

Informe del Laboratorio 02

Tema 2: Análisis de algoritmos (parte 1)

Nota

Estudiante	Escuela	Asignatura
Rodrigo E. Infanzón Acosta rinfanzona@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Lenguaje de Programación 3 Semestre IV

Laboratorio	Tema	Duración
02	Análisis de algoritmos	02 horas académicas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	22/03/24	27/03/24

Actividades a realizar:

1. Elaborar la solución al problema usando Git y GitHub, seleccionado por el profesor, denominado **"2199. Digital Root"**, extraído de la plataforma omegaUp: <https://omegaup.com/arena/problem/DigitalRoot/>
2. Presentar el método iterativo y el método recursivo.
3. Elegir un problema en la plataforma omegaUp: <https://omegaup.com/> y solucionarlo, presentando su debido informe y posterior explicación.
4. Envíe satisfactoriamente todas sus implementaciones a la plataforma omegaUp.
5. Responder la pregunta: ¿Cómo podríamos saber con exactitud, cuál de todas las propuestas algorítmicas es la más óptima? Sugiera soluciones y explique a nivel conceptual.
6. Utilizar todas las recomendaciones dadas por el docente en clase, tales como:
 - **Antecedentes:** describir antecedentes previos que sean necesarios para desarrollar el laboratorio. Las entregadas por el docente y/o las que se buscaron personalmente.
 - **Commits:** elaborar la lista de envíos que permitirán culminar el laboratorio, previo a la implementación.
 - **Source:** explicar porciones de código fuente importantes, trascendentales que permitieron resolver el laboratorio y que reflejen su particularidad única, sólo en trabajos grupales se permite duplicidad.
 - **Ejecución:** muestra comandos, capturas de pantalla, explicando la forma de replicar y ejecutar el entregable del laboratorio.

7. Entregables:

- URL al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia de archivos: binarios, objetos, archivos temporales. Incluir archivos de especificación como: packages.json, requirements.txt o README.md.
- No olvidar que el docente debe ser siempre colaborador a su repositorio que debe ser privado. (Usuario del docente: **@rescobedoulasalle**).
- Se debe de describir sólo los commits más importantes que marcaron hitos en su trabajo, adjuntando capturas de pantalla, del commit, porciones de código fuente, evidencia de sus ejecuciones y pruebas.
- En el informe siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
- Agregar la estructura de directorios y archivos de su laboratorio.

Recursos y herramientas utilizados:

1. Sistema operativo utilizado: Windows 10 pro 22H2 de 64 bits (SO. 19045.4170).
2. Hardware: Ryzen 5 3550H 2.10 GHz, RAM 16 GB DDR4 2400 MHz.
3. Visustin v8.08 Demo
4. Git (versión 2.44.0).
5. Visustin v8.08
6. Cuenta de GitHub creada con el correo institucional proporcionado por la Universidad La Salle de Arequipa (rinfanzona@ulasalle.du.pe).
7. Conocimientos básicos en Git.
8. Conocimientos básicos sobre programación.

Información del repositorio en GitHub:

1. Enlace del repositorio en GitHub: <https://github.com/RodrigoStranger/lp3-24a>
2. Enlace del repositorio para el laboratorio 02 en GitHub:
<https://github.com/RodrigoStranger/lp3-24a/tree/main/lab02>

Actividades realizadas con el repositorio de GitHub en Local:

1. En el desarrollo del segundo laboratorio, desarrollamos algunos algoritmos de la plataforma omegaUp, en las cuales, desarrollé el método iterativo y recursivo del problema “Digital Root”, añadiendo mis algoritmos al área de preparación, haciendo commit y, por último, empujando al servidor de GitHub. Al llegar a casa, solo tuve que actualizar mi repositorio local, para ello, nos ubicamos en nuestro directorio principal lp3-24a, abrimos Git Bash o la línea de comandos y ejecutamos el comando: **git pull**

```
C:\lp3-24a>git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 963 bytes | 24.00 KiB/s, done.
From https://github.com/RodrigoStranger/lp3-24a
  5767191..6abd8bc  main    -> origin/main
Updating 5767191..6abd8bc
Fast-forward
 README.md | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

C:\lp3-24a>
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Windows

Resolución del problema de la plataforma omegaUp “Digital Root”:

Durante el segundo laboratorio de programación, nuestro docente encargado nos explicó el análisis de los algoritmos, empleando los conceptos de análisis y diseño de un problema primero, antes de poder codificarlo, por consiguiente, nos solicitó que apliquemos el análisis al problema expuesto por él de la plataforma omegaUp “Digital Root”, presentando una solución iterativa y otra recursiva, dicho problema es el siguiente:

La raíz digital de un entero positivo es calculada mediante la suma de cada dígito de ese entero. Si el resultado es un entero de un solo dígito entonces esa es su raíz digital. Si el resultado tiene dos o más dígitos, esos dígitos son sumados y el proceso se repite hasta obtener un número de un solo dígito. Por ejemplo, considere el entero positivo 24. Sumar 2 y 4 resulta en 6. Ya que 6 es un solo dígito, 6 es la raíz digital de 24. Ahora considere el entero positivo 39. Sumar 3 y 9 resulta en 12. Ya que 12 no es un solo dígito, el proceso debe ser repetido. Sumando 1 y 2 resulta en 3, de un solo dígito y por tanto la raíz digital de 39.

Entrada: la entrada consistirá de una lista de enteros positivos, uno por línea. El fin de la entrada estará marcada por un entero igual a 0.

Salida: por cada entero en la entrada, escriba su raíz digital, uno por línea en la salida.

Ejemplo:

Entrada	Salida
24	6
39	3
0	

Imagen con propiedad del autor ©omegaUp

Solución en método iterativo:

1. Defino una variable entrada que inicialmente es igual al número n que se pasa a la función.
2. Se define una variable booleana “seguir”, que indica si se debe continuar el proceso de calcular la raíz digital.
3. Se entra en un bucle “while” que continuará mientras seguir sea verdadero.
4. Dentro del bucle, declaro una variable entera, que representa el último dígito de “entrada” (obtenido mediante el operador módulo %) y una variable residuo que representa el resto de la división de entrada por 10.
5. Sumo: “entero” y “residuo”, verifico si la suma es menor que 9. Si es así, significa que hemos obtenido un solo dígito, por lo que establecemos seguir en “falso”, para salir del bucle.
6. Si la suma no es menor que 9, asigno a “entrada” la suma de entero y residuo, lo que representa el nuevo número del que se calculará la raíz digital en la siguiente iteración del bucle.
7. Una vez que el bucle termina, se devuelve el valor final de “entrada”, que representa la raíz digital del número original “ n ”.

8. En el int main () verificamos si el numero es 0, entonces no imprime la raíz digital ya que 0 no posee raíz digital.

Mi lógica posee:

- Uso de una función.
- Uso de bucles condicionales “while”
- Matemática básica
- Programación básica.

Diseño del algoritmo:

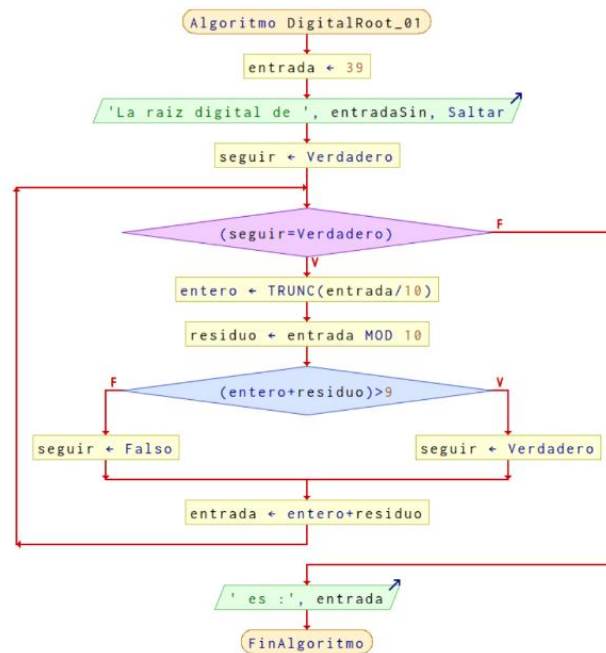


Imagen con propiedad del autor ©Richart Escobedo Quispe, ©UlaSalle (1)

Codificación:

Una vez diseñado nuestra lógica resolviendo el problema, podemos iniciar con la codificación del programa. Para ello, debo ubicarme en mi directorio: c:/lp3-24a/lab02/exercises/ y crear un archivo denominado “digitalRootLoop”, en dicho archivo se debe de codificar el programa para posteriormente:

- Añadirlo al área de preparación.
- Hacer un commit del archivo.
- Empujar el archivo al servidor GitHub.

Posteriormente abrir el archivo y codificar:

```

C:\lp3-24a\lab02\exercises>git add digitalRootLoop.cpp
C:\lp3-24a\lab02\exercises>git commit -m "version final"
[main 7ba4765] version final
1 file changed, 20 insertions(+), 14 deletions(-)
C:\lp3-24a\lab02\exercises>git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 643 bytes | 214.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/RodrigoStranger/lp3-24a
e147245..7ba4765 main -> main
  
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Windows

En código:

```
1 #include <iostream>
2 using namespace std;
3
4 int DigitalRoot (int n) {
5     int entrada = n;
6     bool seguir = true;
7     while (seguir == true) {
8         int ent = 0;
9         int res = 0;
10        ent = entrada % 10;
11        res = entrada / 10;
12        if((ent + res) > 9){
13            seguir = true;
14        } else {
15            seguir = false;
16        }
17        entrada = ent + res;
18    }
19    return entrada;
20 }
21
22 int main () {
23     int n;
24     cout<<"Ingrese un numero: ";
25     cin>>n;
26     if(n!=0){
27         cout<<"El numero digital de "<<n<<" es: "<<DigitalRoot(n);
28     }
29     return 0;
30 }
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

Ejecución del código: ejemplo cuando n: 39

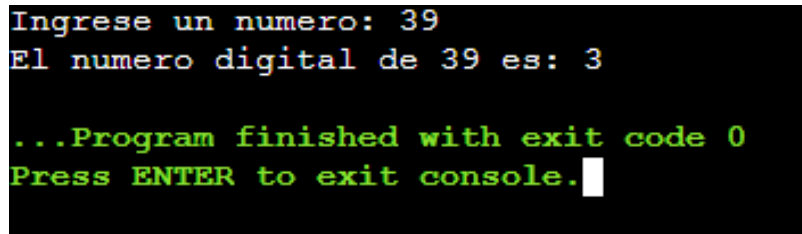


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

Solución en método recursivo:

1. Cuando se trata de un método recursivo, técnicamente debe de tener una condición de parada, para este caso, uso como condición que si el numero es menor a 10, entonces no hay raíz digital que calcular, consecuentemente se retorna n.
2. En caso sea mayor a 10, declaro dos variables: “ent” y “res”, las cuales guardara la división y el resto correspondientemente, sumándolos en una variable llamada “suma”.
3. Después, se evalúa si la suma de la división con el resto es menor a 9, de ser menores la función se llama a si misma, cumpliendo el método recursivo.
4. Caso contrario, se retorna la “suma” de la división con el residuo.

Diseño del algoritmo:

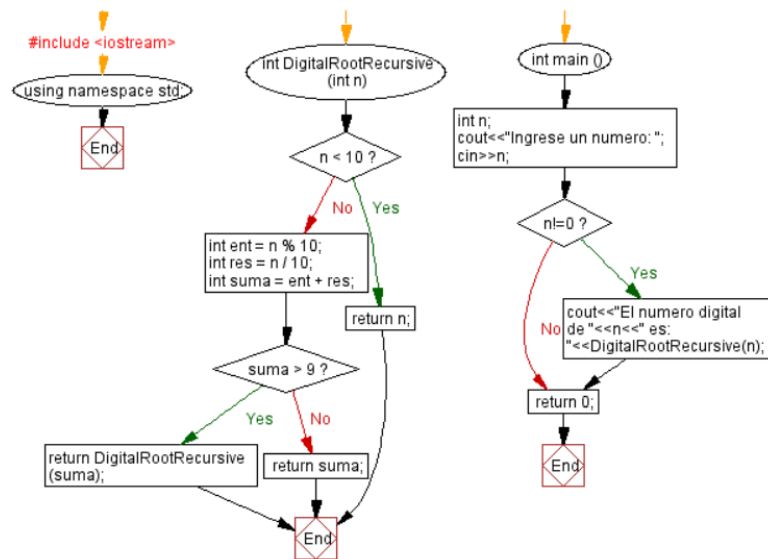


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Visustin

Codificación:

Una vez diseñado nuestra lógica resolviendo el problema, podemos iniciar con la codificación del programa. Para ello, debo ubicarme en mi directorio: c:/lp3-24a/lab02/exercises/ y crear un archivo denominado “digitalRootRecursive”, en dicho archivo se debe de codificar el programa para posteriormente:

- Añadirlo al área de preparación.
- Hacer un commit del archivo.
- Empujar el archivo al servidor GitHub.

Posteriormente abrir el archivo y codificar:

```

C:\lp3-24a\lab02\exercises>git add digitalRootRecursive.cpp

C:\lp3-24a\lab02\exercises>git commit -m "version final"
[main 974fed6] version final
1 file changed, 20 insertions(+), 12 deletions(-)

C:\lp3-24a\lab02\exercises>git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 635 bytes | 317.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/RodrigoStranger/lp3-24a
   b00707b..974fed6  main -> main

C:\lp3-24a\lab02\exercises>
  
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Windows

Descripción de Commits:

- **Commit “versión final”:** muestra la versión final del código, agregando una interfaz amigable para poder leer el número, se agregó también la condicional restante, tanto en los dos programas, digitalRootLoop y digitalRootRecursive.

En código:

```
1 #include <iostream>
2 using namespace std;
3
4 int DigitalRootRecursive(int n) {
5     if (n < 10) {return n;}
6
7     int ent = n % 10;
8     int res = n / 10;
9     int suma = ent + res;
10
11     if (suma > 9) {
12         return DigitalRootRecursive(suma);
13     } else {
14         return suma;
15     }
16 }
17 int main () {
18     int n;
19     cout<<"Ingrese un numero: ";
20     cin>>n;
21     if(n!=0) {
22         cout<<"El numero digital de "<<n<<" es: "<<DigitalRootRecursive(n);
23     }
24     return 0;
25 }
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

Ejecución del código: ejemplo cuando n: 154

```
Ingrese un numero: 154
El numero digital de 154 es: 1

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

OmegaUp:

Envíos

📅 Fecha y hora	Lenguaje	Porcentaje	Ejecución	Salida	📊 Memoria	🕒 Tiempo	Acciones
2024-03-23 21:39	cpp20-gcc	100.00%	Terminada	Correcta ✓	3.43 MB	0.00 s	🔍
2024-03-23 21:16	cpp17-clang	100.00%	Terminada	Correcta ✓	3.29 MB	0.00 s	🔍
2024-03-23 21:14	cpp17-clang	0.00%	Terminada	Incorrecta ✗	3.38 MB	0.00 s	🔍
2024-03-23 20:59	cpp17-clang	0.00%	Terminada	Incorrecta ✗	3.13 MB	0.01 s	🔍
2024-03-23 20:58	cpp17-clang	0.00%	Terminada	Incorrecta ✗	3.39 MB	0.00 s	🔍

Imagen con propiedad del autor ©OmegaUp

Nota: al parecer el método recursivo utiliza un poco mas de memoria.

Elección de un problema en omegaUp: 8586. Calculadora-

Factorial: <https://omegaup.com/arena/problem/Calculadora-Factorial/#problems>

Los amigos de James están batallando para calcular la factorial de varios números ya que no tienen calculadoras y no tienen acceso a internet porque utilizaron el internet para controlar un robot. Por lo que te pidió ayuda a ti, para que le programes una calculadora de factorial.

Entrada: será un número entero **N** del cual debes obtener su factorial (los números son menores a 15).

Salida: imprimir únicamente el resultado de la factorial de **N**.

Solución:

Analizando el problema, obtener la factorial de un número requiere que los números anteriores a él, necesitan multiplicarse entre sí, hasta llegar al 1, por ejemplo, la factorial de 5 es $5*4*3*2*1=120$.

Por lo cual, se puede utilizar un bucle, o también una función recursiva ya que ambas metodologías permiten calcular la factorial de un número al realizar operaciones repetitivas de multiplicación.

La utilización de un bucle, como un bucle for o while, implica iterar desde el número dado hacia abajo hasta 1, multiplicando los números en cada iteración.

Por otro lado, una función recursiva es aquella que se llama a sí misma para resolver un problema más pequeño, en este caso, para calcular la factorial de un número, la función se llama a sí misma con un argumento más pequeño hasta llegar al caso base que es cuando el número es 1.

Diseño del algoritmo:

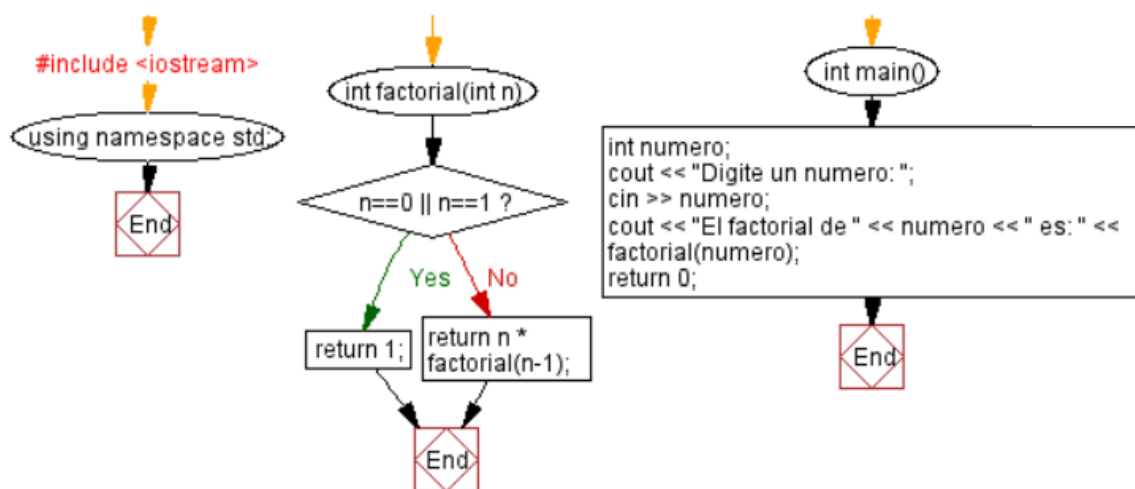


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Visustin

Codificación:

Una vez diseñado nuestra lógica resolviendo el problema, podemos iniciar con la codificación del programa. Para ello, debo ubicarme en mi directorio: `c:/lp3-24a/lab02/exercises/` y crear un archivo denominado “factorial-de-n”, en dicho archivo se debe de codificar el programa para posteriormente:

- Añadirlo al área de preparación.
- Hacer un commit del archivo.
- Empujar el archivo al servidor GitHub.

Posteriormente abrir el archivo y codificar:

```
Microsoft Windows [Versión 10.0.19045.4170]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\lp3-24a\lab02\exercises>type NUL > factorial-de-n.cpp

C:\lp3-24a\lab02\exercises>notepad factorial-de-n.cpp

C:\lp3-24a\lab02\exercises>git add factorial-de-n.cpp

C:\lp3-24a\lab02\exercises>git commit -m "version final"
[main 74361f2] version final
1 file changed, 17 insertions(+)
create mode 100644 lab02/exercises/factorial-de-n.cpp

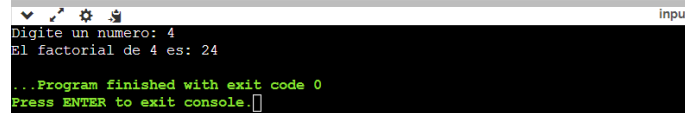
C:\lp3-24a\lab02\exercises>git push origin main
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 608 bytes | 304.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/RodrigoStranger/lp3-24a
974fed6..74361f2 main -> main

C:\lp3-24a\lab02\exercises>
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Windows

En código y ejecución:

```
1 #include <iostream>
2 using namespace std;
3
4 int factorial(int n){
5     if(n==0 || n==1){
6         return 1;
7     } else {
8         return n * factorial(n-1);
9     }
10 }
11 int main() {
12     int numero;
13     cout << "Digite un numero: ";
14     cin >> numero;
15     cout << "El factorial de " << numero << " es: " << factorial(numero);
16     return 0;
17 }
18
```



```
input
Digite un numero: 4
El factorial de 4 es: 24
...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

Se observa que al introducir el número 4 en este caso, entra en la función factorial, ingresa por la condicional “si 4 es igual a 0 o si es igual a 1”, al ver que no se cumple, nos retorna la multiplicación del mismo numero “4” * factorial de 4-1, lo cual es 3, y se llama a sí misma la función factorial hasta que cumpla la condición de parada, por lo tanto: $4 * \text{factorial}(3) * \text{factorial}(2) * 1 = 24$.

OmegaUp:

Envíos



Fecha y hora	Lenguaje	Porcentaje	Ejecución	Salida	Memoria	Tiempo	Acciones
2024-03-23 23:16	cpp17-gcc	100.00%	Terminada	Correcta 	3.30 MB	0.01 s	
Nuevo envío							
« < 1 > »							

Imagen con propiedad del autor ©OmegaUp

Versión Iterativa:

```

1  #include <iostream>
2  using namespace std;
3  int main() {
4      int factorial = 1 , numero;
5      cout<<"Digite un numero: ";
6      cin>>numero;
7      for(int i=2;i<=numero;i++){
8          factorial = factorial * i;
9      }
10     cout<<"El factorial de "<<numero<<" es: "<<factorial<<endl;
11     return 0;
12 }

```

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©GDB Online

Diagrama de flujo:

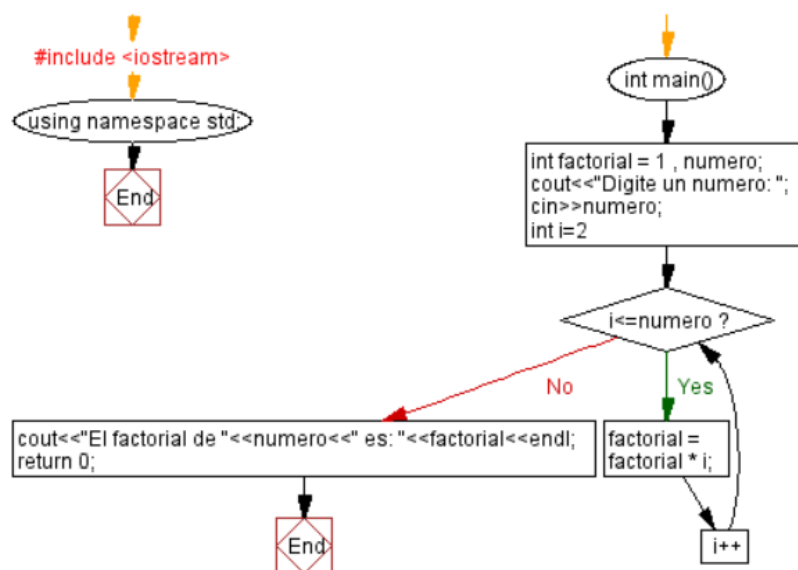


Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Visustin

OmegaUp:

Envíos

📅 Fecha y hora	Lenguaje	Porcentaje	Ejecución	Salida	📊 Memoria	🕒 Tiempo	Acciones
2024-03-27 07:46	cpp17-gcc	100.00%	Terminada	Correcta ✓	3.37 MB	0.01 s	🔍
2024-03-23 23:17	cpp17-gcc	100.00%	Terminada	Correcta ✓	3.30 MB	0.01 s	🔍
Nuevo envío							
<div> « < 1 > » </div>							

Imagen con propiedad del autor ©OmegaUp

¿Cómo podríamos saber con exactitud, cuál de todas las propuestas algorítmicas es la más óptima? Sugiera soluciones y explique a nivel conceptual.

Determinar cuál algoritmo es el más óptimo para un problema específico puede depender de varios factores, como el tamaño de la entrada, los recursos disponibles (hardware), y las restricciones del sistema. Por ello debemos realizar algunas estrategias para analizar la eficiencia de los algoritmos y determinar cuál debe ser el más óptimo:

1. **Análisis de Complejidad:** podemos evaluar la complejidad computacional de los algoritmos para comprender su comportamiento en términos de tiempo de ejecución y uso de memoria, tal como el estándar de medición Big O.
2. **Selección de Conjuntos de Datos Representativos:** podemos elegir conjuntos de datos que sean representativos de los casos de uso reales del algoritmo. Esto puede incluir conjuntos de datos pequeños, medianos y grandes, así como casos extremos que puedan desafiar el rendimiento del algoritmo.
3. **Comparación de Casos Extremos:** podemos identificar situaciones extremas en las que un algoritmo pueda mostrar un rendimiento excepcionalmente bueno o malo, lo que ayuda a comprender sus fortalezas y debilidades.

Recursividad vs Iteratividad:

La recursividad y la iteratividad son dos técnicas fundamentales en programación que se utilizan para resolver problemas repetitivos de manera eficiente. Ambas tienen sus ventajas y desventajas, y la elección entre una u otra depende del problema específico, así como de las características del lenguaje de programación y del entorno en el que se está trabajando. A continuación, se presenta un análisis comparativo de ambas técnicas:

Recursividad:

Definición: La recursividad es una técnica mediante la cual una función se llama a sí misma directa o indirectamente para resolver un problema. En esencia, un problema se divide en sub problemas más pequeños que se resuelven de forma recursiva hasta alcanzar un caso base.

Posee las siguientes características:

- **Facilidad de comprensión:** La recursividad puede ser más fácil de entender para algunos programadores, especialmente cuando el problema se puede expresar de manera natural en términos recursivos.
- **Facilidad de implementación:** En algunos casos, la implementación de una solución recursiva es más simple y directa que su equivalente iterativo.
- **Consumo de memoria:** La recursividad puede consumir más memoria que la iteratividad, especialmente si no se manejan adecuadamente las llamadas recursivas o si hay muchos niveles de recursión.
- **Eficiencia:** En general, la recursividad puede ser menos eficiente en términos de tiempo de ejecución y consumo de recursos que una solución iterativa equivalente. Esto se debe a la sobrecarga asociada con las llamadas recursivas y la gestión de la pila de llamadas.

Iteratividad:

Definición: La iteratividad implica utilizar bucles (como “for”, “while”) para repetir un conjunto de instrucciones hasta que se cumple una condición de terminación.

Posee las siguientes características:

- **Facilidad de comprensión:** Algunos programadores pueden encontrar más fácil entender y seguir el flujo de ejecución en soluciones iterativas, especialmente cuando el problema se puede abordar de manera secuencial.
- **Facilidad de implementación:** La iteratividad puede requerir más código que la recursividad en algunos casos, pero suele ser más eficiente en términos de recursos.
- **Consumo de memoria:** Las soluciones iterativas tienden a consumir menos memoria que las recursivas, ya que no hay llamadas a funciones adicionales que se acumulen en la pila de llamadas.
- **Eficiencia:** En muchos casos, las soluciones iterativas son más eficientes en términos de tiempo de ejecución y consumo de recursos que sus contrapartes recursivas. Esto se debe a la falta de sobrecarga asociada con las llamadas recursivas y la gestión de la pila de llamadas.

Consideraciones Finales:

- La recursividad y la iteratividad son técnicas complementarias que pueden utilizarse para resolver una variedad de problemas en programación.
- La elección entre una solución recursiva y una iterativa depende de factores como la naturaleza del problema, la eficiencia requerida, la legibilidad del código y las características del lenguaje de programación utilizado.
- Es importante comprender las ventajas y desventajas de cada técnica y seleccionar la más apropiada para cada situación específica.
- En muchos casos, es posible convertir una solución recursiva en una iterativa y viceversa, aunque el proceso puede requerir cierto grado de refactorización y ajuste del algoritmo original.

Estructura del laboratorio 02:

```
|--- lab02 [DIRECTORIO]
    |--- exercises [DIRECTORIO]
        |--- digitalRootLoop.cpp
        |--- digitalRootRecursive.cpp
        |--- factorial-de-n.cpp
    |--- report [DIRECTORIO]
        |--- report02.dox
        |--- report02.pdf
```

3 directorios, 5 archivos|

Imagen con propiedad del autor ©Rodrigo Infanzón Acosta, ©Notepad

Referencias:

1. https://www.onlinegdb.com/online_c++_compiler
2. <https://github.com/>
3. <https://pseint.sourceforge.net/>
4. <https://omegaup.com/>
5. <https://git-scm.com/>
6. <https://www.aivosto.com/visustin-es.html>

Referencias Bibliográficas:

1. Escobedo, R. (2023). *Laboratorio 02: Análisis de Algoritmos*. Recuperado de: https://drive.google.com/file/d/1DLXsEdN6M7YeWcoc80xPP_8uoIjDWpr5/view
2. Martínez, M. (2019). *Diferencia entre iteración y recursividad*. Recuperado de: https://codigofacilito.com/articulos/articulo_16_10_2019_16_22_35

Calificación:

Rúbrica para el contenido de informes y evidencias

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
GitHub	El repositorio se pudo clonar y se evidencia la estructura adecuada para revisar los entregables. (Se descontará puntos por error u observación)	4	X	4	
Commits	Hay porciones de código fuente asociado a los commits planificados con explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Ejecución	Se incluyen comandos para ejecuciones y pruebas del código fuente explicadas gradualmente que permitirán replicar el proyecto. (Se descontará puntos por cada omisión)	4	X	4	
Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
Ortografía	El documento no muestra errores ortográficos. (Se descontará puntos por error encontrado)	2	X	2	
Madurez	El informe muestra de manera general una evolución de la madurez del código fuente con explicaciones puntuales pero precisas, agregando diagramas generados a partir del código fuente y refleja un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	