**DISSENY D’UN TECLAT**

# **Descripció de les estructures de dades i algorismes**

## **Projectes de Programació:** Quadrimestre Tardor 2023-24

## **Versió del lliurament: 2.1**

## **2ª Entrega**

**Grup 12.3:**

Pau Costafreda Tur

Xavier López Mañes

Hakima Marouan Marouan

Diego Rodriguez Milagro





# **Breu descripció de les estructures de dades i algorismes**

# HILL CLIMBING

## Explicació de les estructures de dades utilitzades

1. **teclatF**: Teclat que ens quedarà al final de l'execució de l'algoritme.
2. **teclat**: Teclat provisional que s'utilitzarà per millorar-lo i trobar un màxim local.
3. **teclatAux** i **teclatAux2**: Teclat al que se li ha aplicat un canvi des del teclat provisional.
4. **lletraFreq**: Mapa de les lletres de l'alfabet i el nombre de vegades que s'utilitzen.
5. **costLletres**: Matriu dels cops que va una lletra després de l’altre en l’input de l’usuari.
6. **x** i **y**: Posició natural del polze en coordenades (x,y) dins del teclat.
7. **alfabet**: String que conté totes les lletres de l’idioma del teclat.
8. **porVisitar**: Cua de prioritat de teclats (resultants d'aplicar canvis a teclats ja visitats) els quals encara no s'han visitat. És a dir, que no s'han generat els teclats resultants d'aplicar canvis en aquests teclats. Aquests teclats estan organitzats de major a menor qualitat de solució, per així, quan haguem de generar més teclats, els creem a partir dels millors.
9. **visitados**: Conjunt de teclats que ja han estat afegits a porVisitar. Serveix per a no quedar-nos en un bucle a l'hora d'agafar el teclat amb millor qualitat de la cua de teclats visitats.
10. **heuristica**: Control de qualitat del teclat per minimitzar la distància entre les lletres que es troben més freqüentment juntes a les paraules.
11. **heuristica2**: Control de qualitat del teclat perquè les lletres més utilitzades es trobin més a prop d'on es troba el polze de l'usuari usualment.
12. **heuristicaTotal**: Control de qualitat total. És a dir, la suma de les dues heurístiques.

## Explicació de l’algorisme

Aquest algoritme ens serveix per reorganitzar un teclat amb l’objectiu de trobar una distribució que sigui més còmode i ràpida d’utilitzar respecte a les paraules, frase o idioma introduït.

L'algorisme de hill climbing és cridat des de la classe de distribució en la qual s'haurà generat un teclat amb les lletres de l’alfabet escollit, endreçades aleatòriament amb les dimensions sol·licitades. El teclat generat es passa com a paràmetre (amb el nom de **teclat**), juntament amb una matriu que indica el número de vegades que cada lletra de l’alfabet es troba abans d’una altra lletra en la frase o conjunt de paraules de l’input (**costLletres**), un mapa amb el nombre de vegades que apareix cada lletra en l’input (**lletraFreq**), la posició natural del polze dins del teclat (**x** i **y**) i l'alfabet seleccionat (**alfabet**).

A partir d'aquests paràmetres, es crea una cua de teclats per visitar (**porVisitar**) i un conjunt de teclats ja visitats (**visitados**). A aquestes dues estructures s'afegirà el teclat original. A partir d'aquí, es comença un bucle en el qual es buscarà la millor distribució del teclat en l'arbre de solucions.

En aquest bucle es parteix des de la solució actual, i des d'aquí es generen dos teclats més per cada lletra que es trobi al teclat. Un dels teclats es crea canviant la posició de la lletra actual per la que té a la seva dreta, i l'altre es genera canviant-la per la que té abaix. En cas que la lletra estigui a la vora dreta o la vora de baix del teclat, es canviarà per la lletra respectiva de l'extrem contrari. Per tant, sigui n el número de lletres que hi ha a l’alfabet, el factor de ramificació serà O(2n).

Totes les solucions generades s'afegiran a **porVisitar,** per generar, més endavant, més solucions a partir de les noves, i a **visitados**, per si es tornen a generar aquestes solucions, no tornar-les a afegir a **porVisitar**. A més, per cada solució generada, es mantindrà a la variable **teclatF** si aquesta solució és la millor trobada fins el moment.

Per calcular quina de les solucions és millor, es farà amb la funció de **heurísticaTotal**, que és una combinació de **heurística** i **heurística2**. La primera heurística calcula com de prop estan les lletres que són adjacents en les paraules de l'input. I per tant quant més vegades vagin seguides dues lletres a les paraules o frase de l’input, més important serà que aquestes estiguin juntes al teclat. La segona heurística mesura com de lluny estan les tecles de la posició natural del polze. Novament, quan més vegades aparegui una lletra, la solució serà millor si aquesta està més a prop de la posició del polze.

L’algoritme va descobrint l’arbre de solucions, sempre obtenint els teclats “fills” del teclat amb millor heurística actual ja descobert. Com que podria estar infinitament descobrint solucions fins visitar totes les solucions possibles, se li afegeix un límit de 100 iteracions. Al final d’aquestes iteracions es retornarà el teclat visitat (o “fill” de visitat) amb millor heurística.

És possible que en una execució de hill Climbing, el millor teclat que es trobi no sigui molt bo, i és per això que desde la classe distribució es crida l’algoritme 10 cops amb 10 teclats aleatoris i es queda amb el millor.

En resum, l’algoritme rep un teclat amb una distribució aleatòria i des d'aquí es van fent intercanvis en les posicions de les tecles fins que es troba la millor solució local. L’algoritme es crida varis cops per trobar una solució millor i entre les diferents solucions, es queda amb la millor.

# QAP

## Explicació de les estructures de dades utilitzades

1. **alt** i **amp**: dimensions d’alçada i amplada del teclat
2. **teclat**: Matriu de caràcters sobre la qual es va creant la distribució teclat
3. **costLletres**: Igual que en l’anterior algoritme, aquesta estructura de dades és una matriu que serveix per saber el flux entre cada parell de lletres en les paraules i frase de l’input.
4. **alfabet**: Es l’alfabet amb el que es farà el teclat
5. **lletresColocades**: Vector de booleans que indica si la lletra ja ha sigut colocada al teclat.
6. **Cota**: Valor enter que representa la cota de un teclat creat durant el qap
7. **C**: Matriu auxiliar creada pel càlcul de la cota.
8. **T**: Vector auxiliar utilitzat per calcular la matriu C. El vector representa els traffics des de la tecla que es vol colocar a la resta de tecles no col·locades. Aquest vector està endreçat creixentment.
9. **D**: Vector auxiliar utilitzat per calcular la matriu C. El vector representa las distàncies des de la posició en la que es vol colocar la lletra fins les altres posicions no ocupades. Aquest vector està endreçat decreixentment.
10. **teclats**: Mapa dels teclats generats amb la seva respectiva cota. Està endreçat creixentment segons de cota.

## Explicació de l’algorisme

Aquest algoritme ens serveix per crear un teclat amb l’objectiu de trobar una distribució que sigui ràpida d’utilitzar respecte a les paraules, frase o idioma introduït.

L'algorisme de qap és cridat des de la classe de distribució. Es passa com a paràmetre el flux de les lletres (**costLletres**), l’alfabet de l’input (**alfabet**) i les dimensions demandades del teclat (**alt** i **amp**).

Es partirà de la solució buida, és a dir un teclat de dimensions [**alt**][**amp**], i a partir d’aquí es farà un bucle que tindrà tantes iteracions com lletres tingui l’alfabet utilitzat. Per cada iteració es crearan varios teclats auxiliars. Un per cada combinació posible entre les lletres no col·locades i les posicions no ocupades. I per cada teclat creat, es calcularà una **Cota** (en aquest cas utilitzarem la cota Gilmore-Lawler) i s’afegirà a al mapa **teclats**. Al final de cada iteració, i un cop ja s'han afegit tots teclats creats, s’escollirà el teclat auxiliar amb millor cota dins de **teclats**

Aquesta **Cota**, s’obtindrà com la suma de dos valors. El primer valor (terme 1) s’obtindra fent una suma del flux entre cada parell de lletres que ja esta colocada en una tecla del teclat actual multiplicat per la distància que hi ha entre les dues tecles. El segon valor (terme 2 + terme 3) és més complex. Primer de tot haurem de calcular les matrius C1 i C2 (de dimensions (M-N)x(M-N), sent M el número de lletres totals i N el número de lletres ja col·locades)

→ C1: Cada posició (i, k) d’aquesta matriu representa el cost de col·locar la lletra iéssima en la posició késsima i es calcularà amb la suma del flux de la iéssima lletra amb les lletres ja col·locades multiplicat per la distancia de la késsima posició a les posicions de les respectives lletres ja col·locades.

→ C2: Aquesta matriu es calcula amb el producte escalar dels vectors **T** (flux entre la iéssima lletra i les altres lletres no col·locades) i **D** (distancia entre la késsima posició i les altres no ocupades).

Un cop calculades, es sumaran i obtenint aixi la matriu **C**. Cada valor (i,k) d’aquesta matriu representa el cost addicional que suposaria colocar la iéssima lletra en la késsima tecla. I ara caldria esbrinar la assignació òptima de les lletres no col·locades.

Per fer-ho, utilitzarem el Hungarian Algorithm, que es farà desde una funció auxiliar. Aquest algoritme té dos parts:

1. En aquesta part es es modificarà la matriu per a que sigui cuadrada, obtenint primer el valor més gran de la matriu i afegint les files o columnes necessàries on totes les posicions tindran aquest valor trobat. Després s’obtindrà el valor més gran de cada fila i es restarà a cada posició de la fila el seu valor més gran respectiu. Per últim, es farà el mateix procediment que s’ha fet amb les files de la matriu però per cada columna de la matriu.
2. En la segona part es repetirà en el que es comprovarà si hi ha un zero en cada fila i columna de la matriu. Un cop es compleixi, s’acabarà el bucle. Per cada iteració del bucle, es calcularan les línies mínimes\* necessaries per cobrir cada 0 de la matriu. Per cada línia que recobreix un valor de la matriu se li sumarà a aquest valor el número més petit que no esta cobert per una línia. I per últim, es buscarà el menor valor de la matriu i es restarà a totes les posicions.

Un cop ja s’hagi acabat el bucle buscarem la assignació òptima\*\*, i es calcularà la suma dels valors de la matriu **C** assignats. D’aquesta manera ja tindrem el segon valor i per tant, sumant-lo al primer valor tindrem la cota de la distribució.

Com s’ha mencionat anteriorment, es repetirà aquest procés fins que ja s’hagin col·locat totes les lletres. Llavors es retornarà la distribució resultant.

[\*] Per calcular les línies mínimes s’utilitzarà una funció auxiliar. En aquesta funció es retornarà en forma de vector de booleans les línies i les columnes necessàries mínimes que cal utilitzar per tapar tots els zeros. Com a paràmetres es passarà la matriu **C** i l’altura i l’amplada del teclat. Seguidament, es buscarà l'assignació optima\*\*, i s’utilitzarà un altre vector de booleans **filesMarcades** per marcar aquelles files que no tenen una assignació. Desprès s’entrarà en un bucle en el que es marcaran, en un nou vector de booleans **colMarcades,** aquelles columnes que tinguin un zero en una fila marcada i per últim es marcaran totes les files que tinguin un zero en una columna marcada.

[\*\*] Per trobar l’assignació óptima d’una matriu, s’utilitzarà una funció auxiliar recursiva. Cada execució de la funció serà per fer l’assignació d’una fila. De tal manera que la funció es cridarà tants cops com files hi hagin. Els paràmetres que es passaran són la matriu **C**, el vector **assignacio** (que indicarà quina posició de cada columna està asignada i serà -1 en cas de que la columna no tingui cap posició asignada), **pos** (que indica en quina fila de la matriu es troba actualment l’execució) i **n0Final** (que guarda el numero de columnes on hi ha una assignació).

Es començarà creant un vector **asignacioFinal** com a copia del vector **assignacio**. Llavors en cas de que **pos** sigui menor al número de files es recorrerà tota la fila. En cas de que es trobi un 0, i no hi hagi un 0 ja assignat en una fila anterior, s'assignarà i es cridarà a assignació amb **pos+1, n0Final+1**. En cas que no es trobi un zero o que ja hi hagi un zero assignat en una fila anterior, es cridarà a assignació pero només fent el canvi de **pos+1**.

Quan es retorni de la crida a assignació, es guardarà el valor de la crida a **newAssignacio**. Si aquesta assignació té més zeros que **asignacioFinal** llavors s’actualitzara **assignacioFinal** amb el valor de **newAssignació**.

# Altres classes

També hi ha altres estructures de dades rellevants que hem utilitzat en diferents classes del sistema, i són les següents:

* Map<Character, Integer> lletraFreq & Map<String,Integer> frequenciaParaules: són dos maps, on en el primer cas la clau és la lletra i conté el cops que apareix i en el segon cas, la claus són diferents paraules i el valor és la frequència en la que apareix.
* HashMap<Integer, Pos> LletraPos: és un map on la clau és la lletra del alfabet i que conté la posició eix x i y on s’hi troba.
* Pair<T,T>: per a identificar la posició dels diferents caràcters (Pos), vam trobar que l'ús de pairs facilitaria molt la feina a fer i faria més fàcil els retorns d’algunes funcions. És per això que es va crear una classe Pair genèrica.