UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Búsqueda en profundidad[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

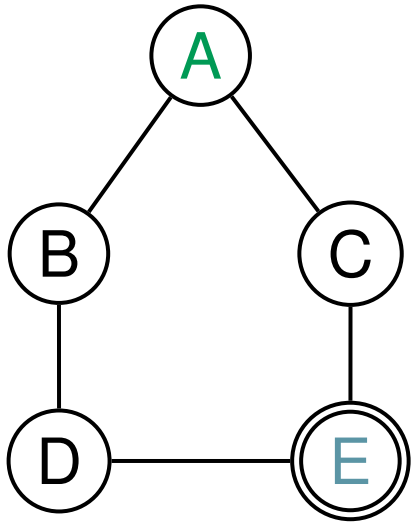
- Analizar búsqueda en profundidad.
- Describir búsq. en profundidad en la variante de *backtracking*.
- Aplicar búsqueda en profundidad iterativa.

Índice

1. Introducción

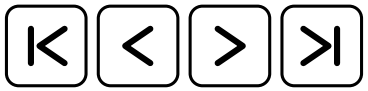
Búsqueda en profundidad (DFS, de Depth-first search) consiste en enumerar caminos hasta encontrar una solución, priorizando els más profundo (largo) y limitando la longitud máxima (sólo caminos finitos):

Grafo de estados



Búsqueda (limitada)

A
AB
AC
ABA
ABD
ABAB
ABAC
ABDB
ABDE sol.!



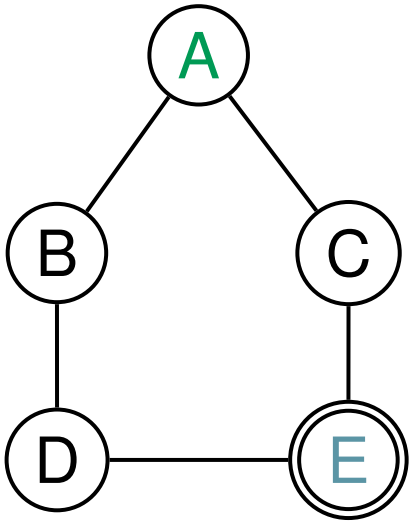
Nota: no se almacenan nodos cerrados por eficiencia espacial.

2. Búsqueda en profundidad [1, 2]

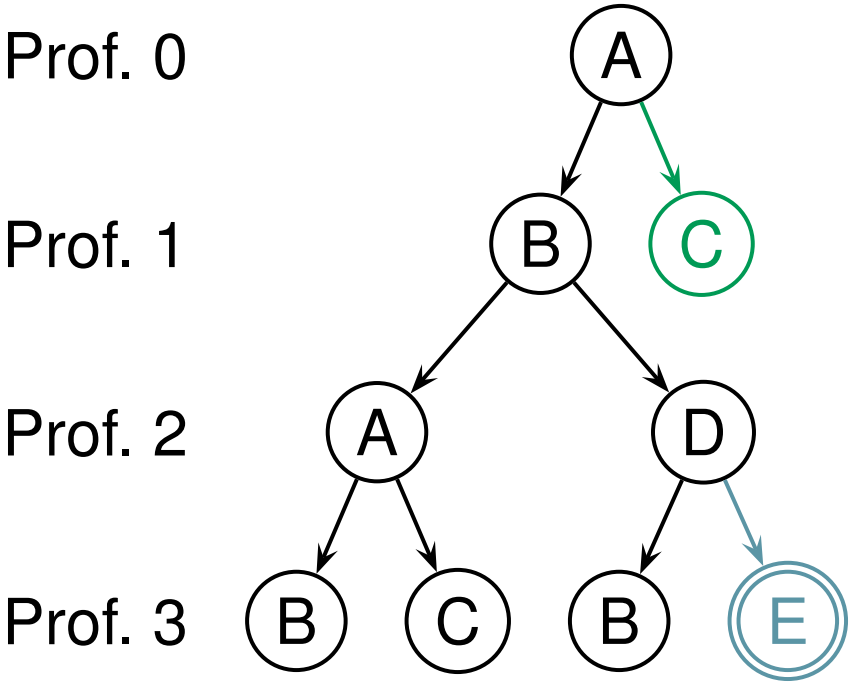
DFS(G, s', m**)** // ***Depth-first search*** con profundidad máxima m
  $O = IniPila(s')$ // ***Open: frontera-pila*** de la búsqueda
  mientras no $PilaVacia(O)$:
    $s = Desapila(O)$ // selección ***LIFO (Last in, first out)***
    si $Objetivo(s)$ **retorna** s // solución encontrada!
    si $Profundidad(s) < m$: // no a profundidad máxima
      para toda $(s, n) \in Adyacentes(G, s)$: // generación: n hijo de s
        $Apila(O, n)$ // añadimos n a la pila
  retorna NULL // ninguna solución encontrada

El árbol de búsqueda en profundidad ($m = 3$)

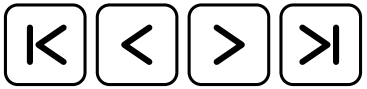
Grafo de estados



Árbol de búsqueda



It. 8: ABABCDBEC E sol.!



Calidad: incompleta y subòptima.

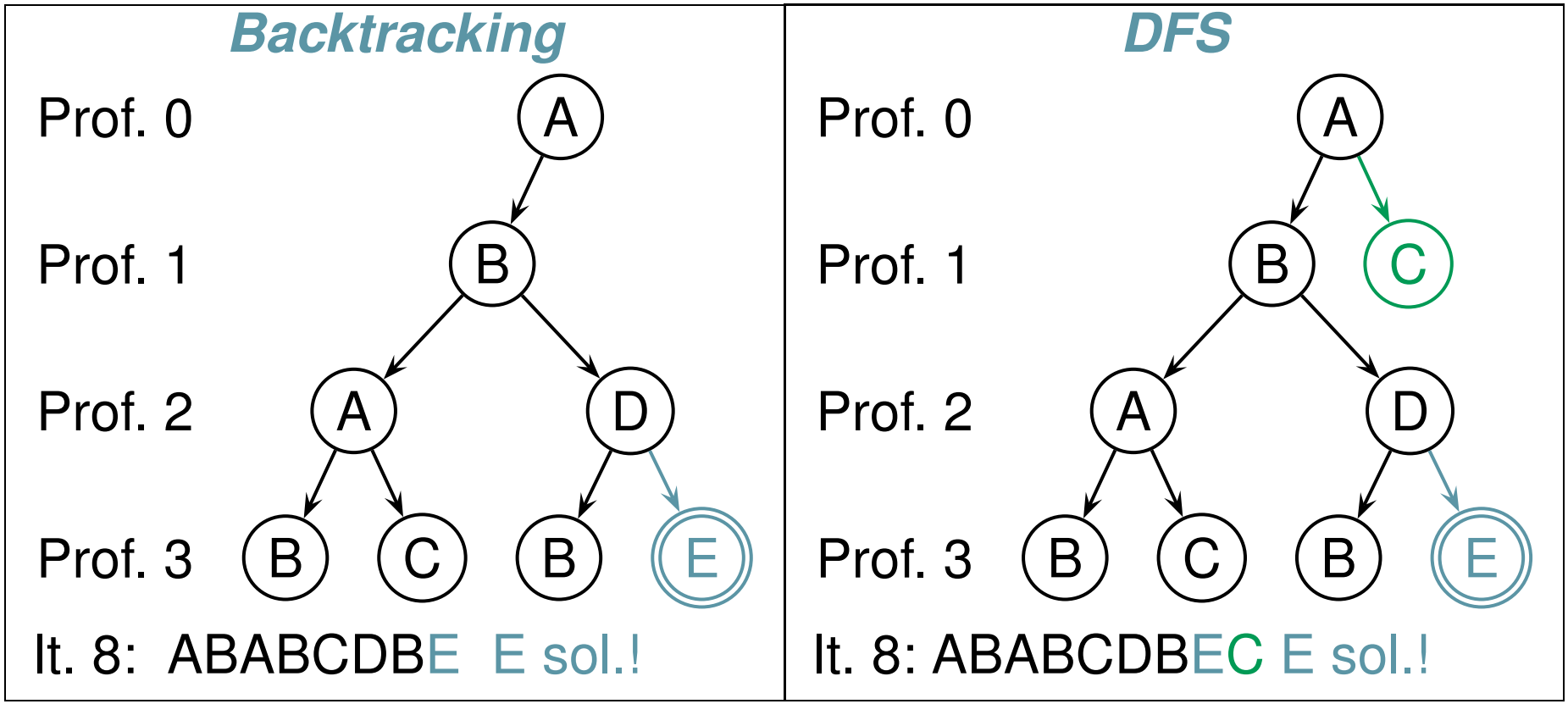
Complejidad: $O(b^m)$ temporal y $O(bm)$ espacial.

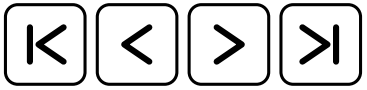
3. Backtracking

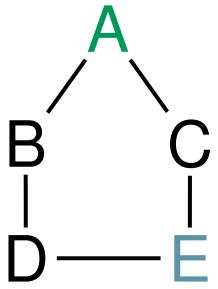
Variante de DFS (recursiva) con generación individual de hijos:

```
BT(G, s, m)                    // Backtracking con profundidad máxima m
  si Objetivo(s) retorna s                          // solución encontrada!
  si m = 0 retorna NULL                             // profundidad máxima
  n = PrimerAdyacente(G, s)                         // generación: n primer hijo de s
  mientras n ≠ NULL:
    r = BT(G, n, m − 1)                             // resultado del hijo actual
    si r ≠ NULL: retorna r                          // si r es solución, acabamos
    n = SiguienteAdyacente(G, s, n)                 // generación: n sig. hijo de s
  retorna NULL                                      // ninguna solución encontrada
```

El árbol de búsqueda con backtracking ($m = 3$)







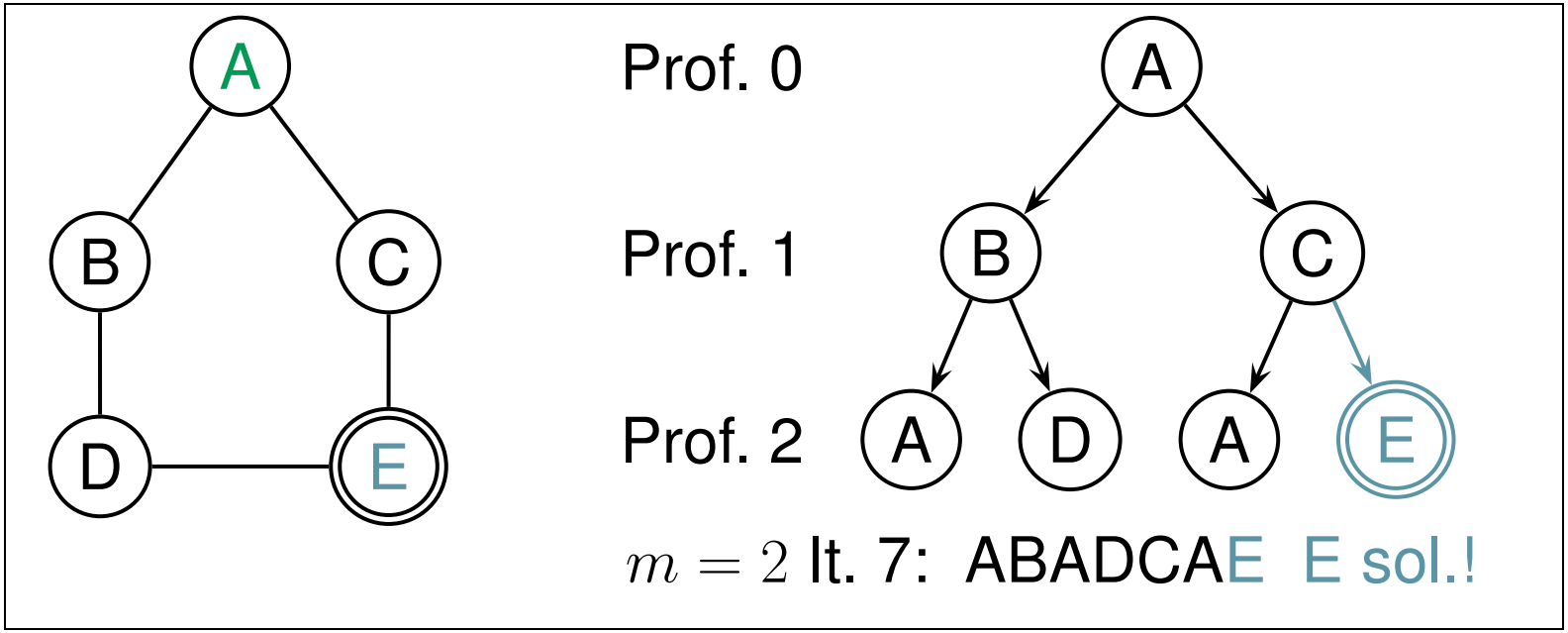
Calidad: incompleta y subóptima.

Coste temporal: $O(b^m)$.

Coste espacial: $O(m)$, mejor que el $O(bm)$ de DFS.

4. Búsqueda en profundidad iterativa [3]

PI(G, s) // ***Profundidad Iterativa***
  para $m = 0, 1, 2, \ldots$: **si** $(r = \text{DFS}(G, s, m)) \neq \text{NULL}$: **retorna** r



Calidad: completa y óptima si acciones de coste positivo idéntico.

Complejidad: $O(b^d)$ temporal y $O(bd)$ espacial.

5. Conclusiones

Hemos visto:

- Búsqueda en profundidad.
- La variante con *backtracking* de búsq. en profundidad recursiva.
- La extensión llamada búsqueda en profundidad iterativa.

Algunos aspectos a destacar sobre DFS:

- Incompleta y subòptima.
- Coste espacial razonable, sobre todo con *backtracking*.
- Puede ser buena opción para buscar soluciones profundas, especialmente si la longitud (coste) del camino no es muy relevante.
- La extensió de búsqueda en profundidad iterativa es completa, óptima con aristas de coste idéntico, y mantiene un coste espacial razonable, por lo que es generalmente preferible a BFS.

Referencias

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

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

[3] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 1985.

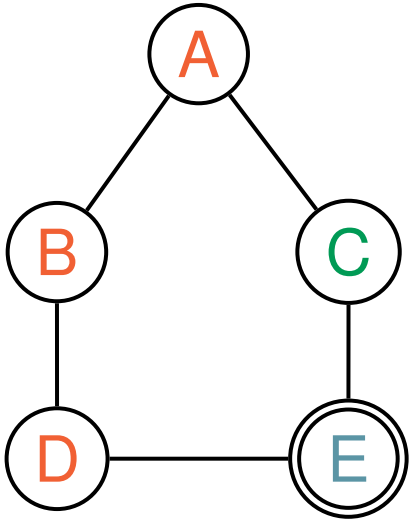
Búsqueda en profundidad (con grafo) [1]

Búsqueda en grafo: mantiene un conjunto de nodos explorados C.

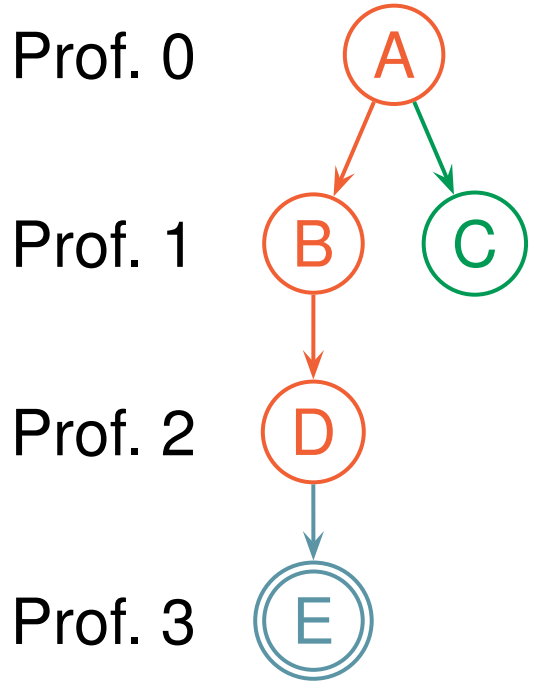
DFS(G, s'**)** // ***Depth-first search***; G grafo y s nodo inicial
  $O = IniPila(s')$ // ***Open: frontera-pila*** de la búsqueda
  $C = \emptyset$ // ***Closed:*** conjunto de nodos explorados
  mientras no $PilaVacia(O)$:
    $s = Desapila(O)$ // selección ***LIFO (Last in, first out)***
    si $Objetivo(s)$ **retorna** n // solución encontrada!
    $C = C \cup \{s\}$ // 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
        $Apila(O, n)$ // añadimos n a la pila
  retorna NULL // ninguna solución encontrada

Búsq. profundidad (con grafo): árbol de búsqueda

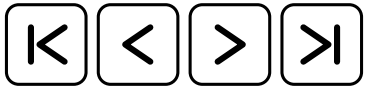
Grafo de estados



Árbol de búsqueda



It. 4: ABDEC E sol.!



Propiedades: completa, subóptima, $O(|V|+|E|)$ temporal y espacial.

En general, peor que búsqueda en anchura!

dfs.py

```
#!/usr/bin/env python3
from collections import deque
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
→ 'D':['B','E'],'E':['C','D']}
def dfsi(G,s,m,t):
→O=deque(); O.append((s,[s]))
→while O:
→→s,path=O.pop()
→→if s==t: return path
→→if len(path)<=m:
→→→for n in list(reversed(G[s])):
→→→→O.append((n,path+[n]))
print(dfsi(G,'A',3,'E'))
```

dfs.py.out

```
['A', 'B', 'D', 'E']
```

bt.py

```
#!/usr/bin/env python3
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
→ 'D':['B','E'],'E':['C','D']}
def bt(G,s,m,t):
→if s==t: return [s]
→if m==0: return None
→for i in [0,1]:
→→n=G[s][i]; path=bt(G,n,m-1,t)
→→if path!=None: return [s]+path
print(bt(G,'A',3,'E'))
```

bt.py.out

```
['A', 'B', 'D', 'E']
```

pi.py

```
#!/usr/bin/env python3
G={'A':['B','C'],'B':['A','D'],'C':['A','E'],
→ 'D':['B','E'],'E':['C','D']}
def dfs(G,s,m,t):
→if s==t: return [s]
→if m==0: return None
→for n in G[s]:
→→path=dfs(G,n,m-1,t)
→→if path!=None: return [s]+path
def pi(G,s,t):
→for m in range(len(G)):
→→path=dfs(G,s,m,t)
→→if path!=None: return path
print(pi(G,'A','E'))
```

pi.py.out

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