Informe de la pràctica de PA2

Pablo Abella i Jaume Mora

7 de juny de 2024

Universitat Politècnica de Catalunya

Grau en Intel·ligència Artificial

Programació i Algorísmia II

Índex

1 Introducció

Aquest document conté una descripció i explicació de la pràctica realitzada per Pablo Abella i Jaume Mora per a l'assignatura de Programació i Algorísmia II.

L'objectiu de la pràctica és, mitjançant tota la programació apresa durant el curs, desenvolupar capaç d'estudiar la relació entre gens i trets entre diferents individus utilitzant cromosomes.

1.1 Descripció del Problema

Cada experiment inclou diversos individus, cadascun amb un parell de cromosomes de longitud fixa. Els individus es presenten en un arbre genealògic que defineix les seves relacions entre ells. Cada experiment conté:

- El nombre d'individus (n) i la longitud dels cromosomes (m).
- L'arbre genealògic en preordre.
- La composició dels parells de cromosomes de cada individu.

1.2 Funcionalitats del Programa

Anem executant diferentes operacions amb el nostre programa, totes elles explicades més avall. Les 6 operacions són:

- **experiment**: Crea un nou experiment.
- **afegir_tret**: Afegeix un tret a un individu.
- **consulta_tret**: Consulta la combinació de gens associada a un tret i els individus que el contenen.
- **consulta_individu**: Consulta la informació d'un individu.
- **distribucio_tret**: Mostra la distribució d'un tret dins de l'arbre genealògic.
- **fi**: Acaba l'execució del programa.

2 Descripció del `main.py`

El fitxer `main.py` és l'entrada principal del programa i gestiona el fitxer i les comandes d'entrada, coordinant totes les operacions a fer dins cada experiment. A continuació detallem les seves funcions principals:

2.1 Funcions Principals

- **Funció `main()`**: És la funció principal del programa, un bucle que llegeix cada instrucció fins a topar amb la instrucció `'fi'` que finalitza l'execució del programa. Llavors, amb un condicional explicat a continuació, crida a les funcions corresponents depenent de l'instrucció rebuda.
- **Gestió d'Instruccions**: Tenim un condicional que depenent de la instrucció fa un operació o una altra. A continuació expliquem cada operació possible:
  - `experiment`: Inicialitza un nou experiment amb els paràmetres especificats (nombre d'individus n i nombre de gens per cromosoma m). Crea els individus i el seu arbre genealògic i assigna els cromosomes a cada individu.
  - `afegir_tret`: Afegeix un nou tret a un individu especificat. Actualitza la combinació de gens que es creu que manifesta aquest tret.
  - `consulta_tret`: Consulta la combinació de gens associada a un tret específic i els individus que el manifesten.
  - `consulta_individu`: Mostra la informació d'un individu específic, incloent els seus cromosomes i els trets que manifesta.
  - `distribucio_tret`: Genera i mostra el subarbre genealògic que inclou tots els individus que manifesten un tret específic, etiquetant els nodes que no manifesten el tret amb l'identificador negat.
  - `fi`: Finalitza l'execució del programa.

La funció de main ens permet, de manera eficient, ordenar i estructurar cada una de les instruccions rebudes per part del fitxer d'entrada.

3 Descripció de l'`arbre_binari.py`

El fitxer `arbre_binari.py` conté la implementació de la classe que gestiona l'arbre genealògic dels individus. Aquesta classe permet representar i manipular l'arbre genealògic, incloent-hi la creació de nodes per a cada individu i l'establiment de les seves relacions de parentiu. Aquest fitxer va ser proporcionat a classe pel que no en fem una llarga descripció.

- **Explicació i justificació de l'estructura de dades**: Com ja hem mencionat, aquest fitxer ens va ser proporcionat per part del professorat, per tant l'estructura de dades per emmagatzemar l'arbre binari és la que van fer ells, de totes maneres aquests arbres estan implementats mitjançant una classe que en conté una altra. Tenim *ArbreBinari* que conté `Node` a part de moltes altres funcions.
  - **Node**: La classe `Node` representa un node en un arbre binari amb tres atributs:

* `val`: el valor emmagatzemat.
* `left`: una referència al fill esquerre.
* `right`: una referència al fill dret.

Justificació: Com ens van explicar a classe, l'ús de l'arbe arbre binari amb la classe `Node` ens facilita organitzar i accedir a les dades de manera jeràrquica, de manera que fem més eficients les operacions de cerca, inserció i eliminació.

4 Descripció del `conjunt_individus.py`

El fitxer `conjunt_individus.py` implementa la classe `ConjuntIndividus`, que serveix per gestionar un conjunt d'individus dins d'un experiment genètic. Aquesta classe permet mantenir organitzada tota la informació sobre els individus i proporciona diversos mètodes per interactuar amb ells.

- **Inicialització i representació**:
  - `__init__(self, num_ind)`: Constructora del conjunt d'individus amb un nombre determinat d'individus. Crea una llista buida per emmagatzemar els individus i un diccionari d'individus.
  - `__repr__(self)`: Retorna una representació de `ConjuntIndividus` (incloent la informació dels individus).
  - La classe `ConjuntIndividus` utilitza dues estructures de dades principals: una llista i un diccionari.
    * `self._lst_ind`: Tenim una **llista** que s'inicia sencera com a `None`. Aquesta estructura permet un accés ràpid als individus per índex, cosa que ens facilita operacions que requereixen recerca o recorregut per l'índex.
    * `self._individus`: Aquest **diccionari** s'utilitza per emmagatzemar els individus amb els seus identificadors com a claus. Això permet una cerca eficient dels individus per identificador.
  - La combinació entre **la llista i el diccionari** permet utilitzar tant l'accés per índex com la cerca per identificador. Aquesta dualitat d'estructures ajuda amb l'eficiència de totes les funcions de gestió d'individus.
- **Gestió d'Individus**:
  - `_list_ind(self)`: Retorna la llista d'individus.
  - `ind(self, individu)`: Retorna un individu específic de la llista donat el seu id.
  - `afegir_tret(self, nom_tret, persona)`: Afegeix un nou tret a un individu específic.
  - `llegir_individu(self, num_gen)`: Llegeix i inicialitza els cromosomes per a cada individu.
  - `consulta_individu(self, individu)`: Consulta la informació completa d'un individu (cromosomes i trets).
  - `obtenir_individu(self, id)`: Obté un individu específic pel seu identificador.

Aquesta estructura modular permet gestionar eficientment els individus i les seves relacions dins dels experiments, facilitant les operacions de consulta i actualització de les dades genètiques.

5 Descripció del `individu.py`

El fitxer `individu.py` defineix la classe `Individu`, que emmagatzema la informació genètica de cada individu. Aquesta classe proporciona mètodes per gestionar els cromosomes i trets associats a cada individu.

- **Inicialització i representació**:
  - `__init__(self, parell, num_gen)`: Constructor que inicialitza un individu amb un parell de cromosomes i el nombre de gens especificat. També inicialitza les llistes i conjunts necessaris per gestionar els trets.
    - La classe `Individu` utilitza diverses estructures de dades:
      - `self._cromosomes`: Aquesta estructura és **una llista** que conté dos elements, cadascun representant un cromosoma. Aquesta configuració facilita l'accés i manipulació dels cromosomes de manera eficient.
      - `self._trets`: Aquest **conjunt** s'utilitza per emmagatzemar els trets que manifesta l'individu. L'ús d'un **conjunt** garanteix la unicitat dels trets i permet operacions d'afegir i cercar trets de manera ràpida.
    - La combinació de **llistes i conjunts** permet gestionar eficientment la informació genètica i els trets dels individus. Les llistes ofereixen un accés ràpid als cromosomes, mentre que els conjunts asseguren una gestió òptima dels trets, evitant duplicats i facilitant la cerca.
- **Gestió dels individus**:
  - `_cromosoma_1(self) i _cromosoma_2(self)`: Retornen el primer i segon cromosoma de l'individu respectivament.
  - `parell_cromosomes(self)`: Retorna el parell de cromosomes complet de l'individu.
  - `_llista_trets(self)`: Retorna la llista de trets que manifesta l'individu.
  - `nou_tret(self, tret)`: Afegeix un nou tret a l'individu, retornant `False` si l'individu ja conté el tret.
  - `existeix_tret(self, tret)`: Comprova si l'individu conté un tret específic.
  - `consulta_individu(self)`: Retorna la informació completa de l'individu (cromosomes i trets).

La classe `Individu` facilita una gestió detallada dels cromosomes i trets de cada individu de l'experiment.

6 Descripció de `conjunt_trets.py`

Al fitxer `conjunt_trets.py` es defineix la classe `Conjunt_trets`, que ens permet tenir organitzats els diversos trets, així com guardar la instància de l'arbre binari introduït com a paràmetre. La utilitat d'aquesta classe és organitzar l'accés a les instàncies de la classe `tret` i generar-ne més en cas de que es demani. També serveix per generar diversos missatges d'erros en cas de tractar d'executar comandes, dins de l'experiment, sobre instàncies de `tret` inexistents.

- **Inicialització i lectura de l'arbre binàri**

- `__init__(self,num_ind)`: Constructor del conjunt de trets que inicialitza un **diccionari** buit per emmagatzemar les instàncies de la classe tret i la variable on es guardarà la instància d'arbre binàri. També guarda en una variable el tamany dels gens, ja que aquest valor serà necessari posteriorment. S'utilitza un diccionari perquè permet accedir a les instàncies de la classe `tret` utilitzant el propi nom com a clau.
- `llegeix_arbrebinari_int`: Es correspon amb la funció proporcionada a la guia de laboratori. És cridada just després de crear la instància de la classe `Conjunt_trets` per llegir l'àrbre binari i guardar la informació a la variable pertinent.

- **Gestió de trets**
  - `_list_tret(self)`: Retorna la instància del diccionari on s'emmagatzemen les instàncies de `tret`.
  - `_element_llista(self,element)`: Retorna la instància de `tret` que es correspon a la clau del tret proporcionada.
  - `afegir_tret(self,nom_tret,persona,con_ind)`: Crida al mètode de la instància de tret corresponent `afegir_individu(self,individu,con_ind)` per afegir la informació del nou individu al que se li ha afegit el tret. Prèviament comprova si la instància d'aquest tret existeix, en cas negatiu es crea i després es crida al mètode ja mencionat.
  - `consulta_tret(self,tret)`: Comprova si existeix el tret que es vol consultar, en cas contrari retorna "error". En cas afirmatiu crida al mètode `consulta_tret(self,cond_ind)` de la instància de `tret` corresponent per obtenir la informació desitjada, que és retornada posteriorment.
  - `distribucio_tret(self,tret)`: Comprova si existeix la instància de `tret` sobre la que es vol fer la consulta, en cas negatiu es retorna "error". En cas afirmatiu es crida al mètode `distribucio_tret(self,inordre,arbre_gen)` per obtenir la informació que després retorna.

7 Descripció de `tret.py`

En el fitxer `tret.py` es defineix la classe `tret`, que emmagatzema els individus que presenten aquest tret, així com el nom del propi tret i el cromosoma que representa la intersecció dels cromosomes dels individus que el manifesten.

- **Inicialització**
  - `__inint__(self,tret,num_gen)`: Constructor que inicialitza la **llista** dels individus que presenten el tret i la variable que guardarà la instància de `Cromosoma`. La raó de utilitzar una llista, i no una altra tipus de dada, és perquè necessitàvem que fos iterable, i no era rellevant poder accedir a un element concret de forma directa amb una clau. Per acabar, es guarden en dues variables privades tant el nom del propi tret com el número de gens que corresponen a cada cromosoma.

- **Gestió dels trets i mètodes sobre aquests**
  - `_lst_individus(self)`: Retorna la llista dels individus que presenten el tret.

- `afegir_individu(self,individu,con_ind)`: Mètode que s'encarrega d'afegir a la llista d'individus el nou individu, sempre que aquest no presenti ja el tret; en aquest cas es retornarà `False`. A més, es modificarà la instància de `Cromosoma` que es guarda de forma privada perquè es correspongui amb la intersecció dels cromosomes dels individus. En cas de que es tracti del primer individu en presentar el tret en qüestió, la istància de `Cromosoma` serà idèntica a la d'aquest.
- `_camins(self,lst,arbre)`: Aquest mètode, amb l'ajut de la funció auxiliar `camins_aux(arbre,node,cami)`, busca el camí, dins de l'arbre genealògic, que va des de l'arrel fins a cadascun dels individus que presenten el tret. Conforme va passant pels diferents nodes de l'arbre, la funció guarda el valor de les arrels en una **llista** que acava retornant. La gràcia d'aquest tipus de dades és que és iterable, qualitat que busquem i raó per la que l'hem triada. Aquests valors són posteriorment guardats en un **set**, que finalment retorna el mètode principal. La raó d'utilitzar aquest tipus de dada (el `set`) és que evitem que hagin repeticions de nodes, fet bastant freqüent en els camins del arbres, ja que aquests perteixen de l'arrel.
- `distribucio_tret(self,inordre,arbre_gen)`: Mètode que es centra en distingir quins elements de l'inordre de l'arbre genealògic que es passa com a paràmetre, o bé presenten el propi tret, o bé pertanyen al camí que va des de l'arrel de l'arbre a un dels inividus que el presenten. Aquells individus de l'inordre que no compleixin cap d'aquestes condicions seran descartats. D'aquells que sí que en formin part, els que no presentin el tret romandran positius, mentre que els que no en formin, passaran a ser negatius. A mesura que es van fent les comprovacions, aquells elements acceptats seran transformats en `strings` que s'aniran concatenant per poder-ne retornar una única al final de l'execució.
- `consulta_tret(self)`: Retorna la representació del tret segons els cànons estipulats.

8 Descripció de `cromosoma.py`

En el fitxer `cromosoma.py` es defineix la classe `Cromosoma`, que emmagatzema la informació necessària sobre cada element, així com defineix els mètodes que actuen sobre aquests.

- **Inicialització**
  - `__init__(self,parell,num_gen)`: Constructor que divideix el parell de cromosomes incial en dos i els guarda en dues variables privades del tamany que indica l'argument `num_gen` (número de gens de cada cromosoma). Aquestes són del tipus **string** perquè no necessitem modificar-les amb els mètodes, sinó només un objecte iterable i fàcilment comparable. A més, la lectura de l'input amb `Pytokr` ve donada com a `string`.
- **Mètodes sobre els cromosomes**
  - `primer_cromosoma(self)`: Retorna l'string que es correspon amb el primer cromosoma.
  - `segon_cromosoma(self)`: Retorna l'string que es correspon amb el segon cromosoma.
  - `interseccio(self,altre_crom)`: Amb l'ajut de la funció auxiliar `interseccio_aux(crom_1,crom_2,interseccio)` es busquen les coincidències element a element, respectivament, entre el primer i segon cromosoma de cada instància de `Cromosoma`. Seguidament, es modifiquen les strings de les interseccions per tal de que només es mantingui el valor numèric en aquelles posicions en que ha hagut coincidència en ambós elements del parell. Finalment, es retornen ambues interseccions.

9 Jocs de proves propis

9.1 Creador de jocs de prova en Python

Per a poder crear jocs de proves hem implementat un fitxer propi en Python que crea diversos jocs de proves amb diversos experiments. Per començar cal definir unes variables constants que ens definiran quants experiments farem i com seran aquests experiments. Les variables constants a definir són les següents:

- `NUM_EXPERIMENTS`: Nombre d'experiments a generar.
- `MAX_INDIVIDUS`: Nombre màxim d'individus per experiment.
- `MAX_LLARG_CROMO`: Longitud màxima dels cromosomes.
- `NUM_MAXIM_INSTRUCCIONS`: Nombre màxim d'instruccions per experiment.
- `NOM_TRET_LLARGADA`: Longitud del nom dels trets generats.

El fitxer de jocs de proves té diverses funcions per generar les dades necessàries, tot el que genera és sempre aleatori dins d'uns marges marcats per les variabes constants anteriors. Generem diferents coses, com estan detallades a continuació:

- **Funció** `generar_arbre_binari(n)`: Genera un arbre binari amb `n` nodes imparells.
- **Funció** `gererador_de_trets(num)`: Genera una llista de trets aleatoris amb longitud `NOM_TRET_LLARGADA`.
- **Funció** `generar_instruccions(n)`: Genera una llista d'instruccions aleatòries entre les possibles que tenim.

Per cada experiment, generem diferents individus i cromosomes. A més a més es crea un arbre binari representant l'estructura de l'experiment. Finalment, generem les instruccions per manipular els individus i els seus trets durant l'experiment.

Amb aquesta configuració hem pogut generar i fer servir diferents jocs de proves que ens han ajudat a assegurar que la implementació de la pràctica fos correcta. Tots els jocs de prova que hem utilitzat estan adjunts al zip de la pràctica. Els hem comprovat a mà per mirar que tinguessin sentit.