

PRÀCTICA PA2

LABORATORI DE BIOLOGIA

Assignatura Programació i algorismia 2

Curs 2022-2023

Alumnes Bennàssar Martín, Joan

Macià Codera, Nil

Professors Álvarez Napagao, Sergio

Balcázar Navarro, José Luis

Delgado Pin, Jordi

ÍNDEX

INTRODUCCIÓ	2
ENUNCIAT	3
PLANTEJAMENT I DISSENY	6
EXPLICACIÓ I JUSTIFICACIONS DELS ARXIUS	8
Interfície	8
Conjunt d'individus	9
Individu	11
Conjunt de trets	12
Parell de Cromosomes	12
PROVES FETES AMB L'APLICACIÓ	14
CONCLUSIONS	20

INTRODUCCIÓ

Aquest informe pertany a la pràctica de l'assignatura de Programació i Algorismia 2 i es realitza en grups de dos components, en aquest cas, en Nil Macià Codera i en Joan Bennàssar Martín. L'objectiu de la pràctica és aplicar els conceptes apresos a PA2, principalment orientació a objectes i l'abstracció que proporcionen aquests, i la implementació d'estructures de dades i algorismes sobre aquestes.

L'entrega conté, a més a més de l'informe en qüestió, una carpeta amb els fitxers Python que hem programat i són una possible solució al problema que se'ns ha plantejat. Cadascun d'aquests es correspon a una classe del programa. També s'inclou en l'entrega els jocs de prova creats per tastar l'aplicació amb les seves corresponents sortides.

La pràctica consisteix en la creació d'una aplicació per un laboratori de biologia per tal d'estudiar les relacions entre gens (parells de cromosomes) i trets d'individus a l'hora de realitzar experiments amb individus que tenen relacions de parentiu entre ells. Amb aquesta aplicació podrem conèixer com els trets es distribueixen en la família, així com l'estructura dels parells de cromosomes que fan que aquest tret es manifesti en l'individu. Coneixerem més detalls de l'enunciat en el següent apartat de l'informe.

Al llarg de l'informe explicarem com hem plantejat i dissenyat l'aplicació per donar resposta a l'enunciat en qüestió. Explicarem el funcionament dels diferents arxius python, justificant sempre les estructures de dades utilitzades, així com les corresponents implementacions, ja que hi ha estructures de dades de les quals coneixem diverses implementacions, que hem anat veient al llarg del curs). Finalment presentarem els jocs de prova creats per testar l'aplicació i les sortides d'aquests.

ENUNCIAT

En un laboratori de biologia estan fent experiments per intentar relacionar gens i trets ("rasgos" en castellà) d'individus pertanyents a espècies amb genomes relativament fàcils de manipular amb les eines d'edició genètica de les que disposa el laboratori.

Per simplificar els experiments, en comptes de tractar amb tot el genoma de cada individu, tractarem només amb un parell de cromosomes de cada individu. Tots els parells de cromosomes seran anàlegs, és a dir, ocuparan la mateixa posició (que no ens importa) dintre del genoma.

Tots els cromosomes d'un experiment correspondran a individus de la mateixa espècie, tindran la mateixa longitud m i constaran de m valors (zeros i uns) que ens indicaran si el gen corresponent és dominant o recessiu. Per a cada experiment, el laboratori ens proporcionarà el nombre d'individus de l'experiment $n \geq 3$, i el nombre de gens de cada cromosoma $m \geq 1$. Els individus estaran identificats pels valors $1, 2, \dots, n$. També ens indicaran les relacions de parentiu entre els diferents individus. Tots els individus tindran dos progenitors o cap. Els que no tenen progenitors seran els individus inicials, a partir dels quals es generaran la resta de d'individus de l'experiment. **Tots els individus tindran un únic fill, excepte un sol individu que no en tindrà cap.** Aquesta informació de parentiu es pot veure com l'arbre genealògic de l'únic individu sense fills.

Un cop que coneixem les relacions de parentiu entre els individus, ens proporcionaran la composició del parell de cromosomes que hem d'estudiar de cadascu d'ells. A continuació, s'aniran introduint els trets que es vagin observant en cada individu a mesura que el laboratori ens vagi indicant els resultats de les seves proves per a cada individu.

Un tret serà una paraula que necessàriament començarà amb una lletra i que podrà constar de lletres, nombres i el caràcter '_'. En tot moment s'ha de saber quins trets té cada individu i per cada tret s'ha de saber quins individus el presenten i quina és la combinació de gens que suposadament el produeix. Aquesta combinació de gens és la intersecció del parells de cromosomes de tots els individus que presenten aquest tret. Quan s'afegeix el tret a un individu és possible que aquesta combinació sigui més restringida.

Per entendre com va aquesta intersecció el més fàcil és un exemple. Estem tractant amb parells de cromosomes que tenen longitud 10. Suposem que només l'individu 5 i 8 tenen el tret *asdfg_23* i que el seus parells de cromosomes són pel que fa a l'individu 5:

1 0 1 0 1 0 1 0 1 0

1 0 1 0 1 0 1 0 1 1

i pel que fa a l'individu 8:

1 1 1 0 1 0 0 0 1 0

1 0 0 1 1 0 0 1 0 1

La intersecció dels dos parells de cromosomes queda així:

1 - - - 1 0 - - - 0

1 - - - 1 0 - - - 1

Això significa que pel moment, suposem que per tal que es manifesti el tret *asdfg_23* cal que el primer parell de gens sigui el 1 i 1; el cinquè també 1 i 1; el sisè 0 i 0 i el desè i darrer 0 i 1. La resta de parells de gens aparentment no intervenen en la manifestació d'aquest tret concret. Si més endavant es descobreix que algun individu té aquest tret és possible que aquesta intersecció sigui més restringida.

Quan s'afegeix per primer cop el tret, la combinació que el manifesta és el parell de cromosomes de l'individu a qui se li ha afegit.

També s'ha de poder consultar les dades d'un individu a partir del seu identificador (un enter entre 1 i n) i les dades d'un tret a partir del seu identificador (que és el nom del tret).

Per últim, el laboratori ens demana poder estudiar la distribució d'un tret dins de l'arbre

genealògic. Donat l'arbre genealògic i el nom d'un tret, volem el subarbre genealògic on apareguin tots els camins des de l'individu sense descendents fins a cada individu que manifesti aquest tret. Els nodes d'aquest subarbre que manifestin el tret hauran d'estar etiquetats amb l'identificador de l'individu i, els que no el manifestin, amb l'identificador de l'individu canviat de signe.

Les operacions del nostre programa són les següents:

- **experiment:** S'introduiran dos valors enters $n > 0$ i $m > 0$ que correspondran al nombre d'individus i al nombre de gens de cada cromosoma de cada parell de cada individu. Després s'introduirà en preordre l'arbre genealògic d'n nodes de l'únic individu que no té descendents. L'identificador d'aquest individu pot ser qualsevol valor entre 1 i n. Cada individu apareixerà un cop i només un cop. Per últim, s'introduiran les dades del parell de cromosomes de cada individu per ordre d'identificador d'individu.
- **afegir_tret:** S'introduirà el nom d'un tret t i un enter $1 \leq id \leq n$. Si l'individu id ja té aquest tret s'escriurà error. Si no, quedarà constància de que l'individu id té el tret t i es calcularà la combinació de parells de cromosomes que fan que es manifesti el tret t.
- **consulta_tret:** S'introduirà el nom d'un possible tret. Si aquest no existeix s'escriurà error. Si existeix s'escriurà la combinació que suposadament fa que es manifesti i els identificadors dels individus que la manifestin per ordre creixent d'identificador.
- **consulta_individu:** S'introduirà un enter $1 \leq id \leq n$ i s'escriurà la informació de l'individu amb identificador id, és a dir, el seu parell de cromosomes i el nom dels seus trets per ordre alfabètic.
- **distribucio_tret:** S'introduirà el nom d'un possible tret. Si aquest no existeix s'escriurà error. Si existeix s'escriurà el subarbre resultat en inordre.
- **fi:** S'acaba l'execució del programa.

Observeu que en una execució del programa es poden realitzar molts experiments. A cada execució del programa es farà com a mínim un experiment. Per facilitar la llegibilitat s'escriuran totes les instruccions a la primera columna i tots els seus paràmetres separats per un espai. Qualsevol resultat s'escriurà a partir de la columna 3.

La nostra aplicació ha de respondre al el joc de proves públic, l'entrada del qual és la següent: [clik aquí](#).

La sortida que el nostre programa ha de retornar pel joc de proves públic anterior ha de ser igual que la següent: [click aquí](#).

DISSENY DEL PROGRAMA

Per al conjunt de programes que constitueixen la solució al problema plantejat hem utilitzat la implementació de classes pròpies, les estructures list i set de python i un arbre binari limitat al nostre problema. La solució consta d'una interfície i les quatre classes principals (hi ha una cinquena que forma part de la implementació interna d'una d'aquestes):

- **Interfície:** Llegeix les instruccions mencionades anteriorment: 'experiment', 'afegir_tret', 'consulta_tret', 'consulta_individu', 'distribucio_tret' i 'fi'. Tret de la instrucció fi, que termina el programa, el que fa aquest és llegir els arguments de cada instrucció per poder cridar les classes els mètodes de les classes Conjunt d'Individus i Conjunt de Trets.
- **Conjunt d'Individus:** L'estat (privat) de les instàncies d'aquesta classe (una per experiment) consistirà en una col·lecció amb les instàncies d'**Individu** i l'arbre que defineix la relació entre individus. Aquesta classe contindrà les següents operacions (mètodes públics):
 - Afegir: afegirà un tret a un individu de la col·lecció si aquest no en té (si ja el té escriurà 'error')
 - Consulta: escriu la informació de l'individu de la col·lecció en qüestió (el seu parell de cromosomes i el nom dels seus trets per ordre alfabètic)
 - Distribució: retorna el subarbre des del node sense fills cap a tots els nodes amb el tret, posant en negatiu els nodes que no el tenen però que pertanyen a l'arbre ja que formen part del camí per arribar a un que sí que el té. Aquest subarbre s'escriu en inordre.

La classe també conté una subclasse **Arbre** (privada), que consisteix en un arbre binari construït a partir d'una llista en preordre (amb les fulles indicades amb dos zeros), i amb el mètode (públic) distribució, que troba el subarbre que format pel recorregut en inordre per arribar a tots els nodes amb un tret des del node sense descendència (l'arrel de l'arbre).

- **Conjunt de Trets:** L'estat (privat) de les instàncies d'aquesta classe (una per experiment) consistirà en un diccionari al que s'accedeix amb el noms dels trets i els elements el parell de cromosomes i el conjunt d'individus (el número) amb el tret. Aquesta classe conté les següents operacions (els mètodes públics):
 - Afegir: afegeix un individu al tret en cas de que ja existeixi, en cas contrari, es crea el tret corresponent amb l'individu en qüestió. També es calcula la intersecció dels parells de cromosomes dels individus, que seran els possibles gens que donen lloc al tret.
 - Consulta: escriu la informació del tret en qüestió, si aquest existeix (la intersecció de cromosomes que el manifesten i els individus que el presenten de forma ordenada segons el seu identificador).

- **Individu:** L'estat (privat) de les instàncies d'aquesta classe consistirà en una instància de Parell de Cromosomes i un conjunt de noms de tret, aquells que l'individu mostra. Les operacions (mètodes públics) que la classe posseeix són els següents:
 - Afegir: s'afegeix el tret a l'individu, i retorna el parell de cromosomes per poder afegir-lo als del tret.
 - Te: retorna un valor booleà en funció de si l'individu posseeix o no el tret en qüestió.
 - Str (privada): la forma de imprimir les instàncies de la classe ve donada per aquesta funció, que determina que s'ha d'imprimir el parell de cromosomes com un string en primer lloc, seguidament de tots els trets que posseeix l'individu ordenats alfabèticament.
- **Parell de Cromosomes:** L'estat (privat) de les instàncies d'aquesta classe consistirà en dos contenidors amb els bits de cadascun dels dos cromosomes de m. La classe posseeix també les següents operacions (mètodes públics):
 - Intersecció: es comparen un a un els bits de cada cromosoma de un individu amb els bits de cada cromosoma d'un altre i es canvien aquells que no coincideixen per un guió, trobant així la intersecció entre els cromosomes dels dos individus.
 - Str (privada): la forma de imprimir les instàncies de la classe ve donada per aquesta funció, que determina que s'ha d'imprimir el primer cromosoma en una línia i el segon en la següent.

EXPLICACIÓ I JUSTIFICACIONS

A continuació expliquem i justifiquem la implementació de la nostra solució:

Interfície

```
from pytokr import make_tokr
from conj_individus import ConjuntIndividus
from conj_trets import ConjuntTrets
```

Hem decidit utilitzar la funció 'make_tokr' de 'pytokr' perquè simplificava la tasca de llegir tots els arguments pertinents quan es trobaven en diferents línies, ja que només cal cridar 'f_item':

```
with open(0) as file:
    _, f_item = make_tokr(file)
    while True:
        token = f_item()
```

Per iniciar un experiment cal utilitzar la comanda 'experiment' seguida del nombre d'individus i la llargada dels parells de cromosomes. Després s'ha d'introduir l'arbre genealògic dels individus en preordre amb un parell de 0s indicant les fulles, i els parells de cromosomes com una sola paraula, primer els bits del primer cromosoma i després els del segon. Aleshores el preordre es passa a la inicialització de la classe Conjunt d'Individus com a una llista d'enters, i els parells com a una llista d'strings. També s'ha de crear una instància de Conjunt de Trets.

Pel preordre cal agafar $n*2+1$ elements perquè és el nombre d'elements del preordre contant els 0s, ja que el nombre de fulles sempre és la meitat de nodes de l'arbre arrodonint a la alça.

```
if token == 'experiment':
    n = int(f_item())
    m = int(f_item())
    print(token, n, m)
    preordre = [int(f_item()) for _ in range(n*2+1)]
    parells = [f_item() for _ in range(n)]
    conj_individus = ConjuntIndividus(preordre, parells)
    conj_trets = ConjuntTrets()
```

Per afegir un tret a un individu la comanda és 'afegir_tret' i ha d'anar seguida del nom del tret i l'identificador de l'individu. Aquest nom i l'enter corresponent a l'identificador es passen com a arguments a la crida al mètode 'afegir' de Conjunt d'Individus, el valor retornat del qual es guarda a 'parell' per passar-lo al mètode 'afegir' de Conjunt de Trets per poder fer la intersecció. També es passen el nom del tret i l'identificador de l'individu:

```
elif token == 'afegir_tret':  
    tret = f_item()  
    i = int(f_item())  
    print(token, tret, i)  
    parell = conj_individus.afegir(tret, i)  
    conj_trets.afegir(tret, i, parell)
```

Per consultar un tret la comanda és 'consulta_tret' seguida del nom del tret, el qual es passa al mètode consulta de Conjunt d'Individus:

```
elif token == 'consulta_tret':  
    tret = f_item()  
    print(token, tret)  
    conj_trets.consulta(tret)
```

Per consultar un individu la comanda és 'consulta_individu' seguit de l'identificador, el qual es passa com a enter a consulta de Conjunt d'Individus:

```
elif token == 'consulta_individu':  
    i = int(f_item())  
    print(token, i)  
    conj_individus.consulta(i)
```

Per veure la distribució del tret dins l'arbre genealògic en inordre la comanda és 'distribucio_tret' seguida del nom del tret que es passa a distribucio de Conjunt d'Individus:

```
elif token == 'distribucio_tret':  
    tret = f_item()  
    print(token, tret)  
    conj_individus.distribucio(tret)
```

I per acabar el programa només cal escriure la comanda 'fi':

```
elif token == 'fi':  
    print(token)  
    quit()
```

Conjunt d'individus

Per aquesta classe cal incloure la classe Individu, ja que són els objectes que manipula per a fer les operacions:

```
from individu import Individu
```

Els individus es creen dins una llista __individus el primer element de la qual és None per poder accedir amb els identificadors dels individus directament (el primer individu tindrà índex 1 enlloc de 0), i necessiten la string del seu parell de cromosomes. Per a representar la relació entre els individus s'utilitza una subclasse privada __Arbre per facilitar la creació de l'estructura:

```
class ConjuntIndividus:
    def __init__(self, preordre, parells):
        self.__individus = [None] + [Individu(s) for s in parells]
        self.__arbre = self.__Arbre(preordre)
```

Per afegir un tret primer es comprova si l'individu ja té aquell tret associat per poder avisar de l'error, igualment es retorna el valor de afegir el tret a l'individu ja que això no afecta l'estat de l'individu i d'aquesta manera no es generen errors en altres programes:

```
def afegir(self, tret, i):
    if self.__individus[i].te(tret):
        print(' error')
    return self.__individus[i].afegir(tret)
```

Per consultar un individu es comprova que l'identificador sigui correcte per notificar de l'error en cas contrari, i si és correcte només cal escriure l'individu ja que aquest ja defineix com és la seva string:

```
def consulta(self, i):
    if i < len(self.__individus):
        print(self.__individus[i])
    else:
        print(' error')
```

Per veure la distribució d'un tret es crida el mètode distribució de l'arbre i aquest retorna la llista amb l'inordre corresponent, després s'encarrega de transformar aquesta llista en l'string pertinent, o escriure error si és buida:

```
def distribucio(self, tret):
    d = self.__arbre.distribucio(tret, self.__individus)
    if d:
        print(' ', *d)
    else:
        print(' error')
```

Arbre

Al inicialitzar l'arbre s'assigna el primer valor de la llista (que és esborrat d'aquesta) a l'arrel de l'arbre. Si era diferent de 0, els fills esquerra i dret (en aquest ordre) es creen a partir de la mateixa llista. Aquest algorisme recursiu segueix el preordre i aprofita la mutabilitat de la llista per què quan arriba al fill dret, l'esquerra ja ha esborrat els seus elements de la llista, i l'últim element de tots serà un 0 que acabarà amb la recursivitat:

```
class __Arbre:
    def __init__(self, preordre):
        assert preordre
        self.__root = preordre.pop(0)
        if self.__root != 0:
            self.__left = __class__(preordre)
            self.__right = __class__(preordre)
```

Per obtenir el subarbre en inordre, si l'arrel és diferent de 0 (és un arbre no buit), primer s'obté l'inordre dels fills esquerre i dret, perquè si són buits vol dir que només caldrà incloure el l'arrel de l'arbre si té el tret. Si no són buits però l'arrel de l'arbre no té el tret s'inclourà en negatiu (per indicar que no el té però forma part del subarbre), i si no son buits però l'arrel també té el tret doncs quedarà tot en positiu. L'ordre per ajuntar-los és el fill esquerre primer, després l'arrel després el dret, el descrit per l'inordre:

```
def distribucio(self, tret, individus):
    if self.__root != 0:
        left = self.__left.distribucio(tret, individus)
        right = self.__right.distribucio(tret, individus)
        if individus[self.__root].te(tret):
            n = [self.__root]
        elif left or right:
            n = [-self.__root]
        else:
            n = []
        return left + n + right
    else:
        return []
```

Individu

Els individus han de poder crear instàncies de Parell de Cromosomes, ja que serà un dels seus atributs juntament amb el conjunt de trets, que hem decidit que fos un set per poder-ne afegir indiscriminadament sense que es repeteixin:

```
from parell_cromosomes import ParellCromosomes
```

```
class Individu:
    def __init__(self, s):
        self.__parell = ParellCromosomes(s)
        self.__tretos = set()
```

Per afegir un tret s'utilitza el mètode add del conjunt de trets i es retorna el parell de cromosomes per a poder fer la intersecció en el tret:

```
def afegir(self, tret):
    self.__tretos.add(tret)
    return self.__parell
```

Per comprovar si es té un tret literalment es retorna el booleà associat a la qüestió tret in set:

```
def te(self, tret):  
    return tret in self.__trets
```

Hem aprofitat el mètode `__str__` de python per a poder escriure les característiques de l'individu amb la crida `'print(individu)'`. Aquesta consisteix amb la string associada al parell de cromosomes i una llista ordenada alfabèticament dels trets de l'individu. Per obtenir l'ordre alfabètic s'itera sobre la llista retornada per `sorted(set)`:

```
def __str__(self):  
  
    s = str(self.__parell)  
    for tret in sorted(self.__trets):  
        s += '\n ' + tret  
    return s
```

Conjunt de trets

El conjunt de trets s'inicialitza amb un únic atribut que consisteix en un diccionari buit:

```
class ConjuntTrets:  
    def __init__(self):  
        self.__trets = {}
```

Per afegir un individu a un tret, si el tret encara no s'havia observat en cap individu es defineix l'entrada amb el nom del tret al diccionari com a la tupla parell de cromosomes de l'individu i un conjunt amb els identificadors dels individus. Si el tret ja s'havia contemplat s'interseccionen els parells de cromosomes i s'afegeix l'identificador de l'individu al conjunt:

```
def afegir(self, tret, i, parell):  
    if tret in self.__trets:  
        self.__trets[tret][0].interseccio(parell)  
        self.__trets[tret][1].add(i)  
    else:  
        self.__trets[tret] = parell, {i}
```

Per consultar un tret s'escriu el nom del tret, seguit de i parell de cromosomes associat (els gens que comparteixen els individus que mostren el tret) i els identificadors ordenats dels individus, novament iterant sobre `sorted(set)`:

```
def consulta(self, tret):
    if tret in self.__trets:
        print(' ', tret)
        print(self.__trets[tret][0])
        for i in sorted(self.__trets[tret][1]):
            print(' ', i)
    else:
        print(' error')
```

Parell de Cromosomes

El parell de cromosomes separa la string en els dos cromosomes i amés els separa com a caràcters en una llista cada un:

```
class ParellCromosomes:
    def __init__(self, s):
        self.__m = len(s)//2
        self.__c1 = [g for g in s[:self.__m]]
        self.__c2 = [g for g in s[self.__m:]]
```

Per a calcular la intersecció amb un altre parell s'itera sobre les llistes per parelles i si una parella no coincideix es substitueix el bit de cada cromosoma per un guió '-':

```
def interseccio(self, parell):
    for i in range(self.__m):
        if self.__c1[i] != parell.__c1[i] or self.__c2[i] != parell.__c2[i]:
            self.__c1[i], self.__c2[i] = '--'
```

Per escriure el parell de cromosomes es creen les dues string associades als cromosomes i s'ajunten afegint els pertinents espais:

```
def __str__(self):
    s1, s2 = '', ''
    for g1, g2 in zip(self.__c1, self.__c2):
        s1 += g1
        s2 += g2
    return ' ' + s1 + '\n ' + s2
```

PROVES FETES AMB L'APLICACIÓ

Per comprovar si la nostra solució és vàlida li hem passat com a entrada el document proporcionat pels professors de l'assignatura i comparat amb el que ens han proporcionat de sortida, si són idèntics, és que el programa funciona correctament. A més hem contemplat la possibilitat que el programa falli en un cas que no s'hagi donat en aquesta proposta, però no n'hem trobat cap, ja que es donava per fet que l'entrada, tot i que podia contenir una comanda en diferents línies, sempre acabava proporcionant els arguments necessaris. Si no es proporcionéssin el codi fallaria, perquè l'entrada no és l'esperada.

Document d'entrada

```
experiment 5 8
3 1 4 0 0 2 0 0 5 0 0
1111111111111111
0000000011111111
0101010100001111
1010101010101010
1100110010101010
consulta_individu 1
consulta_individu 2
consulta_individu 3
consulta_individu 4
consulta_individu 5
consulta_tret qwerty_12
distribucio_tret qwerty_12
afegir_tret qwerty_12 3
consulta_tret qwerty_12
distribucio_tret qwerty_12
afegir_tret qwerty_12 3
consulta_individu 3
consulta_tret qwerty_12
afegir_tret asdf_34 5
distribucio_tret asdf_34
consulta_tret asdf_34
afegir_tret asdf_34 4
distribucio_tret asdf_34
consulta_tret asdf_34
afegir_tret asdf_34 2
distribucio_tret asdf_34
consulta_tret asdf_34
afegir_tret zxcv_13 1
distribucio_tret zxcv_13
afegir_tret asdf_34 3
distribucio_tret asdf_34
consulta_individu 3
afegir_tret asdf_31 3
consulta_individu 3
```

```
experiment 3 3
1 2 0 0 3 0 0
111010
111000
111111
consulta_individu 1
consulta_individu 2
consulta_individu 3
consulta_tret qwert1
afegir_tret qwert1 1
```

```
consulta_tret qwert1
afegir_tret qwert1 2
consulta_tret qwert1
afegir_tret qwert1 3
consulta_tret qwert1

experiment 3 3
1 2 0 0 3 0 0
111111
111111
111111
consulta_individu 1
consulta_individu 2
consulta_individu 3
afegir_tret qwert1 1
consulta_individu 1
consulta_tret qwert1
distribucio_tret qwert1
distribucio_tret asdf

fi
```

Document de sortida

```
experiment 5 8
consulta_individu 1
  11111111
  11111111
consulta_individu 2
  00000000
  11111111
consulta_individu 3
  01010101
  00001111
consulta_individu 4
  10101010
  10101010
consulta_individu 5
  11001100
  10101010
consulta_tret qwerty_12
  error
distribucio_tret qwerty_12
  error
afegir_tret qwerty_12 3
consulta_tret qwerty_12
  qwerty_12
  01010101
  00001111
```

```

3
distribucio_tret qwerty_12
3
afegir_tret qwerty_12 3
error
consulta_individu 3
01010101
00001111
qwerty_12
consulta_tret qwerty_12
qwerty_12
01010101
00001111
3
afegir_tret asdf_34 5
distribucio_tret asdf_34
-3 5
consulta_tret asdf_34
asdf_34
11001100
10101010
5
afegir_tret asdf_34 4
distribucio_tret asdf_34
4 -1 -3 5
consulta_tret asdf_34
asdf_34
1--01--0
1--01--0
4
5
afegir_tret asdf_34 2
distribucio_tret asdf_34
4 -1 2 -3 5
consulta_tret asdf_34
asdf_34
-----
-----
2
4
5
afegir_tret zxcv_13 1
distribucio_tret zxcv_13
1 -3
afegir_tret asdf_34 3
distribucio_tret asdf_34
4 -1 2 3 5
consulta_individu 3
01010101

```

```
00001111
asdf_34
qwerty_12
afegir_tret asdf_31 3
consulta_individu 3
01010101
00001111
asdf_31
asdf_34
qwerty_12
experiment 3 3
consulta_individu 1
111
010
consulta_individu 2
111
000
consulta_individu 3
111
111
consulta_tret qwert1
error
afegir_tret qwert1 1
consulta_tret qwert1
qwert1
111
010
1
afegir_tret qwert1 2
consulta_tret qwert1
qwert1
1-1
0-0
1
2
afegir_tret qwert1 3
consulta_tret qwert1
qwert1
---
---
1
2
3
experiment 3 3
consulta_individu 1
111
111
consulta_individu 2
111
```

```
111
consulta_individu 3
111
111
afegir_tret qwert1 1
consulta_individu 1
111
111
qwert1
consulta_tret qwert1
qwert1
111
111
1
distribucio_tret qwert1
1
distribucio_tret asdf
error
fi
```

CONCLUSIONS

En un inici havíem partit de la idea de afegir classes Gen, Tret i incloure la classe BinTree. Però hem apostat per la simplicitat, per afavorir que el codi s'entengui i optimitzar-lo. D'aquesta manera hem inclòs les funcions que haguéssim associat a aquestes classes a les que les haguessin inclòs, tret de BinTree, ja que necessitàvem una classe a part per la relació de parentesc. Però com hem comentat és una classe enfocada únicament a la seva labor. D'aquesta manera hem confiat completament en l'abstracció de les diferents classes i assolit el nostre objectiu d'una manera senzilla i elegant.