Algoritmos de Búsqueda
Consideraciones de Diseño e Implementación

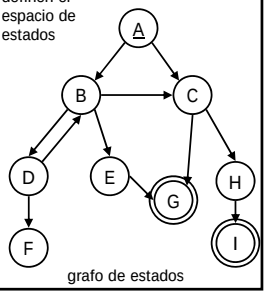
Inteligencia Artificial
2º cuatrimestre de 2009

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Formulación de un Problema de Búsqueda

- estado inicial
- conjunto de acciones
  Se usa el término operador para denotar la descripción de una acción en términos de qué estado se alcanzará al llevar a cabo la acción en un estado particular.
- test de meta (goal test)
  Permite determinar si un dado estado es meta.
- función de costo de camino
  Asigna un costo a un camino. En gral. se define como la suma de los costos de las acciones a lo largo del camino

Juntos definen el espacio de estados

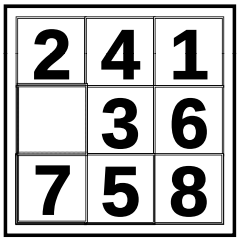


• Problema: Hallar un camino desde el estado inicial hasta una meta.

grafo de estados

Ejemplo

8-puzzle



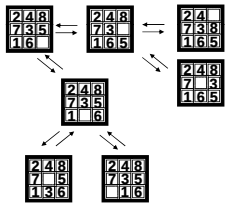
Ejemplo

- estado inicial:

- conjunto de operadores:
  intercambiar el lugar vacío con una de las posiciones adyacentes.
- test de meta:

- función de costo de camino:
  la cantidad de movimientos (acciones) efectuadas a lo largo del camino.

espacio de estados: todas las posibles configuraciones.



(parte del) grafo de estados

Algoritmos de Búsqueda

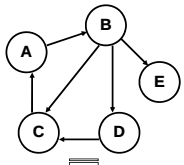
Formulación del Problema de Búsqueda:
- estado inicial
- conj. de acciones
- test de meta
- fc. de costo de camino

→ Algoritmo de Búsqueda → **Solución:** un camino desde el estado inicial a una meta

El valor asociado a la **solución** por la **función de costo** determina la calidad de la misma. ⇒ Cuanto **MENOR** el costo, **MEJOR** la solución

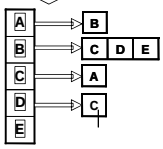
5

Representación de grafos



Matriz de adyacencia

	A	B	C	D	E
A		1			
B			1	1	1
C	1				
D			1		
E					



Listas de adyacencia (incidencia)

6

Una posible representación en PROLOG

• El grafo es una colección de hechos que modelan explícitamente cada uno de sus **arcos**.

```
arc(a,b).
arc(b,c).
arc(b,d).
arc(b,e).
arc(c,a).
arc(d,c).
```

• Así la consulta por un arco en particular se resuelve en forma directa.

7

Algoritmos de Búsqueda

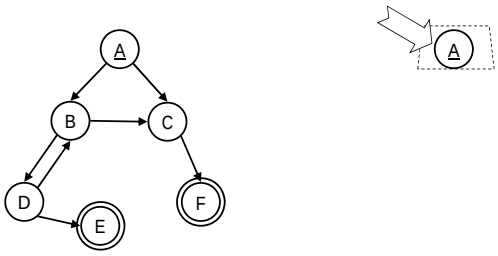
• Un algoritmo de búsqueda va generando y explorando los nodos (estados) del espacio de estados.

• Podría pensarse en el proceso de búsqueda como construyendo un árbol de búsqueda.

• El estado inicial es la raíz del árbol y los nodos hoja corresponden a estados que no tienen sucesores en el espacio/grafo de estados o no fueron expandidos aún.

8

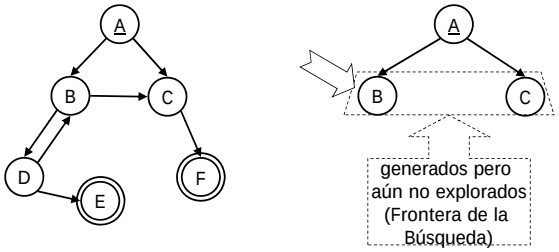
Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

9

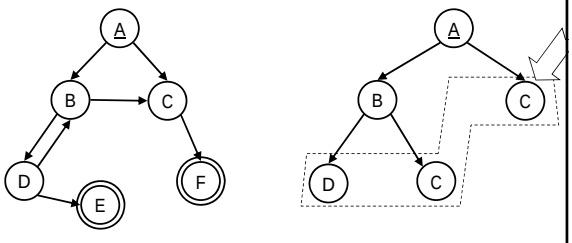
Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

10

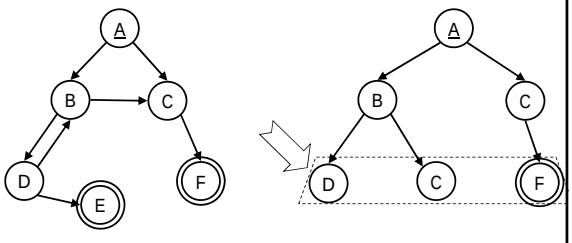
Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

11

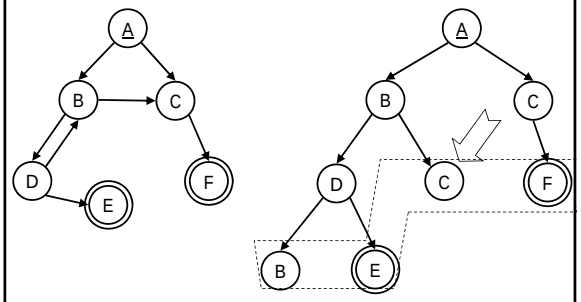
Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

12

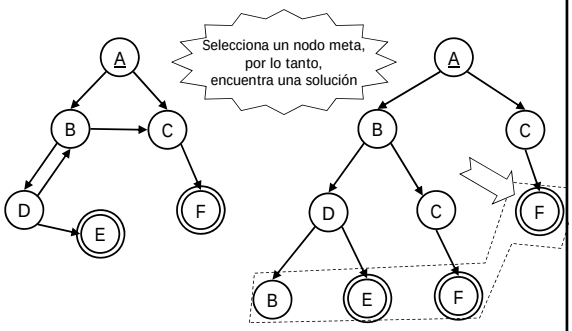
Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

13

Arbol asociado a una Búsqueda



Grafo de Estados Arbol de Búsqueda

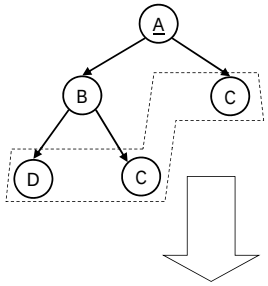
14

Algoritmos de Búsqueda

- El algoritmo de búsqueda mantiene en todo momento una lista representando la Frontera de la búsqueda, es decir, aquellos nodos generados pero aún no explorados.
- Utilizaremos **`nodo(Estado, Camino)`** para representar un nodo de la frontera de la búsqueda (hoja del árbol de búsqueda) etiquetado con **`Estado`**, y donde **`Camino`** es el camino desde el estado inicial hasta dicho nodo en el árbol.

15

Representación de la Frontera



Árbol de Búsqueda en un instante dado de la ejecución del algoritmo.

Representación mantenida por el algoritmo (lista Front).

[nodo(C, [A]), nodo(D, [B,A]), nodo(C, [B,A])]

16

Algoritmo general de búsqueda

```
% buscar(+Front, -Solucion)

buscar(Front,[Estado|Camino] ):-
    seleccionar(Nodo, Front, _FrontSinNodo),
    Nodo = nodo(Estado, Camino),
    es_meta(Estado).

buscar(Front, Solucion):-
    seleccionar(Nodo, Front, FrontSinNodo),
    vecinos(Nodo, Vecinos),
    agregar_frontera(Vecinos, FrontSinNodo, NewFront),
    buscar(NewFront, Solucion).
```

test de meta

Determinan la estrategia de búsqueda

```
?- buscar( [ nodo(EstadoInic,[]) ], Solución)
```

17

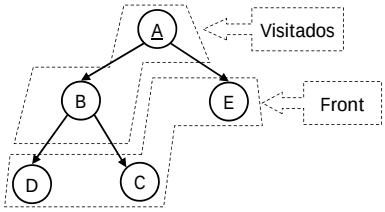
Generación de Vecinos

```
% vecinos(+Nodo, -Vecinos)

vecinos(Nodo, Vecinos):-
     Nodo = nodo(N, Camino),
     findall(nodo(V,[N|Camino]),
             arc(N,V),
             Vecinos).
```

18

Control de Visitados



Árbol de Búsqueda en un instante dado de la ejecución del algoritmo.

Control de Visitados

```
% buscarV(+Front, +Vis, -Solucion)

buscarV(Front, _Vis, [Estado|Camino]):-
    seleccionar(Nodo, Front, _FrontSinNodo),
    Nodo = nodo(Estado, Camino),
    es_meta(Estado).

buscarV(Front, Vis, Solucion):-
    seleccionar(Nodo, Front, FrontSinNodo),
    vecinosV(Nodo, Front, Vis, Vecinos),
    agregar_frontera(Vecinos, FrontSinNodo, NewFront),
    Nodo = nodo(Estado, _Camino),
    buscarV(NewFront, [Estado|Vis], Solucion).
```

No genera nodos que ya fueron generados antes

```
?- buscarV([NodoInicial],[], Solución)
```


Generación de Vecinos

• ¿Cómo chequear que un nodo V no esté en Vis ni en Front?

```
not member(V, Vis),
not member(node(V,_), Front)
```

```
% vecinosV(+Nodo, +Front, +Vis -Vecinos)
vecinos(Nodo, Front, Vis, Vecinos):-
    Nodo = nodo(N, Camino),
    findall(nodo(V,[N|Camino]),
    (arc(N, V),
    not member (V, Vis),
    not member (nodo(V,_), Front)),
    Vecinos).
```


Búsqueda Ciega

Depth-first search (DFS)

• DFS siempre selecciona para expandir el nodo de la frontera más profundo.

• De esta manera:
- la búsqueda se extiende en profundidad sobre uno de los caminos del grafo.
- cuando este camino no puede extenderse más, la búsqueda continúa desde el nodo no explorado (ie, en la frontera) que se encuentre a mayor profundidad.

• Es decir, DFS toma de la Frontera el nodo que ha sido agregado último.

• Esto sugiere la manipulación de la Frontera como una **pila**.

Depth-first search (DFS)

• Sobre grafos sin ciclos no tiene ninguna complicación.

• Sobre grafos con ciclos debe evitar la exploración repetida de nodos.

• Por esta razón utilizaremos la versión de buscar que controla visitados para implementar DFS.

Implementación de DFS

• La implementación de DFS (con control de ciclos) se obtiene a partir de `buscarV/3` implementando `seleccionar/3` y `agregar/3` como se muestra a continuación:

- La selección de un nodo a explorar consiste en tomar el primero de la Frontera:

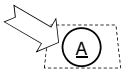
```
% seleccionar(-Nodo, +Front, -FrontSinNodo)
seleccionar(Nodo, [Nodo|RestoFront], RestoFront).
```

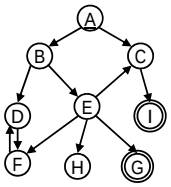
- Los vecinos se agregan al comienzo de la frontera:

```
% agregar(+Vecinos, +FrontSinNodo, -NewFront)
agregar(Vecinos, FrontSinNodo, NewFront):-
      append(Vecinos, FrontSinNodo, NewFront).
```

25

DFS: Traza

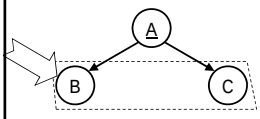


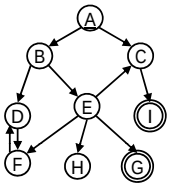


Front	Vis
A - []	

26

DFS: Traza

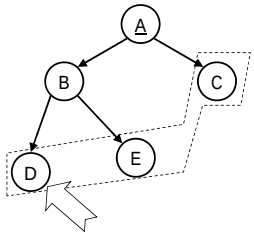


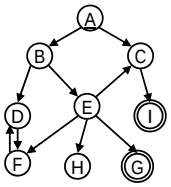


Front	Vis
B – [A]	A
C – [A]	

27

DFS: Traza

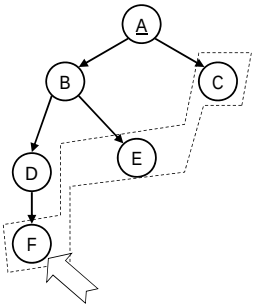


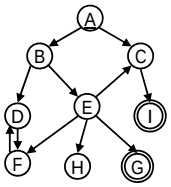


Front	Vis
D – [B,A]	A
E – [B,A]	B
C – [A]	

28

DFS: Traza

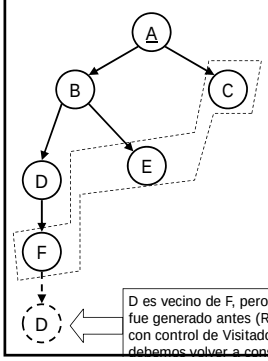


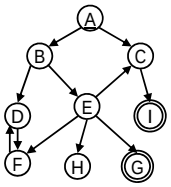


Front	Vis
F – [D,B,A]	A
E – [B,A]	B
C – [A]	D

29

DFS: Traza

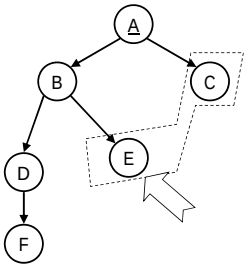


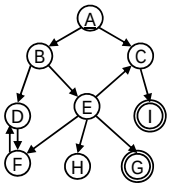


Front	Vis
F – [D,B,A]	A
E – [B,A]	B
C – [A]	D

D es vecino de F, pero no volvemos a considerarlo debido a que D ya fue generado antes (Recordar que, de acuerdo al alg. de búsqueda con control de Visitados, si un nodo ya está en Front o en Vis no debemos volver a considerarlo)

DFS: Traza

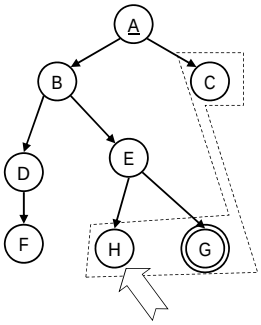


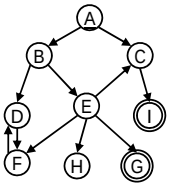


Front	Vis
E – [B,A]	A
C – [A]	B
	D
	F

31

DFS: Traza

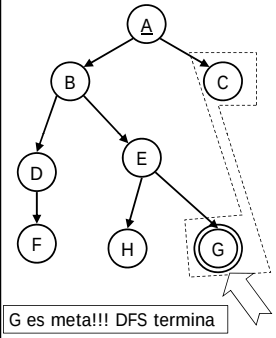




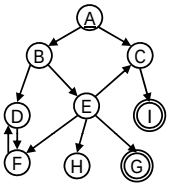
Front	Vis
H – [E,B,A]	A
G – [E,B,A]	B
C – [A]	D
	F
	E

32

DFS: Traza



G es meta!!! DFS termina



Front	Vis
G – [E,B,A]	A
C – [A]	B
	D
	F
	E
	H

SOLUCIÓN

33

Breadth-first search (BFS)

• BFS explora el grafo por niveles. Esto evita que la exploración entre en ciclos y garantiza encontrar el camino más corto en cantidad de arcos a una meta (optimalidad).

• Es decir, BFS toma de la Frontera el nodo que hace más tiempo se encuentra en ella, lo que sugiere la implementación de la Frontera como una **cola**.

• Aunque BFS no puede entrar en ciclos, podría resultar útil controlar visitados por cuestiones de eficiencia.

34

Implementación de BFS

• La implementación de BFS se obtiene a partir de `buscar/2` (o `buscarV/3`) implementando `seleccionar/3` y `agregar/3` como se muestra a continuación:

▪ La selección de un nodo a explorar consiste en tomar el primero de la Frontera:

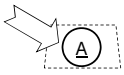
```
% seleccionar(-Nodo, +Front, -FrontSinNodo)
seleccionar(Nodo, [Nodo|RestoFront], RestoFront).
```

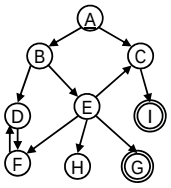
▪ Los vecinos se agregan al final de la frontera:

```
% agregar(+Vecinos, +FrontSinNodo, -NewFront)
agregar(Vecinos, FrontSinNodo, NewFront):-
      append(FrontSinNodo, Vecinos, NewFront).
```

35

BFS: Traza

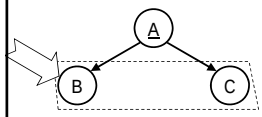


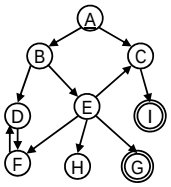


Front	Vis
A - []	

36

BFS: Traza

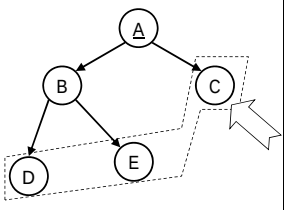


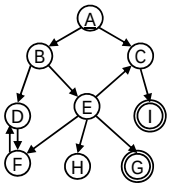


Front	Vis
B – [A]	A
C – [A]	

37

BFS: Traza

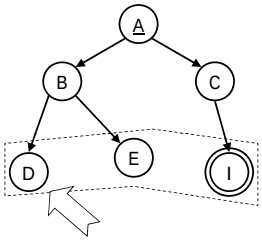


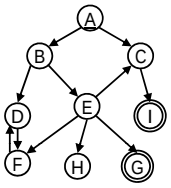


Front	Vis
C – [A]	A
D – [B,A]	B
E – [B,A]	

38

BFS: Traza

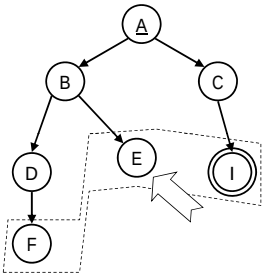


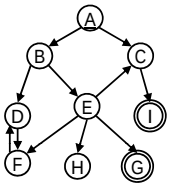


Front	Vis
D – [B,A]	A
E – [B,A]	B
I – [C,A]	C

39

BFS: Traza

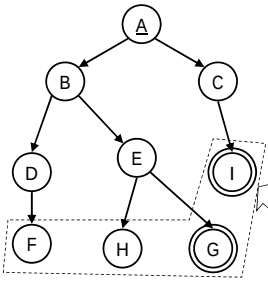


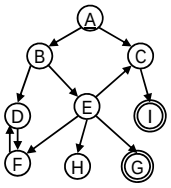


Front	Vis
E – [B,A]	A
I – [C,A]	B
F – [D,B,A]	C
	D

40

BFS: Traza





Front	Vis
I – [C,A]	A
F – [D,B,A]	B
H – [E,B,A]	C
G – [E,B,A]	D
	E

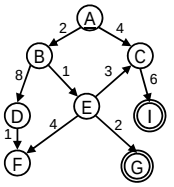
SOLUCIÓN

I es meta!!! BFS termina

41

Limitaciones de BFS

- BFS garantiza encontrar el camino más corto en cantidad de arcos a una meta.
- En consecuencia, BFS es optimal para aquellos problemas de búsqueda donde todos los opedores (arcos) tienen el mismo costo, ya que más corto significa menor costo.
- Si los operadores tienen diferente costo, entonces BFS no es optimal:



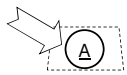
42

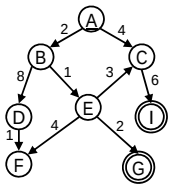
Uniform Cost Search (UCS)

- UCS es una adaptación de la estrategia de búsqueda BFS para problemas con operadores de diferentes costos.
- BFS selecciona para visitar (expandir) el nodo de la frontera de menor profundidad. En consecuencia, los caminos mantenidos en la frontera se van expandiendo de manera uniforme (justa) en términos de profundidad.
- UCS selecciona para visitar el nodo de la frontera con menor costo de camino asociado.
- De esta forma, los caminos mantenidos en la frontera se van expandiendo de manera uniforme (justa) de acuerdo a sus costos.
- La solución encontrada por UCS será siempre la de menor costo de camino, es decir, optimal.

43

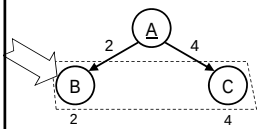
UCS: Traza

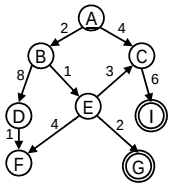




44

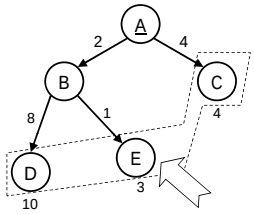
UCS: Traza

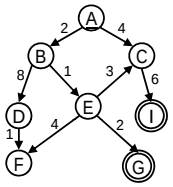




45

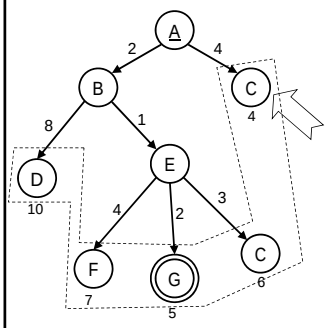
UCS: Traza

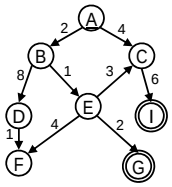




46

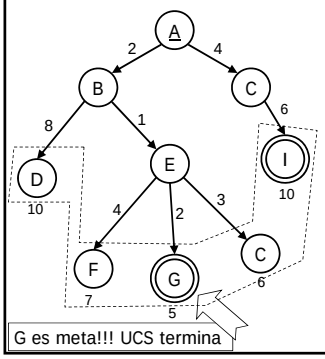
UCS: Traza



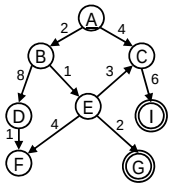


47

UCS: Traza



G es meta!!! UCS termina



Solución:
[A, B, E, G]
(costo 5)

48

Búsqueda Heurística

Búsqueda Informada o Heurística

• Desventaja de búsqueda ciega: se explora sistemáticamente el espacio de búsqueda sin ninguna guía. Por lo tanto, en la mayoría de los casos, es extremadamente ineficiente.

• En búsqueda informada o heurística consideramos información específica acerca del problema para guiar la exploración hacia una (buena) solución.

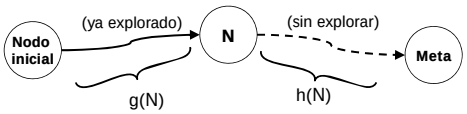
• Una heurística es una función h/1 que, aplicada a un estado E, provee una estimación del costo de alcanzar una meta a partir de E.

50

Algoritmo A*

• Al momento de seleccionar un nodo a expandir, A* elige aquel que *promete arribar* a una mejor solución.

• Para cada nodo N en la frontera, A* conoce:
- el costo del camino usado para alcanzar N: g(N)
- la estimac. del costo del camino desde N hasta una meta: h(N).



• Entonces f(N)= g(N)+h(N) es una estimación del costo total de un camino a la meta (i.e., de una solución) que pasa por N.

• A* selecciona el nodo que tenga menor valor asociado por f.

51

Algoritmo A*

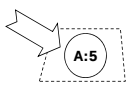
• Al efectuar búsqueda A* puede efectuarse control de ciclos por razones de eficiencia, podando aquellas ramas del árbol correspondientes a estados ya generados previamente.

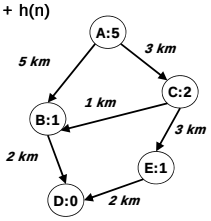
• No obstante, un mismo estado E puede pertenecer a distintos caminos, con distintos costos.

• Por lo tanto si un estado es generado nuevamente pero sobre un mejor camino, vale la pena volver a considerarlo.

52

A*: f(n) = g(n) + h(n)

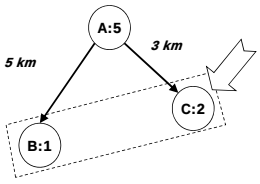


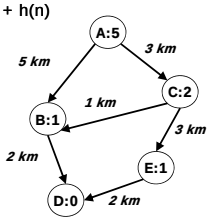


Front	Vis
A - [ ] - 5	

53

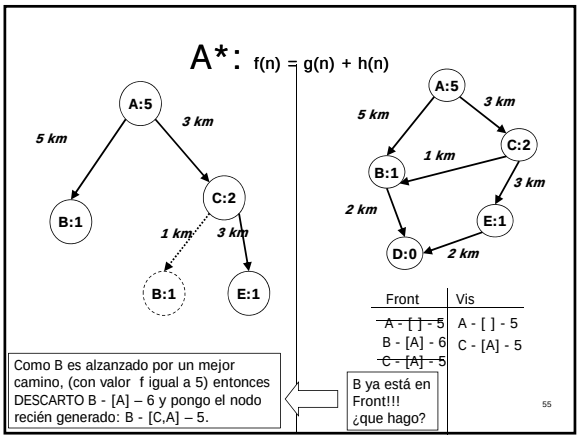
A*: f(n) = g(n) + h(n)

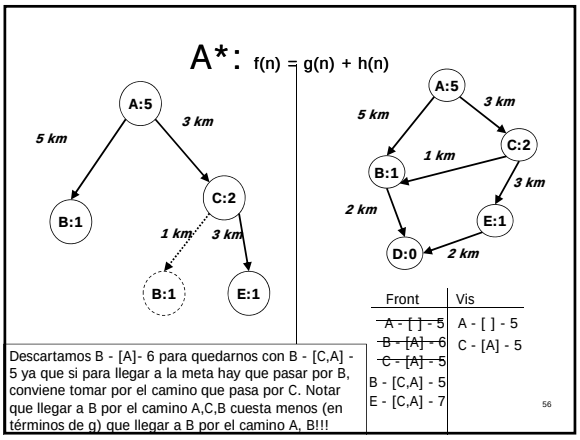


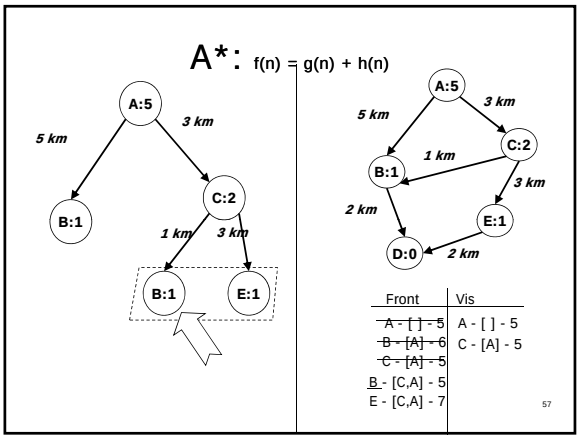


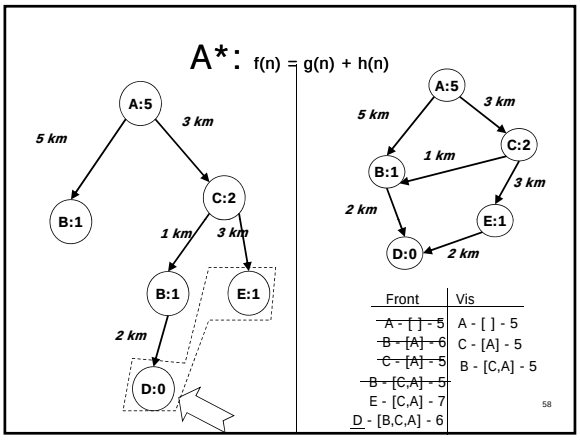
Front	Vis
~~A - [ ] - 5~~	A - [ ] - 5
B - [A] - 6	
C - [A] - 5	

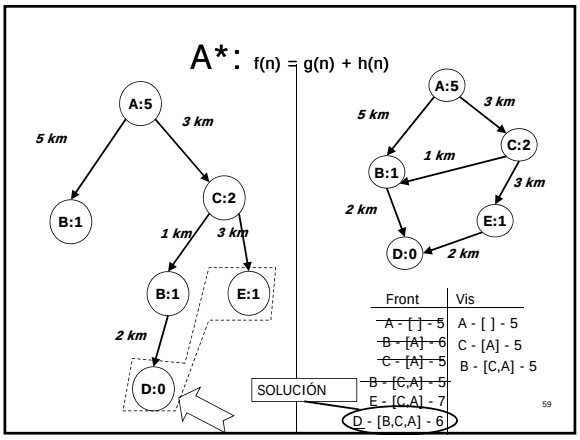
54







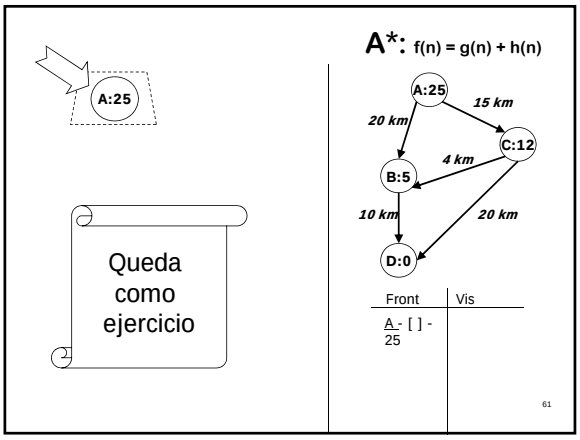


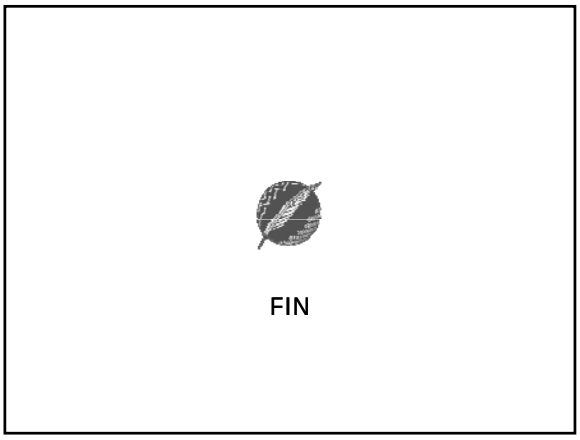


Generación repetida de un nodo

- Si existe en Front un nodo N etiquetado con un estado E, y generamos E por un mejor camino que el representado por N, entonces reemplazamos N en Front por un nodo N' para E con este nuevo camino.
- Si existe en Vis un nodo N etiquetado con un estado E, y generamos E por un mejor camino, entonces N es eliminado de Vis y se agrega a Front un nuevo nodo N' para E con este nuevo camino.
- De esta forma, E podrá ser ser reconsiderado en el futuro.

60


A:25
Queda como ejercicio
A*: f(n) = g(n) + h(n)
A:25
15 km
20 km
C:12
4 km
B:5
10 km
20 km
D:0
Front
Vis
A - [ ] - 25
61


FIN