UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Búsqueda en anchura[1]

Albert Sanchis
Alfons Juan

DSIC

Departamento de Sistemas Informáticos y Computación

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

Objetivos formativos

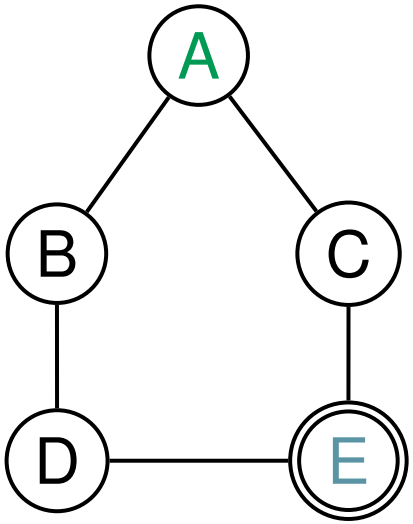
- Describir búsqueda en anchura.
- Construir el árbol de búsqueda en anchura.
- Analizar la calidad y complejidad de búsqueda en anchura.

Índice

1 Introducción

Búsqueda en anchura (BFS, de Breadth-first search) consiste en enumerar caminos (desde el nodo inicial) hasta encontrar una solución, priorizando los más cortos y evitando ciclos (sin repetir nodos):

Grafo de estados



Búsqueda

A

AB

AC

ABD

ACE sol.!

2 Búsqueda en anchura [1, 2, 3, 4]

BFS(G, s'**)** // ***Breadth-first search***; G grafo y s' nodo inicial

  $O = IniCola(s')$ // ***Open: frontera-cola*** de la búsqueda

  $C = \emptyset$ // ***Closed:*** conjunto de nodos explorados

  mientras no $ColaVacia(O)$:

    $s = Desencola(O)$ // selección ***FIFO (First in, first out)***

    $C = C \cup \{s\}$ // s ya explorado

    para toda $(s, n) \in Adyacentes(G, s)$: // generación: n hijo de s

      si $n \notin C \cup O$: // n no descubierto hasta ahora

        si $Objetivo(n)$ **retorna** n // solución encontrada!

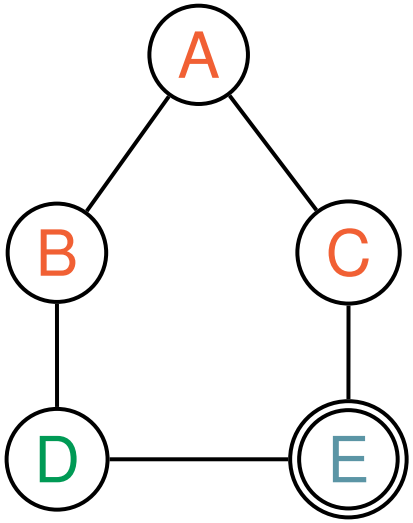
        $Encola(O, n)$ // añadimos n a la cola

  retorna NULL // ninguna solución encontrada

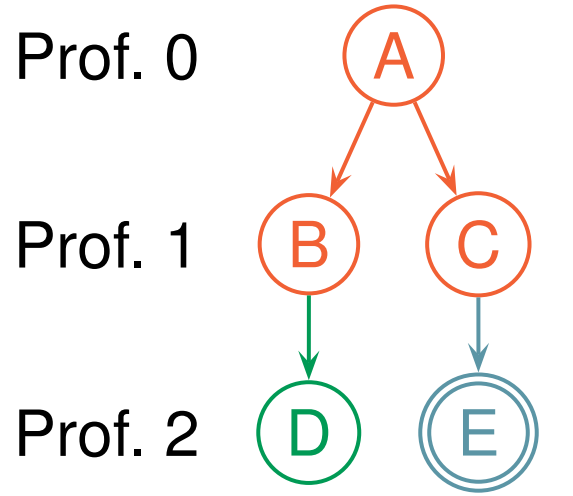
3 El árbol de búsqueda en anchura

BFS genera un ***árbol de búsqueda*** arraigado al nodo inicial y ***profundidad*** d igual a la longitud "del" (un) camino más corto hacia una solución:

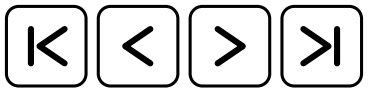
Grafo de estados



Árbol de búsqueda



It. 3: ABCD E sol.!



Nota: desempates resueltos por orden alfabético ("1ro el izq.").

4 Completitud, optimalidad y complejidad

- ***Completitud:*** Sí, siempre encuentra solución (si existe).
- ***Optimalidad:*** Sí, con acciones de coste positivo idéntico.
- ***Complejidad:***
  - $G = (V, E)$ *explícito:* $O(|V| + |E|)$ temporal y espacial.
  - G *implícito,* ***factor de ramificación*** b y hasta ***profundidad*** d**:**

    Árbol completo (*Peor caso*): $O(b^d)$ temporal y espacial.

    Si $Objetivo$ después de selección: $O(b^{d+1})$ temporal y espacial.

5 Conclusiones

Hemos visto:

- El algoritmo de búsqueda en anchura.
- El árbol de búsqueda en anchura.
- La calidad y complejidad de búsqueda en anchura.

Algunos aspectos a destacar sobre BFS:

- Completa y óptima con aristas de coste idéntico.
- Coste espacial excesivo, sobre todo con soluciones profundas.
- Puede ser una buena opción para grafos dispersos (pocas aristas), soluciones superficiales y aristas de coste idéntico.

Referencias

[1] E. Moore. The shortest path through a maze. In *Proc. of the Int. Symposium on the Theory of Switching, Part II*, pages 285–292. Harvard University Press, 1959.

[2] C. Y. Lee. An algorithm for path connections and its applications. *IRE Trans. on Electronic Computers*, EC-10, 1961.

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

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

bfs.py

```
#!/usr/bin/env python3
from queue import Queue
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
→ 'D':['B','E'],'E':['C','D']}
def bfs(G,s,t):
→if s==t: return [s]
→O=Queue(); O.put((s,[s])) # Open queue
→OCs=set(); OCs.add(s)     # Open and closed set
→while O:
→→s,path=O.get()
→→for n in G[s]:
→→→if n not in OCs:
→→→→if n==t: return path+[n]
→→→→O.put((n,path+[n]))
→→→→OCs.add(n)
print(bfs(G,'A','E'))
```

bfs.py.out

```
['A', 'C', 'E']
```