UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Departamento: Lenguajes y Ciencias de la Computación

Área de Conocimiento: Ciencias de la Computación e Inteligencia Artificial

---

SISTEMAS INTELIGENTES

Práctica 1: BÚSQUEDA CON A*

Autor: Arturo Aguilera González (DNI: 44668545S)

Titulación: Doble Grado en Matemáticas e Ingeniería Informática

Málaga, 13 de abril de 2021

Índice

1. Diario de Sesiones

1.1. Sesión 1

El objetivo de esta sesión fue crear y definir la clase Malla.java. Esta clase estará formada por dos funciones principales: el constructor y el método *void ver()*.

Este constructor generará una matriz (nuestra malla) donde alguno de sus atributos (numero de filas, de columnas, de obstaculos y semilla) se definirán pasándolo como argumentos en el constructor creado y otros (posición inicial, posición final y posición de los obstaculos) son generados aleatoriamente con la semilla recibida a través de la clase *Random()*.

Esta última parte considero que ha podido ser la más compleja de esta sesión ya que hemos tenido que informarnos más acerca de la clase *java.util.Random*. Esta clase, se puede invocar a partir de dos constructores. En el primero no hará falta indicarle una semilla ya se inicializará en base al instante de tiempo actual. En cuanto al segundo constructor, debemos de tener cuidado porque puede generar problemas ya que debemos tener en cuenta que si dos instancias Random se crean utilizando la misma semilla, se generarán exactamente la misma secuencia de números.

Finalmente diseñamos también los métodos equals y hashCode y un metodo void ver() que nos permite ver el estado de la malla representado graficamente. Por convenio del grupo decidimos que la matriz sea del tipo de dato *String* ya que con este tipo de dato hemos tenido más facilidad para diferenciar los obstáculos, de los huecos libres, de la posición inicial...

Figura 1: Representación de la malla en la terminal al invocar el metodo *void ver()* una vez hallado el camino óptimo.

1.2. Sesión 2

En esta sesión empezamos a analizar y entender el funcionamiento del algoritmo A*. La primera parte de este entendimiento fue desarrollar los métodos de la clase *EstadoMalla.java* de manera correcta. Para poder entender esta clase, el alumno debe tener conocimientos acerca de interfaces en java, ya que EstadoMalla.java implementa a la ya desarrollada interfaz *Estado.java*.

Para poder realizar este trabajo sin ningún problema de ejecución tuvimos que crear los conocidos Getters de la clase *Malla.java*. Una vez creados no tuvimos mucha dificultad en completar estas funciones, quizás, el más complejo de todos pudo ser calculaSucesores() ya que tuvimos que recordar un poco acerca de la clase *java.util.List* y sus metodos. Aún así no resultó muy complejo recordar su uso.

En el metodo coste(), a pesar de su trivial resolución, el entender porque debía ser así abrió algún que otro debate.

Tuvimos también que utilizar la librería de java *java.lang.Math* para el valor absoluto de la heurística de la clase h().

Nuestro equals se basó simplemente en comparar si ambos estados tenian asociados las mismas posiciones actuales y para el hashCode() pudimos reusar el desarrollado en la clase *Malla.java* y modificarlo sumándolo con el hashCode de la posición actual asociada al EstadoMalla.

Por último desarrollamos el metodo void ver() que tras su ejecución imprime por pantalla la posición actual en la que se encuentra este estado.

1.3. Sesión 3

La sesión 3 se basará en el desarrollo de una clase que represente una lista de nodos abiertos. Para poder realizar un estudio de comparación de metodos usados al final de esta práctica, tanto en la sesión 4 como en esta desarrollaremos de dos maneras distintas una misma clase. Para esto, se utilizará el concepto teórico de clase abstracta (*Abiertos.java*). En este caso desarrollaremos las clases extendidas ***AbiertosL.java*** y ***AbiertosPQ.java***. Ambas clases crearán una lista con los nodos abiertos del problema asociado a la práctica. La diferencia principal es que la clase AbiertosL.java utilizará **listas** para almacenar los nodos abiertos mientras que AbiertosPQ.java usará una cola de prioridad (**PriorityQueue**). Por tanto hemos tenido que asentar conocimientos sobre estos tipos de datos para poder desarrollar el código correctamente.

En cuanto a las operaciones necesitadas, en el caso de *AbiertosPQ.java* la implementación de estas han sido sencillas basándose casi en un par de líneas de código por operación ya que las colas de prioridad traían integradas en sí operaciones similares a las pedidas. Aún así se debe conocer el concepto teórico de *Iterator* ya que se ha implementado la representación de la cola en la terminal utilizando esta herramienta.

Las operaciones en la clase que ha utilizado las listas (*AbiertosL.java*) quizás si han sido un tanto más complejas de desarrollar, aunque la influencia de esta diferencia la veremos en la sesión 5.

1.4. Sesión 4

Como hemos comentado en la sesión 3, en esta sesión dividiremos también una clase común para realizarla a partir de dos estructuras de datos distintas. Este trabajo será la última parte del desarrollo del algoritmo A* y consistirá en desarrollar las clases que extienden a la clase abstracta *Arbol.java*. Esta clase abstracta quiere representar el árbol de búsqueda el cual almacenará los estados solución posibles de cada camino generado. El árbol almacenará los estados a partir de la clase Nodo.

La primera estructura de datos utilizadas será ***List*** donde se guardaran los Nodos por profundidad (la lista guardará los Nodos por orden nivel a nivel, de arriba a abajo). La implementación de las operaciones no han sido realmente costosas pero no hemos podido hacerlas a partir de funciones propias de la clase *List.java* (a diferencia de HashMap como ahora veremos) por tanto serán más complejas y podrán tardar más a la hora de invocarlas (conclusión en Sesión 5).

En la segunda estructura, ***HashMap*** la resolución ha sido mucho más sencilla ya que esta propia clase contenía métodos propios que realizaban las operaciones necesarias. Esta estructura almacena los datos en un mapa formando un par **clave-valor**, en este caso definimos como clave un Estado solución y como valor el Nodo asociado a dicho Estado. Al igual que *AbiertosPQ.java* de la sesión 3, para el metodo ver() se ha utilizado la interfaz ListIterator.

1.5. Sesión 5

1.5.1. Conceptos y aspectos teóricos relevantes

Esta sesión se ha basado en probar la funcionabilidad del algoritmo y analizar experimentalmente a partir de dos implementaciones distintas la eficiencia y complejidad de las clases creados en los hitos 3 y 4.

Antes de generar gráficas, analizar resultados, etc. decidimos crear un main (***IgnoreMain.java***) que probaba el correcto funcionamiento del algoritmo. Para ello creamos un problema inicial con una malla e invocamos a las funciones desarrolladas en los anteriores hitos para que lo resolviera y mostrara por pantalla su solución. Una vez comprobado que todo funcionaba correctamente diseñamos un archivo principal (***Main.java***) que constaba de la declaración **try-catch** donde se invoca al tipo de dato ***PrintWriter***, que sirve para crear y escribir archivos de texto desde java. Dentro del *try* invocaremos dos escritores de texto, pw y pw2, donde cada uno creará un archivo de texto (***resultadosTestA.java*** y ***resultadosTestB.java*** respectivamente) en los que se almacenarán los tiempos de ejecución de ambas implementaciones. La estructura de estos archivos estan hechas para que se puedan pasar directamente a una tabla de *Matlab* o *Excel* (usando como delimitadores los dos puntos).

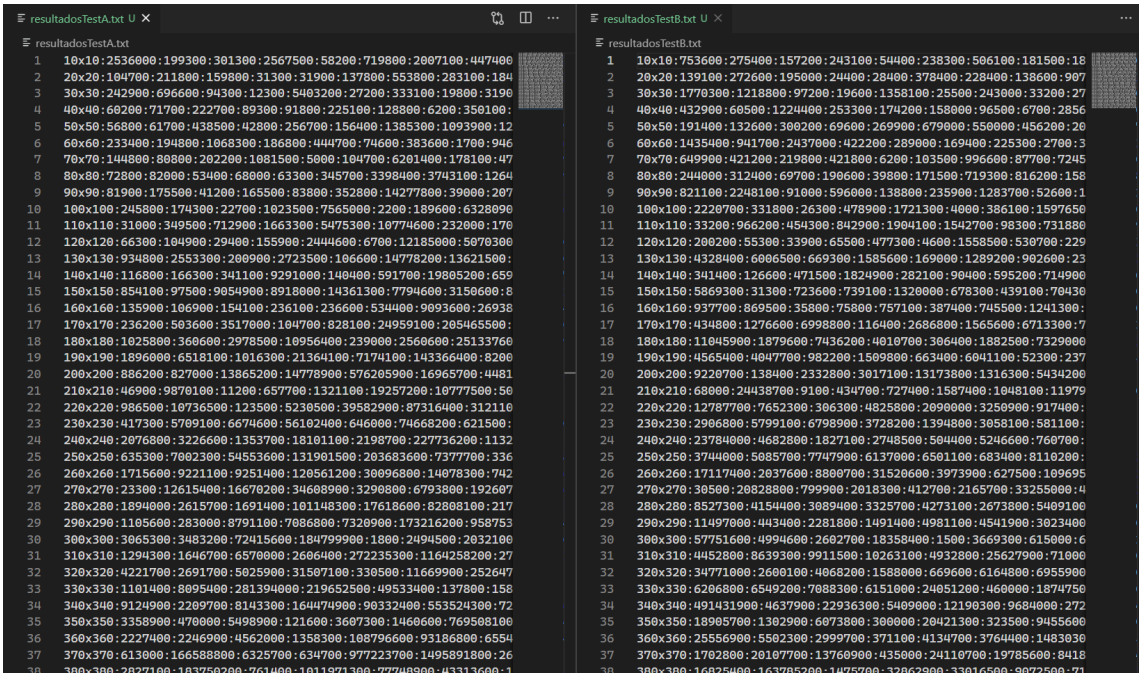

Figura 2: Formato de archivos .txt.

1.5.2. Resumen y valoración de los experimentos realizados

Para poder trabajar con la máxima exactitud llegamos a hacer cerca de 1000 pruebas variando la dimension de la malla (mallas desde 10x10 hasta 400x400) y la densidad de obstáculos en ella (mallas con hasta el 95 % de las casillas ocupadas con obstáculos). Todas estas pruebas se han reflejado en las siguientes tablas.

Resultados A	0	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55
10x10	2536000	199300	301300	2567500	58200	719800	2007100	447400	15800	27700	62000	34200
20x20	104700	211800	159800	31300	31900	137800	553800	283100	1842000	192800	13400	7900
30x30	242900	696600	94300	12300	5403200	27200	333100	19800	31900	10000	25100	18700
40x40	60200	71700	222700	89300	91800	225100	128800	6200	350100	2140800	44100	51700
50x50	56800	61700	438500	42800	256700	156400	1385300	1093900	127400	1493600	94500	107200
60x60	233400	194800	1068300	186800	444700	74600	383600	1700	946400	231600	3200	166800
70x70	144800	80800	202200	1081500	5000	104700	6201400	178100	4704700	5930300	42600	19100
80x80	72800	82000	53400	68000	63300	345700	3398400	3743100	12643600	22900	64400	1400
90x90	81900	175500	41200	165500	83800	352800	14277800	39000	20730800	35500	3200	22100
100x100	245800	174300	22700	1023500	7565000	2200	189600	632809000	288664000	34500	163700	9600
110x110	31000	349500	712900	1663300	5475300	10774600	232000	170959800	10900	437800	714500	5800
120x120	66300	104900	29400	155900	2444600	6700	12185000	5070300	69385900	42800	63300	3000
130x130	934800	2553300	200900	2723500	106600	14778200	13621500	69180000	14800	1692600	12100	900
140x140	116800	166300	341100	9291000	140400	591700	15805200	6596500	82773700	561400	104900	38200
150x150	854100	97500	9054900	8918000	14361300	7794600	3150600	8333300	2586600	1300	38700	500
160x160	135900	106900	154100	236100	236600	534400	9093600	26938100	1454549500	11700	128600	12100
170x170	236200	503600	3517000	104700	828100	24959100	205465500	274329600	51925900	4654900	2000	2000
180x180	1025800	360600	2978500	10956400	239000	2560600	251337600	3400	95335500	1829000	15600	1700
190x190	1896000	6518100	1016300	21364100	7174100	143366400	82000	53900	941455400	128200	800	8400
200x200	886200	827000	13865200	14778900	576205900	16965700	4481900	10556400	26314700	5600	80200	47700
210x210	46900	9870100	11200	657700	1321100	19257200	10777500	506839500	2673400	40510400	1600	1200
220x220	986500	10736500	123500	5230500	39582900	87316400	3121100	385391800	74764100	52502500	6500	900
230x230	417300	5709100	6674600	56102400	646000	74668200	621500	275937000	130930700	25500	8800	16800
240x240	2076800	3226600	1353700	18101100	2198700	227736200	11326300	201569200	8294818200	7981800	15800	1000
250x250	635300	7002300	54553600	131901500	203683600	7377700	336231200	8026800	484700	5000	139600	62700
260x260	1715600	9221100	9251400	120561200	30096800	14078300	742184000	690203000	462625400	9765900	4100	800
270x270	23300	12615400	16670200	34608900	3290800	6793800	1926076200	106112200	61194800	1165700	7900	38500
280x280	1894000	2615700	1691400	101148300	17618600	82808100	217731900	3056812000	3695100	222000	4500	1100
290x290	1105600	283000	8791100	7086800	7320900	173216200	95875300	32224400	14812700	2586600	51600	11400
300x300	3065300	3483200	72415600	184799900	1800	2494500	2032100	3028100	12169400	35200	900	2900
310x310	1294300	1646700	6570000	2606400	272235300	1164258200	274500	3636783600	356419900	8600	2800	1500
320x320	4221700	2691700	5025900	31507100	330500	11669900	252647600	203142700	41725605200	342300	15700	9600
330x330	1101400	8095400	281394000	219652500	49533400	137800	1582954400	532557700	7408441900	12995000	68800	6200
340x340	9124900	2209700	8143300	164474900	90332400	553524300	72888600	6808767000	7300	15800	10800	2500
350x350	3358900	470000	5498900	121600	3607300	1460600	769508100	2722604900	439300	768400	28300	6000
360x360	2227400	2246900	4562000	1358300	108796600	93186800	655493800	746884800	5900	46500	15000	8200
370x370	613000	1665888800	6325700	634700	97723700	1495891800	26534940	2,699E+11	161113400	83300	4000	5800
380x380	2827100	183750200	761400	1011971300	77748900	43313600	13861768500	32662600	3015203100	792100	211700	7000
390x390	2474400	361800	1955400	2827462000	50549300	273160500	389946500	15407584000	27300	115300	34400	70500
400x400	2313800	167100	183300	284398300	13061200	4395200	5447141800	44028082600	1,81587E+11	38200	6600	5099
Media	1287147,5	11163187,5	13160772,5	131996145	64259882,5	114030590	679806602,5	8763406098	6159173175	3737277,5	57907,5	20467,475

Figura 3: Tabla de tiempos utilizando la implementación A.

Resultados B	0	0,05	0,1	0,15	0,2	0,25	0,3	0,35	0,4	0,45	0,5	0,55
10x10	753600	275400	157200	243100	54400	238300	506100	181500	18200	30500	47100	27400
20x20	139100	272600	195000	24400	28400	378400	228400	138600	907900	106300	13600	8000
30x30	1770300	1218800	97200	19600	1358100	25500	243000	33200	27700	10200	25600	24500
40x40	432900	60500	1224400	253300	174200	158000	96500	6700	285600	927800	21800	45800
50x50	191400	132600	300200	69600	269900	679000	550000	456200	201200	757800	79000	87100
60x60	1435400	941700	2437000	422200	289000	169600	225300	2700	367400	134000	3200	145200
70x70	649900	421200	219800	421800	6200	103500	996600	87700	724500	621000	28400	15100
80x80	244000	312400	69700	190600	39800	171500	719300	816200	1587800	17800	37800	1600
90x90	821100	2248100	91000	596000	138800	235900	1283700	52600	1572100	22600	3200	16100
100x100	2220700	331800	26300	478900	1721300	4000	386100	15976500	6819400	9000	53800	6400
110x110	33200	966200	454300	842900	1904100	1542700	98300	7318800	6100	99500	305800	5300
120x120	200200	55300	33900	65500	477300	4600	1558500	530700	2295700	18300	21800	1900
130x130	4328400	6006500	669300	1585600	169000	1289200	902600	2368700	8100	223900	6200	900
140x140	341400	126600	471500	1824900	282100	90400	595200	714900	2659300	89000	30200	28200
150x150	5869300	31300	723600	739100	1320000	678300	439100	704300	300900	1100	16300	400
160x160	937700	869500	35800	75800	757100	387400	745500	1241300	20057900	4300	44900	5700
170x170	434800	1276600	6998800	116400	2686800	1565600	6713300	7812000	2696800	479300	1600	1800
180x180	11045900	1879600	7436200	4010700	306400	1882500	7329000	1200	3168500	208200	8400	1300
190x190	4565400	4047700	982200	1509800	663400	6041100	52300	23700	17865500	51400	1200	5800
200x200	9220700	138400	2332800	3017100	13173800	1316300	5434200	726100	2079700	4200	35400	22300
210x210	68000	24438700	9100	434700	727400	1587400	1048100	11797700	236000	1634600	900	1100
220x220	12787700	7652300	306300	4825800	2090000	3250900	917400	8911800	2758500	2375800	3500	600
230x230	2906800	5799100	6798900	3728200	1394800	3058100	581100	6015600	4406600	11600	5000	8500
240x240	23784000	4682800	1827100	2748500	504400	5246600	760700	5366800	73875800	569800	7800	700
250x250	3744000	5085700	7747900	6137000	6501100	683400	8110200	808500	108900	4600	49300	37700
260x260	17117400	2037600	8800700	31520600	3973900	627500	10969500	12771400	9870500	473500	2100	400
270x270	30500	20828800	799900	2018300	412700	2165700	33255000	4704600	1849200	202400	5000	16400
280x280	8527300	4154400	3089400	3325700	4273100	2673800	5409100	38808100	436900	61400	2600	700
290x290	11497000	443400	2281800	1491400	4981100	4541900	3023400	15376000	916800	259600	22800	7200
300x300	57751600	4994600	2602700	18358400	1500	3669300	615000	694300	828700	17000	800	1900
310x310	4452800	8639300	9911500	10263100	4932800	25627900	71000	35832500	8531900	5000	1400	1200
320x320	34771000	2600100	4068200	1588000	669600	6164800	6955900	5983600	183928000	97700	8500	5500
330x330	6206800	6549200	7088300	6151000	24051200	460000	18747500	9192000	60882700	597100	35900	4400
340x340	491431900	4637900	22936300	5409000	12190300	9684000	2720300	56450800	1300	13200	5500	1900
350x350	18905700	1302900	6073800	300000	20421300	323500	9455600	35464900	116400	126600	11600	4400
360x360	25556900	5502300	2599700	371100	4134700	3764400	14830300	16131500	3800	25800	8300	5300
370x370	1702800	20107700	13760900	435000	24110700	19785600	8418700	806491300	4837000	35400	2500	3900
380x380	16825400	163785200	1475700	32862900	33016500	9072500	71950600	33583800	132300	87000	4400
390x390	6515900	3482500	4364500	33913200	3527800	5188400	7822800	95400300	12800	52000	15100	65900
400x400	14742400	255400	778800	6840700	7194000	12765600	46746700	336723700	454905000	15200	4401	3399
Media	20124032,5	7964817,5	3316942,5	4730747,5	4623225	3432572,5	7037797,5	38253567,5	2264352,5	263170	26632,525	15657,475

Figura 4: Tabla de tiempos utilizando la implementación B.

Una vez pasados los datos a una tabla y a una gráfica veíamos claramente que la primera implementación (implementación con listas) era mucho menos eficiente que la segunda llegando a ir a una magnitud de hasta 100 veces más lento. Para poder ver esta diferencia claramente hemos creado una tabla con la relación entre la primera implementación y la segunda, donde si el coeficiente es mayor que uno significa que la implementacion A tarda más que la B y viceversa.

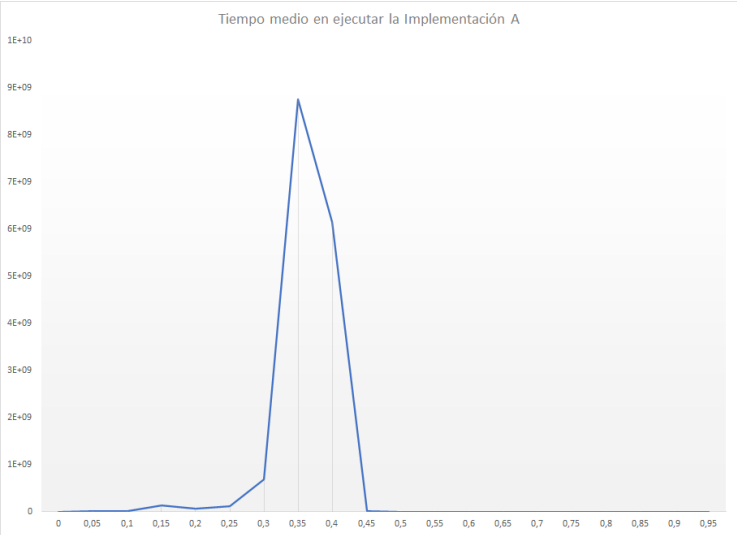

Figura 5: Tiempo medio en ejecutar la Implementación A.

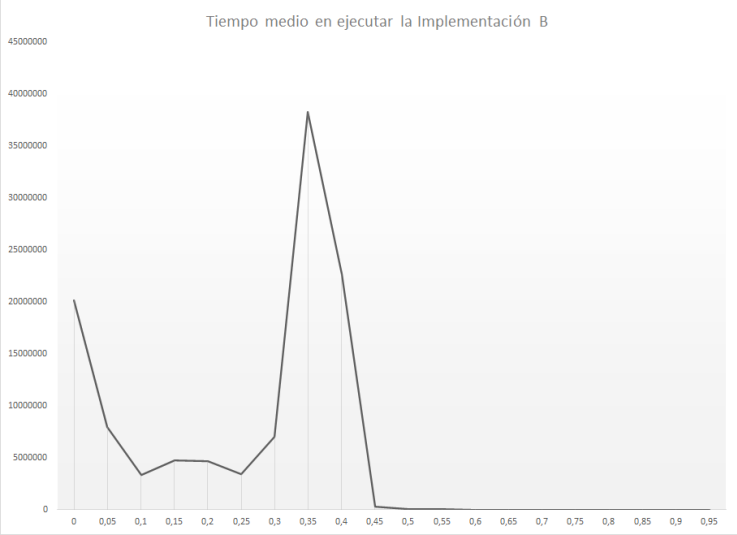

Figura 6: Tiempo medio en ejecutar la Implementación B.

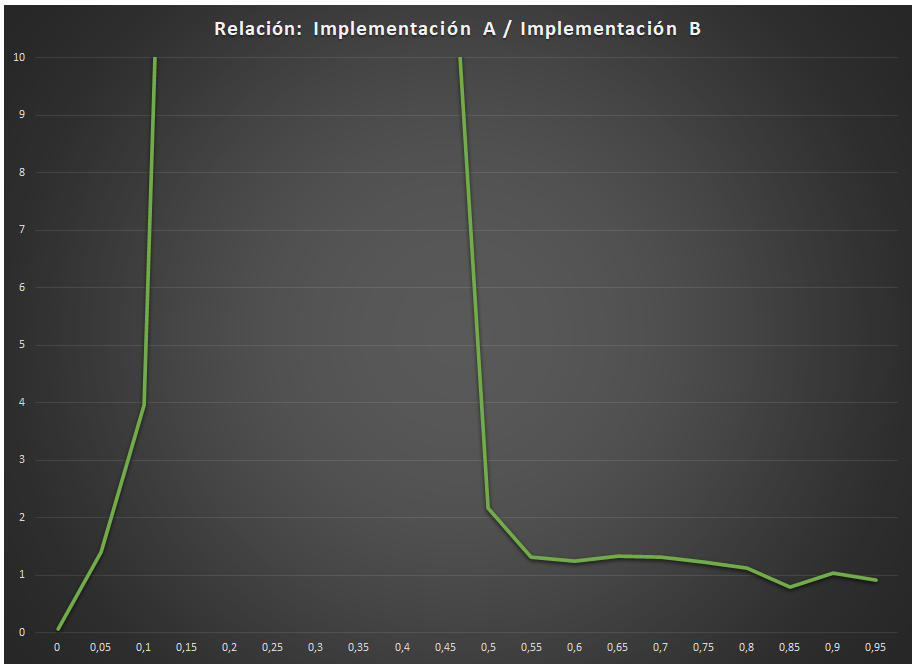

Figura 7: Coeficiente de relación entre las implementaciones.

Podemos, por tanto, concluir que la implementación B es mucho más optima en general que la A, descartando los casos en los que o no hay casi ningún objeto (los posibles caminos son muy elevados por tanto ambas implementaciones tardará mucho) o los casos en los que casi toda la malla está llena de obstaculos (ambas implementaciones tardan poco en descubrir que no hay ninguna solución).

2. Valoración de la práctica

Como valoración personal de la práctica me ha parecido una práctica muy entretenida en la que hemos conseguido solucionar el problema de encontrar un camino en una malla cuadrada con obstáculos. Además debido a su representación gráfica por la terminal, el resultado es muy satisfactorio. En cuanto al grupo hemos sabido trabajar muy coordinadamente y no han habido conflictos.