Métodos de Inteligencia Artificial

L. Enrique Sucar (INAOE)

esucar@inaoep.mx

ccc.inaoep.mx/esucar

Tecnologías de Información

UPAEP

Búsqueda

- Representación
- Tipos búsquedas:
  - Sin información
  - Con información
  - Óptimas
- Análisis de complejidad

Solución de Problemas

Asociado a la inteligencia

- Identificación y definición del problema
- Identificación del criterio de evaluación
- Generación de alternativas

Solución de muchos problemas en IA: básicamente búsqueda y evaluación

Representación del espacio de estados

- define un espacio de estados (explícito / implícito)
- especifica los estados iniciales
- especifica los estados finales (metas)
- especifica las reglas que definen las acciones disponibles para moverse de un estado a otro

Representación de espacio de estados

En este contexto:

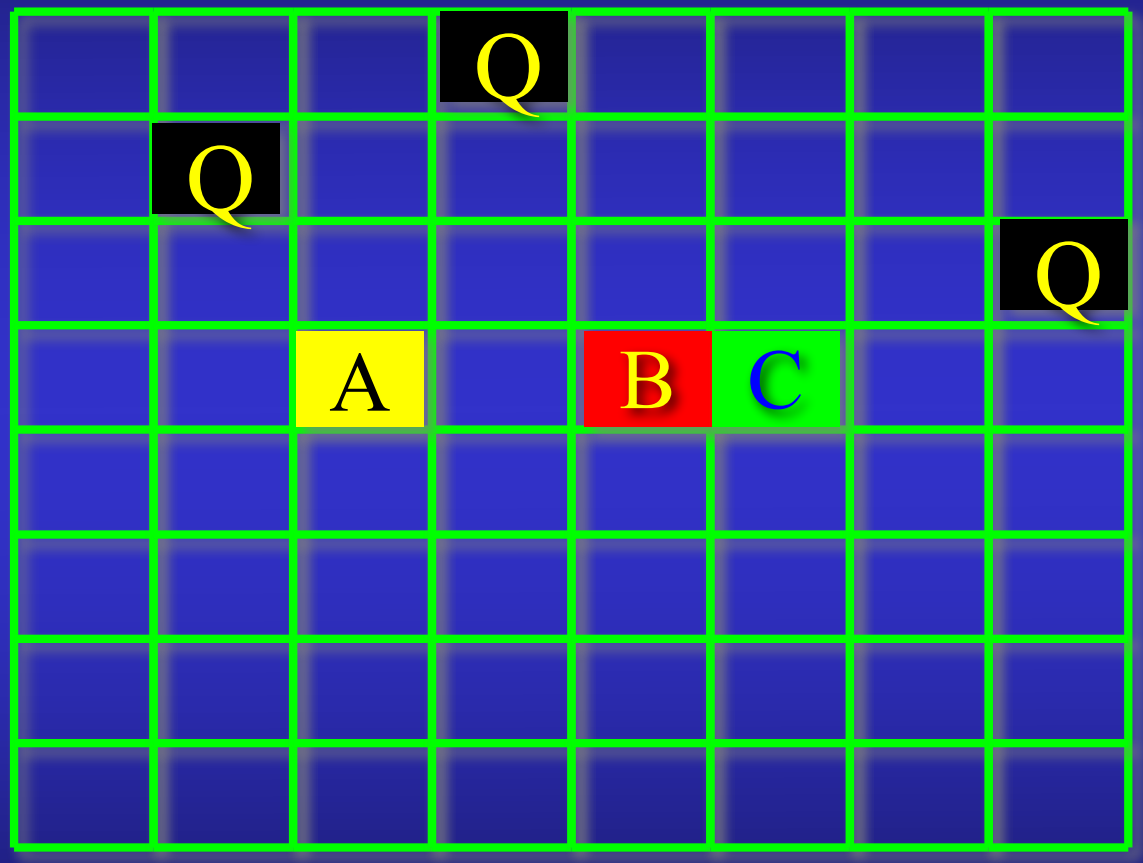
El proceso de solución de problemas = encontrar una secuencia de operaciones que transformen al estado inicial en uno final

Se requiere una estrategia de búsqueda

Ejemplo: problema de las 8 reinas

Reinas (Q)

Tablero de Ajedrez



A=8 B=9 C=10

Para el problema de las 8 reinas podemos tener diferentes opciones:

- solución incremental: acercarse a la meta haciendo decisiones locales
- sistemática: no repetir y no excluir

Medios (espacio de estados):

- transformar (hasta encontrar la meta)

vs.

- construir (poco a poco)

Posible heurística: poner una reina por renglón en un lugar que deje el mayor número de lugares sin atacar

Cómo encontramos una buena heurística?

Factores a considerar:

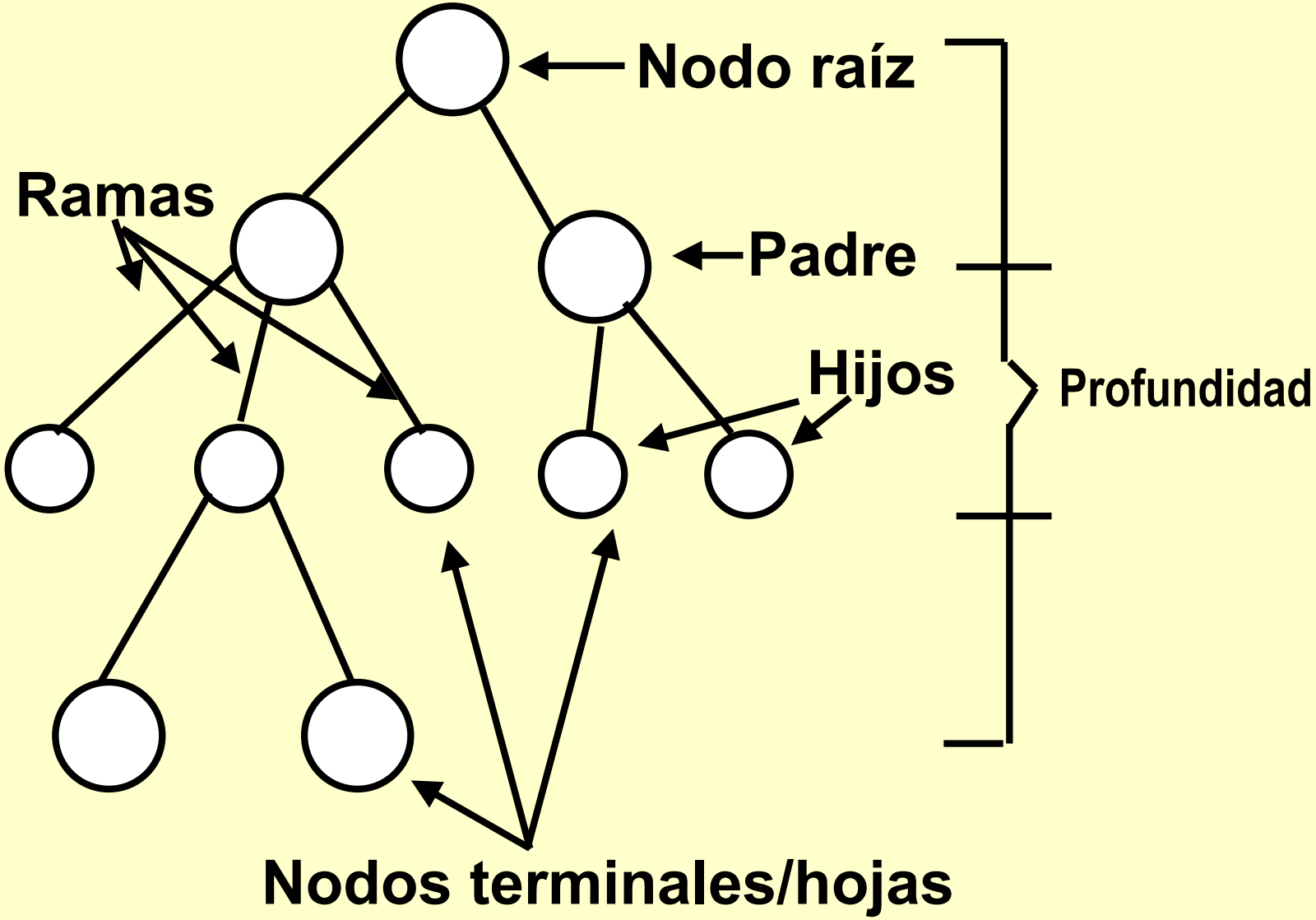
- calidad de la solución (a veces puede no importar)
- diferencia en complejidad entre una solución solución óptima
- en general, se busca encontrar la solución más barata

Qué necesitamos:

1. Estructura simbólica que represente subconjuntos de soluciones potenciales (agenda)
2. Operaciones/reglas que modifiquen símbolos de la agenda y produzcan conjuntos de soluciones potenciales más refinadas
3. Estrategia de búsqueda que decida qué operación aplicar a la agenda

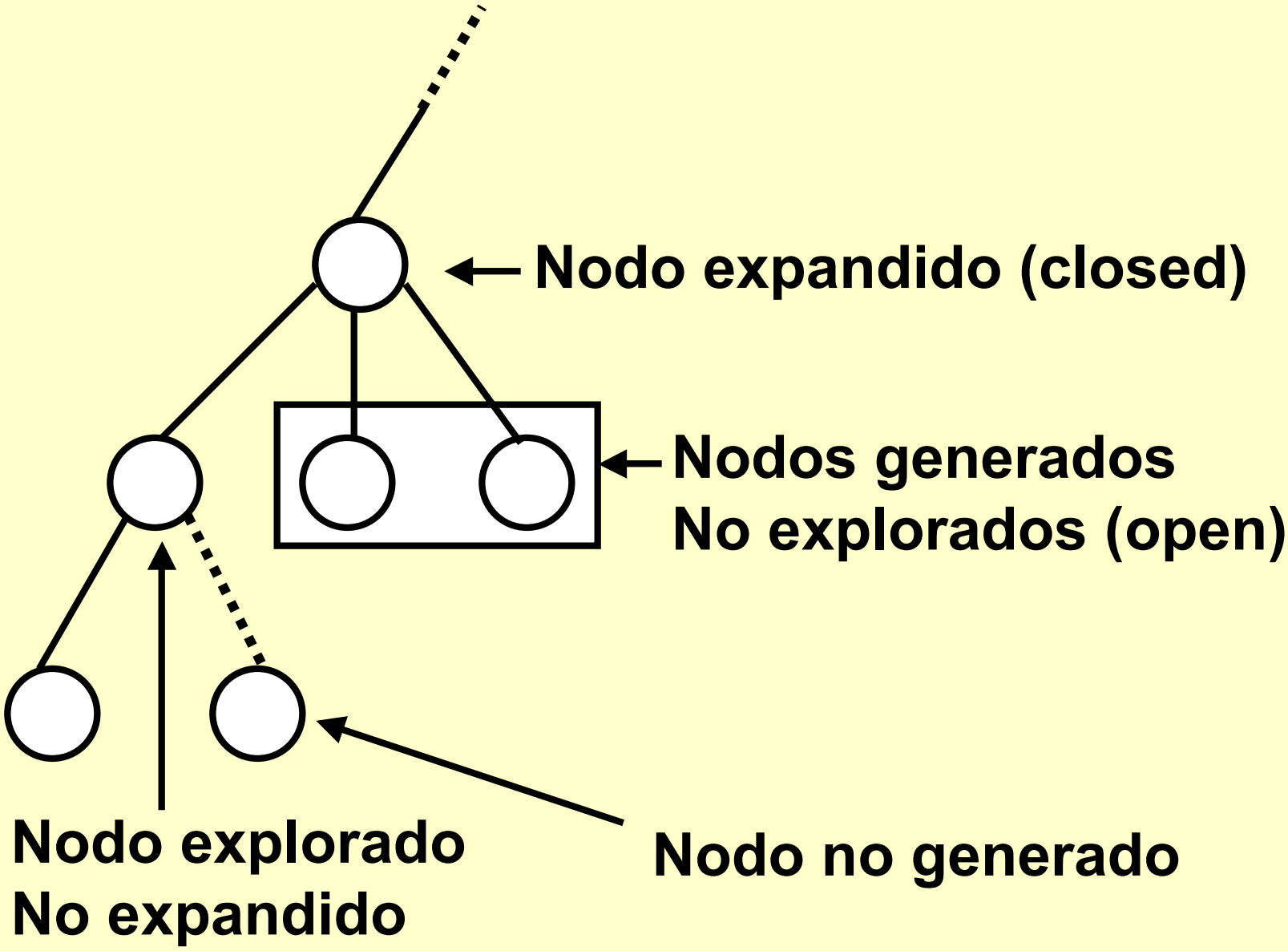
Terminología:

nodo, árbol, hoja, nodo-raíz, nodo-terminal, *branching factor* (factor de arborescencia), ramas, padres, hijos, árbol uniforme, ...


Nodo raíz
Ramas
Padre
Hijos
Profundidad
Nodos terminales/hojas

- nodos expandidos (*closed*):
  todos los sucesores
- nodos explorados pero no expandidos :
  sólo algunos sucesores
- nodos generados pero no explorados
  (*open*)
- nodos no generados

Paso computacional primordial:
expansión de nodos


Nodo expandido (closed)
Nodos generados
No explorados (open)
Nodo explorado
No expandido
Nodo no generado

Propiedades

La estrategia de control es *sistemática* si:

1. no deja un sólo camino sin explorar (completo)

2. no explora un mismo camino más de una vez (eficiencia)

Propiedades de algoritmos de búsqueda (heurísticas):

1. *Completo*: si encuentra una solución cuando ésta existe

2. *Admisible*: si garantiza regresar una solución óptima cuando ésta existe

Propiedades de algoritmos de búsqueda (heurísticas):

3. *Dominante*: un algoritmo *A1* se dice que domina a *A2* si todo nodo expandido por *A1* es también expandido por *A2*
("más eficiente que")

4. *Óptimo*: un algoritmo es óptimo sobre una clase de algoritmos si domina a todos los miembros de la clase

Procedimientos de Búsqueda

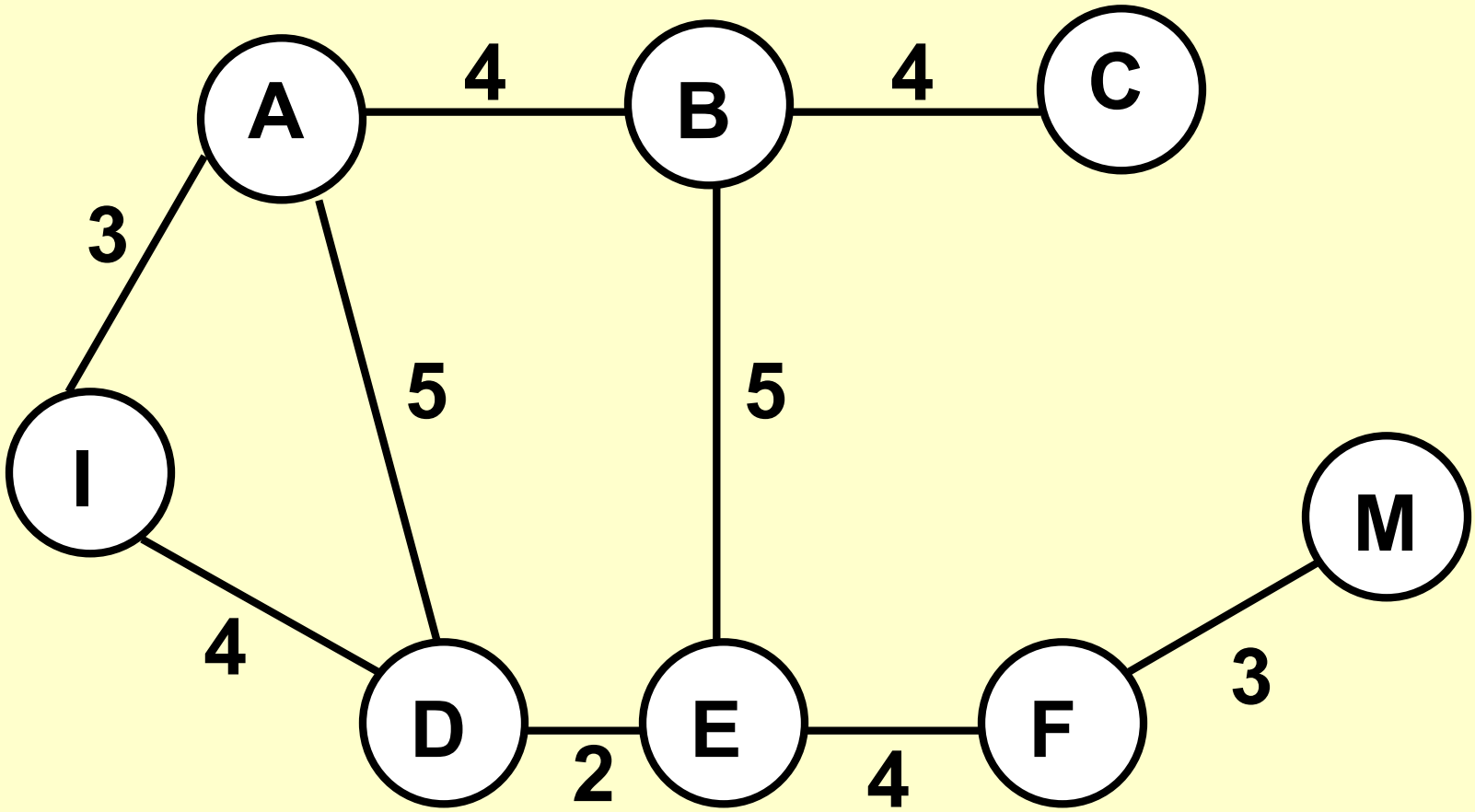
Algún camino:

- Sin información:
  - depth-first (en profundo)
  - breadth-first (a lo ancho)

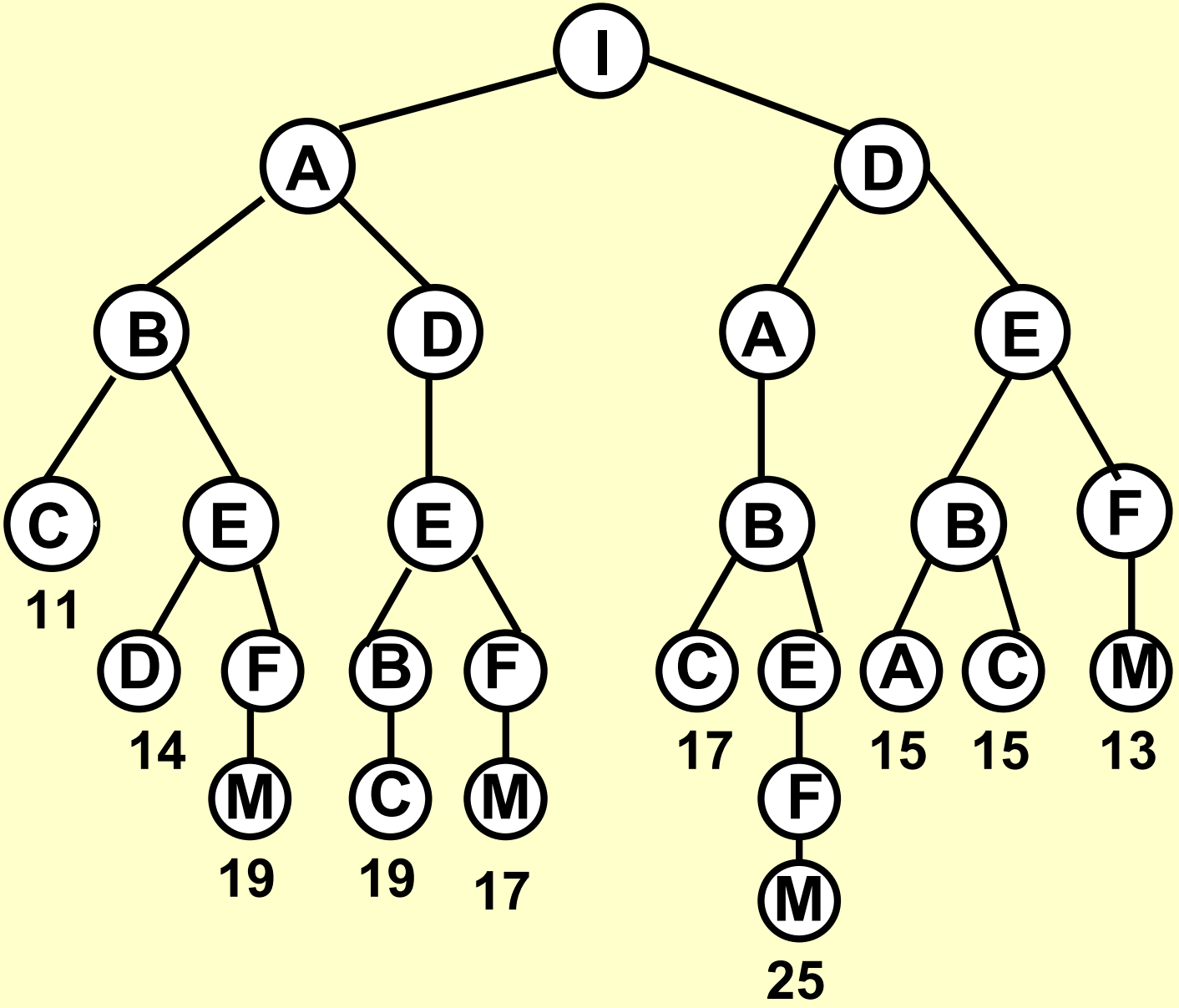
- Con información:
  - hill climbing
  - beam search
  - best first

- El mejor camino (óptimo):
  - British museum
  - branch and bound
  - A*

GRAFO



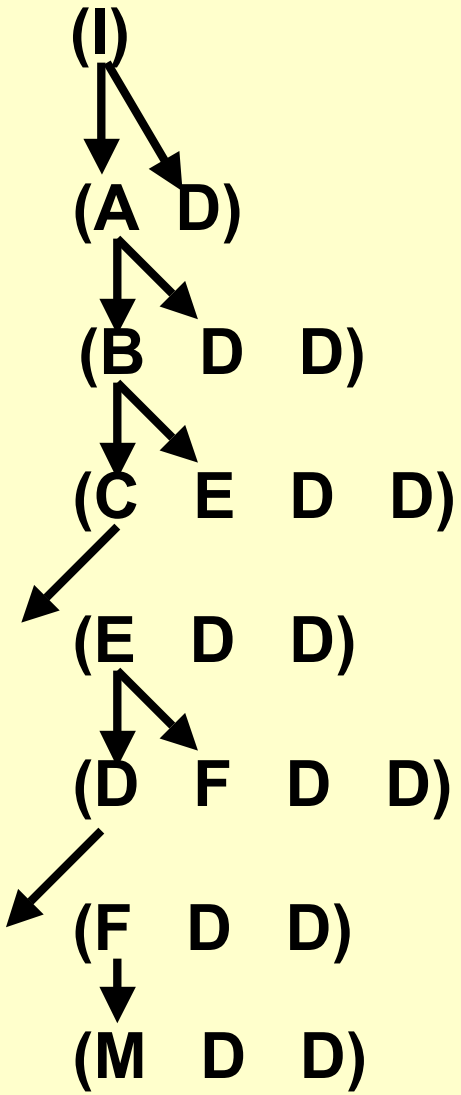
ÁRBOL DE BÚSQUEDA

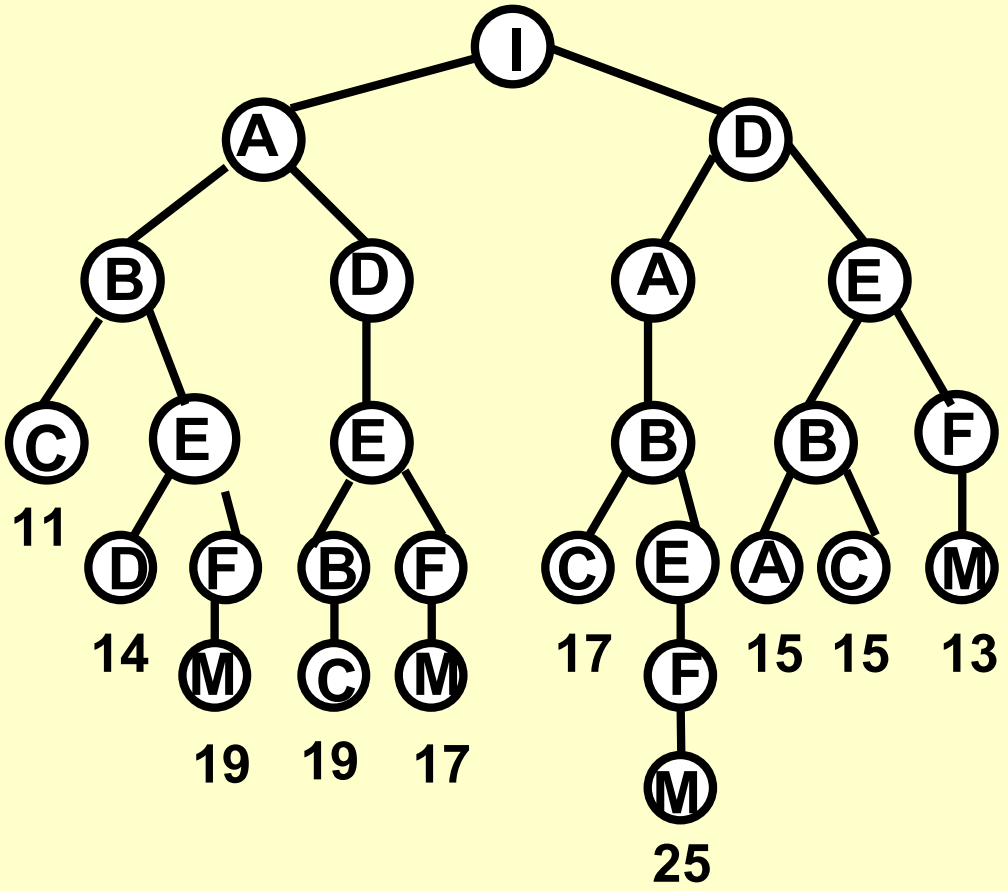


Depth first - backtracking (LIFO)

Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta
si el primer elemento es la meta
entonces acaba
si no elimina el primer elemento y
añade sus sucesores al *frente* de la agenda

DEPTH-FIRST SEARCH





Problemas:

árboles con caminos de profundidad muy grande

Variaciones:

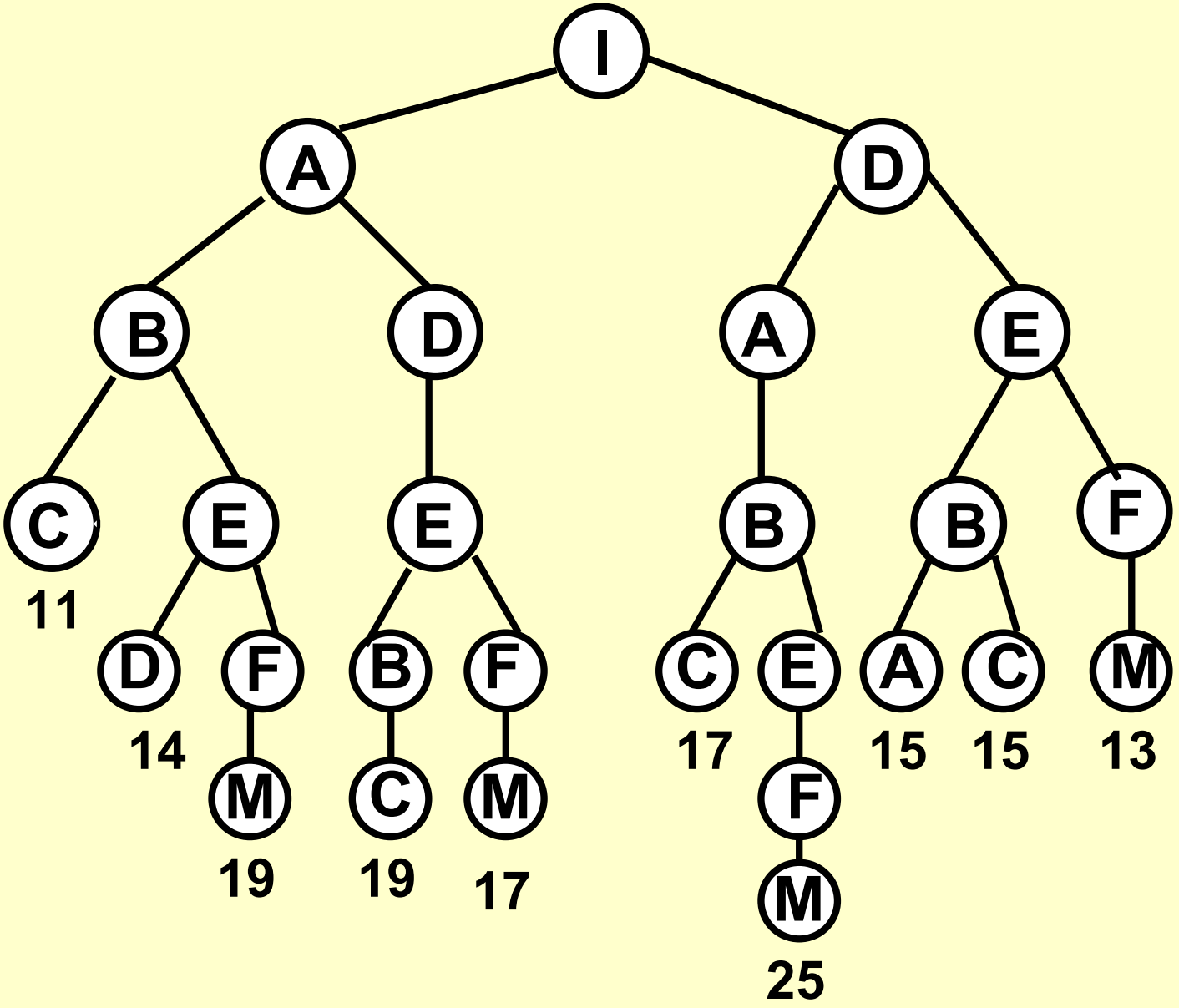
- depth-bound (casi todos): limitar la búsqueda hasta cierto límite de profundidad
- interative-deepening: explorar a profundidad progresivamente
- con algo de información: ordena los nodos expandidos

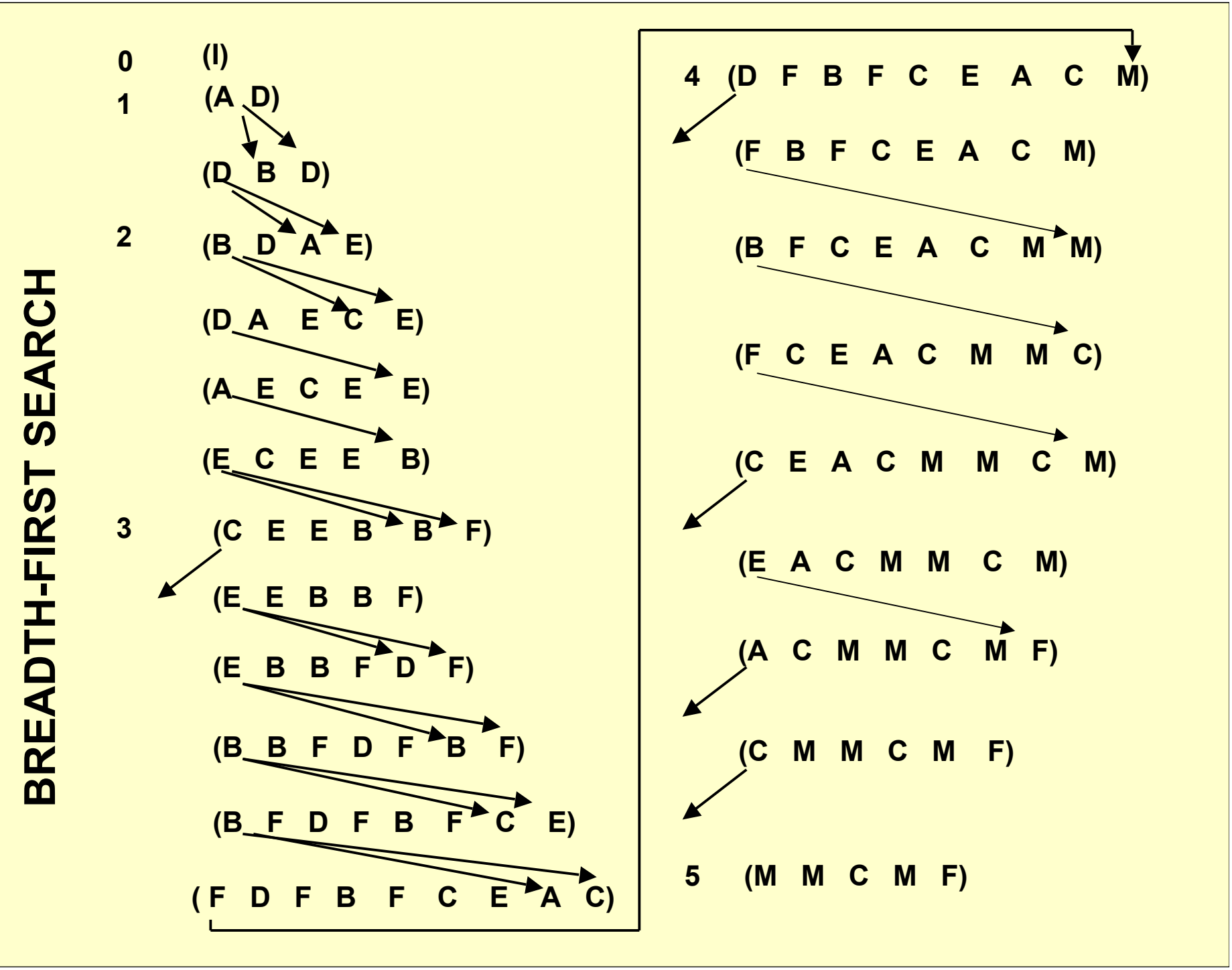
Breadth first

Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta
si el primer elemento es la meta
entonces acaba
si no elimina el primer elemento y
añade sus sucesores al *final* de la agenda

Problemas: árboles con arborescencia muy grande

ÁRBOL DE BÚSQUEDA




BREADTH-FIRST SEARCH
0 (I)
1 (A D)
(D B D)
2 (B D A E)
(D A E C E)
(A E C E E)
(E C E E B)
3 (C E E B B F)
(E E B B F)
(E B B F D F)
(B B F D F B F)
(B F D F B F C E)
(F D F B F C E A C)
4 (D F B F C E A C M)
(F B F C E A C M)
(B F C E A C M M)
(F C E A C M M C)
(C E A C M M C M)
(E A C M M C M)
(A C M M C M F)
(C M M C M F)
5 (M M C M F)

Profund.	Nodos	Tiemp	Memoria
0	1	1 miliseg.	100 bytes
2	111	.1 seg.	11 kilobytes
4	11,111	11 seg.	1 megabyte
6	10^6	18 min.	111 megabytes
8	10^8	31 hr.	11 gigabytes
10	10^{10}	128 días	1 terabyte
12	10^{12}	35 años	111 terabytes
14	10^{14}	3500 años	11,111 terabytes

Requerimientos de tiempo y memoria para breadth-first.
Factor de arborecencia = 10; 1,000 nodos/sec; 100 bytes/nodo

Complejidad

Comparación en nodos buscados:
Si n = profundidad del árbol
b = braching factor
d = profundidad de un nodo meta

depth-first:

- mejor caso: d nodos buscados
- peor caso:

$$\sum_{i=0}^{n} b^i - \sum_{i=0}^{n-d} b^i = \frac{b^{n+1} - b^{n+1-d}}{b-1} \approx b^n$$

breadth-first:

- mejor caso:

$$\sum_{i=0}^{d-1} b^i = \frac{b^d - 1}{b-1} \approx b^d - 1$$

- peor caso:

$$\sum_{i=0}^{d} b^i = \frac{b^{d+1} - 1}{b-1} \approx b^d$$

Criterio	Breadth first	Costo uniforme	Depth first	Depth limited	Iterative deepening	Bidireccional
Tiempo	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Espacio	b^d	b^d	$b \times m$	$b \times l$	$b \times d$	$b^{d/2}$
Optimo	si	si	no	no	si	si
Completo	si	si	si	si(si $l \geq d$)	si	si

Comparación de estrategias.

b = factor de arborecencia;

d = profundidad solución;

m = máxima profundidad árbol;

l = límite de profundidad.

Algoritmos con Información

Hill-Climbing

Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta
si el primer elemento es la meta
entonces acaba
si no elimina el primer elemento y añade sus sucesores a la agenda
ordena todos los elementos de la agenda
selecciona el mejor y *elimina el resto*

BÚSQUEDA HILL-CLIMBING

Hill climbing

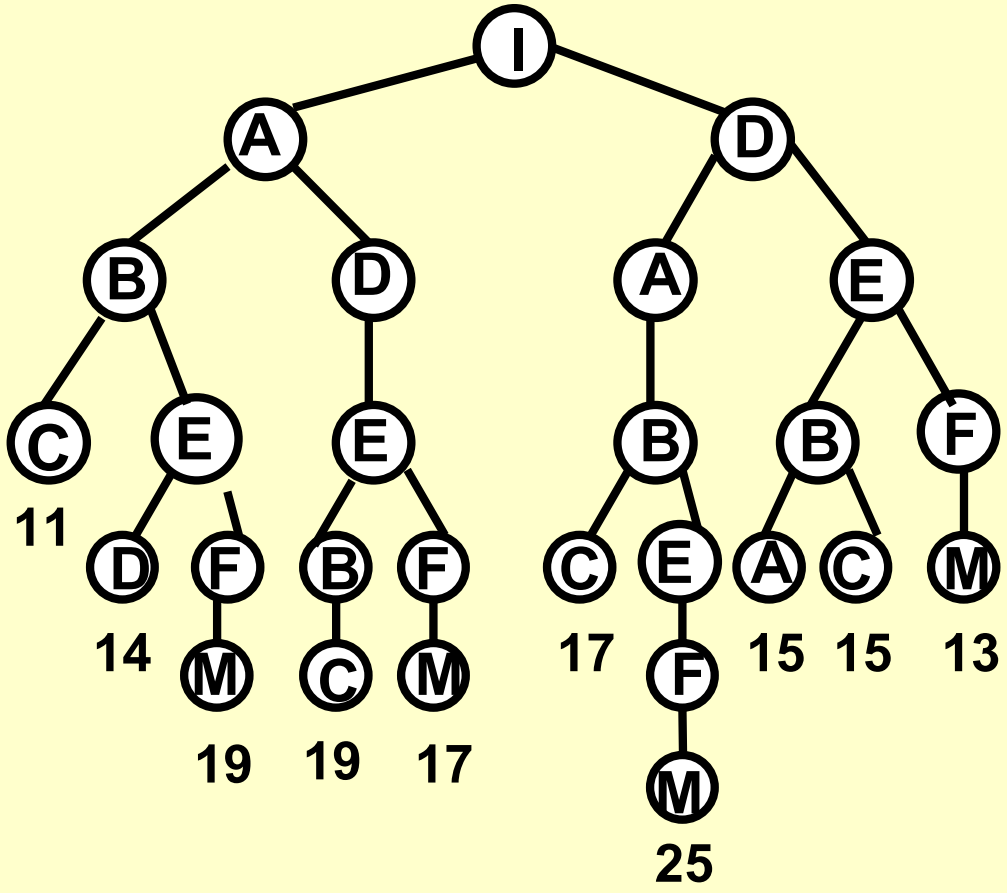
Heurística: ve a la ciudad más cercana

(I)

(A ~~D~~)

(~~D~~ B)

(~~E~~ C)



Best-first

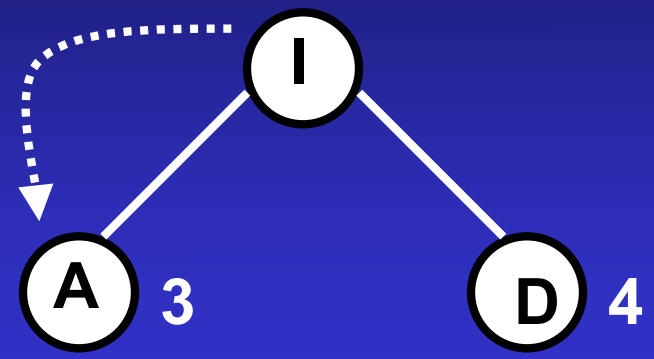
Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta
si el primer elemento es la meta
entonces acaba
si no elimina el primer elemento y añade sus sucesores a la agenda
ordena todos los elementos de la agenda

EJEMPLO BÚSQUEDA BEST-FIRST

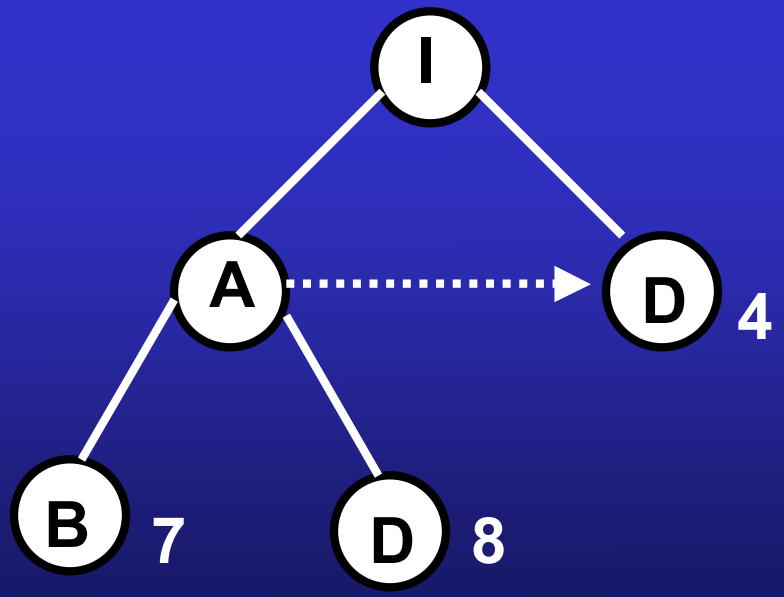
BEST FIRST

Heurística:Distancia acumulada más corta

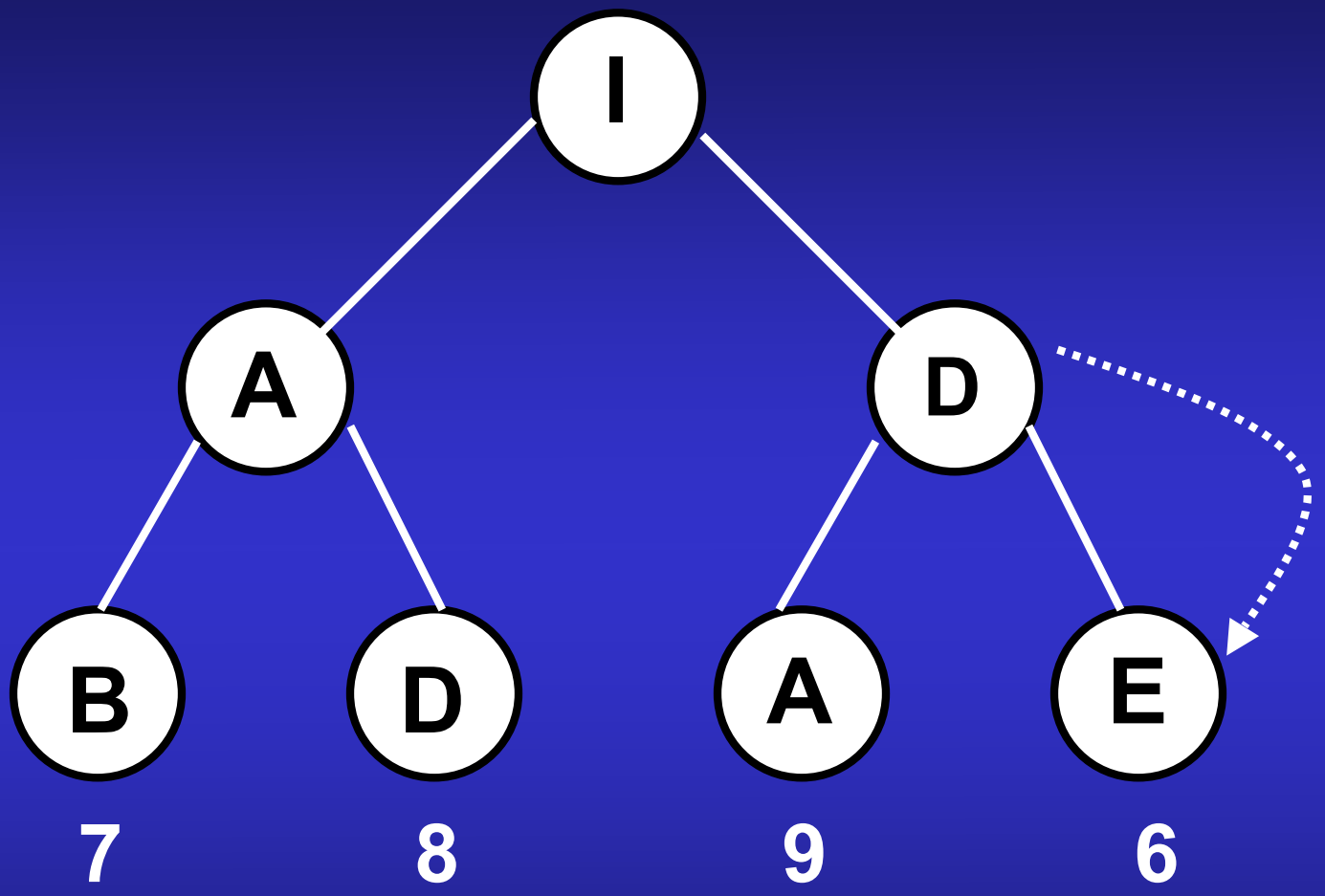
1

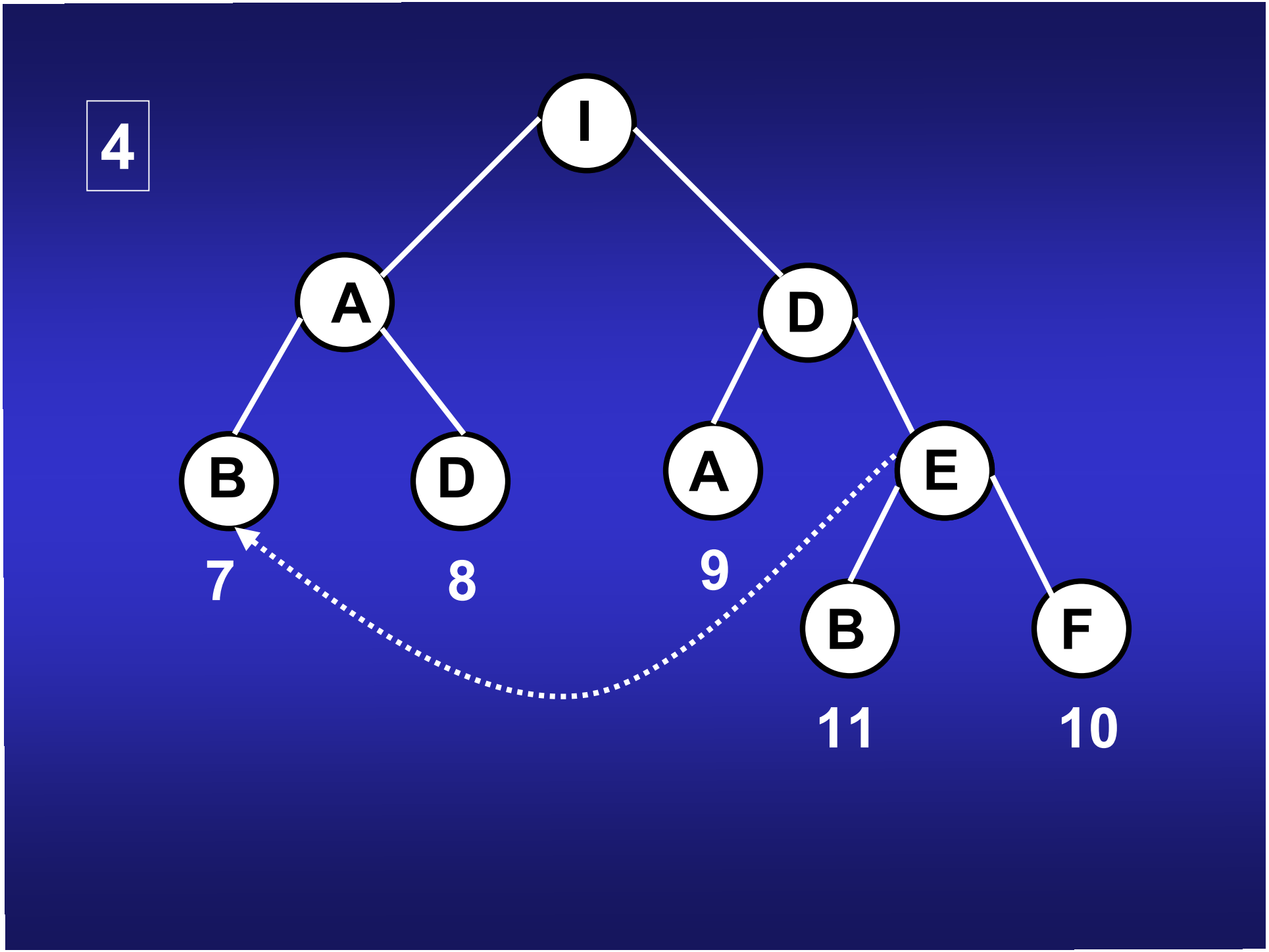


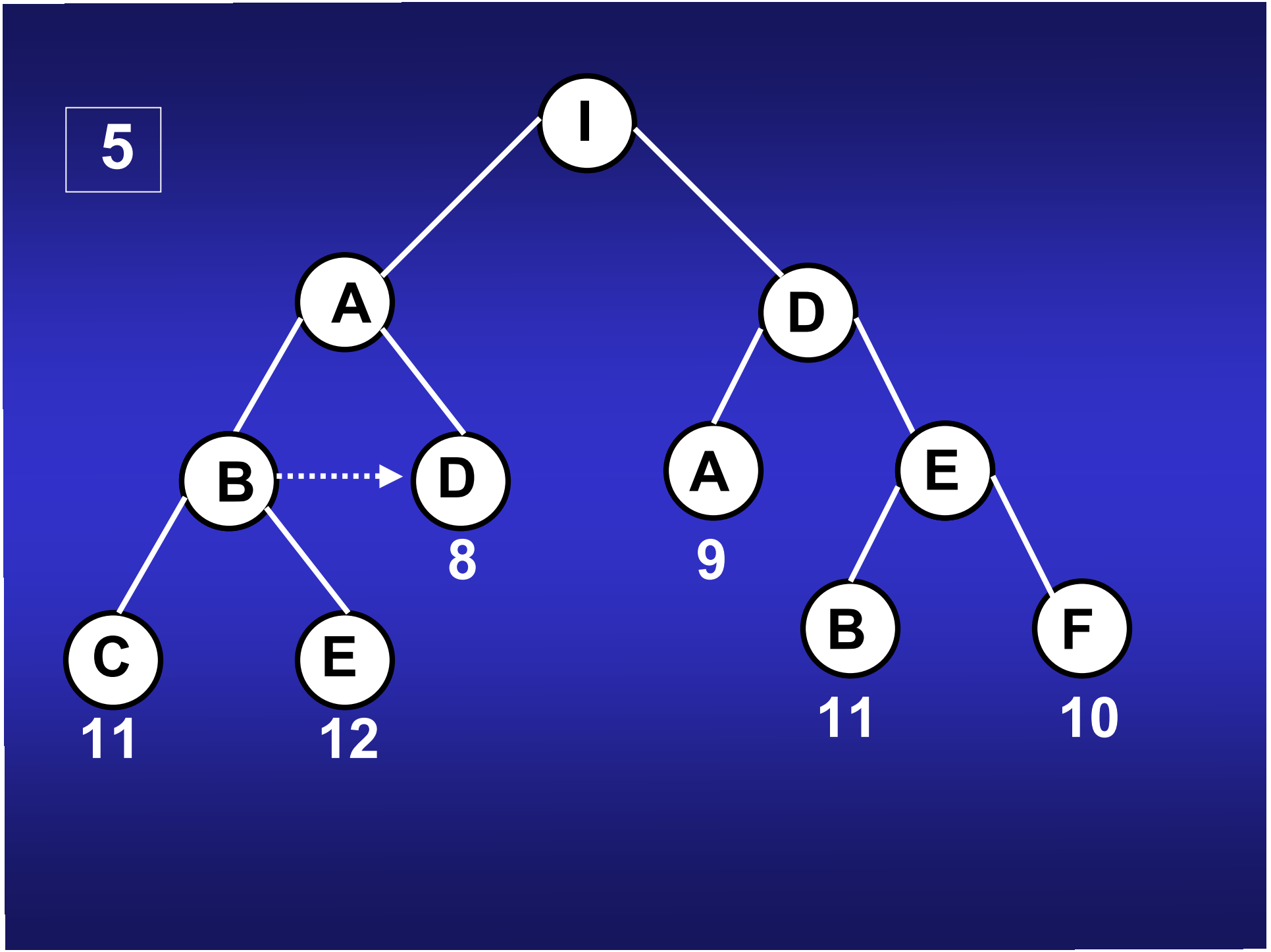
2

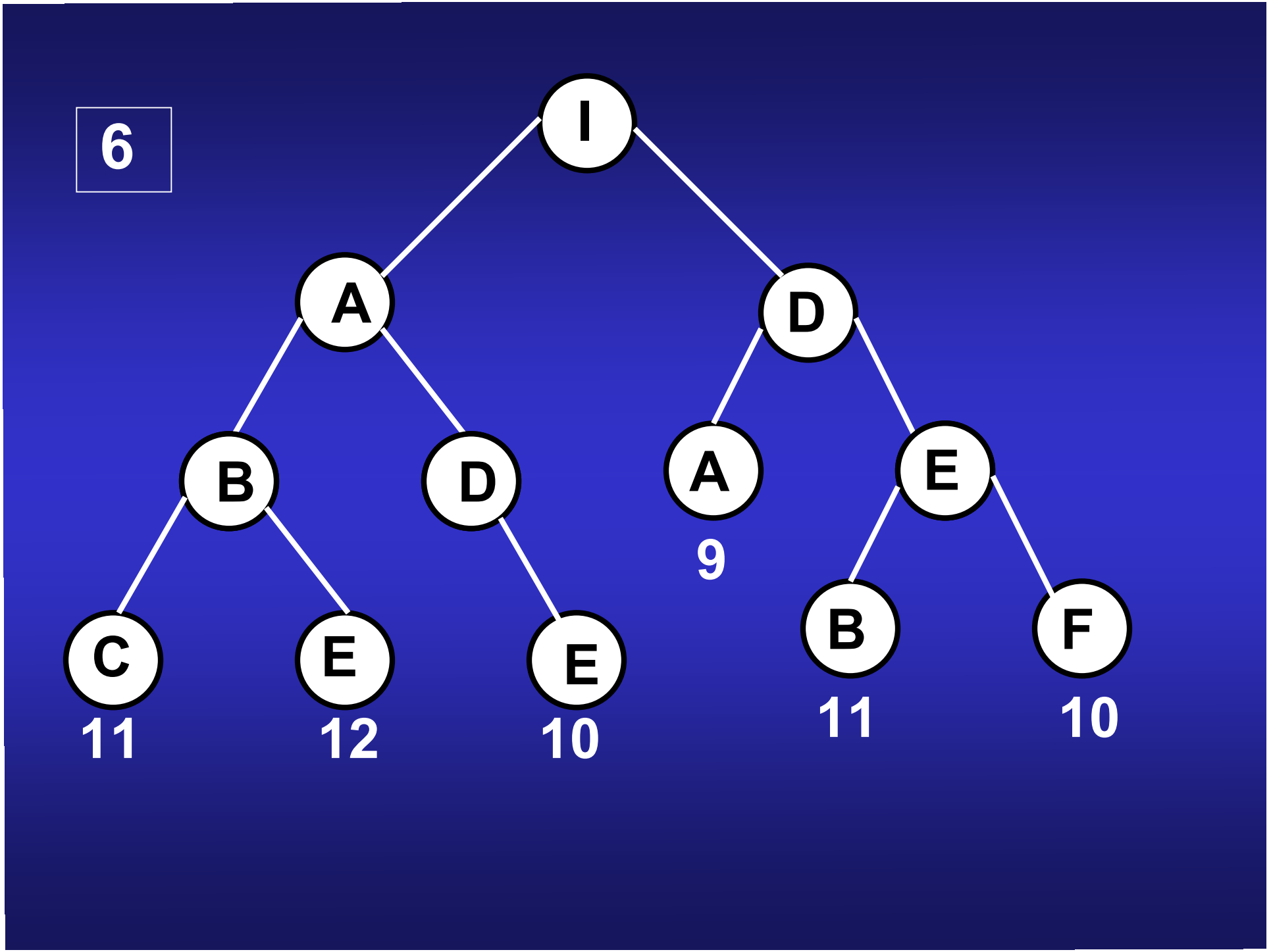


3

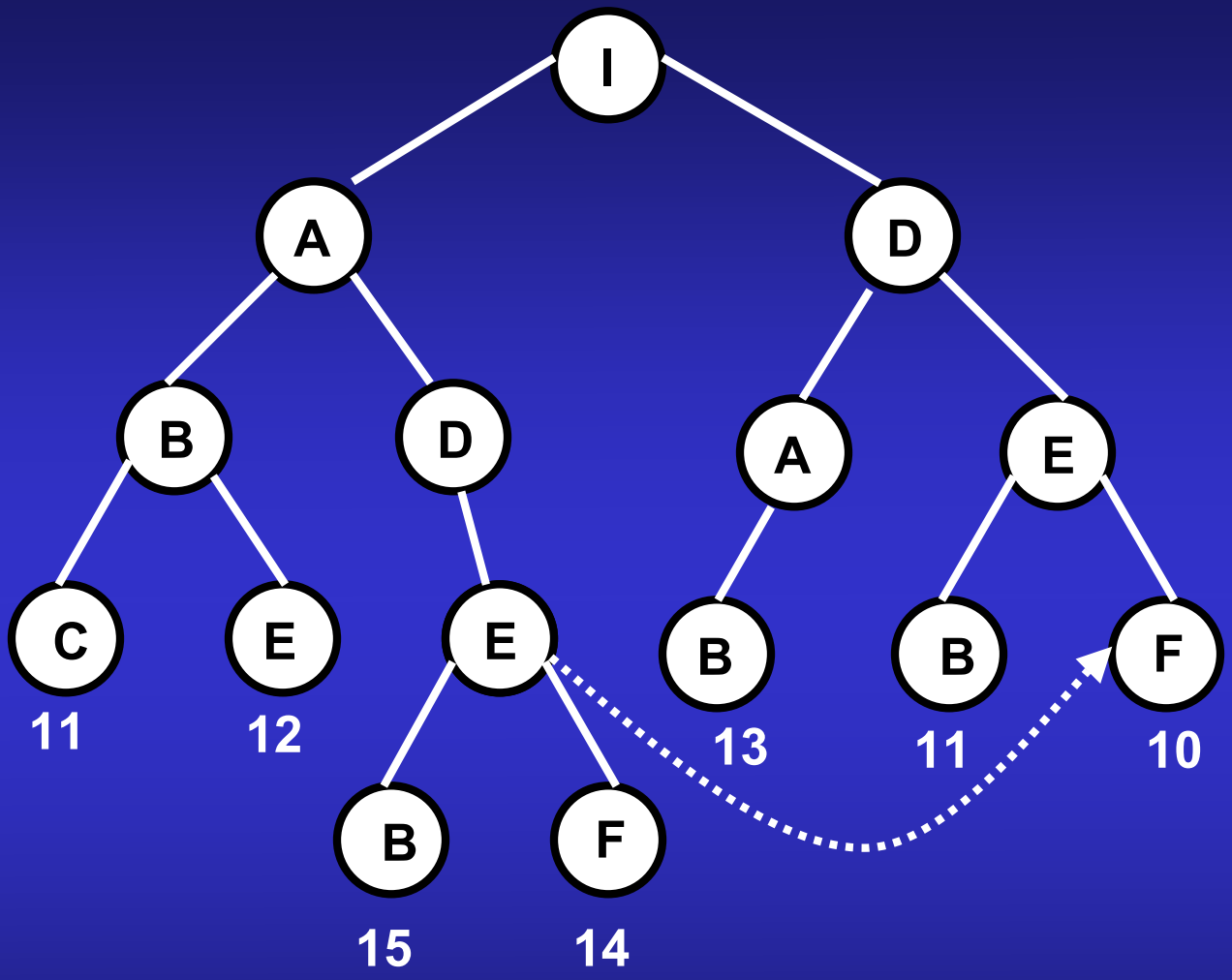



4
I
A
D
B
D
A
E
7
8
9
B
F
11
10

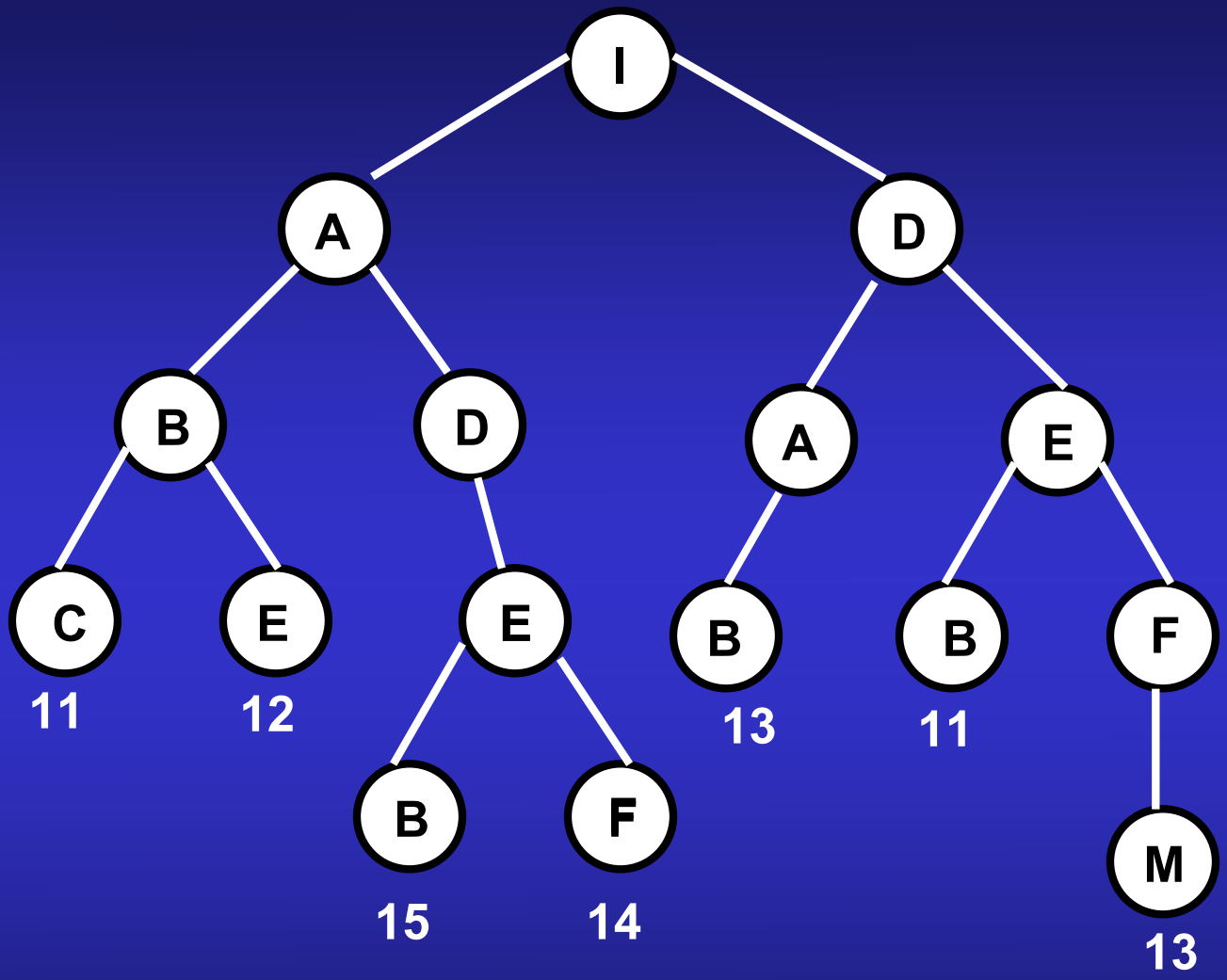

5
I
A
D
B
D
A
E
C
E
B
F
8
9
11
12
11
10


6
I
A
D
B
D
A
E
C
E
E
B
F
9
11
12
10
11
10

7



8

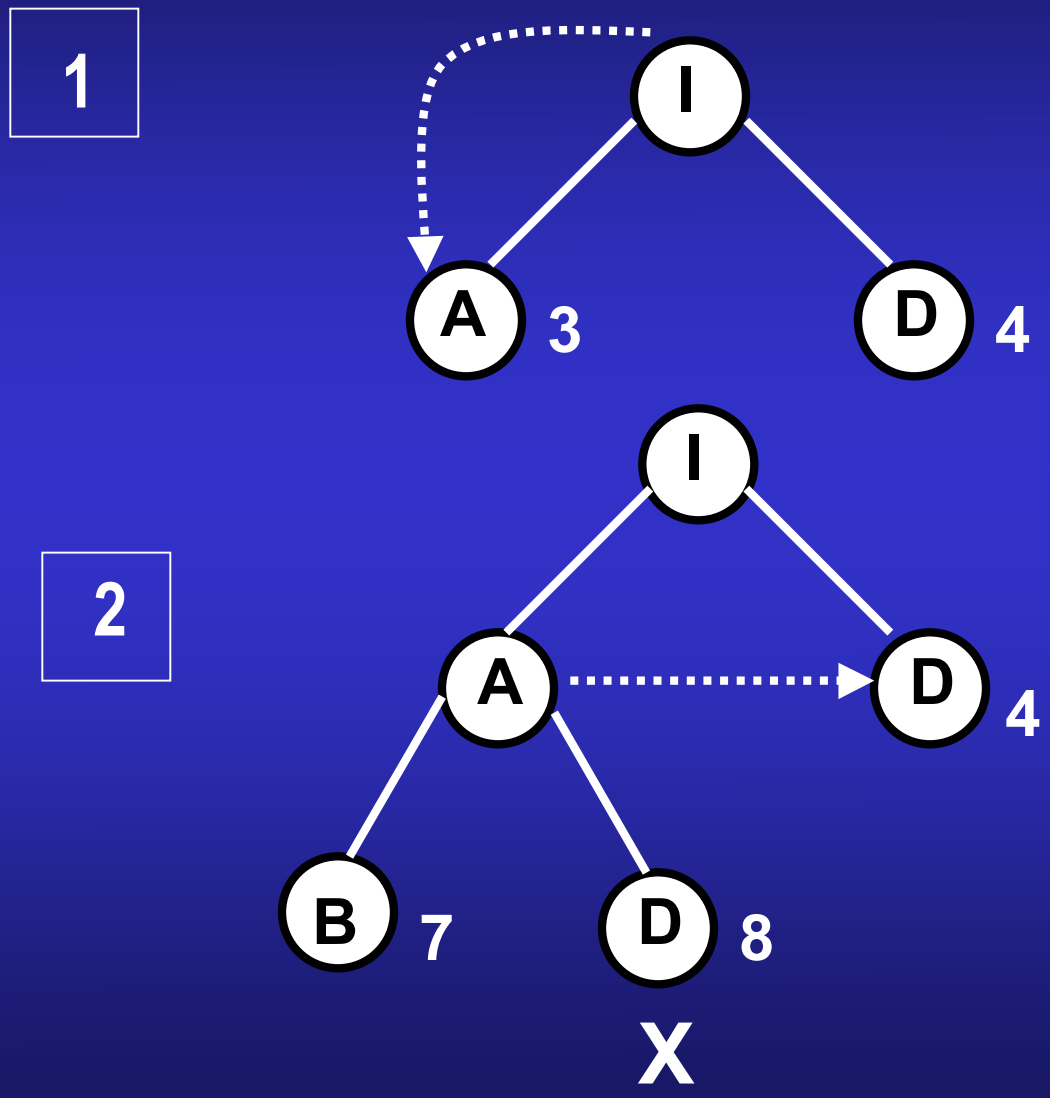


Beam search

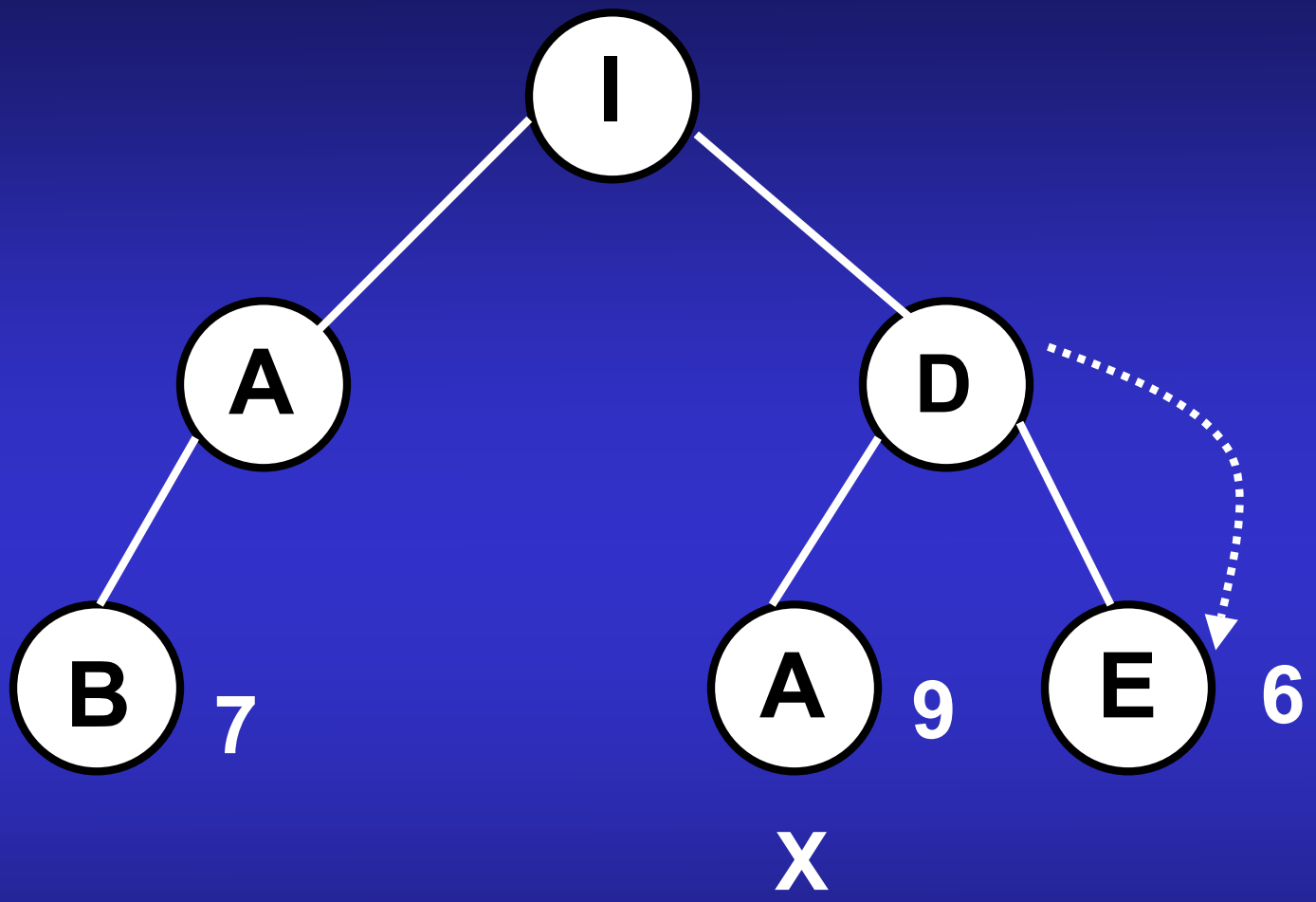
Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta
si el primer elemento es la meta
entonces acaba
si no elimina el primer elemento y añade sus sucesores a la agenda
ordena todos los elementos de la agenda y selecciona los *N* mejores (los demás eliminalos)

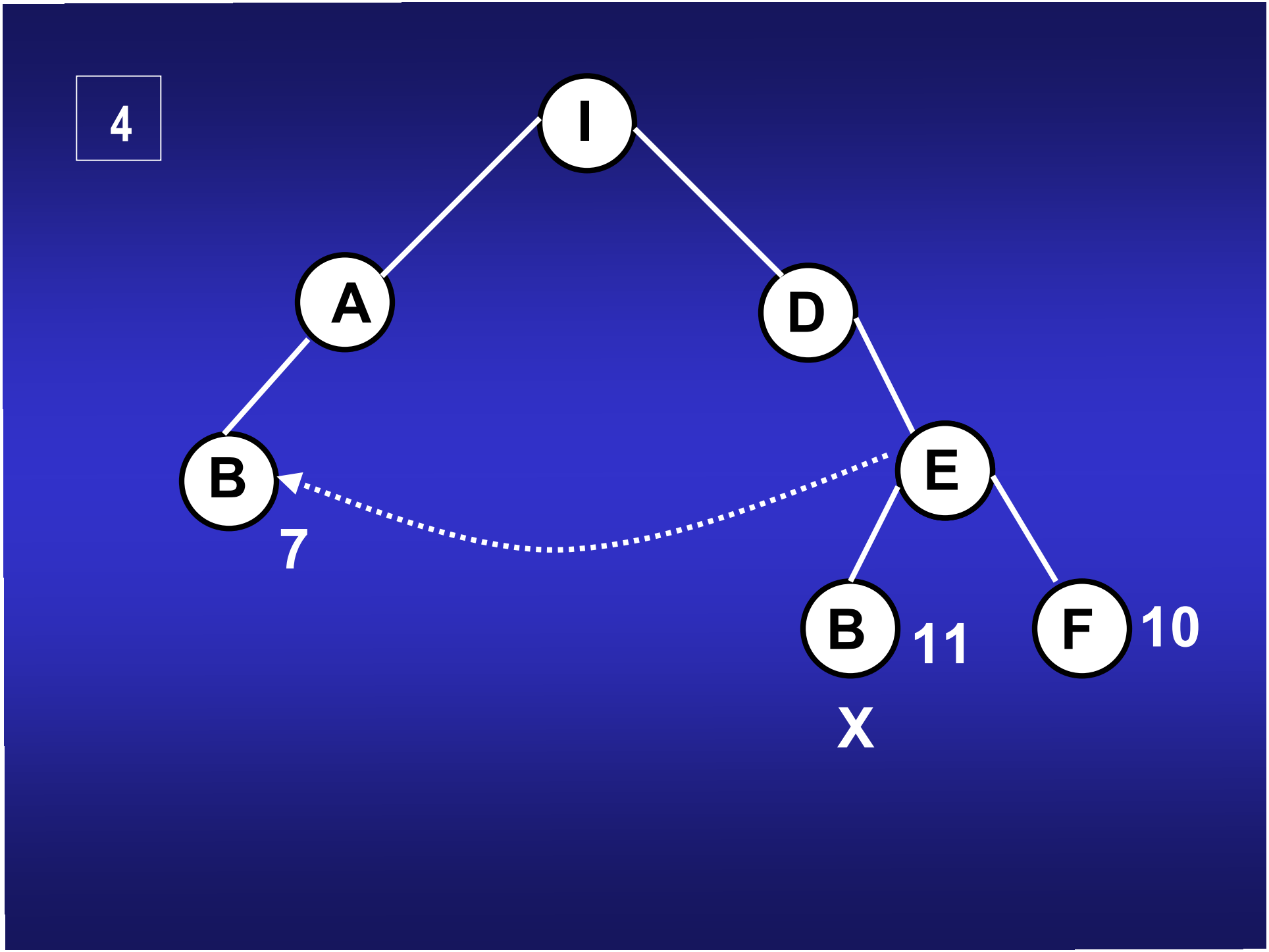
EJEMPLO DE BÚSQUEDA
BEAM SEARCH

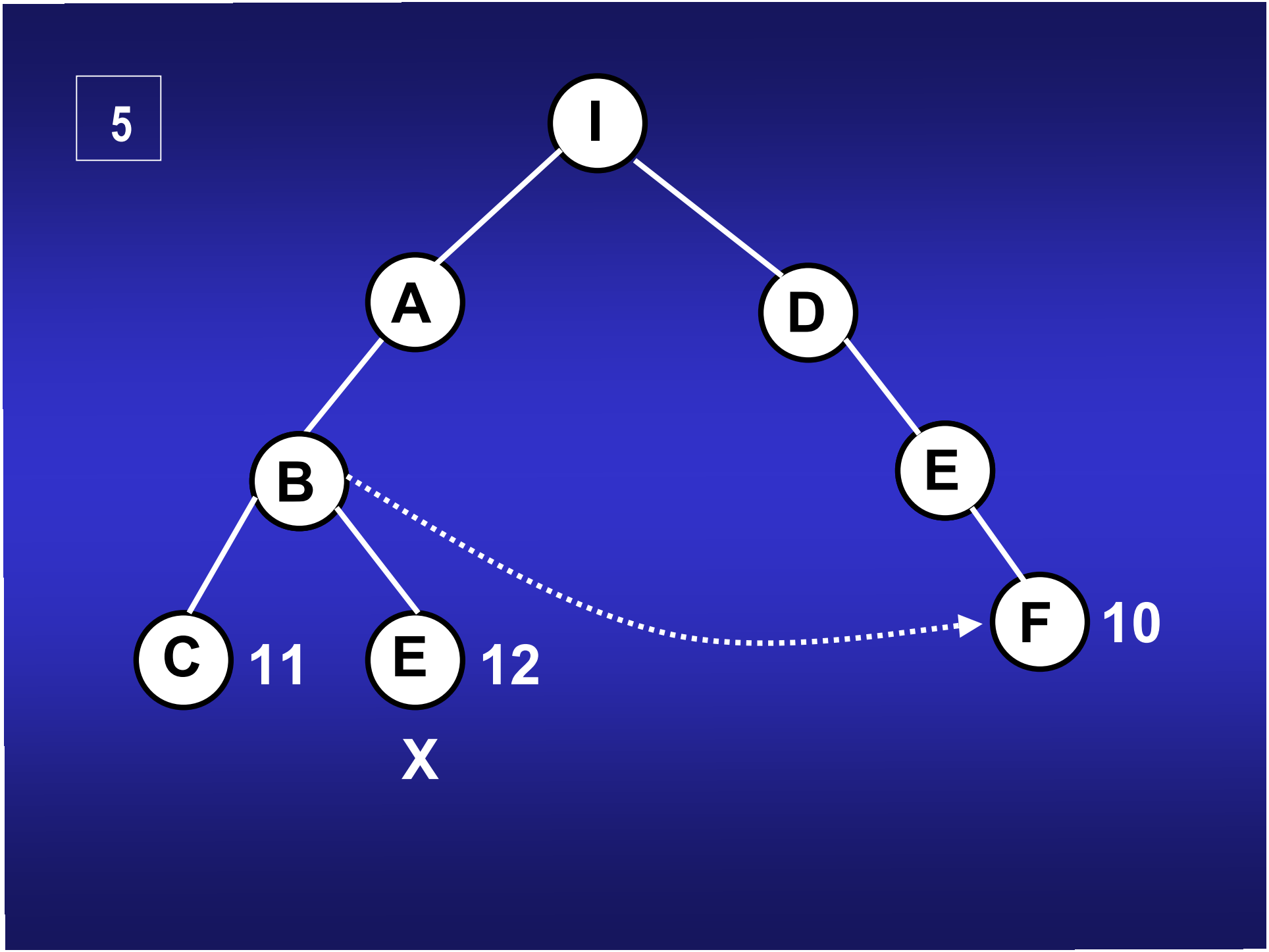
Beam search *Beam=2*

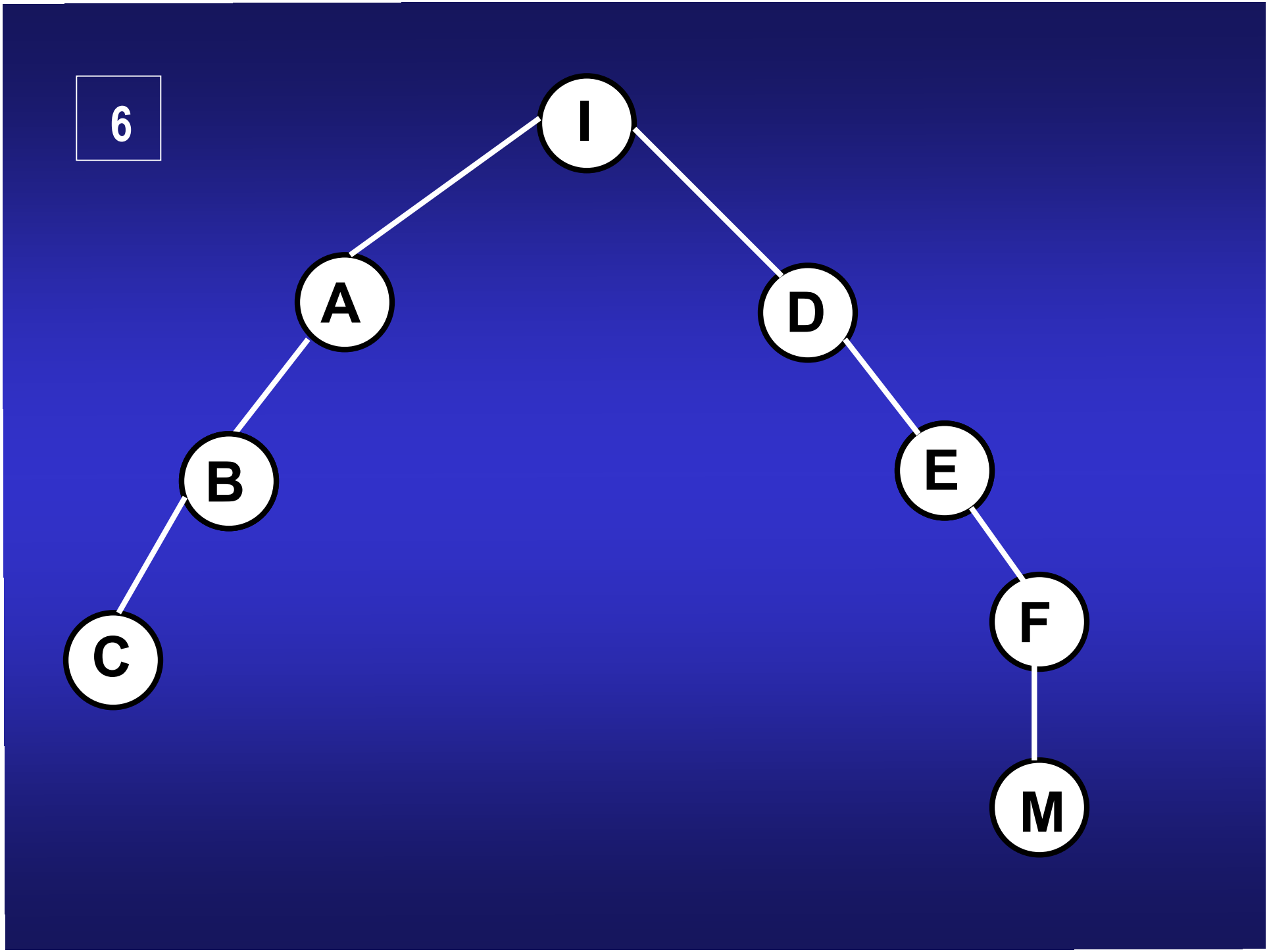


3




4
I
A
D
B
E
7
B
11
F
10
X


5
I
A
D
B
E
C
11
E
12
F
10
X


6
I
A
D
B
E
C
F
M

Espacio Usado

- depth-first:*(b-1)*n* + 1
- breadth-first: b^d
- hill-climbing: 1
- best-first: entre b^n y b^d
- beam-seach: *beam*

Mejor Solución

Cuando importa el costo de encontrar una solución

Si *g(P)* es el costo de camino o solución parcial, la solución óptima es aquella con *g(P)* mínima.

Una forma segura: búsqueda exhaustiva y seleccionar el de menor costo (Brittish Museum)

Best-first no es óptimo, pero con una pequeña variante ya lo es.

Branch and Bound trabaja como best-first pero en cuanto se encuentra una solución, sigue expandiendo los nodos de costos menores al encontrado

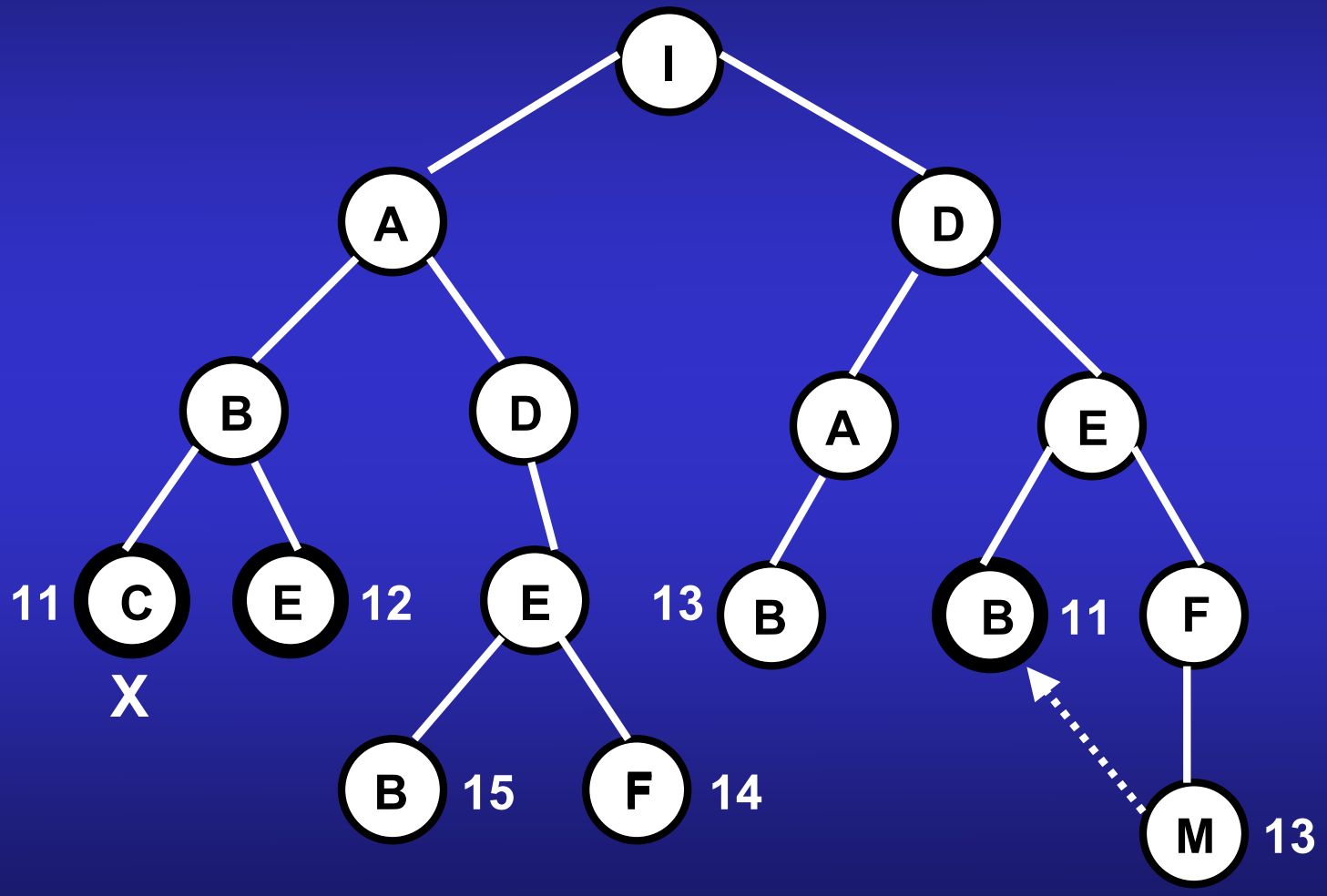
Branch and Bound

Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta y los demás nodos sean de costos mayores o iguales a la meta
si el primer elemento es la meta y los demás nodos son de menor o igual costo a la meta
entonces acaba
si no elimina el primer elemento y añade sus sucesores a la agenda
ordena todos los elementos de la agenda

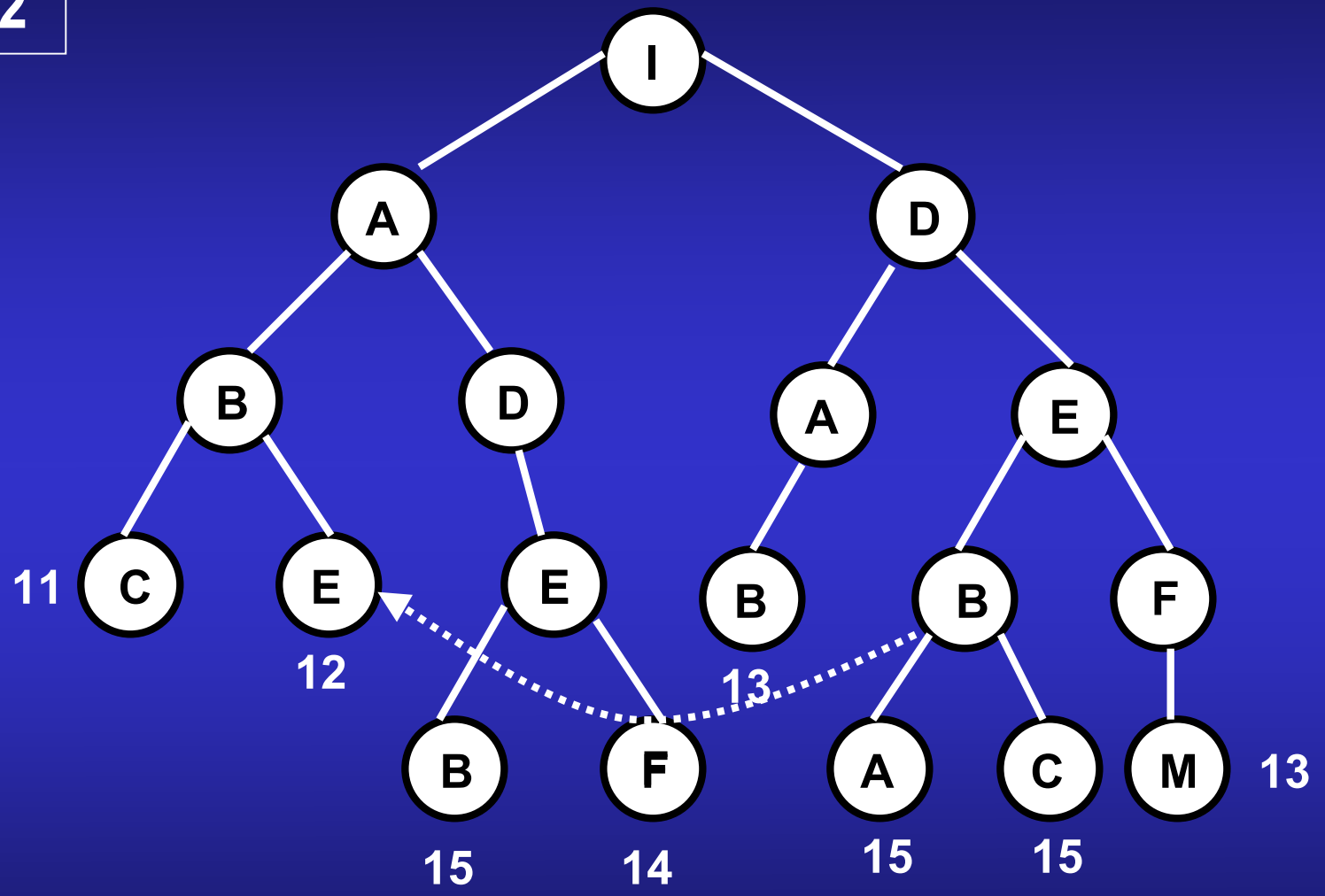
EJEMPLO DE BÚSQUEDA BRANCH AND BOUND

1

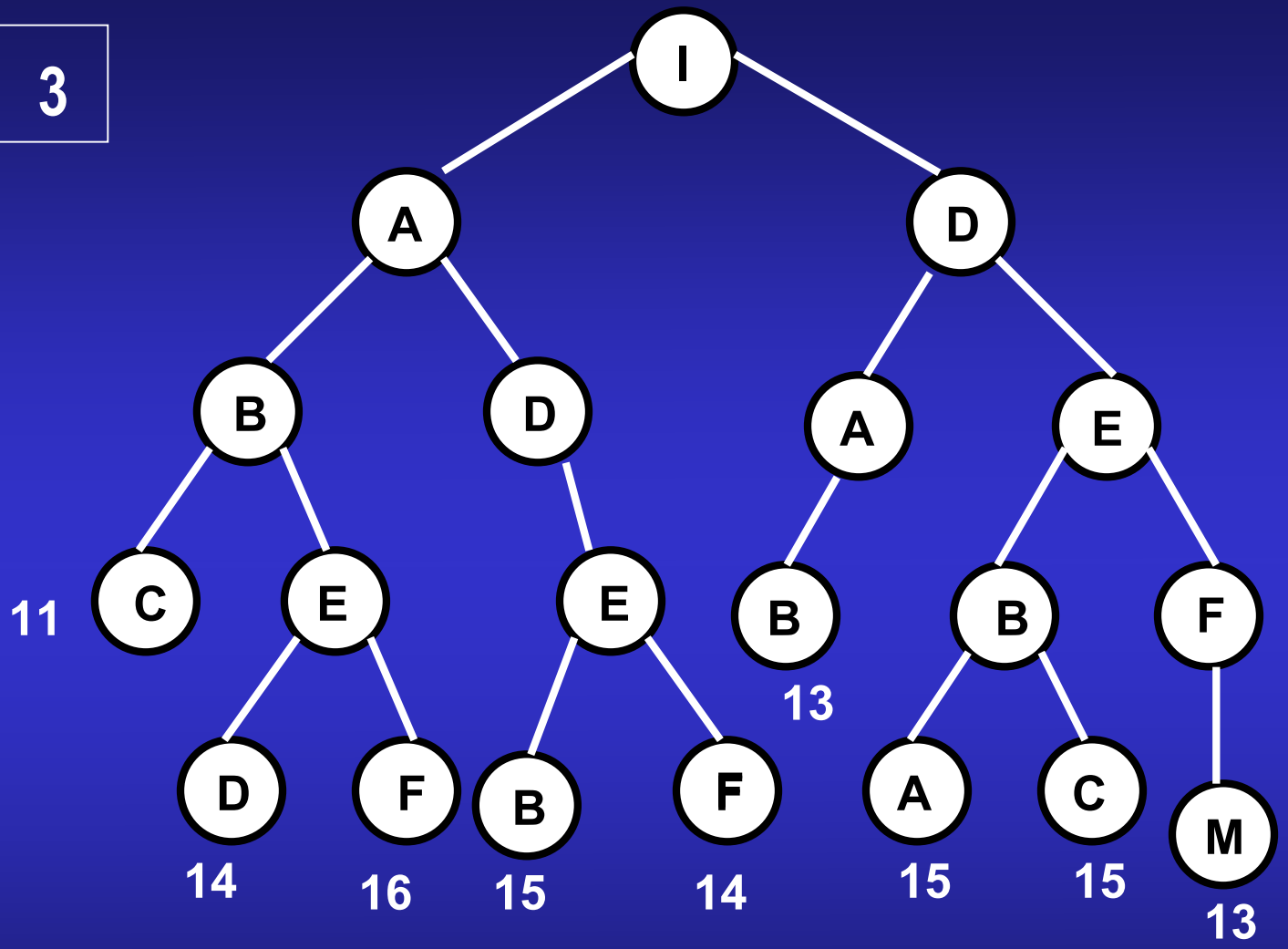
Búsqueda Branch and Bound



2



3



Dynamic Programming

Idea: no explorar caminos a los que ya llegamos por caminos más cortos/baratos

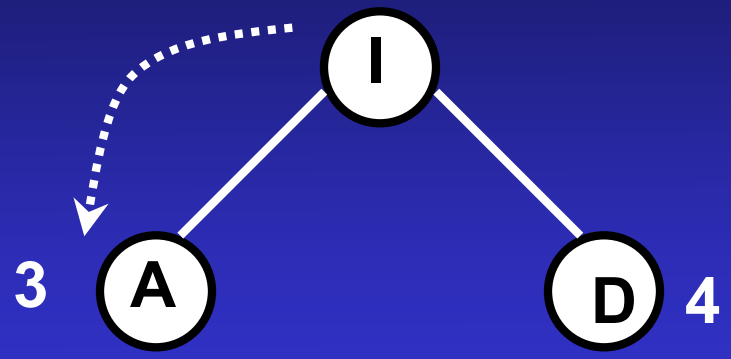
El algoritmo es igual sólo hay que añadir la condición:

elimina todos los caminos que lleguen al mismo nodo excepto el de menor costo

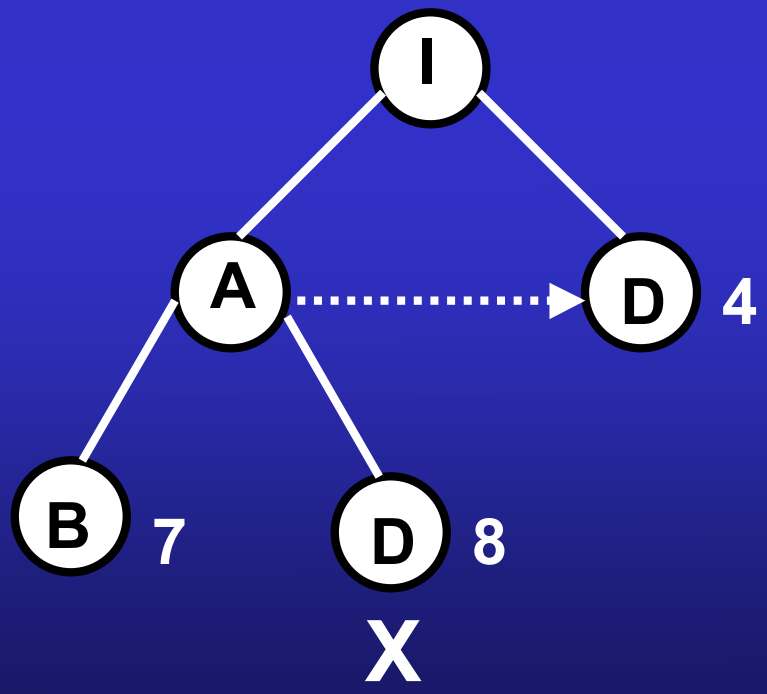
EJEMPLO CON *DYNAMIC PROGRAMMING*

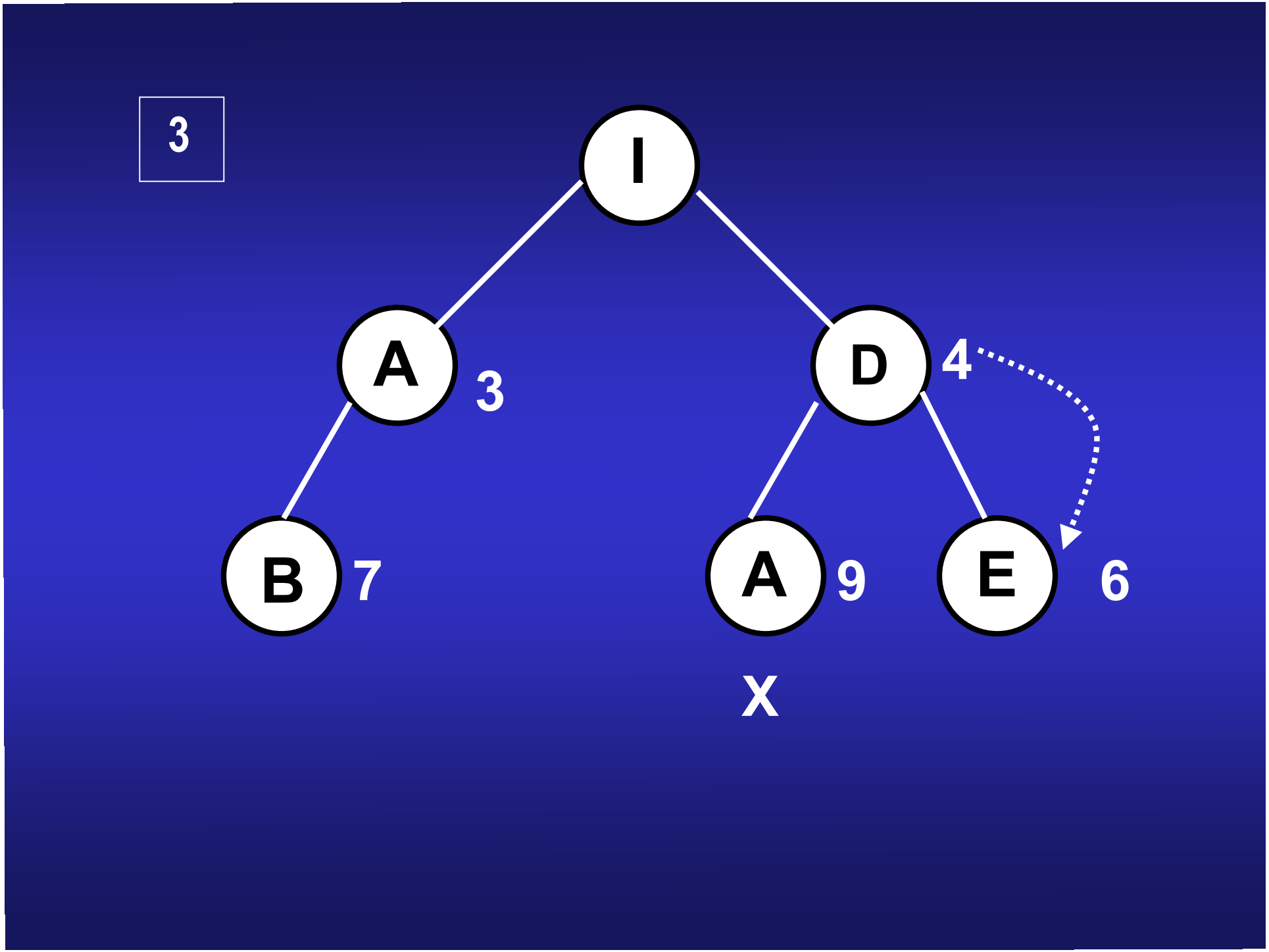
Dynamic programming

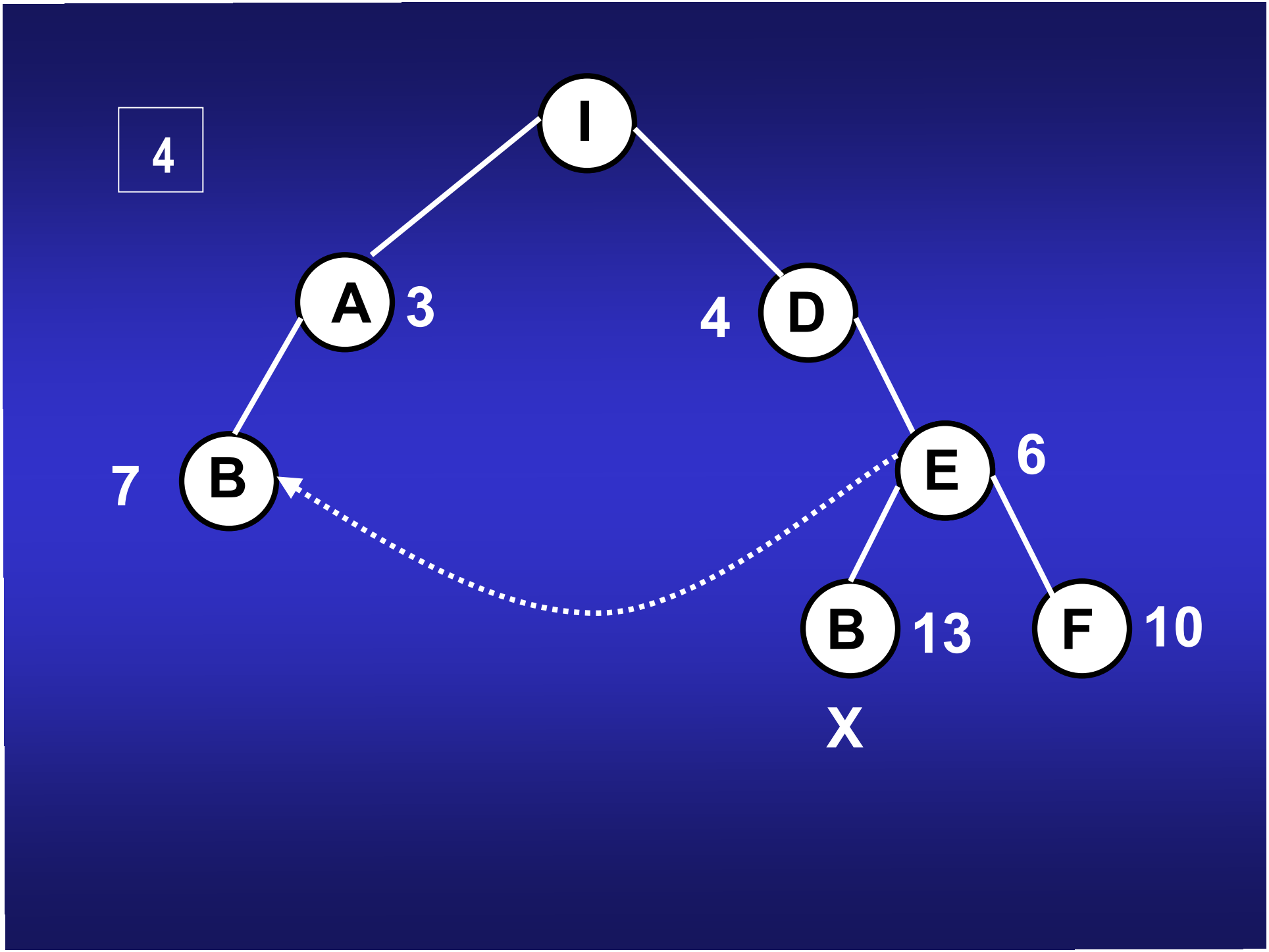
1

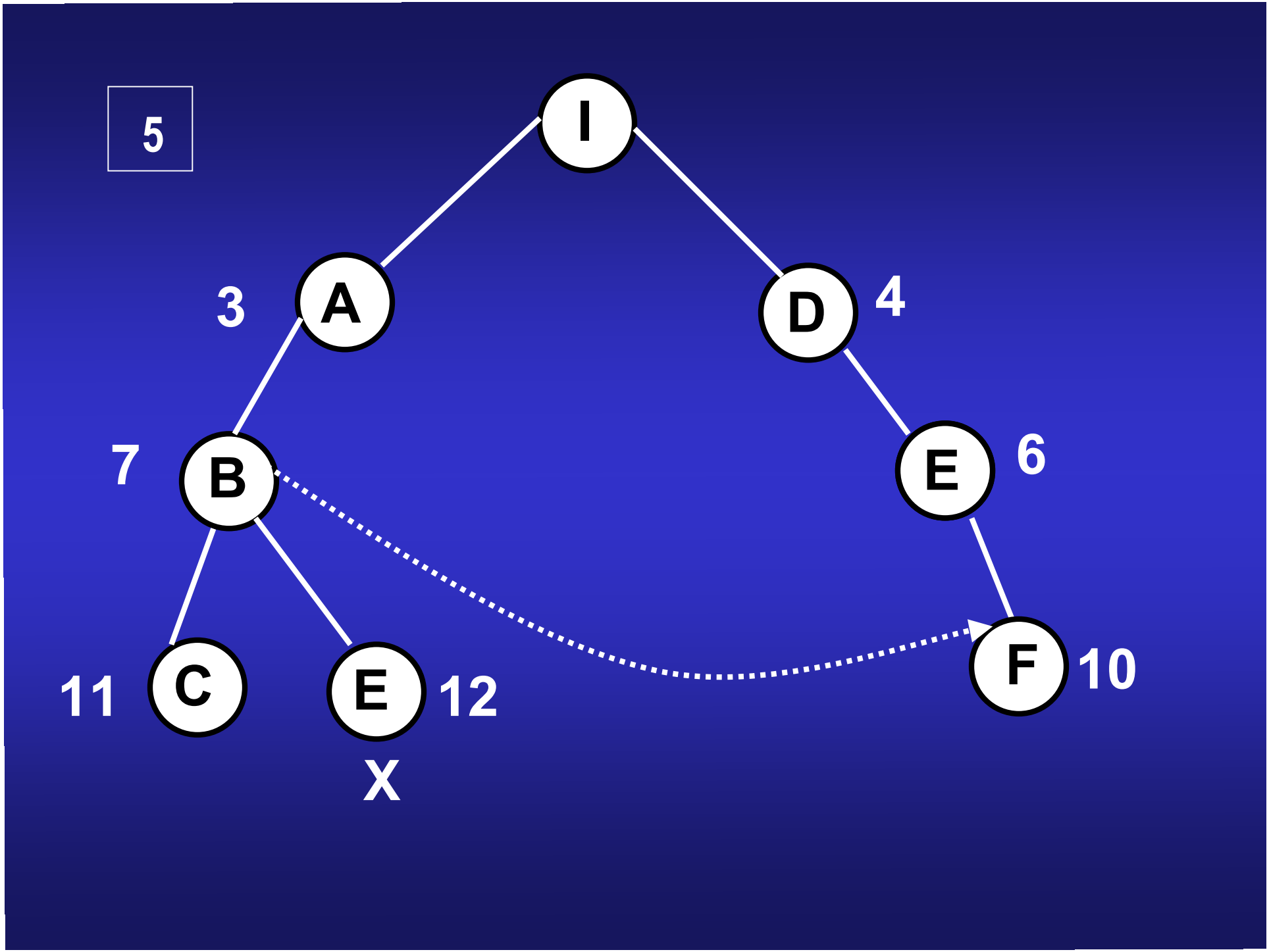


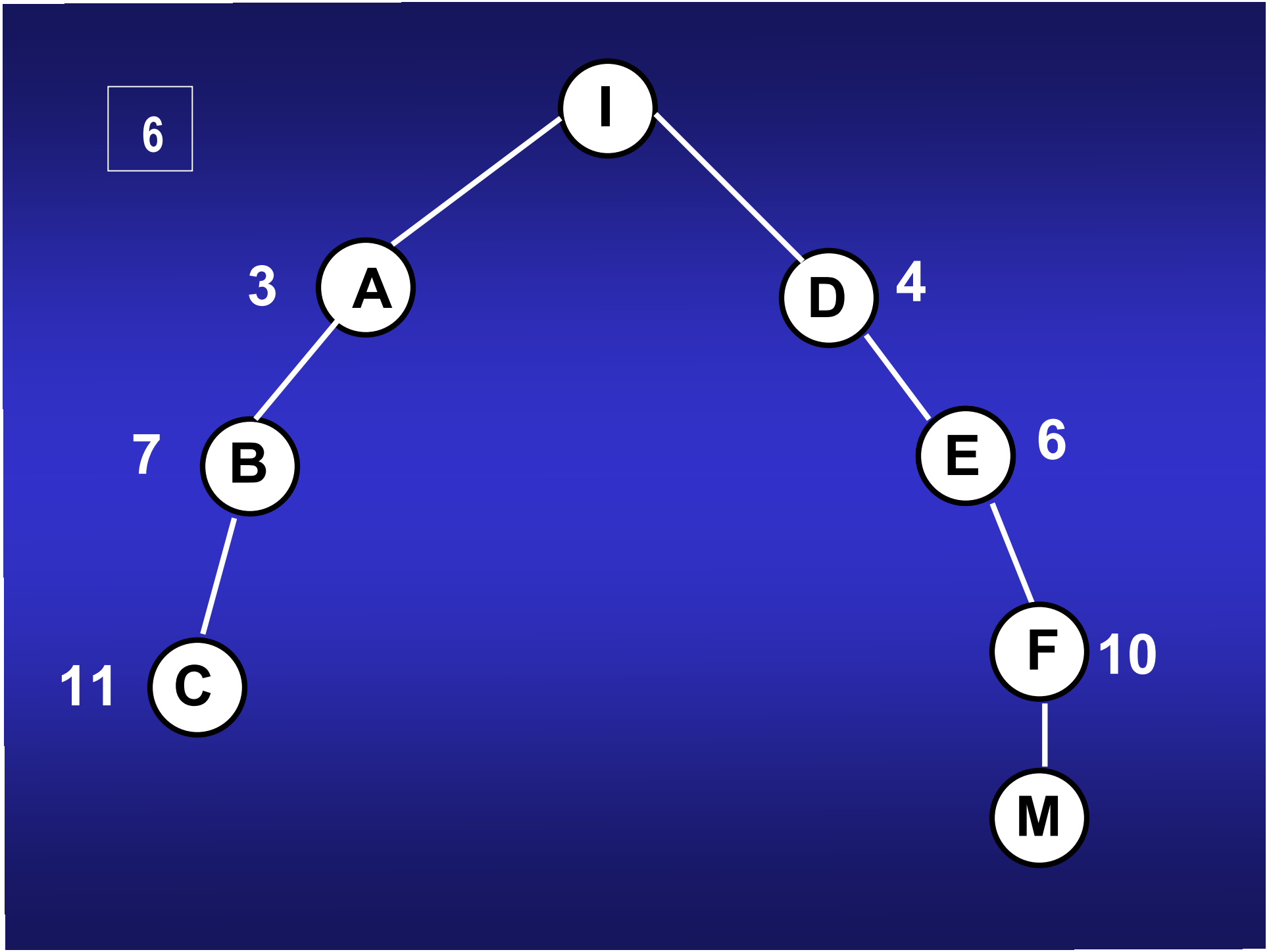
2




3
I
A
3
D
4
B
7
A
9
E
6
X


4
I
A
3
4
D
7
B
E
6
B
13
F
10
X


5
I
3
A
D
4
7
B
E
6
11
C
E
12
X
F
10


6
I
3
A
D
4
7
B
E
6
11
C
F
10
M

A*: utiliza dos medidas

Idea: usar estimaciones de los costos/distancias que faltan junto con los costos/distancias acumuladas

estim(total) = costo(camino recorrido) +
estim(camino que falta)

Las estimaciones no son perfectas, por lo que se usan sub-estimaciones

subestim(total) = costo(camino recorrido) + subestim(camino que falta)

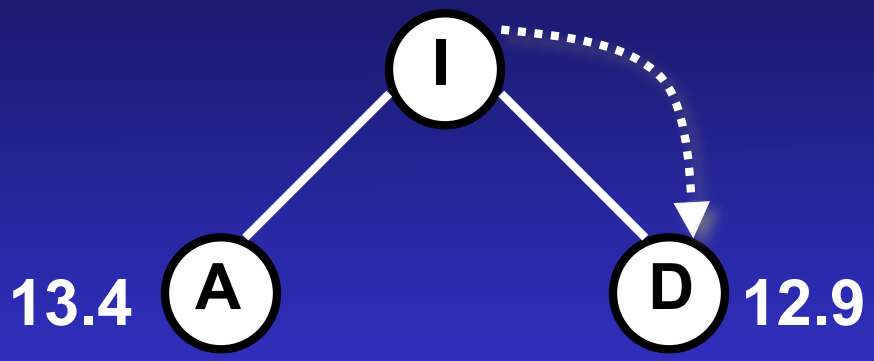
De nuevo expande hasta que los demás tengan sub-estimaciones más grandes (v.g., subestimaciones de distancias entre ciudades pueden ser líneas rectas)

Crea una agenda de un elemento (el nodo raíz)
hasta que la agenda este vacía o se alcance la meta y los demás nodos sean de costos mayores o iguales a la meta
_si_ el primer elemento es la meta y los demás nodos son de menor o igual costo a la meta
entonces acaba
_si no_ elimina el primer elemento y añade sus sucesores a la agenda
ordena todos los elementos de la agenda de acuerdo al costo acumulado más las subestimaciones de los que falta

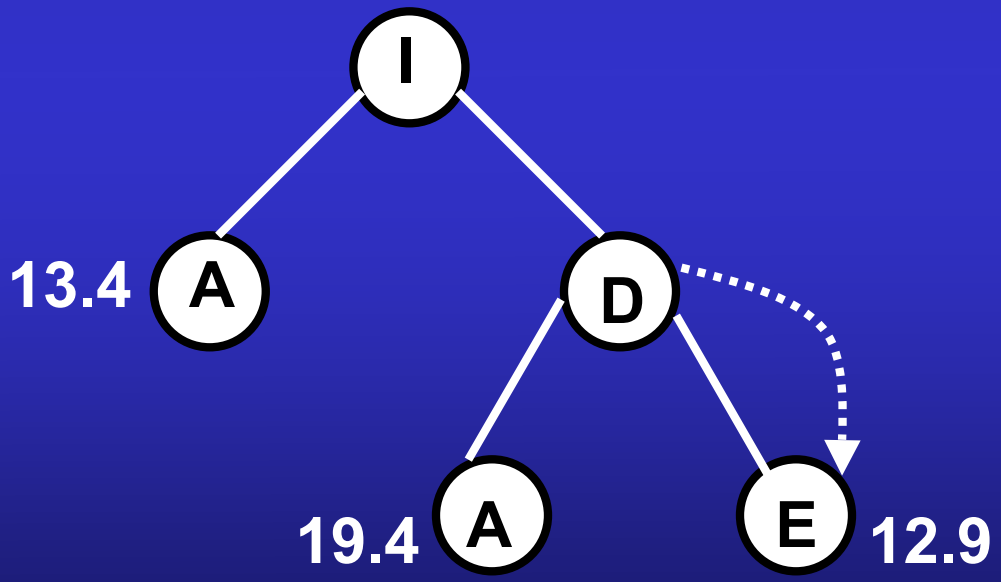
EJEMPLO DE BÚSQUEDA A*

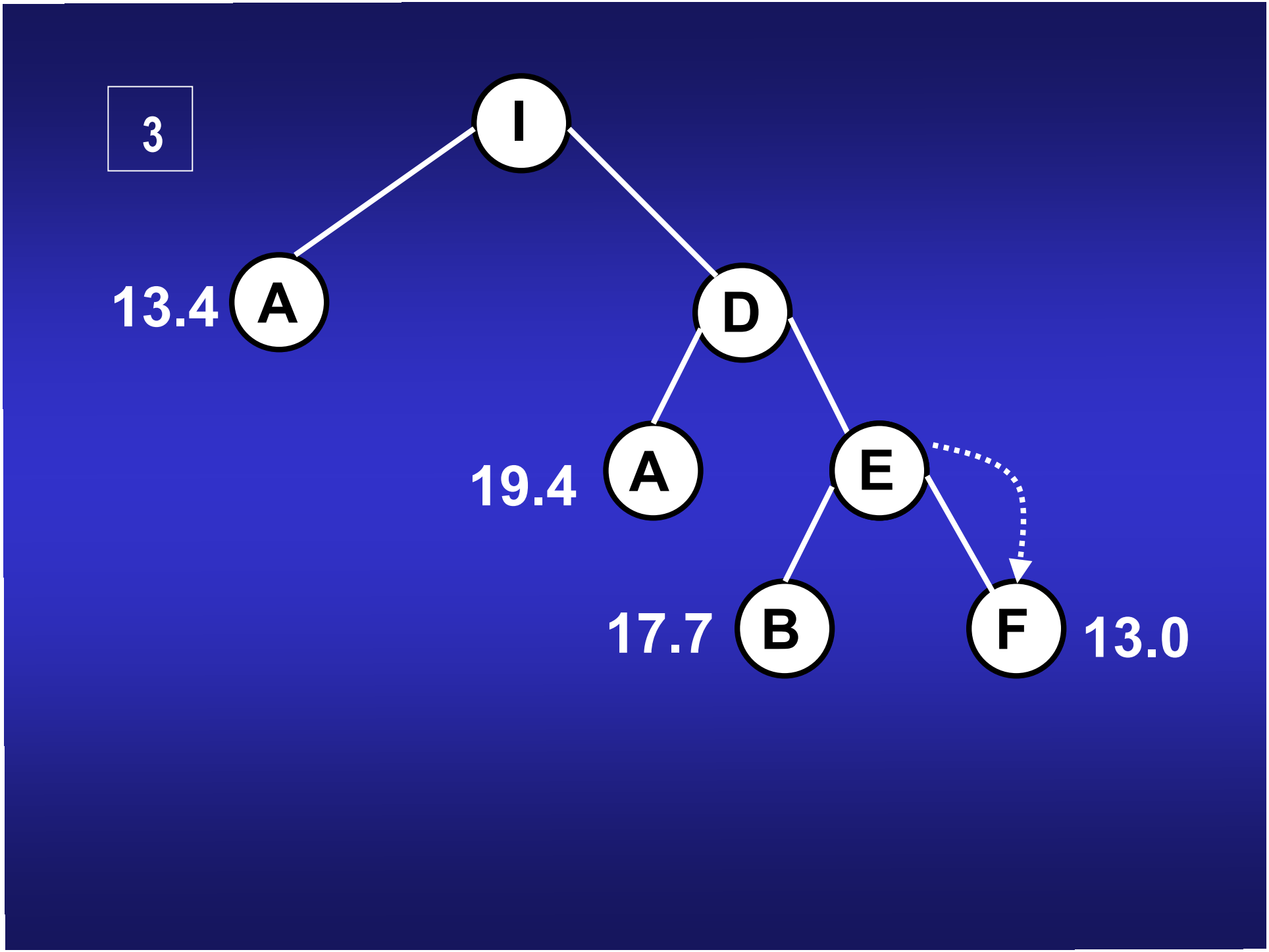
A*

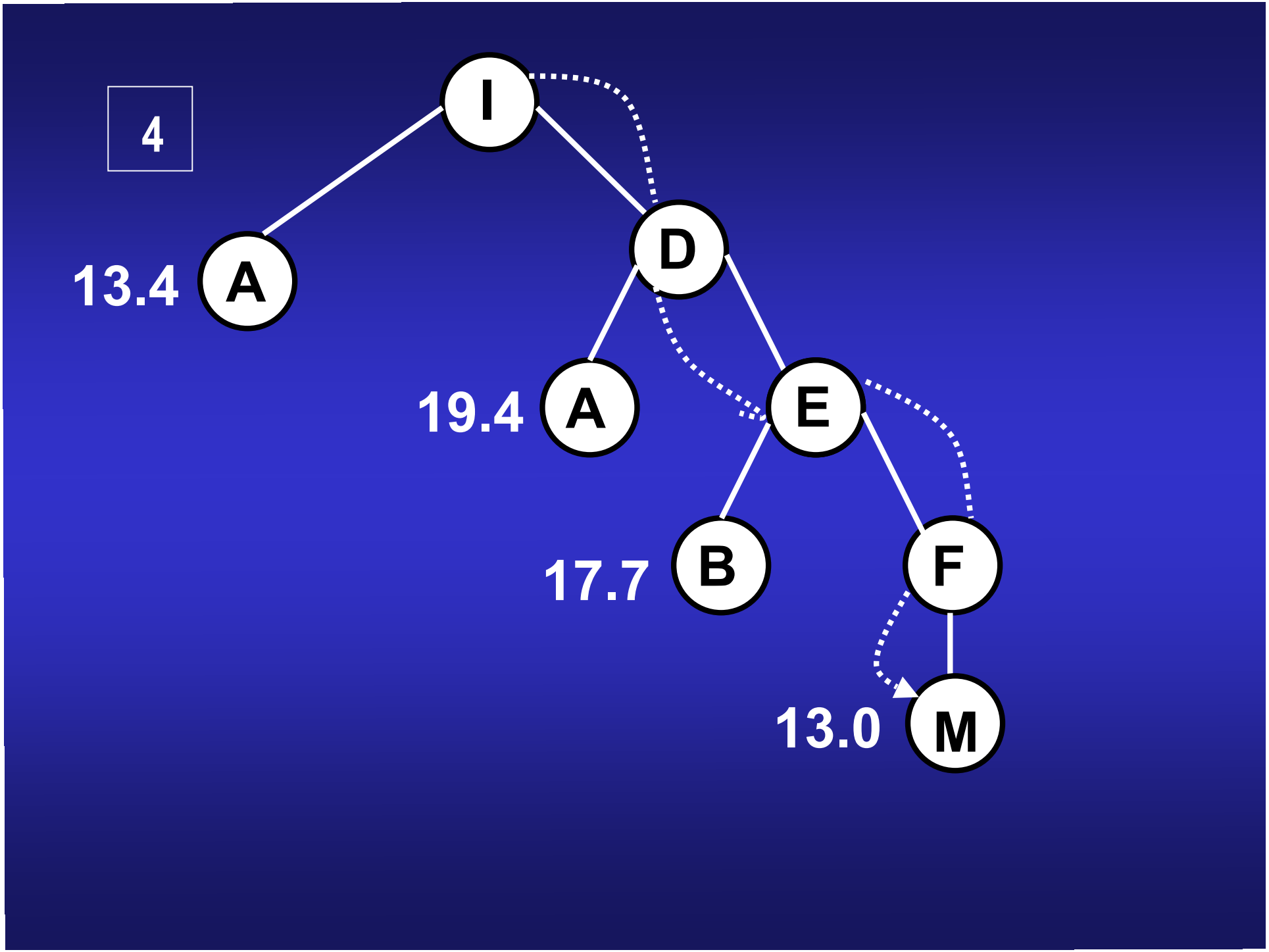
1



2




3
I
13.4
A
D
19.4
A
E
17.7
B
F
13.0


4
I
13.4
A
D
19.4
A
E
17.7
B
F
13.0
M

Costo de búsqueda				Arborescencia efectiva		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Comparación Interactive Deepening (IDS) con A* con dos heurísticas (h1 y h2). Resultados promedios de 100 instancias del *8-puzzle*.

¿Cuándo usamos cada una?

- si el tamaño de búsqueda es pequeño (rara vez), podemos hacer búsqueda exhaustiva
- sin información depth-first con progressive-deepening
- branch and bound en general está bien
- dynamic programming cuando existen muchos posibles caminos con cruces
- A* cuando podemos tener una buena subestimación

Tarea

- Leer Capítulo 3 de Russell
- Plantear el problema la Torres de Hanoi como un problema de búsqueda:
  - Especificar la representación de los estados, y el estado inicial y meta
  - Especificar los operadores para generar estados
  - Mostrar el árbol para el caso de 3 discos
  - Se te ocurre alguna heurística para hacer más eficiente la búsqueda?