Descripció de les ED+Algorisme

Grup 43.4:

Samuel Cilleruelo González (samuel.cilleruelo)

Sara Díaz Morales (sara.diaz.morales)

David Durban Rodrigues (david.durban)

Mateu Villaret Abio (mateu.villaret)

Versió del lliurament: 2.0

Índex

- **Alfabet:**
  Per mantenir la referència dels teclats que fan ús de l'alfabet, tenim un arraylist que guarda les id del teclat. Això ho hem fet, ja que volíem un array mantenir la llista d'ids en una estructura de mida dinàmica i, per tant, hem considerat a l'arraylist com una bona opció. Per mantenir guardat l'alfabet, hem fet ús d'un Set de caràcters inicialitzat com a TreeSet. L'estructura del set ens permet controlar que no hi hagin caràcters repetits. Està inicialitzat com a Treeset ja que volem que mantingui un ordre l'alfabet.

- **QAP**
  Rebem com a parametre de la funcio ejecutarQAP la matrix que representa el trafic entre les instal·lacions, te la forma de matriu ja que així sabem facilment que la posició (i,j) de la matriu representa el trafic entre la instalacio *i*,*j*. La matriu distancia que utilitzem per representar la distancia euclidiana entre les ubicacions, també té forma de matriu degut a que resulta més senzill de representar la distancia entre dues instalacions. Utilitzem per tenir constancia de la permutació de on estan ubicades les instalacions un array de integers degut al sistema que hem utilitzat per denotar ubicacions. Cada ubicacio esta representada per un nombre que hem assignat, per tant nomes ens cal un integer per representar una ubicació. Utilitzem ArrayList<point> durant el hungarian per guardarse la localització dels zeros assignats i Arraylist<int> per guardarse les columnes per tenir constancia de les instal·lacions ubicades i no ubicades.

- **SimulatedAnnealing**
  Utilitzem la matriu de int[][] i una matriu de double[][] per representar el trafic entre dues instal·lacions i la distància entre dues instal·lacions respectivament, com hem mencionat anteriorment. També l'estructura de array de ints per mantenir les ubicacions de les instal·lacions. Hem mantingut el sistema d'assignar un nombre per representar una ubicació de l'anterior algorisme.

- **Collections**
  En la classe Collections emprem HashMaps i ArrayLists. Els ArrayLists són de Strings i s'utilitzen per guardar els noms de les entrades, dels teclats i dels alfabets. Hem decidit fer servir aquesta estructura perquè es poden fer accessos de forma ràpida. Els HashMaps els hem usat per emmagatzemar instàncies de les classes teclat, alfabet i entrades i hem considerat que era la millor opció per així tenir com a clau els identificadors i com a valor les

instàncies. D'aquesta manera es poden recuperar les classes de forma eficient mitjançant l'identificador.

- **Node**
  Node és utilitzat només com a estructura on guardar dades d'un estat. Hem fet ús d'un array d'integers per guardar-se la ubicació de les instal·lacions.

- **Processadora**
  Hem fet ús d'un set de characters per rebre com a paràmetre l'alfabet,
  donat que és el format amb el qual hem treballat fora de la classe alfabet, i ens permet representar el conjunt de lletres que forma l'alfabet. Un map que té com a key un string i de valor un integer que ens permet guardar la llista de paraules amb freqüència i finalment un array d'ints que guarda la ubicació de les instal·lacions, com hem vist a l'algorisme, i la processadora l'usa per canviar el format d'aquest al format que volem que hi arribi al teclat.

- **Teclat**

  A teclat hem fet servir una matriu de characters per guardar quina és la distribució de les tecles. Hem considerat que era bona opció perquè necessitem una estructura que tingui files i columnes i que cada posició de la matriu contingui una lletra com un teclat real.

- **Entrada, Text i LlistaParaules**

  Hem fet servir dues estructures: HashMap i ArrayList. L'ArrayList l'hem escollit per tal d'emmagatzemar els identificadors dels teclats als quals està associada les entrades i que es puguin modificar i accedir de forma ràpida.
  El HashMap l'utilitzem per guardar les entrades en forma de llista de paraules per passar-ho posteriorment com a entrada a l'algoritme. En aquest les claus són les paraules i els valors són les freqüències. Així que ens va perfecte perquè no hi ha possibles repeticions de paraules, ens permet tenir associada la paraula amb la seva freqüència i a més, la freqüència es pot cercar i actualitzar ràpidament.

- **Netejadora**

  A netejadora utilitzem l'estructura HashMap. Aquest és perquè la classe té un mètode que s'encarrega de netejar l'entrada i com hem dit abans volem que quan actuï d'entrada a l'algoritme mantingui el mateix format.

- **Usuari**

  A usuari hem utilitzat una tres ArrayLists d'integers que emmagatzemen una els identificadors de les entrades, una altra els dels alfabets i per últim de teclats. Hem pres aquesta decisió perquè volíem fer consultes i afegir elements de forma ràpida i per fer-ho l'ArrayList és un bon recurs.

- **CtrlAlgorisme**

  El controlador rep com a paràmetre una llista de paraules amb format HashMap per passar-ho posteriorment com a entrada a l'algoritme. En aquest les claus són les paraules i els valors són les freqüències. Rebem també l'alfabet com a paràmetre que, com hem mencionat anteriorment, està format per un set de characters. Una matriu d'ints que ens donarà la processadora perquè l'algorisme en pugui fer ús, aquesta matriu representa el tràfic entre instal·lacions. Finalment, un array d'ints que registra les ubicacions de les instal·lacions.

- **CtrlDomini, CtrlEntrada**

  En aquestes classes utilitzem les estructures ArrayList de integers, Set<Character>, HashMap<String,Integer> i la matriu de caràcters Character[][] perquè com hem explicat anteriorment, són les millors opcions que hem considerat per tal de representar els ids de teclats/alfabets/entrades, per representar l'alfabet, les llistes de paraules i el teclat respectivament.

Descripció Algorismes

Primer Algorisme - QAP

Hem implementat el mètode basat en el mètode d'optimització QAP. Per implementar-la hem fet ús d'un greedy, que ens permet obtenir una cota inicial, i un branch & bound. El greedy assigna la instal·lació amb més tràfic en la primera ubicació, i llavors col·loca en la següent ubicació la instal·lació que tingui més tràfic amb les instal·lacions ubicades fins a col·locar totes les instal·lacions.

El branch & bound per poder dur a terme el bound, fa ús de la cota de Gilmore-Lawler. Aquesta cota es calcula mitjançant l'addició de tres termes. El primer terme pot ser calculat de manera exacta, però els altres dos, a priori, no. Per tant, hem utilitzat l'Hungarian algorithm, per fer una aproximació d'aquests dos termes. Generem com a estat inicial cap instal·lació col·locada. Fem el Branch & Bound eliminant aquells fills amb cost superior al de cota i guardant-nos la solució amb menor cota. Finalment retornem la millor permutació.

L'Hungarian algorithm rep una matriu amb els costos que suposaria col·locar una instal·lació no assignada a una ubicació buida. Treu el valor mínim de cada columna i cada fila de la matriu i calcula el mínim nombrede línies o columnes necessàries per cobrir tots els zeros de la matriu, això ho hem fet fent ús de backtracking. En cas que el mínim nombre de línies o columnes de zeros sigui el nombre d'instal·lacions no assignades, hem trobat la solució. En cas contrari, marquem totes les columnes que tinguin algun zero en una fila marcada. Després es marquen les files que tinguin la seva assignació en una columna marcada. Hem fet que aquests dos últims passos es repeteixin fins que no es generi cap variació. Afegim a cada número cobert dues vegades el mínim número no cobert. Sostraiem el mínim nombre no cobert a les línies no cobertes. Finalment, tornem al pas en què consultàvem el nombremínim de línies o columnes per cobrir tots els zeros de la matriu fins a trobar una solució.

En la realització del mètode s'han pres una sèrie de decisions, tals com:

- Per poder fer la matriu distància necessària per a l'Hungarian algorithm, hem fixat un nombre d'ubicacions igual al nombre d'instal·lacions per col·locar. El problema és que l'elecció d'aquestes ubicacions es arbitraria i pot donar lloc a no considerar millors distribucions de les instal·lacions.
- Per permetre un teclat que optimitzi velocitat de tecleig, hem decidit formar el teclat a partir d'un centre, sense límit de línies o columnes.

Segon Algorisme - SimulatedAnnealing

Hem implementat el mètode basat en l'algorisme de cerca local simulated annealing. Per implementar-la hem fet ús d'un greedy, que ens permet obtenir una solució inicial. El greedy assigna la instal·lació amb més tràfic en la primera ubicació, i llavors col·loca en la següent ubicació la instal·lació que tingui més tràfic amb les instal·lacions ubicades fins a col·locar totes les instal·lacions.

Després establim quatre variables que ens calen per fer el Simulated Annealing que són:

- *Steps* - nombre total d'iteracions.
- *limit* - Nombre total d'iteracions per cada canvi de temperatura (ha de ser un divisor de l'anterior)
- *lambda* - Paràmetre λ de la funció d'acceptació d'estats
- *k* - Paràmetre k de la funció d'acceptació d'estats

A continuació farem l'operador swap que ens permet intercanviar la posició de dos instal·lacions col·locades. Serà l'únic operador que ens caldrà donat que treballarem amb el resultat del greedy com a solució inicial i sempre tindrem totes les instal·lacions en ubicacions. Això ens permet utilitzar d'heuristic el primer terme, ja que estarem en espai de solucions completes. Ens mourem per gairebé tot l'espai de solucions guardant-nos la millor solució trobada fins al moment. Amb una funció que treballa en funció de les variables prèviament mencionades ens mourem per l'espai de solucions amb una aleatorietat que s'anirà reduint de manera progressiva fins que només escollim successors que millorin, l'aleatorietat vindrà determinada per la temperatura.

Capa Presentació

En la capa presentació principalment hem hagut de crear tres tipus d'estructures diferents: Botons(JButton), camps d'escriptura(JTextField) i desplegables(JComboBox). Els Botons ens han servit per a qualsevol mena de confirmació sobre una acció: consultes, creacions, eliminacions, modificacions, logins, signups, logouts i tornar enrere. Els camps ens han servit per recollir les dades introduïdes de qualsevol vista que ho demanés. Finalment, els desplegables els hem usat per mostrar els objectes disponibles d'un tipus d'un usuari abans d'executar alguna funcionalitat on es demani l'id d'algun d'aquests.

Capa Persistència

En la capa de persistència hem usat la següent estratègia: hem creat 5 fitxers els quals mantenen informació d'una classe específica, en format CSV (Coma Separated Value). Aquesta estructura és bastant còmoda de transportar i modificar, ja que en el fons és un String que amb el mètode split es converteix en una array d'Strings on cada element de l'array és un atribut de la classe, ja que els separem per comes. Hem usat també BufferedReaders i BufferedWriters per llegir i escriure dels fitxers respectivament, i StringBuilders per convertir les llistes que arriben com a paràmetres a Strings.