BREU DESCRIPCIÓ DE L'ALGORTIME

La funcionalitat principal del programa és l'ordenació de les lletres d'un alfabet de mida **n**, per poder ser col·locades en un teclat. Hem optat per definir el layout com una matriu de **f** files per **c** columnes. És a dir, només es poden tenir teclats rectangulars, i en cas que la mida de l'alfabet sigui inferior a **f*c**, s'afegiran lletres en blanc perquè encaixi el layout.

La classe **ControladorAlgortime** és la que s'encarrega de generar aquestes ordenacions. Hi ha dues funcions per ordenar les lletres d'un alfabet, en funció de l'objectiu que es busqui, és a dir, quin és el cost que es vol minimitzar. Les dues reben els mateixos paràmetres i són suficients per tenir una noció absoluta del problema:

- `HashMap<String, Integer> lpf`: on tenim les paraules i el seu nombre d'aparicions
- `ArrayList<Character> alfabet`: on tenim les lletres a ordenar
- `int files`: nombre de files del teclat
- `int columnes`: nombre de columnes del teclat

El que retornen les dues funcions és un `ArrayList<Character>` amb les lletres ordenades, on les **f** primeres lletres són les que van a la primera fila del teclat i així successivament.

Ara veurem les dues funcions i què es busca amb cadascuna d'elles.

1.1. CALCULAR DISTRIBUCIÓ POLZES

Aquesta manera d'ordenar les lletres busca reduir el cost del moviment que han de fer els polzes quan escrivim en un teclat de mòbil. Per a resoldre el problema, el traduïm a un **LAP**. Aquest LAP és resolt per la classe **AlgortimeLAP** que resol aquest tipus de problemes. Tot i això, és una variació del problema, ja que en comptes d'una matriu de costos tenim un **vector de costos**. Això és així perquè el cost d'una tecla del teclat és independent de la lletra que se li assigni, i com que el cost total de les assignacions és la multiplicació de la freqüència de cada lletra pel cost de la seva tecla assignada, resoldrem el problema **assignant les lletres amb més freqüència a les tecles amb menys cost**. Dit de manera més simple, les lletres que fem servir més sovint estaran més a prop dels polzes. **ControladorAlgortime** crida a la següent funció de **AlgortimeLAP**:

```
int[] resoldreCostosConstants(int[] freqüències, int[] costos)
```

Càlcul de les freqüències

Per calcular les freqüències, recorrem la **lpf** i per cada lletra augmentem la seva freqüència en el nombre de vegades que apareix la paraula. D'aquesta manera, creem un `int[]` on per cada lletra (mateixos índexos que al vector d'entrada **alfabet**) tenim la seva freqüència. Cal tenir en compte que si el vector de freqüències no és de mida f*c, s'afegiran lletres amb freqüència zero, perquè encaixi el layout.

Càlcul dels costos

Per calcular el cost de cada tecla, calculem per a cada tecla el **mínim** de les distàncies d'aquesta tecla a qualsevol de les cantonades inferiors del teclat. En un layout de **f** files per **c** columnes, decidim que la tecla (0, 0) és la cantonada esquerra superior, i la (**f** - 1, **c** - 1) és la cantonada dreta inferior. Ens quedem sempre amb el mínim ja que ens és indiferent amb quin dels dos polzes la polsem.

En resum, **ControladorAlgortime** genera un `int[]` de **f*c** on per cada tecla tenim el seu cost.

AlgortimeLAP

Aquesta classe resoldrà les assignacions, com hem dit anteriorment, assignant les lletres amb més freqüència a les tecles amb menys cost. Retornarà un `int[]` on per cada element **i** s'indica l'índex de la tecla associada a la **lletra alfabet[i]**.

1.2. CALCULAR DISTRIBUCIÓ DUES MANS

Aquesta altra manera d'ordenar les lletres busca minimitzar el cost a l'hora d'escriure en un teclat de portàtil. Ho fa transformant el problema a un **QAP**, que s'encarrega de resoldre la classe **AlgortimeQAP**, mitjançant la funció:

`int[] resoldreQAP(int[][] fluxos, int[][] costos)`.

Càlcul dels fluxos

Els fluxos representen la quantitat de vegades que dues lletres apareixen seguides. Per tant, calculem una matriu de **(f*c)*(f*c)** recorrent la **lpf** i incrementant el valor de la matriu corresponent per cada parell de lletres de cada paraula, en la mateixa magnitud que el nombre de vegades que apareix la paraula.

Càlcul dels costos

En aquest cas, tenim una matriu de **(f*c)*(f\c)** on per cada parell de tecles tenim el cost de polsar-les consecutivament. En calcular aquests costos penalitzem només dues coses:

- Que dues tecles estiguin a la mateixa columna, amb un cost de 2 (greu). Utilitzar el mateix dit dues vegades és ineficient, ja que busquem distribuir les polzades per tots els dits igualment.
- Que dues tecles estiguin a la mateixa meitat (vertical) del layout, amb un cost de 1. Utilitzar la mateixa mà dues vegades seguides és ineficient.

AlgortimeQAP

Per a resoldre el problema es fa ús de l'Algortime HillClimbing, implementat en la classe **HillClimbing**. El que fa aquest Algortime és definir una solució inicial aleatòria i anar fent tots els intercanvis d'assignacions possibles mentre es redueixi el cost total. Com que és un Algortime amb molt risc de trobar mínims locals, la funció `public int[] resoldreVarisIntents(int numIntents)` de la classe **HillClimbing** resol el mateix problema amb diferents solucions inicials diverses vegades i retorna la millor. La classe **AlgortimeQAP** crida aquesta funció amb un nombre d'intents de 50.