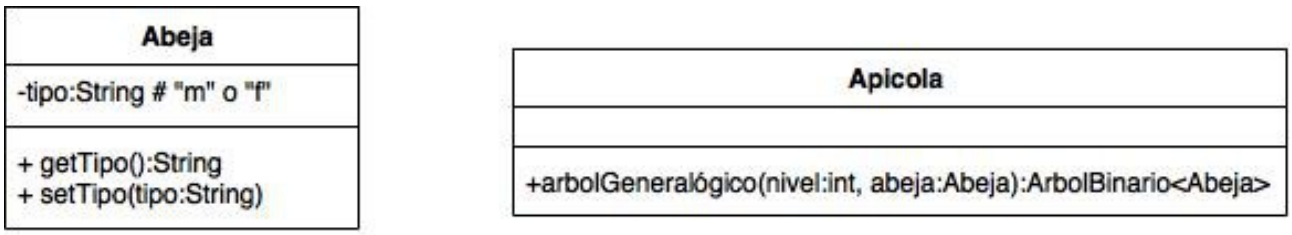
Resolución del ejercicio de árboles:

Willy es una abeja macho y está interesado en saber como está formado su árbol genealógico y cuantas hembras y machos hay en él.
Willy sabe que las féminas tienen 2 padres (un padre y una madre), pero los machos tienen unicamente madre (no padre).
Usted debe construir el árbol genealógico de Willy hasta el nivel “x” (el cual es pasado como parámetro).
Tenga en cuenta que la estructura del árbol genealógico puede deducirse a partir de cualquer abeja (por ejemplo: Willy es macho, por lo tanto, tiene sólo madre. La madre de Willy es hembra, por lo tanto, tiene padre y madre. El abuelo de Willy, es macho, por lo tanto tiene sólo madre... etc.)
Teniendo las siguientes clases, implemente en la clase Apicola el método ***arbolGenealogico(int nivel,Abeja abeja):ArbolBinario<Abeja>*** que recibe como parámetros el nivel, del árbol genealógico a crear según lo enunciado anteriormente, y la abeja, de la cual hay que crear el árbol.
Asuma que el resto de los métodos se disponen.




```
class Apicola:
  def arbolGeneralogico(self,nivel,abeja):
    if nivel >= 0:
       arbol = ArbolBinario(abeja)
       arbol.agregarHijoIzquierdo(self.arbolGeneralogico(nivel-1,Abeja("f")))
       if arbol.getDato().getTipo() == "f":
          arbol.agregarHijoDerecho(self.arbolGeneralogico(nivel-1,Abeja("m")))
      return arbol
```

Resolución ejercicio de grafos:

Implemente en la clase Grafo el método ***dfsCaminoMaximo(v1:Vertice, v2:Vertice):List<Vertice>*** que devuelve el camino más largo entre v1 y v2 con un recorrido dfs.

```
def dfsCaminoMaximo(self,v1,v2):
    visitados = []
    for i in self.listaDeVertices():
        visitados.append(False)
    path_temp = []
    path_max = []
    self.dfs(v1,visitados,v2,path_max,path_temp)
    return path_max
```

```
def dfs(self,vertice,visitados,destino,camino_maximo,camino_temporal):
    camino_temporal.append(vertice)
    visitados[vertice.getPosicion()] = True
    if vertice.getPosicion() == destino.getPosicion():
        if len(camino_temporal)>len(camino_maximo):
            for i in range(len(camino_maximo)):
                del camino_maximo[0]
            for v in camino_temporal:
                camino_maximo.append(v)
        else:
            for arista in vertice.obtenerAdyacentes():
                if not visitados[arista.verticeDestino().getPosicion()]:
                    self.dfs(arista.verticeDestino(),visitados,destino,camino_maximo,camino_temporal)
    del camino_temporal[len(camino_temporal)-1]
    visitados[vertice.getPosicion()] = False
```

Resolución de ejercicio de Tiempo de ejecución:

```
def main(cantidad):
        for i in range(cantidad):
                for j in range(i):
                        algo_de_O(1)
        if cantidad > 1:
                for i in range(4):
                        main(cantidad/2)
```

$$T(n) = \begin{cases} \text{cte1, } n<=1 \\ 4*T\left(\frac{n}{2}\right) + \sum_{i=1}^{n}\sum_{j=1}^{i} cte2 \quad \text{, } n>1 \end{cases}$$

1º resuelvo la sumatoria

$$\sum_{i=1}^{n}\sum_{j=1}^{i} cte2 = cte2 * \sum_{i=1}^{n} i = \text{cte2 } *\left(\frac{n*(n+1)}{2}\right) = \frac{cte2}{2}*n^2+\frac{cte2}{2}*n$$

ahora reemplazo:

$$T(n) = \begin{cases} \text{cte1, } n<=1 \\ 4*T\left(\frac{n}{2}\right) + \frac{cte2}{2}*n^2+\frac{cte2}{2}*n \end{cases}$$

paso 1: $4*T(\frac{n}{2}) + \frac{cte2}{2}*n^2+\frac{cte2}{2}*n$

paso2 : $4*\left[4*T(\frac{n}{2^2}) + \frac{cte2}{2}*(\frac{n}{2})^2+\frac{cte2}{2}*(\frac{n}{2}) \right] + \frac{cte2}{2}*n^2+\frac{cte2}{2}*n$

$4^2*T(\frac{n}{2^2}) + \frac{cte2}{2}*n^2+\frac{cte2}{2}*2*n + \frac{cte2}{2}*n^2+\frac{cte2}{2}*n$

$4^2*T(\frac{n}{2^2}) + \frac{cte2}{2}*2*n^2+\frac{cte2}{2}*3*n$

paso 3: $4^2 * \left[4*T(\frac{n}{2^3}) + \frac{cte2}{2}*(\frac{n}{2^2})^2+\frac{cte2}{2}*(\frac{n}{2^2}) \right] + \frac{cte2}{2}*2*n^2+\frac{cte2}{2}*3*n$

$4^3 * T(\frac{n}{2^3}) + \frac{cte2}{2}*3*n^2+\frac{cte2}{2}*7*n$

paso k : $4^k*T(\frac{n}{2^k})+\frac{cte2}{2}*k*n^2+\frac{cte2}{2}*(2^k-1)*n$

aplico la igualdad al caso base para obtener el valor de k

$\frac{n}{2^k}=1 \rightarrow k=\log_2 n$

$4^{(\log_2 n)}*T(\frac{n}{2^{(\log_2 n)}})+\frac{cte2}{2}*\log_2 n*n^2+\frac{cte2}{2}*(2^{(\log_2 n)}-1)*n$

$T(n) = n^2 * T(1) + \frac{cte2}{2}*\log_2 n*n^2+\frac{cte2}{2}*(n-1)*n$

$T(n) = n^2 + \frac{cte2}{2}*\log_2 n*n^2+\frac{cte2}{2}*n^2-\frac{cte2}{2}*n$

T(n) = O($\log_2 n*n^2$ **)**