INTELIGENCIA ARTIFICIAL
Algoritmos de búsqueda
Búsqueda A*

Juan Siverio Rojas

Objetivo Práctica 1:
Proponer, implementar y evaluar búsquedas A* para encontrar el camino mínimo entre dos vértices de un grafo.

Lenguaje Utilizado:
C++ versión 11, bajo el entorno de QtCreator 5.0.

Compilación:
Se ha compilado el programa tanto para versiones Linux 64 bits(concretamente se ha probado en Ubuntu) como para versiones Windows a 64 bits. La compilación para Windows ha sido mediante Mingwx64 integrado ya en QtCreator 5.0

Versiones: He implementado dos versiones del algoritmo.

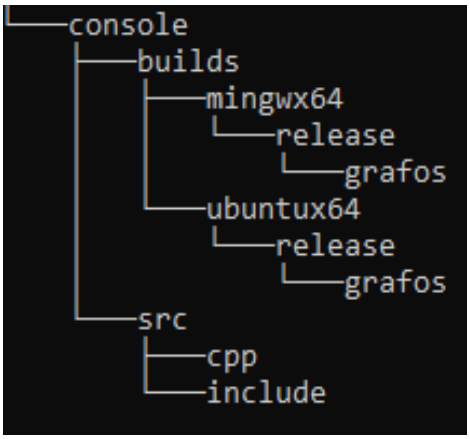
1. Version 1(Recomendada): Utilizo solo un contenedor SET, en el que se incluyen todos los nodos tanto generados como inspeccionados, diferenciados por un atributo boleano llamado estudiado_(dentro de NodeIA_t). En esta versión es capaz de reconocer si el nodo a generar ya existe previamente en su camino hacia el nodo raíz, si no es así, lo genera, aunque existiera en otro camino del árbol. Se consigue con puntero dentro del nodo que apunta a su padre. Siempre consigue la solución óptima, con una función heurística admisible.

2. Version 2: Aquí utilizo dos contenedores SET, uno para nodos generados, y otro para nodos inspeccionados. En esta versión, los nodos a generar se comprueba que no existan ni en la lista de nodos generados ni en la de inspeccionados, si es así se genera el nodo, de lo contrario, el nodo no se genera. Una vez inspeccionado un nodo, se extrae de generados y se introduce en inspeccionados. Esta versión consume menos memoria, y es más rápida que la anterior, puesto que nunca generará un mismo nodo dos veces, pero no siempre encuentra la solución óptima, ya que podría estar en un camino que ya no puede generar(por requerir de un nodo ya creado en otro camino), aún así, con una buena función heurística, funciona bastante bien.

Conclusión sobre las 2 versiones:
Como se comprobará con los resultados de diferentes pruebas, la versión 1 te a segura que siempre se obtendrá la solución óptima incluso cuando la heurística no sea la adecuada, pero si admisible, pero cuando se busca una

solución y no importa que NO siempre se consiga la óptima, la versión 2 es la candidata perfecta, puesto que es mucho más rápida y consume mucha menos memoria al generar muchos menos nodos ya que impide que estos se repitan.

Estructura de la práctica:



- ⇨ **Ejecutable versión Windows.**
- ⇨ **Grafos de pruebas**
- ⇨ **Ejecutable versión Linux.**
- ⇨ **Grafos de prueba**
- ⇨ **Código fuente.**
- ⇨ **Ficheros de definiciones**
- ⇨ **Cabeceras/Declaración de clases**

Git:

Se ha utilizado control de versiones GIT. Con “git log“ dentro de la carpeta se puede ver todo el histórico.

Estructuras de Datos:

Clases:

1. **NodeIA_t:**
   Esta clase es la que genera todos los valores necesarios para un nodo que sirve tanto para un nodo del grafo de estados como para un nodo del árbol de búsqueda generado por el algoritmo A*.
2. **EstadoIA_t:**
   Esta clase está incluida como un atributo dentro de la clase NodeIA_t, es la que internamente contiene el estado actual en el que se encuentra el nodo, como en este caso solo estamos utilizando números como identificador del nodo, pues los atributos de esta clase son su ID(numero de nodo) y un string identificativo, pero la clase está pensada con vistas a que ella misma personalize sus propios métodos necesarios para pasar de un estado a otro dependiendo de la acción concreta o del estado en el que se encuentre.
3. **GrafoIA_t:**
   Esta es la clase prinpal, encargada de la carga de ficheros de grafos y de ejecutar el algoritmo A*.

Contenedores:
Los contenedores de datos más importantes son:

1. **Vector <pair<int,double> >LS_ : Incluido en cada NodeIA_t**, es un vector de parejas de nodos sucesores conteniendo numero de nodo y valor de coste para alcanzarlo
2. **Vector<NodeIA_t> grafo_** : **Incluido en GrafoIA_t**, es un vector de nodos creado por la función de lectura de ficheros, cada nodo es personalizado con sus sucesores correspondientes y sus costes. Esto lo realiza el método **bool actualizar(char nombrefichero[])** de la clase
3. **set<NodeIA_t> nodos_: Incluido en GrafoIA_t,** es un conjunto en el que se van ingresando los nodos que va generando el algoritmo a estrella. Estos nodos se ordenan de menor a mayor basándose en el valor del atributo **costoCaminoMasHeuristico_ de la clase NodeIA_t**, este atributo corresponde al valor f(n) del algoritmo.
4. **SOLO VERSION 2 del algoritmo: Set<NodeIA_t> generados_:** conjunto de nodos generados.
5. **SOLO VERSION 2 del algoritmo: set<NodeIA_t> inspeccionados_:** conjunto de nodos inspecionados, este conjunto se rellena eliminando el nodo de los generados_ e introduciéndolo en los inspeccionados.

Funcionamiento:

1. La carga de ficheros de grafos crea la estructura grafo_. Se puede hacer directamente desde el constructor especificando el nombre del fichero a leer, o bien mediante el método de la case GrafoIA_t, **actualizar(nombrefichero[]).**

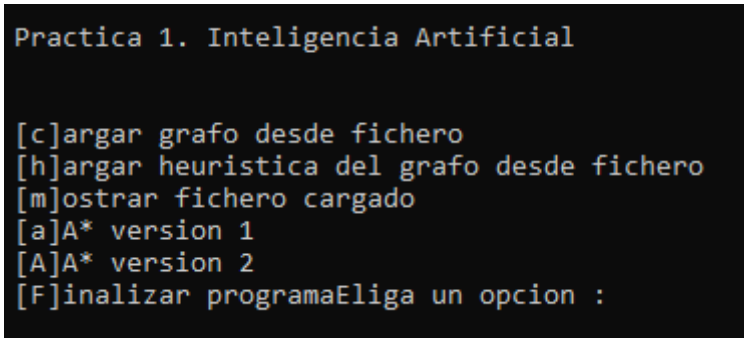
2. (opcional) Ejecutar el método **aplicarHeuristica(char nombrefichero[])** para cargar un fichero de heurística que será aplicado directamente a la estructura grafo_.

3. Ejecutar **aEstrella(int nodoOrigen, int nodoDestino, bool version1)**. Donde version1 será **true** para ejecutar el algoritmo versión 1 y false para el versión 2.

4. Ejecutar **mostrarCaminoSolucion(ostream &os)**. Muestra el resultado en forma de tabla, indicando un ofstream, se puede enviar a un fichero.

Menús:

Se han creado menús para facilitar el uso de la aplicación, basado en los métodos anteriormente descritos.

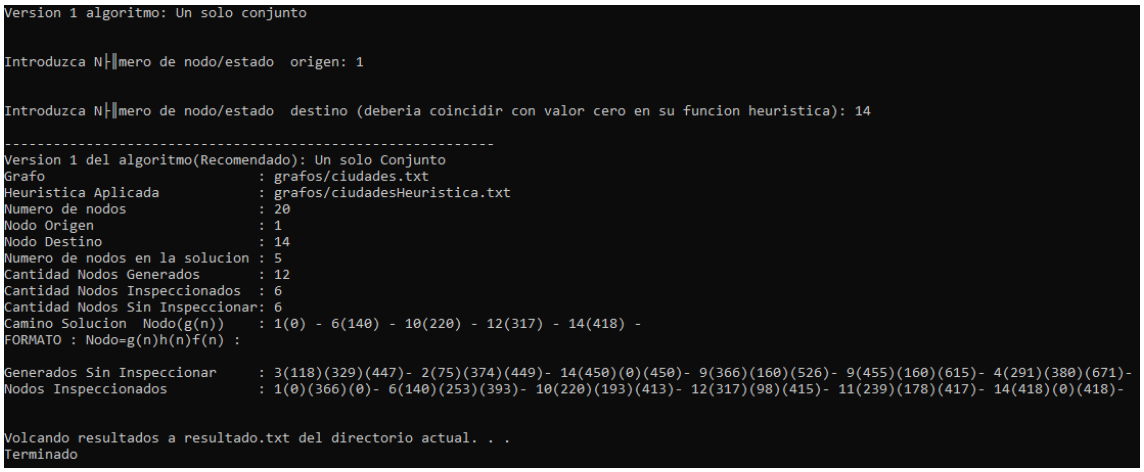


- La opción “mostrar fichero cargado” muestra la estructura del contenedor **grafo_ ,** que contiene la definición del grafo de estado con sus transiciones y costes.
- Las opciones **“A* versión 1” y “A* versión 2”** , permiten a continuación insertar nodo origen y nodo destino, y los resultados del algoritmo los muestran por pantalla y los agregan a un fichero llamado **resultado.txt** en el directorio raíz de ejecución.

Nota1: Los métodos se encargan de todo lo necesario para permitir abrir tantos ficheros de grafos como se quiera y tantos ficheros heurísticos como se quiera sin tener que cerrar y volver a ejecutar la aplicación.

Nota2: Se puede ejecutar un fichero de grafos sin necesidad de aplicar un fichero de heurística, en cuyo caso se supone que la heurística desde cada nodo es la misma para todos que es cero.

<u>Pruebas y resultados obtenidos:</u>



```
Version 2 algoritmo: Dos conjuntos


Introduzca N├╠mero de nodo/estado  origen: 1


Introduzca N├╠mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 14

------------------------------------------------------------
Version 2 del algoritmo: Dos conjuntos, no se repiten nodos
Grafo                          : grafos/ciudades.txt
Heuristica Aplicada            : grafos/ciudadesHeuristica.txt
Numero de nodos                : 20
Nodo Origen                    : 1
Nodo Destino                   : 14
Numero de nodos en la solucion : 5
Cantidad Nodos Generados       : 10
Cantidad Nodos Inspeccionados  : 6
Cantidad Nodos Sin Inspeccionar: 4
Camino Solucion  Nodo(g(n))    : 1(0) - 6(140) - 10(220) - 12(317) - 14(418) -
FORMATO : Nodo=g(n)h(n)f(n) :

Generados Sin Inspeccionar     : 3=(118)(329)(447)- 2=(75)(374)(449)- 9=(366)(160)(526)- 4=(291)(380)(671)-
Nodos Inspeccionados           : 1(0)(366)(0)- 6(140)(253)(393)- 10(220)(193)(413)- 12(317)(98)(415)- 11(239)(178)(417)- 14(418)(0)(418)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

Sobre los dos resultados anteriores: Se puede observar que en el caso anterior, con la versión dos, se consigue la misma solución , que además es óptima pero generando menos nodos, un total de 10 frente a 12.

```
Version 1 algoritmo: Un solo conjunto


Introduzca N├╠mero de nodo/estado  origen: 1


Introduzca N├╠mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 14

------------------------------------------------------------
Version 1 del algoritmo(Recomendado): Un solo Conjunto
Grafo                          : grafos/ciudades.txt
Heuristica Aplicada            :
Numero de nodos                : 20
Nodo Origen                    : 1
Nodo Destino                   : 14
Numero de nodos en la solucion : 5
Cantidad Nodos Generados       : 26
Cantidad Nodos Inspeccionados  : 18
Cantidad Nodos Sin Inspeccionar: 8
Camino Solucion  Nodo(g(n))    : 1(0) - 6(140) - 10(220) - 12(317) - 14(418) -
FORMATO : Nodo=g(n)h(n)f(n) :

Generados Sin Inspeccionar     : 14(450)(0)(450)- 9(455)(0)(455)- 12(474)(0)(474)- 8(486)(0)(486)- 9(494)(0)(494)- 12(504)(0)(504)- 9(523)(0)(523)- 14(607)(0)(607)-
Nodos Inspeccionados           : 1(0)(0)(0)- 2(75)(0)(75)- 3(118)(0)(118)- 6(140)(0)(140)- 4(146)(0)(146)- 10(220)(0)(220)- 5(229)(0)(229)- 11(239)(0)(239)- 4(291)(0
291)- 6(297)(0)(297)- 7(299)(0)(299)- 12(317)(0)(317)- 2(362)(0)(362)- 9(366)(0)(366)- 8(374)(0)(374)- 10(377)(0)(377)- 11(396)(0)(396)- 14(418)(0)(418)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

```
Version 2 algoritmo: Dos conjuntos


Introduzca N├╠mero de nodo/estado  origen: 1


Introduzca N├╠mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 14

------------------------------------------------------------
Version 2 del algoritmo: Dos conjuntos, no se repiten nodos
Grafo                          : grafos/ciudades.txt
Heuristica Aplicada            :
Numero de nodos                : 20
Nodo Origen                    : 1
Nodo Destino                   : 14
Numero de nodos en la solucion : 4
Cantidad Nodos Generados       : 13
Cantidad Nodos Inspeccionados  : 13
Cantidad Nodos Sin Inspeccionar: 0
Camino Solucion  Nodo(g(n))    : 1(0) - 6(140) - 11(239) - 14(450) -
FORMATO : Nodo=g(n)h(n)f(n) :

Generados Sin Inspeccionar     :
Nodos Inspeccionados           : 1(0)(0)(0)- 2(75)(0)(75)- 3(118)(0)(118)- 6(140)(0)(140)- 4(146)(0)(146)- 10(220)(0)(220)- 5(229)(0)(229)- 11(239)(0)(239)- 7(299)(0)(
299)- 12(317)(0)(317)- 9(366)(0)(366)- 8(374)(0)(374)- 14(450)(0)(450)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

Sobre los dos resultados anteriores: En este caso, comprobamos el mismo grafo pero sin ningún fichero de heurística aplicado, la versión 2 sigue consiguiendo una solución en menos nodos que la versión 1, 13 nodos frente a 26 generados, pero no es óptima, la versión 1 si obtiene la versión óptima incluso sin heurística.

```
Version 1 algoritmo: Un solo conjunto


Introduzca N├╢mero de nodo/estado  origen: 1


Introduzca N├╢mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 8

------------------------------------------------------------
Version 1 del algoritmo(Recomendado): Un solo Conjunto
Grafo                          : grafos/Grafo1_1_8.txt
Heuristica Aplicada            : grafos/Grafo1H1.txt
Numero de nodos                : 15
Nodo Origen                    : 1
Nodo Destino                   : 8
Numero de nodos en la solucion : 6
Cantidad Nodos Generados       : 36
Cantidad Nodos Inspeccionados  : 17
Cantidad Nodos Sin Inspeccionar: 19
Camino Solucion  Nodo(g(n))    : 1(0) - 3(10) - 11(19) - 5(20) - 12(25) - 8(36) -


- - -  FORMATO : Nodo=g(n)h(n)f(n) - - -

Generados Sin Inspeccionar     : 3(28)(8.54)(36.54)- 11(28)(8.94)(36.94)- 15(28)(9.43)(37.43)- 12(27)(10.82)(37.82)- 5(27)(11.4)(38.4)- 15(30)(9.43
)(39.43)- 12(29)(10.82)(39.82)- 5(29)(11.4)(40.4)- 3(32)(8.54)(40.54)- 2(33)(8.25)(41.25)- 15(32)(9.43)(41.43)- 3(35)(8.54)(43.54)- 2(36)(8.25)(44.
25)- 15(35)(9.43)(44.43)- 15(37)(9.43)(46.43)- 15(39)(9.43)(48.43)-

Nodos Inspeccionados           : 1(0)(1)(0)- 2(9)(8.25)(17.25)- 3(10)(8.54)(18.54)- 2(11)(8.25)(19.25)- 4(19)(6.32)(25.32)- 4(21)(6.32)(27.32)- 11(
19)(8.94)(27.94)- 4(23)(6.32)(29.32)- 5(20)(11.4)(31.4)- 11(23)(8.94)(31.94)- 4(26)(6.32)(32.32)- 5(22)(11.4)(33.4)- 11(25)(8.94)(33.94)- 11(26)(8.
94)(34.94)- 5(24)(11.4)(35.4)- 12(25)(10.82)(35.82)- 8(36)(0)(36)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

```
Version 2 algoritmo: Dos conjuntos


Introduzca N├╢mero de nodo/estado  origen: 1


Introduzca N├╢mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 8

------------------------------------------------------------
Version 2 del algoritmo: Dos conjuntos, no se repiten nodos
Grafo                          : grafos/Grafo1_1_8.txt
Heuristica Aplicada            : grafos/Grafo1H1.txt
Numero de nodos                : 15
Nodo Origen                    : 1
Nodo Destino                   : 8
Numero de nodos en la solucion : 6
Cantidad Nodos Generados       : 10
Cantidad Nodos Inspeccionados  : 9
Cantidad Nodos Sin Inspeccionar: 1
Camino Solucion  Nodo(g(n))    : 1(0) - 2(9) - 4(19) - 5(22) - 12(27) - 8(38) -


FORMATO : Nodo=g(n)h(n)f(n) :

Generados Sin Inspeccionar     : 6(36)(7)(43)-

Nodos Inspeccionados           : 1(0)(1)(0)- 2(9)(8.25)(17.25)- 3(10)(8.54)(18.54)- 4(19)(6.32)(25.32)- 11(19)(8.94)(27.94)- 5(22)(11.4)(33.4)- 15(
28)(9.43)(37.43)- 12(27)(10.82)(37.82)- 8(38)(0)(38)-



Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

Sobre los dos resultados anteriores: Este es otro grafo en el que la versión 1 obtiene la solución óptima y la versión 2 no, pero la versión dos solo generó 10 nodos frente a 36 de la versión 1.

```
Version 1 algoritmo: Un solo conjunto


Introduzca N├╠mero de nodo/estado  origen: 1


Introduzca N├╠mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 8

------------------------------------------------------------
Version 1 del algoritmo(Recomendado): Un solo Conjunto
Grafo                          : grafos/Grafo1_1_8.txt
Heuristica Aplicada            : grafos/Grafo1H1_admisible_nodo_11.txt
Numero de nodos                : 15
Nodo Origen                    : 1
Nodo Destino                   : 8
Numero de nodos en la solucion : 6
Cantidad Nodos Generados       : 37
Cantidad Nodos Inspeccionados  : 18
Cantidad Nodos Sin Inspeccionar: 19
Camino Solucion  Nodo(g(n))    : 1(0) - 3(10) - 11(19) - 5(20) - 12(25) - 8(36) -


- - -  FORMATO : Nodo=g(n)h(n)f(n) - - -

Generados Sin Inspeccionar     : 3(28)(8.54)(36.54)- 15(28)(9.43)(37.43)- 12(27)(10.82)(37.82)- 5(27)(11.4)(38.4)- 15(30)(9.43)(39.43)- 12(29)(10.82)(39.
82)- 5(29)(11.4)(40.4)- 3(32)(8.54)(40.54)- 2(33)(8.25)(41.25)- 15(32)(9.43)(41.43)- 3(35)(8.54)(43.54)- 2(36)(8.25)(44.25)- 15(35)(9.43)(44.43)- 15(37)(
9.43)(46.43)- 15(39)(9.43)(48.43)-

Nodos Inspeccionados           : 1(0)(1)(0)- 2(9)(8.25)(17.25)- 3(10)(8.54)(18.54)- 2(11)(8.25)(19.25)- 11(19)(6.31)(25.31)- 4(19)(6.32)(25.32)- 4(21)(6.
32)(27.32)- 11(23)(6.31)(29.31)- 4(23)(6.32)(29.32)- 11(25)(6.31)(31.31)- 5(20)(11.4)(31.4)- 11(26)(6.31)(32.31)- 4(26)(6.32)(32.32)- 5(22)(11.4)(33.4)-
11(28)(6.31)(34.31)- 5(24)(11.4)(35.4)- 12(25)(10.82)(35.82)- 8(36)(0)(36)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

```
Version 2 algoritmo: Dos conjuntos


Introduzca N├╠mero de nodo/estado  origen: 1


Introduzca N├╠mero de nodo/estado  destino (deberia coincidir con valor cero en su funcion heuristica): 8

------------------------------------------------------------
Version 2 del algoritmo: Dos conjuntos, no se repiten nodos
Grafo                          : grafos/Grafo1_1_8.txt
Heuristica Aplicada            : grafos/Grafo1H1_admisible_nodo_11.txt
Numero de nodos                : 15
Nodo Origen                    : 1
Nodo Destino                   : 8
Numero de nodos en la solucion : 6
Cantidad Nodos Generados       : 9
Cantidad Nodos Inspeccionados  : 8
Cantidad Nodos Sin Inspeccionar: 1
Camino Solucion  Nodo(g(n))    : 1(0) - 3(10) - 11(19) - 5(20) - 12(25) - 8(36) -


FORMATO : Nodo=g(n)h(n)f(n) :

Generados Sin Inspeccionar     : 15(28)(9.43)(37.43)-

Nodos Inspeccionados           : 1(0)(1)(0)- 2(9)(8.25)(17.25)- 3(10)(8.54)(18.54)- 11(19)(6.31)(25.31)- 4(19)(6.32)(25.32)- 5(20)(11.4)(31.4)- 12(25)(10
.82)(35.82)- 8(36)(0)(36)-


Volcando resultados a resultado.txt del directorio actual. . .
Terminado
```

Sobre los dos resultados anteriores: En esta ocasión, comprobamos el mismo grafo que en el resultado predecesor, pero modificando la heurística solo en el nodo 11 para que sea admisible, y conseguimos con la versión 2 una solución que ahora si es óptima y generando solo 9 nodos, frente a los 37 nodos que genera la versión 1.