Diseño y Análisis de Algoritmos

TEMA 5. VUELTA ATRÁS

Universidad
Rey Juan Carlos

Tema 5. Vuelta atrás (*Backtraking*)

Introducción

- Introducción
- Estrategia general
- Esquema de la técnica
- Ejemplos
  - Mochila 0-1 con *n* objetos
  - *N* Reinas
  - Laberinto
  - Coloreado de grafos
  - Ciclos hamiltonianos

Tema 5. Vuelta atrás (*Backtraking*)

Introducción

- Algunos problemas requieren realizar una **búsqueda exhaustiva** por el espacio de posibles soluciones hasta:
  - Encontrar una solución
  - Constatar que no existe
- El esquema **Vuelta Atrás** genera de manera **sistemática** posibles soluciones
- El espacio de posibles soluciones se puede plantear en términos de un **grafo (árbol) implícito**

Grafo implícito:

- Se conoce la **descripción** de nodos y arcos/aristas
  - **Nodos**: Soluciones parciales
  - **Arcos/aristas**: Incorporación de un elemento más en la solución parcial
- Se construye cada nodo según se realiza el **recorrido**
- Una vez examinados, los nodos se pueden **liberar**

Su uso permite:

- Manipulación de grafos (árboles) **infinitos**
- Ahorro de **memoria**

Tema 5. Vuelta atrás (*Backtraking*)

Introducción

- Realiza un recorrido en profundidad en un grafo dirigido implícito y sin ciclos $\Rightarrow$ *Árbol de Exploración*

Ámbito de aplicación:

- Encontrar **una** solución: El algoritmo para cuando encuentra la primera solución factible
- Encontrar **todas** las soluciones: El algoritmo recorre todo el espacio de soluciones
- Encontrar la **mejor** solución: El algoritmo recorre todo el espacio de soluciones, devolviendo la mejor

Tema 5. Vuelta atrás (*Backtraking*)

Estrategia general

- A medida que **progresa** el recorrido se van **construyendo** soluciones parciales (*n*-tupla)
- Si se llega a una **hoja** y la solución no es completa el recorrido no tiene éxito y **vuelve atrás**:
  - **Elimina** el último elemento de la tupla
  - Intenta construir añadiendo otro **elemento diferente**
- Si se llega a una **hoja** que es una s**olución completa** el recorrido ha tenido éxito:
  - Si sólo se busca **una** solución, el algoritmo se **detiene**
  - En caso contrario, se realiza una **Vuelta Atrás**

Una *n*-tupla es una **solución** si verifica las **restricciones**:

- **Explícitas**: indican los valores que puede tomar cada componente de una tupla solución (*Soluciones factibles* + *no factibles*)
- **Implícitas**: relaciones que se han de establecer entre las componentes de una solución (Solución *factible*)

Tema 5. Vuelta atrás (*Backtraking*)

Estrategia general

- **Espacio de soluciones**: conjunto de tuplas que satisfacen las restricciones explícitas (*factibles* + *no factibles*)
- **Árbol de exploración**: forma de estructurar el espacio de soluciones factibles
- **Nodo del árbol de exploración**: tupla parcial (*prometedora*) o completa (*solución factible*)
- Función de poda o **test de factibilidad**: determina cuándo una tupla parcial es o no prometedora

Tema 5. Vuelta atrás (*Backtraking*)

Estrategia general

- Un problema **puede resolverse** con un algoritmo vuelta atrás cuando

  la solución puede expresarse como una *n*-tupla $[x_1, x_2, ..., x_n]$ siendo x_i una componente elegida en cada etapa de entre un conjunto finito de valores
- **Recorrido en profundidad** sobre el árbol (implícito) de exploración
- Cada **etapa** representará un **nivel** en el árbol de búsqueda (añadir un elemento a una tupla)

Coste en el caso peor:

- Orden del tamaño del **espacio de soluciones**
- Suele ser, al menos, **exponencial**

Tema 5. Vuelta atrás (*Backtraking*)

Esquema de la técnica

```
procedimiento VueltaAtras(v[1..k])
 si EsSolucion(v[1..k]) entonces
     escribir v {o almacenar v}
  sino
  para ei={e1,..,ej} hacer {ei: comp.    disp.}
       v[k+1] ← ei
    si EsPrometedor(v[1..k+1]) entonces
     VueltaAtras(v[1..k+1])
    fsi
     fpara
 fsi
fprocedimiento
```

Tema 5. Vuelta atrás (*Backtraking*)

Esquema de la técnica

- `v[1..k]`: una tupla (un nodo)
- `EsSolucion()`: una función que indica si una tupla es una solución
- `{e1,..,ej}`: conjunto de posibles sucesores dada una tupla (componentes disponibles)
- `EsPrometedor(v[1..k+1])`: una función que determina si una tupla se puede podar

Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. El problema de la mochila 0-1 y *n* tipos de objetos

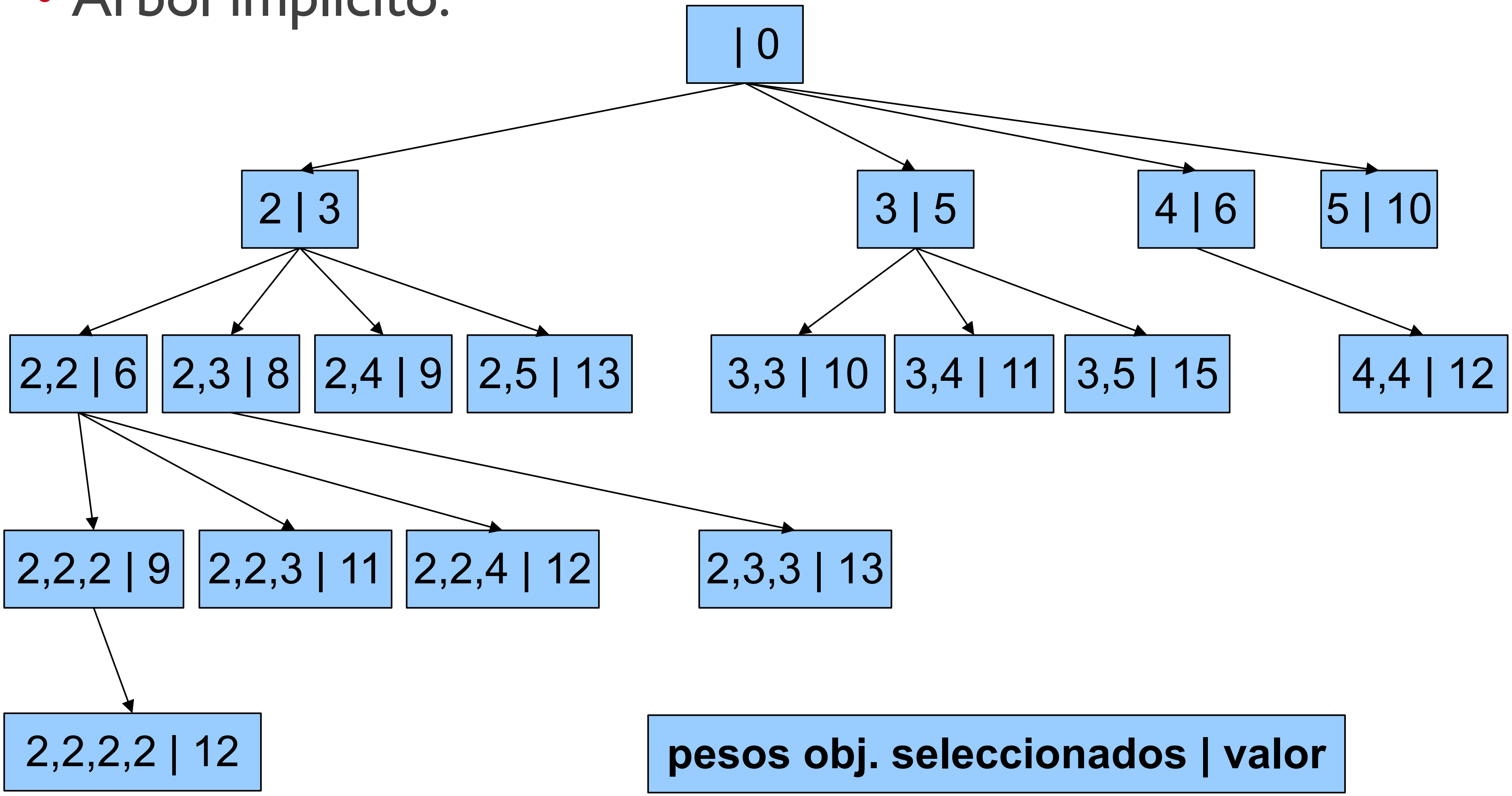
- Variante del problema de la mochila.
- No se pueden dividir objetos (0-1).
- En lugar de tener *n* objetos, tenemos *n* tipos de objetos.

$$\max \sum_{i=1}^{n} x_i v_i \quad \text{sujeto a} \quad \sum_{i=1}^{n} x_i w_i \leq W$$

$n = 4,\ W = 8$				
w	2	3	4	5
v	3	5	6	10

Ejemplo. El problema de la mochila 0-1 y *n* tipos de objetos

- Árbol implícito:



Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. El problema de la mochila 0-1 y *n* tipos de objetos

```
funcion mochilaVA(i,r:entero):entero
{i: construir usando los elementos i al n y cuyo
 peso no sobrepase r}
 b ← 0;
 para k ← i hasta n hacer
   si (w[k] ≤ r) entonces
     b ← max(b,v[k]+mochilaVA(k,r-w[k]));
   fsi
 fpara
 devolver b
ffun
```

Nota: se supone que `w[k]`, `v[k]` y `n` se encuentran disponibles.

Ejemplo. Las *n* Reinas

Problema: Colocar *n* reinas en un tablero de forma que no se puedan comer

Una reina puede comer a otra si están en la misma: Fila, Columna, Diagonal

Posibles planteamientos:

- Ir probando todas las formas de colocar *n* reinas en un tablero
- Para *n* = 8, el nº se situaciones posibles > 4.000.000.000.

coste demasiado elevado

Posibles planteamientos:

- Introducir una **restricción**: no poner más de una reina en una fila. La implementación del tablero sería con un vector:

  tablero[1..*n*] donde la reina de la fila *i* está en la columna tablero[*i*]

- Una restricción no es suficiente ya que las reinas pueden coincidir en la misma:
  - Columna
  - Diagonal

Ejemplo. Las *n* Reinas

```
procedimiento n_reinas()
para i1 ← 1 hasta n hacer
  para i2 ← 1 hasta n hacer
  ...
     para in ← 1 hasta n hacer
          sol ← [i1, i2, i3, ...in]
          si solución(sol) entonces
               escribir sol
          fsi
     fpara
  ...
  fpara
fpara
fprocedimiento
```

- Se van variando las posiciones de las n reinas
- solución() tendría en cuenta las restricciones
- Si n = 8, el nº de situaciones 88 ⍰ 16.000.000

- Se puede **añadir** la restricción de que no puede haber más de una reina en la misma columna
  - La implementación mediante un vector sería la misma
  - El vector contendrá 8 valores diferentes

Para ello:

- se fija un valor
- se generan las permutaciones de los *n*-1 valores restantes

El nº de situaciones $8! \approx 40.000$

Característica común de las 3 estrategias:

hasta que no se colocan las 8 reinas no se comprueba si esa propuesta es una solución

Enfoque vuelta atrás:

- permite construir la solución por etapas
- en cada etapa se **comprueba** si la **solución parcial** que se va produciendo es **prometedora**

Un vector $v[1..k]$ de enteros entre 1 y n es ***k*-prometedor** para $1 \leq k \leq n$ si las reinas en $(1, v[1])$, $(2, v[2])$ …$(k, v[k])$ no se amenazan entre ellas

- Si $k \leq 1$ todo vector v es prometedor
- Si $k = n$ el vector v es una solución

Ejemplo. Las *n* Reinas

Descripción del árbol de exploración:

Sea $T = (N, A)$ un árbol t.q.:

- N = vectores k-prometedores, $0 \leq k \leq n$
- $(u, v) \in A$ si y solo si existe un entero k, $0 \leq k \leq n$ t.q.:
  - u es k-prometedor
  - v es $k+1$-prometedor
  - $u[i] = v[i]$, $i \in [1..k]$

La estructura de T será:

- raíz: $k = 0$ (vector vacío)
- hojas:
  - si $k = n$ solución completa
  - si $k < n$ sin posible solución en esa rama

Para decidir si un nodo es *k*-prometedor:

- se parte de un (*k*-1)-prometedor
- se comprueba la última reina (*k*)

La implementación del algoritmo necesita:

- sol[1..*k*]: vector *k*-prometedor
- col: conjunto de columnas ocupadas
- diag45: conjunto de diagonales positivas ocupadas (columna - fila)
- diag135: conjunto de diagonales negativas ocupadas (columna + fila)

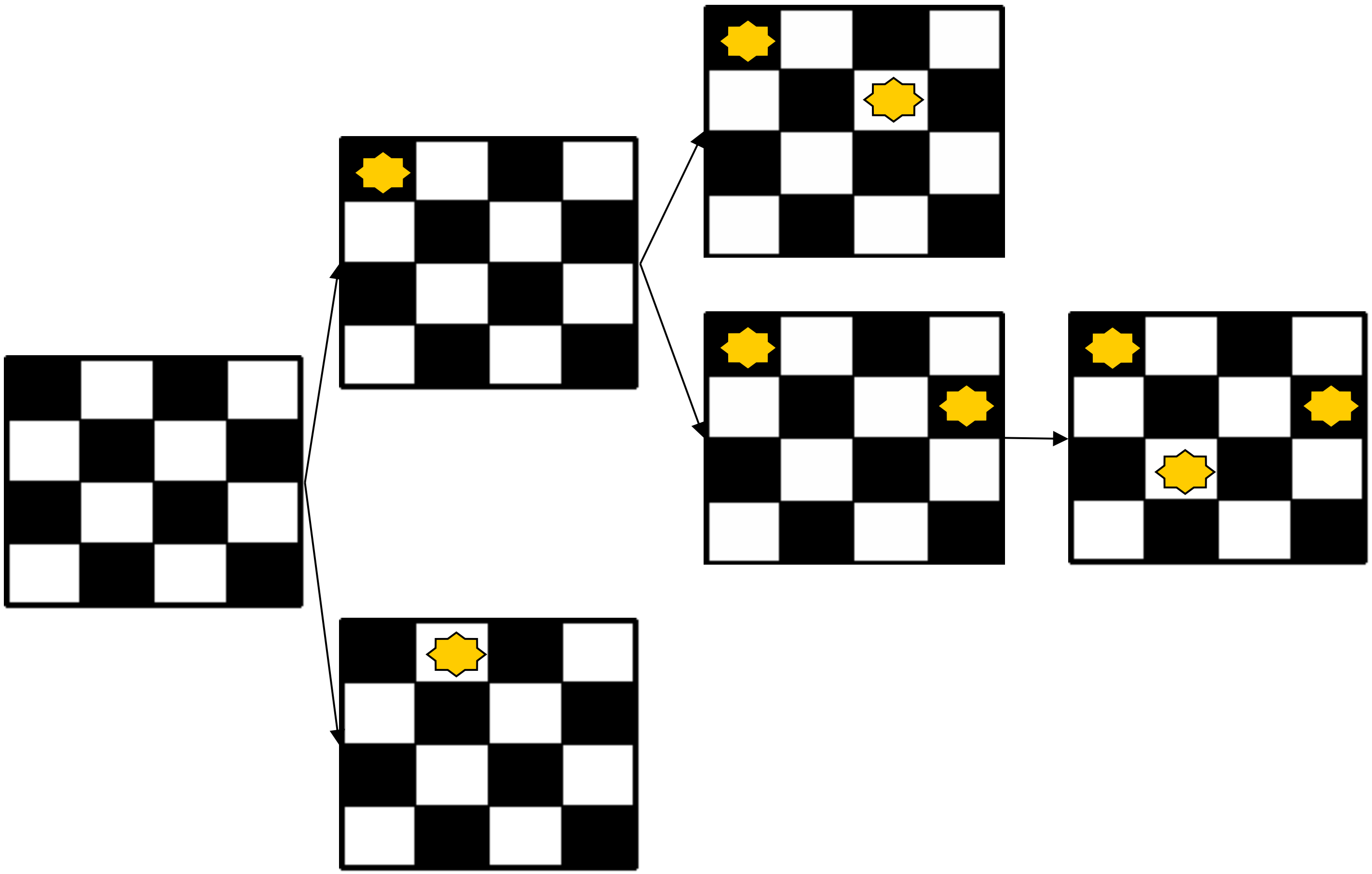
Ejemplo. Las *n* Reinas

```
procedimiento n_reinas_va(k, col, diag45, diag135)
 si k = n entonces
  escribir sol
 sino
  para j ← 1 hasta n hacer
    si j ∉ col y j-k ∉ diag45 y j+k ∉ diag135 entonces
       sol[k+1] ← j
          n_reinas_va(k+1,col∪{j},diag45∪{j-k},diag135∪{j+k})
    fsi
  fpara
 fsi
fprocedimiento
```

La llamada al procedimiento sería: `n_reinas_va(0, ∅, ∅, ∅)`

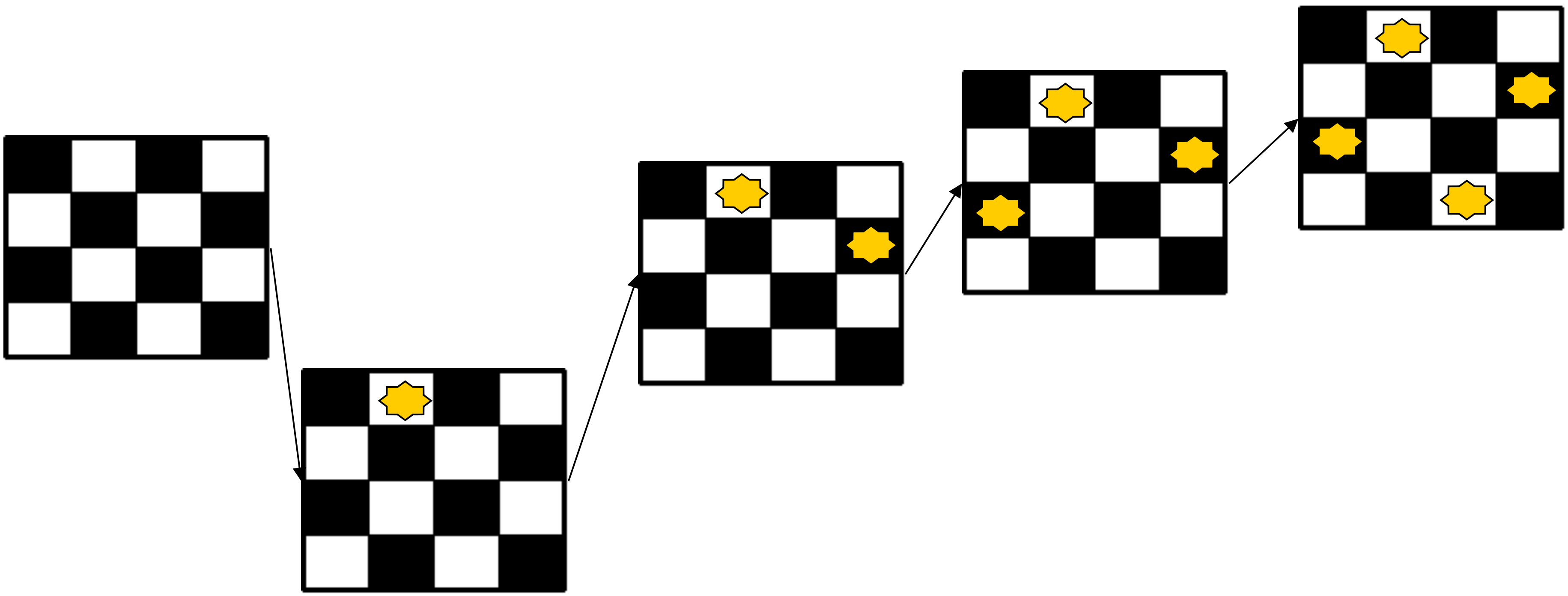
Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. Las *n* Reinas



Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. Las *n* Reinas



Ejemplo. El laberinto

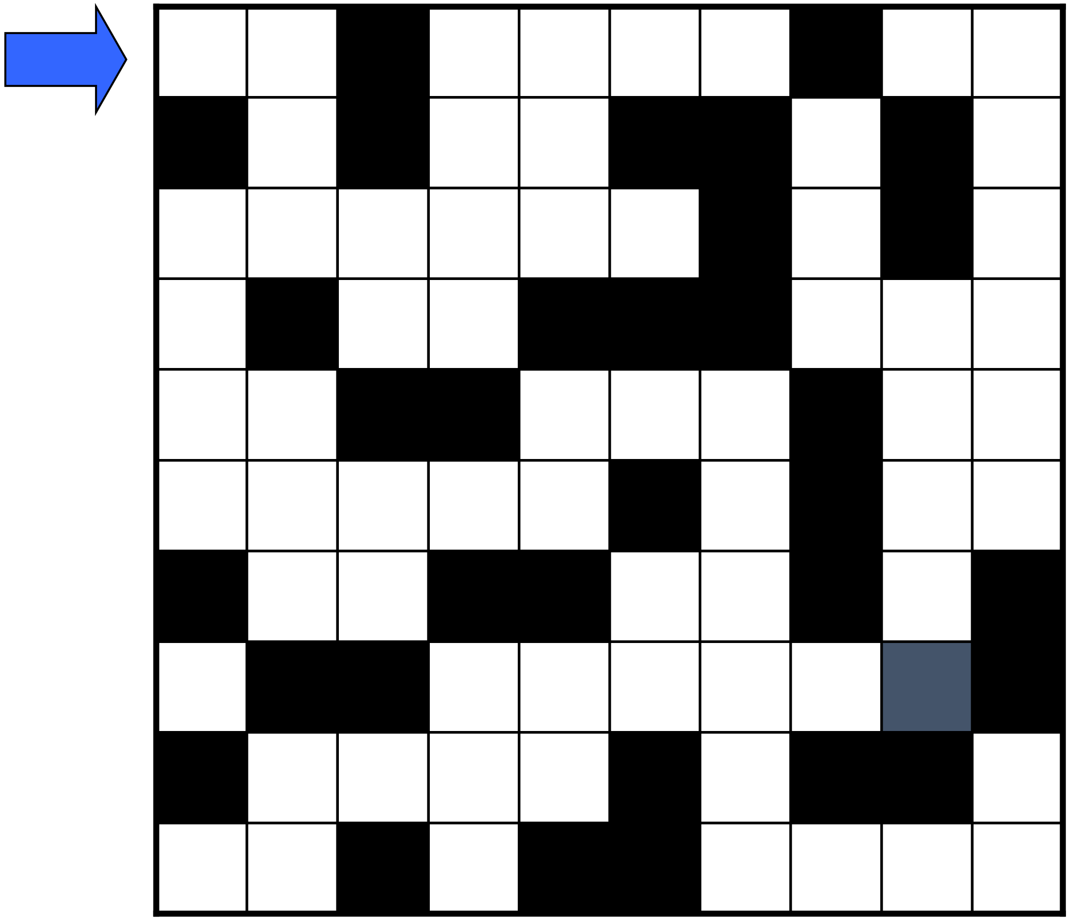
Una matriz bidimensional NxN representa un laberinto.

Si la casilla es transitable contiene un 0, si no contiene un ∞.

Las casillas [1,1] y [N,N] corresponden a la entrada y salida del laberinto y siempre son transitables.

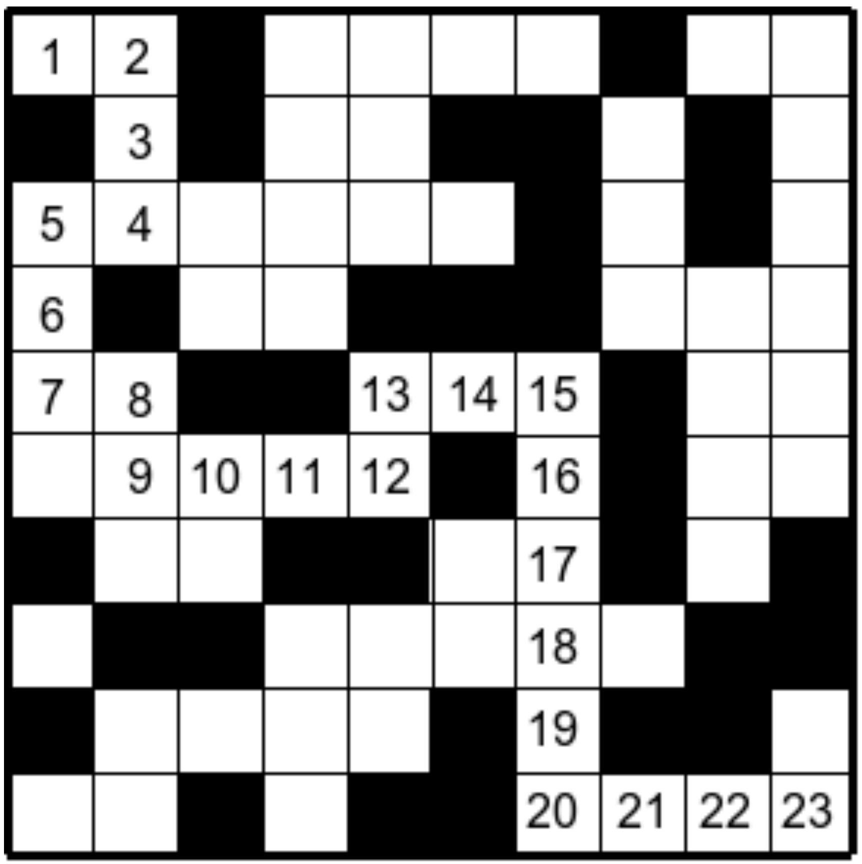
Se pide encontrar un camino, si existe.

Ejemplo. El laberinto



Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. El laberinto



- Devolvemos el camino recorrido.
- Marcamos cada casilla visitada en el orden en el que se visita.
- Si se vuelve atrás, se marcan las casillas con 0.

Ejemplo. El laberinto

```
procedimiento Laberinto(f,c,k: entero; VAR tab:[1..n][1..n]);
   tab[f,c] ← k
   si (f = N) y (C = N) entonces
    MostrarSolucion(tab)
   sino
    si Esposible(tab,f,succ(c)) entonces
        Laberinto(f,succ(c),succ(k),tab)
    fsi;
    si Esposible(tab,succ(f),c) entonces
        Laberinto(succ(f),c,succ(k),tab)
    fsi;
    si Esposible(tab,pred(f),c) entonces
        Laberinto(pred(f),c,succ(k),tab)
    fsi;
    si Esposible(tab,f,pred(c)) entonces
        Laberinto(f,pred(c),succ(k),tab)
    fsi;
    tab[f,c] ← 0
fprocedimiento;
```

Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. El laberinto

```
funcion Esposible(tab: [1..n][1..n]; f,c: integer);
  si (1 ≤ f ≤ n) y (1 ≤ c ≤ n) entonces
   devolver (tab[f,c] = 0)
  sino
   devolver FALSO;
ffuncion
```

Ejemplo. Coloreado de grafos

Problema: sea G = <N,A> conexo y m > 0. Se desea determinar todas las formas de colorear los nodos de G utilizando como máximo m colores

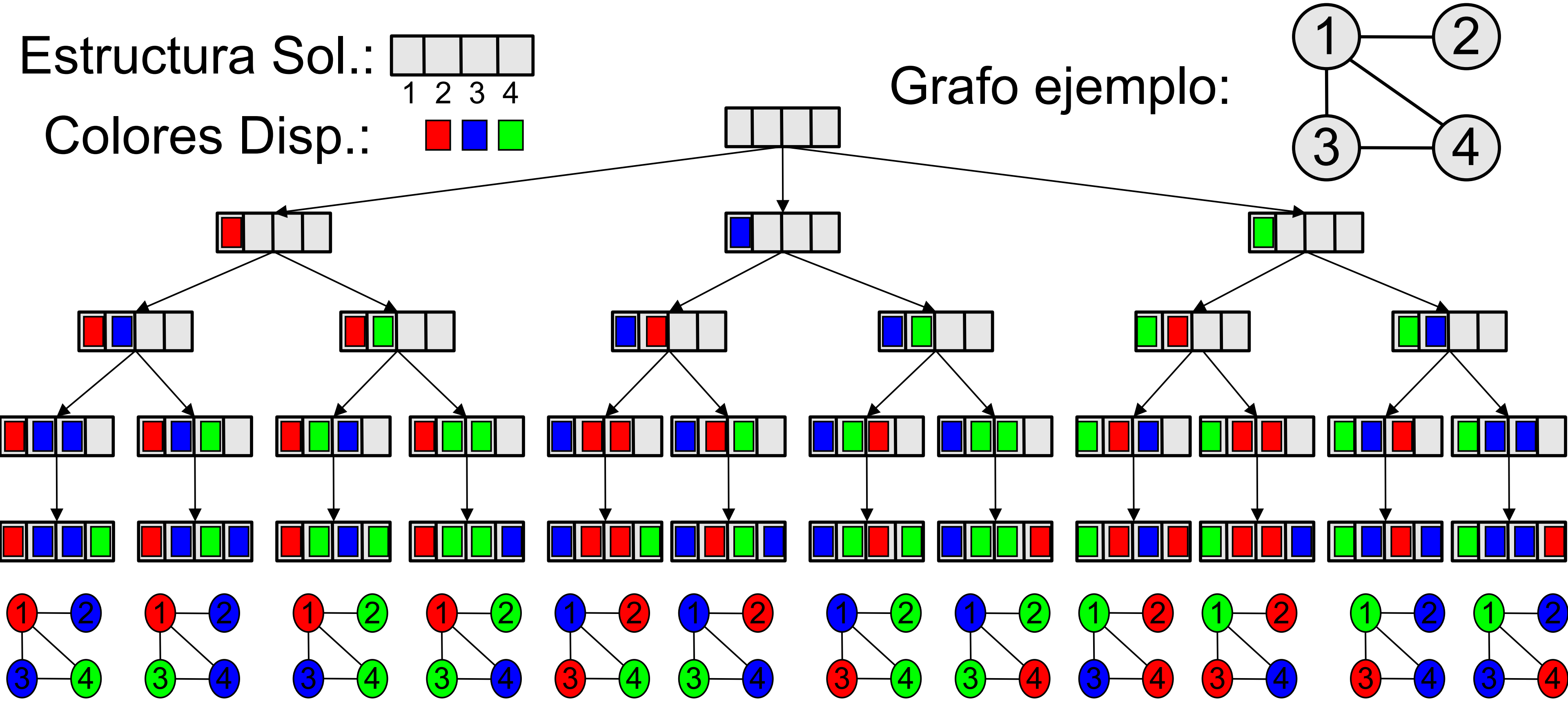
Restricción: no puede haber dos nodos adyacentes con el mismo color

Solución: vector x[1..n], siendo n el número de nodos de G. Cada elemento x[i], $1 \leq i \leq n$, representa el color asignado al nodo i, $1 \leq x[i] \leq m$

Tema 5. Vuelta atrás (*Backtraking*)

Ejemplo. Coloreado de grafos

- El árbol de exploración se puede organizar como un árbol de grado máximo *m* y altura *n*+1:



Ejemplo. Coloreado de grafos

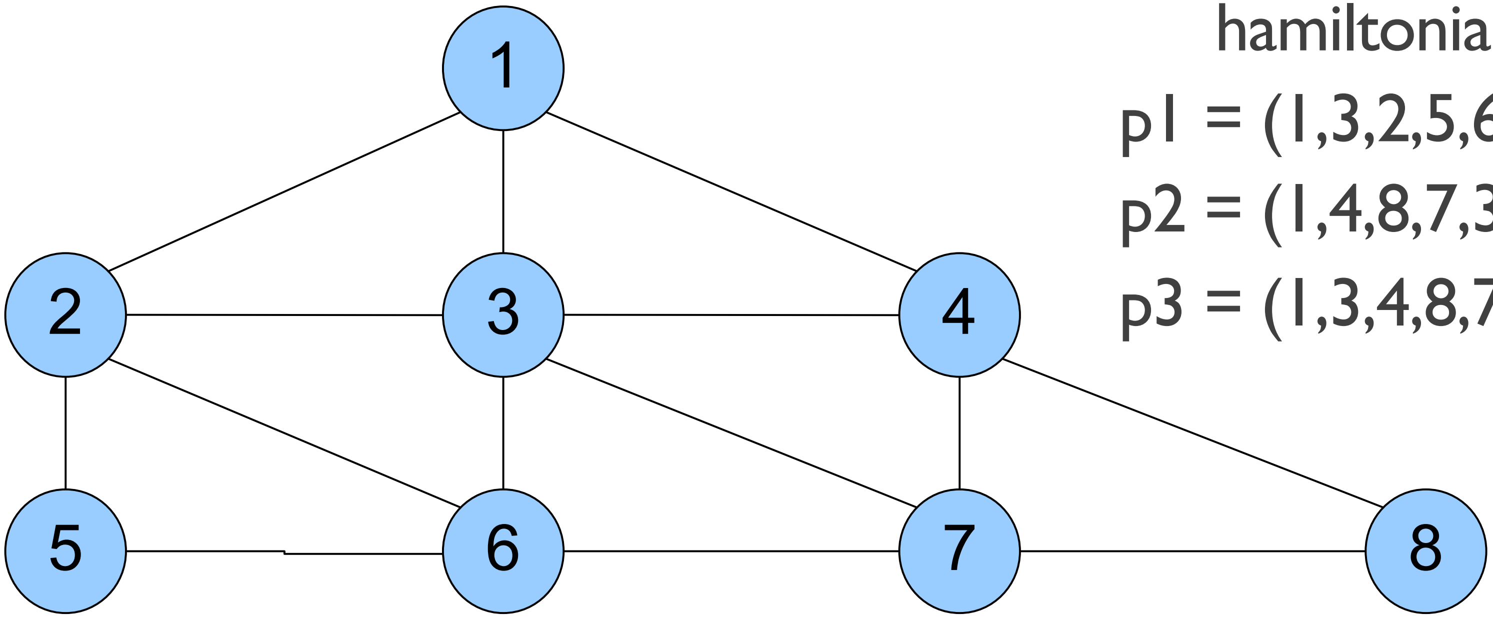
```
Procedimiento Colorear(k: entero; VAR solucion:[1..n]);
   solucion[k]  ← 0
   repetir
    solucion[k] ← solucion[k] + 1
    si valido(k,solucion) entonces
        si k < n entonces
            Colorear(k+1,solucion)
        sino
            MostrarSolucion(solucion)
        fsi
    fsi
   hasta que solucion[k] = m
fprocedimiento
```

Ejemplo. Coloreado de grafos

```
funcion valido(k: entero; sol:[1..n]): boolean
var correcto:booleano
   correcto ← VERDADERO
   ConjAdyacentes ← Adyacentes(k)
   para cada i ∈ ConjAdyacentes hacer
      si (sol[i] = sol[k]) entonces
         correcto ← FALSO
      fsi
   fpara
   devolver correcto
ffuncion
```

Ejemplo. Ciclos hamiltonianos

Problema: Sea G={V,A} conexo con n vértices. Un ciclo hamiltoniano es un camino con n+1 vértices (x1x2...xn+1) que visita una vez cada vértice y vuelve al vértice inicial.



Ejemplos de ciclos hamiltonianos:

p1 = (1,3,2,5,6,7,8,4,1)

p2 = (1,4,8,7,3,6,5,2,1)

p3 = (1,3,4,8,7,6,5,2,1)

En el vector solución $(x_1,x_2...x_{n+1})$ x_i representa el vértice visitado en *i*-ésimo lugar.

Dada una tupla incompleta $(x_1...x_{k-1})$ los valores posibles para x_k son:

- x_k, $k=1$: x_1 puede ser cualquiera. Elegimos $x_1=1$.
- x_k, $1<k<n$: debe ser distinto de $x_1,...,x_{k-1}$ y conectado con x_{k-1}.
- x_k, $k=n$: x_n solo puede ser el vértice que queda sin visitar. Además, debe estar conectado con x_{n-1} y con x_1.

Ejemplo. Coloreado de grafos

```
procedimiento HamiltonianoVA(VAR x:[1..n]; k: integer)
var c: TipoConjunto
    c ←adyacentes(x[k-1]) {c es un conjunto de adyacentes}
    mientras not esvacio(c)hacer
        candidato ← DameElemento(c)
        QuitarElemento(c,candidato)
        si Prometedor(x,candidato,k-1) entonces
            x[k] ← candidato
            si (k=n) entonces
                 si SonAdyacentes(candidato,x[1]) entonces
                     imprimir(x) {o almacenar(x)}
            sino
                 HamiltonianoVA(x,k+1)
            fsi
        fsi
    fmientras
fprocedimiento
```

Ejemplo. Coloreado de grafos

```
funcion Prometedor(x:[1..n]; elem, lim: integer)
var  j:integer
    encontrado: boolean
    j ← 1
    encontrado ← FALSO
    mientras (j ≤ lim) and not encontrado hacer
        si x[j] = elem entonces
            encontrado ← VERDADERO
        sino
            j ← j + 1
        fsi
    fmientras
    devolver not encontrado {es prometedor si no se ha utilizado}
ffuncion
```