Documentació 1a Entrega PROP

Diagrama de Classes

Quadrimestre de tardor, curs 2023/24

Versió lliurament: 1.0

Grup 31.3

Marc Mostazo

Arnau Tajahuerce

Agustí Costabella

Francisco Torredemer

Diagrama del model conceptual de dades:

Taula de Continguts

1. Teclat

Aquesta classe fa referència als teclats que han estat creats pels usuaris, que es generen a partir d'una llista de freqüències i tenen un idioma determinat.

1.1. Atributs:

- **nom:** String
- **disposició:** char[][]. Matriu de caràcters que representa la disposició del teclat. Cada posició la ocupa una lletra. La mida d'aquesta matriu la determinen dimX i dimY.
- **dimX:** Integer. Representa el nombre de files del teclat
- **dimY:** Integer. Representa el nombre de columnes del teclat

1.2. Mètodes:

- **comprovaLayoutValid(Integer n, Integer m)**
  - Si n*m < nombre de lletres de l'alfabet de l'idioma del teclat, aleshores retorna l'excepció [LayoutNoValid]
  - Si n*m > nombre de lletres de l'alfabet de l'idioma del teclat i la mida total deixa files o columnes totalment buïdes, aleshores retorna l'excepció [LayoutMassaGran]
- **comprovaldiomes(String i1, String i2)**

    - Si l'idioma i1 != i2, aleshores retorna l'excepció [IdiomesDiferents]

- **Teclat(String nom, LlistaFrequencies llistafreq, Idioma i , Integer n, Integer m)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **Teclat (String, Idioma, Integer, Integer)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **modificarLayout(Integer n, Integer m)**
    - Comprova que el layout (n*m) sigui vàlid i si ho és, modifica la disposició amb el nou layout.

1.3. Relacions:

- Relació d'associació amb la classe Idioma. Associació "És de l'idioma" amb multiplicitat molts a 1, un Teclat és d'un únic Idioma mentre que un Idioma pot tindre molts Teclats. Navegabilitat unidireccional cap a Idioma.
- Relació d'associació amb la classe LlistaFreqüències. Associació "Es genera a partir de" amb multiplicitat molts a 1, un Teclat té és d'una única LlistaFreqüències mentre que una LlistaFreqüències pot tindre diversos teclats. Navegabilitat unidireccional cap a LlistaFreqüències.
- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, un Teclat és d'un únic Perfil mentre que un Perfil disposa de diversos teclats. No es necessita navegabilitat cap a aquesta classe.
- Relació d'agregació amb la classe Estrategia. Multiplicitat molts a 1. Navegabilitat unidireccional cap a la classe Estrategia.

2. Alfabet

Aquesta classe fa referència als alfabets que hi ha disponibles al sistema i que es generen a partir d'una llista de caràcters.

2.1. Atributs:

- **nom:** String
- **lletres:** Set<Character>

2.2. Mètodes:

- **Alfabet(String nom, Set<Character> lletres)**
  - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **afegirIdioma(String nomIdioma)**
  - S'afegeix a idiomes el nomIdioma

- **treureIdioma(String nomIdioma)**
  - Es treu d'idiomes el nomIdioma

- **numIdiomes()**
  - Retorna idiomes.size()

- **getNumLletres()**
  - Retorna lletres

- **getInfo()**
  - Retorna un String amb una frase que inclou el nom, el número de lletres de l'alfabet i les lletres de l'alfabet

2.3. Relacions:

- Relació d'associació amb la classe Idioma. Associació "Té" amb multiplicitat 1 a molts, un Alfabet pot tindre diversos idiomes però un Idioma només té un únic alfabet. No necessita navegabilitat cap a la classe Idioma.

3. HungarianAlgorithm

Aquesta classe fa referència a l'algorisme hongarès. En aquest cas, s'utilitza per calcular a partir d'una matriu de costs, on cada fila és una lletra i cada columna és una posició del teclat, l'assignació òptima, de tal manera que el cost d'assignar a totes les lletres a una determinada posició, sigui mínim.

3.1. Atributs:

- **matriu:** double[][]. Matriu que s'utilitza per a fer els càlculs de l'assignació òptima. És una matriu quadrada.
- **copiamatriu:** double[][]. Matriu original, s'utilitza per a obtenir els costs inicials de cada posició, ja que l'altre matriu es va modificant al llarg de l'execució.
- **zeroFila:** int[].Vector de 0 o 1. Sent i la fila, zeroFila[i] indica si hi ha un zero a la fila, si és 1 hi ha un zero marcat, altrament no.
- **zeroColumna:** int[].Vector de 0 o 1. Sent i la columna, zeroColumna[i] indica si hi ha un zero a la columna, si és 1 hi ha un zero marcat, altrament no.
- **filaCoberta:** int[]. Vector de 0 o 1. Sent i la fila, filaCoberta[i] indica si està coberta, si és 1 està coberta, altrament no.
- **columnaCoberta:** int[] Sent i la columna, columnaCoberta[i] indica si està coberta, si és 1 està coberta, altrament no.
- **zerosEstrellaEnFila:** int[]. Guarda les posicions dels zeros "estrella". Sent i la fila, zerosEstrellaEnFila[i] indica la columna del zero "estrella.
- **valoroptim:** double. S'obté a partir de la suma dels costos assignats de manera òptima.

3.2. Mètodes:

- **totesLesColumnesEstanCobertes():Boolean**
  - Retorna cert si totes les columnes estan cobertes, és a dir, columnaCoberta[i]=1 on i > 0 , i <columnaCoberta.length. Altrament, retorna fals.

- **trobarAssignacioOptima():Double**
    - Calcula amb la resta de mètodes el valor òptim de l'assignació de costos donats per la matriu.
- **pas1()**
    - Redueix la matriu de manera que en cada fila i columna hi hagi com a mínim un zero. Resta el valor mínim de cada fila a cada element de la fila i resta el valor mínim de cada columna a cada element de la columna.
- **pas2()**
    - Marca cada zero que troba si no hi ha cap altre zero marcat a la mateixa fila i columna.
- **pas3()**
    - Cobreix totes les columnes que tenen un zero marcat
- **pas4()**
    - Troba el primer zero no cobert (zero "estrella) i guarda la seva posició al vector zeroEstrellaEnFila. Sent i la fila del zero "estrella", zeroEstrellaEnFila[i] és la columna d'aquest.
- **pas5()**
    - Comença amb un zeroPrincipal que és el primer zero no cobert. Després, crea una cadena K de "Zeros" i "0*" alternats. Si troba un camí de zeros marcats, alterna entre "Zeros" i "0*". Si no troba un nou zero marcat, acaba la seqüència.
- **pas6()**
    - Troba el valor no cobert més petit de la matriu. Després, realitza operacions de resta i suma:
        - Es resta a tots els elements no coberts.
        - Es suma a tots els elements que estiguin coberts dues vegades.

3.3. Relacions:

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

4. Idioma

Aquesta classe fa referència als idiomes que hi ha disponibles al sistema i que es generen a partir d'un alfabet i una llista de freqüències predeterminada.

4.1. Atributs:

- **nom:** String

4.2. Mètodes:

- **Idioma(String nom, Alfabet alfabet)**
  - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **Idioma(String nom, Alfabet alfabet, Map<String, Integer> llistaParaules)**
  - Creadora de la classe, s'assignen els paràmetres i associacions corresponents, creant una nova llista de freqüències amb nom "LlistaPrednomIdioma" a partir de la llistaParaules.

- **afegirLlistaFreqPredeterminada(LlistaFrequencies llistaFreq)**
  - La llistaFreq passa a ser la nova llista de freqüències predeterminada de l'idioma si abans no en tenia cap.

- **canviarLlistaFreqPredeterminada(LlistaFrequencies llistaFreq)**
  - La llistaFreq passa a ser la nova llista de freqüències predeterminada de l'idioma

- **getLletres()**
  - Retorna el les lletres de l'alfabet de l'idioma

- **getFrequencies()**
    - Retorna les paraules i les freqüències de la llista de freqüències predeterminada de l'idioma.

- **getInfo()**
    - Retorna un String amb una frase que inclou el nom de l'idioma, el nom de l'alfabet i el nom de la llista de freqüències predeterminada de l'idioma

4.3. Relacions:

- Relació d'associació amb la classe Alfabet. Associació "Té " amb multiplicitat molts a 1, un alfabet pot ser de molts idiomes mentre un Idioma només té un alfabet. Navegabilitat unidireccional cap a la classe Alfabet.
- Relació d'associació amb la classe LlistaFreqüències. Associació "Té llista predeterminada" amb multiplicitat 1 a 1, un idioma té una llista de freqüències predeterminada i una llista de freqüències és d'un idioma. Navegabilitat bidireccional.
- Relació d'associació amb la classe Teclat. Navegabilitat unidireccional de Teclat a Idioma.

5. Perfil

Aquesta classe fa referència als perfils que hi han donats d'alta al sistema.

5.1. Atributs:

- **Usuari:** String. Identificador del perfil
- **Contrasenya:** String. Contrasenya del perfil per seguretat

5.2. Mètodes:

- **comprovaLListaNoExisteix(String nomLlista)**
    - Comprova si la llista identificada per nomLlista existeix al perfil i si no llença una excepció

- **comprovaLlistaJaExisteix(String nomLlista)**
    - Comprova si la llista identificada per nomLlista ja existeix al perfil i si ja existeix

- **Perfil(String User, String pswd)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **Perfil(String User)**
    - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **canviaUsuari(String newUs)**
    - Canvia l'usuari del perfil per newUs

- **canviaContrasenya(String newCon)**
    - Canvia la contrasenya del perfil per newCon

- **afegirLlistaFreq(String llista)**
    - Afegeix la llista de freqüències 'llista' al perfil

- **eliminaLlista(String nomLlista)**
    - elimina la llista de freqüències identificada per nomLlista del perfil

- **modificarLlista(String nomLlista, Map<String, Integer> novesEntrades)**
    - Modifica la llista de paraules de la llista identificada per nomLlista i ho substitueixi per novesEntrades\

- **getNomAllLlistes(): List<String>**
    - Obté el conjunt de noms de totes les llistes del perfil

- **consultaLlista(String nomSeleccio): Map <String,Integer>**

    - Consulta el contingut de la llista identificada per nomSeleccio

- **crearTeclatLlistaPropia(String NomTeclat, String NomLlista, idioma, int n, int m):Teclat**
    - Es crea un teclat amb NomTeclat a partir de la llista identificada per NomLLista, idioma ‘idioma’ i disposició n*m, i es guarda al perfil, retorna el teclat creat

- **crearTeclatLlistaIdioma(String NomTeclat, Idioma i, int n, int m):Teclat**
    - Es crea un teclat amb NomTeclat a partir de la llista predeterminada de l’idioma ‘idioma’ i disposició n*m, i es guarda al perfil, retorna el teclat creat

- **eliminarTeclat(String NomTeclat)**
    - S’elimina el teclat identificat per NomTeclat i s’elimina per tant de perfil

- **getNomsTeclats():List<String>**
    - obté el conjunt de noms dels teclats del perfil, retorna una llista amb els noms

- **consultaTecalt(String nomTeclat):char[][]**
    - Es consulta la disposició del teclat identificat per nomTeclat, retorna la dispocició obtinguda

- **modificarLayoutTeclat(String nomTeclat, int n, int m)**
    - Es modifica el layout del teclat nomTeclat que canvia a n*m

5.3. Relacions:

- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, una LlistaFreqüències és d'un únic Perfil mentre que un Perfil pot tindre diverses LlistaFreqüències. Navegabilitat unidireccional cap a LlistaFreqüències.
- Relació d'associació amb la classe Teclat. Associació "Té" amb multiplicitat 1 a molts, un Perfil disposa de diversos teclats mentre que un Teclat és d'un únic Perfil. Navegabilitat unidireccional cap a la classe Teclat.

6. LlistaFreqüències

Aquesta classe fa referència al les llistes que ha creat un donat perfil al sistema, que es generen a partir d'un input de l'usuari (fitxers de llista, text, o entrada manual).

6.1. Atributs:

- **Nom:** String. Identificador de la LlistaFreqüències
- **LlistraParaules:** Map<String, Integer>. Llista de paraules amb freqüències de la LlistaFrequencies

6.2. Mètodes:

- **LlistFrequencies(String nom, Idioma i)**
  - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **LlistaFrequencies(String nom, Idioma i,Map<String, Integer> LlistaParaules)**
  - Creadora de la classe, s'assignen els paràmetres i associacions corresponents

- **quitarTilde(char c): String**
  - retorna el caracter c sense accent

- **comprovarLletres(Set<String> paraules, Idioma i)**

    - COmprova que les lletres del conjunt de paraules estan incloses a l'alfabet de l'idioma i

- **modificarLlista(String novaLlista)**
    - Substitueix la llista de paraules de la classe per novaLLista

6.3. Relacions:

- Relació d'associació amb la classe Idioma. Associació "Té llista predeterminada" amb multiplicitat 1 a 1, un idioma té una llista de freqüències predeterminada i una llista de freqüències és d'un idioma. Navegabilitat bidireccional.
- Relació d'associació amb la classe Perfil. Associació "Té" amb multiplicitat molts a 1, una LlistaFreqüències és d'un únic perfil mentre que un Perfil pot tenir diverses LlistaFreqüències. No necessita navegabilitat cap a la classe Perfil.
- Relació d'associació amb la classe Teclat. Associació "Es genera a partir de" amb multiplicitat 1 a molts, una LlistaFreqüències pot tindre molts teclats mentre que un Teclat es genera a partir d'una única LlistaFreqüències. No necessita navegabilitat cap a la classe Teclat.

7. Branch and Bound

Aquesta classe fa referència al algoritme empleat per resoldre l'assignació de les lletres de l'abecedari al teclat segons les freqüències introduïdes.

7.1. Atributs:

- **Matriu de distàncies:** Double[][], guarda la distància de cada ubicació amb totes les altres ubicacions. La seva mida serà de n per n, amb n igual al nombre de lletres en el abecedari.
- **Matriu de trànsit:** Double[][], guarda el trànsit que hi ha per cada parell de lletres. La seva mida serà també per tant de n per n, amb n igual al nombre de lletres de l'abecedari.
- **Best_sol:** char[][], es el layout de la millor solució encontrada pel algoritme.

7.2. Mètodes:

- **solve(Map<String, Integer> palabrasFrec, Set<Character> lletres, int n_filas, int n_columnas)**
    - Es el mètode que es crida per la realització del algoritme. És una funció abstracta a Estratégia, ja que es defineix el mètode a cada subclase d'aquesta. Retorna l'assignació òptima del teclat per els valors introduïts.
- **pos_valida(Integer i, Integer j, Integer n_columnas, Integer n):**
    - Retorna fals per les posicions del layout de teclat que excedeixen del nombre de lletres de l'abecedari, i cert si es una posició utilitzable en el layout.
- **calculaMatDist(int n, int n_filas, int n_columnas)**
    - Omple la matriu de distàncies del Branch and Bound segons la distància Euclidiana per cada parell de posicions en el layout.
- **calculaMatTraf(Map<String, Integer> palabrasFrec, Map<Character, Integer> lletres, int n)**
    - Omple la matriu de trànsit calculant la freqüència per cada parell de lletres segons la llista de paraules i freqüències introduïda.
- **quitarTilde(char c)**
    - Retorna un string amb el caràcter introduït sense accents ni signes.
- **algoritm_bab(int n_filas, int n_columnas, Set<Character> lletres, Map<Character, Integer> letra_pos)**
    - Es la funció que realitza l'algoritme branch and bound amb una estratègia eager per recorrer las posibles solucions i arribar a la més òptima. Quan arriba a la solució definitiva, iguala la solució encontrada al layout de la classe. Comença amb una solució inicial que obté amb el mètode solucion_inicial de la classe Greedy.

7.3. Relacions:

- Relació de dependència amb la classe HungarianAlgorithm, Greedy, Nodo, NodoComparator i pos.
- Relació d'implementació amb la interfície Estrategia. BranchandBound implementa aquesta.

8. Pos

Identifica per una ubicació del teclat de 0 a n, sent n el nombre de lletres que té l'abecedari, la seva posició 'x' que fa referència a la seva fila al layout del teclat i la seva posició 'y' que fa referència a la seva columna al layout del teclat.

8.1. Atributs:

- **X:** int, la fila de la posició
- **Y:** int, la columna de la posició.

8.2. Mètodes

- **pos(int pos_x, int pos_y)**
  - La creadora de la posició que assigna els paràmetres corresponents al objecte.

8.3. Relacions:

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

9. Nodo

Representa cada solució parcial que explora l'algorisme, que segons la seva cota serà seleccionat per el procés de bounding per trobar altres nodes fins arribar al node o solució final.

9.1. Atributs:

- **Layout:** char[][] es la distribució del teclat de la solució parcial o final del node, amb un número n de lletres instal·lades a n posicions del layout.
- **Cota:** Double es el valor de l'estimació del tràfic i les distàncies de les lletres instal·lades i les que no.
- **Letras Usades:** Map<Character, pos>, conté cada lletra instalada al teclat (a la matriu layout del node) i la seva posició.

9.2. Mètodes:

- **Nodo(char[][] matriz, double cota, Map<Character, pos> ini)**
  - La creadora del node que assigna els paràmetres corresponents.
- **pos_valida(Integer i, Integer j, Integer n_columnas, Integer n)**
  - Retorna fals per les posicions del layout de teclat que excedeixen del nombre de lletres de l'abecedari, i cert si es una posició utilitzable en el layout.
- **sumaMatrices(double[][] mat1, double[][] mat2)**
  - Retorna una matriu de doubles amb la suma de les dos matrius dels paràmetres.
- **calcular_termino_1(Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
  - Retorna un double amb el cost del trànsit entre les instal·lacions ja col·locades a la matriu layout del node.
- **calculo_C1(int m, ArrayList<pos> pos_libres, ArrayList<Character> letras_libres, Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
  - Aproxima el cost del trànsit entre les instal·lacions ja col·locades i les encara no col·locades. Retorna una matriu de doubles on per cada element i j de la matriu tenim el cost de col·locar la iéssima lletra lliure a la jéssima posició lliure amb respecte les instal·lacions ja fetes.

- **calculo_C2(int m, ArrayList<pos> pos_libres, ArrayList<Character> letras_libres, Map<Character, Integer> letra_pos, double[][] Mat_dist, double[][] Mat_traf)**
    - Aproxima el cost de trànsit entre les instal·lacions no col·locades. Retorna una matriu de doubles on per cada element i j de la matriu tenim el cost de col·locar la iéssima lletra lliure a la jéssima posició lliure amb respecte les instal·lacions encara no emplaçades.

- **calcular_termino_2(int m, Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
    - Retorna un double que és el cost del terme 2, calculant les dues matrius C1 i C2 amb els dos mètodes anteriors, fent la suma d'aquestes dos matrius i posteriorment cridant al Hungarian Algorithm per resoldre l'assignació lineal òptima d'aquesta suma de matrius, que es el resultat que retorna.

- **calcular_cota(Map<Character, Integer> letra_pos, int n_columnas, double[][] Mat_dist, double[][] Mat_traf)**
    - Assigna al paràmetre cota del node el valor de la suma del terme 1 i el terme 2, utilizant els mètodes que els calculen.

9.3. Relacions:

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

10. Greedy

Aquesta classe fa referència al algoritme greedy, que busca una solució inicial per la resolució del problema QAP amb el algorisme Branch and Bound.

10.1. Atributs:

- **Millor posició:** int, guarda la millor posició del layout donat, és dir, la que té una suma acumuluda de distàncies amb les altres posicions menor.
- **Pitjor posició:** int, guarda la pitjor posició del layout donat, és dir, la que té una suma acumuluda de distàncies amb les altres posicions menor.
- **Lletra més freqüent:** character, guarda la lletra més utilitzada.
- **Lletra menys freqüent:** character, guarda la lletra menys utilitzada.

10.2. Mètodes:

- **solucion_inicial(double[][] mat_traf, double[][] mat_dist, int n_filas, int n_col, Set<Character> lletres)**
  - Primer busca la lletra més utilitzada i la menys, segons la matriu de freqüencies del paràmetre. Després busca també la pitjor i millor posició segons la matriu de distàncies que rep. Llavors el mètode retorna un node amb una matriu de layout buida, excepte a la millor posició calculada abans on posa la lletra més freqüent, i la pitjor posició on guarda la lletra menys freqüent.

10.3. Relacions:

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

11. NodoComparator

Aquesta classe és utilitzada per l'algorisme Branch and Bound per comparar els nodes segons la seva cota i ordenar-los a una cola de prioritat de menor a major cota.

11.1. Mètodes:

- **compare(Nodo o1, Nodo o2)**
  - Si el node o1 te una cota menor que la del node o2, retornarà negatiu, si son iguals, retornarà 0 i si la cota del node o2 es superior a la del node o1 retornarà un nombre positiu.

11.2. Relacions:

- Relació de dependència amb la classe BranchandBound. BranchandBound usa (depèn de) aquesta clase.

12. Estrategia

Interfície que utilitzem per aplicar el patró estrategia en el nostre disseny.

12.1. Mètodes:

- **calculaDisposicio(Map<String, Integer> palabrasFrec, Set<Character> lletres, int n_filas, int n_columnas)**
  - Mètode abstracte

12.2. Relacions:

- Relació d'implementació amb la classe BranchandBound. BranchandBound implementa aquesta interfície.