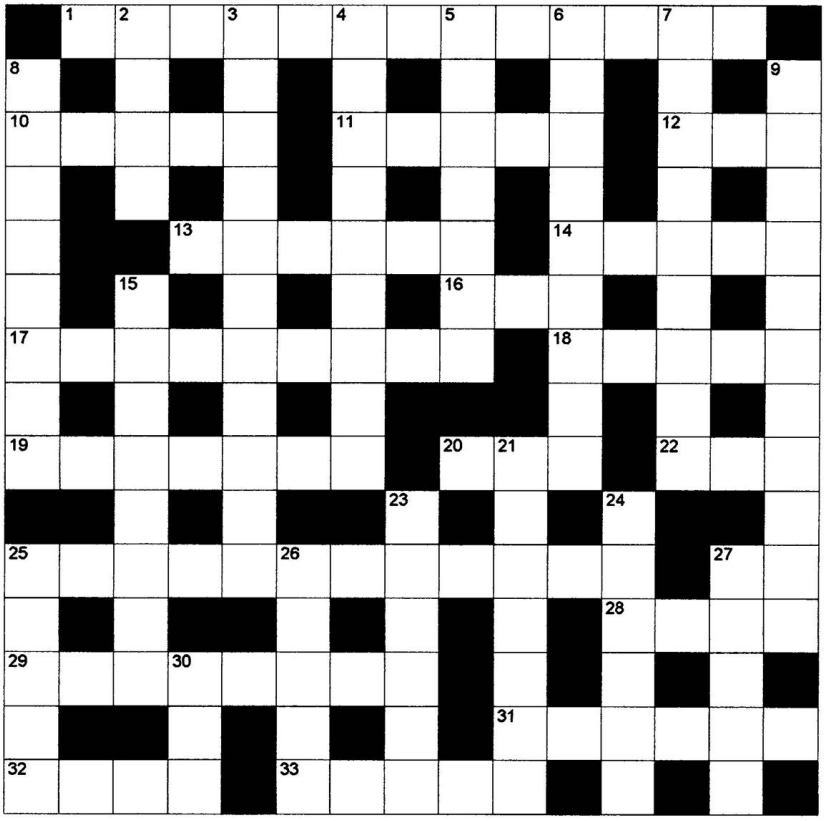
Pràctica 1

Cerca amb restriccions (mots encreuats)



Universitat Autònoma de Barcelona

Nom: Daniel Alonso Pérez
NIU: 1531551

Index

Introducció

En aquesta pràctica estarem estudiant i entenent les cerques amb restriccions. En aquest cas se'ns proposa un problema que consisteix en, donat un diccionari amb múltiples paraules[fig. 1] i un esquema d'un tauler de mots encreuats[fig. 2], fer que el programa trobi, si hi existeix, una solució a aquest amb les paraules del diccionari[fig. 3].

fig.1

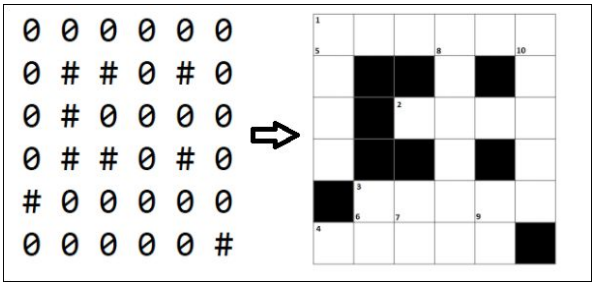


fig.2

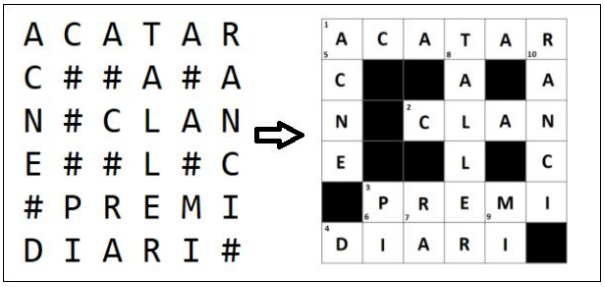


fig.3

La pràctica està dividida en tres apartats.

Primer apartat (C)

En el primer apartat ens donen un tauler de 6x6[fig. 2] i un diccionari de tan sols 100 paraules. Ens demanen que implementem l'algorisme de backtracking per trobar una solució al tauler.

Com sabem, per aplicar l'algorisme de backtracking, haurem de declarar unes variables, uns dominis i unes restriccions. Aquests són els principals problemes que trobem.

Segon apartat (B)

En aquest apartat, també ens donen el mateix tauler i diccionari, però ara ens demanen que implementem, a més, backtracking.

Potser haurem de canviar les variables, dominis i restriccions per tal de poder aplicar-ho

Tercer apartat (A)

Aquest és l'últim apartat. Ens demanen que modifiquem el nostre algorisme per a que pugui trobar una solució amb un tauler de 12x12 i amb un diccionari de 580.000 paraules en un temps prou curt.

Això vol dir que haurem de modificar l'algorisme per fer-ho ràpid i eficient.

Solució proposada

Ara explicaré les solucions que he pensat per resoldre aquest problema:

Primer apartat (C)

Per començar, vaig haver de pensar quines serien les variables, les restriccions i els dominis.

De manera general vaig decidir que les variables serien les paraules a omplir en el tauler. Contarem que una paraula és vàlida quan hi tingui dues lletres com a mínim. Per tant, en aquest tauler podem concloure que hi haurien 10 variables que omplir[fig. 4].

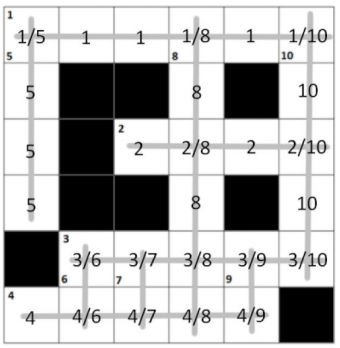


fig. 4

D’altra manera, el domini passaria a ser totes les paraules del diccionari, ja que cada variable ha d’acabar tenint assignada una paraula del diccionari.

Finalment, anem amb les restriccions.

- La primera restricció és que no pot haver-hi dues paraules iguals al tauler.

- La segona restricció és que, la paraula que triem per a assignar a una variable ha de tenir exactament la mateixa longitud com caselles disponibles hi hagi al tauler per a aquesta variable.

- I per últim, si dues variables col·lisionen entre elles, les paraules escollides han de tenir la mateixa lletra a la posició de la col·lisió.

Una vegada tenim això establert, podem començar a treballar amb el codi. El primer que farem serà **transformar l’esquema del tauler en la informació de les paraules que necessitem.**

Crearem una classe *Tauler* i una classe *Paraula* provisionals.

A la classe tauler declararem una variable *tauler* que serà una matriu per poder guardar el tauler un cop llegit per a analitzar-lo. També tindrà un array de *Paraula* anomenat *paraules* que ens permetrà guardar totes les variables i les seves informacions.

La clase paraula té un atribut anomenat *contingut*, que serà el resultat de la paraula, un altre atribut anomenat *longitud,* que guardarà quina és la longitud que ha de tenir aquesta paraula, i per últim tindrà un diccionari anomenat *restriccions* a on guardarem les col·lisions que té amb les altres paraules.

A la classe *tauler* declarem una funció anomenada *llegirTauler()*, rep un string amb la direcció del tauler a llegir.

Entrant més en detall, el que fa aquesta funció per obtenir la informació útil és, en primera instancia llegir el fitxer per fileres, mirant si en la filera que està llegint hi ha alguna paraula, d’aquesta manera llegim primerament totes les paraules horitzontals.

Per cada paraula que trobem, fem un append *paraules* amb una *Paraula* inicialitzada amb la longitud d’aquesta. Fent això acabem tenint un identificador únic per a cada paraula, que és la posició dintre de l’array de *paraules*.

Per tal d'agilitzar el procés de trobar col·lisions, mentre va comptant la llargaria de les paraules, va sobreescribint el tauler substituint cada casella per una dupla composta per *[ID de la paraula, posició de la paraula]*. Una vegada que acabem de llegir les files i emmagatzemar-les a la variable *tauler*, el tauler quedaria de la següent manera:

[0, 0]	[0, 1]	[0, 2]	[0, 3]	[0, 4]	[0, 5]
0	#	#	0	#	0
0	#	[1, 0]	[1, 1]	[1, 2]	[1, 3]
0	#	#	0	#	0
#	[2, 0]	[2, 1]	[2, 3]	[2, 4]	[2, 5]
[3, 0]	[3, 1]	[3, 2]	[3, 3]	[3, 4]	#

fig. 5

Ara llegirem les paraules verticals que hi ha. Quan trobem un valor diferent a 0 i a # sabrem que hi ha una col·lisió, per tant, ja podrem posar-la al diccionari *restriccions* abans de fer l'append amb la paraula. A més, anirem a l'altre paraula i també li setejarem la restricció amb la paraula actual.

El diccionari *restriccions* funciona de la següent manera, les claus son els ID de les paraules amb les que tenen col·lisions i el valor es una dupla composta per *[Posició de la col·lisió de la meva paraula, Posició de la col·lisió de l'altre paraula]*. Fem això per un motiu i es que, d'aquesta manera, quan anem a comprovar si una paraula que acabem d'escollir compleix amb les condicions, no farà falta mirar totes les condicions, sinó només les que estan relacionades amb aquella paraula.

Una vegada fet això ja podem passar al següent pas, que és **llegir el diccionari**.

Per fer això, crearem una funció en la classe *Tauler* anomenada *llegirDiccionari()*, que rebrà com a paràmetre la direcció del fitxer del diccionari. També declararem un diccionari anomenat *diccionari*.

El que farem ara serà separar les paraules per longitud. La manera de funcionar del diccionari és la següent, la clau serà un enter, i el valor serà una llista amb les paraules de longitud igual a l'enter de la clau.

Fet això només en queda llegir paraula per paraula, mirar la longitud i guardar-la a la seva posició pertinent al diccionari.

Ara ja podem **assignar els dominis a cada variable**. Crearem una funció anomenada *assignarDominiParaula()* a la classe *Tauler*, i crearem una llista anomenada *domini* a la classe *Paraula*.

La resposta a per què hem separat les paraulas per longitud és per a que, d'aquesta manera, posarem inicialment al domini de cada variable només les paraules amb la longitud que volem, d'aquesta manera evitem probar un munt de possibilitats impossibles i, a més, simplifiquem l'algorisme ja que no hem de tenir en compte aquesta restricció.

Per tant, el que farà aquesta funció és assignar les paraules de longitud adecuada com a domini de cada paraula.

Ja hem acabat amb els previs, ara ens podem posar amb l'**algorisme de BackTracking**. Totes aquestes classes son provisionals, ja que sinó l'algorisme tardaria moltíssim.

Per fer això crearem la funció *BackTracking()* i *TrobarSolucioBT().* Aquesta última s'encarregarà de cridar la funció *BackTracking()* amb tots els paràmetres incialitzats correctament.

BackTracking() rep 5 paràmetres:

- **LVA:** Llista de Variables Assignades. Llista amb els IDs de les variables que ja tenen una paraula assignada.
- **LVNA:** Llista de Variables No Assignades. Llista amb els IDs de les variables que encara no tenen una paraula assignada.
- **D:** Llista dels dominis de cada variable. Cada índex correspón a l'ID de cada variable. (per exemple, D[2] correspón al domini de la variable amb ID 2)
- **R:** Llista de les restriccions de cada variable. Cada índex correspón a l'ID de cada variable.
- **V:** Llista de la paraula escollida finalment de cada variable. En cas de que encara no estigui escollida la paraula hi haurà un "_". Cada índex correspón a l'ID de cada variable.

Aquesta funció retorna V, la llista de paraules finals ordenades per l'ID. En cas de que no hi hagi solució, retorna una llista buida.

Finalment vaig implementar l'algorisme de BackTracking, però amb petites modificacions.

Al començament, quan escollim el primer item de la **LVNA**, recorrem tota la llista de **restriccions** d'aquesta variable, i comprovem quines *keys()* (ID de les variables amb les que tenim les col·lisions) estan a la **LVNA** i les guardem a un diccionari auxiliar. Fem això per, quan mirem si una paraula és vàlida o no, només mirar les restriccions necessàries i estalviar-nos un if a dintre del bucle e iteracions amb restriccions innecessàries.

Per la resta ja és l'algorisme normal. Un cop fet això mirem per a tots els possibles valors del domini d'aquesta paraula, primer comprovem que la paraula escollida no estigui a **V**, per evitar repeticions, i després mirem que compleixi totes les restriccions necessàries del nou diccionari creat.

Finalment si la paraula pot ser assignada, tornem a cridar a *BackTracking()* però havent tret l'ID d'aquesta variable de **LVNA**, havent-la posat a **LVA**, i finalment, havent actualitzat la paraula a la posició corresponent de **V**.

Per tal de poder **visualitzar gràficament** el resultat, a l'hora de llegir el tauler, ens guardem a cada *Paraula* de la llista *paraules* les coordenades X i Y de la seva posició inicial i si és horitzontal o vertical. A més, un cop la funció *TrobarSolucioBT()* obté la llista V, omple la llista de *paraules* amb els valors de cada variable. Després creem la funció *mostrarGraficament()*, que s'encarrega de sobreescriure el tauler i mostrar-ho per la consola una vegada sobreescrit.

Segon apartat (B)

Des del començament vaig dissenyar el codi per a poder fer els apartats *(B)* i *(A)* sense fer molts canvis.

Vaig crear una funció anomenada *ForwardChecking()* i *TrobarSolucioFC().* Aquesta última fa el mateix que *TrobarSolucioBT(),* s'encarrega de setejar correctament tots el paràmetres d'entrada de *ForwardChecking().* Les variables que rep *ForwardChecking()* son les mateixes que rebia *BackTracking().*

La funció *ForwardChecking()* conté el mateix que *BackTracking()* però amb una petita modificació. Quan ja sabem que una variable es quedarà amb una paraula, anem a totes les variables que tenen una col·lisió amb ella gràcies a la variable **R**, i traiem totes les paraules que no compleixin amb les restriccions del domini **D** d'aquestes variables. Si alguna variable es queda sense cap paraula possible al domini fem marxa enrere i agafem una altra paraula ja que ja sabem que amb aquesta el problema no té solució.

Tercer apartat (A)

Si provem a executar el codi amb el diccionari gran tarda un munt de temps, cosa que es inadmisible. Per tal de reduir el temps vaig decidir que **triaria la variable a omplir dinàmicament**. És a dir, ara ja no agafaria sempre la primera variable de **LVNA**, sinó que ara agafaria la millor seguint un criteri.

Vaig crear una funció anomenada *ForwardCheckingRapid()* i *trobarSolucioFCRapid()*, que funcionen igual que les dels altres apartats i reben els mateixos paràmetres.

ForwardCheckingRapid() conté el mateix codi que *ForwardChecking()* però amb una petita modificació, i és que ara al començament del codi, com hem dit abans, escollim la variable de **LVNA** a escollir seguint un criteri.

En el nostre cas aquest criteri serà la quantitat de valors que hi hagi al domini, és a dir, triarem la variable que tingui un menor valor de possibles paraules al domini. I, en cas d'empat, mirarem com a segon criteri el nombre de restriccions. En aquest cas agafarem les variables que tinguin un nombre més gran de restriccions.

Resultats

Exposarem els resultats obtinguts utilitzant aquests métodes.

Primer apartat (C)

Els resultats són molt positius. Pràcticament no tarda res de temps en llegir el taulell, el diccionari i assignar el domini de les paraules, tarda tan poc que la sensibilitat de la llibreria *time* de python no és capaç de mesurar-ho. Finalment tarda aproximadament 0,001 segons en realitzar l'algorisme.

```
Temps trigat en llegir el taulell dels apartats B i C: 0.0
Temps trigat en llegir el diccionari dels apartats B i C: 0.0
Temps trigat en assignar el domini a les paraules dels apartats B i C: 0.0
Temps trigat en realitzar l'algorisme BackTracking: 0.0010209083557128906
['A', 'C', 'A', 'T', 'A', 'R']
['C', '#', '#', 'A', '#', 'A']
['N', '#', 'C', 'L', 'A', 'N']
['E', '#', '#', 'L', '#', 'C']
['#', 'P', 'R', 'E', 'M', 'I']
['D', 'I', 'A', 'R', 'I', '#']
```

fig. 6

Segon apartat (B)

Els temps del taulell, del diccionari i de les assignacions son els mateixos degut a que tant les funcions com els arxius son els mateixos. En el cas de l'algorisme, com podem veure triga encara menys, tampoc és capaç de mesurar-ho la llibreria *time* de python.

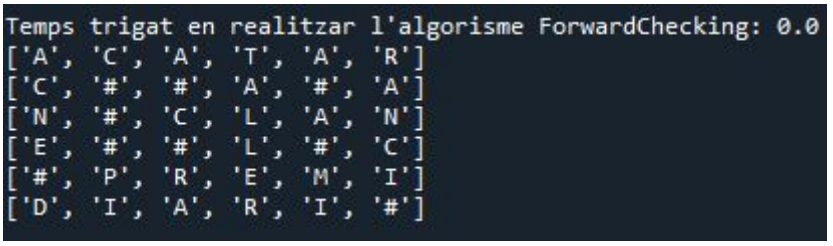


fig. 7

Tercer apartat (A)

En aquest tercer apartat com podem veure el temps en llegir el tauler i el diccionari sí que és significatiu, arribant, aquest últim, a trigar més de 0,2 segons. L'algorisme triga pràcticament 6 segons.

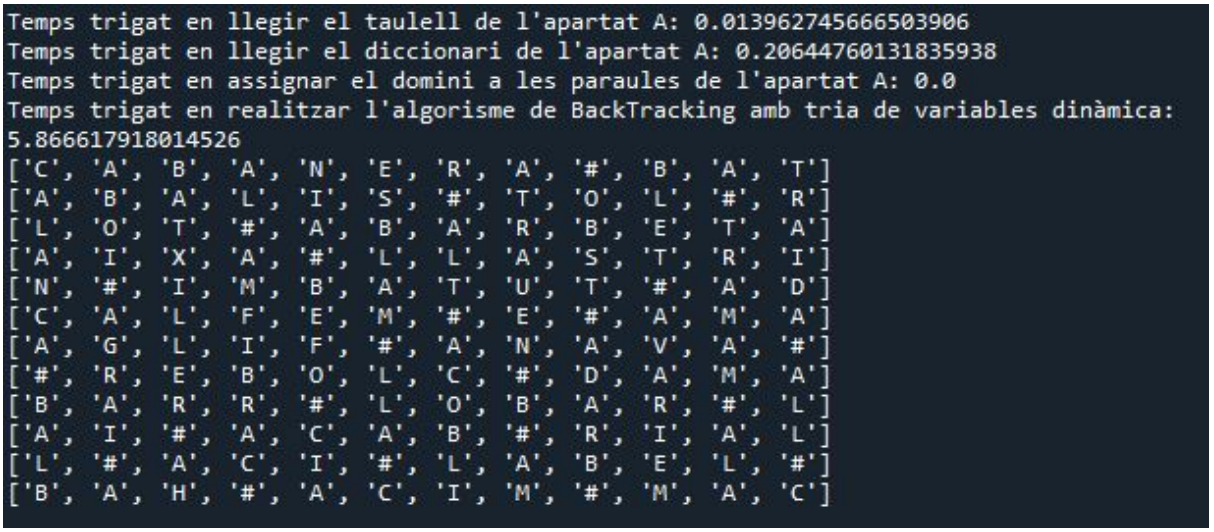


fig. 8

Analitzant els resultats

Tant a l'apartat B com al C els resultats són molt positius, el temps que triga és gairebé nul, i a més resol el problema correctament.

Relacionat amb l'apartat A podriem pensar que tarda molt en llegir el diccionari i el taulell, ja que el programa porta pràcticament mig segon i encara no ha començat amb l'algorisme, però en quant veiem els resultats de l'execució de l'algorisme veiem que no ens importa pedre mig segon perquè s'executa en només sis segons, un temps força bo tenint en compte que estem parlant d'un diccionari de 580.000 paraules i un diccionari de 12x12.

Per tant podem concloure que utilitzar BackTracking + ForwardChecking + Assignar variables dinàmicament funciona molt bé amb problemes de l'estil CrossWord.

Conclusió del projecte i treballs futurs.

He après molt fent aquest treball, t'ajuda molt a canviar de mentalitat i afrontar els problemes d'una altra manera. A més personalment és un tema que m'interessa molt, i començar a veure com ja som capaços de programar algorismes d'un nivell tirant cap a alt, que son capaços de treballar amb diccionaris d'aquestes magnituds en un temps raonable, m'apassiona. Ja tinc ganes de veure el següent treball per veure de que tracta i començar a treballar en ell.

Crec que he dedicat un temps raonable al projecte i que he treballat bé, per tant de cara als pròxims treballs he d'intentar continuar amb la mateixa dedicació.