Información útil

Este cuaderno complementa la información que aparece en los resúmenes en pdf que acompañan al examen

1. Complejidad

Repaso de series finitas

$$\sum_{i=inf}^{sup}(1) = sup - inf + 1$$

$$\sum_{i=0}^{n-1}(i) = \frac{n(n-1)}{2}$$

Siendo el índice i que va creciendo como una progresión aritmética de diferencia d y $f(i)$ es el coste en tiempo de ejecución de una iteración completa del bucle dependiendo del valor del índice i

$$\sum_{i=a}^{b} f(i) \approx \frac{1}{d}\int_{a}^{b} f(x)dx$$

Siendo el índice i que va creciendo como una progresión geométrica de razón r y $f(i)$ es el coste en tiempo de ejecución de una iteración completa del bucle dependiendo del valor del índice i

$$\sum_{i=a}^{b} f(i) \approx \frac{1}{\ln r}\int_{a}^{b} \frac{f(x)}{x}dx$$

Cota de complejidades

Función cota	Nombre	n= 1	n=100	n = 1000
1	constante	1	1	1
$\log n$	logarítmica	0	2	3
n	lineal	1	100	1000
$n \log n$	lineal-logarítmica	0	200	3000
n^2	cuadrática	1	10^4	10^6
n^3	cúbica	1	10^6	10^9
$\ldots$	$\ldots$			
n^k	polinómica	1	10^(2k)	10^(3k)
2^n	exponencial	2	31 dígitos	302 dígitos

Función cota	Nombre	n= 1	n=100	n = 1000
3^n	exponencial	3	48 dígitos	478 dígitos
...	...			
c^n	exponencial			
$n!$	factorial	1	161 dígitos	inimaginable

Teorema maestro (versión recursiva)

Un caso típico de algoritmo recursivo tiene esta ecuación en recurrencia:

$$T(n) = \begin{cases} cn^k & si \quad 1 \leq n \leq b \\ aT(n-b) + cn^k & si \quad n > b \end{cases}$$

Se puede demostrar que:

$$T(n) \in \begin{cases} \Theta(n^{k+1}) & si \quad a = 1 \\ \Theta(a^{n/b}) & si \quad a > 1 \end{cases}$$

Teorema maestro (versión divide y vencerás)

En algunos algoritmos recursivos las llamadas recursivas se producen dividiendo el tamaño del problema por un número entero, por lo que aplicaría la siguiente ecuación en recurrencia:

$$T(n) = \begin{cases} cn^k & si \quad 1 \leq n < b \\ aT(n/b) + cn^k & si \quad n \geq b \end{cases}$$

Se puede demostrar que:

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}) & si \quad k < \log_b a \\ \Theta(n^k \log_b n) & si \quad k = \log_b a \\ \Theta(n^k) & si \quad k > \log_b a \end{cases}$$

2. Programación Orientada a objetos

Herencia

Determinación de la clase padre e hija:

```
def ClaseHija(ClasePadre):
```

Inicialización del objeto

```
def __init__():
```

Inicialización de la parte heredada:

```
super().__init__()
```

Dunder methods

Common Syntax	Special Method Form
a[k]	a.__getitem__(k)
a[k] = v	a.__setitem__(k,v)
del a[k]	a.__delitem__(k)
a(arg1, arg2, ...)	a.__call__(arg1, arg2, ..
len(a)	a.__len__()
hash(a)	a.__hash__()
iter(a)	a.__iter__()
next(a)	a.__next__()
bool(a)	a.__bool__()
float(a)	a.__float__()
int(a)	a.__int__()
repr(a)	a.__repr__()
reversed(a)	a.__reversed__()
str(a)	a.__str__()

Sobrecarga

Common Syntax	Special Method Form
a + b	a.__add__(b);
a − b	a.__sub__(b);
a * b	a.__mul__(b);
a / b	a.__truediv__(b);
a // b	a.__floordiv__(b);
a % b	a.__mod__(b);
a ** b	a.__pow__(b);
a << b	a.__lshift__(b);
a >> b	a.__rshift__(b);
a & b	a.__and__(b);
a ^ b	a.__xor__(b);
a | b	a.__or__(b);
a += b a −= b a *= b ...	a.__iadd__(b) a.__isub__(b) a.__imul__(b) ...

Common Syntax	Special Method Form
+a	a.__pos__()
−a	a.__neg__()
~a	a.__invert__()
abs(a)	a.__abs__()
a < b	a.__lt__(b)
a <= b	a.__le__(b)
a > b a >= b	a.__gt__(b) a.__ge__(b)
a == b	a.__eq__(b)
a != b	a.__ne__(b)
v in a	a.__contains__(v)