Descripció de les Estructures de Dades i Algorismes

Universitat Politècnica de Catalunya
Projectes de Programació

Marc Expósito Francisco | ***marc.exposito.francisco***
Pau Mayench Caro | ***pau.mayench***
Josep Díaz Sosa | ***josep.diaz.sosa***
Víctor Hernández Barragán | ***victor.hernandez.barragan***

Continguts

Per a la descripció de les estructures de dades i algorismes s'han tingut en compte únicament aquelles classes de caire algorísmic a la capa de domini: *Frequencies*, *BranchAndBoundEager* o *SimulatedAnnealing*. Classes com per exemple *Usuari* o *Teclat* no fan servir algorismes sofisticats, sinó que es dediquen a ser classes contenidores d'informació i totes les seves operacions són relativament senzilles.

1. Generació de Freqüències de paraules

La generació de freqüències de paraules fa referència a un càlcul previ i imprescindible a la generació de la distribució d'un teclat per a un tipus o registre concret de text per a un alfabet concret. Aquest càlcul previ és aquell que determina quantes vegades apareix cada paraula a un text o a una llista de freqüències, així com la detecció automàtica de l'alfabet que l'usuari desitja fer servir, doncs aquest no és decidible per l'usuari, sinó que, d'entre els existents al sistema, els algorismes de lectura de o bé un text o bé una llista de freqüències determinen l'alfabet sobre el que es construirà el teclat.

Definim una llista de freqüències com una llista de parells Paraula-Aparicions, on, cada paraula té assignat un natural que indica el nombre de vegades que apareix a un text, la importància d'aquesta llista de freqüències resideix en que aquesta és l'input fet servir per l'algorisme de resolució de QAP per a saber quines paraules apareixen més seguidament.

La classe contempla dos formes mitjançant les quals un usuari pot aportar la informació de la reiteració de les paraules: Aportant un text convencional amb estructura humana, fent ús de la funció <**genera**>, o bé aportant directament el llistat de parelles Paraula-Freqüència per mitjà de la classe <**llegir**>. Ambdós mètodes, a més de rebre com a paràmetres l'String corresponent al text o llista, reben un array d'objectes de la classe Alfabets amb cadascun dels 5 alfabets disponibles al sistema (els Europeus, que seran els que farem servir per al driver), els quals són proporcionats al controlador de domini per l'Stub del gestor dels alfabets, ja que aquests, es troben emmagatzemats a la base de dades i són els que són, sense opció a crear-ne cap.

Els dos mètodes actualitzen un l'atribut llistFrequencies de l'objecte, amb la informació de les freqüències de les paraules, emmagatzemat en un TreeMap estàndard de Java. Aquesta estructura de dades ha sigut considerada la més adient, ja que permet aparellar claus String corresponent a les paraules amb valors Integer corresponent a les vegades que apareix aquesta clau. El TreeMap és una clara opció avantatjosa gràcies a ser implementat per mitjà d'un arbre de cerca equilibrat vermell-negre. Tot i que les seves operacions bàsiques tinguin un cost logarítmic molt eficaç encara que pitjor que el HashMap, les claus ja disposen d'una ordenació natural (per definició de l'arbre de cerca equilibrat) al contrari de la taula de Hash que fa servir el clàssic HashMap i per tant, com que no sabem quants elements emmagatzemarem, és molt millor fer servir un TreeMap.

1.1. Algorismes de comptatge de les freqüències

Els dos algorismes parteixen d'una primera fase comú i idèntica consistent en:

- Transformació del text a majúscules i, per mitjà de la llibreria *Normalizer* de Java, elimina totes les marques diacrítiques que pugui presentar l'String (eliminem tots els diacrítics específics de cada idioma i ens quedem amb l'alfabet arrel), fent servir una manipulació de la codificació Unicode, ja que, així tindrem un teclat amb els caràcters base i en majúscules, com tot teclat.

- Una passada al text per a detectar els casos d'excepció, a la vegada que es detecta el codi Unicode a quin alfabets dels presents a l'Array passat per paràmetre pertanyen els caràcters per inicialitzar-ne l'atribut alfabet de l'objecte a aquest, de forma que ja sabem quin alfabet tractar-ne.

Un cop sabem a quin alfabet pertany l'entrada i està ben formatada (en majúscules i sense diacrítics), fem un segon recorregut amb un tractament personalitzar segons s'hagi escollit <**genera**> o <**llegir**>.

1.1.1. Tractament Mètode Genera

El mètode genera farà un recorregut lineal al text agrupant els caràcters que apareixen junts i separats entre els elements que no es detecten com a caràcters (els anomenats "prohibits"), que es poden veure al codi. No fem servir la funció booleana isLetter de la classe wrapped Character ja que no consideraríem elements com per exemple (') propis de llenguatges com l'anglès a les seves paraules.

Aquestes agrupacions de caràcters són les paraules del text essencialment. Cadascuna d'aquestes es consultarà en temps logarítmic al TreeMap llistaFrequencies que anem omplint, per a comprovar si ja hi és (i per tant sumar-ne 1 al comptatge), o bé encara no ha aparegut i afegir-la com a clau amb valor 1. Es procedeix fins a finalitzar el text.

1.1.2. Tractament Mètode Llegir

El mètode llegir senzillament, ja té les parelles en el format demanat, tot i que encara no assumim que estigui ben formatada l'entrada, però sí que tindrà les paraules en majúscules i sense diacrítics. En realitza un recorregut lineal detectant, per mitjà del format, paraules i els seus valors i els va introduint un a un a llistaFrequencies, a la vegada que comprova que no

estigui llegint res estrany impropi del format exigit, altrament llençarà l'excepció de '*Format incorrecte*' en temps de lectura. Aquest, clarament, és un treball computacional molt més senzill que genera.

2. Algorisme: Branch and Bound Eager

Aquesta classe s'encarrega de generar la distribució que optimitza la velocitat de tecleig i minimitza el temps entre premut de tecles, ja que, a partir de la seva entrada, la qual és la llista de freqüències, i realitzant un tractament exhaustiu a cadascuna de les paraules prestant atenció també a la seva raresa a la hora d'aparèixer. Aquesta classe s'encarrega de fer això per mitjà de la resolució del famós problema del Quadratic Assignment Problem, que cerca trobar la solució òptima per a l'efecte atenent a dues matrius que expressen els pesos i les distàncies dels caràcters.

Descrivim, per tant, de manera detallada a alt nivell el procés algorísmic per a assolir l'objectiu proposat.

Inicialització de Matrius: L'algorisme comença inicialitzant dues matrius, una per a les distàncies entre tecles (matriuDistancies), això ho fem fent la distància euclidiana i suposant que les tecles es troben a distància 1, i una altra per a la proximitat basada en la freqüència d'ús dels caràcters (matriuProximitat) en la que iniciem amb valors random de 0 a 15 així si les freqüències que rebem ens donen poca informació no tenim matrius gegants de 0 i és molt eficient, d'altra banda, si ens donen moltes dades "moltes freqüències", aquests valors random són insignificants i, per tant, no afecten a les freqüències reals donades per l'usuari; després d'omplir-la amb valors random completem aquesta matriu proximitat amb les freqüències de les paraules obtingudes, sumant a cada caràcter que està junt en una paraula la freqüència entre aquests dos caràcters.

Inicialització de Solució Parcial: Com tractem amb alfabets molt grans i teclats molt grans i aquest problema és quadràtic, el que fem és afegir caràcters directament al principi per a què el programa no es quedi penjat calculant l'opcio més eficient. El que fem és agafar els caràcters amb menys freqüència i ficar-los a les tecles més llunyanes de totes i aixi ens tenim unes assignacions amb pes mínim a les tecles amb distancia màxima.

Processament amb Branch and Bound Eager: Despres apliquem l'algorisme Branch and Bound Eager que fa servir una cua de prioritat (PriorityQueue) per explorar de manera eficient les possibles seguents solucions, prioritzant aquelles amb el millor potencial basant-se en una la cota de Gilmore-Lawler. Això ho fem creant i ficant a la cua les solucions parcials afegint a la seva següent tecla buida cada caràcter disponible, tenint així totes les

possibles solucions. El següent pas és agafar la solució parcial amb la cota mínima i buidem la cua, així ho podem fer per la definició de la cota de Gilmore-Lawler. A continuació realitzem el mateix algorisme de calcular totes les solucions parcials i agafar la que té millor cota fins que arribem a una solució subòptima.

Solucions Parcials: Es treballa amb objectes SolucioParcial, que representen estats intermedis en la cerca de la disposició òptima. Aquests objectes permeten guardar a la cua mantenir un registre de les assignacions fetes fins al moment i les opcions encara disponibles.

Calcul de la Cota de Gilmore-Lawler: Per cada solució parcial abans de ficar-la a la cua es calcula la Cota de Gilmore-Lawler, que es una cota inferior del cost de les assignacions restants. Aixo ho fem calculant una matriu amb els costos aproximats i obtenint una solució òptima d'aquest problema LAP utilitzant L'Hungarian Algorithm.

Hungarian Algorithm: El hungarian algorithm resol aquest problema d'optimització fent servir una funció complexa que calcula el mínim de línies necessàries per a cobrir tots els 0 i a part altres funcions auxiliars que aconsegueixen trobar la solució.

En resum, aquest algorisme és un mètode per trobar la disposició més eficient de les tecles en un teclat, optimitzant basat en la freqüència d'ús dels caràcters i la distància entre les tecles, utilitzant tècniques com el Branch and Bound Eager amb la Cota Gilmore Lawler i l'Hungarian Algorithm.

- Pel que fa als teclats, aquests, a nivell de domini, es representen com una matriu quadrada de char, on cada posició de la mateixa representa una tecla, i les posicions sobrants s'emplenen amb caràcters "brossa" que així ho indiquen.

3. Algorisme: Simulated Annealing

Aquesta classe s'encarrega de generar una altre distribució alternativa que optimitza la velocitat de tecleig i minimitza el temps entre premut de tecles. La generació d'aquesta parteix igual que l'anterior classe mencionada, a partir d'una llista de freqüències realitza una búsqueda que trobar una solució subóptima a la distribució del teclat, mitjançant un algorisme metaheuristic. Aquest algorisme és una bona opció per la implementació d'aquest problema ja que és un algorisme capaç de trobar una solució subóptimo, sinó óptima, a un problema combinatori i continua, com pot ser el Quadratic Assignment Problem, en un temps inferior a un algorisme de búsqueda exhaustiva mitjançant una funció heurística.

Expliquem de manera exhaustiva i a un nivell superior el procediment algorítmic per a aconseguir l'objectiu establert.

Inicialització de Matrius: Aquesta part és exactament identica a la explicació de l'apartat anterior: mitjançant les freqüències donades i les distàncies inicialitzem les matrius. A diferència de l'anterior, parlant en un nivell més baix, aquesta funció edita les variables globals en comptes de retornar-les.
Inicialització de la solució parcial: Inicialitzem una solució per donar-li al algorisme un punt de partida. En aquest cas, l'array solució és la combinació dels índexs del màxim de les sumes de les files de la matriu de proximitat on es col·locarà als índexs del mínim de les sumes de les files de la matriu de distàncies.

Paràmetres de l'algoritme: El Simulated Annealing té uns paràmetres associats que fa que l'algoritme funcioni. Aquests paràmetres són la temperatura, el ratio d'enfredament i les iteracions. Aquests són valors experimentals que modifiquen el temps que trigarà l'algorisme en trobar la solució. En aquest cas, tenim una inicialització de la temperatura de 1000, del ratio d'enfriament de 0.05 i d'iteracions de 1000.

Simulated Annealing: Aquest és l'algorisme que donada una solució inicial, una, o unes, operacions i una funció heurística, retornarà una solució final subóptima al problema:

- *Operació (swap)*: L'operació que fa el algorisme serveix per millorar la solució inicialment plantejada fins arribar a una solució millor. En aquest cas, la operació "swap" intercanvia els elements de la solució de manera aleatòria per comprovar si aquest canvi millora la solució o no. És fàcil justificar que aquesta operació és capaç de fer la busqueda per totes les àrees del domini establerts pel problema.
- *La funció heurística*: Aquesta funció és la que determina si la solució parcial establerta per l'operació és una millor solució o no. En aquest cas, l'heurística es calcula mitjançant la diferència del cost de la solució donada per l'operació i el cost de la solució anterior entre la temperatura acutal, i es comprova si aquesta probabilitat es superior a un valor aleatori. El càlcul del cost d'una solució és la suma de les distàncies entre tot parell de valors de la matriu de proximitat, sempre que no sigui 0, per el valor de la matriu de proximitat.

Donada aquestes funcions, l'algorisme comença amb la solució proposada inicialment fent-li la operació, en aquest cas la de swap prèviament explicada, de manera aleatòria. Després calcula el cost d'aquestes soluciones i fa la funció heurística per comprovar si aquesta nova solució és millor o no. Si l'heurística accepta, es queda amb la nova solució. Aquest procés es farà tantes vegades com el paràmetre d'iteracions digui. Una vegada ha acabat el bucle, la temperatura s'actualitza depenent del ratio d'enfriament. Al final de l'algorisme tindrem una solució subóptima al problema.

Aquesta última part es podria explicar fent una analogia a les temperatures que tenen dos metalls mentres es fan una aleació, ja que l'algorisme es basa en aquests canvis de temperatura.