Algorítmica I
Grado en Ingeniería Informática
EPD - Prueba evaluable # 1

Enunciado: El Mapa del Merodeador: búsqueda recursiva en Hogwarts

El Mapa del Merodeador muestra los pasillos de Hogwarts y la posición de las puertas mágicas. Harry necesita encontrar el número de caminos posibles para llegar desde la Entrada Principal hasta la Sala de los Menesteres, moviéndose solo hacia abajo o a la derecha en una cuadrícula mágica.
Cada casilla del mapa puede estar:

- Libre (1) → puede moverse.
- Bloqueada (0) → hay una pared encantada.

El programa debe usar recursividad simple para:

1. Calcular el número total de caminos posibles desde (0,0) hasta (N-1, N-1).
2. Calcular el camino de menor coste si cada celda tiene un valor de energía.
3. Analizar el tiempo de ejecución de la versión recursiva y de una versión iterativa optimizada.

Objetivos de la práctica

- Aplicar recursividad (solo llamadas hacia abajo y derecha).
- Implementar versiones recursiva pura e iterativa (programación dinámica).
- Realizar un análisis temporal empírico (comparando tiempos y llamadas).
- Comprender la diferencia entre crecimiento exponencial vs. polinómico.
- Trabajar en Java sin objetos, solo con métodos estáticos y arrays.

Representación del mapa

El mapa de Hogwarts es una matriz cuadrada N×N de enteros:

```
int[][] mapa = {
  {1, 1, 1, 1},
  {1, 0, 1, 1},
  {1, 1, 1, 0},
  {1, 1, 1, 1}
};
```

- 1 → pasillo transitable
- 0 → obstáculo mágico

Harry empieza en (0,0) y debe llegar a (N-1, N-1).

Funciones a implementar

Función 1 — Mostrar el mapa.
void mostrarMapa(int[][] mapa)

- Imprime la matriz con formato legible

Función 2 — Número de caminos posibles (recursiva).
int contarCaminosRec(int[][] mapa, int i, int j)

- Devuelve el número total de caminos válidos desde (i, j) hasta la meta, moviéndose solo hacia abajo o derecha

Reglas:

- Si (i, j) está fuera del mapa → devolver 0
- Si (i, j) es una pared (0) → devolver 0
- Si (i, j) es la meta → devolver 1
- En otro caso: Habrá que contar los caminos recursivos

Función 3 — Número de caminos (iterativa optimizada)
int contarCaminosIter(int[][] mapa)

- Utiliza programación dinámica con una matriz dp de tamaño N×N:

dp[i][j] = dp[i-1][j] + dp[i][j-1]
siempre que mapa[i][j] == 1.

Función 4 — Camino de menor coste (recursiva)
En otra versión del mapa, cada casilla tiene un coste positivo:

```
int[][] energia = {
  {1, 3, 1},
  {1, 5, 1},
  {4, 2, 1}
};
```

Implementa:
int caminoMinimoRec(int[][] energia, int i, int j)

- Devuelve el coste mínimo total desde (i,j) hasta la meta:

```
return energia[i][j] + Math.min(
  caminoMinimoRec(energia, i+1, j),
  caminoMinimoRec(energia, i, j+1)
);
```

Condiciones base:

- Si sale del mapa → devuelve Integer.MAX_VALUE
- Si llega a la meta → devuelve energia[i][j]

Función 5 — Medición de tiempos
long medirTiempo(Runnable f)

- Ejecuta una función y devuelve el tiempo en nanosegundos:

```
long inicio = System.nanoTime();
f.run();
long fin = System.nanoTime();
return fin - inicio;
```

Función 6 — Main
public static void main(String[] args)
Debe:

1. Leer un mapa desde un CSV (o definirlo manualmente).
2. Ejecutar:
   - contarCaminosRec(mapa, 0, 0)
   - contarCaminosIter(mapa)
   - caminoMinimoRec(energia, 0, 0)
3. Medir el tiempo de ejecución de cada método.
4. Mostrar resultados:

--- EL MAPA DEL MERODEADOR ---
Tamaño del mapa: 6x6
Caminos posibles (recursivo): 133
Tiempo: 18.230 ms
Caminos posibles (iterativo): 133
Tiempo: 0.421 ms
Coste mínimo del recorrido mágico: 23

Ejemplo de mapa (5x5)
1,1,1,1,1
1,0,1,1,1
1,1,1,0,1
1,1,1,1,1
1,1,1,1,1

Resultado esperado:
- Caminos posibles: 84
- Coste mínimo (si se usa matriz de energía): depende del input.

Análisis temporal requerido

El alumno deberá incluir en el informe:

Método	Complejidad teórica	Tiempo empírico (ms)	Llamadas recursivas
contarCaminosRec	—	—	
contarCaminosIter	—	—	
caminoMinimoRec	—	—	

Además:
- Representar gráficamente el tiempo vs. tamaño del mapa (por ejemplo, para N = 4, 6, 8, 10).
- Discutir cómo crece el tiempo al duplicar N.
- Justificar por qué la recursión simple se vuelve inviable para valores grandes.

Evaluación (50% código, 50% memoria)

Criterio	Peso
Código	
Implementación recursiva correcta	20%
Implementación iterativa correcta	20 %
Claridad y estilo del código	10%

Criterio	Peso
Código	
Memoria	
Redacción clara y comprensión del problema	10%
Medición y comparación de tiempos	20%
Análisis de complejidad bien razonado	20%

"Los caminos de Hogwarts son cambiantes y traicioneros...
pero un buen mago de la recursión siempre encuentra la salida."
— Profesor Flitwick

Entrega

- Trabajo en parejas.
- La inscripción de los grupos se realizará mediante el Campus Virtual.
- La entrega se realizará a través del Campus en la Actividad: EPD-Prueba evaluable # 1, dónde cada grupo deberá subir un archivo .zip que contenga:
  - Memoria.pdf → informe con análisis, gráficos y conclusiones.
  - ProyectoJava.zip → código fuente completo del proyecto.

- Nomenclatura de los archivos
  Bloque1_GrupoX_Nombre1_Nombre2.zip
  Ejemplo:
  Bloque1_Grupo3_HermioneGranger_HarryPotter.zip

Recomendaciones

- Comprobar que el código compila correctamente antes de comprimir.
- Comentar las funciones principales.
- Incluir en la memoria:
  - Descripción de cada versión del algoritmo.
  - Tabla de tiempos y número de llamadas.
  - Gráfico comparativo.
  - Conclusiones sobre eficiencia.

"El Mapa del Merodeador nunca miente...
pero los tiempos de ejecución sí pueden sorprenderte."
— Fred & George Weasley