‘Programación competitiva'

¡Resuelve los problemas lo más rápido posible!.

Ejemplo del Libro de Programación Competitiva

An illustration: UVa Online Judge [44] Problem Number 10911 (Forming Quiz Teams).
Abridged Problem Description:

Let (x, y) be the integer coordinates of a student's house on a 2D plane. There are $2N$ students and we want to **pair** them into N groups. Let d_i be the distance between the houses of 2 students in group i. Form N groups such that $cost = \sum_{i=1}^{N} d_i$ is **minimized**. Output the minimum $cost$ as a floating point number with 2 digits precision in one line. Constraints: $1 \leq N \leq 8$ and $0 \leq x, y \leq 1000$.

Sample input (with explanation):
$N = 2$; Coordinates of the $2N = 4$ houses are $\{1, 1\}$, $\{8, 6\}$, $\{6, 8\}$, and $\{1, 3\}$.

Sample output (with explanation):
$cost = 4.83$.

Can you solve this problem?
If so, how many minutes would you likely require to complete the working code?
Think and try not to flip this page immediately!

Ejemplo del Libro de Programación Competitiva

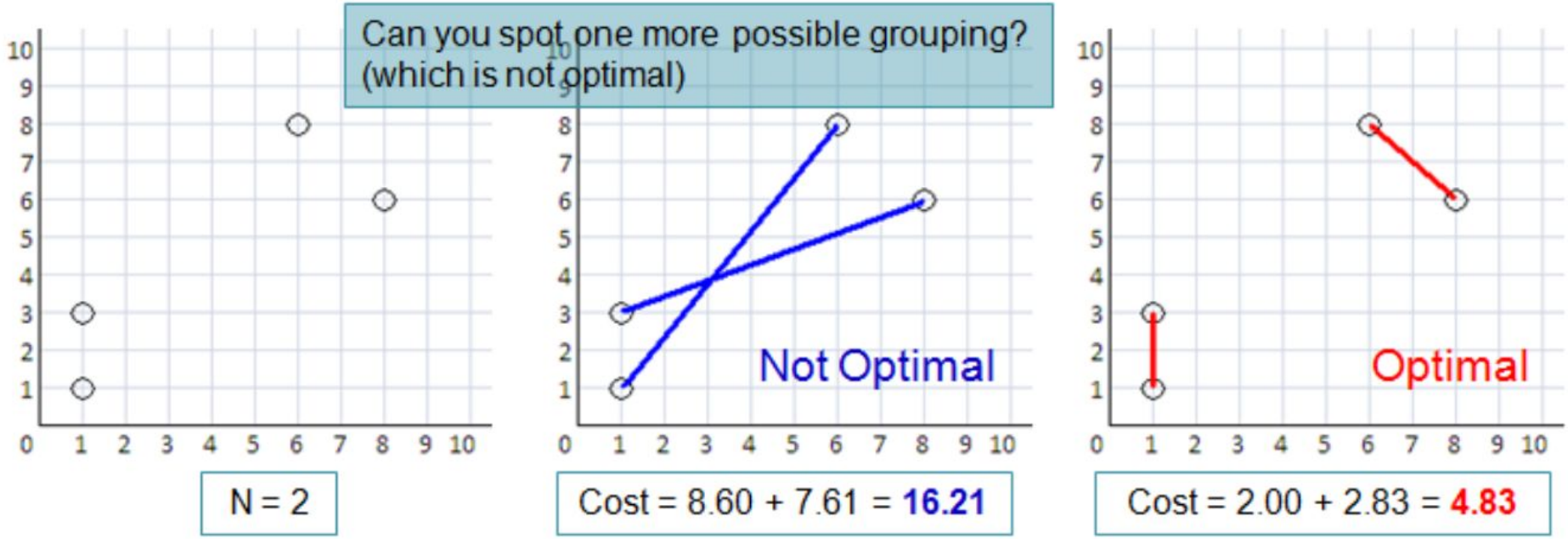

Figure 1.1: Illustration of UVa 10911 - Forming Quiz Teams

¿Qué tipo de programador eres?

Programador no competitivo A (también conocido como **the blurry one**)

Paso 1: Lee el problema y se confunde. (Este problema es nuevo para A).

Paso 2: Intenta codificar algo: Leer la entrada y salida no trivial.

Paso 3: A se da cuenta de que estas dos ideas a continuación no son aceptadas (AC):

Greedy: Emparejar repetidamente a los dos estudiantes restantes con las distancias de separación más cortas da la respuesta incorrecta (WA).

Complete Search: Usando retroceso recursivo (Sección 3.2) y probando todos los emparejamientos posibles se obtiene Time Limit Exceeded (TLE).

¿Qué tipo de programador eres?

Programador no competitivo B (Se rinde):

Paso 1: Lee el problema y se da cuenta de que es un problema de grafos. Pero B no ha aprendido a resolver este tipo de problema ...

B no conoce la solución de programación dinámica (DP) (Sección 3.5)...

Paso 2: omite el problema y lee otro problema en el conjunto de problemas.

¿Qué tipo de programador eres?

(Aún) **Programador no competitivo C** (lento):

Paso 1: Lee el problema y se da cuenta de que es un problema difícil: **'minimum weight perfect matching on a weighted complete graph'**. Sin embargo, dado que el tamaño de entrada es pequeño, este problema se puede solucionar utilizando DP. El estado para DP es una máscara de bits que describe un estado coincidente, y los alumnos no coincidentes i y j activarán dos bits i y j en la máscara de bits (consulte el libro 2).

Paso 2: Codifica la rutina de E/S, escribe DP recursivo de arriba hacia abajo, prueba, depura, ...

Paso 3: Después de 3 horas, la solución de C es AC (pasó todos los datos de prueba).

¿Qué tipo de programador eres?

Programador competitivo D:

Completa todos los pasos realizados por el programador no competitivo C en 30 minutos.

Programador muy competitivo E:

Un programador muy competitivo (por ejemplo, los codificadores rojos en Codeforces) resolvería este problema "bien conocido" en 10 minutos y posiblemente también sería consciente de las otras soluciones posibles para la(s) variante(s) más difícil(es) de este problema...

Consejos para ser competitivo

1. ¡Escriba el código más rápido!

Prueba de mecanografía en https://www.typingtest.com y siga las instrucciones sobre cómo mejorar su habilidad de mecanografía.

El de Steven es ⇠85-95 wpm,

el de Felix es ⇠55-65 wpm,

y el de Suhendry es ⇠70-80 wpm.

Si su velocidad de escritura es mucho menor que estos números, ¡tome este consejo en serio!

Sobretodo los símbolos más usados en programación como

[ ] { } < > | ~ & ; “ ‘ ! entre otros

2. Identifique rápidamente el tipo de problema

No	Category	In This Book	Frequency
1.	Ad Hoc	Section 1.4-1.6	1-2
2.	(Heavy) Data Structure	Chapter 2	0-1
3.	Complete Search (Iterative/Recursive)	Section 3.2++	1-2
4.	Divide and Conquer	Section 3.3	0-1
5.	Greedy (the non-classic ones)	Section 3.4	1
6.	Dynamic Programming (the non-classic ones)	Section 3.5++	1-2
7.	Graph (except Network Flow/Graph Matching)	Chapter 4	1
8.	Mathematics	Chapter 5	1-2
9.	String Processing	Chapter 6	1
10.	Computational Geometry	Chapter 7	1
11.	Some Harder/Rare/Emerging Trend Problems	Chapter 8-9	2-3
	Total in Set is usually ≤ 14		10-17

Clasificación de problemas según su perspectiva

No	Category	Confidence and Expected Solving Speed
A1.	I have solved this type before	I am sure that I can re-solve it again (and fast)
A2.	I have solved this type before	I am sure that I can re-solve it again (but slow)
B.	I have seen this type before	But that time I know that I cannot solve it yet
C.	I have not seen this type before	See the discussion below

Reducir el problema dado a otro problema (más fácil),
Reducir a un problema difícil conocido (NP-)
Identificar sugerencias sutiles o propiedades especiales en el problema,
Atacar el problema desde un ángulo no obvio / hacer una pregunta diferente,
Comprimir los datos de entrada,
Reelaboración de fórmulas matemáticas,
Enumerar observaciones/patrones,
Realización de análisis de casos de posibles subcasos del problema, etc.

3. Hacer análisis de algoritmos

Una vez que haya diseñado un algoritmo para resolver un problema particular en un concurso de programación, se debe hacer esta pregunta:

Dado el límite máximo de entrada (generalmente explicado en la descripción del problema), ¿puede el algoritmo desarrollado actualmente, con su complejidad de tiempo / espacio, pasar el límite de tiempo / memoria dado para ese problema en particular?

A veces, hay más de una forma de atacar un problema. Algunos enfoques pueden ser incorrectos, otros no lo suficientemente rápidos, y otros "exagerar".

Una buena estrategia es hacer una lluvia de ideas para los algoritmos posibles y luego elegir la solución más simple que funcione (es decir, que sea lo suficientemente rápida como para pasar el límite de tiempo y memoria y producir la respuesta correcta)

4. Dominar los lenguajes de programación

Hay varios lenguajes de programación para programación competitiva, incluyendo C/C++, Java y Python. ¿Qué lenguajes de programación se debe dominar?

Por experiencia preferimos C ++ (std = gnu ++ 17) con su biblioteca de plantillas estándar (STL) incorporada, pero aún necesitamos dominar Java y algunos conocimientos de Python.

Aunque es más lento, Java tiene potentes bibliotecas integradas y APIs como BigInteger / BigDecimal, GregorianCalendar, Regex, etc. Los programas Java son más fáciles de depurar con la capacidad de la máquina virtual para proporcionar un seguimiento de pila cuando se bloquea (a diferencia de los volcados de núcleo o los errores de segmentación en C / C ++).

Del mismo modo, el código Python puede ser sorprendentemente muy corto para algunas tareas adecuadas.

Por otro lado, C / C++ también tiene sus propios méritos. Dependiendo del problema en cuestión, cualquiera de los dos idiomas puede ser la mejor opción para implementar una solución en el menor tiempo posible.

Suppose that a problem requires you to compute 40! (the factorial of 40). The answer is very large: 815 915 283 247 897 734 345 611 269 596 115 894 272 000 000 000. This far exceeds the largest built-in primitive integer data type (`unsigned long long`: $2^{64} - 1$). As there is no built-in arbitrary-precision arithmetic library in C/C++ as of yet, we would have needed to implement one from scratch. The Python code, however, is trivially short:

```
import math
print(math.factorial(40))
```

The Java code for this task is also simple (more details in Section 2.2.4):

```
import java.util.Scanner;
import java.math.BigInteger;
class Main {                                          // default class name
  public static void main(String[] args) {
    BigInteger fac = BigInteger.ONE;
    for (int i = 2; i <= 40; ++i)
      fac = fac.multiply(BigInteger.valueOf(i)); // it is in the library!
    System.out.println(fac);
  }
}
```

5. Domina el arte de probar el código

Pensaste que habías resuelto un problema en particular. Identificaste el tipo de problema, diseñaste el algoritmo, verificaste que el algoritmo (con las estructuras de datos que utiliza) se ejecutaría en el tiempo (y dentro de los límites de memoria) considerando la complejidad del tiempo (y el espacio) e implementaste el algoritmo, pero su solución aún no es aceptada (AC).

Dependiendo del concurso de programación, puede o no obtener crédito por resolver el problema parcialmente.

En ICPC, solo obtendrá puntos por un problema en particular si el código de su equipo resuelve todos los casos de prueba secretos para ese problema. Otros veredictos como Error de presentación (PE), Respuesta incorrecta (WA), Límite de tiempo excedido (TLE), Límite de memoria excedido (MLE), Error de tiempo de ejecución (RTE), etc., no aumentan los puntos de su equipo.

En el IOI actual (2010-2019), se utiliza el sistema de puntuación de subtareas. Los casos de prueba se agrupan en subtareas que son las variantes más simples de la tarea original con límites de entrada más pequeños.

6. Práctica y más práctica

Los programadores competitivos, como los atletas deportivos, deben entrenar regularmente y mantenerse "en forma para programar".

El éxito viene como resultado de un esfuerzo continuo para mejorarse a sí mismo.

https://onlinejudge.org

https://uhunt.onlinejudge.org

https://www.udebug.com

https://open.kattis.com

Trabajo en equipo

Practique la codificación (o escribir pseudocódigo) en un papel en blanco. Esto es útil cuando tu compañero de equipo está usando la computadora. Cuando sea su turno de usar la computadora, puede escribir el código lo más rápido posible.

La estrategia de "enviar e imprimir": si su código obtiene un veredicto de AC, ignore la impresión. Si todavía no es AC, depure su código usando esa copia impresa (y deje que su compañero de equipo use la computadora para otro problema). Tenga cuidado: depurar sin la computadora no es una habilidad fácil de dominar.

Si su compañero de equipo está codificando actualmente (y no tiene idea de otros problemas), prepare los datos de prueba de casos extremos (y con suerte el código de su compañero de equipo pasa todos esos). Con dos miembros del equipo de acuerdo en la corrección de un código, la probabilidad de tener una penalización menor aumenta.

Si sabes que tu compañero de equipo es (significativamente) más fuerte en un determinado tipo de problema que tú y actualmente estás leyendo un problema con dicho tipo (especialmente en la etapa inicial del concurso), considera pasar el problema a tu compañero de equipo en lugar de insistir en resolverlo tú mismo.

Practique la codificación de un algoritmo bastante largo / complicado juntos como un par o incluso como un triple (con una presión de límite de tiempo de codificación) para el final en la que el equipo apunta a obtener un AC más en los últimos minutos.

El factor X: hazte amigo de tus compañeros de equipo fuera de las sesiones de entrenamiento y concursos.

Mensajes Comunes en los Jurados

WA (Wrong Answer - Respuesta Incorrecta): Esto ocurre cuando el programa produce una salida incorrecta para al menos uno de los casos de prueba. Significa que la solución no es correcta o no cumple con los requisitos del problema. Puede haber un error en la lógica del programa, en la implementación o en la interpretación del problema. Es importante revisar cuidadosamente el código y los algoritmos utilizados para encontrar y corregir el error.

TLE (Time Limit Exceeded - Tiempo de ejecución excedido): Esto sucede cuando el programa tarda demasiado tiempo en ejecutarse y excede el límite de tiempo establecido para el problema. En la programación competitiva, los problemas tienen restricciones de tiempo muy estrictas. Si tu solución es lenta y no cumple con los límites de tiempo establecidos, se considera inaceptable. En este caso, debes optimizar tu código y utilizar algoritmos más eficientes para resolver el problema.

Mensajes Comunes en los Jurados

RE (Runtime Error - Error de tiempo de ejecución): Esto indica que el programa encontró un error durante la ejecución. Puede deberse a diversos motivos, como una división por cero, un desbordamiento de memoria, una referencia a un puntero nulo u otros errores similares. Los errores de tiempo de ejecución deben ser identificados y corregidos para que el programa funcione correctamente.

AC (Accepted - Aceptado): Significa que el programa ha pasado todos los casos de prueba y ha producido la salida correcta para cada uno de ellos. Es el resultado deseado y significa que tu solución es correcta y cumple con todos los requisitos del problema. Obtener el resultado "AC" es el objetivo principal en la programación competitiva.

Mensajes Comunes en los Jurados

MLE (Memory Limit Exceeded - Límite de memoria excedido): Esto ocurre cuando el programa utiliza más memoria de la permitida. Al igual que el límite de tiempo, los problemas también tienen restricciones en la cantidad máxima de memoria que se puede utilizar. Si tu solución consume más memoria de la permitida, obtendrás un resultado "MLE". En este caso, es necesario optimizar la utilización de memoria, como reducir el tamaño de las estructuras de datos o utilizar técnicas de programación más eficientes.

CE (Compilation Error - Error de compilación): Esto indica que hubo un error durante la compilación del programa. Puede haber un error sintáctico o semántico en el código fuente, lo que impide que se genere el ejecutable correctamente. Antes de enviar el código para su evaluación, es importante asegurarse de que no haya errores de compilación.