Algorísmica Avançada
Algorismes enumeratius

Sergio Escalera

Parciales y prácticas Notas

Algorismes enumeratius

- Exercicis PD
- Recorregut vs Cerca
- Backtracking
- Ramificació i poda

Programació dinàmica

- Exercici

		A	L	G	O	R	I	S	M	I	C	A
	0	1										
A	2	0										
V												
A												
N												
Ç												
A												
D												
A												

$$\gamma(i,j,k) = d(i,j,k) + \min\{\gamma((i-1,j-1),(i-1,j),(i,j-1)\times\{1,..,K\})\}$$

Programació dinàmica

1 1 1 OO 1 O 1

O O 1

1 1 1

	0	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	1	1
1	2	0	0	0	0	0	0	0	1	1
1	3									
1	4									
0	5									
0	6									
0	7									
1	8									

Programació dinàmica

- Floyd-Warshall

for $i = 1$ to n:
    for $j = 1$ to n:
        $\texttt{dist}(i, j, 0) = \infty$

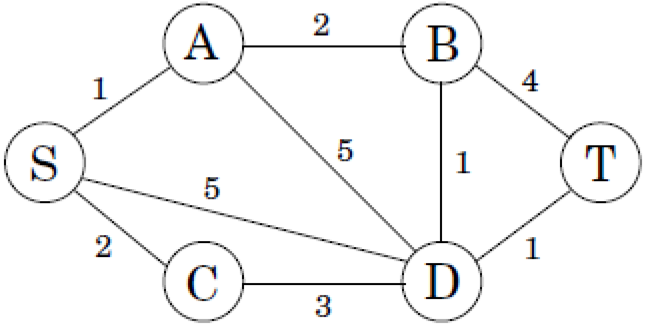
for all $(i, j) \in E$:
    $\texttt{dist}(i, j, 0) = \ell(i, j)$
for $k = 1$ to n:
    for $i = 1$ to n:
        for $j = 1$ to n:
            $\texttt{dist}(i, j, k) = \min\{\texttt{dist}(i, k, k-1) + \texttt{dist}(k, j, k-1),\ \texttt{dist}(i, j, k-1)\}$

- matriz(:,:,1) =

- Inf 1 Inf 2 5 Inf
- 1 Inf 2 Inf 5 Inf
- Inf 2 Inf Inf 1 4
- 2 Inf Inf Inf 3 Inf
- 5 5 1 3 Inf 1
- Inf Inf 4 Inf 1 Inf

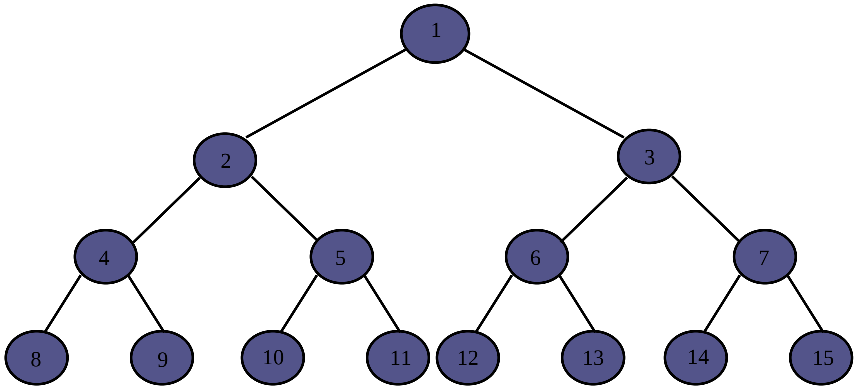
- matriz(:,:,2) =

- Inf 1 Inf 2 5 Inf
- 1 2 2 3 5 Inf
- Inf 2 Inf Inf 1 4
- 2 3 Inf 4 3 Inf
- 5 5 1 3 10 1
- Inf Inf 4 Inf 1 Inf



Cerca – exemple amb arbres

Arbres



Arbre binari

Conjunt de **nodes i arestes**, on el node superior es diu el **root** i els nodes externs es diuen **fulles**

En el cas dels **arbres binaris** cada node te com a **màxim 2 arestes sortints**

Un arbre binari, al **nivell *i* té com a màxim 2^i nodes** (suposant el root i=0)

Cerca - recorregut

Cóm podem recórrer un arbre binari?

Si denominem amb

- L: moviment a l'esquerre
- V: "visitar" el node
- R: moviment a la dreta

Tenim sis combinacions possibles de recorregut:

- LVR, LRV, VLR, VRL, RVL, RLV

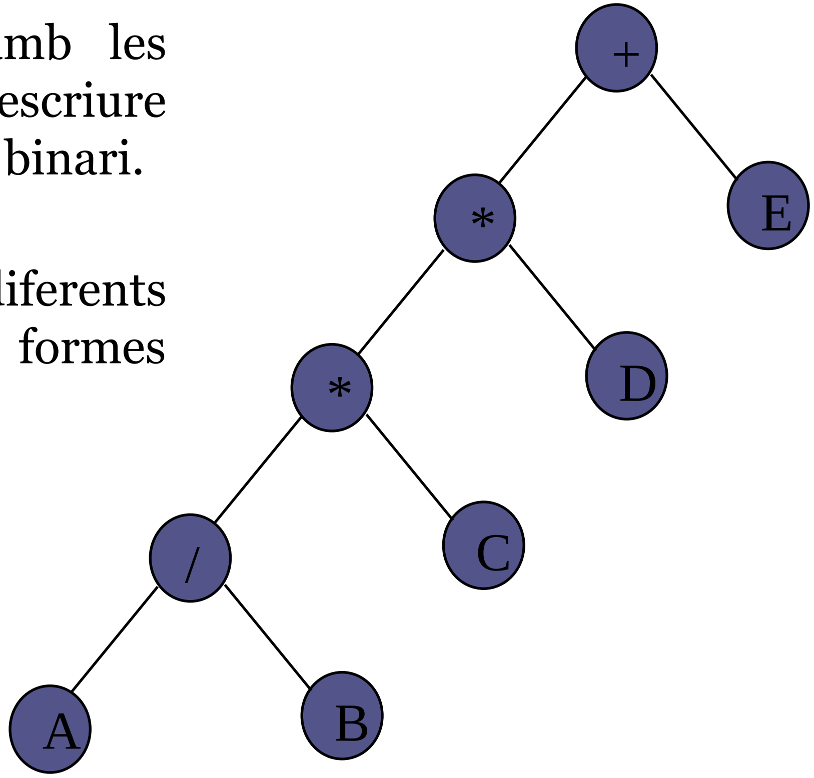
Si optem per realitzar primer el moviment a l'esquerra, tenim les tres primeres possibilitats

- LVR denominarem inordre
- LRV denominarem postordre, i
- VLR denominarem preordre

Cerca - recorregut

- Els recorreguts es corresponen amb les formes infixa, postfixa i prefixa d'escriure una expressió aritmètica en un arbre binari.

- Donat l'arbre binari, els diferents recorreguts porten a les següents formes d'escriure l'expressió:
  - Inordre LVR : A/B*C*D+E
  - Preordre VLR : +**/ABCDE
  - Postordre LRV : AB/C*D*E+



Backtracking

- El backtracking introdueix uns “criteris” per reduir la complexitat de la cerca recursiva.
- Aplicacions:
  - → Comprovar si un problema té solució
  - → Buscar múltiples solucions o una de totes les possibles

Backtracking

- Els esquemes de backtracking que veurem són directament aplicables a qualsevol tipus de graf (en molts exemples suposarem que són arbres)

```
algoritmo backtracking(i)
{
    si (i = n) {
        X ← X^i
    }
    sino {
        para v = cada valor posible de x_{i+1} {
            x_{i+1} ← v
            si (factible(X^i ∪ {x_{i+1}})) {
                X^{i+1} ← X^i ∪ {x_{i+1}}
                backtracking(i + 1)
            }
        }
    }
}
```

Volem arribar a n solucions del problema

Backtracking

- Una solució o totes?
  - → Una variable global serviria per finalitzar el programa quan trobem una solució.
  - → Si les volem totes hauríem d'imprimir els valors que va agafant X per no veure només l'últim

algoritmo backtracking(i)
{
  si $(i = n)$ {
    $X \leftarrow X^i$
  }
  sino {
    para $v =$ cada valor posible de x_{i+1} {
      $x_{i+1} \leftarrow v$
      si $(\text{factible}(X^i \cup \{x_{i+1}\}))$ {
        $X^{i+1} \leftarrow X^i \cup \{x_{i+1}\}$
        backtracking$(i+1)$
      }
    }
  }
}

Backtracking

- Reduïm camins

```
ruta_optima(i, j, ruta, ruta_optima)
{ Calcula la ruta óptima entre i y j y la concatena en la lista ruta_optima. }
    si i = j entonces
        medir_ruta(ruta)
        si es mejor que ruta_optima entonces ruta_optima ← ruta
    sino
        marcar i como visitado
        ∀ k : k no visitado, k adyacente a i :
        ⌈   añadir k al final de ruta
        |   ruta_optima(k, j, ruta, ruta_optima)
        ⌊   quitar k del final de ruta
        marcar i como no visitado
    fin
```

Backtracking

- Exemple: veure la utilitat del backtracking
- Problema de les 8 reines:
  - Volem col·locar 8 reines en un tauler d'escacs de 8x8 sense que hi hagi amenaça.
  - Comencem per una solució exhaustiva

```
bool ocho_reinas()
{
    for (int i=1; i<=8; i++)
        for (int j=1; j<=8; j++)
            for (int k=1; k<=8; k++)
                for (int l=1; l<=8; l++)
                    for (int m=1; m<=8; m++)
                        for (int n=1; n<=8; n++)
                            for (int o=1; o<=8; o++)
                                for (int p=1; p<=8; p++)
                                    if (solucion(i,j,k,l,m,n,o,p) return true;
    return false;
}
```

Backtracking

- Reduïm la complexitat: incloure restriccions
  - → Les reines han d'estar a diferents files i diferents columnes
  - → No podem estar a la mateixa diagonal:

- $T[i]+i == T[j]+j, \;(\searrow)$.
- $T[i]-i == T[j]-j, \;(\nearrow)$.

```
class ocho_reinas {

    bool factible(unsigned int s){
        int T[8];
        int i = s;
        for (int j=0; j<8; j++) {
            T[j] = i % 10;
            i = i / 10;
            if (T[j] == 0 || T[j] == 9) return false;
            for (int k=0; k<j; k++)
                if (T[k] == T[j] ||
                    T[k]-k == T[j]-j ||
                    T[k]+k == T[j]+j) return false;
        }
        return true;
    }

public:
    bool encontrada;
    unsigned int z;
    ocho_reinas() {
        z=12345678;
        while (!factible(z++) && z<87654321);
        z--;
        encontrada = (z<87654321);
    }
};
```

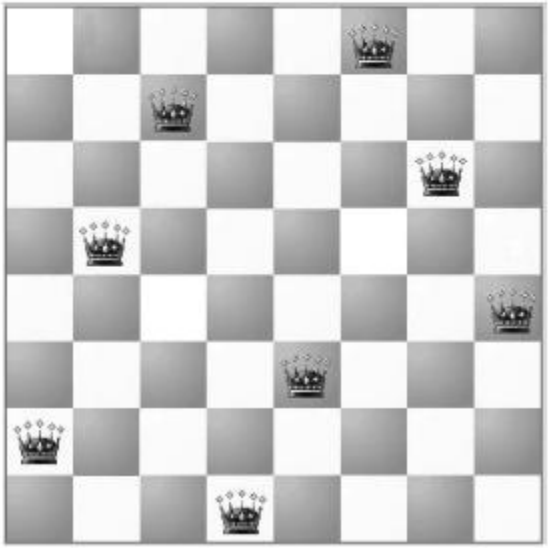
Backtracking

```
class ocho_reinas {

    bool factible(unsigned int s){
        int T[8];
        int i = s;
        for (int j=0; j<8; j++) {
            T[j] = i % 10;
            i = i / 10;
            if (T[j] == 0 || T[j] == 9) return false;
            for (int k=0; k<j; k++)
                if (T[k] == T[j] ||
                    T[k]-k == T[j]-j ||
                    T[k]+k == T[j]+j) return false;
        }
        return true;
    }

public:
    bool encontrada;
    unsigned int z;
    ocho_reinas() {
        z=12345678;
        while (!factible(z++) && z<87654321);
        z--;
        encontrada = (z<87654321);
    }
};
```

Hem reduït la complexitat, però no suficient



25713864

Backtracking

- Generalització: problema de les *n* reines

```
extern void imprimir(vector<int> T, int ns);

clas n_reinas {
    int ns;

    bool factible(int i) {
        for (int j=0; j<i; j++)
            if (T[i]==T[j] || T[i]-i==T[j]-j || T[i]+i==T[j]+j) return false;
        return true;
    }

    void busca(int i) {
        if (i==n) {
            imprimir(T,++ns);
            return;
        }
        for (int j=0; j<n; ++j) {
            T[i] = j;
            if (factible(i)) busca(i+1);
        }
    }

public:
    vector<int> T;

    n_reinas(int n) {
        ns = 0;
        T.crea(n);
        busca(0);
    }
    ~n_reinas() { T.destruye(); }
    bool hay_solucion() { return (ns>0); }
};
```

Backtracking

Backtracking - optimització

```
algoritmo BackTracking(ent k:entero;
     entsal  X:vector[1..n]de valor)
{Pre: X[1..k-1] es completable,
     cota(X,k-1)<CosteMejorSol}

para todo v en Ci hacer
  X[k]:=v;
  si (completable(X,k) ∧
     cota(X,k)<CosteMejorSol) entonces
     si Sol(X,k) entonces
                    MejorSol:= X;
             CosteMejorSol:= Coste(X)
     fsi;
     si k<n entonces
                    BackTracking(k+1,X)
     fsi;
   fsi
fpara
```

Backtracking - iteratiu

```
algoritmo BackTracking()

variable P es pila de nodo;
CosteMejorSol:=∞
pvacia(P); apilar(P,<X,1>)
mientras ¬vacia(P) hacer
  <X,k>:=cima(P); desapilar(P);
  para todo v en Ci hacer
     X[k]:=v;
     si (completable(X,k) ∧
        cota(X,k)<CosteMejorSol) entonces
        si Sol(X,k) entonces
                        MejorSol:= X;
             CosteMejorSol:= Coste(X)
        fsi;
        si k<n entonces
                        apila(P,<X,k+1>)
        fsi;
     fsi
  fpara
fmientras
devuelve (MejorSol, CosteMejorSol)
```

```
tipo nodo es tupla
   X: vector[1..n]de valor
   k: [1..n+1]
ftipo // X[1..k-1] es una
asignación parcial
```

Ramificació i poda

- La ramificació i poda implementen l'algorisme de backtracking, però no a l'inversa
- L'objectiu de ramificació i poda consisteix en reduir l'espai de cerca, per això s'introdueixen **cotes**. Guardant informació sobre l'execució d'un algorisme podem fer ús d'aquestes cotes per **ramificar o podar**

Ramificació i poda

- Per a les cotes hem de guardar informació dels estats no explorats per retomar-les en el procés d'exploració
- Distingim 2 tipus de cotes: heurístiques i relaxacions:
  - → Heurístiques: reflexionem en el nostre problema
  - → Relaxacions: si hem de complir *n* variables, primer assegurem $k<n$ i després les restants

Ramificació i poda

- Exemple d'assignació de costos:
  - → Assignar un projecte a cada empresa minimitzant el cost total

	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

  - → Inicialitzem una cota superior

$$a \rightarrow A, b \rightarrow B, c \rightarrow C, d \rightarrow D$$
$$z^* \leq 11 + 15 + 19 + 28 = 73$$

Ramificació i poda

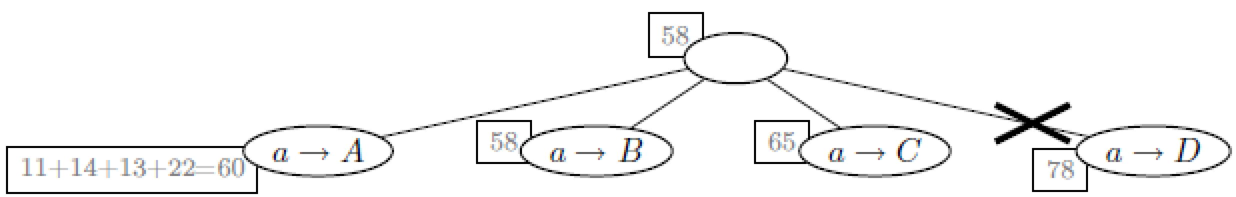
- Cota inferior: suma mínims de cada columna

$$z^* \geq 11 + 12 + 13 + 22 = 58$$

	A	B	C	D
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

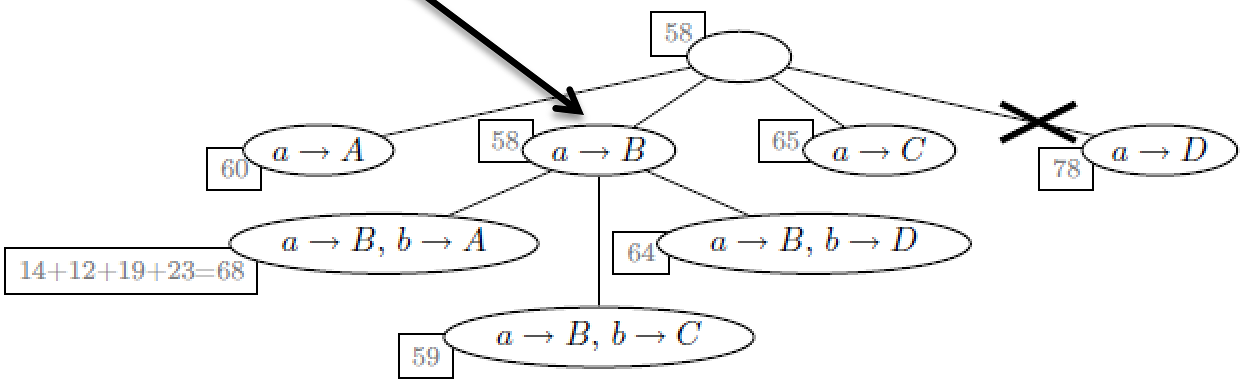
Coste de las empresas a,b,c y d, por los proyectos A,B,C y D.

- Comencem la cerca: $58 \leq z^* \leq 73$

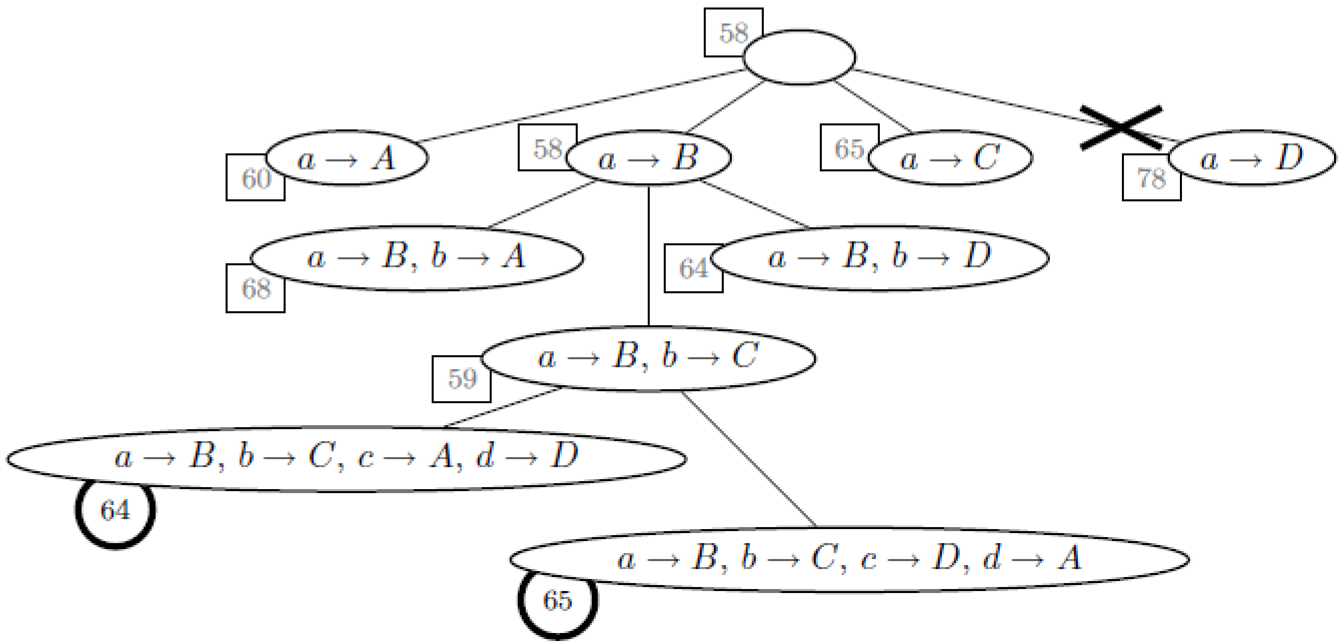


Ramificació i poda

- *Branching:* quina rama mereix la pena ser explorada?



Ramificació i poda

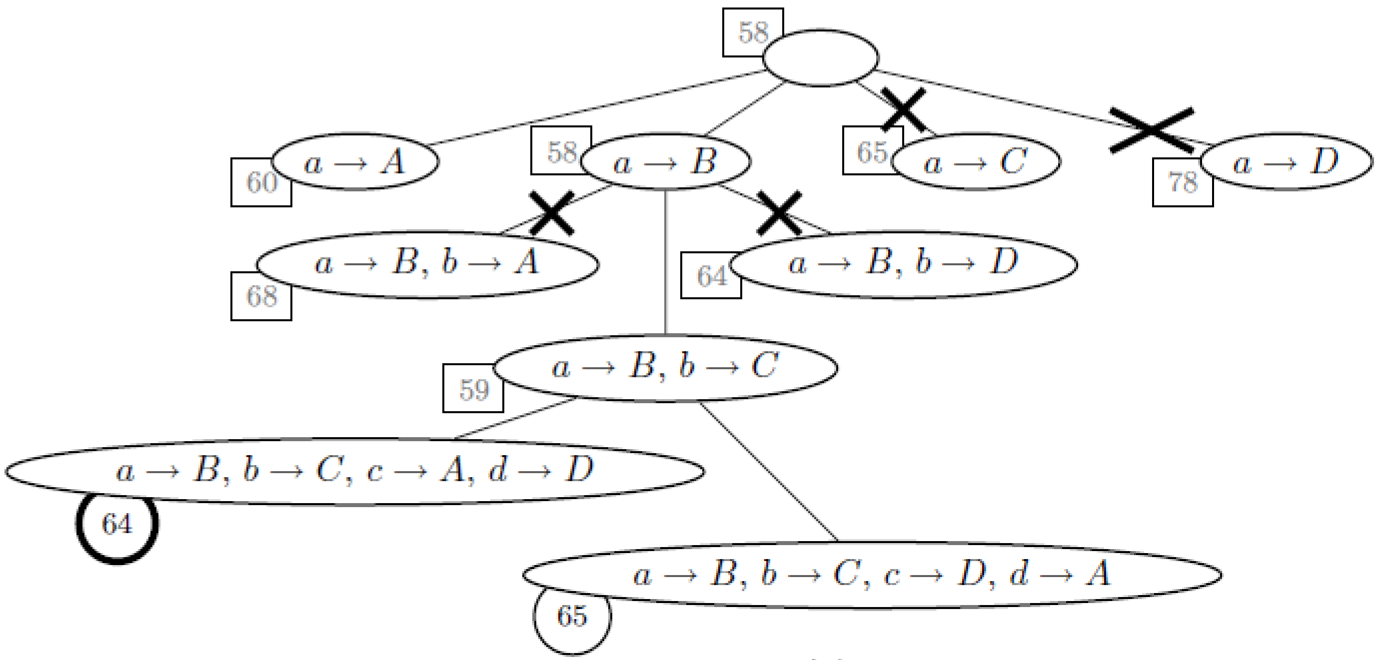


- Ja no és una cota, sinó un possible valor de la funció objectiu

$$58 \leq z^* \leq 64$$

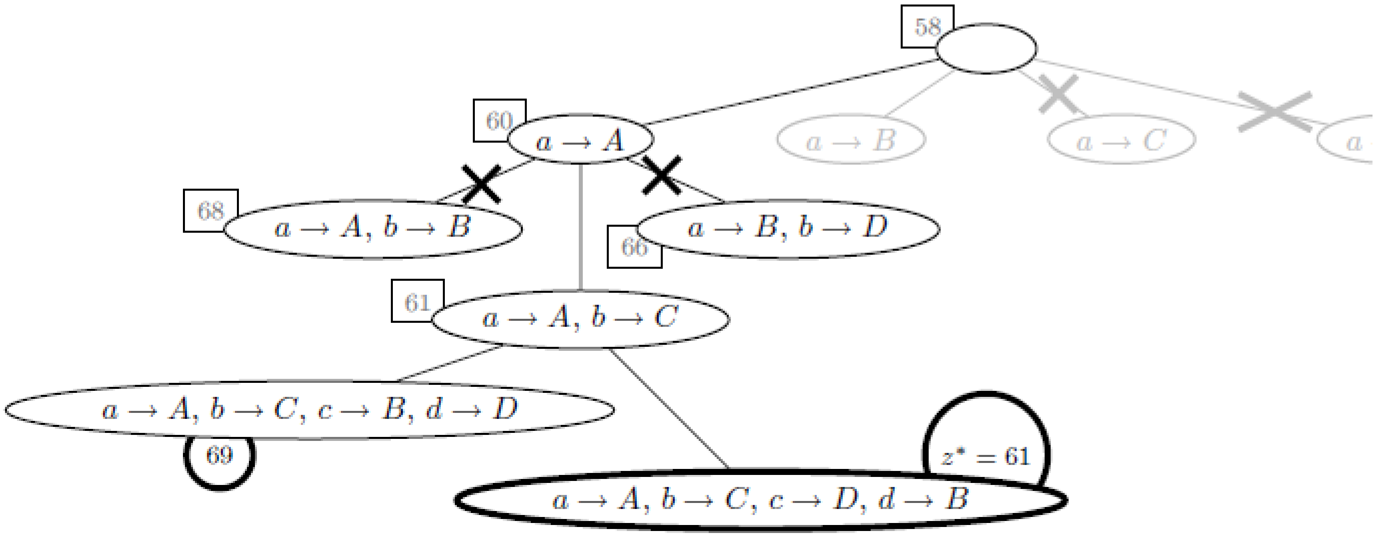
Ramificació i poda

$$58 \leq z^* \leq 64$$



Ramificació i poda

- Solució: 61



Ramificació i poda - Exercici

- Fes l'arbre de ramificació i poda de la següent taula (seguint el problema de l'exemple anterior). Numera els passos i actualitza els valors de les cotes.

	A	B	C	D	E
a	1	13	3	18	2
b	3	5	9	20	6
c	5	10	2	17	5
d	7	2	10	21	10
e	9	10	15	16	4

Ramificació i poda - cua amb prioritats!

“Probabilitat”

```
tipo nodo es tupla
    X: vector[1..n]de valor
    k: [1..n+1]
ftipo // X[1..k-1] es una asignación
parcial
```

```
algoritmo BranchAndBound()

variable C es cola_prioritaria de nodo;
CosteMejorSol:= ∞
cvacia(C); encolar(C,<X,1,pr(X,1)>)
mientras ¬ vacia(C) hacer
  <X,k>:=primero(C); desencolar(C);
   para todo v en Ci hacer
     X[k]:=v;
     si (completable(X,k) ∧
        cota(X,k)<CosteMejorSol) entonces
        si Sol(X,k) entonces
                        MejorSol:= X;
             CosteMejorSol:= Coste(X)
        fsi;
        si k<n entonces
                        encola(C,<X,k+1,pr(X,k+1)>)
        fsi;
     fsi
   fpara
fmientras
devuelve (MejorSol, CosteMejorSol)
```

Ramificació i poda

- La **clau**: funció de **prioritat i cota**
- Problema del viatjant de comerç
  - → Passar per tots els nodes minimitzant el cost de la ruta, passant només un cop per cada node i acabant en el node de sortida (circuit ***hamiltonià***)

Ramificació i poda

- Formalització:
  - Sean $G=(V,A)$ un grafo orientado,
    $V=\{1,2,\ldots,n\}$,
    $D[i,j]$ la longitud de $(i,j)\in A$,
    $D[i,j]=\infty$ si no existe el arco (i,j).
  - El circuito buscado empieza en el vértice 1.

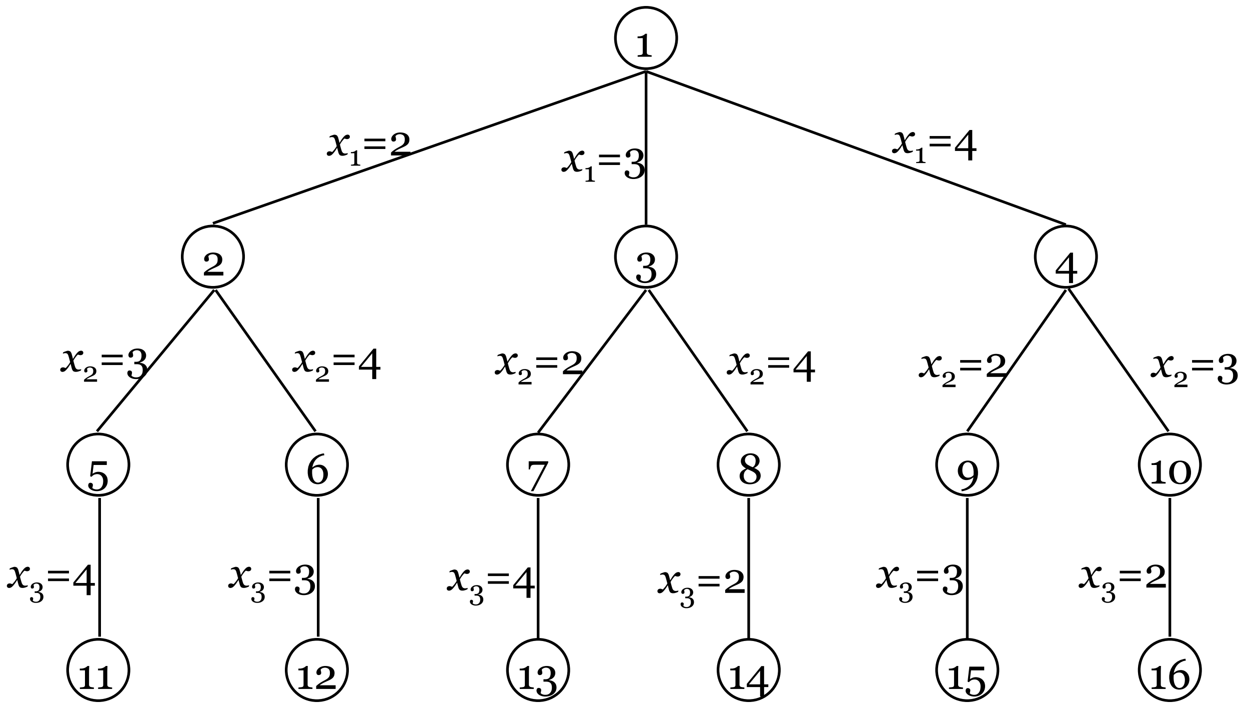
  - Candidatos:
    $E = \{ 1,X,1 \mid X$ es una permutación de $(2,3,\ldots,n) \}$
    $|E| = (n-1)!$

  - Soluciones factibles:
    $E = \{ 1,X,1 \mid X = x_1,x_2,\ldots,x_{n-1}$, es una permutación de $(2,3,\ldots,n)$ tal que $(i_j,i_{j+1})\in A$, $0<j<n$, $(1, x_1) \in A$, $(x_{n-1},1) \in A\}$

  - Funcion objetivo:
    $F(X)=D[1,x_1]+D[x_1, x_2] + D[x_2, x_3]+\ldots+D[x_{n-2}, x_{n-1}]+D[x_n,1]$

Ramificació i poda

- Representació per |V|=4



Ramificació i poda

- Definición de una *cota*(X,k) muy sencilla:
  - Suma de aristas ya escogidas
  - $cota(X,k)=D[1,X[1]]+\Sigma_{i=1..k-2}\, D[X[i],X[i+1]]$
- Ejemplo: (n=5)

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Ramificació i poda

- Si se elige t como el mínimo de los elementos de la fila (columna) i-ésima y se resta t de todos los elementos de esa fila (columna), la fila resultante es **reducida**.
- La cantidad total L restada de filas y columnas es una cota inferior de la longitud de un hamiltoniano de longitud mínima

Reducción de la matriz, $L = 25$.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Ramificació i poda

- Cálculo de la cota inferior para los nodos distintos de la raíz y de las hojas:
  - Sea *A* la matriz de distancias reducida para el nodo *y*.
  - Sea *x* un hijo de *y* que corresponda a incluir el arco (*i*,*j*) en el recorrido y que no sea hoja.
  - La matriz *B* reducida para *x*, y por tanto cota(x), se calcula de la siguiente forma:

  1. Cambiar todos los elementos de la fila *i* y de la columna *j* de *A* por ∞.
     - Esto evita el incluir más arcos que salgan de *i* o lleguen a *j*.
  2. Cambiar el elemento (*j*,1) de *A* por ∞
     - Esto evita considerar el arco (*j*,1).
  3. *B* es la matriz que se obtiene al reducir todas las filas y columnas de la matriz resultante (excepto aquéllas formadas sólo por "∞").

  - Si *r* es el valor total restado en el paso (3): cota(x)=cota(y) + D[i,j] + r

Ramificació i poda

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

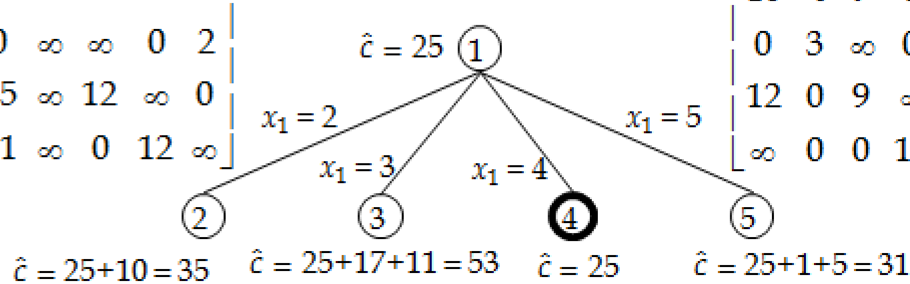
Grafo original.

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Matriz reducida, $L = 25$.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

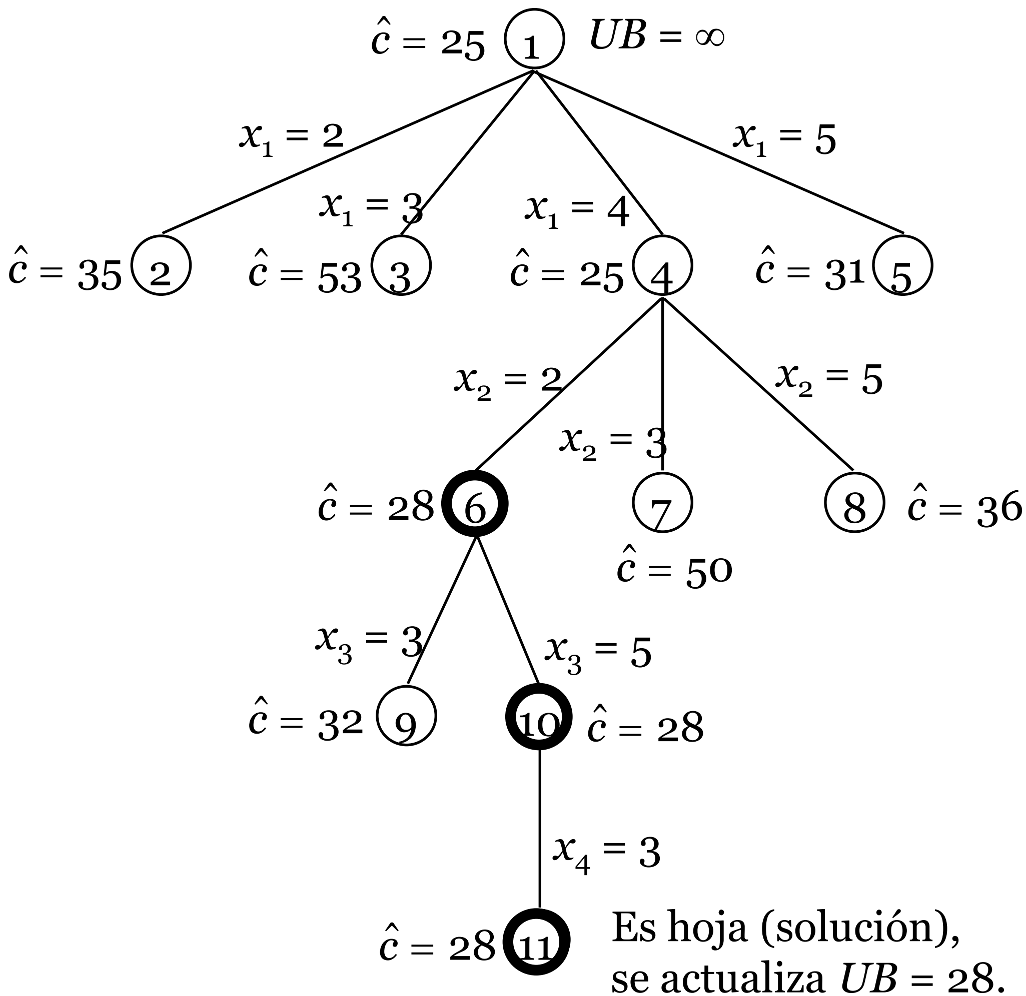
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$



$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Ramificació i poda



El siguiente nodo en curso sería el 5, pero cota(5)>UB luego el algoritmo termina y el hamiltoniano mínimo es 1,4,2,5,3,1.