SISTEMES INTEL·LIGENTS

ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA INFORMÀTICA

Pràctica 1:

"RESOLUCIÓ DE PROBLEMES MITJANÇANT CERCA EN UN ESPAI D'ESTATS"

DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

València, Setembre 2024

1. Introducció

L'objectiu de la pràctica consisteix a avaluar l'eficiència, cost temporal i espacial de diferents estratègies de cerca aplicades al joc del 8-puzle. Per a això es proporciona el programa *puzle*[1] implementat en Python. Per a instal·lar i executar el programa s'han de realitzar les següents accions:

1. Còpia el fitxer **puzle.zip** que està en PoliformaT, en la carpeta recursos/2024-2025/Pràctica 1, al teu ordinador
2. Descomprimeix el fitxer**.** En descomprimir-lo es crearà un directori *puzle* on se situen els fitxers font.
3. Des d'un entorn de desenvolupament de Python (per exemple **spyder**, disponible en els laboratoris) o des d'un terminal (si té Python instal·lat en el PATH) executa el programa **interface.py**.

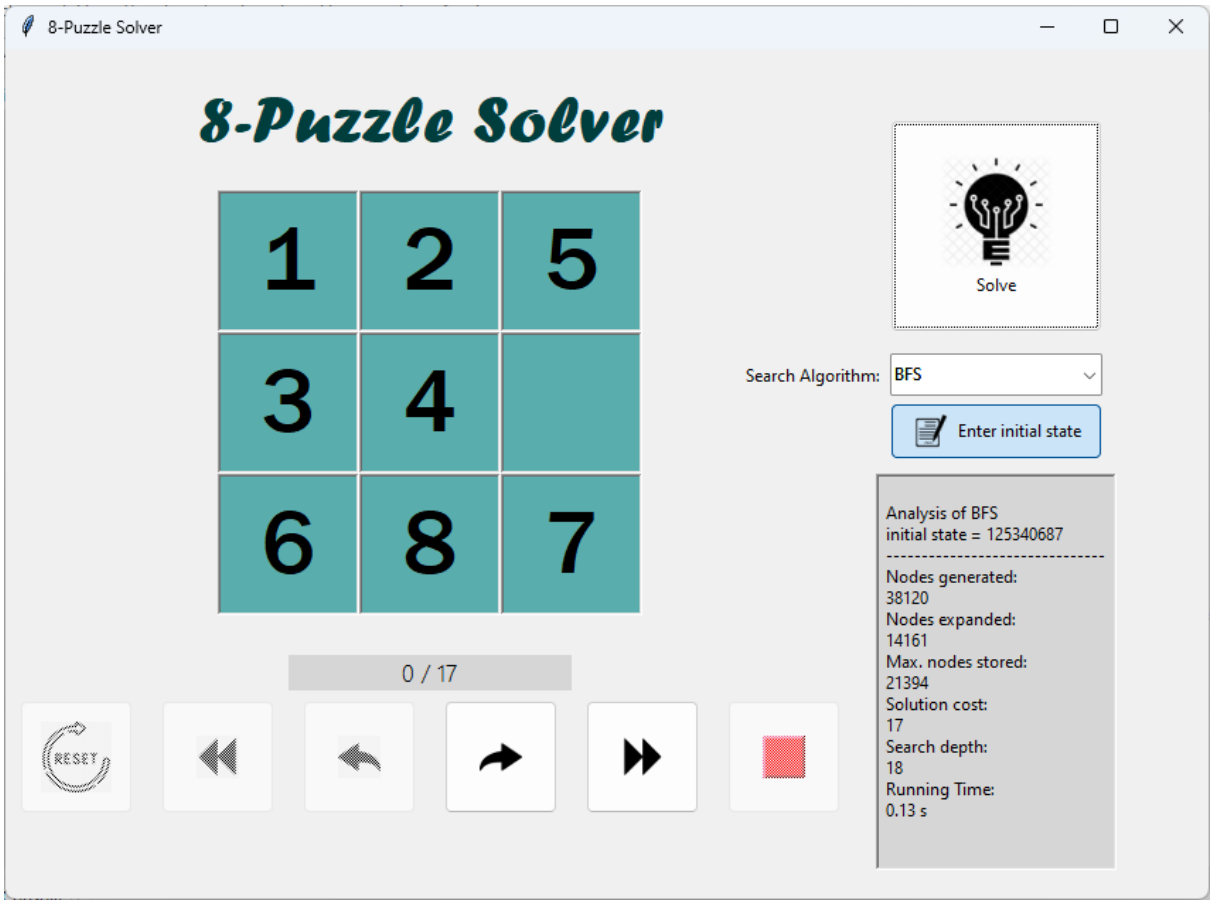


Figura 1 Interfície principal del programa del puzle.

4. Prem en el botó '**Enter initial state'** i insereix el fet inicial com una seqüència de números on la casella blanca es representa com un 0. L'estat que apareix en la figura 1 s'introduiria com 125340687.

[1] Ampliació del programa del mateix nom desenvolupat per Adham Mohamed Alya et al.

5. Selecciona l'estratègia de resolució desitjada en la pestanya desplegable '**Search Algorithm'**.
6. Si se selecciona una estratègia que requereix un nivell de tall de profunditat s'obrirà una finestra on es podrà introduir aquest valor.
7. Prem el botó '**Solve'.**
8. Es mostraran les dades de l'execució del procés de cerca realitzat, i, a més, es pot veure el camí de la solució prement sobre els botons davall de la finestra del puzle.

Per a poder fer una comparativa entre les diferents estratègies de cerca, emprarem sempre el mateix estat objectiu.

1	2	3
8		4
7	6	5

Nota 1: Si en intentar resoldre una configuració del puzle apareix el missatge '**Cannot solve'**, això indica que la configuració introduïda no es pot resoldre mitjançant una combinació de moviments de les peces del puzle en les quatre direccions permeses: a dalt, a baix, dreta i esquerra.

Nota 2: En les estratègies que requereixen la introducció d'un nivell màxim de profunditat, s'obtindrà el missatge '**The state you entered is unsolvable'** si el nivell introduït és inferior al nivell de la solució òptima.

2. Aspectes generals del joc del puzle

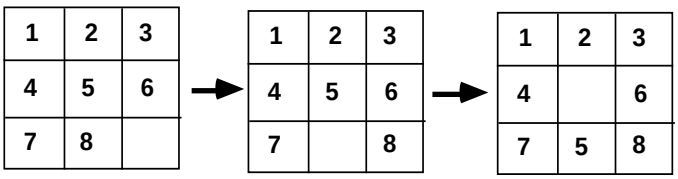
En aquest apartat es resumeixen breument els aspectes més importants del joc del puzle.

2.1. Representació

El joc del puzle es representa sobre un tauler de 3*3 caselles. 8 de les caselles contenen una peça o fitxa que es pot lliscar al llarg del tauler horitzontal i verticalment. Les fitxes venen marcades amb els números de l'1 al 8. Hi ha una casella lliure en el tauler que permet els moviments de les fitxes.

El nombre total d'estats o configuracions del problema que es pot generar en el joc del puzle és 9! = 362.880 estats.

Exemples de moviments:



L'**objectiu** del problema és obtenir, com a solució, la seqüència de 'moviments del quadre buit' que transforma l'estat inicial a l'estat objectiu:

1	2	3
8		4
7	6	5

2.2. Operadors

Els operadors del joc del puzle s'estableixen com els moviments del quadre buit. Aquests moviments són: a dalt, a baix, dreta, esquerra. Note's que, només quan el quadre buit està en la casella central, es podran realitzar els 4 moviments. Si el quadre buit està en una cantonada llavors el nombre de moviments vàlids és 2; en qualsevol altre cas existeixen 3 possibles moviments per al quadre buit.

3. Estratègies de cerca.

Les estratègies de cerca que ofereix el programa *puzle* són:

BFS. Aquesta és l'estratègia d'Amplària que utilitza $f(n)=g(n)$ per a l'expansió de nodes, on $g(n)$ és el factor cost associat al nivell de profunditat del node n en l'arbre de cerca.

DFS (Graph Search). Aquesta estratègia implementa una cerca en ***Profunditat*** GRAPH-SEARCH. Per a això, utilitza la funció $f(n)=-g(n)$, on $g(n)$ és el factor cost associat al nivell de profunditat del node n en l'arbre de cerca.

DFS (Bactracking). Aquesta estratègia implementa una cerca en ***Profunditat*** amb backtracking o ***Profunditat*** TREE-SEARCH. Per a això, utilitza la funció $f(n)=-g(n)$, on $g(n)$ és el factor cost associat al nivell de profunditat del node n en l'arbre de cerca. A diferència de DFS (Graph Search), aquesta estratègia només manté en memòria els nodes explorats que pertanyen al camí actual (llista PATH), la resta de nodes explorats no es mantenen en la llista CLOSED sinó que s'eliminen.

Voraç (Manhattan). Aquesta és una estratègia que implementa un *Algorisme voraç* seguint la funció $f(n)=D(n)$, on $D(n)$ és la distància de Manhattan.

ID. Aquesta estratègia implementa una cerca per ***Aprofundiment Iteratiu*** (Iterative Deepening). Es tracta de realitzar successives cerques en profunditat amb backtracking fins que s'aconsegueix la solució. Cada nova cerca incrementa el nivell de profunditat de l'exploració en 1.

A* Manhattan. Aquesta és una cerca de tipus A que utilitza la distància de Manhattan com a heurística; l'expansió dels nodes es realitza seguint la funció $f(n)=g(n)+D(n)$, on $g(n)$ és el factor cost corresponent a la profunditat del node n en l'arbre de cerca i $h(n)=D(n)$ és la distància de Manhattan, és a dir, la suma de les distàncies en horitzontal i vertical de cada fitxa del puzle des de la seua posició actual a la posició que ocuparà en l'objectiu.

A* Euclidean: Aquesta és una cerca de tipus A que utilitza la distància Euclidiana com a funció heurística; l'expansió dels nodes es realitza seguint la funció $f(n)=g(n)+D(n)$, on $g(n)$ és el factor cost corresponent a la profunditat del node n en l'arbre de cerca i $h(n)=D(n)$ és la distància Euclidiana, és a dir, la suma de les distàncies en línia recta de cada fitxa del puzle des de la seua posició actual a la posició que ocuparà en l'objectiu (arrel quadrada de la suma dels quadrats de les distàncies horitzontal i vertical).

IDA* Manhattan. Aquesta estratègia de cerca implementa una cerca IDA* on el límit de la cerca ve determinat pel valor-f, és a dir, per la funció $f(n)=g(n)+D(n)$, on $g(n)$ és el factor de cost i $h(n)=D(n)$ és la distància de Manhattan. El límit d'una iteració i de l'algorisme IDA* és el valor-f més xicotet de qualsevol node que haja excedit el límit en la iteració anterior $i-1$.

NOTA IMPORTANT:

Les estratègies DFS (Graph Search) i DFS (Backtracking) funcionen amb un nivell màxim de profunditat. Si el nivell màxim actual és menor que el nivell de la solució òptima llavors l'estratègia no trobarà solució.

4. Codi a modificar.

Quan es vol introduir una nova estratègia de cerca, són diverses les zones del codi que has de modificar o afegir.

4.1 Fitxer 'interface.py'

- Afegir el nom de la nova funció de cerca perquè aparega en la llista desplegable d'estratègies de cerca:

Funció `def __init__(self, master=None)`: Línies 88-89 (aprox.):

```
self.algorithmbox.configure(cursor="hand2", state="readonly",
                            values=('BFS',   'DFS   (Graph   Search)',   'DFS
                           (Backtracking)',  'Voraç  (Manhattan)',  'ID',  'A*
                           Manhattan', 'A* Euclidean', 'IDA* Manhattan'))
```

- Si l'estratègia requereix una profunditat màxima, cal afegir el nom de la funció utilitzat en el punt anterior, en la llista que s'utilitza sol·licitar la màxima profunditat:

Funció `selectAlgorithm`, línia 248 (aprox):

```
if algorithm in ['DFS (Graph Search)', 'DFS (Backtracking)']:
 cutDepth = int(simpledialog.askstring('Cut depth', 'Please, enter your max
            depth'))
```

- Realitzar la cridada a la funció de cerca. Qualsevol estratègia de cerca ***sense bactracking*** usa la mateixa cridada `main.graphSearch` però amb diferents paràmetres. La modificació ha de realitzar-se en la funció `solveState` (línia 381, aprox) afegint un nou cas per a la nova estratègia amb la cridada a la funció graphSearch. Exemple de A* Manhattan:

```
elif str(algorithm) == 'A* Manhattan':
     main.graphSearch(initialState,main.function_1,main.getManhattanDistance)
     path, cost, counter, depth, runtime, nodes, max_stored,memory_rep = \
          main.graphf_path, main.graphf_cost, main.graphf_counter,
          main.graphf_depth,main.time_graphf, main.node_counter, main.max_counter,
          main.max_rev_counter
```

La tercera línia és la recol·lecció de resultats i és igual per a totes les heurístiques *sense bactracking*. Els paràmetres que se li passen a la funció `graphSearch` per ordre són:

- Estat inicial del puzle a resoldre. S'emmagatzema en la variable `initialState`.

- Funció que calcula el valor *g*(n). Hi ha tres funcions predefinides que es poden usar:
  - `function_1`. Retorna 1, representa el cost habitual d'un moviment en el puzle
  - `function_0`. Retorna 0, s'usa per a l'estratègia voraç on no es té en compte *g*(n)
  - `function_N`. Retorna -1. Usat en estratègies de profunditat on els nodes més profunds són més prioritaris

- Funció que calcula h(n). Ací es fa la cridada a la funció específica que calcula h(n) segons l'estratègia, com pot ser `getManhattanDistance`, `getEuclideanDistance` o les noves heurístiques que s'implementen.

- Profunditat màxima de l'arbre. És un paràmetre opcional; si s'utilitza, el valor s'emmagatzema en la variable `cutDepth`. Només s'usa per a estratègies que necessiten una profunditat de tall. Si no és així, s'omet.

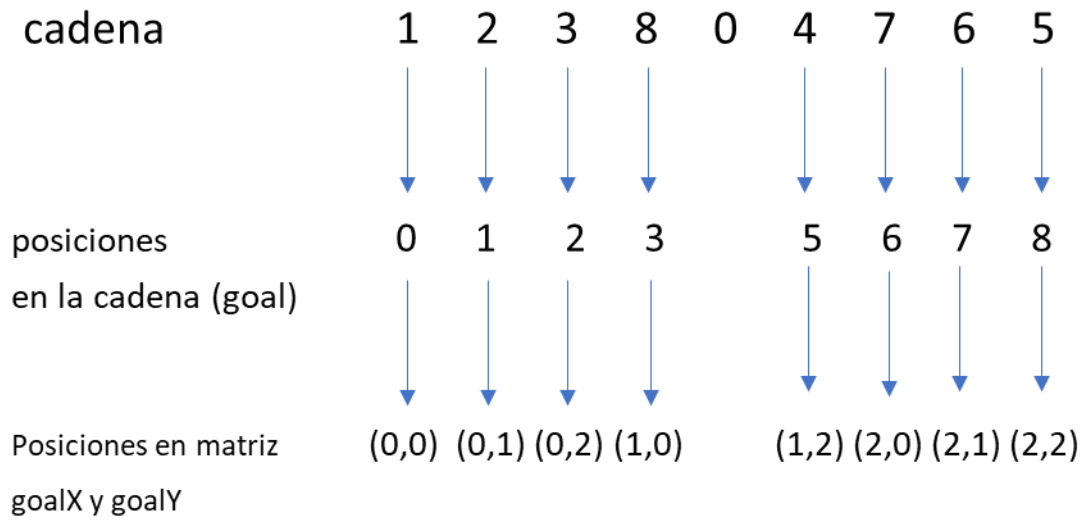
4.2 Fitxer 'main.py'

En aquest fitxer cal incloure una funció que implemente la nova funció heurística. Com a exemple expliquem la funció `getManhattanDistance` ja inclosa en el codi:

```
def getManhattanDistance(state):    #state és l'estat actual a avaluar (en forma de cadena text)
  tot = 0
  for i in range(1,9):              #recorre les 8 peces
    goal = end_state.index(str(i))  #posició de la peça i en l'estat final (end_state)
    goalX = int(goal/ 3)            #passem l'end state de cadena a una matriu de 3x3
    goalY = goal % 3                #fila de la peça (golX) i columna (goalY)
    idx = state.index(str(i))       #mateixa operació però per a la fila (idx) i columna (idy)
    itemX = int(idx / 3)            #que ocupa aqueixa peça en l'estat actual
    itemY = idx % 3
    tot += (abs(goalX - itemX) + abs(goalY - itemY)) #Calcul de distàncies
  return tot
```

La funció rep en `state` l'estat actual (en format cadena) a avaluar. Per a cadascuna de les 8 peces (l'espai en blanc es representa amb un 0 però no és una peça) es calcula la distància des de la posició actual d'aqueixa peça a la seua posició en l'estat final (representat en la variable `end_state`). Les operacions intermèdies són per a per a calcular les posicions de cada peça en una matriu de 3x3 com en el puzle a partir de la representació lineal en una cadena.

Per exemple, per a l'estat final (`end_state`), la transformació de representació lineal (cadena) a matriu seria:



TREBALL A REALITZAR

En aquest apartat s'exposa la tasca que ha de realitzar cada grup de pràctiques (ben individual o un grup de dues persones com a màxim) sobre el programa del 8-puzle.

Es demana implementar tres funcions heurístiques i respondre a una sèrie de preguntes. El dia de l'examen de pràctiques tots els estudiants han de pujar a la seua tasca:

- El codi Python de cadascuna de les tres funcions heurístiques
- Un fitxer de text amb les respostes a les preguntes formulades

En el cas d'un grup de pràctiques de dues persones, ambdues pujaran els mateixos fitxers a les seues corresponents tasques. **NOTA: Indiqueu clarament els noms de les dues persones en el fitxer de text que conté les respostes a les preguntes formulades.**

1. Implementa la funció heurística **PECES DESCOL·LOCADES** per a una cerca de tipus A.

2. Implementa la funció heurística **SEQÜÈNCIES** (h(n)=*Sec*(n)) que es descriu a continuación per a una cerca de tipus A:

   Seqüències. h(n)= *Sec*(n)= 3*S(n), on S(n) és la seqüència obtinguda recorrent successivament les caselles del puzle, excepte la casella central, i anotant 2 punts en el compte per cada fitxa no seguida per la seua fitxa successora i 0 punts per a les altres; si hi ha una fitxa en el centre, s'anota 1. Exemple:

2	8	3
1	6	4
7		5

Per a la fitxa 2:	Sumar 2 (perquè no va seguida de la seua fitxa successora 3)
Per a la fitxa 8:	Sumar 2 (perquè no va seguida de la seua fitxa successora 1)
Per a la fitxa 3:	Sumar 0 (perquè sí que va seguida de la seua fitxa successora 4)
Per a la fitxa 4:	Sumar 0 (perquè sí que va seguida de la seua fitxa successora 5)
Per a la fitxa 5:	Sumar 2 (perquè no va seguida de la seua fitxa successora 6)
Per a la fitxa 0:	Sumar 2 (se sumarà 2 punts sempre que no estiga en la casella central)
Per a la fitxa 7:	Sumar 2 (perquè no va seguida de la seua fitxa successora 8)
Per a la fitxa 1:	Sumar 0 (perquè va seguida per la seua fitxa successora 2)
Per a la fitxa 6:	Sumar 1 (perquè és la fitxa central).
TOTAL	**11 punts**
Heurística	**h(n) = *Sec*(n) = 3 * S(n) = 33**

3. Implementa la funció heurística **FILES_COLUMNES** (h(n)=Fil*C*ol(n)) per a una cerca de tipus A. Aquesta heurística funciona del següent mode: per a cada fitxa del tauler, si la fitxa no està en la seua fila destí correcta se suma 1 punt; si la fitxa no està en la seua columna destí correcta se suma 1 punt. Per tant, el mínim valor d'aquesta heurística per a una fitxa és 0 (quan la fitxa està col·locada correctament en la seua posició objectiu), i el màxim valor és 2 (quan ni la fila ni la columna de la posició de la fitxa són iguals a valor de la fila i columna de la posició objectiu).

4. Respon a les següents preguntes. Per a això es recomana executar diverses configuracions del puzle i anotar els resultats de totes les estratègies de cerca que es mostren en el programa, a més de les tres noves funcions heurístiques implementades (Descol·locades, Seqüències i FilCol). Com a exemple, es pot executar les següents configuracions proposades (es recomana executar més configuracions per a analitzar detalladament el comportament de les estratègies):

	2	**3**
8	**4**	**5**
7	**1**	**6**

	BFS	**DFS (GS)**	**DFS (Back)**	**Voraz**	**ID**	**A* Manh**	**A* Euclid**	**IDA* Manh**	**Desc**	**Sec**	**FilCol**
Nodes generated											
Nodes expanded											
Max nodes stored											
Solution cost											
Search depth											
Running time											

7	**8**	**1**
4		**6**
2	**3**	**5**

	BFS	DFS (GS)	DFS (Back)	Voraz	ID	A* Manh	A* Euclid	IDA* Manh	Desc	Sec	FilCol
Nodes generated											
Nodes expanded											
Max nodes stored											
Solution cost											
Search depth											
Running time											

4.1. L'estratègia de cerca implementada amb la funció heurística Seqüències, és un algorisme A*? Justifica la resposta.

4.2. L'estratègia de cerca implementada amb la funció heurística FilCol, és un algorisme A*? Justifica la resposta.

4.3. Compara l'estratègia A* Manhattan amb Seqüències i indica quin de les dues estratègies retorna millors solucions (qualitat de la solució i cost de la cerca).

4.4. Compara les estratègies de cerca implementades amb les heurístiques Descol·locades i FilCol i indica quina de les dues estratègies retorna millors solucions (qualitat de la solució i cost de la cerca).