PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMOS AVANZADOS
Hoja de Enunciados
Backtracking

Compilado por Andrés Melgar

---

1. Problema: *Knight's Tour* (Recorrido del Caballo)

Contexto

En el ajedrez, el **caballo** es una pieza que se mueve en forma de "L": dos casillas en una dirección (horizontal o vertical) y luego una casilla en dirección perpendicular. Esto da lugar a exactamente ocho movimientos posibles desde cualquier posición intermedia, siempre que el movimiento no lo lleve fuera del tablero.

Un **recorrido del caballo** es una secuencia de movimientos válidos del caballo que le permite visitar cada una de las casillas del tablero exactamente una vez. Este problema es uno de los clásicos de la teoría de algoritmos y estructuras de datos, y es un caso típico donde se puede aplicar ***backtracking*** para encontrar una solución.

Enunciado

Dado un tablero de ajedrez de dimensiones $N \times N$, se requiere implementar un algoritmo que encuentre un recorrido completo del caballo, es decir, una secuencia de movimientos válidos en la que el caballo visite cada una de las casillas exactamente una vez.

El caballo puede empezar en cualquier casilla del tablero. No es necesario que el recorrido sea cerrado, es decir, no es obligatorio que el caballo regrese a la casilla inicial después de completar el recorrido.

Se pide desarrollar una solución basada en ***backtracking*** que intente todas las posibilidades y retroceda cuando llegue a una situación sin salida.

Objetivo

El objetivo es construir un programa en **C++** que:

- Genere y explore, usando recursividad y *backtracking*, todos los movimientos válidos del caballo.
- Almacene la secuencia de pasos en una matriz $N \times N$ que indique el orden en que el caballo visita cada casilla.
- Detenga la ejecución cuando se haya encontrado una solución válida.

Ejemplo de salida esperada (tablero de 8x8)

```
01 60 39 34 31 18 09 64
38 35 32 61 10 63 30 17
59 02 37 40 33 28 19 08
36 49 42 27 62 11 16 29
43 58 03 50 41 24 07 20
48 51 46 55 26 21 12 15
57 44 53 04 23 14 25 06
52 47 56 45 54 05 22 13
```

2. Problema: Sudoku (Adaptado de lab2 2013-1 Algoritmia)

El objetivo del juego *Sudoku* es asignar dígitos a las celdas vacías de un tablero de $9x9$ (81 celdas), dividido en subtableros de $3x3$ (ver siguiente figura), de forma tal que cada columna, fila y cada subtablero contengan exactamente una instancia de dígitos del 1 al 9. Las celdas iniciales se asignan para restringir el juego de tal manera que sólo exista una manera de terminarlo.

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

Dada una matriz de 9x9 que representa un tablero del juego *Sudoku* con las celdas iniciales ya asignadas, se le pide a Ud. que elabore un programa en c++ que usando la estrategia de ***backtracking*** encuentre la solución del juego. A continuación se presenta una posible solución al tablero del ejemplo anterior.

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

3. Problema: Las N reinas

Contexto

El problema de las **N reinas** es un problema clásico de la informática y las matemáticas. Su origen se remonta al siglo XIX, y plantea el reto de colocar N reinas en un tablero de ajedrez de $N \times N$ de manera que **ninguna reina ataque a otra**.

En ajedrez, una reina puede moverse (y por tanto atacar) en cualquier dirección horizontal, vertical o diagonal. Por lo tanto, dos reinas se atacan si están en la misma fila, columna o diagonal. El objetivo es encontrar todas las posibles formas de ubicar las reinas cumpliendo esta condición.

Este problema es un ejemplo típico donde se puede aplicar ***backtracking***, ya que implica explorar múltiples configuraciones y descartar aquellas que violan las restricciones del problema.

Enunciado

Se desea encontrar una o más soluciones al problema de las N reinas, que consiste en colocar N reinas en un tablero de ajedrez de $N \times N$ de modo que no haya dos reinas que se amenacen entre sí.

Se pide implementar una función recursiva que explore todas las configuraciones posibles, retrocediendo (*backtracking*) cuando una colocación no conduce a una solución válida.

Cada solución puede representarse como un vector de N enteros, donde el valor en la posición i representa la columna en la que se colocó la reina en la fila i.

Objetivo

El objetivo del ejercicio es desarrollar una solución en **C++** que:

- Utilice una función recursiva para construir posibles soluciones fila por fila.
- Verifique, en cada paso, que la reina colocada no ataque a ninguna reina colocada previamente.
- Genere e imprima todas las soluciones válidas para un valor dado de N.

Ejemplo de salida (N = 4)

Para $N = 4$, existen dos soluciones válidas. Una de ellas puede representarse así:

```
. Q . .
. . . Q
Q . . .
. . Q .
```

Otra solución es:

```
. . Q .
Q . . .
. . . Q
. Q . .
```

Cada punto representa una celda vacía, y la letra `Q` indica la posición de una reina.

Notas adicionales

Este problema tiene una complejidad combinatoria exponencial. La implementación por ***backtracking*** permite explorar el espacio de soluciones de manera eficiente descartando configuraciones inválidas de manera anticipada.

El problema puede extenderse a variantes como:

- Contar el número total de soluciones para un valor dado de N.
- Imprimir solo una solución.
- Visualizar las soluciones gráficamente o en consola.

4. Problema: *Word Search* (Búsqueda de palabras en una matriz)

Contexto

En muchos juegos de tipo crucigrama, sopa de letras o motores de búsqueda en documentos escaneados, es común tener que localizar si una palabra específica está contenida dentro de una grilla o matriz de caracteres. Este tipo de problemas es un caso práctico de búsqueda con restricciones y es ampliamente utilizado para ejercitar técnicas de ***backtracking***.

Enunciado

Se le da una matriz de caracteres de tamaño $m \times n$ representada como un vector de vectores de tipo `char`, y una palabra dada representada como un string. Debe determinar si la palabra puede ser formada a partir de una secuencia de letras adyacentes en la grilla.

Dos letras son consideradas adyacentes si están horizontal o verticalmente contiguas. **No se permite reutilizar una misma celda más de una vez durante la formación de una palabra**.

- La matriz de entrada es de tamaño $m \times n$ con letras mayúsculas del alfabeto inglés.
- La palabra a buscar tiene longitud k, $1 \leq k \leq m \times n$.
- Debe explorar todas las posibilidades empezando desde cualquier celda que contenga la primera letra de la palabra.

Objetivo

Implementa un programa en C++ que que determine si la palabra existe en la matriz usando para ello ***backtracking*** . Si la palabra puede ser construida, retorna `true`; de lo contrario, retorna `false`.

Ejemplo

```
Matriz:
{
  {'A','B','C','E'},
  {'S','F','C','S'},
  {'A','D','E','E'}
}

Palabra: "ABCCED" => true
Palabra: "SEE"    => true
Palabra: "ABCB"   => false
```

Restricciones y Consideraciones

- Se puede mover hacia arriba, abajo, izquierda o derecha.
- No se permite usar una misma celda dos veces en una misma búsqueda.
- El algoritmo debe ser eficiente y utilizar técnicas de *backtracking* con desmarcado (*backtrack*) adecuado para evitar reutilización de celdas.
