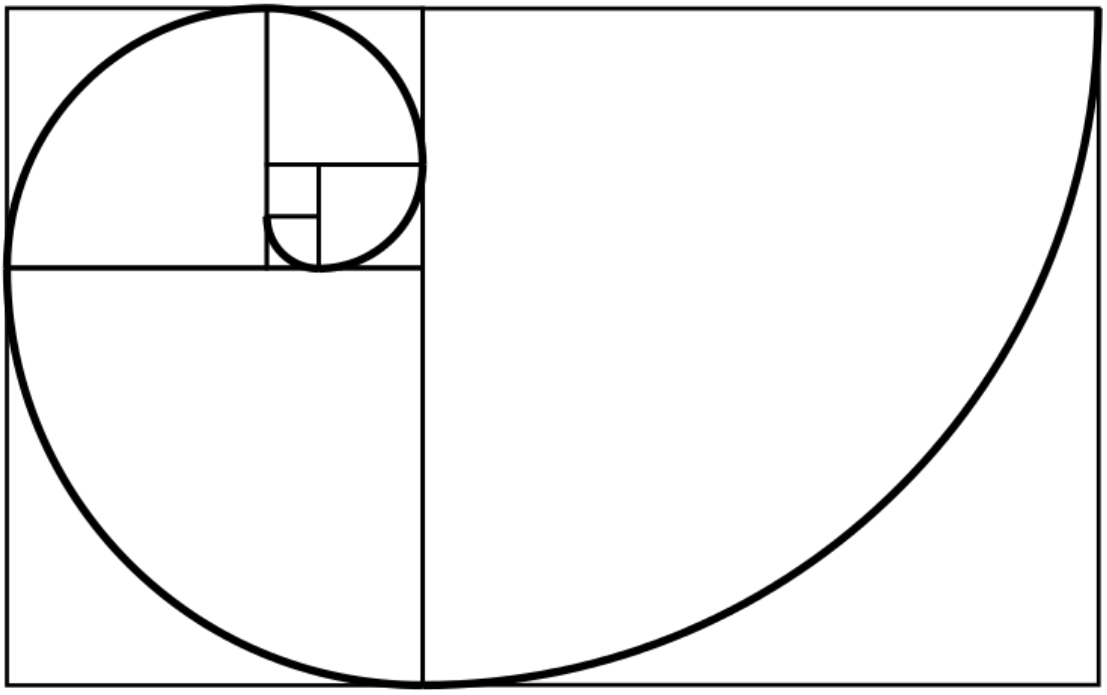
Algorismes i estructures de dades



Facultat d'Informàtica de Barcelona

Curs 2023-2024

Quadrimestre Tardor

Daniel Cañizares

Jordi Otal

Pau Rambla

Algorisme Branch & Bound

L'algorisme implementat per a la resolució de QAP és el proposat a la pàgina web de l'assignatura, seguint els passos de [https://www.cs.upc.edu/prop/data/uploads/prop_q1_2023_2024.pdf]. En aquest apartat de documentació es comenten alguns dels aspectes de disseny de l'algorisme més importants, així com una descripció general del seu funcionament.

La solució Branch & Bound al problema QAP presentat s'ha separat en tres classes diferenciades: *BranchBound*, *Hungarian* i *CompleteAssignation*.

class *BranchBound*

És la classe encarregada d'executar l'algorisme de branching, i retornar una solució a partir de la qual es genera el teclat.

- **Descripció:**

Branching: Per a l'algorisme de branching s'ha escollit una estratègia similar a *eager*, on anem desenvolupant l'arbre de solucions fulla per fulla, en funció de la solució parcial que tenim i dels caràcters que queden per emplaçar en una tecla.

L'algorisme parteix d'una solució parcial (la qual pot estar o no buida), i calcula el cost d'afegir tots els caràcters no emplaçats a la següent tecla disponible. Escull el caràcter que ha proporcionat un cost més baix, actualitza la solució parcial afegint el nou caràcter a la següent posició i repeteix el bucle fins a tenir tots els caràcters emplaçats en una tecla.

S'ha optimitzat la velocitat de l'algoritme utilitzant una estratègia *greedy*. Després de realitzar múltiples proves, s'ha optat per a inicialitzar la solució parcial amb els 4 caràcters més freqüents a les primeres posicions del teclat. D'aquesta forma es redueix el temps d'execució de l'Hungarian Algorithm, ja que disminueix la mida de les matrius amb les que ha de treballar. També ofereix cotes amb valors molt baixos (com més baixos millor), ja que la distribució del teclat s'ha dissenyat de forma que les primeres posicions de la solució parcial corresponen a les tecles més cèntriques en el teclat, així que assignem per defecte els caràcters més freqüents a les tecles que són accessibles més fàcilment.

Bounding: En aquest cas, la funció de bounding que implementa l'algorisme és la cota Gilmore, descrita al PDF. El primer terme de la cota de Gilmore es pot calcular de forma exacta, però per als termes 2 i 3 utilitzem una aproximació calculada mitjançant Hungarian. Cal destacar que per a realitzar el càlcul de la cota donada una solució parcial són necessàries les classes Hungarian i CompleteAssignation descrites posteriorment.

Cal destacar que en la solució proposada es tenen en compte les freqüències de caràcters en ambdós sentits, és a dir, la freqüència en que el caràcter 'a' va després de 'b', i també la freqüència en què el caràcter 'b' va després de 'a'. Per a calcular el cost de l'aresta entre dos caràcters 'a' i 'b', situats en

dues tecles diferents, calculem la mitjana de la freqüència A→B i B→A, i la multipliquem per la distància entre les dues tecles on es troben emplaçats els caràcters.

- **Estructures de dades:**

Algunes de les estructures de dades utilitzades en els atributs de la classe són:

double[] AbsoluteFreqs és la array de freqüències absolutes de cada caràcter.

double[][] Frequencies és la matriu de freqüències per a cada parella de caràcters (i,j).

double[][] Distancies és la matriu de distàncies per a cada parella de tecles (i,j).

Altres tipus d'estructures de dades utilitzats han estat:

ArrayList<Integer>: un tipus de dades clau utilitzat en aquest algoritme, gràcies a la flexibilitat que ofereix a l'hora d'afegir, modificar i consultar valors emmagatzemats. Tant les solucions parcials com la solució final de l'algorisme s'implementen en forma d'ArrayList. per a interpretar un ArrayList solució: l'índex de l'element de llista representa l'ID de la tecla on es troba un caràcter, el valor de l'element d'una llista indica l'ID del caràcter.

Exemple: Solució Parcial {3, 1, 0} → En aquesta solució parcial, el caràcter amb ID=3 es troba a l'índex 0, per tant emplaçat a la tecla 0. De la mateixa forma, el caràcter ID=1 es trobarà a la tecla 1 i el caràcter ID=0 a la tecla 2.

class *Hungarian*

És la classe encarregada d'executar l'Hungarian Algorithm, algorisme el qual a partir d'una matriu ens ofereix una assignació òptima, a partir de la qual obtenim una aproximació del càlcul dels termes 2 i 3 de la cota de Gilmore. Requereix de la classe CompleteAssignation.

- **Descripció:**

Implementació de l'algorisme: l'Hungarian Algorithm s'ha implementat seguint els passos descrits en el PDF proporcionat. S'ha separat el càlcul de l'assignació més completa possible en una altra classe degut a la seva complexitat. El resultat proporcionat per l'Hungarian és en forma d'ArrayList, on l'índex de cada element representa la fila i el valor representa la columna on es troba un zero de l'assignació.

- **Estructures de dades:**

La classe que implementa l'algorisme Hungarian no té cap atribut, però l'estructura de dades més important que utilitza aquesta classe són els ***ArrayList<Integer>***, que utilitza per a representar la solució de l'algorisme, amb un funcionament similar al de la classe *BranchBound*. Altres estructures

que utilitza són arrays i matrius. Cal destacar que també crea instàncies de la classe *CompleteAssignation* per al càlcul d'assignacions que requereix l'Hungarian.

class *CompleteAssignation*

És la classe encarregada de trobar la millor assignació possible donada una matriu, s'utilitza tant en passos intermitjos de l'Hungarian com per a trobar l'assignació completa final. La seva implementació és crítica per a l'eficiència de l'algorisme.

- **Descripció:**

Implementació de l'algorisme: Per a trobar la millor assignació possible s'ha d'implementar un algorisme de *backtracking*, el qual és recursiu. Degut al gran nombre de combinacions que ha de processar aquest esdevé el coll d'ampolla de tot l'algorisme que l'implementa.

- **Estructures de dades:**

Algunes de les estructures de dades utilitzades en els atributs de la classe són:

double [][] matrix és la matriu de la qual volem obtenir la millor assignació.

int[] currentAssig és l'array que recull l'assignació que està sent comprovada per l'algorisme en el temps d'execució.

La resta d'atributs de la classe estan formats per estructures de tipus array o matrius, ja que en aquest cas no era necessari utilitzar cap *ArrayList* per a realitzar el backtracking.

Algorisme Genètic

L'ús d'un algorisme genètic per a donar una solució aproximada al problema QAP ha donat molts bons resultats, en un temps d'execució diversos ordres de magnitud inferiors a l'algorisme *BranchBound* i proporcionant cotes significativament més baixes per a les solucions resultants.

S'ha implementat aquest algorisme en una sola classe: *Genetic*.

class *Genetic*

L'algorisme genètic s'inspira en el funcionament de l'evolució biològica. En termes generals, l'algorisme parteix d'una població aleatòria i va combinant els millors membres de la població per a formar la següent generació, i així N vegades fins a acabar.

La classe *Genetic* implementa un algorisme d'aquest tipus per a trobar la millor solució aproximada, i la retorna per a poder generar un teclat.

- **Descripció:**

Primer de tot, per a calcular el cost d'una solució s'ha utilitzat un mètode vist anteriorment, el primer terme de la cota Gilmore, que és resultat del sumatori del cost de totes les arestes entre els caràcters emplaçats en una tecla en la solució.

L'algorisme parteix d'una població inicial generada aleatòriament, calcula el cost de cada individu de la població, escull els millors i els recombina per a formar la següent generació, i així fins arribar a N generacions, que acaba l'algorisme.

Les funcions més importants que realitza l'algorisme són: la funció de *Crossover* (o de combinació) que calcula un nou individu "fill" a partir de dos individus "pares", fet que ens permet combinar les solucions amb millor resultat per a fer convergir l'algorisme en una solució, i en segon lloc la funció de Mutation (o de mutació) que aplica mutacions aleatòries

- **Estructures de dades:**

La majoria de les estructures de dades utilitzades són les mateixes que en el primer algorisme, les freqüències i distàncies s'emmagatzemen en forma de matrius i les solucions es representen en forma *ArrayList*, de forma idèntica a com s'explica en la classe *BranchBound* (en el context de l'algorisme genètic una solució és equivalent a un individu, ja que es representen de la mateixa forma).

En aquest cas, es treballa també amb un ArrayList 2D, de la forma:

ArrayList<ArrayList<Integer>> Population. Aquest atribut de la classe emmagatzema tots els individus de la població.

Estructures de dades de les funcionalitat principals

class *CtrlDomini*

La classe ctrlDomini, que representa el controlador de domini, requereix de estructures de dades per emmagatzemar en memòria les diferents instàncies d'Alfabet, Teclat i Layout que s'estan usant durant l'execució del programa.

- **Estructures de dades:**

hashMap<String, Alfabet> Alfabets un diccionari on la clau és el nom de l'alfabet i el valor és la instància d'Alfabet amb aquell nom.

hashMap<String, Teclat> Teclats un diccionari on la clau és el nom del Teclat i el valor és la instància de Teclat amb aquell nom.

hashMap<Integer, Layouts> Layouts un diccionari on la clau és la mida d'un Layout i el valor és la instància de Layout amb aquella mida.

class *Pair*

La classe Pair és l'estructura de dades usada a Layout per identificar dues coordenades. És un objecte amb dos atributs públics, **first** i **second,** cadascun amb els seus corresponents metòdes *getter* i *setter*.

class *Layout*

La classe Layout és la classe que representa la distribució de les tecles disponibles on hi aniran els diferents caràcters de l'Alfabet d'un Teclat. Cada posició disponible té una id, per tant cada id té unes coordenades i una distància amb altres ids.

- **Estructures de dades:**

Per fer-ho es fan servir les estructures de dades següents.

double [][] distancies és la matriu que indica les distàncies entre dues ids, la distància entre dues ids estarà guardada així *distancies[id1][id2]*.

int [][] distribucio és la matriu que dona forma al Layout, en cada posició (i, j) amb valor diferent de -1 s'hi troba la id de la qual les coordenades són (i, j). Per al teclat es genera una matriu igual de caràcters i buida.

List<Pair<Integer, Integer>> coordenades és la llista on cada index representa una id de posició del Layout i en el valor s'hi troben les seves coordenades.

class *Alfabet*

La classe Alfabet representa un alfabet, és a dir, la classe guarda informació del caràcters que conté un alfabet, les seves freqüències i la probabilitat que donada una lletra aparegui la següent.

- **Estructures de dades:**

A continuació es detallen les estructures de dades més complexes d'aquesta classe:

Map<Character, Double> characters és un diccionari que guarda totes les lletres de l'abecedari amb la seva respectiva freqüència.

double[][] frequencies és una matriu que guarda la probabilitat que hi ha que donada una lletra aparegui la següent. La matriu està estructurada manera que si la mida de l'Alfabet és n, la matriu és de mida nxn. Llavors cada fila està associada a un caràcter de l'alfabet i amb aquest mateix ordre, les columnes. L'ordre de les lletres per cada posició ve determinat per un array de caràcters (atribut de la mateixa classe Alfabet).

A continuació es mostra un exemple simple i no real amb un Alfabet amb tres lletres a, b i c.

	A	B	C
A	0.30	0.10	0.4
B	1.0	0.0	0.0
C	0.75	0.05	0.2

Aquesta matriu s'enten com donada la lletra C, la probabilitat de que la següent lletra sigui una A és 0.75, una B és 0.05 i de nou una segona C és 0.2. A més es veu com la suma de tots els valors d'una fila és igual a 1.0

class *Teclat*

La classe Teclat guarda tota la informació que suposa un Teclat com l'Alfabet pel qual està generat, el seu Generador, el seu Layout, etc, a més gestiona totes aquestes dades.

- **Estructures de dades:**

Algunes de les estructures de dades més rellevants per la classe Teclat són les següents:

Map<Character, Integer> teclat és un diccionari que guarda la relació entre les lletres de l'Alfabet i les seves ids (posicions) en el Layout que venen generades pel generador.

char[][] distribucioCharacters és una matriu que representa un teclat. Al principi està tot buit i després de que la generació faci la relació entre caràcter i id s'omple la matriu a partir d'aquesta informació i pot donar una matriu com la següent:

X	R	S	D	L	V	Z
Ñ	I	A	T	E	U	P
W	B	O	C	M	J	Y