PROYECTO DE LA ASIGNATURA DE INTELIGENCIA ARTIFICIAL Y SISTEMAS INTELIGENTES 2019/2020

Alumnos:

- Carlos García Garay
- Gabriel Fernando Guiberteau García
- Rafael Canito Rubio

1. Resumen del problema de la práctica:

Se pide diseñar tres, como mínimo, tipos de algoritmos los cuales consigan resolver de forma rápida, aleatoria y automática el típico 14 puzle. Este tipo de puzle consta de una matriz la cual contiene en sus casillas valores de 0 a 14. Si estos valores se encuentran en su posición, es decir, el valor 1 en la posición 1 (0 en la matriz), el valor 2 en su posición, etc.

Los movimientos a realizar solo se pueden hacer respecto al cero, éste es el único valor que se puede mover, por lo que se debe tener en cuenta para la resolución de nuestro problema.

Por ejemplo:

1	2	0	4
5	6	3	7
9	0	11	8
13	10	14	12

Una posible solución:

3N , 7O , 8N , 12N , 10N , 14O

1	2	3	4
5	6	7	8
9	10	11	12
13	14	0	0

La solución nos indica los movimientos que ha debido realizar los diferentes ceros para llegar a la solución.

Estos movimientos se realizarán de forma aleatoria, así como, la selección del cero en cuestión a mover.

2. Descripción de la instalación, interfaz y manejo de la práctica

Las únicas librerías extras usadas, además de las típicas de c como cstdlib, iostream, fstream, han sido:

-Math.h: Librería usada para el cálculo del valor absoluto del valor de la función heurística.

En concreto se ha usado la función “fabs” la cuál devuelve el valor absoluto de una operación matemática.

```
#include <math.h>

        Int dif = fabs(x - i) + fabs(y - j);
```

-time.h: Esta librería la hemos usado para el cálculo del tiempo de ejecución de nuestro programa.

Algunas de las funciones de esta librería son las siguiente:

Functions

Time manipulation

clock	Clock program (function)
difftime	Return difference between two times (function)
mktime	Convert tm structure to time_t (function)
time	Get current time (function)

Conversion

asctime	Convert tm structure to string (function)
ctime	Convert time_t value to string (function)
gmtime	Convert time_t to tm as UTC time (function)
localtime	Convert time_t to tm as local time (function)
strftime	Format time as string (function)

Macro constants

CLOCKS_PER_SEC	Clock ticks per second (macro)
NULL	Null pointer (macro)

types

clock_t	Clock type (type)
size_t	Unsigned integral type (type)
time_t	Time type (type)
struct tm	Time structure (type)

En nuestro caso únicamente hemos debido usar clock_time y time, lo cuales nos consigue calcular el tiempo de ejecución de la siguiente manera:

```
#include <time.h>

        clock_t time = clock();

        ....

                codigo

        .....

        time = (clock() - time)/1000; // para que de en segundos
```

Gracias a esto podemos calcular el tiempo de ejecución de cada uno de nuestros algoritmos.

-listaPI.h: Librería importada del proyecto de Estructura de Datos y la Información, la cual es usada para la creación y utilización de lista con punto de interés.

Este tipo de estructuras lineal nos facilitará el almacenamiento de los distintos hijos de nuestros nodos, lo cual explicaremos más adelante.

Las operaciones más importantes de esta librería son las siguientes:

crear: Crea una lista

insertar: Inserta un elemento en la lista

borrar: Elimina un elemento de la lista

principio: Mueve el puntero al inicio de la lista

avanzar: Avanza el puntero una posición

consultar: Nos devuelve el elemento actual de la lista

finLista: Nos devuelve true si el punto de interés de la lista se encuentra al final y false en caso contrario.

vacía: Devuelve true o false si la lista aparece vacía o llena.

Nuestro programa cuenta con un pequeño menú, el cuál nos pedirá un número del 1 al 10 que corresponde al puzzle a solucionar. Además, nos pide qué algoritmo desea utilizar, siendo un número del 1 al 4 esta vez. Cada número corresponde a uno de los algoritmos, siendo el uno escalada simple, el dos máxima pendiente, el tres primero mejor y el cuatro todos a la vez.

3. Descripción de los algoritmos de resolución usados en la solución

Hemos utilizado los siguientes algoritmos:

• Escalada Simple: Desde un estado, se prueba un nuevo operador y se evalúa con h'. Si es mejor que el estado actual, el nuevo estado será el estado actual. Si no, se prueba un nuevo operador. Si todos los estados nuevos son "peores" que el actual, se para el algoritmo.

• Escalada de Máxima Pendiente: Desde un estado, se prueban todos los operadores y se evalúan con h'. Se selecciona el mejor estado (con h') que se transforma en el estado actual. Si los estados nuevos son "peores" que el actual, se para el algoritmo.

• Búsqueda del Primero Mejor: el nodo posee, a parte de su matriz y su puntero a padre, dos listas de nodos. Una de abiertos (son los nodos aún no expandidos) ordenada de mejor h' a peor y otra de cerrados (nodos ya expandidos). La lista de nodos cerrados se mantiene para no repetir movimientos.

- Generamos todos los hijos del nodo más prometedor (mejor h') y los insertamos en la lista de abiertos de manera ordenada. El nodo expandido se borra de abiertos y se inserta en cerrados. Finalmente se continúa con el que se encuentre en primer lugar de la lista de abiertos hasta llegar al estado objetivo.

Algoritmos descartados:

Hemos utilizado los tres algoritmos anteriores y no otros como Algoritmo A* o Generación y Prueba porque son los que nos parecían más acordes al problema que se nos planteaba. Además, así podemos comparar entre los dos algoritmos de escalada cuál de ellos es más rápido o eficaz a la hora de hallar la solución.

4. Descripción de las soluciones seguidas para resolver el problema

A continuación, se explicarán de forma detallada cada uno de los algoritmos usados, incluyendo las variables y estructuras necesarias para la resolución de los mismos. Primero se explicará el funcionamiento de cada algoritmo y seguidamente el uso y objetivo de cada variable.

Pero antes de nada explicaré la función heurística que hemos empleado para la resolución de este puzzle, la distancia del taxista o distancia Manhattan.

Función heurística para cálculo de h':

Funcionamiento:

Nos devuelve un valor numérico el cuál corresponde a la resta (con valor absoluto) de la posición actual y la posición en la cuál un número debe quedar colocado. La suma sucesiva de esta operación con cada valor de la matriz.

$$\sum_{j=0}^{4}(\sum_{i=0}^{4}(\,|x-i|+|y-j|)\,)$$

Siendo i,j las coordenadas de fila y columna correspondiente a la posición actual de un valor.

Y siendo x, y las coordenadas de la posición el cual ese valor debe quedar colocado al solucionar nuestro puzzle.

Algoritmo de escalada simple:

Funcionamiento:

Tras inicializar las variables (explicadas en Variables y funciones usadas), se copia la matriz con los valores actuales a una matriz auxiliar, llamada mAux.

Seguidamente se comprueba si la h' es mayor que 0 (estado objetivo), entonces se podrá hacer un movimiento.

Mientras no encontremos una solución (hecho de encontrar un movimiento válido y realizarlo), se genera un número aleatorio entre 0 y 3 (ambos inclusive).

En primer lugar, comprobamos en el vector de fail que ese número aleatorio no se ha usado ya en esta iteración, si se ha usado, se actualizará la variable esta colocándola en true. Seguidamente, si ya se ha usado, cont aumenta en 1 y al llegar a 4 (se han dado las cuatro posibilidades de la r) el algoritmo termina diciendo que no tiene solución pues se habrían probado los cuatro movimientos posibles siendo ninguno de ellos válido, ya sea porque no mejore la h' o porque no se pueda realizar.

Si por el contrario no hemos realizado el movimiento, entramos en el if (!esta) y se creará un nuevo nodo hijo, el cuál hará que la variable correspondiente al conteo de número de nodos creados aumente en uno. Aquí, dependiendo de la r y de si se puede realizar el movimiento (que no se salga del tablero, no intercambie ceros o no se haya probado ya y esté en el vector fallidos), realizamos el movimiento (**N,S,E,O**) y si la h' no mejora, borramos el nodo creado para realizarlo, y devolvemos la matriz al estado previo al movimiento. Si, por el contrario, se puede realizar y es factible (la h' mejora) ponemos sol a true para salir del bucle, aumentamos en uno la variable que guarda los movimientos realizados hasta llegar al estado objetivo y mostramos el movimiento. Por último, llamamos al módulo localizarCeros() y, dependiendo de si hay un cero ya colocado o no, llamamos de nuevo al algoritmo de manera recursiva con el nodo hijo, un cero aleatorio o en caso de que uno esté ya colocado, se usará el no colocado, y la variable nMovimientos para ir actualizándola. Esto se realizará recursivamente hasta que encuentra la solución en la que h' es 0, lo que significará que todos los valores están colocados (estado objetivo) o bien hasta que no se pueda realizar ningún otro movimiento al no mejorar la h'.

Variables y funciones usadas:

Int i, j: Valores usados para almacenar la posición i y j (fila y columna) de los ceros correspondientes a nuestra matriz.

Nodo *h: Nodo hijo el cual será creado para almacenar un movimiento en concreto. Este nodo almacenará en su matriz los valores correspondientes al resultado del movimiento realizar, así como un puntero a otro nodo con el movimiento sin realizar. También almacenará un string correspondiente al valor y movimiento realizado.

Int nMovimientos: Variable tipo int usada de entrada/salida que nos indica el número de movimientos realizados.

Int nNodos: Variable tipo int usada de entrada/salida que nos indica el número de nodos creados.

Bool solución: Variable tipo booleana (true o false) la cual comprueba si se puede realizar un movimiento o no.

TipoMatriz mAux: Matriz auxiliar utilizada para copiar la original al realizar un movimiento, no perdiendo así, los valores de la matriz original (sin realizar dicho movimiento).

TipoVector fallidos: vector inicializado a 1, el cual se actualizará a 0 si un movimiento se ha intentado realizar, pero no ha sido satisfactorio. Con esto ganamos que el algoritmo realice menos iteraciones.

TipoVector fail: Evita que se pruebe más de una vez con la misma r. Evitando así iteraciones innecesarias.

Int b: entero usado en el índice del vector anterior.

String movAux: variable que guarda el movimiento realizado.

Función GetHp(): Nos devuelve un valor numérico el cuál corresponde a la resta (con valor absoluto) de la posición actual y la posición en la cual un número debe quedar colocado. La suma sucesiva de esta operación con cada valor de la matriz.

$$\sum_{j=0}^{4}(\sum_{i=0}^{4}(\,|x-i|+|y-j|)\,)$$

Siendo i,j las coordenadas de fila y columna correspondiente a la posición actual de un valor.

Y siendo x, y las coordenadas de la posición el cual ese valor debe quedar colocado al solucionar nuestro puzzle.

Función getPosReal(): Comprueba en qué posición de nuestra matriz se encuentra un valor en concreto, devolviendo la fila y columna donde está.

srand(time(NULL)) : Genera una semilla respecto al tiempo para la generación de número aleatorios.

Int r: Variable tipo int usada para guardar el valor aleatorio generado.

Bool esta: Variable tipo booleana (true o false) iniciada a false, que sirve para comprobar si el r se encuentra en el vector de fail.

Estructura nodo: Estructura usada para almacenar cada uno de los nodos de nuestro árbol.

Puntero padre: Puntero correspondiente a la estructura usada para el padre, en primer lugar se encontrará en nulo.

String Movimiento: String correspondiente al movimiento realizado. Es el conjunto del valor y la dirección al cual se ha movido (**N**orte, **S**ur, **E**ste, **O**este).

lAbiertos: Lista de nodos abiertos, los cuales se podrán seguir expandiendo. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor. Se encuentran ordenados respecto a h' de menor a mayor.

lCerrados: Lista de nodos cerrados, los cuales ya han sido expandidos. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor. No tienen orden concreto.

cambiarCelda(i,j,int): Cambia el valor de la matriz en la posición i,j (fila y columna) con el valor int de entrada.

to_string: Función correspondiente a la librería std la cual convierte un entero a un string. Es usada para la unión del valor+Movimiento dentro de una única variable.

LocalizarCeros(): Función que actualiza el vector con la posición de los ceros en la matriz. Este vector tendrá cuatro posiciones, la cero correspondiente a la fila del primer cero, la primera corresponderá a la columna del primero y de la misma manera con la posición 2 y 3 del vector con los valores correspondientes al segundo cero.

getCeros(i): Esta función nos devuelve el valor de la posición "i" del vector correspondiente al almacenamiento de los ceros.

Algoritmo de escalada, máxima pendiente:

Funcionamiento

A continuación, explicaré paso por paso que realiza nuestro algoritmo de máxima pendiente:

Usaré un pseudocódigo con funciones y variables explicadas más adelante en este documento.

En primer lugar, usa la función localizarCeros() y crea una matriz auxiliar, un nodo *hijo y un string movAux.

El algoritmo elegirá el primero de los ceros, y realizará las siguientes operaciones:

Comprobará que este cero no se encuentra en la pared izquierda (lo cual impediría que dicho valor se moviera hacia el Este) con j > 0, si esto se cumple copiará los valores de la matriz en la matriz auxiliar y realizará el movimiento, almacenándolo a continuación en nuestra variable movAux. Todo esto lo almacenará en la estructura anteriormente creada hijo, creando así una nueva instancia de la misma, gracias a la función h = new nodo (this, movAux) y justamente después cambiando todos los valores de la matriz correspondiente al hijo con la función

h->cambiarCelda(i,j, mAux[i][j]) realizando así una actualización de la matriz correspondiente al hijo con los valores una vez realizado el movimiento. Una vez realizado esto, es insertado en la lista de Hijos.

Esto mismo se repetirá con los distintos movimientos, comprobando que no se encuentre en la

parte superior (i > 0), en la parte derecha (j < 3) o en la parte inferior (i < 3) lo cual, en cualquiera de estos casos, no podría realizarse el movimiento.

Una vez realizado todos los movimientos posibles, se elegirá el segundo cero, realizando de nuevo todas estas operaciones.

Por último, con todos los movimientos realizados, se recorrerá la lista de hijos de este nodo, la cual contiene todos los movimientos posibles como se ha explicado anteriormente, comprobando cuál de todos ellos tiene la h' menor.

Si ninguno de ellos tiene una h' menor, significará que no se ha encontrado una solución, por lo que nuestro algoritmo nos informará de ello y parará la ejecución.

Una vez elegido el que tiene esta h' mejor, se realizará de nuevo todas estas operaciones sobre él, salvo que este contenga de h' = 0.

Si tiene este valor igual a 0, significará que todos los números se encuentran en su posición, por lo que habremos llegado a una solución.

Variables y funciones usadas

Bool solución: Variable tipo booleana (true o false) la cual comprueba si nos encontramos ante la solución o no. Esta es usada al realizar la llamada recursiva, permitiendo realizarla sin problema de que esto provoque un bucle infinito. Al realizar la última llamada (cuando se haya encontrado la solución) esta es llamada de forma true, lo que conlleva a finalizar el programa.

Int nMovimientos: Variable tipo int usada de entrada/salida que nos indica el número de movimientos realizados.

Int nNodos: Variable tipo int usada de entrada/salida que nos indica el número de nodos creados.

LocalizarCeros(): Función que actualiza el vector con la posición de los ceros en la matriz. Este vector tendrá cuatro posiciones, la cero correspondiente a la fila del primer cero, la primera corresponderá a la columna del primero y de la misma manera con la posición 2 y 3 del vector con los valores correspondientes al segundo cero.

TipoMatriz mAux: Matriz auxiliar utilizada para copiar la original al realizar un movimiento, no perdiendo así, los valores de la matriz original (sin realizar dicho movimiento).

String movAux: variable que guarda el movimiento realizado.

Estructura nodo: Estructura usada para almacenar cada uno de los nodos de nuestro árbol.

- Puntero padre: Puntero correspondiente a la estructura usada para el padre, en primer lugar se encontrará en nulo.
- String Movimiento: String correspondiente al movimiento realizado. Es el conjunto del valor y la dirección al cual se ha movido (**N**orte, **S**ur, **E**ste, **O**este).
- lAbiertos: Lista de nodos abiertos, los cuales se podrán seguir expandiendo. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor.
- lCerrados: Lista de nodos cerrados, los cuales se podrán seguir expandiendo. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor.

Int i, j: Valores usados para almacenar la posición i y j (fila y columna) de los ceros correspondientes a nuestra matriz.

to_string: Función correspondiente a la librería std la cual convierte un entero a un string. Es usada para la unión del valor+Movimiento dentro de una única variable.

cambiarCelda(i,j,int): Cambia el valor de la matriz en la posición i,j (fila y columna) con el valor int de entrada.

Estructura listaHijos:

Insertar(): Inserta un elemento en la lista

Consultar(nodo): Devuelve un elemento de la lista

Avanzar(): Avanza una posición

MoverInicio(): Vuelve al inicio de la lista.

FinLista(): Booleano que nos indica si estamos o no al final de la lista.

Int HpMin: Variable usada auxiliarmente para almacenar la h' menor. Esta se va actualizando si encuentra una mejor a ella.

Bool enc: Booleano que nos indica cuándo se ha realizado un movimiento satisfactorio.

Algoritmo de Búsqueda de primero mejor:

Funcionamiento

A continuación, explicaré paso por paso que realiza nuestro algoritmo de Búsqueda de primero mejor:

Usaré un pseudocódigo con funciones y variables explicadas más adelante en este documento.

Todo lo a continuación explicado se realizará siempre y cuando la h' sea mayor que 0.

Lo primero que realiza este algoritmo es localizar los Ceros de la matriz y crear una matriz y un movimiento auxiliar, así como un nodo hijo.

El algoritmo elegirá el primero de los ceros, y realizará las siguientes operaciones:

Comprobará que este cero no se encuentra en la pared izquierda (lo cual impediría que dicho valor se moviera hacia el Este) con j > 0, si esto se cumple copiará los valores de la matriz en la matriz auxiliar y realizará el movimiento, almacenándolo a continuación en nuestra variable movAux.

Todo esto lo almacenará en la estructura anteriormente creada hijo, creando así una nueva instancia de la misma, gracias a la función h = new nodo (this, movAux) y justamente después cambiando todos los valores de la matriz correspondiente al hijo con la función

h->cambiarCelda(i,j, mAux[i][j]) realizando así una actualización de la matriz correspondiente al hijo con los valores una vez realizado el movimiento.

A continuación, se comprueba que este movimiento no se encuentra ni en abiertos ni en cerrados y lo insertas en la lista de abiertos en el lugar que le corresponde.

Esto mismo se repetirá con los distintos movimientos, comprobando que no se encuentre en la parte superior (i > 0), en la parte derecha (j < 3) o en la parte inferior (i < 3) lo cual, en cualquiera de estos casos, no podría realizarse el movimiento.

Una vez realizado todos los movimientos posibles, se elegirá el segundo cero, realizando de nuevo todas estas operaciones.

Al realizar todos los movimientos posibles con ambos ceros, nos dirigiremos al inicio de la lista de nodos Abiertos. Borraremos este de la lista y se comprueba si se encuentra en la lista de nodos Cerrados los cuales son los que ya no podrán expandirse más. Si ya está en la lista de Cerrados, pasaremos al siguiente nodo de la lista de Abiertos. Este último lo insertaremos en la lista de Cerrados y lo expandiremos llamando al algoritmo de forma recursiva.

Una vez expandidos todos los nodos posibles, llegando así a una función heurística de h' = 0, llamaremos a la función solPrimeroMejor.

Esta función es llamada con el último nodo expandido. Recorrerá nuestro árbol de forma recursiva a través del puntero al padre de cada nodo, mostrando los movimientos realizados hasta llegar a la raíz del árbol.

Variables y funciones usadas

Muchas de las variables usadas en este algoritmo aparecen explicadas anteriormente, por lo que me limitaré a explicar las nuevas o usadas exclusivamente en este:

lAbiertos: Lista de nodos abiertos, los cuales se podrán seguir expandiendo. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor. Se encuentran ordenados respecto a h' de menor a mayor.

lCerrados: Lista de nodos cerrados, los cuales ya han sido expandidos. Esto únicamente es usado en el algoritmo correspondiente a búsqueda de primero mejor. No tienen orden concreto.

yaInsertado(): Función booleana que pasandole una lista y un nodo devuelve si el nodo se encuentra ya en esa lista o no.

matricesIguales(): Función booleana que nos devuelve true o false si dos matrices son iguales valor a valor.

4.1 Conclusión sobre la implementación de los diferentes algoritmos

Hemos observado varios fallos y arreglos que se podrían realizar a estos algoritmos, por eso creemos necesario este punto.

Hemos observado que se podrían realizar muchas acciones, como, por ejemplo, la realización de los movimientos en un solo módulo y así no repetir tantas veces lo mismo en los diferentes algoritmos, pero nos surgían problemas al pasar por referencias las diferentes matrices para poder realizarle los cambios. Nos dimos cuenta al finalizar que esto podría solucionarse realizando estas operaciones sobre diferentes nodos, pero por falta de tiempo y miedo a no poder entregarlo a tiempo decidimos dejarlo como hasta ahora.

También nos surgen varios problemas procedentes de la implementación en el primero de los algoritmos, el cuál tarda notablemente más que el resto de ellos. Realizamos muchas trazas con

la herramienta “debug” pero sin llegar a una solución clara. Pensamos en que podría ser cuestión de la aleatoriedad de los movimientos, por lo que colocamos más condiciones para que, por ejemplo, no se pudiera repetir un movimiento si este resulta ser no satisfactorio, pero esto no cambió nada al tiempo de ejecución.

Al realizar trazas no observamos nada que pudiera estar causando este retraso computacional, lo único que se nos ocurrió fue una mala gestión de memoria al realizar la creación y borrado de los distintos nodos.

Una vez comparado con el segundo algoritmo no pudimos saber cuál era el causante de todo esto, y una vez consultado, finalizamos rindiéndonos ante este problema.

También comparamos las operaciones con el segundo algoritmo, el cuál realiza muchas interacciones y accesos a memoria, al crear todos los posibles hijos y no solo uno como hace el primero y no supimos por qué debería “costarle” tanto al primero.

Somos conscientes de este problema.

Las mejoras sugeridas por los profesores de la asignatura serían fáciles de integrar gracias a nuestro diseño del proyecto. Por ejemplo, el aumento de tamaño de la matriz bastaría con cambiar todos los bucles encargados en recorrer la matriz por la variable TAM creada.

Otra mejora fácil de implementar es la sugerida de añadir algún cero más al puzle. Para eso habría que aumentar el tamaño de nuestro vector almacenador de la posición de los diferentes ceros y realizar las llamadas oportunas en los diferentes algoritmos con los siguientes ceros.

5. Resultados obtenidos en las baterías de pruebas para cada una de las soluciones

A continuación, se expondrán las diferentes salidas de los puzles proporcionados por los profesores de la asignatura y en el siguiente apartado se hará una breve descripción de ellos.

Los parámetros de entrada son ficheros contenedores de matrices 4x4 con valores de 1 a 14 y dos ceros. Estos son los valores que el programa deberá colocar en su posición, quedando finalmente los dos ceros en las últimas posiciones de la matriz. Puede que esto no sea posible con alguno de nuestros algoritmos, lo cual se verá reflejado en los resultados.

En cada uno de ellos viene señalado el número de movimientos que ha realizado cada algoritmo, el tiempo de ejecución y el número de nodos generados en memoria.

Puzzle 1

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

3 N, 10 N, 14 O, 7 O, 8 N, 12 N

El número de movimientos ha sido: 6

Ha tardado en ejecutarse: 10.7142 segundos

El número de nodos generados es: 8

Algoritmo de Escalada de Máxima Pendiente:

10 N, 14 O, 3 N, 7 O, 8 N, 12 N

El número de movimientos ha sido: 6

Ha tardado en ejecutarse: 6.5e-05 segundos

El número de nodos generados es: 46

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

3 N, 7 O, 8 N, 10 N, 12 N, 14 O

El número de movimientos ha sido: 6

Ha tardado en ejecutarse: 0.000281 segundos

El número de nodos generados es: 80

Puzzle 2

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

14 O, 12 N

El número de movimientos ha sido: 2

Ha tardado en ejecutarse: 5.84578 segundos

El número de nodos generados es: 5

Algoritmo de Escalada de Máxima Pendiente:

14 O, 12 N

El número de movimientos ha sido: 2

Ha tardado en ejecutarse: 2.8e-05 segundos

El número de nodos generados es: 17

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

12 N, 14 O

El número de movimientos ha sido: 2

Ha tardado en ejecutarse: 4.9e-05 segundos

El número de nodos generados es: 28

Puzzle 3

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

No encuentro solucion

El número de movimientos ha sido: 0

Ha tardado en ejecutarse: 19.855 segundos

El número de nodos generados es: 2

Algoritmo de Escalada de Máxima Pendiente:

13 S, 5 O, 10 N, 9 O

No encuentra solucion

El número de movimientos ha sido: 4

Ha tardado en ejecutarse: 0.000114 segundos

El número de nodos generados es: 27

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

13 S, 5 O, 1 O, 11 O, 12 S, 7 O, 14 O, 3 N, 8 E, 6 E, 5 N, 8 E, 1 O, 10 N, 9 O, 6 E, 14 S, 7 E, 5 N, 1 N, 10 O, 14 S, 7 S, 9 E, 14 S, 10 E, 5 E, 1 N, 7 O, 5 S, 7 S, 5 O, 6 O, 3 S, 2 O, 2 O, 3 N, 11 N, 12 O, 4 N, 8 N, 4 N, 12 E, 11 S, 4 O, 8 S, 12 S, 8 S, 4 E, 4 N, 8 N, 12 N, 11 N, 9 N, 14 E, 10 S, 9 O, 11 S, 6 E, 5 E, 7 N, 9 O, 5 S, 7 E, 9 N, 5 O, 7 S, 6 O, 11 N, 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N, 14 E, 10 S, 5 E, 9 S, 6 O, 5 N, 10 N, 13 E, 9 S, 6 S, 5 O, 10 N, 6 E, 9 N, 13 O, 14 O, 11 S, 6 E, 10 S, 7 O, 6 N, 11 N, 14 E, 10 S, 7 S, 6 O, 11 N, 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N

El número de movimientos ha sido: 118

Ha tardado en ejecutarse: 9.65819 segundos

El número de nodos generados es: 7028

Puzzle 4

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

No encuentro solucion

El número de movimientos ha sido: 0

Ha tardado en ejecutarse: 6.6606 segundos

El número de nodos generados es: 2

Algoritmo de Escalada de Máxima Pendiente:

2 O

No encuentra solucion

El número de movimientos ha sido: 1

Ha tardado en ejecutarse: 3.9e-05 segundos

El número de nodos generados es: 11

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

7 E, 8 N, 8 E, 13 O, 14 O, 3 N, 1 O, 11 N, 4 E, 13 N, 6 N, 5 O, 14 S, 6 E, 5 N, 10 N, 9 O, 14 S, 1 O, 11 O, 12 S, 3 E, 7 S, 8 E, 6 N, 1 N, 10 E, 5 S, 1 O, 6 S, 13 E, 1 N, 5 N, 9 N, 11 S, 7 S, 8 S, 2 O, 3 N, 8 E, 6 E, 13 S, 2 O, 3 O, 9 S, 5 S, 13 O, 6 O, 7 N, 12 O, 4 N, 8 N, 4 N, 12 E, 10 E, 5 E, 13 S, 6 O, 5 N, 10 O, 11 N, 14 E, 9 E, 13 S, 6 S, 5 O, 10 N, 6 E, 5 S, 10 O, 6 N, 9 N, 14 O, 11 S, 7 S, 6 E, 10 E, 5 N, 9 O, 10 S, 6 O, 7 N, 11 N, 12 S, 4 S, 8 S, 11 S, 7 S, 8 O, 4 N, 4 N, 8 E, 7 N, 11 N, 12 N

El número de movimientos ha sido: 95

Ha tardado en ejecutarse: 10.001 segundos

El número de nodos generados es: 6382

Puzzle 5

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

9 N, 13 O, 13 O

No encuentro solucion

El número de movimientos ha sido: 3

Ha tardado en ejecutarse: 25.1723 segundos

El número de nodos generados es: 10

Algoritmo de Escalada de Máxima Pendiente:

13 O, 11 O, 9 N, 13 O

No encuentra solucion

El número de movimientos ha sido: 4

Ha tardado en ejecutarse: 0.000133 segundos

El número de nodos generados es: 36

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

9 N, 13 O, 13 O, 14 S, 14 O, 11 O, 11 N

El número de movimientos ha sido: 7

Ha tardado en ejecutarse: 0.000264 segundos

El número de nodos generados es: 65

Puzzle 6

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

13 O, 1 O, 2 O

No encuentro solucion

El número de movimientos ha sido: 3

Ha tardado en ejecutarse: 3.85779 segundos

El número de nodos generados es: 5

Algoritmo de Escalada de Máxima Pendiente:

7 N, 6 N, 13 O, 1 O, 2 O

No encuentra solucion

El número de movimientos ha sido: 5

Ha tardado en ejecutarse: 0.000145 segundos

El número de nodos generados es: 33

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos…

13 O, 1 O, 2 O, 13 S, 1 O, 2 O, 9 N, 4 N, 10 N, 11 E, 14 S, 10 O, 11 N, 3 O, 9 E, 4 N, 5 O, 9 S, 4 E, 5 N, 8 E, 13 E, 7 N, 6 N, 12 S, 9 S, 8 E, 5 S, 6 S, 7 S, 13 O, 5 O, 11 N, 3 N, 11 N, 3 N, 9 O, 12 N, 14 E, 6 E, 7 S, 13 S, 5 O, 10 N, 6 N, 7 E, 13 S, 6 O, 7 N, 14 O, 9 S, 7 E, 10 S, 3 O, 7 N, 9 N, 14 E, 10 S, 9 O, 7 S, 3 E, 5 E, 6 N, 9 O, 10 N, 14 O, 12 S, 8 S, 3 E, 11 S, 4 O, 3 N, 8 N, 12 N, 7 S, 11 S, 4 S, 3 O, 12 S, 8 S, 4 E, 4 N, 8 N, 11 E, 7 N, 7 N, 11 O, 12 N, 14 E, 13 E, 9 S, 6 S, 5 O, 10 N, 6 E, 9 N, 13 O, 14 O, 11 S, 6 E, 10 S, 7 O, 6 N, 11 N, 14 E, 10 S, 7 S, 6 O, 11 N, 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N

El número de movimientos ha sido: 120

Ha tardado en ejecutarse: 5.84107 segundos

El número de nodos generados es: 6133

Puzzle 7

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

2 N, 7 N, 10 N, 13 O, 1 N, 5 N, 9 N, 13 O

No encuentro solucion

El número de movimientos ha sido: 8

Ha tardado en ejecutarse: 16.2498 segundos

El número de nodos generados es: 13

Algoritmo de Escalada de Máxima Pendiente:

2 N, 7 N, 10 N, 13 O, 1 N, 5 N, 9 N, 13 O

No encuentra solucion

El número de movimientos ha sido: 9

Ha tardado en ejecutarse: 0.000159 segundos

El número de nodos generados es: 63

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

1 N, 2 N, 5 N, 14 O, 6 N, 9 N, 7 E, 14 S, 6 O, 7 N, 14 E, 10 N, 13 O, 13 O, 14 S, 14 O, 11 O, 11 N

El número de movimientos ha sido: 18

Ha tardado en ejecutarse: 0.000882 segundos

El número de nodos generados es: 142

Puzzle 8

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

5 N, 10 N, 13 O, 9 N, 13 O

No encuentro solucion

El número de movimientos ha sido: 5

Ha tardado en ejecutarse: 19.9026 segundos

El número de nodos generados es: 12

Algoritmo de Escalada de Máxima Pendiente:

10 N, 13 O, 11 O, 5 N, 9 N, 13 O

No encuentra solucion

El número de movimientos ha sido: 6

Ha tardado en ejecutarse: 0.000114 segundos

El número de nodos generados es: 50

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos…

5 N, 9 N, 6 O, 14 S, 7 E, 6 N, 10 N, 13 O, 13 O, 14 S, 14 O, 11 O, 11 N

El número de movimientos ha sido: 13

Ha tardado en ejecutarse: 0.000659 segundos

El número de nodos generados es: 114

Puzzle 8

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

1 O, 2 O

No encuentro solucion

El número de movimientos ha sido: 2

Ha tardado en ejecutarse: 4.19855 segundos

El número de nodos generados es: 5

Algoritmo de Escalada de Máxima Pendiente:

11 N, 10 O, 3 O, 1 O, 2 O

No encuentra solucion

El número de movimientos ha sido: 5

Ha tardado en ejecutarse: 0.000116 segundos

El número de nodos generados es: 38

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

1 O, 2 O, 14 S, 8 E, 13 S, 1 O, 2 O, 9 N, 4 N, 10 N, 11 E, 14 S, 10 O, 11 N, 3 O, 9 E, 4 N, 5 O, 9 S, 4 E, 5 N, 8 E, 13 E, 7 N, 6 N, 12 S, 9 S, 8 E, 5 S, 6 S, 7 S, 13 O, 5 O, 11 N, 3 N, 11 N, 3 N, 9 O, 12 N, 14 E, 6 E, 7 S, 13 S, 5 O, 10 N, 6 N, 7 E, 13 S, 6 O, 7 N, 14 O, 9 S, 7 E, 10 S, 3 O, 7 N, 9 N, 14 E, 10 S, 9 O, 7 S, 3 E, 5 E, 6 N, 9 O, 10 N, 14 O, 12 S, 8 S, 3 E, 11 S, 4 O, 3 N, 8 N,

12 N, 7 S, 11 S, 4 S, 3 O, 12 S, 8 S, 4 E, 4 N, 8 N, 11 E, 7 N, 7 N, 11 O, 12 N, 14 E, 13 E, 9 S, 6 S, 5 O, 10 N, 6 E, 9 N, 13 O, 14 O, 11 S, 6 E, 10 S, 7 O, 6 N, 11 N, 14 E, 10 S, 7 S, 6 O, 11 N, 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N

El número de movimientos ha sido: 121

Ha tardado en ejecutarse: 6.06801 segundos

El número de nodos generados es: 6154

Puzzle 8

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

5 N, 10 N, 13 O, 9 N, 13 O

No encuentro solucion

El número de movimientos ha sido: 5

Ha tardado en ejecutarse: 19.9026 segundos

El número de nodos generados es: 12

Algoritmo de Escalada de Máxima Pendiente:

10 N, 13 O, 11 O, 5 N, 9 N, 13 O

No encuentra solucion

El número de movimientos ha sido: 6

Ha tardado en ejecutarse: 0.000114 segundos

El número de nodos generados es: 50

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

5 N, 9 N, 6 O, 14 S, 7 E, 6 N, 10 N, 13 O, 13 O, 14 S, 14 O, 11 O, 11 N

El número de movimientos ha sido: 12

Ha tardado en ejecutarse: 0.000659 segundos

El número de nodos generados es: 114

Puzzle 9

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

1 O, 2 O

No encuentro solucion

El número de movimientos ha sido: 2

Ha tardado en ejecutarse: 4.19855 segundos

El número de nodos generados es: 5

Algoritmo de Escalada de Máxima Pendiente:

11 N, 10 O, 3 O, 1 O, 2 O

No encuentra solucion

El número de movimientos ha sido: 5

Ha tardado en ejecutarse: 0.000116 segundos

El número de nodos generados es: 38

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

1 O, 2 O, 14 S, 8 E, 13 S, 1 O, 2 O, 9 N, 4 N, 10 N, 11 E, 14 S, 10 O, 11 N, 3 O, 9 E, 4 N, 5 O, 9 S, 4 E, 5 N, 8 E, 13 E, 7 N, 6 N, 12 S, 9 S, 8 E, 5 S, 6 S, 7 S, 13 O, 5 O, 11 N, 3 N, 11 N, 3 N, 9 O, 12 N, 14 E, 6 E, 7 S, 13 S, 5 O, 10 N, 6 N, 7 E, 13 S, 6 O, 7 N, 14 O, 9 S, 7 E, 10 S, 3 O, 7 N, 9 N, 14 E, 10 S, 9 O, 7 S, 3 E, 5 E, 6 N, 9 O, 10 N, 14 O, 12 S, 8 S, 3 E, 11 S, 4 O, 3 N, 8 N, 12 N, 7 S, 11 S, 4 S, 3 O, 12 S, 8 S, 4 E, 4 N, 8 N, 11 E, 7 N, 7 N, 11 O, 12 N, 14 E, 13 E, 9 S, 6 S, 5 O, 10 N, 6 E, 9 N, 13 O, 14 O, 11 S, 6 E, 10 S, 7 O, 6 N, 11 N, 14 E, 10 S, 7 S, 6 O, 11 ,N 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N

El número de movimientos ha sido: 121

Ha tardado en ejecutarse: 6.06801 segundos

El número de nodos generados es: 6154

Puzzle 9

Algoritmo de Escalada Simple:

Calculando, esto puede tardar unos segundos...

14 S, 2 O, 1 O, 9 O

No encuentro solucion

El número de movimientos ha sido: 4

Ha tardado en ejecutarse: 6.55923 segundos

El número de nodos generados es: 7

Algoritmo de Escalada de Máxima Pendiente:

11 N, 10 O, 3 O, 1 O, 2 N, 4 N, 3 N

No encuentra solucion

El número de movimientos ha sido: 7

Ha tardado en ejecutarse: 0.000188 segundos

El número de nodos generados es: 53

Algoritmo de Primero Mejor:

Calculando, esto puede tardar unos segundos...

8 E, 14 S, 8 S, 1 O, 1 O, 9 O, 9 O, 2 N, 8 E, 9 S, 2 O, 8 N, 8 E, 4 N, 4 N, 5 O, 5 N10 N, 12 S, 8 S, 4 E, 11 E, 14 S, 6 E, 13 S, 9 O, 6 N, 10 O, 11 N, 14 E, 7 E, 13 S, 10 O, 7 N, 14 O, 11 S, 7 E, 10 E, 9 S, 6 O, 5 O, 7 N, 11 N, 3 O, 12 S, 11 E, 3 N, 7 N, 3 N, 11 O, 12 N, 11 S, 3 S, 7 S, 12 S, 8 S, 7 E, 3 N, 3 N, 7 O, 8 N, 11 N, 12 N, 14 E, 13 E, 9 S, 6 S, 5 O, 10 N, 6 E, 9 N, 13 O, 14 O, 11 S, 6 E, 10 S, 7 O, 6 N, 11 N, 14 E, 10 S, 7 S, 6 O, 11 N, 7 E, 10 N, 14 O, 7 S, 11 S, 12 S, 11 E, 7 N, 7 N, 11 O, 12 N

El número de movimientos ha sido: 95

Ha tardado en ejecutarse: 7.16256 segundos

El número de nodos generados es: 6221

6. Conclusiones sobre los resultados para cada una de las técnicas de resolución empleadas.

En este apartado analizaremos la información dada por los resultados.

En algunos puzles, aparece tanto en el primero como en el segundo algoritmo que no encuentra solución. Esto es porque ninguno de los movimientos posibles mejora la función heurística h', por lo que el algoritmo para.

Esto pasa solo en estos dos porque son algoritmos que usan una estrategia de búsqueda de la solución de tipo "escalada" la cual puede producir este resultado y cambiar el mismo según la ejecución. Es decir, cada vez que lo ejecutemos nos dará un resultado distinto.

También observamos que el último algoritmo, el que usa la estrategia de Primero Mejor, siempre encuentra la solución. Este algoritmo puede ser realmente más lento que los otros dos, ya que explora todas las posibilidades hasta encontrar una solución, por ello también genera muchos más nodos que los otros dos.

7. Referencias.

Búsqueda heurística:

https://www.nebrija.es/~cmalagon/ia/transparencias/busqueda_heuristica.pdf

https://www.gestiopolis.com/busqueda-funciones-evaluacion-heuristica/

Varias páginas complementarias a los apuntes de la asignatura del campus. Las hemos usado para terminar de entender y profundizar en todo lo relacionado con la búsqueda heurística.

Librería math.h:

https://es.wikipedia.org/wiki/Math.h

Se ha usado para realizar el valor absoluto en el cálculo de la función heurística (h').

Librería time.h:

http://www.cplusplus.com/reference/ctime/

https://www.lawebdelprogramador.com/foros/Dev-C/707658-medir-tiempo-de-ejecucion.html

https://mascandobits.es/programacion/medir-tiempo-de-ejecucion-en-c/

https://davidcapello.com/blog/cpp/medir-el-tiempo-de-una-rutina/

Función heurística

https://es.wikipedia.org/wiki/Geometr%C3%ADa_del_taxista

Todo esto ha ido siempre acompañado y contrastado con los apuntes del campus virtual, suministrados por los profesores de la asignatura, así como el uso del foro de la asignatura.