Competencias de programación

Víctor Racsó Galván Oyola

10 de mayo de 2021

Índice

Estrategias

Estrategia cíclica

Esta es una de las estrategias más usadas en competencias individuales donde el tiempo es un factor clave cuando hay empate de puntos.

- El proceso consiste en leer un problema, resolverlo y pasar al siguiente hasta que ya no hayan problemas restantes o se acabe el tiempo de la competencia.
- Ya que se resuelven los problemas apenas se dé con la solución, no hay retraso, así que no se obtiene mucha penalidad de tiempo si se obtiene Accepted a la primera.

Estrategia cíclica

Esta es una de las estrategias más usadas en competencias individuales donde el tiempo es un factor clave cuando hay empate de puntos.

- El proceso consiste en leer un problema, resolverlo y pasar al siguiente hasta que ya no hayan problemas restantes o se acabe el tiempo de la competencia.
- Ya que se resuelven los problemas apenas se dé con la solución, no hay retraso, así que no se obtiene mucha penalidad de tiempo si se obtiene Accepted a la primera.

Ventajas

Si hay algún error, se corrige de inmediato y en promedio la penalidad generada es menor a comparación de otros métodos.

Estrategia cíclica

Esta es una de las estrategias más usadas en competencias individuales donde el tiempo es un factor clave cuando hay empate de puntos.

- El proceso consiste en leer un problema, resolverlo y pasar al siguiente hasta que ya no hayan problemas restantes o se acabe el tiempo de la competencia.
- Ya que se resuelven los problemas apenas se dé con la solución, no hay retraso, así que no se obtiene mucha penalidad de tiempo si se obtiene Accepted a la primera.

Ventajas

Si hay algún error, se corrige de inmediato y en promedio la penalidad generada es menor a comparación de otros métodos.

Desventajas

Si uno se estanca en un problema que nunca le llega a salir, puede perder la oportunidad de haber resuelto otro.

Estrategia de tempos

Esta es una estrategia recomendada por Gennady Korotkevich (tourist) en uno de sus blogs.

- Se leen todos los problemas inicialmente.
- Analizar los problemas y pensar en las soluciones de todos los que se pueda.
- Una vez que se llegue a un problema que uno ya no puede resolver en concepto, comenzar a implementar los problemas resueltos pendientes (se tiene la idea pero no el código).
- Enviar las soluciones cuando uno crea que es conveniente, no apenas se tengan.

Estrategia de tempos

Esta es una estrategia recomendada por Gennady Korotkevich (tourist) en uno de sus blogs.

- Se leen todos los problemas inicialmente.
- Analizar los problemas y pensar en las soluciones de todos los que se pueda.
- Una vez que se llegue a un problema que uno ya no puede resolver en concepto, comenzar a implementar los problemas resueltos pendientes (se tiene la idea pero no el código).
- Enviar las soluciones cuando uno crea que es conveniente, no apenas se tengan.

Ventajas

Se obtiene la sensación de control en todo momento durante la competencia y no se da información a los contrincantes sobre qué problema podría salir fácilmente o pronto.

Estrategia de tempos

Esta es una estrategia recomendada por Gennady Korotkevich (tourist) en uno de sus blogs.

- Se leen todos los problemas inicialmente.
- Analizar los problemas y pensar en las soluciones de todos los que se pueda.
- Una vez que se llegue a un problema que uno ya no puede resolver en concepto, comenzar a implementar los problemas resueltos pendientes (se tiene la idea pero no el código).
- Enviar las soluciones cuando uno crea que es conveniente, no apenas se tengan.

Ventajas

Se obtiene la sensación de control en todo momento durante la competencia y no se da información a los contrincantes sobre qué problema podría salir fácilmente o pronto.

Desventajas

El enviar las soluciones dependiendo del ritmo de la competencia puede generar penalidad extra a comparación de la estrategia cíclica, así como que si se envían múltiples soluciones que fallan por bugs, el tiempo de corrección puede ser alto y la moral puede disminuir.

Estrategia híbrida

Esta es una estrategia que combina partes de las dos anteriores.

- Resolver todos los problemas que uno estime que le pueden salir en un tiempo menor que un umbral fijado (pueden ser 10 o 15 minutos aproximadamente), saltearse los problemas no cumplan con dicha condición.
- Cuando se llegue a un problema que no salga tan rápido, irlo pensando por un tiempo prudente (máximo el doble del umbral) y luego saltar a otros problemas que aún no se hayan leído.
- Luego de haber leído todos los problemas, decidir un orden de dificultad estimado y analizar los problemas por orden de dificultad ascendente.

Estrategia híbrida

Esta es una estrategia que combina partes de las dos anteriores.

- Resolver todos los problemas que uno estime que le pueden salir en un tiempo menor que un umbral fijado (pueden ser 10 o 15 minutos aproximadamente), saltearse los problemas no cumplan con dicha condición.
- Cuando se llegue a un problema que no salga tan rápido, irlo pensando por un tiempo prudente (máximo el doble del umbral) y luego saltar a otros problemas que aún no se hayan leído.
- Luego de haber leído todos los problemas, decidir un orden de dificultad estimado y analizar los problemas por orden de dificultad ascendente.

Ventajas

La penalidad en los problemas "fáciles" para uno es baja debido a que se afrontan lo más pronto posible y hay menos probabilidad de estancamiento al leer todos los problemas.

Estrategia híbrida

Esta es una estrategia que combina partes de las dos anteriores.

- Resolver todos los problemas que uno estime que le pueden salir en un tiempo menor que un umbral fijado (pueden ser 10 o 15 minutos aproximadamente), saltearse los problemas no cumplan con dicha condición.
- Cuando se llegue a un problema que no salga tan rápido, irlo pensando por un tiempo prudente (máximo el doble del umbral) y luego saltar a otros problemas que aún no se hayan leído.
- Luego de haber leído todos los problemas, decidir un orden de dificultad estimado y analizar los problemas por orden de dificultad ascendente.

Ventajas

La penalidad en los problemas "fáciles" para uno es baja debido a que se afrontan lo más pronto posible y hay menos probabilidad de estancamiento al leer todos los problemas.

Desventajas

El enviar las soluciones dependiendo del ritmo de la competencia puede generar penalidad extra a comparación de la estrategia cíclica, así como que si se envían múltiples soluciones que fallan por bugs, el tiempo de corrección puede ser alto y la moral puede disminuir.

Problemas interactivos

Definición

- Un problema interactivo se caracteriza por tener información oculta que el concursante puede consultar bajo ciertas condiciones para deducir una respuesta solicitada.
- Siempre el enunciado del problema brinda el *Protocolo de interacción*, mediante el cual el concursante conocerá el formato con el cual realizar las consultas y dar su respuesta final.
- Algunos problemas interactivos manejan archivos, otros manejan funciones que uno debe diseñar para resolver el problema (no manipularemos nada más, solo una función que se nos da como plantilla) y otros leen y responden directamente en la entrada y salida estándar.

Formato Codeforces

En el caso de Codeforces, casi siempre basta con usar la función `fflush(stdout)` o imprimir usando `std::endl`. Veamos el siguiente ejemplo:

Formato Codeforces

En el caso de Codeforces, casi siempre basta con usar la función `fflush(stdout)` o imprimir usando `std::endl`. Veamos el siguiente ejemplo:

Guess the Number

Se nos pide determinar qué numero tiene oculto el juez usando solo una consulta en la que damos un valor x y se nos dice si este es mayor que el del juez o no. Un dato extra para saber el rango de búsqueda es que el número oculto está entre 1 y n ($1 \leqslant n \leqslant 10^6$). Determinar la respuesta en máximo 25 consultas.

Formato Codeforces

En el caso de Codeforces, casi siempre basta con usar la función `fflush(stdout)` o imprimir usando `std::endl`. Veamos el siguiente ejemplo:

Guess the Number

Se nos pide determinar qué numero tiene oculto el juez usando solo una consulta en la que damos un valor x y se nos dice si este es mayor que el del juez o no. Un dato extra para saber el rango de búsqueda es que el número oculto está entre 1 y n ($1 \leqslant n \leqslant 10^6$). Determinar la respuesta en máximo 25 consultas.

Solución

Podríamos aplicar búsqueda binaria para resolver el problema y la cantidad de consultas no excederá de 21.

Código en C++

Modelo de solución

```
void solve(){
        int n;
        scanf("%d", &n);
        int lo = 1, hi = n;
        while(lo < hi){
                int mi = lo + (hi - lo + 1) / 2;
                if(ask(mi)) lo = mi; // Si Numero oculto >= mi
                else hi = mi - 1;
        }
        answer(lo);
}
```

Mientras diseñemos correctamente las funciones `ask` y `answer`, podemos resolver el problema.

Formato OmegaUp

En el caso de OmegaUp, se tiene la librería `libinteractive`, por lo tanto se debe descargar un paquete de archivos para poder resolver el problema y testearlo localmente.

Formato OmegaUp

En el caso de OmegaUp, se tiene la librería `libinteractive`, por lo tanto se debe descargar un paquete de archivos para poder resolver el problema y testearlo localmente.

Linux

Nos basta descargar el archivo, extraer la información en una carpeta y modificar la plantilla de código que se nos ha brindado para resolver el problema.
Para ejecutar nuestra solución, deberemos usar

```
make run
```

Y deberemos darle la entrada por terminal. Otra opción es pasarle el archivo en el mismo comando:

```
make run < examples/sample.in
```

Formato OmegaUp

En el caso de OmegaUp, se tiene la librería `libinteractive`, por lo tanto se debe descargar un paquete de archivos para poder resolver el problema y testearlo localmente.

Linux

Nos basta descargar el archivo, extraer la información en una carpeta y modificar la plantilla de código que se nos ha brindado para resolver el problema.
Para ejecutar nuestra solución, deberemos usar

```
make run
```

Y deberemos darle la entrada por terminal. Otra opción es pasarle el archivo en el mismo comando:

```
make run < examples/sample.in
```

Windows

Se debe tener Code::Blocks instalado (versión MinGW) y basta con abrir el archivo de proyecto que se nos da en el ZIP. El comando `run` funciona de manera similar a Linux, solo que ya no es necesario el `make`. Se da la opción de usar `test` para probar el caso de prueba brindado en el ZIP (este lo pueden modificar en el proyecto para correr otros casos).

Debugging

Debuggear una solución

Si uno obtiene veredicto Wrong Answer o Runtime Error, lo más natural es buscar primero si nuestra implementación tiene algún tipo de error como división/módulo respecto a 0, acceso a memoria no reservada, etc.

Debuggear una solución

Si uno obtiene veredicto Wrong Answer o Runtime Error, lo más natural es buscar primero si nuestra implementación tiene algún tipo de error como división/módulo respecto a 0, acceso a memoria no reservada, etc.

Debug clásico

Lo que se suele hacer, más por la premura del tiempo, es imprimir luego de procesos clave del código, para verificar en dónde se genera el error. También se suele imprimir los valores de las variables. Sin embargo, esto puede resultar contraproducente si se usan muchas impresiones, pues se deberán comentar todas luego de arreglar el código. Esto puede costarnos un envío agónico a último segundo por no haber podido borrar los mensajes a tiempo.

Debuggear una solución

Si uno obtiene veredicto Wrong Answer o Runtime Error, lo más natural es buscar primero si nuestra implementación tiene algún tipo de error como división/módulo respecto a 0, acceso a memoria no reservada, etc.

Debug clásico

Lo que se suele hacer, más por la premura del tiempo, es imprimir luego de procesos clave del código, para verificar en dónde se genera el error. También se suele imprimir los valores de las variables. Sin embargo, esto puede resultar contraproducente si se usan muchas impresiones, pues se deberán comentar todas luego de arreglar el código. Esto puede costarnos un envío agónico a último segundo por no haber podido borrar los mensajes a tiempo.

GDB (Linux)

Es una herramienta que permite insertar pausas en la ejecución del código, ver los valores actuales que tiene cada variable y detectar errores. El usarla evita modificar el código directamente para ver lo que está mal.

Testear una solución

Si uno tiene veredicto Wrong Answer o Time Limit Exceeded, asumiendo que el código da los resultados que uno espera de su solución, toca revisar qué caso hace que falle.

Testear una solución

Si uno tiene veredicto Wrong Answer o Time Limit Exceeded, asumiendo que el código da los resultados que uno espera de su solución, toca revisar qué caso hace que falle.

Casos manuales

Probar los casos pequeños y los que tengan alguna condición de maximalidad (no necesariamente es el máximo valor posible, a veces es el número con mayor cantidad de divisores, etc.). También es bueno probar casos que tengan una estructura predefinida (todos los elementos iguales, si la estructura es un grafo, probar para un camino, para una estrella, etc.)

Testear una solución

Si uno tiene veredicto Wrong Answer o Time Limit Exceeded, asumiendo que el código da los resultados que uno espera de su solución, toca revisar qué caso hace que falle.

Casos manuales

Probar los casos pequeños y los que tengan alguna condición de maximalidad (no necesariamente es el máximo valor posible, a veces es el número con mayor cantidad de divisores, etc.). También es bueno probar casos que tengan una estructura predefinida (todos los elementos iguales, si la estructura es un grafo, probar para un camino, para una estrella, etc.)

Caso aleatoriamente generados

Se puede modificar el código a enviar para que haya dos formas de asignar los valores a las variables: leyendo o aleatoriamente. Para testear casos aleatorios podemos establecer una función que compare nuestra solución eficiente pero incorrecta con otra que sea ineficiente pero correcta (fuerza bruta), podríamos manejar archivos para hacer la verificación o sino hacer ambas funciones en el mismo código, lo cual dará un código un poco largo pero sin necesidad de manejar archivos.