UNIVERSITAT DE BARCELONA

Pràctica 1: Algorismes de cerca

Alejandro Guzman

23 septembre 2022

ÍNDEX

1 OBJECTIUS DE LA PRÀCTICA

L'objectiu principal de la pràctica és implementar diversos algorismes per tal de poder realitzar una cerca amb la finalitat d'arribar a un cert objectiu. Implementarem dos tipus d'algorismes:

- **Algorismes de cerca no informada**: Amb aquests algorismes no utilitzarem cap mena d'informació adicional que ens ajudi a poder saber aproximadament quanta distància ens separa en aquell precís moment de l'objectiu al qual volem arribar. Són algorismes més fàcils d'implementar però més lents ja que realitza subrecorreguts que amb informació adicional es podria estalviar.
  Els que implementarem en la pràctica són el DFS (Depth first search) i el BFS (Breadth first search).
- **Algorismes de cerca informada**: Aquests algorismes, a diferència dels de cerca no informada, utilitzen informació adicional la qual permet terminar el recorregut en una quantitat menor de temps amb una major probabilitat d'haver trobat el camí òptim fins l'objectiu. Són algorismes amb una major complexitat alhora d'implementar però amb més precisió alhora d'arribar a l'objectiu.
  Els que implementarem en la pràctica són el BFS (Best first search), conegut per ser un algorisme Greedy, i el A*.

A continuació mostraré algunes dificultats que han aparegut durant el desenvolupament de la pràctica i com les he pogut solucionar.

2 ALGORISMES IMPLEMENTATS

1 DFS (Depth First Search))

El DFS ha sigut el primer algorisme implementat. Aquest realitza una cerca en profunditat, és a dir, visualitzant un graf en forma d'arbre, l'algorisme primer realitza una cerca per una branca sencera i després pasa a una altra branca, visitant l'últim node que s'ha descobert. Es pot implementar de dues formes.

- **Iterativament**: Mitjançant una pila amb estructura LIFO (last in, first out).
- **Recursivament**

En el meu cas he realitzat la implementació de forma iterativa ja que, a priori, em resultava més senzill de fer, tot i que he trobat problemes amb la implementació.

Primer de tot he tingut diversos problemes i errors a conseqüència de la implementació de tot el codi i les classes facilitades pel professorat, la causa ha sigut que no havia prestat atenció a com estava fet el joc en si i per exemple, s'havia de moure les peces manualment amb la crida d'una funció.

En segona part, el fet de tornar enrere si s'arribava a un camí sense sortida (una branca on no n'hi han més nodes a visitar) no s'havia com implementar-ho a primera vista, i al final he decidit fer servir diversos diccionaris com a estructures per emmagatzemar informació que puc recuperar posteriorment en casos com aquest.

Finalment la construcció del *path* un cop trobada la solució ha sigut l'últim obstacle que m'he trobat, i es que utilitzant un diccionari tal i com havia fet no funcionava, per tant he utilitzat una estratègia *backtracking* per tal de construir-ho, aquesta consisteix en, amb un diccionari

d'estat actual (*key*) i predecessor (*value*), anar fent el camí fins a l'inici des de l'estat final.

Ha sigut un algorisme difícil d'implementar, però que fonamenta les bases dels següents.

2 BFS (Breadth First Search)

El BFS ha sigut més sencill d'implementar que el DFS, bàsicament perquè només s'ha cambiat una cosa, i és l'estructura de dades bàsica de l'algorisme. El BFS realitza una cerca en amplada, per tant els nodes respecten l'ordre en el que han sigut descoberts, llavors en comptes d'utilitzar una pila amb una estructura LIFO (last in, first out), s'utilitza una cua amb estructura FIFO (first in, first out), amb la qual els nodes que s'han descobert primers, seràn els primers en visitar.

Aquest algorisme acostuma a trobar el camí més ràpidament que el DFS de no ser que el DFS no estigués limitat per la profunditat dels nodes, que en aquest cas el temps que triguen és molt similar.

3 BFS (Best First Search)

L'algorisme Best First Search és el primer que s'implementarà utilitzant informació adicional amb la qual poder saber aproximadament quina distància falta fins a l'objectiu i així optimitzar el temps per tal de trobar una solució que segurament, no sigui la òptima. La informació adicional que he utilitzat ha sigut la heurística ($h(n)$), la qual és una estimació de lo proper que es troba un node a l'objectiu. Les heurístiques únicament són vàlides per a un problema (no són generals).

El Best First Search és un algorisme *greedy* el qual expandeix el node que sembla estar més a prop del objectiu (aquell que minimitza $h(n)$) mitjançant una funció d'evaluació de la heurística $f(n)$. En aquest cas és simple i només és $f(n) = h(n)$. La heurística utilitzada en aquest cas és simple i tracta només de la distància que separa el node actual de l'objectiu. Per realitzar una aproximació més precisa s'han separat els dos nodes ja que tenen moviments diferents, s'apliquen diferents funcions matemàtiques a ambdós nodes i després es sumen els resultats. Cal tenir en compte que en tots els casos el rei contrari sempre suposem que es troba en la mateixa posició, en el problema se'ns proposa la posició $O = (0,4)$

- **Rei**: El rei només es pot moure una casella en qualsevol direcció, per tant la distància que el separa de l'objectiu es pot visualitzar com el nombre de caselles que s'ha de moure (d'una en una), això es redueix a realitzar el càlcul de la **distància de Manhattan** entre el rei i la casella on es vol posicionar (objectiu). Com la casella on es vol posicionar el rei sempre serà la mateixa per al nostre problema (0,4), només tindrem una posició a situar al nostre rei per a realitzar un escac i mat (2,4).
  La Distància de Manhattan es calcula sumant els valors absoluts de la diferència de les dues coordenades, tal i com mostra la fórmula

  $$h_1(K_1, K_2) = |O_1 - K_1| + |O_2 - K_2|$$

  on $R = (K_1, K_2)$ és el nostre rei amb les seves respectives coordenades i $O = (O_1, O_2)$ és el rei contrari al qual se li vol realitzar un escac i mat.
- **Torre**: La torre es pot moure tant horitzontal com verticalment, i per estar en posició d'escac i mat ha de situar-se a la mateixa altura horitzontal que el rei contrari, per tant

s'aplicarà una funció per parts per determinar el nombre de moviments mínim per situar-la en la posició desitjada. La funció determinada per aquesta part del càlcul de la heurística és

$$h_2(R_1,R_2) = \begin{cases} 0 & si \quad R_2 = O_2 \\ 1 & si \quad R_1 \neq O_1 \ \& \ R_2 \neq O_2 \\ 2 & si \quad R_1 = O_1 \ \& \ R_2 \neq O_2 \end{cases}$$

on $R = (R_1, R_2)$ és la torre amb les seves respectives coordenades i $O = (O_1, O_2)$ és el rei contrari al qual se li vol realitzar un escac i mat. La funció està pensada per a realitzar com a màxim dos moviments, depenent de si la torre està posicionada en la mateixa distància horitzontal que el rei contrari (cas de dos moviments), o en diferent distància horitzontal i vertical (cas d'un sol moviment), o en la mateixa distància vertical (posició desitjada, per tant 0 moviments s'han de fer).

Finalment la heurística final es calcula amb la fórmula

$$h(n) = h_1(K_1, K_2) + h_2(R_1, R_2)$$

on $n = (R, K), R = (R_1, R_2), K = (K_1, K_2)$ és el node actual que indica la posició tant de la torre com del rei.

A l'hora de la implementació s'ha reutilitzat gran part del codi dels dos anteriors algorismes, l'únic canvi realitzat és l'estructura de dades on es guarden les fronteres, i es que s'ha reordenat segons la heurística anteriorment mencionada. Aquest canvi s'ha fet sense gaire problemes i l'execució ha sigut satisfactòria, reduint el temps de còmput per trobar la solució, tot i que tal i com està comprovat, aquest algorisme no garanteix trobar la solució òptima.

4 A*

Alhora d'implementar l'últim algorisme, s'ha reutilitzat gran part del codi del Best First Search i l'únic que s'ha afegit ha sigut una modificació en el càlcul de la heurística.

L'A* té una particularitat i es que evita expandir nodes que ja són cars, per tant per a cada node es calcula la fórmula

$$f(n) = h(n) + g(n)$$

on g(n) és el cost d'assolir n des de l'estat inicial, h(n) el cost estimat des de n fins a l'objectiu i f(n) el cost estimat de la millor solució que passi per n.

En aquest cas les imatges de g(n) són el depth que porta l'algorisme en un punt concret, que significa els moviments que porta des de l'inici.

L'implementació ha sigut molt senzilla i només ha fet falta implementar la nova heurística.

3 QÜESTIÓ PLANTEJADA

Ara, inicialitzeu el tauler amb el rei blanc a la posició [7,7] i torneu a executar el vostre codi. Els vostres algorismes funcionen exactament de la mateixa manera? Has hagut de canviar

alguna cosa?

El problema en realitzar una cerca en un graf es que es poden formar bucles. Llavors com s'afegeixen només els estats no visitats i els visitats amb l'excepció de que es millori la profunditat (per tal de trobar el camí òptim), cau la possibilitat de que es formin bucles.

Per a solucionar-ho es pot intercanviar (en cas de que l'estat ja hagi estat visitat però es millori la profunditat) l'estat antic amb el mateix estat però amb millor profunditat en forma de tupla per tal d'evitar possibles bucles.

En el meu cas no es formen bucles i s'executa amb normalitat, trobant la solució correctament, perquè des d'un primer moment tenia els algorismes implementats per tal d'evitar repetir estats. Tot i que es podria afegir la millora per a reemplaçar estats amb menor profunditat en cas de repetició.

4 PROVES REALITZADES

Per a provar l'optimització que suposen els algoritmes amb cerca informada, s'ha mesurat el temps d'execució dels quatre algoritmes amb les posicions del rei negre en (4,0), el rei blanc en (7, 7) i la torre blanca en (0,7). En el dfs s'ha realitzat un control de profunditat de 10. Els resultats han sigut els següents:

Algorisme	Temps (en segons)	Nº de moviments
Depth First Search	1.0274770259857178	11
Breadth First Search	3.2434282302856445	7
Best First Search	1.8738856315612793	7
A* Search	1.2371528148651123	7

Table 4.1: Temps d'execució de cada algorisme amb uns paràmetres determinats

Com es pot veure, el dfs tarda molt menys temps amb control de profunditat, en canvi la solució no és la òptima i és més llarga que les altres. El Breadth First Search troba la solució òptima però triga molt més temps. Si afegim adicional, en el cas del Best First Search troba la solució òptima i triga la meitat del temps pràcticament i si millorem la informació adicional la solució és la mateixa i es redueix el temps de còmput una mica.

5 CONCLUSIONS

La pràctica ha sigut en conclusió una bona primera toma de contacte amb els algorismes de cerca. També s'ha pogut veure com poder millorar la eficiència a partir d'informació adicional oferida durant la execució.

Personalment, també potser perquè he fet la pràctica sol, al principi he trobat diversos problemes que m'han costat continuar amb el desenvolupament del primer algorisme, però després de diversos intents he aconseguit solucionar-ho i després no he trobat gaire problemes en implementar la resta i veure les principals diferències dels algorismes de cerca no informada i de cerca informada.

Finalment, ha sigut una pràctica interessant, amb la qual he pogut aprendre a optimitzar un algorisme per tal de que trobi la mateixa solució en un temps de còmput reduït, a part de continuar aprenent programació en Python.