UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Búsqueda Primero-El-Mejor: Búsqueda voraz[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

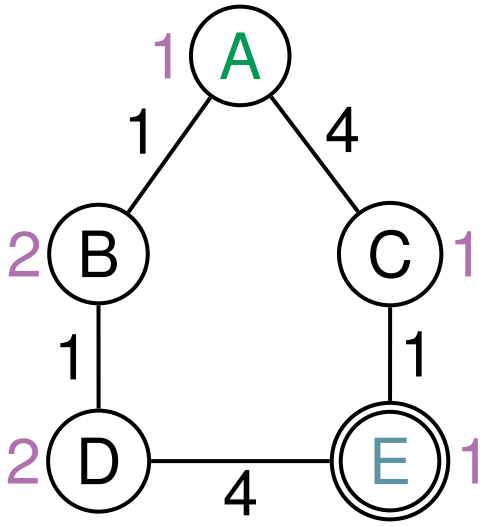
- Describir el algoritmo de búsqueda Primero-El-Mejor general.
- Aplicar búsqueda (Primero-El-Mejor) voraz.
- Analizar la optimalidad y complejidad de búsqueda voraz.
- Ilustrar la incompletitud de voraz con búsqueda en árbol.

Índice

1. Introducción

Búsqueda Primero-El-Mejor consiste en enumerar caminos hasta encontrar una solución, priorizando los de menor "coste" (f) y evitando ciclos:

Grafo de estados

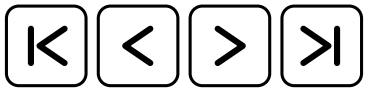


Búsqueda

A 1

AB 2
AC 1

ACE 1 sol.!



Primero-El-Mejor generaliza A* permitiendo el uso de cualquier ***función de evaluación (heurística)*** f, no necesariamente del tipo $f = g + h$.

2. El algoritmo Primero-El-Mejor [1]

BF(G, s', f**)** // ***Best-First;*** G, s', f ***función de evaluación***

$O = IniCola(s', f(s'))$ // ***Open: cola de prioridad*** f

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

mientras no $ColaVacia(O)$: // ***1ro el mejor:*** $s = \arg\min_{n \in O} f_n$

$s = Desencola(O)$ // desempates a favor de objetivos

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

$x = f(n)$ // posible f_n nuevo

si $n \notin C \cup O$: $Encola(O, n, f_n \triangleq x)$

si no si $n \in O$ **y** $x < f_n$: $Modcola(O, n, f_n \triangleq x)$

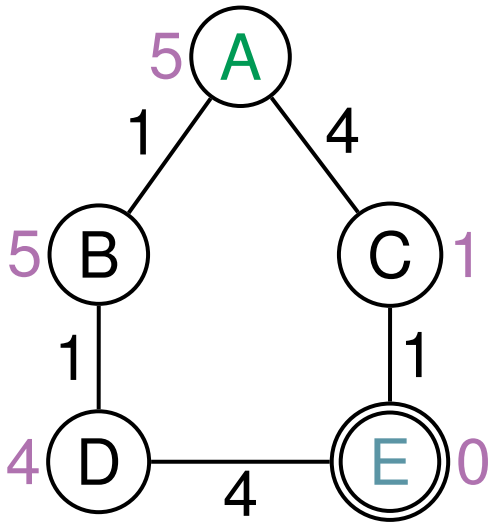
si no si $n \in C$ **y** $x < f_n$: $C = C \setminus \{n\}$; $Encola(O, n, f_n \triangleq x)$

retorna NULL // ninguna solución encontrada

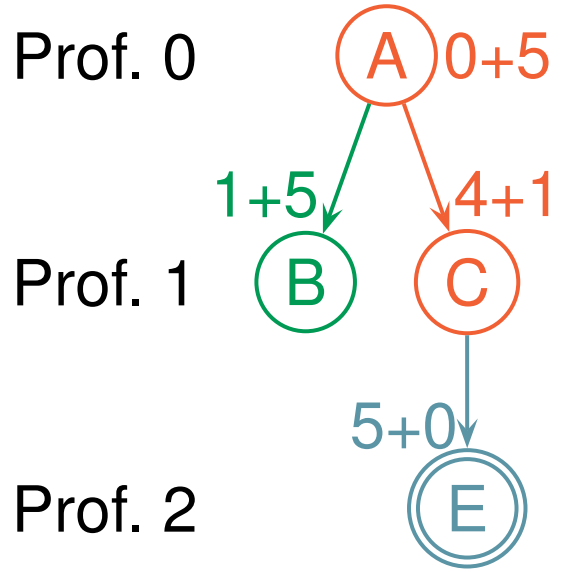
3. Búsqueda (Primero-El-Mejor) voraz

Búsqueda (Primero-El-Mejor) voraz consiste en usar $f = h$:

Grafo de estados

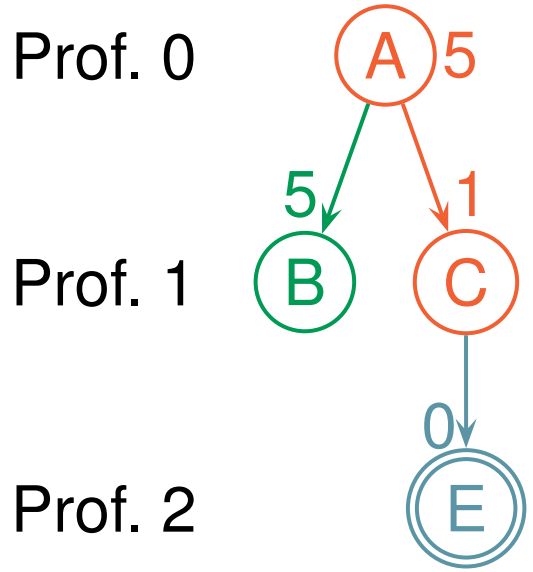


$f = g + h$ ***(A*)***



It 3: A5C5E5B6

$f = h$ ***(voraz)***



It 3: A5C1E0B5

Intuición: aproximarse rápidamente a soluciones.

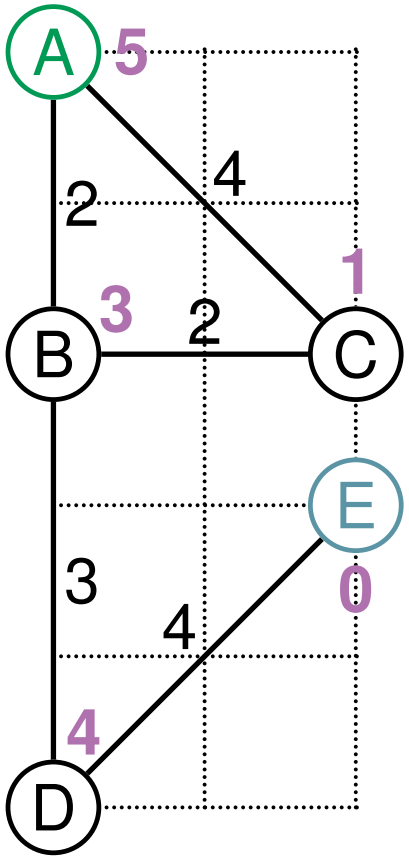
Optimalidad: completa en grafos finitos y subóptima.

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

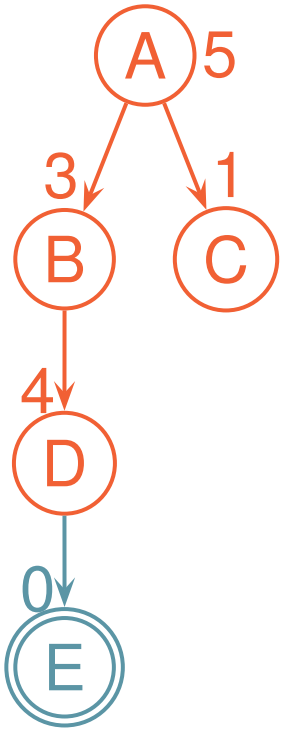
4. Voraz con búsqueda en árbol

Voraz con búsqueda en árbol [2] ($C = \emptyset$) es incompleta:

Grafo de estados

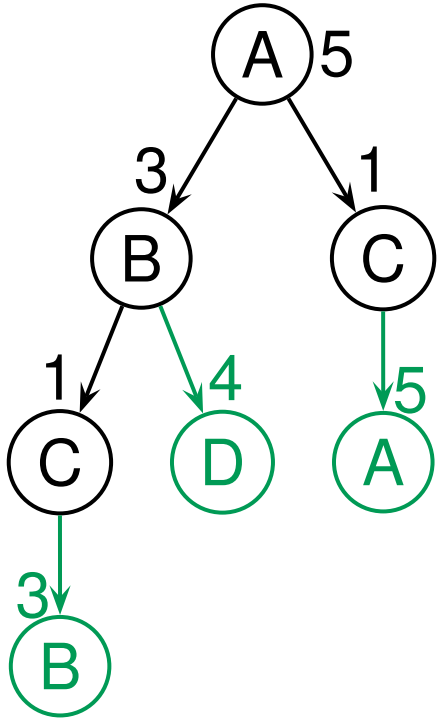


Búsqueda en grafo



5: A5C1B3D4E0

Búsqueda en árbol



5^+: A5 $[B3C1]^+$ D4A5

5. Conclusiones

Hemos visto:

- El algoritmo de búsqueda Primero-El-Mejor general.
- El árbol de búsqueda (Primero-El-Mejor) voraz.
- La optimalidad y complejidad de búsqueda voraz.
- La incompletitud de voraz con búsqueda en árbol.

Referencias

[1] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

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

gbfs.py

```
#!/usr/bin/env python3
import heapq
G={'A':[('B',1),('C',4)],'B':[('A',1),('D',1)],
→ 'C':[('A',4),('E',1)],'D':[('B',1),('E',4)],
→ 'E':[('C',1),('D',4)]}
hstar={'A':5,'B':5,'C':1,'D':4,'E':0}
def bf(G,s,t,f):
→fs=f[s]; Od={s:(0,fs)}; Cd={} # Open and Closed g,f dict
→Oh=[]; heapq.heappush(Oh,(fs,s,[s])) # Open heap
→while Od:
→→s=None
→→while s not in Od: fs,s,path=heapq.heappop(Oh) # delete-min
→→gs,fs=Od[s]
→→if s==t: return gs,path
→→del Od[s]; Cd[s]=gs,fs
→→for n,wsn in G[s]:
→→→gn=gs+wsn; fn=f[n]
→→→if n in Cd:
→→→→if fn<Cd[n][1]: del Cd[n] # for variable f (eg with path)
→→→→else: continue
→→→elif n in Od and fn>=Od[n][0]: continue
→→→Od[n]=gn,fn; heapq.heappush(Oh,(fn,n,path+[n]))
print(bf(G,'A','E',hstar))
```

gbfs.py.out

```
(5, ['A', 'C', 'E'])
```