
Algoritmo de recorrido mínimo con CLIPS, mediante CLIPSPy
Jesús Roque Armas Martín - CEIABD

Índice de contenido

Introducción

El planteamiento del ejercicio es adaptar el algoritmo de Dijkstra, para encontrar el recorrido mínimo entre dos nodos de un grafo dado, mediante CLIPS, una herramienta de desarrollo para sistemas expertos , embebida en Python, mediante ClipsPy.

Edsger Dijkstra fue un científico informático de los Países Bajos, que desarrolló un algoritmo que lleva su nombre, también llamado algoritmo de caminos mínimos.

Este algoritmo tiene como cometido determinar el camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Fue concebido en 1956, aunque no se publicó hasta 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la búsqueda de costo uniforme y, como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

CLIPS es una herramienta que provee un entorno de desarrollo para la producción y ejecución de sistemas expertos. Fue creado a partir de 1984, en el Lyndon B. Johnson Space Center de la NASA.

Es un acrónimo de C Language Integrated Production System (Sistema de Producción Integrado en Lenguaje C). En la actualidad, entre los paradigmas de programación que soporta CLIPS se encuentran la Programación lógica, la Programación imperativa y la Programación Orientada a Objetos.

Se basa en el uso de hechos y reglas, para representar el conocimiento y un motor de inferencia para evaluar dichas reglas y deducir nuevas conclusiones, a partir de los hechos.

Estrategia

Para resolver el problema , pretendo que los hechos sean la definición del grafo, es decir, nodo origen, nodo destino, coste de la ruta entre ellos y si existe algún atajo (hecho que, en principio, el algoritmo no conoce).

Por otra parte, la definición teórica de la regla que queremos que ejecute, también es sencilla: "Si encuentras dos o más nodos cuya suma de los costes de las rutas entre ellos sea inferior a la ruta directa entre origen y destino, sustituye el coste por esa suma y muestra el atajo"

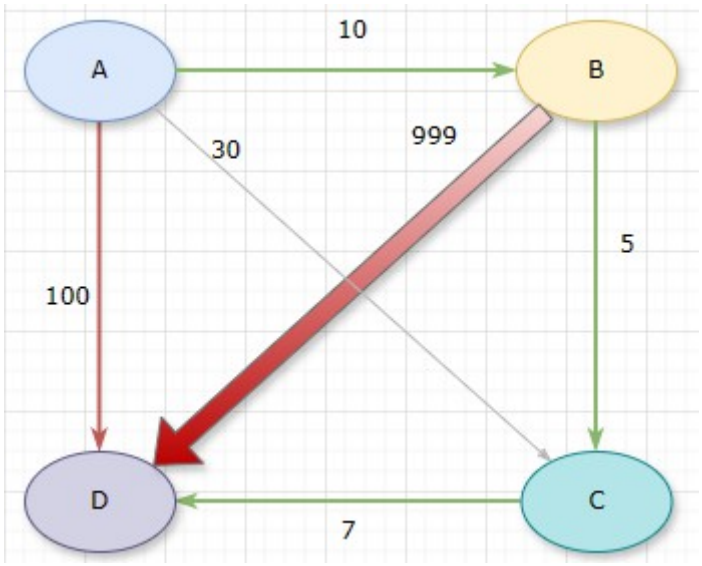
Para implementar dicha regla, hemos de tener en cuenta varias cuestiones:

- **La ruta entre todos los nodos implicados, debe existir**. Como nos basamos en reglas lógicas, no podemos pedir que calcule una ruta alternativa entre origen y destino, si no existe una ruta directa.

- **Si no existiera ruta directa, la creo con un valor alto**. De tal manera, que nunca elija la ruta directa, por ser demasiado costosa.

Concretización del problema

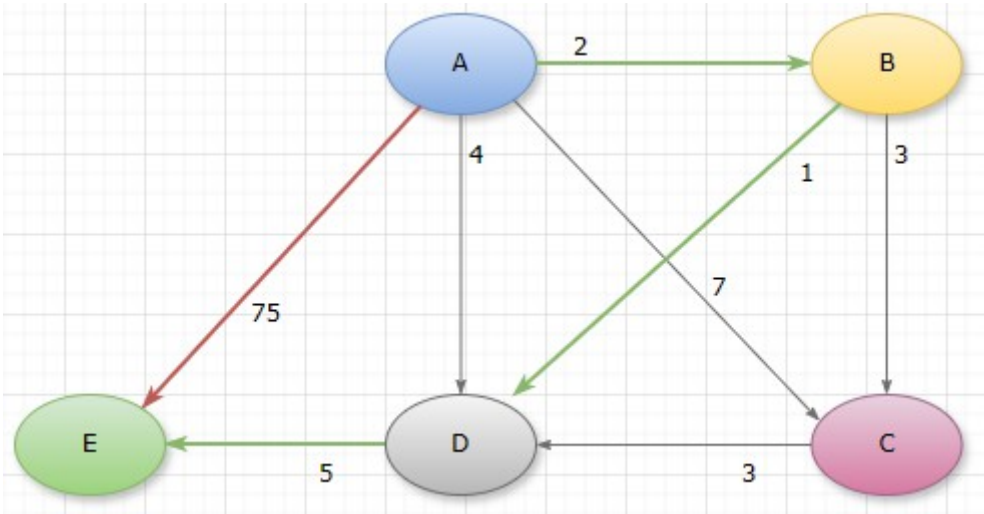
Vamos a probar con los siguientes grafos:



El primero, propuesto en clase , y del que conocemos el comportamiento deseado:

- Si se elige la ruta de A → C , el algoritmo deberá resolver 15, con la ruta final A → B → C, porque la ruta directa es 30, que es el doble de la que debería encontrar.
- Si se pide la ruta de A → D, la respuesta debería ser 22, ya que sería la ruta de A → C más la ruta de C → D y la ruta devuelta debería ser A → B → C → D.

En el segundo grafo



- Si se quiere conocer la ruta mínima de A → D , debería devolver un coste de 3 y la ruta A → B → D.
- Si se le pregunta por la ruta de A → E, la respuesta debería ser 8, con la ruta propuesta A → B → D → E (A→D + D → E).

Desarrollo

Se ha elegido la librería ClipsPy, para comunicar con CLIPS desde Python y, que desde un cuaderno jupyter se pueda interactuar con CLIPS.

Se trata de invocar a CLIPS, mediante ClipsPy, para resolver el problema de camino mínimo entre nodos de un grafo, aplicando la teoría del algoritmo de Dijkstra, si existe un camino con coste menor a la ruta directa, sustituye dicho camino por la ruta directa. Si no existiera dicha ruta, mostrará un mensaje que diga que no existe.

El desarrollo en python sigue los siguientes pasos:

1. Instalar la librería ClipsPy en el entorno de ejecución de Python.

   ```
   pip install clipspy
   ```

   Se debe incluir ! Si se instala en un entorno local

2. Definir un entorno CLIPS.

   ```
   from clips import Environment
   environment = Environment()
   ```

3. Construir los objetos de CLIPS, incluyendo la plantilla para las rutas, los hechos que representan las distancias entre los nodos y la regla para calcular la ruta más corta.

   ```
   DEFTEMPLATE_STRING = """
   (deftemplate RUTA
     (field ORIGEN)
     (field DESTINO)
     (field COSTE)
     (field ATAJO))
   """

   DEFFACTS_STRING = """
   (deffacts DISTANCIAS
     (RUTA (ORIGEN A) (DESTINO B) (COSTE 10) (ATAJO ""))
     (RUTA (ORIGEN A) (DESTINO C) (COSTE 30) (ATAJO ""))
     (RUTA (ORIGEN A) (DESTINO D) (COSTE 100) (ATAJO "") )
     (RUTA (ORIGEN B) (DESTINO C) (COSTE 5) (ATAJO ""))
     (RUTA (ORIGEN B) (DESTINO D) (COSTE 999) (ATAJO ""))
     (RUTA (ORIGEN C) (DESTINO D) (COSTE 7) (ATAJO ""))
     )
   """
   ```

```
DEFRULE_STRING = """
(defrule CALCULA
  ?HECHO1 <-(RUTA (ORIGEN ?ORI) (DESTINO ?DEST) (COSTE ?COSTE1) )
  (RUTA (ORIGEN ?OTRO) (DESTINO ?DEST) (COSTE ?COSTE2) (ATAJO ?ATAJO1)    )
  (RUTA (ORIGEN ?ORI) (DESTINO ?OTRO) (COSTE ?COSTE3) )
  (test (< (+ ?COSTE2 ?COSTE3) ?COSTE1))
  =>
  (modify ?HECHO1 (COSTE (+ ?COSTE2 ?COSTE3)) (ATAJO (str-cat ?OTRO ?
ATAJO1)))
)
"""
# Construye los objetos de CLIPS
environment.build(DEFTEMPLATE_STRING)
environment.build(DEFFACTS_STRING)
environment.build(DEFRULE_STRING)

# Recupera la plantilla de hecho
template = environment.find_template('RUTA')
```

4. Solicitar al usuario que ingrese el nodo de origen y destino.

```
nodo_origen  = str(input("Nodo Origen : ")).upper()
nodo_destino = str(input("Nodo Destino: ")).upper()
```

5. Ejecutar las activaciones en la agenda, lo que implica que las reglas se activarán y evaluarán en función de los hechos disponibles.

```
environment.reset()
environment.run()
```

6. Buscar los hechos en el entorno.

```
facts = environment.facts()
environment.facts()
```

7. Encontrar la ruta más corta entre el nodo de origen y destino o a través de nodos intermedios, mediante la ejecución de una regla de CLIPS.

```
ruta_mas_corta = None
for fact in facts:
  print(fact['ORIGEN'] , '-->' , fact['DESTINO'] , ' - Coste =' ,
fact['COSTE'] , ' - Ruta Intermedia:' , fact['ATAJO'])
  if fact.template.name == 'RUTA' and ((fact['ORIGEN'] == nodo_origen and
fact['DESTINO'] == nodo_destino) or (fact['ORIGEN'] == nodo_origen and
fact['ATAJO'] == nodo_destino)):
    ruta_mas_corta = fact
    break
```

En este punto, introduje un print, para monitorizar por dónde iba pasando CLIPS, qué hechos iba generando la regla ***CALCULA.***

```
print(fact['ORIGEN'] , '-->' , fact['DESTINO'] , ' - Coste =' , fact['COSTE'] , ' - Ruta Intermedia:' , fact['ATAJO'])
```

8. Mostrar la ruta más corta encontrada, incluyendo el coste y la ruta alternativa completa,si corresponde, o indicar si no se encontró una ruta alternativa.

```
if ruta_mas_corta:
        print(f"La ruta más corta desde {nodo_origen} a {nodo_destino} es:")
        print(f"Coste: {ruta_mas_corta['COSTE']}, Atajo: {nodo_origen}
                        {ruta_mas_corta['ATAJO']}{nodo_destino}")
else:
        print(f"No se encontró una ruta de {nodo_origen} a {nodo_destino}.")
```

En la parte de CLIPS, este se comportará como un sistema experto basado en reglas que utiliza la lógica y la manipulación de hechos para encontrar la ruta más corta entre dos nodos en un grafo representado por hechos de distancias, que le será creado desde Python.

La ruta del cuaderno en Google Colab es :

https://colab.research.google.com/drive/1Sn6AlcRXHm7HWR_yBXRVKoKgX7clou6d?usp=sharing

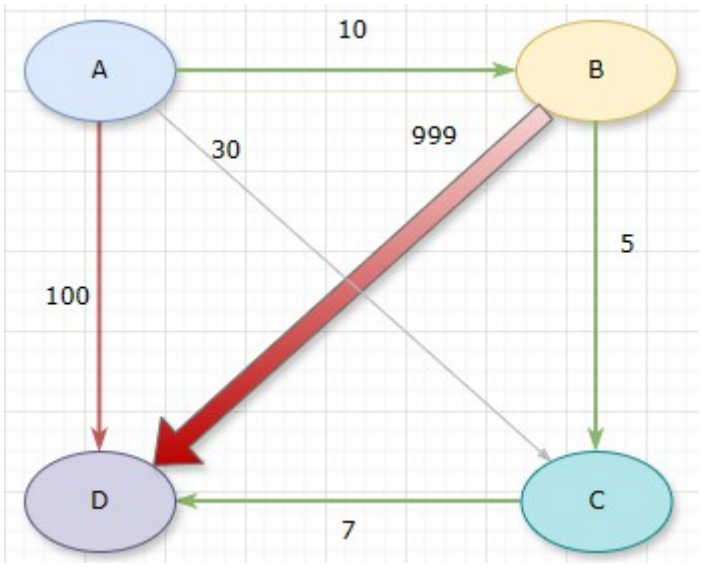
Y la de GitHub es:

https://github.com/roquearmas/IABD_DATA/blob/main/Djikstra_Clipspy.ipynb

Resultados

A continuación, los resultados de las pruebas con los dos grafos descritos anteriormente:

Grafo 1



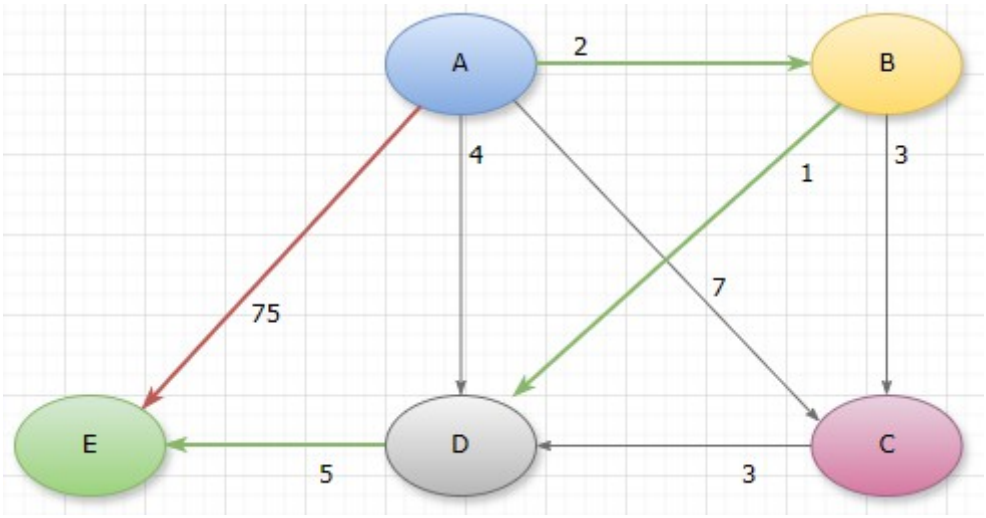
Ruta A → C

```
Nodo Origen : a
Nodo Destino: c
A --> B  - Coste = 10  - Ruta Intermedia:
A --> C  - Coste = 15  - Ruta Intermedia: B
La ruta más corta desde A a C es:
Coste: 15, Atajo: ABC
```

Ruta A → D

```
Nodo Origen : A
Nodo Destino: D
A --> B  - Coste = 10  - Ruta Intermedia:
A --> C  - Coste = 15  - Ruta Intermedia: B
A --> D  - Coste = 22  - Ruta Intermedia: BC
La ruta más corta desde A a D es:
Coste: 22, Atajo: ABCD
```

Grafo 2



Ruta A → D

```
Nodo Origen : A
Nodo Destino: D
A --> B  - Coste = 2  - Ruta Intermedia:
A --> C  - Coste = 5  - Ruta Intermedia: B
A --> D  - Coste = 3  - Ruta Intermedia: B
La ruta más corta desde A a D es:
Coste: 3, Atajo: ABD
```

Ruta A → E

```
Nodo Origen : A
Nodo Destino: E
A --> B  - Coste = 2  - Ruta Intermedia:
A --> C  - Coste = 5  - Ruta Intermedia: B
A --> D  - Coste = 3  - Ruta Intermedia: B
A --> E  - Coste = 8  - Ruta Intermedia: D
La ruta más corta desde A a E es:
Coste: 8, Atajo: ADE
```

En principio, no nos da el resultado esperado. El problema , es que no existe una ruta definida entre B y E, por lo que no se puede calcular una ruta alternativa entre estos dos nodos.

Inserto un hecho que defina ese camino, con un coste superior al de B → D → E

```
(RUTA (ORIGEN B) (DESTINO E) (COSTE 20) (ATAJO ""))
```

El resultado es el siguiente :

```
Nodo Origen : A
Nodo Destino: E
A --> B  - Coste = 2  - Ruta Intermedia:
A --> C  - Coste = 5  - Ruta Intermedia: B
A --> D  - Coste = 3  - Ruta Intermedia: B
A --> E  - Coste = 8  - Ruta Intermedia: BD
La ruta más corta desde A a E es:
Coste: 8, Atajo: ABDE
```

Ahora, la regla si es capaz de encontrar una ruta alternativa que mejore los costes de las rutas intermedias.

Conclusiones

Se ha conseguido una implementación exitosa, resultado de adaptar el algoritmo de Dijkstra para encontrar el camino mínimo entre los nodos de un grafo dado, utilizando CLIPS embebido en Python, a través de ClipsPy.

El sistema fue capaz de encontrar rutas alternativas, menos costosas que las rutas directas y que las rutas intermedias.

Ante la ausencia de rutas directas o intermedias, el algoritmo muestra la siguiente ruta existente menos costosa. Se detecta esa carencia, que también puede verse como un desafío, que se supera, simplemente, insertando un nuevo hecho, una ruta directa con un coste alto, para que el algoritmo no la seleccione.

Se destaca el potencial de CLIPS, como herramienta para sistemas expertos, junto con la implementación en Python, que simplifica las interacciones con dicho sistema.