Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 14

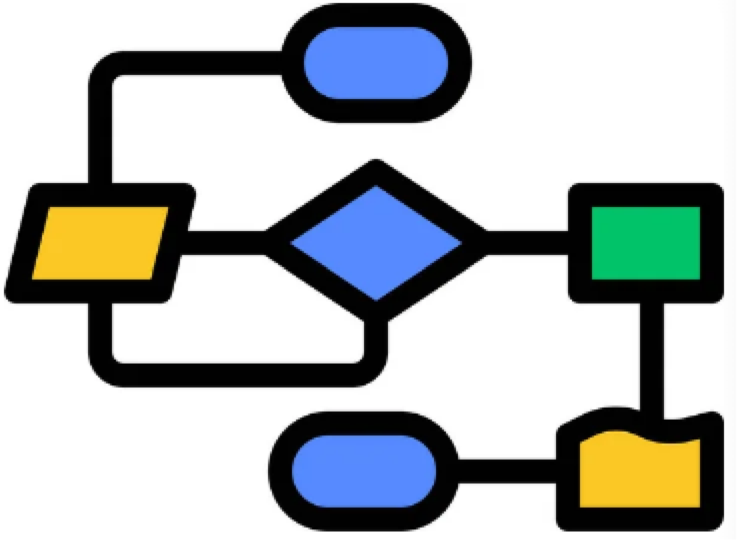
Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T4. Técnicas Algorítmicas

- Divide y Vencerás
- Algoritmos Voraces
- Backtracking

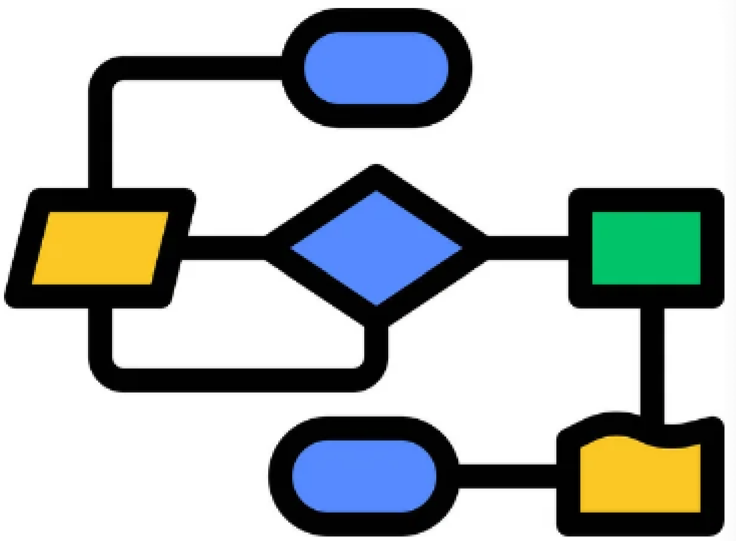


UNIVERSIDAD NEBRIJA

ÍNDICE

T4. Técnicas Algorítmicas

- Divide y Vencerás
- Algoritmos Voraces
- **Backtracking**



UNIVERSIDAD NEBRIJA

ÍNDICE

El diseño o técnica **Backtracking** o **Vuelta Atrás** proporciona una manera **sistemática** de generar **todas las posibles soluciones** siempre que dichas soluciones sean susceptibles de **resolverse en etapas**.

Realiza una búsqueda **exhaustiva** y **sistemática** en el **espacio de soluciones**, donde añade y quita los elementos para **probar todas las combinaciones posibles**.

En general, se consideran problemas donde la **solución** se construye por **etapas** y puede **representarse como una n-tupla (x1,x2,...,xn)**, donde cada **xi** es seleccionado de un conjunto **Si** que representa la decisión de la **etapa i-ésima**, del conjunto de alternativas existente.

Además, una solución tendrá que minimizar, maximizar o simplemente satisfacer una cierta **función objetivo**.

Por tanto, hay **dos tipos de restricciones**:

- **Explícitas**: indican los conjuntos **Si**.
- **Implícitas**: relaciones entre los componentes de la tupla solución para satisfacer **la función objetivo**.

El espacio de soluciones deben satisfacer las restricciones explícitas y se puede estructurar en un **árbol de exploración**.

En cada nivel se toma la decisión de la etapa correspondiente:

- **Nodo estado**: nodo correspondiente a una tupla parcial o completa que satisface las **restricciones explícitas**.
- **Nodo solución**: nodo de las tuplas completas que satisfacen las **restricciones implícitas**.

Otro elemento importante en la aplicación de esta técnica son las **funciones de poda o test de factibilidad** (se obtienen a partir de la **función objetivo**). Estas funciones permiten determinar cuándo una tupla parcial nunca llegará a ser solución. Por tanto, no se debe de mantener.

Partiendo del **árbol de exploración**, el algoritmo para resolver el problema consiste en realizar un **recorrido del árbol en profundidad**, hasta encontrar la primera solución, o recorrer el árbol completo (salvo las zonas podadas) para obtener todas las soluciones o la solución óptima.

Durante el proceso, **para cada nodo se van generando sus nodos sucesores**:

- **Nodos vivos**: los que todavía tienen hijos pendientes de generarse.
- **Nodos en expansión**: los que tienen hijos que están siendo generados.
- **Nodos muertos**: los que no pueden ser expandidos, porque no han superado el test de factibilidad o porque todos sus hijos ya han sido generados.

De las posibles formas de recorrer el árbol, destacan dos que dan lugar a dos técnicas:

- **Vuelta atrás**: **recorrido en profundidad**, de forma que los nodos vivos se gestionan mediante una pila. Método sencillo y eficiente en espacio.
- **Ramificación y poda**: corresponde a una ´búsqueda más inteligente porque siempre se expande el nodo vivo más prometedor, de forma que los nodos vivos se gestionan a través de una cola con prioridad.

Esquema general:

Cuando se hace el recorrido en profundidad y se alcanza un nodo muerto, hay que deshacer la última decisión tomada, para tomar la siguiente alternativa (igual que en un laberinto al alcanzar un callejón).

```
void vueltaAtras (vector<int>  &S, int k)  {
    preparar_recorrido_nivel (k);
    while (!ultimo_hijo_nivel(k)) {
       S.at(k) = siguiente_hijo_nivel(k);
       if (esSolucion(S, k)) {
          tratarSolucion(S);
       } else if (esCompletable(S, k)) {
                 vueltaAtras(S, k+1);
             }
     }
}
```

Backtracking

El coste temporal de un algoritmo de vuelta atrás suele depender de:

- v(n): Número nodos del espacio de búsqueda que se visitan.
- f(n): El coste de las funciones **esSolucion** y **esCompletable** en cada nodo.

Luego, el total será: ***O(v(n)f(n))***

Si la función esCompletable descarta muchos nodos, v(n) se reduce:

- Si queda un solo nodo, el coste será: ***O(nf(n))***
- Si no se descarta ninguno (peor caso), el coste será: $\boldsymbol{O(n^k f(n))}$

El problema de las N reinas

Consiste en ubicar N reinas en un tablero sin que se "amenacen" la una a la otra, es decir, que nunca se encuentren dos reinas ni en la misma fil, ni en la misma columna, ni en la misma diagonal.

Si numeramos las filas y las columnas de 1 a n y lo mismo con las reinas. Se puede asumir que la reina **i** estará en alguna posición de la fila **i**. Luego las soluciones se pueden representar por tuplas: (x1, …, xn) donde **xi** será la columna que ocupa la reina **i** en la fila **i**.

El problema de las N reinas

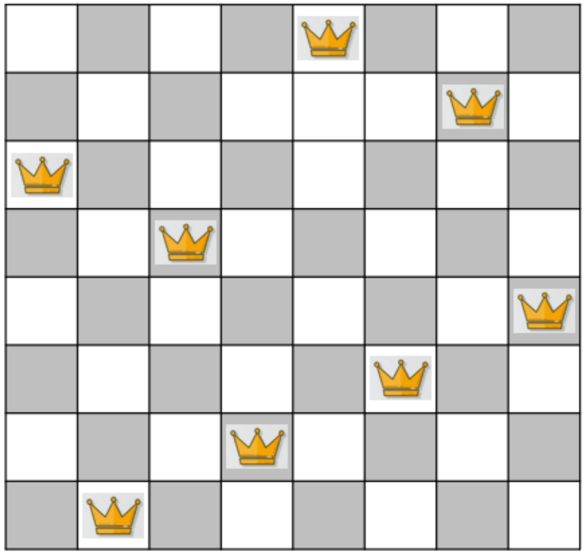
Vamos a realizar el análisis para 8 reinas en un tablero de 8x8.

La solución estará representada por:

(x1, x2, x3, x4, x5, x6, x7, x8)

Donde xi: columna donde está la reina de la fila i.

Una posible solución sería: (5, 7, 1, 3, 8, 6, 4, 2)



Las restricciones a considerar son:

- **Restricciones explícitas**: la tupla solución debe de contener los valores (1, 2, 3, 4, 5, 6, 7, 8).
- **Restricciones implícitas**:

  (1) dos reinas no pueden ocupar la misma columna, luego no pueden existir dos xi iguales en la tupla. Requiere que cada tupla sea permutación de (1,…, n).

  (2) dos reinas no pueden estar en la misma diagonal. Así, si las coordenadas de dos reinas en el tablero son (x, y) y (x’, y’), entonces, están en la misma diagonal si y solo si:

$$|x - x'| = |y - y'|.$$

Por tanto, en cada etapa k se irán generando las k-tuplas que sean factibles (con posibilidad de solución).

Respecto a la segunda restricción hay que tener en cuenta que las posiciones sobre una **misma diagonal descendiente** se cumple que tienen el mismo valor fila - columna. Mientras que posiciones sobre una **misma diagonal ascendiente** se cumple que tienen el mismo valor fila + columna.

$$x - y = x' - y' \Rightarrow x - x' = y - y'$$

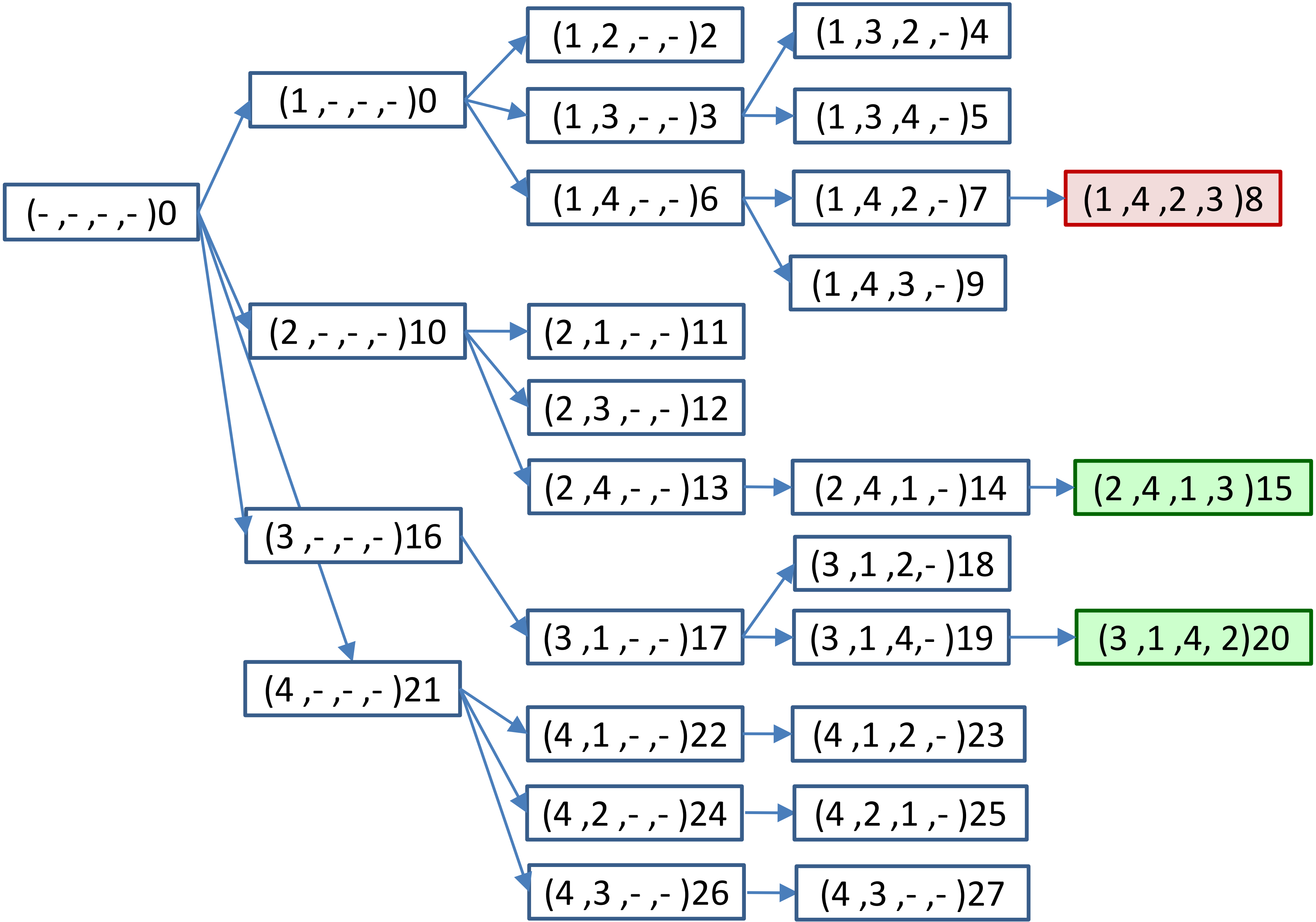
$$x + y = x' + y' \Rightarrow x - x' = y' - y$$

De ahí se obtiene que están en la misma diagonal si y solo si:

$$|j - l| = |i - k|$$

EJEMPLO 1: Vuelta Atrás o Backtracking

Problema de las N reinas



```
// Funcion que genera las posibles soluciones del problema.
bool crearSolucionNREINAS(vector <vector <int>> & tablero, int col) {
    // Caso base, cuando ya se han recorrido todas las columna del tablero
    if (col == N) {
        mostrarSolucion(tablero);
        return true;
    }
    bool res = false;
    for (int i = 0; i < N; ++i) {
        if (esPosicionFactible(tablero, i, col)) {
            tablero[i][col] = 1;
            res = crearSolucionNREINAS(tablero, col + 1) || res;
            tablero[i][col] = 0;
        }
    }
    return res;
}
```