Introducción a la programación competitiva

Guillermo Hernández

2024-03-26

Versión en línea: https://www.dih5.es/intro-cp.html.

1 Presentación

La programación competitiva (CP, *competitive programming*) es un deporte mental cuyo objetivo es escribir código que resuelva problemas concretos de diversa dificultad. Aunque principalmente se trabaja la algoritmia, también tienen cabida problemas que requieran conocimientos matemáticos (combinatoria, teoría de números, teoría de grafos, geometría…), dependiendo de la competición. Habitualmente se evalúan las soluciones mediante un *juez en línea*, un programa que ejecuta el código frente a pares de entrada y salida, verificando el funcionamiento correcto.

A nivel **preuniversitario**, el evento nacional más importante es la Olimpiada Informática Española. A nivel **universitario**, en la Universidad de Salamanca por el momento participamos en el concurso AdaByron a través de la modalidad de Regional Multisede, que sirve de clasificatorio para la fase nacional.

2 Recursos

Si quieres introducirte en el mundo de la Programación Competitiva, lo más importante es practicar en alguna plataforma. Algunas opciones son:

- Kattis Repositorio y juez en línea con una gran cantidad de problemas (en inglés) ordenados por dificultad y con una interfaz moderna. Existe una clasificación de problemas mantenida por Steven Halim que puede ayudar a seleccionarlos para aprender para practicar temas concretos u obtener pistas de resolución.
- ¡Acepta el reto!, repositorio de problemas **en español** y juez en línea mantenido por un grupo de profesores de la UCM.
- Online Judge, uno de los jueces más extendidos, con una gran colección de problemas.

- JU:EZ es un juez que pusimos en la USAL para la realización de ejercicios de la asignatura Programación I. Aunque el formato es similar al de la programación competitiva, el contenido está realmente orientado, en general, a dicha asignatura. Además, las cuentas se crean solamente para alumnos de la asignatura.

Existe también algún libro dedicado a la programación competitiva. El mejor que conozco es el manual de Steven Halim, Felix Halim y Suhendry Effendy. Existe una traducción al español realizada por Miguel Revilla Rodríguez en dos volúmenes. El primero es suficiente para un primer acercamiento, mientras que el segundo incluye material más especializado para concursantes de nivel avanzado.

3 Guía básica

En este apartado propondremos un plan de trabajo para estudiantes que quieran participar en sus primeros concursos de programación competitiva, quizá para descubrir si les interesa este dominio. Es una guía corta, pensada para coger un poco de panorámica del campo. Los participantes convencidos que quieran prepararse más seriamente pueden acudir al manual citado arriba o alguna de las fuentes que se desarrollan al final de este artículo para continuar su aprendizaje.

3.1 Escoger lenguaje(s) de programación

Los lenguajes disponibles dependen de los concursos o de los portales con problemas, pero los **más habituales** son **C++, Java y Python**. En algunos concursos, como la Olimpiada Informática, es posible que sea necesario utilizar C++ para completar todos los problemas. Esto se debe a que se suelen imponer restricciones de tiempo de ejecución en la evaluación de los problemas para verificar que el algoritmo es computacionalmente aceptable. Dependiendo de las circunstancias, es posible que se imponga un mismo límite de tiempo en todos los lenguajes y que por ello C++ se vuelva una necesidad.

A una persona convencida ya en formarse en programación competitiva le recomendaría aprender Python (muchos de los problemas son más rápidos de implementar y los errores de implementación suelen ser más fáciles de detectar y corregir) y C++ (para los casos más complejos o para aprovechar la completa biblioteca estándar de la que dispone).

A una persona interesada simplemente en conocer la programación competitiva le recomendaría simplemente participar con el lenguaje que le sea más familiar. En el caso de los programadores de lenguaje C, aunque este lenguaje no esté directamente soportado (depende de la plataforma), gran cantidad del código C será directamente compatible con C++, así que pueden usar también dicho

lenguaje. La principal carencia es que la biblioteca estándar es mucho más básica, con lo que es difícil ser competitivo.

Otra opción es formarse a nivel básico en Python (y quizá aprovechar la oportunidad para aprender el lenguaje a través de la programación competitiva). Un recurso interesante es el libro Automate the Boring Stuff with Python, accesible de forma gratuita. Los primeros 7 capítulos pueden ser los más interesantes para la programación competitiva. El resto de capítulos se empiezan a organizar más en torno a proyectos que, aunque son interesantes también, siguen ya más por otra línea.

3.2 Escoger plataformas para practicar

Mi plataforma favorita es Kattis, por la clasificación que ofrece directamente por dificultad y por las temáticas en que lo clasifica Steven Halim en su web. En lo que sigue en la guía se enlazarán algunos problemas de Kattis, así que es una buena opción para empezar.

Si el inglés es un problema (¡que solo puede ser temporal!), otra buena opción es ¡Acepta el reto!, con problemas en español. Si estás interesado en preparar el concurso AdaByron, es también muy buena opción, porque es el mismo que se usará allí y dispone de colecciones de problemas anteriores.

Si eres o has sido alumno de Programación I en el Grado en Ingeniería Informática en la USAL, tendrás también cuenta en la plataforma JU:EZ. Aunque los ejercicios que hay están orientados a la asignatura (y no a la programación competitiva), el formato es similar, y puedes también intentar resolverlos en varios lenguajes de programación.

3.3 Empezando a procesar entradas y salidas

El primer reto que normalmente ofrece la programación competitiva es saber procesar las entradas y salidas. Aunque esto simplemente requiere usar las funciones de entrada y salida, el ser estricto con no escribir nada de más puede resultar chocante a quien no haya utilizado antes un juez en línea.

Algunos problemas elementales para practicar:

- Stuck In A Time Loop.
- R2.
- Digit Swap.

3.4 Esquemas típicos de lectura

Un patrón muy típico es aquel en el que hay que resolver un mismo problema para varios casos de entrada. La casuística está muy bien explicada en la guía de ¡Acepta el reto!, a la que remito al lector para continuar su aprendizaje.

A continuación se recoge una versión en Python de los tres esquemas

```
# Plantilla para el esquema 1: número de casos al principio
def caso():
    # TODO: programar la lectura del caso y la solución del mismo
    pass

def main():
    for _ in range(int(input())):
        caso()

main()
```

```
# Plantilla para el esquema 2: un caso especial marca el final
def caso():
    # TODO: hacer aquí la lectura de datos, guardando en condicion_fin si el caso que marca
    condicion_fin = False

    if condicion_fin:
        return False
    else:
        # TODO: programar aquí la solución del caso
        return True

def main():
    while caso():
        pass

main()
```

```
# Plantilla para el esquema 3: lectura hasta final de entrada (poco usual)
def caso():
    try:
        # TODO: Leer aquí con input el caso completo
        xxx = input()

        # TODO: programar aquí la solución del caso

        return True
    except EOFError:
        return False

def main():
    while caso():
        pass

main()
```

Alternativamente, se puede usar `sys.stdin.readlines()` para leer todas las líneas de golpe como un array de cadenas. En los casos en que la entrada sea especialmente extensa puede ser fundamental para construir una solución suficientemente rápida. Esta situación se suele advertir en los enunciados.

Algunos problemas para practicar el esquema 1:

- Oddities
- Last Factorial Digit
- Turn It Up! (no son exactamente casos independientes, pero las lecturas siguen un patrón similar)

Algunos problemas para practicar el esquema 2:

- Speed Limit
- Mixed Fractions

3.5 Arrays y ordenación

Muchos de los problemas requieren almacenar una secuencia de datos homogéneos, a lo que responde de forma natural el concepto de array unidimensional, y ordenarlos con algún criterio. En Python la herramienta fundamental para ello son las "funciones clave" (*key functions*), que permiten variar los criterios de ordenación de forma sencilla. Se puede consultar más en este enlace.

En C++ existe una solución similar, a través de la función std::sort, aunque en este caso lo que se debe suministrar es un **comparador**, esto es, una función que reciba dos elementos y que devuelva verdadero si el primero es menor que el segundo.

Alternativamente, los programadores de C tienen una herramienta parecida, que es la función `qsort`. Requiere usar punteros a funciones, así que será más complicada de usar, pero si solo llevas ese lenguaje merece la pena conocerla. Se puede consultar la explicación y un ejemplo aquí.

Algunos problemas para practicar:

- Jolly Jumpers
- Closing the Loop (orientación: construye dos arrays y ordénalos)
- Stacking Cups (ejemplo sencillo con funciones clave)
- A Classy Problem (ejemplo más complicado con funciones clave)

3.6 Grafos

Puede ser conveniente familiarizarse con la noción matemática de grafo, que puede darse con frecuencia en CP. Un curso muy completo, que puedes leer hasta donde tu paciencia y tiempo permitan, es del GeeksForGeeks.

Algunos ejercicios relativamente sencillos para empezar son los siguientes:

- Weak Vertices (orientación: directamente a partir de la matriz de adyacencia se pueden analizar todas las combinaciones de tres vértices e ir almacenando en un `set` los que están en un triángulo).
- Where's My Internet?? (orientación: haz algún recorrido por el grafo para encontrar los vértices conectados al primero).
- Sheba's Amoebas (orientación: es un algoritmo de relleno (*floodfill*)).

4 Sugerencias para continuar

Como se decía antes, esto solo es una guía para facilitar una primera experiencia con la programación competitiva. Entre las principales omisiones se tienen que destacar los paradigmas de resolución de problemas propios de la algoritmia (en la USAL se estudian en segundo curso del GII).

Para aquellos interesados en continuar el aprendizaje, lo mejor en mi opinión es intentar hacer un acercamiento por temas y, sobre todo, practicar mucho. El libro de Halim *et al.* puede ser una buena guía para ello.

Otro recurso para practicar pueden ser los entrenamientos oficiales de los concursos. AdaByron tiene en marcha una edición de estos en el momento de redacción de esta entrada. Una guía para detectar los más fáciles es mirar la cuenta de envíos aceptados (AC) y empezar por aquellos que tengan más.