Búsqueda Heurística II

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain
pedro@iiia.csic.es

Algoritmos

- Algoritmo: procedimiento computacional que termina
  - si en algún caso no termina, *hay que* especificarlo
- Características: algoritmo $A(x) \rightarrow y \mid fallo$
  - correcto: y es lo que A dice que es
  - completo:
    - satisfacción: y es solución
    - optimización: y es la solución óptima

Complejidad

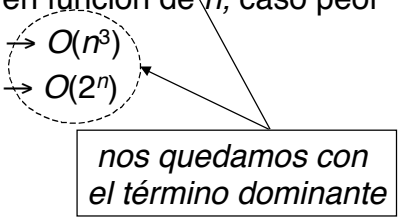
Algoritmo $A(x)$, $|x| = n$

Coste en tiempo: pasos que da A en función de n, caso peor

- polinomio de n: $n^2 + 2n + 3 \rightarrow O(n^2)$
- exponencial: $3^n + n\log(n) \rightarrow O(3^n)$

Coste en espacio: memoria de A en función de n, caso peor

- polinomio de n: $2n^3 + 4n \rightarrow O(n^3)$
- exponencial: $2^n + n\log(n) \rightarrow O(2^n)$



Complejidad caso peor / caso medio

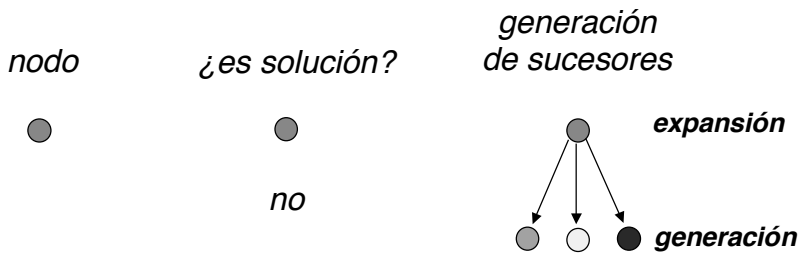
- Caso peor: peor opción en tiempo / memoria
- Caso medio: coste medio en ejemplares reales
- Si caso peor no es frecuente:
  - complejidad caso peor no es una estimación realista y... ... una complejidad alta no nos asusta
- Si el caso peor cercano al caso medio:
  - la función de complejidad va a determinar la bondad del algoritmo

Búsqueda ciega

- Búsqueda ciega (fuerza bruta): sin función heurística
- Algoritmos:
  - Busqueda en anchura (BFS)
  - Búsqueda en profundidad (DFS)
  - Profundización iterativa (ID)
- Evaluación:
  - calidad solución: ¿óptimo?
  - coste en tiempo: proporcional a los nodos generados
  - coste en memoria: proporcional a los nodos almacenados

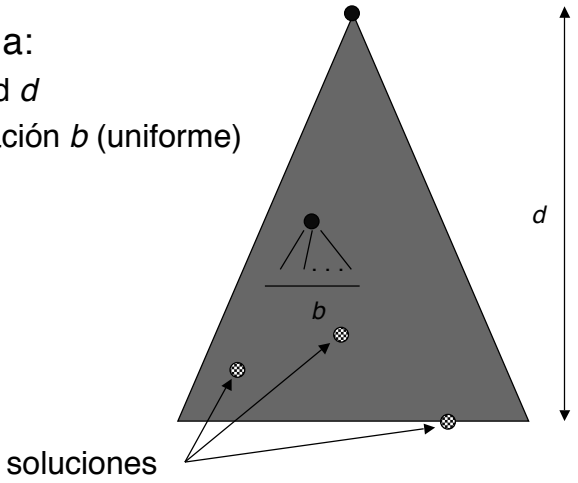
Nodo: operaciones

- Generación: cuando se crea
- Expansión: cuando se generan sus sucesores



Búsqueda en árbol

Árbol de búsqueda:
- finito: profundidad *d*
- factor de ramificación *b* (uniforme)



Esquema de búsqueda ciega

1. *L* ← lista de nodos iniciales del problema.
   L contiene la lista de nodos no visitados.
2. Si *L* vacía, fallo, stop.
   Sino, *n* ← extrae un nodo de *L* (*n* se elimina de *L*)
3. Generar los sucesores de *n*. Si un sucesor es solución, retornar el camino desde la raíz, stop.
4. Sino, añadir a *L* los sucesores de *n*, etiquetando sus respectivos caminos desde la raíz. Ir a 2.

Opciones**:**
- extrae un nodo de *L* *¿al principio? ¿al final ?*
- añadir a *L* *¿al principio? ¿al final ?*
- sucesores de *n* *¿todos? ¿unos pocos?*

Búsqueda en anchura

Algoritmo BFS (*breadth-first search*)

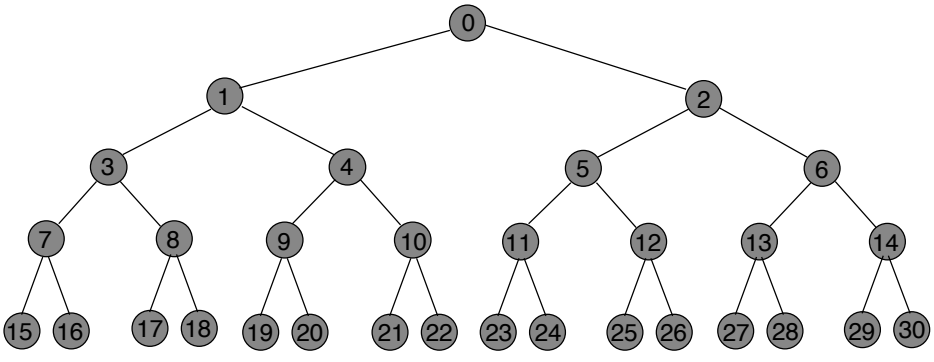
1. Lista $L \leftarrow$ nodo raíz

2. Si L vacía, fallo, stop.
   Sino, $n \leftarrow$ extrae-primero(L).

3. Generar los sucesores de n. Si alguno es solución, retornar el camino desde la raíz, stop.

4. Sino, añadir al final de L todos los sucesores de n, etiquetando cada sucesor con su camino desde la raíz. Ir a 2.

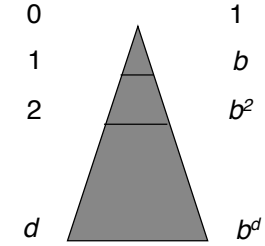
Búsqueda en anchura (II)



orden de generación

Búsqueda en anchura (III)

- Calidad: se visita por niveles → encuentra la solución más cercana a la raíz
- Tiempo: proporcional a los nodos generados



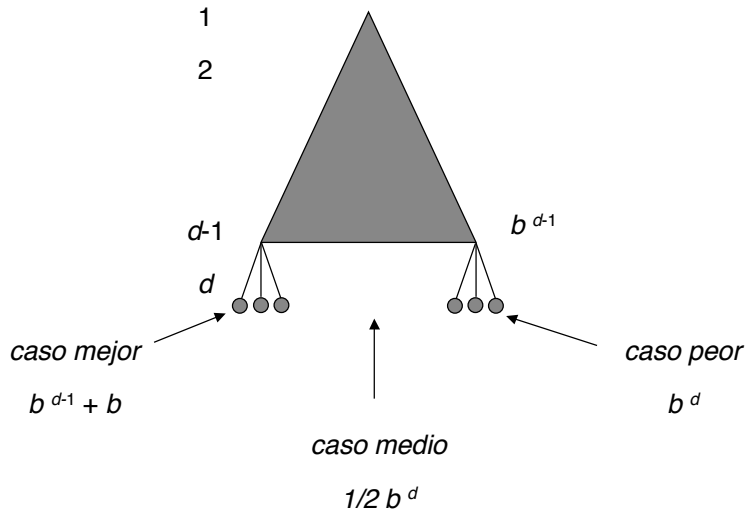
$$1 + b + b^2 + ... + b^d = \frac{b^{d+1} - 1}{b - 1} \approx \frac{b^{d+1}}{b - 1}$$

Complejidad temporal: $O(b^d)$

- Espacio: nodos memorizados b^d

Complejidad espacial: $O(b^d)$

Búsqueda en anchura IV



Complejidad espacial exponencial

¡¡  !!

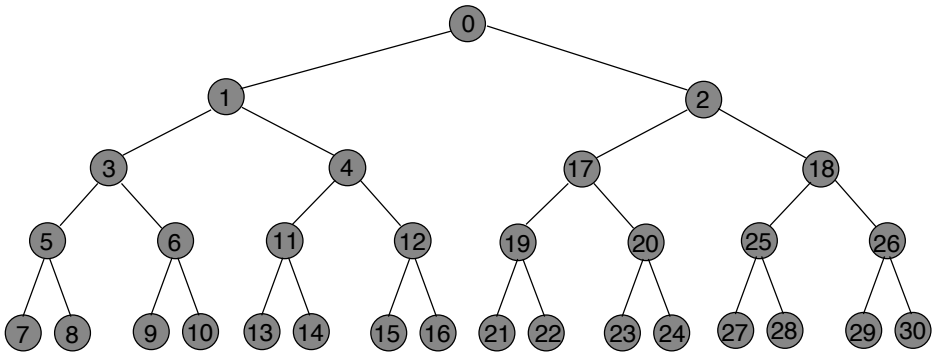
- Velocidad generación nodos: 10^7 por seg
- Memoria disponible: $2{,}5 \times 10^8$ nodos
- Si guardamos todos los nodos: 25 seg
- Si guardamos solo el último nivel:
  - en cuanto $b^d > 2{,}5\ 10^8$ → *memory overflow*
  - en la práctica, cuestion de minutos

Búsqueda en profundidad

Algoritmo DFS (*depth-first search*):

1. Lista $L \leftarrow$ nodo raíz
2. Si L vacía, fallo, stop.
   Sino, $n \leftarrow$ extrae-primero(L).
3. Generar los sucesores de n. Si alguno es solución, retornar el camino deste la raíz, stop.
4. Sino, añadir al principio de L todos los sucesores de n, etiquetando cada sucesor con su camino desde la raíz. Ir a 2.

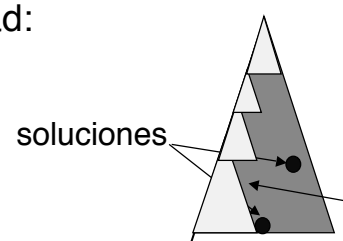
Búsqueda en profundidad (II)



orden de generación

Búsqueda en profundidad (III)

- Calidad:



Supongamos que DFS no se pierde en ramas infinitas

soluciones

zona visitada por DFS

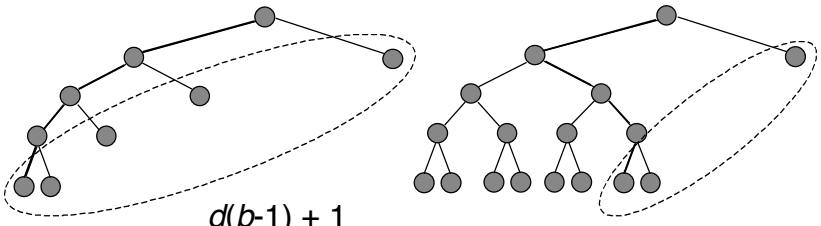
rama infinita

DFS encuentra una solución, pero no es la más cercana a la raíz

DFS no encuentra solución

Búsqueda en profundidad (IV)

- Espacio: en cada momento, 1 sola rama en memoria



$d(b\text{-}1) + 1$

Complejidad espacial: $O(bd)$

- Tiempo: nodos generados igual que BFS

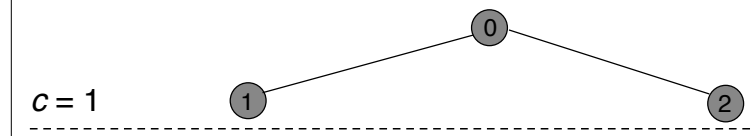
Complejidad temporal: $O(b^d)$

Profundización iterativa

Algoritmo ID (*iterative deepening*):

1. Cota profundidad $c \leftarrow 1$
2. Lista $L \leftarrow$ nodo raíz.
3. Si L vacia, $c \leftarrow c + 1$, ir a 2.
   Sino, $n \leftarrow$ *extrae-primero*(L).
4. Si *profundidad*(n) < c, generar los sucesores de n.
   Si alguno es solución, retornar el camino desde la raíz. Stop.
5. Sino, añadir al principio de L los sucesores de n,
   etiquetando cada sucesor con su camino desde la raíz.
   En cualquier caso, ir a 3.

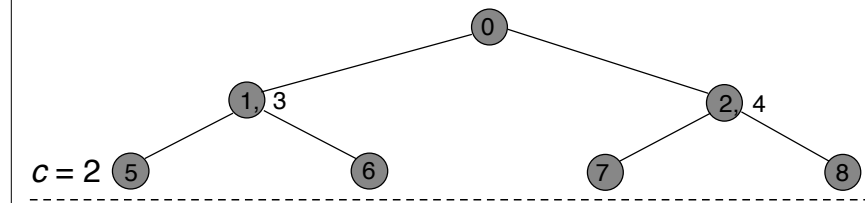
Profundización iterativa (II)



$c = 1$

orden de generación

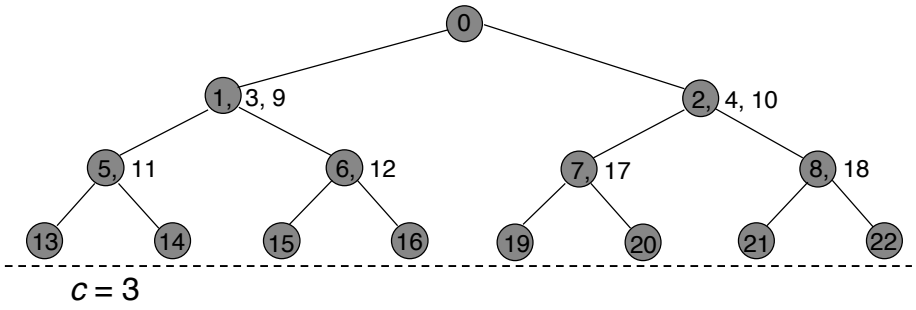
Profundización iterativa (II)



$c = 2$

orden de generación

Profundización iterativa (II)



orden de generación

Profundización iterativa (III)

- Calidad: solución más cercana a la raíz
- Espacio: es DFS, $d(b\text{-}1)+1$ Complejidad espacial $O(bd)$
- Tiempo: d búsquedas DFS

$$b^d \frac{b}{b-1} + b^{d-1} \frac{b}{b-1} + \ldots + b \frac{b}{b-1} + \frac{b}{b-1} =$$

$$b^d \left(\frac{b}{b-1} \right)^2$$

Complejidad temporal $O(b^d)$

Comparación ID y DFS

- Complejidad espacial igual
- Complejidad temporal:
  - asintóticamente (*d* grande) igual: la mayor parte del trabajo se debe a la última iteración, y el coste de las iteraciones anteriores es pequeño
  - para *b* y *d* fijos

$$\frac{\text{ID}}{\text{DFS}} = \frac{b^d \left(\frac{b}{b-1} \right)^2}{b^d \left(\frac{b}{b-1} \right)} = \frac{b}{b-1}$$

Resumen

	BFS	DFS	ID
Espacio:	exp	lineal	lineal
Tiempo:	exp	exp	exp
Problema ramas infinitas	no	sí	no
Solución más cercana	sí	no	sí

- ID repite trabajo, pero el coste de las iteraciones anteriores es pequeño comparado con la iteración que encuentra la solución.
- Si sabemos la profundidad de la solución: DFS.