competitive-programming-suite

Un paquete de instalación de comandos útiles para programación competitiva

Instalar el paquete

Para instalar el paquete ejecute en consola (WSL para usuarios Windows)

```
git clone https://github.com/LautaroLasorsa/competitive-programming-suite.git
cd competitive-programming-suit
bash cp-install.sh
```

Esto además creara los siguientes archivos y directorios:

```
$HOME/competitive-programming/
$HOME/competitive-programming/README.pdf # Este archivo en formato pdf
$HOME/competitive-programming/bin/ # Donde se guardan los scripts y sus modulos auxilia
$HOME/competitive-programming/notebook/
$HOME/competitive-programming/notebook/template.cpp
$HOME/competitive-programming/notebook/template.py
$HOME/competitive-programming/template-propuesta/
```

Y creará la variable de entorno **CP**$quecontienelaladirecciónde la carpeta“‘HOME/competitive-programming“‘. Se puede navegar directamente a esta carpeta haciendo

```
cd $CP
```

Comandos generales

Estos son los comandos que serán útiles para cualquiera que utilice este paquete.

cp-update

Este comando, que no lleva argumentos, actualiza el paquete a su última versión.

```
cp-update
```

cp-help

Este comando, que no lleva argumento, abre el archivo README.pdf (este archivo) en VSCODE.

```
cp-help
```

Comandos para competidores

Una lista de comandos útiles a la hora de resolver problemas de programación.

cp-cpp

Este comando permite compilar y ejecutar contra los casos de prueba un archivo cpp

```
cp-cpp A
```

Compila el archivo A.cpp y lo ejecuta contra todos los archivos A*.in

Además, crea un archivo A.print que contiene el código ejecutado y todo lo mostrado por consola.

Se puede agregar un segundo parámetro opcional, para indicar si queremos solo compilar el archivo (generando el ejecutable A) o solo correr el ejecutable contra los casos de prueba.

```
cp-cpp A build # Compilar
cp-cpp A run   # Ejecutar
```

Solo la parte de Ejecutar crea el archivo A.print

cp-python

Este comando permite ejecutar contra todos los casos de prueba un archivo .py (Python 3)

```
cp-python A
```

Ejecuta el archivo A.py contra todos los archivos A*.in

Además, crea un archivo A.print que contiene el código ejecutado y todo lo mostrado por consola.

cp-notebook

Este comando abre la carpeta `/usr/local/competitive-programming/notebook` en VS CODE.

cp-template

Recibe un parametro opcional para indicar si utilizar el template de C++ o de Python.

```
cp-template cpp A      # Copia template.cpp en A.cpp
cp-template python A   # Copia template.py en A.py
```

Tener en cuenta que los templates instalados originalmente son archivos en blanco y hay que poner lo que queramos en ellos.

cp-get-notebook

Permite acceder a notebooks de referencia que estén subidos a GitHub, abriendo el notebook en VS CODE. Si no existe, copia el repositorio como una carpeta dentro de `/usr/local/competitive-programming`.

```
cp-get-notebook nombre
```

Permite acceder al notebook indicado en el parámetro nombre.

Si se llama

```
cp-get-notebook
```

Lista los notebooks disponibles y una breve descripción de cada uno.

cp-update-notebooks

Actualiza todos los notebooks que se hayán descargado a su versión más reciente.

```
cp-update-notebooks
```

Material para desarrolladores de problemas

Se incluye una carpeta template-propuesta que se copia en la carpeta `$HOME/competitive-programming` y tiene la siguiente estructura.

- casos/: Una carpeta donde hay que guardar los casos de prueba del problema. Se recomienda seguir el formato "S{1}E{2}.in". Donde {1} son las subtareas que cumple el caso de prueba (poner todos los casos en una misma subtarea si no hay subtareas) y {2} es el número de caso (numeración única entre todos los casos)
- casos/generados/: Debe contener un archivo gen.py que genera los casos de prueba que se guardan en dicha carpeta. Dentro del archivo se explica cómo modificarlo para generar los casos de nuestro problema.
- soluciones/: Una carpeta con los códigos de las soluciones en C++, Python o Java.
- config.json: Un archivo con datos de configuración.
- corrector: Un archivo C++, Python o Java que dado un caso de prueba, la respuesta de la solución oficial. Este archivo es opcional, de no estar se considerará si una salida es correcta si y solo si es igual a la producida por la solución oficial.
- enunciado.md : Un archivo MarkDown que sirve de plantilla para escribir el enunciado del problema.

Además, se ofrecen los siguientes comandos:

```
cp-propuesta A # Copia el template en el directorio A
cp-ptoolkit    # Un comando con diversas herramientas para nuestro problema
```

Si ejecutamos cp-ptoolkit sin pasar ningún argumento veremos las opciones disponibles:

```
operaciones disponibles:
        l, limpiar
                quitar ejecutables y casos creados por generadores
        g, generar
                ejecutar generadores
        v, verificar
                ejecutar verificadores de casos
        e, empaquetar
                crea un zip con los casos
        r, resolver
                ejecutar las soluciones
        t, todo
                limpiar, generar, verificar y resolver
```

Es importante tener en cuenta que cp-ptoolkit se debe ejecutar en la raíz de un directorio con la estructura de template-propuesta y solo garantiza funcionar correctamente si la estructura.

Además, tomará como solución de referencia a la primera en orden lexicográfico de la carpeta soluciones. Por eso, por ejemplo, se le puede poner 00 como prefijo a la solución que queremos que sea utilizada como referencia.