UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAD DE INFORMÁTICA DE BARCELONA

Asignatura: Programación i algoritmos 2

Trabajo: Practica final

Participantes: Enzo Biasizzo Serra, Albert Campos Gisbert, Daniel López García

Fecha: 16/06/2023

Índice

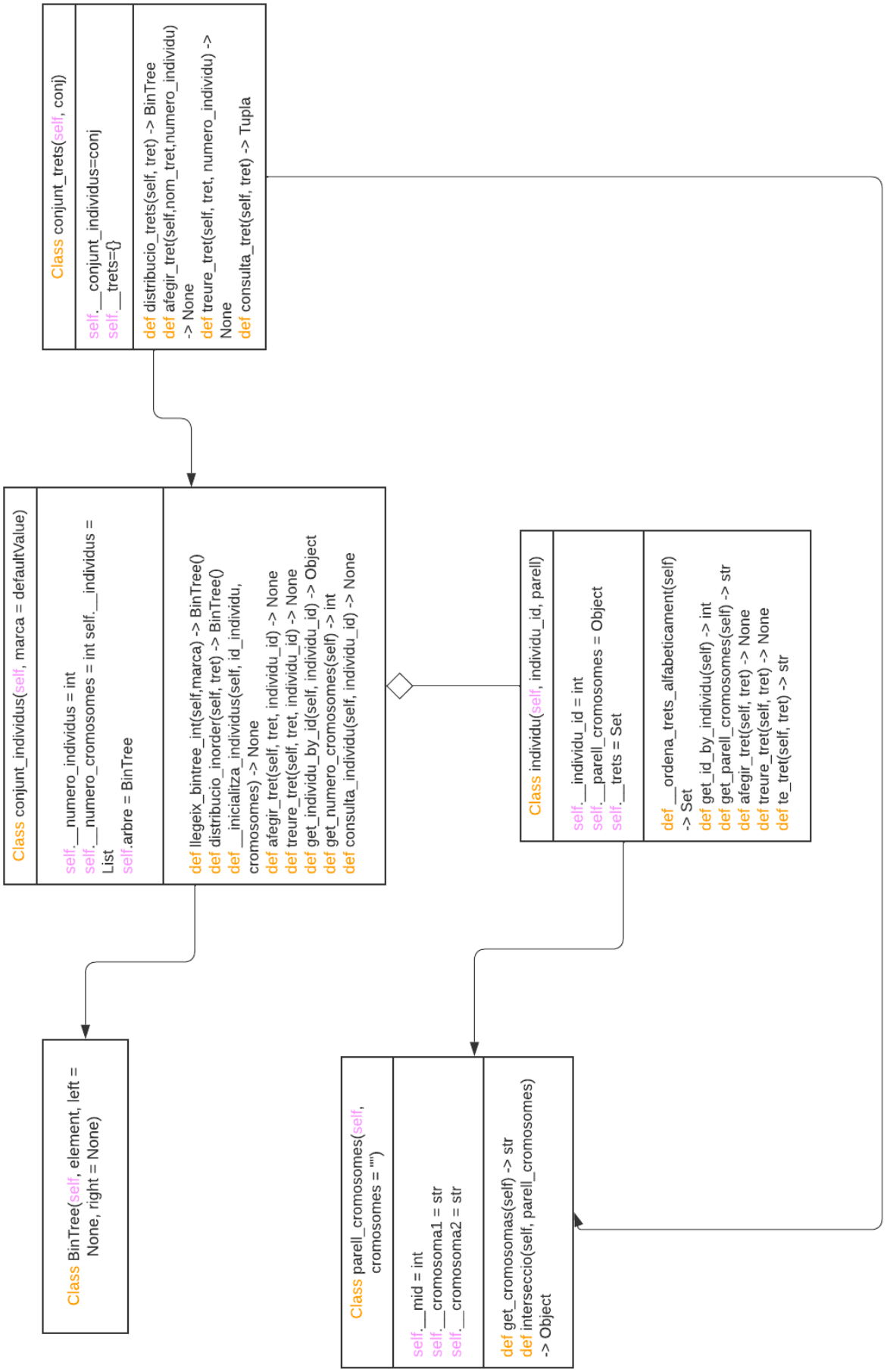
Abstract

Este programa en Python, desarrollado para la práctica final de *programación y algoritmia 2*, tiene como objetivo principal consolidar y aplicar los conceptos aprendidos durante el curso, especialmente en relación a la orientación a objetos y estructuras de datos dinámicas.

La práctica en esencia consiste en implementar las funciones de gestión de datos de un laboratorio que realiza experimentos de manipulación genética.

En este documento se expondrá el diseño de clases del programa, el tratamiento de errores de las diferentes casuísticas posibles, y la argumentación de las diversas estructuras de datos empleadas durante el proyecto.

1. Diseño UML del programa



2. Nuestras clases, métodos públicos, privados y estructuras de datos empleadas

- ***Class conjunt_individus:***

  Esta clase tiene como cometido inicializar y organizar el conjunto de individuos involucrados en el experimento, para ello utiliza un BinTree (self.__arbre), en el que define la relación entre los individuos (si tienen progenitores o no, y cuales), además de una lista(self.__individus) que contiene las instancias “individu”. También tiene los atributos privados recibidos de la comanda “experiment” en el fichero entrada.inp.

  En el constructor se inicializa el arbol mediante el metodo “llegeix_arbre_int” y los individuos mediante “inicialitza_individus”.

  Ha resultado muy práctico usar BinTree como estructura en esta clase, porque es la forma más óptima de definir la relación de sus individuos con sus progenitores, y de poder cambiar la distribución del mismo para trabajar con rasgos y reasignarlos cuando el experimento lo requiere. De la misma forma, el tener los individuos instanciados en una lista, permite poder indexarlos por orden posicional, y acceder a cada uno de ellos mediante un identificador “individu_id”, también, si en el futuro se amplía la funcionalidad del programa, se podrán eliminar si se quiere prescindir de un individuo concreto.

  Excepto “__inicialitzacio_individus” el resto de métodos de la clase son públicos, *getters* que darán información de las instancias “individu”, ya que “conjunt_individus” actuará como clase puente entre “conjunt_trets” e “individu”. El unico metodo publico que se diferencia de los demás és “consulta_individu”, pues será el que muestre la información concreta de un individuo, cuando el main lo requiera.

- ***Class individu:***

  La funcionalidad de esta clase es sencilla, su objetivo es atomizar la información de los sujetos del experimento, para una mejor y más optimizada arquitectura del programa.

Cada individuo, al instanciarse cuenta con un identificador, su par de cromosomas (una instancia de la clase “parell_cromosomes”), y los rasgos asociados a él. Cuenta con dos métodos getters para que “conjunt_individus” pueda conseguir la información que precise, un método que verifica si el individuo tiene un rasgo concreto, y los métodos para añadir y quitar rasgos al individuo. El único método privado que tiene esta clase es “ordena_alfabeticament”, ya que está pensado para ser usado de forma interna por la clase.

Los rasgos del individuo están almacenados en un conjunto(set) porque no es relevante el orden, por lo que al acceder a un rasgo del conjunto, se hará mediante el propio nombre de este.

- **Class conjunt_trets:**

Esta clase tiene clase se encarga de crear un diccionario que guarda todas las características, los “trets”, que se nos dan en el fichero de entrada y una lista de los individuos que las poseen además de la intersección de sus respectivos pares de cromosomas.

Para poder conseguir esto decidimos finalmente que lo más efectivo y simple era usar un diccionario de python donde el nombre de las características, los “trets”, serían usados de claves y como valores almacenaríamos tuplas, específicamente “*namedtuple”* de la librería collections para facilitar su lectura. Estas tuplas contendrán una instancia de la clase parell_cromosomes, la intersección de los pares de cromosomas de los individuos que poseen dicha característica y una lista de python, en vez de un set debido a la gran versatilidad de estas, ocupará el segundo lugar de la tupla guardando aquí individuos que almacenaremos en forma de referencias a las instancias de la clase individuos inicializadas y almacenadas anteriormente en la clase conjunt_individus y que obtenemos a través de de un getter, de dicha clase, llamado get_individu_by_id que nos devuelve la instancia de la clase individuo del individuo indicado usando como argumento el número de individuo, número con que identificamos a estos.
Todo este montaje nos permite guardar parejas clave valor donde la clave es el nombre de la característica en cuestión y el valor es una “*namedtuple”* que contiene la intersección de los individuos que poseen la característica y la lista de individuos que la poseen. Para conseguir esto disponemos de distintos métodos:

- El método __init__ : inicializa el diccionario de python donde guardaremos la información y guarda la instancia de la clase conjunt_individus que nos permitirá acceder a la información y

métodos que necesitamos de esa clase cuya instancia se nos da de argumento al instanciar esta clase.

- El método afegir_trets: este método nos permite añadir a nuestro diccionario una pareja clave valor como la anteriormente indicada. Para obtener la intersección de los pares de cromosomas de los distintos individuos se llama al método interseccio de la clase parells_cromosomes cada vez que se añade un individuo a la característica. Si es el primer elemento en poseer la característica se guardará como intersección su pareja de cromosomas. Además,desde esta método, también se llama a su homólogo de la clase conjunt_individus, de nombre también afegir_trets, que se encarga de añadir en el lugar correspondiente de esa clase que un determinado individuo posee cierta característica.
  Finalmente, este método también envía un mensaje de error si la pareja clave valor que se pretende añadir ya ha sido añadida anteriormente o si el individuo indicado no está entre los establecidos al inicio.

- El método treure_trets: este método nos permite remover un individuo de la lista de individuos que poseen la característica indicada, es decir, eliminamos dicho individuo de la lista de individuos que tenemos como segundo elemento de la tupla guardada como valor en nuestro diccionario. No solo esto, si la lista de individuos está ahora vacía, se elimina toda la pareja clave valor, es decir, se elimina la característica del diccionario. Si la lista de individuos no está vacía, en cambio, se vuelve a calcular la intersección de los individuos restantes que la poseen. De manera similar al caso anterior, se usa el mismo método interseccio de la clase parell_cromosomes y se llama a un método homólogo de la clase conjunt_individus, llamado también treure_trets, que quita la característica indicada de la lista de características de ese individuo. Finalmente, la clase envía error si la característica indicada no estaba todavía en el diccionario o si el individuo indicado no se encontraba ya en la lista de individuos que poseían dicha característica (haciendo innecesario indicar expresamente que si el individuo no es uno de los iniciales de error, porque como no estará en ninguna de las listas nunca siempre dará error).

- El método consulta_trets: este método retorna una tupla con la intersección y la lista de individuos que poseen la característica dada como argumento. La intersección la da como tupla de strings donde el elemento uno es el primer cromosoma del par y el elemento dos el segundo string. La lista de individuos es la lista de instancias de la clase individuos guardada en el diccionario. Esta información la recibe

el main que se encarga de extraer los números, que representan los individuos, de las instancias de la clase individuo de la lista y de mostrar todo por pantalla con el formato indicado en la práctica

- distribució_trets: este método imprime los elementos de aquellos subárboles que contengan un “tret”, en in-order. Todo esto en función del árbol principal. Este método funciona con una función auxiliar la cual hace lo siguiente: verifica si el árbol es vacío o no, en caso afirmativo retornaría None; verifica si la raíz contiene el tret o no, en caso afirmativo la incluye en el BinTree; verifica en los árboles izquierdo y derecho, asignando valores negativos en caso de que los nodos comprendidos entre varios nodos positivos. Una vez retornado el árbol por la función auxiliar se verifica si está vacío o no. Si, en efecto, lo está se escribirá “error”; si no lo está se escribirá el return de la función auxiliar como una string in-order.

- **Class parell_cromosomes:**

Esta clase procura partir en dos los inputs recibidos en dos strings separadas con tal de que puedan ser legibles. Y para que puedan ser posteriormente asignados a sus valores. Consta de los siguientes métodos:

- Método __str__: imprime los dos cromosomas separados por un salto de línea.
- Método get_cromosomas: retorna las strings de ambos cromosomas
- Método interseccio: Este método recibe como argumento una instancia distinta de parell_cromosomes. A continuación guarda en dos variables el primer y segundo cromosoma de la instancia sobre la que aplicamos el método, nuestro self, y el primer y segundo cromosoma de la instancia dada como argumento en otras dos variables y todo transformado a listas(pues originalmente son strings y se realiza dicha transformación para simplificar el proceso de encontrar la intersección).

  Posteriormente, mediante un bucle y diversas condiciones, se guarda el elemento en una nueva lista si forma parte de la intersección y en caso contrario se coloca un guión. Esto se hace comparando el primer cromosoma del par de cromosomas de la primera instancia con el primer cromosoma de la segunda instancia y lo mismo con los segundos pares de cromosomas de la segunda instancia por lo que obtenemos las dos listas que formarán el par de cromosomas de la

intersección, sin embargo, nos fijamos que no siempre coinciden los guiones de las dos listas que representan el nuevamente creado par de cromosomas.

Para solucionar esto, en el mismo bucle, introdujimos una condición que se encarga de ver si en en el primer o segundo cromosoma de la intersección se había puesto un guión y, si ese es el caso, dicho guión se coloca en la posición correspondiente del otro elemento del par. Esto se hace al mismo tiempo que se encuentra la intersección de los dos pares, justo después, lo que nos permite usar un único bucle para hacer todas las operaciones necesarias. Finalmente, a pesar de estar usando listas de python por la facilidad con las que nos permitía realizar estas operaciones, vamos introduciendo estos guiones o números en dos variables string a medida que encontramos la intersección obteniendo, al final, la intersección como strings. A continuación creamos una instancia de parell_cromosomas donde guardamos los strings de la recién encontrada intersección y eso retornamos.

- **main:**

En lo que respecta al *main,* se ha procurado que este ayude a hacer que el código de las clases sea más optimizado, y que sea el main el que lea los elementos del fichero de entrada para pasarlos posteriormente por parámetro a los métodos públicos de las clases relacionadas. Para ello se ha precisado de la librería "*easyinput*", la cual permite de una forma muy sencilla acceder elemento por elemento la entrada recibida por parámetro al ejecutar el programa.

3. Tratamiento de errores y principales problemas en el desarrollo

Algunos de los problemas encontrados durante el desarrollo de la clase conjunt_trets y la creación del método interseccio:
Al principio decidimos implementar el diccionario nosotros mismos creándolo con la implementación de tablas de hash debido a que considerábamos que si lo implementabamos nosotros mismos podríamos personalizarlo a nuestros deseos y requerimientos. Pronto esta demostró ser una tarea que en el tiempo del que disponíamos no podríamos implementar bien, además de provocar la creación de un código cada vez más denso, complejo e ininteligible.

En consecuencia, decidimos rehacer la clase usando diccionarios de python y pronto nuestro código estaba funcionando y además de haberse simplificado considerablemente haciéndolo más inteligible. Por añadidura, cabe remarcar que al principio del proyecto enfocamos mal ciertos elementos de la práctica. Es por esta razón que el método interseccio empezó su vida como un método de la clase conjunt_trets y acabará siendo un método de la clase parell_cromosomes mientras con el método distribucio_trets ocurrió precisamente lo contrario: empezó como un método de parell_cromosomes y acabó como un método de conjunt_trets. En el caso de interseccio esto provocó ciertos quebraderos de cabeza que, afortunadamente, fueron superados.

El principal quebradero de cabeza ocasionado por el desplazamiento del método de una clase a otra fué que como método de conjunt_trets este requería de dos argumentos: las dos parejas de cromosomas de las que deseábamos encontrar la intersección. En cambio, como método de parell_cromosomes uno de los pares de cromosomas ya la teníamos como la instancia de la clase sobre la que se aplicaba el método mientras que la segunda era el que ahora sería el único argumento del método, en otras palabras, lo que antes era el primer argumento era ahora el self del método y lo que antes era el segundo argumento era ahora el único argumento en cuestión.

Otro desafío al momento de desarrollar este trabajo fue la implementación del método distribució_trets. En primer lugar por un enfoque inicial incorrecto de cómo resolver los requerimientos principales de este método y en segundo lugar la implementación de classes tipo BinTree. Esto en consecuencia de la falta de un getter para obtener el árbol extraído de la clase conjunt_individus. La cual acabaría siendo el método get_arbre.

4. Conclusión

Al acabar esta práctica hemos obtenido una resolución satisfactoria de los diferentes objetivos que nos propusimos a lo largo del desarrollo de esta.

Tras una correcta implementación de la arquitectura de programación correspondiente y su consecuente revisión, corrección y puesta en común mediante ensayo y error. Podemos afirmar un desarrollo en el entendimiento tanto como en la utilización de clases y métodos de python aplicados a una situación realista.