibida

La plataforma incluye un sistema de retroalimentación donde la comunidad de codificadores evalúa la **Calidad** y la **Dificultad** de cada reto. Este módulo proporciona una visión estadística sobre la complejidad real del problema comparada con la percepción general de los usuarios.

- **Métrica de Calidad:** Valora la claridad de la redacción y la consistencia de los casos de prueba.
- **Métrica de Dificultad:** Clasifica el reto desde muy fácil hasta muy difícil basándose en la experiencia de resolución de la comunidad.

5.4 Cuadro de Honor y Optimización Extrema

Bajo los resultados personales se despliega la tabla de **Mejores envíos aceptados**. Este panel es una herramienta de aprendizaje fundamental ya que permite comparar nuestro rendimiento contra los programadores más eficientes de la plataforma.

- **Optimización de Tiempo:** Permite observar la brecha de rendimiento entre diferentes lenguajes como C++ y Python.
- **Eficiencia de Memoria:** Muestra qué tan compacto puede llegar a ser un algoritmo mediante el uso de estructuras de datos personalizadas o optimizaciones a nivel de compilador.

6 CONCLUSIONES Y RECOMENDACIONES TÉCNICAS

El dominio de la plataforma **omegaUp** constituye una competencia fundamental para el éxito en la asignatura de **Programación para Competición Avanzada**. La capacidad de interactuar con un juez automático bajo condiciones de estrés temporal prepara al estudiante para desafíos de ingeniería de software en el mundo real.

6.1 Conclusiones sobre el Entorno de Competición

Tras el análisis detallado de la plataforma se extraen las siguientes conclusiones operativas:

- **Estandarización del Código:** La plataforma impone un rigor absoluto en la gestión de entradas y salidas. Esto obliga al programador a enfocarse en la pureza del algoritmo y la eficiencia del procesamiento de datos.
- **Gestión de Recursos:** El monitoreo constante de los límites de memoria y tiempo fomenta el diseño de soluciones óptimas en lugar de implementaciones de fuerza bruta.
- **Aprendizaje Comunitario:** El acceso a rankings de rendimiento y estadísticas de dificultad permite una autoevaluación constante frente a los estándares globales de programación competitiva.

6.2 Tips Avanzados para el Éxito en la Arena

Para maximizar el rendimiento y obtener veredictos de **Accepted** de forma consistente se sugieren las siguientes optimizaciones técnicas:

1. **Optimización de Entrada y Salida:** En lenguajes como C++ el uso de `std::cin` y `std::cout` puede ser lento para archivos de datos masivos. Se recomienda desactivar la sincronización con el siguiente comando:
   `ios_base::sync_with_stdio(0); cin.tie(0);`
2. **Análisis de Complejidad Previo:** Antes de escribir una sola línea de código el estudiante debe estimar la complejidad temporal del algoritmo en notación Big O basándose en los límites de tiempo del problema.
3. **Uso de Tipos de Datos Correctos:** Para evitar errores de desbordamiento en problemas matemáticos se debe priorizar el uso de **long long** sobre **int** cuando los rangos de salida superen los dos mil millones.
4. **Búsqueda de Puntajes Parciales:** En problemas de alta complejidad durante un concurso oficial es válido implementar soluciones sub-óptimas que garanticen una fracción del puntaje total en lugar de dejar el reto en cero puntos.

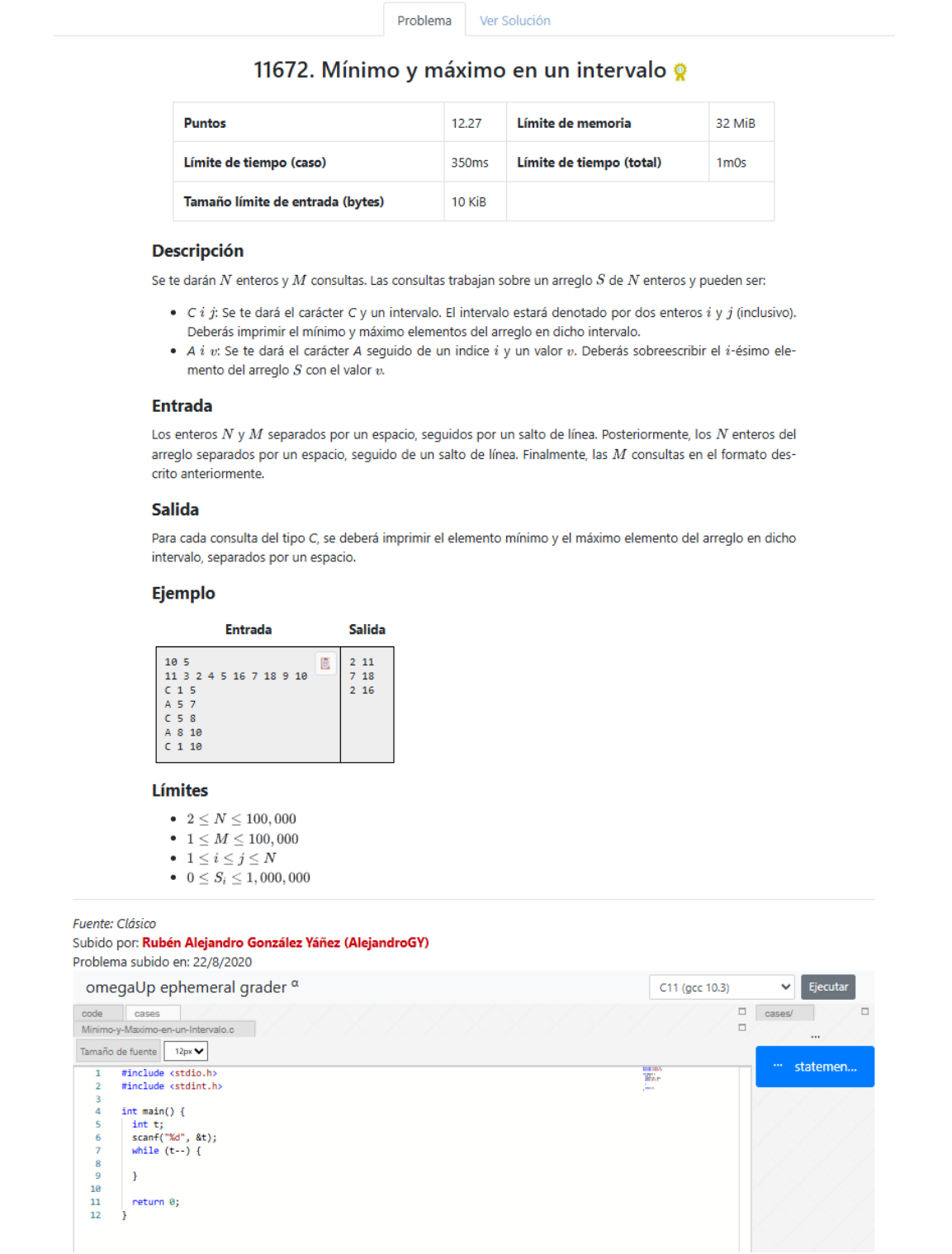
6.3 Material Bibliográfico de Referencia

Como complemento indispensable para el desarrollo de las competencias de este curso se debe consultar el manual de algoritmos y estructuras de datos disponible en el repositorio oficial de la asignatura.

LIBRO GUÍA DE PROGRAMACIÓN COMPETITIVA

https://drive.google.com/file/d/1PLOO3wLCnOVC_cODwiofahsRGeyoJeCU/view

El estudio de esta bibliografía en conjunto con la práctica sistemática en la plataforma asegurará el dominio de las técnicas avanzadas requeridas para la resolución de problemas de nivel internacional.



Problema Ver Solución

11672. Mínimo y máximo en un intervalo

Puntos	12.27	**Límite de memoria**	32 MiB
Límite de tiempo (caso)	350ms	**Límite de tiempo (total)**	1m0s
Tamaño límite de entrada (bytes)	10 KiB		

Descripción

Se te darán N enteros y M consultas. Las consultas trabajan sobre un arreglo S de N enteros y pueden ser:

- *C* i j: Se te dará el carácter *C* y un intervalo. El intervalo estará denotado por dos enteros i y j (inclusivo). Deberás imprimir el mínimo y máximo elementos del arreglo en dicho intervalo.
- *A* i v: Se te dará el carácter *A* seguido de un indice i y un valor v. Deberás sobreescribir el i-ésimo elemento del arreglo S con el valor v.

Entrada

Los enteros N y M separados por un espacio, seguidos por un salto de línea. Posteriormente, los N enteros del arreglo separados por un espacio, seguido de un salto de línea. Finalmente, las M consultas en el formato descrito anteriormente.

Salida

Para cada consulta del tipo *C*, se deberá imprimir el elemento mínimo y el máximo elemento del arreglo en dicho intervalo, separados por un espacio.

Ejemplo

Entrada	Salida
10 5 11 3 2 4 5 16 7 18 9 10 C 1 5 A 5 7 C 5 8 A 8 10 C 1 10	2 11 7 18 2 16

Límites

- $2 \leq N \leq 100,000$
- $1 \leq M \leq 100,000$
- $1 \leq i \leq j \leq N$
- $0 \leq S_i \leq 1,000,000$

Fuente: Clásico
Subido por: **Rubén Alejandro González Yáñez (AlejandroGY)**
Problema subido en: 22/8/2020

omegaUp ephemeral grader α
C11 (gcc 10.3) Ejecutar
code cases
Minimo-y-Maximo-en-un-Intervalo.c
Tamaño de fuente 12px
cases/
... statemen...

```c
#include <stdio.h>
#include <stdint.h>

int main() {
  int t;
  scanf("%d", &t);
  while (t--) {

  }

  return 0;
}
```

Figura 11: Componentes fundamentales de la descripción técnica de un problema.

Nota

Todas las entradas, deberán ser leídas por teclado, puedes ocupar la clase Scanner o BufferedReader que viene precargada por default.

NO es necesario textos adicionales como:

```
"Ingrese el nivel a calcular: X"
"Ingrese un valor: X"
"El resultado es:"
```

Únicamente será el valor y únicamente la sucesión separando cada valor por un espacio

Descripción

La sucesión de fibonacci consta de una serie de número naturales que se suman en pares a partir del 0 y 1, donde siempre se realiza la suma de los dos últimos números. 1,1,2,3,5,8,13,21,31...

Nivel Sucesión

1: 1

2: 0 + 1 = 1 | 1 1

3: 1 + 1 = 2 | 1 1 2

4: 2 + 1 = 3 | 1 1 2 3

5: 3 + 2 = 5 | 1 1 2 3 5

6: 5 + 3 = 8 | 1 1 2 3 5 8

Entrada

N Deberás ingresar el número de la posición que ocupa en la sucesión de fibonacci

Salida

Mostrar la sucesión hasta el número ingresado por pantalla, los número deben estar separados por un espacio en blanco

Ejemplo

Entrada	Salida	Descripción
1	1	1
2	1 1	0 + 1 = 1
5	1 1 2 3 5	3 + 2 = 5

Límites

$0 < N < 20$

Figura 12: Visualización del entorno de trabajo durante un evento de competición oficial.

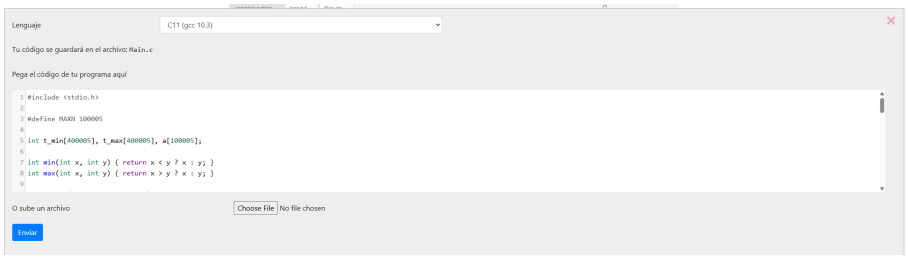


Figura 13: Módulo de carga de código para retos en modo de práctica libre.

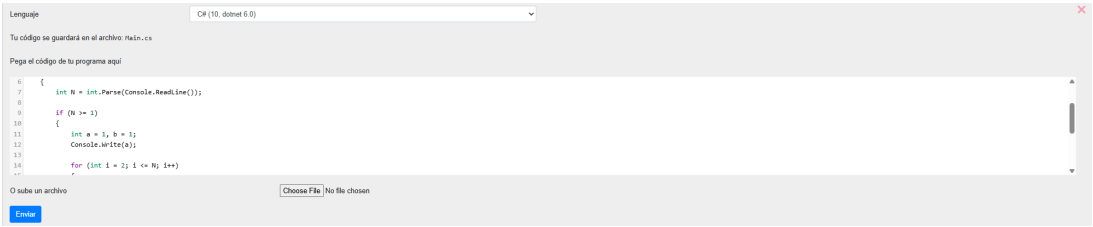


Figura 14: Interfaz especializada de remisión de código fuente durante eventos oficiales.

Envíos

Fecha y hora	Lenguaje	Porcentaje	Ejecución	Salida	Memoria	Tiempo	Acciones
2026-01-08 18:31	c11-gcc	100.00%	Terminada	Correcta	4.74 MB	0.57 s	

Tienes que esperar 50 segundos antes de que puedas hacer otro envío a este problema

« ‹ 1 › »

Opiniones de coders

Calidad

3.2

54 votos en total

Muy bueno	25
Bueno	18
Regular	8
Malo	1
Muy malo	2

Dificultad

1.8

55 votos en total

Muy fácil	10
Fácil	9
Medio	19
Difícil	12
Muy difícil	5

Mejores envíos aceptados

Coder	Lenguaje	Memoria	Tiempo	Fecha y hora
SoyJUAAN	cpp20-gcc	7.36	0.19	2025-08-29 23:35:04
JuanLANDR	cpp20-gcc	7.33	0.20	2025-08-29 23:17:08
carloslagoscortes	cpp20-gcc	4.76	0.37	2023-12-11 14:21:32
Apocryphon	cpp20-gcc	5.00	0.39	2022-09-06 22:04:30
PedroAyon_TAM	cpp17-clang	4.49	0.43	2024-10-27 18:52:37
Yahve666	cpp20-gcc	4.36	0.46	2023-02-23 21:27:19
BYTE-F0RCE-	cpp20-clang	4.48	0.46	2025-02-25 19:41:58
dumac	cpp20-gcc	4.92	0.47	2022-11-14 22:06:33
JesusRuizUwU	cpp20-gcc	4.24	0.48	2025-10-08 00:53:52
s3b4s7i4n	cpp17-gcc	4.83	0.49	2024-09-23 21:23:01

Figura 15: Registro de calificación detallada con métricas de consumo de recursos.

Codigo

```
#include <stdio.h>

#define MAXN 100005

int t_min[400005], t_max[400005], a[100005];

int min(int x, int y) { return x < y ? x : y; }
int max(int x, int y) { return x > y ? x : y; }

void build(int n, int s, int e) {
    if (s == e) { t_min[n] = t_max[n] = a[s]; return; }
```

Casos

Grupo	Caso	Veredicto	Score
caso0			0.0908203125 / 0.0908203125
caso1			0.0908203125 / 0.0908203125
caso2			0.0908203125 / 0.0908203125
caso3			0.0908203125 / 0.0908203125
caso4			0.0908203125 / 0.0908203125
caso5			0.0908203125 / 0.0908203125
caso6			0.0908203125 / 0.0908203125
caso7			0.0908203125 / 0.0908203125
caso8			0.0908203125 / 0.0908203125
caso9			0.0908203125 / 0.0908203125
caso10			0.0908203125 / 0.0908203125

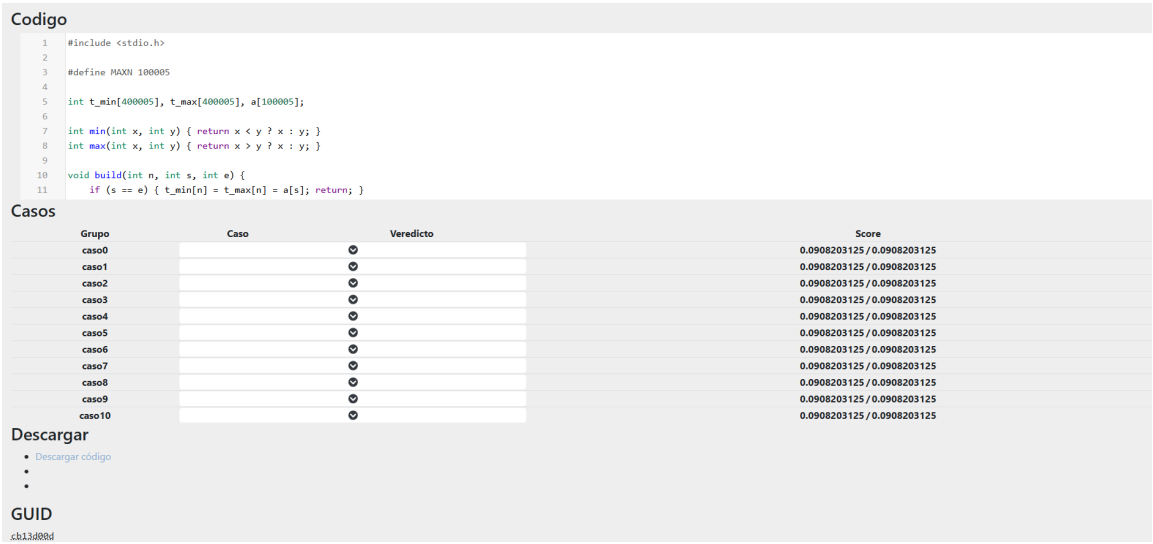
Descargar

- Descargar código
-
-

GUID

cb13d00d

Figura 16: Distribución estadística de opiniones sobre la calidad y nivel de dificultad del reto.



Codigo

```
#include <stdio.h>

#define MAXN 100005

int t_min[400005], t_max[400005], a[100005];

int min(int x, int y) { return x < y ? x : y; }
int max(int x, int y) { return x > y ? x : y; }

void build(int n, int s, int e) {
    if (s == e) { t_min[n] = t_max[n] = a[s]; return; }
```

Casos

Grupo	Caso	Veredicto	Score
caso0			0.0908203125 / 0.0908203125
caso1			0.0908203125 / 0.0908203125
caso2			0.0908203125 / 0.0908203125
caso3			0.0908203125 / 0.0908203125
caso4			0.0908203125 / 0.0908203125
caso5			0.0908203125 / 0.0908203125
caso6			0.0908203125 / 0.0908203125
caso7			0.0908203125 / 0.0908203125
caso8			0.0908203125 / 0.0908203125
caso9			0.0908203125 / 0.0908203125
caso10			0.0908203125 / 0.0908203125

Descargar

- Descargar código
-
-

GUID

cb13d00d

Figura 17: Ranking histórico de las soluciones más eficientes registradas en el sistema.