Búsqueda en coste uniforme: algoritmo de Dijkstra[1]

Albert Sanchis
Alfons Juan

Departamento de Sistemas
Informáticos y Computación

[1] Para una correcta visualización, se requiere Acrobat Reader v. 7.0 o superior

Objetivos formativos

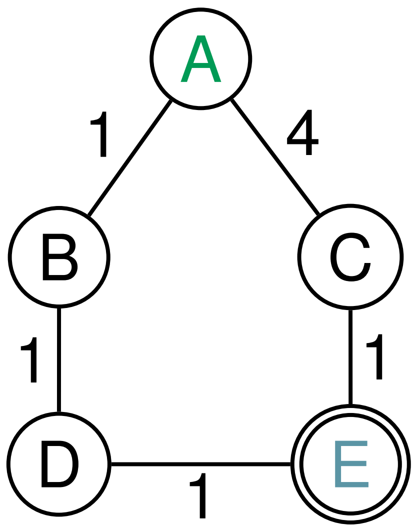
- Describir búsqueda en coste uniforme o algoritmo de Dijkstra.
- Construir el árbol de búsqueda en coste uniforme.
- Analizar optimalidad y complejidad de búsq. en coste uniforme.

Índice

1 Introducción

Búsqueda en coste uniforme (UCS, de Uniform-cost search) o ***algoritmo de Dijkstra*** consiste en enumerar caminos hasta encontrar una solución, priorizando los de menor coste (parcial) y evitando ciclos:

Grafo de estados



Búsqueda

A 0

AB 1
AC 4

ABD 2

ABDE 3 sol.!

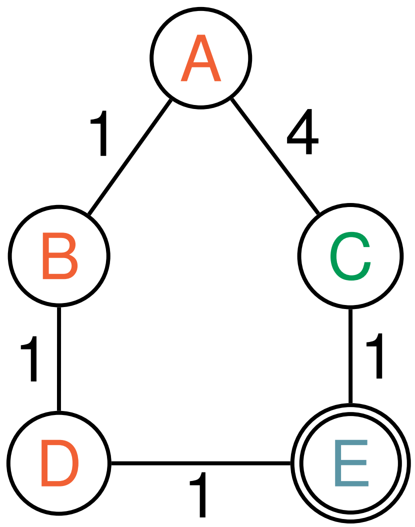
Nota: UCS generaliza BFS para aristas de coste diferente.

2 Coste uniforme o algoritmo de Dijkstra [1, 2, 3]

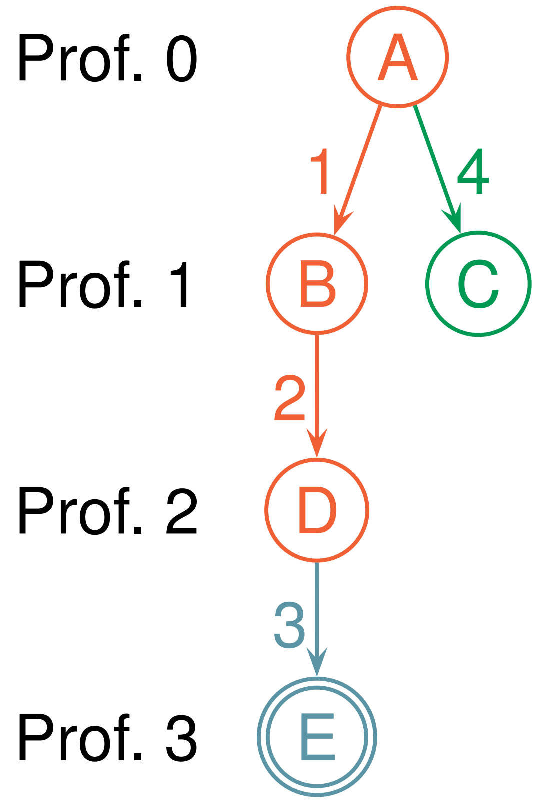
UCS(G, s') //***Uniform-cost search***; G grafo ponderado, s' *start*
$O = IniCola(s', g_{s'} \triangleq 0)$ // ***Open: cola de prioridad*** g
$C = \emptyset$ // ***Closed:*** nodos explorados
mientras no $ColaVacia(O)$: // ***1ro el mejor:*** $s = \arg\min_{n \in O} g_n$
$s = Desencola(O)$ // desempates a favor de objetivo
si $Objetivo(s)$ **retorna** s // solución encontrada!
$C = C \cup \{s\}$ // s explorado
para toda $(s, n) \in Adyacentes(G, s)$: // generación: n hijo de s
$x = g_s + w(s, n)$ // coste del camino de s' a n pasando por s
si $n \notin C \cup O$: $Encola(O, n, g_n \triangleq x)$
si no si $n \in O$ **y** $x < g_n$: $Modcola(O, n, g_n \triangleq x)$
retorna NULL // ninguna solución encontrada

3 El árbol de búsqueda en coste uniforme

Grafo de estados



Árbol de búsqueda

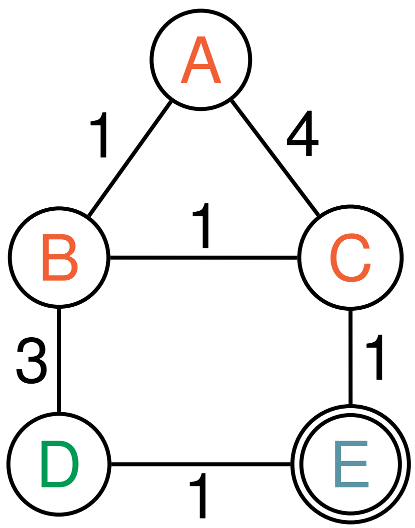


It. 4: A0B1D2E3C4 E3 sol.!

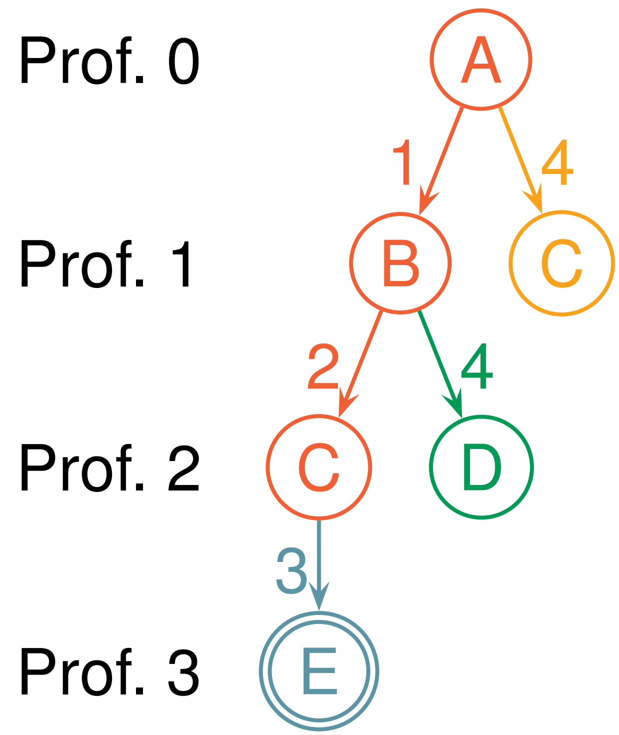
Nota: BFS encontraría ACE, de coste 5, en lugar de ABDE, de coste 3.

El árbol de búsqueda en coste uniforme (cont.)

Grafo de estados



Árbol de búsqueda



It. 4: A0B1C2E3D4 E3 sol.!

Nota: UCS mantiene los ***caminos más cortos*** entre el nodo inicial y cada nodo abierto, ***atravesando nodos explorados únicamente.***

4 Optimalidad y complejidad

- ***Optimalidad:*** Sí, con pesos no negativos.

- ***Complejidad:***

  ▷ $G = (V, E)$ *explícito:* $O(|E| \log|V|)$ con un *heap* [4].

  ▷ G *implícito* con ***factor de ramificación*** b***:***

  Peor caso: solución a profundidad $d = \left\lfloor \frac{C^*}{\epsilon} \right\rfloor$, donde ϵ es el peso mínimo y C^* el coste del camino óptimo.

  Se genera un árbol completo con nodos a profundidad $d + 1$.

  $O(b^{d+1})$ temporal y espacial.

5 Conclusiones

Hemos visto:

- El algoritmo de búsq. en coste uniforme o algoritmo de Dijkstra.
- El árbol de búsqueda en coste uniforme.
- La calidad y complejidad de búsqueda en coste uniforme.

Algunos aspectos a destacar sobre UCS:

- Completa y óptima con aristas de coste positivo.
- Coste espacial excesivo, sobre todo con soluciones profundas.
- El algoritmo de Dijkstra es la principal técnica de referencia para la búsqueda del camino más corto entre dos nodos de un grafo explícito, o todos los caminos más cortos entre un nodo dado y el resto.

Referencias

[1] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1959.

[2] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition, 2010.

[3] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018.

[4] Mo Chen et al. Priority Queues and Dijkstra's Algorithm. Technical report, UTCS TR-07-54, 2007.

ucs.py

```python
#!/usr/bin/env python3
import heapq
G1={'A':[('B',1),('C',4)],'B':[('A',1),('D',1)],
		'C':[('A',4),('E',1)],'D':[('B',1),('E',1)],
		'E':[('C',1),('D',1)]}
G2={'A':[('B',1),('C',4)],'B':[('A',1),('C',1),('D',3)],
		'C':[('A',4),('B',1),('E',1)],'D':[('B',3),('E',1)],
		'E':[('C',1),('D',1)]}
def ucs(G,s,t):
	Od={s:0}; Cd={} # Open and Closed g dict
	Oh=[]; heapq.heappush(Oh,(0,s,[s])) # Open heap
	while Od:
		s=None
		while s not in Od: gs,s,path=heapq.heappop(Oh) # delete-min
		if s==t: return gs,path
		del Od[s]; Cd[s]=gs
		for n,wsn in G[s]:
			gn=gs+wsn
			if n not in Cd and (n not in Od or gn<Od[n]):
				heapq.heappush(Oh,(gn,n,path+[n])) # insert
				Od[n]=gn
print(ucs(G1,'A','E'))
print(ucs(G2,'A','E'))
```

ucs.py.out

```
(3, ['A', 'B', 'D', 'E'])
(3, ['A', 'B', 'C', 'E'])
```