UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cerca en cost uniforme: algorisme de Dijkstra[1]

Albert Sanchis
Alfons Juan

DSIC

Departament de Sistemes Informàtics i Computació

[1] Per a una correcta visualització, es requereix l'Acrobat Reader v. 7.0 o superior

Objectius formatius

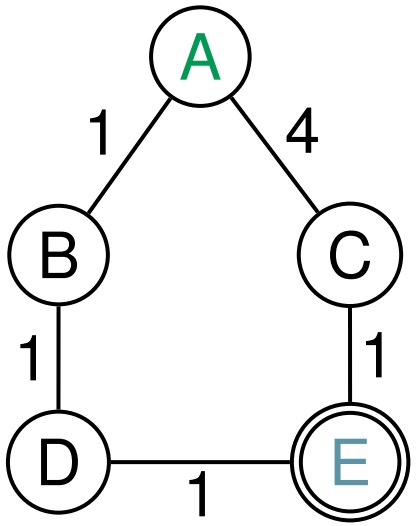
- Descriure cerca en cost uniforme o algorisme de Dijkstra.
- Construir l'arbre de cerca en cost uniforme.
- Analitzar l'optimalitat i complexitat de cerca en cost uniforme.

Índex

1 Introducció

Cerca en cost uniforme (UCS, de Uniform-cost search) o ***algorisme de Dijkstra*** consisteix a enumerar camins fins a trobar una solució, prioritzant els de menor cost (parcial) i evitant cicles:

Graf d'estats



Cerca

A 0

AB 1

AC 4

ABD 2

ABDE 3 sol.!

Nota: UCS generalitza BFS per a arestes de cost diferent.

2 Cost uniforme o algorisme de Dijkstra [1, 2, 3]

UCS(G, s') // ***Uniform-cost search***; G graf ponderat, s' *start*

$O = IniCua(s', g_{s'} \triangleq 0)$ // ***Open: cua de prioritat*** g

$C = \emptyset$ // ***Closed:*** nodes explorats

mentre no $CuaBuida(O)$: // ***1r el millor:*** $s = \arg\min_{n \in O} g_n$

$s = Desencua(O)$ // desempats a favor d'objectius

si $Objectiu(s)$ **retorna** s // solució trobada!

$C = C \cup \{s\}$ // s explorat

per a tota $(s, n) \in Adjacents(G, s)$: // generació: n fill d's

$x = g_s + w(s, n)$ // cost del camí d's' a n passant per s

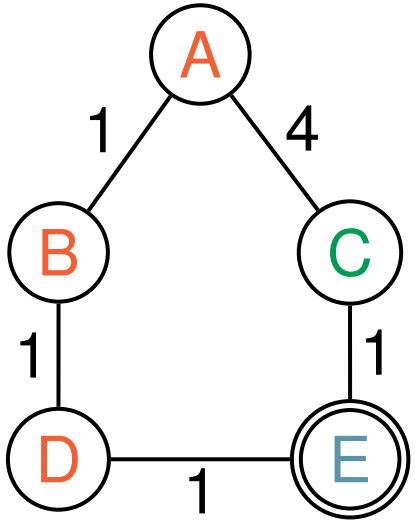
si $n \notin C \cup O$: $Encua(O, n, g_n \triangleq x)$

si no si $n \in O$ **i** $x < g_n$: $Modcua(O, n, g_n \triangleq x)$

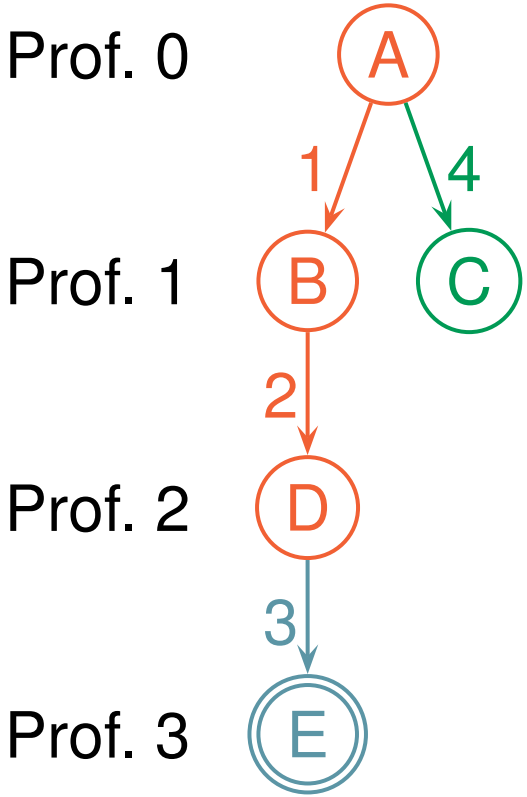
retorna NULL // cap solució trobada

3 L'arbre de cerca en cost uniforme

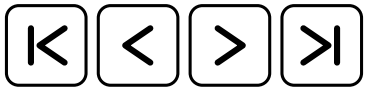
Graf d'estats



Arbre de cerca



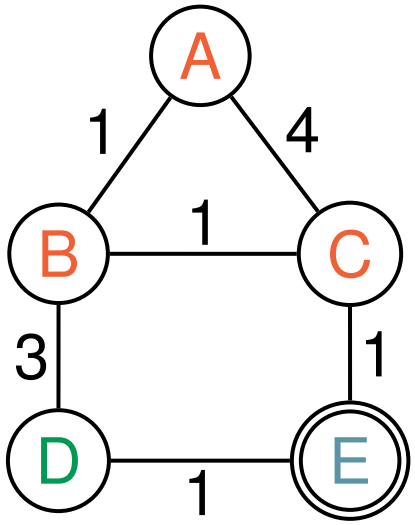
It. 4: A0B1D2E3C4 E3 sol.!



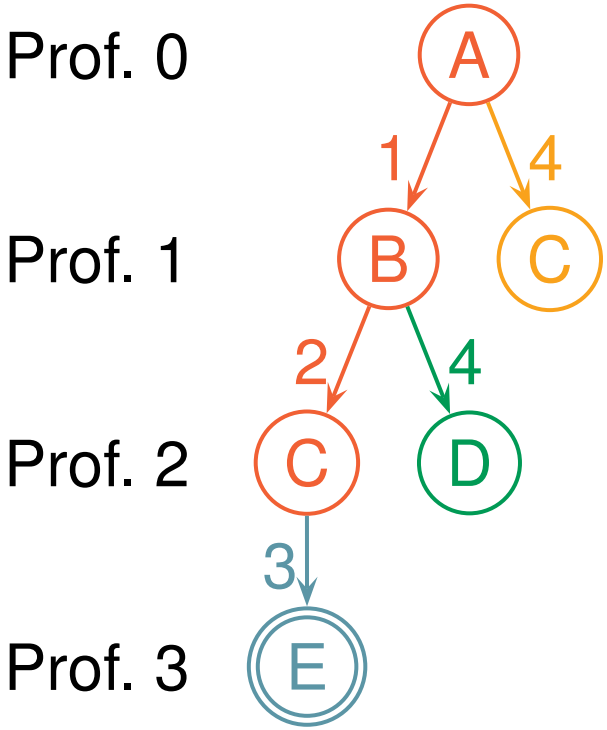
Nota: BFS trobaria ACE, de cost 5, en lloc d'ABDE, de cost 3.

L'arbre de cerca en cost uniforme (cont.)

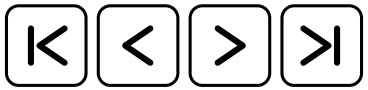
Graf d'estats



Arbre de cerca



It. 4: A0B1C2E3D4 E3 sol.!



Nota: UCS manté els ***camins més curts*** entre el node inicial i cada node obert, ***travessant nodes explorats únicament.***

4 Optimalitat i complexitat

- ***Optimalitat:*** Sí, amb pesos no negatius.

- ***Complexitat:***

  - $G = (V, E)$ ***explícit:*** $O(|E| \log |V|)$ amb un *heap* [4].

  - G *implícit* amb ***factor de brancatge*** b***:***

    Pitjor cas: solució a profunditat $d = \left\lfloor \frac{C^*}{\epsilon} \right\rfloor$, on ϵ és el pes mínim i C^* el cost del camí òptim.

    Es genera un arbre complet amb nodes a profunditat $d + 1$.

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

5 Conclusions

Hem vist:

- L'algorisme de cerca en cost uniforme o algorisme de Dijkstra.
- L'arbre de cerca en cost uniforme.
- La qualitat i complexitat de cerca en cost uniforme.

Alguns aspectes a destacar sobre UCS:

- Completa i òptima amb arestes de cost positiu.
- Cost espacial excessiu, sobretot amb solucions profundes.
- L'algorisme de Dijkstra és la principal tècnica de referència per a la cerca del camí més curt entre dos nodes d'un graf explícit, o tots els camins més curts entre un node donat i la resta.

Referències

[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

```
#!/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'])
```