UTN – FACULTAD REGIONAL BUENOS AIRES
Cátedra: Sintaxis y Semántica de los Lenguajes – 2024
CURSO K2055

TRABAJO PRÁCTICO N°2
"TEÓRICO"

GRUPO 12

Apellido, Nombre	N° de Legajo	Correo institucional
Alvarez, Ramiro.	213.463-9	raalvarez@frba.utn.edu.ar
Anetta, Agustín.	177.343-4	aannetta@frba.utn.edu.ar
Gonzalez, Camila Valentina.	202.703-3	camigonzalez@frba.utn.edu.ar
Ramos, Franco Juan.	214.048-2	frramos@frba.utn.edu.ar
Sosa Traversa, Oriana	213.376-3	ososa@frba.utn.edu.ar

DOCENTE: **Leituz, Roxana.**

Fecha de entrega	**Nota**	**Fecha de aprobación**
9/08/2024		

Comentarios

Índice

1. Consigna.

Investigar dos lenguajes de programación excluidos: C y C++, buscar las BNF del lenguaje elegido y desarrollar una presentación sobre los siguientes temas:

- Breve historia del lenguaje (“BREVE”)
- Comparación centrada en la forma en que los lenguajes manejan LAS FUNCIONES DE ORDENAMIENTO(SORT) con un ejemplo comparando el rendimiento.

2. Selección de Lenguajes.

Para el presente trabajo práctico, se decidió realizar la investigación de los siguientes lenguajes de programación:

1. Fortran.
2. Python.

La selección de estos lenguajes se basó en la popularidad que han tenido o tienen a lo largo del tiempo. Fortran fue el primer lenguaje de programación de alto nivel, llegando a ser dominado por cerca del 60% de los programadores en 1965. En 2019, aproximadamente el 25% de los programadores dominaban Python.

3. BNF

3.1 Fortran.

La sintaxis de los programas Fortran se describe utilizando una variante de la forma Backus-Naur (BNF), la misma se puede consultar en la siguiente página:
https://techpubs.jurassic.nl/library/manuals/3000/007-3694-006/sgi_html/ch01.html

3.2. Python.

La gramática completa de Python se puede consultar la siguiente página:
https://docs.python.org/3/reference/grammar.html

4. Historia

4.1. Fortran.

Fortran es un lenguaje de alto nivel utilizado en matemáticas y áreas científicas. Sus orígenes se remontan a finales de 1953, cuando IBM buscaba una alternativa más práctica al lenguaje ensamblador para programar su computadora central IBM 704.

En 1956, se publicó un manual dirigido al público porque los clientes dudaban en usar un lenguaje de alto nivel a menos que su compilador pudiera generar código con un rendimiento comparable al del código ensamblador hecho a mano.

A pesar del escepticismo inicial, Fortran redujo en veinte veces el número de sentencias necesarias para operar una máquina, ganando rápidamente aceptación. Fortran fue ampliamente adoptado por científicos para escribir programas numéricamente intensivos, lo que incentivó a los desarrolladores a crear compiladores más rápidos y eficientes.

La inclusión de tipos de datos y aritmética de números complejos amplió su rango de aplicaciones, haciéndolo especialmente adecuado para ingeniería eléctrica y otras aplicaciones técnicas. Hacia 1960, había versiones de Fortran disponibles para las computadoras IBM 709, 650, 1620 y 7090.

La creciente popularidad de Fortran impulsó a otros fabricantes de computadoras a ofrecer compiladores Fortran para sus máquinas. Para 1963, existían más de 40 compiladores Fortran.

Debido a esto, Fortran es considerado el primer lenguaje de programación ampliamente utilizado y soportado en diversas arquitecturas de computadoras. Su desarrollo estuvo estrechamente vinculado con la evolución temprana de la tecnología de compiladores, y muchos avances en teoría y diseño de compiladores fueron motivados por la necesidad de generar código eficiente para programas en Fortran.

4.2. Python.

Python fue creado en 1989 en Países Bajos por Guido van Rossum como sucesor del lenguaje ABC. ABC es un lenguaje que comparte algunas características clave con Python, como la no necesidad de declarar variables. Sin embargo, a Van Rossum no le gustaba la falta de extensibilidad de ABC, lo que le causaba problemas al interactuar con el sistema operativo Amoeba, con el que trabajaba a diario. Esto lo llevó a crear su propio lenguaje, modificando esta característica como idea principal. El nombre Python proviene de la afición de su creador por los humoristas británicos Monty Python.

Python es un lenguaje de programación de alto nivel, interpretado, cuya filosofía destaca por la legibilidad de su código. Es un lenguaje multiparadigma, ya que admite la programación orientada a objetos, la programación imperativa y, en menor medida, la programación funcional. Además, es interpretado, dinámico y multiplataforma.

5. Comparación de Funciones de Ordenamiento (Sort):

Las funciones de ordenamiento son algoritmos que ponen elementos de una lista o un vector en una secuencia dada por una relación de orden.

5.1. Fortran.

En Fortran, las funciones de ordenamiento no están incluidas en la biblioteca estándar, por lo que generalmente se implementan manualmente o se usan bibliotecas externas.

Ejemplo de ordenamiento de burbuja en Fortran:

```
program ordenamiento_de_burbuja
  implicit none
  integer, parameter :: cantidad = 10
  integer :: i, j, aux
  integer :: vector(cantidad) = (/ 10, 2, 8, 6, 7, 5, 4, 3, 1, 9 /)

  do i = 1, cantidad -1
    do j = 1, cantidad -i
      if (vector(j) > vector(j+1)) then
        aux = vector (j)
        vector(j) = vector (j+1)
        vector(j+1) = aux
      end if
    end do
  end do

  print *, 'El vector ordenado:'
  print *, vector

end program ordenamiento_de_burbuja
```

5.2. Python.

Python incluye funciones de ordenamiento muy eficientes en su biblioteca estándar, como sorted() y el método sort() para listas , que usan el algoritmo Timsort:

Ejemplo de ordenamiento con sorted() en Python:

```
vector = [10, 2, 8, 6, 7, 5, 4, 3, 1, 9]

vector_ordenado = sorted(vector)

print("'El vector ordenado:")
print(vector_ordenado)
```

Timsort es un algoritmo de ordenamiento híbrido que combina técnicas de ordenamiento por inserción y por mezcla (merge sort). Primero, toma el array original y lo divide en una cierta

cantidad de subarrays llamados runs. Estos runs son secuencias ya ordenadas de manera ascendente o descendente, ya que las descendentes se pueden reordenar fácilmente invirtiendo los valores.

Estas secuencias se conocen como runs naturales. Además, se define un tamaño mínimo de run que, en la práctica, es una constante fija (por ejemplo, 32 elementos).

Primero, se identifican las runs naturales y luego se dividen o fusionan para cumplir con el tamaño mínimo, asegurando que se mantenga lo más cercanas posible al orden ascendente o descendente. Luego, se ordenan los runs: se comparan dos elementos adyacentes y, si el primero es más pequeño que el segundo, se sigue con el segundo comparado con el tercero. Si el tercero es más pequeño que el segundo, se invierten sus posiciones y se compara el tercero con el primero. Si están ordenados, se dejan; si no, se invierten. Este proceso continúa desde la ubicación de la modificación osea se sigue comparando el segundo con el cuarto. Este proceso continúa hasta tener todo el run ordenado.

Una vez ordenados todos los runs, se comparan los primeros elementos de cada run, y el más pequeño se coloca como el primer elemento de la lista final ordenada. Luego, se vuelve a comparar los primeros elementos restantes junto con el segundo de la run de donde se tomó el primero, y así sucesivamente hasta terminar con todas las runs o que quede solo una, la cual se coloca al final de la lista.

Otra característica clave de Timsort es que, en caso de encontrar una run que esté total o parcialmente ordenada, utiliza métodos que consumen menos recursos. Para ello, primero analiza los primeros elementos y, si no hay cambio de orden, toma un valor promedio y analiza si los siguientes son más grandes, asegurando que no hay un valor menor más adelante en la lista.

6. Comparación de Rendimiento

Para comparar el rendimiento de las funciones de ordenamiento en Fortran y Python, se realizaron dos pruebas:

1. Medición del tiempo que tarda en ordenar cada programa una lista con 10.000 elementos aleatorios con ordenamiento burbuja.
2. Medición del tiempo que tarda en ordenar cada programa una lista con 1.000.000 elementos aleatorios con Timsort.

6.1. Ordenamiento Burbuja.

6.1.1 Fortran.

OrdenamientoBurbuja.f95

```
1   program ordenamiento_de_burbuja
2     implicit none
3     integer, parameter :: cantidad = 10000
4     integer :: i, j, aux
5     integer :: vector(cantidad)
6     real :: tiempo_inicial, tiempo_final
7     real :: vector_real(cantidad)
8
9     call random_seed()
10    call random_number(vector_real)
11    vector = int(vector_real * cantidad)
12
13    call cpu_time(tiempo_inicial)  ! Inicia el cronómetro
14
15    do i = 1, cantidad - 1
16      do j = 1, cantidad - i
17        if (vector(j) > vector(j+1)) then
18          aux = vector(j)
19          vector(j) = vector(j+1)
20          vector(j+1) = aux
21        end if
22      end do
23    end do
24
25    call cpu_time(tiempo_final)  ! Detiene el cronómetro
26
27    print *, 'El vector ordenado:'
28    print *, vector(1:10)  ! Imprimir solo los primeros 10 elementos
29    print *, 'Tiempo de ejecución (segundos):', tiempo_final - tiempo_inicial
30
31  end program ordenamiento_de_burbuja
32
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] cd "e:\Proyectos - Estudios\SSL\TP2\" && gfortran OrdenamientoBurbuja.f95 -o OrdenamientoBurbuja && "e:\Proyectos - Estu
 El vector ordenado:
           0           0           0           4           5           5           6           6           7           8
 Tiempo de ejecución (segundos):  0.171875000

[Done] exited with code=0 in 0.399 seconds
```

6.1.1 Python:

OrdenamientoBurbuja.py > ordenamiento_burbuja

```python
import time
import random

# Generar un vector aleatorio con 10,000 elementos
vector = [random.randint(1, 10000) for _ in range(10000)]

def ordenamiento_burbuja(vec):
    n = len(vec)
    for i in range(n):
        for j in range(0, n-i-1):
            if vec[j] > vec[j+1]:
                vec[j], vec[j+1] = vec[j+1], vec[j]

# Medir el tiempo de ejecución del ordenamiento de burbuja
tiempo_inicio = time.time()
ordenamiento_burbuja(vector)
tiempo_fin = time.time()

print("El vector ordenado:")
print(vector[:10])  # Imprimir solo los primeros 10 elementos
print("Tiempo de ejecución (segundos):", tiempo_fin - tiempo_inicio)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "e:\Proyectos - Estudios\SSL\TP2\OrdenamientoBurbuja.py"
El vector ordenado:
[2, 4, 4, 4, 6, 6, 8, 10, 10, 11]
Tiempo de ejecuci◆n (segundos): 3.1942994594573975

[Done] exited with code=0 in 3.258 seconds
```

6.2. Timsort.

6.2.1 Fortran.

timsort_Fortran.f90

```
1   module timsort_module
2     implicit none
3     integer, parameter :: RUN = 32
4     integer :: contador = 0
5
6   contains
7
8     subroutine insertion_sort(arr, left, right)
9       integer, intent(in) :: left, right
10      integer, intent(inout) :: arr(:)
11      integer :: i, j, key
12
13      do i = left + 1, right
14        key = arr(i)
15        j = i - 1
16
17        do while (j >= left .and. arr(j) > key)
18          contador = contador + 1
19          arr(j + 1) = arr(j)
20          j = j - 1
21        end do
22        arr(j + 1) = key
23      end do
24    end subroutine insertion_sort
25
26    subroutine merge(arr, temp, left, mid, right)
27      integer, intent(in) :: left, mid, right
28      integer, intent(inout) :: arr(:)
29      integer, intent(inout) :: temp(:)
30      integer :: i, j, k
31
32      i = left
33      j = mid + 1
34      k = left
35
36      do while (i <= mid .and. j <= right)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
[Running] cd "e:\Proyectos - Estudios\SSL\TP2\" && gfortran timsort_Fortran.f90 -o timsort_Fortran && "e:\Proyectos - Estudios\S
 Array después de ordenar:
           0           1           3           3           4           5           7           7           9           9
 Tiempo de ejecución (segundos):  0.109375000

[Done] exited with code=0 in 0.459 seconds
```

6.2.1 Python:

Sort.py > ...

You, 11 minutes ago | 1 author (You)

```python
import time
import random

# Generar un vector aleatorio con 10000 elementos
vector = [random.randint(1, 1000000) for _ in range(1000000)]

# Medir el tiempo de ejecución de la funcion sort

start_time = time.time()
vector.sort()
end_time = time.time()

print("El vector ordenado:")
print(vector[:10])  # Imprimir solo los primeros 10 elementos
print("Tiempo de ejecución (segundos):", end_time - start_time)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
[Running] python -u "e:\Proyectos - Estudios\SSL\TP2\Sort.py"
El vector ordenado:
[7, 9, 10, 10, 10, 10, 10, 10, 13, 14]
Tiempo de ejecuci□n (segundos): 0.23650884628295898

[Done] exited with code=0 in 0.744 seconds
```

7. Conclusión.

Python tiene una mayor facilidad de uso ya que proporciona funciones de ordenamiento listas para utilizar y muy eficientes incluidas en la biblioteca estándar, mientras que en Fortran es común tener que implementar el algoritmo manualmente o depender de bibliotecas externas.

Python es más sencillo de usar y tiene un código más limpio, mientras que Fortran generalmente puede ser más rápido para tareas computacionalmente intensivas debido a su optimización para cálculos numéricos.