ALGORITMICA I:
EPD EVALUABLE #2

Marcos Velázquez Carmona

Daniel Sanchez-Matamoros Carmona

ÍNDICE

Enunciado: Cartografía de la Tierra Media 1
Representación del Mapa Energético 1
Objetivos de la práctica 2
Problemas a resolver 2
Problema 1: La Marcha Silenciosa 2
Enunciado 2
Elección Algorítmica 3
Alternativas descartadas y motivos 3
Análisis de la complejidad teórica 3
Comparación de tiempos de ejecución reales (tablas y gráficos) 3
Reflexión sobre escalabilidad y eficiencia 4
Propuestas de mejora para futuros cartógrafos reales de la Tierra Media 5
Problema 2: El ojo de Cirith Ungol 5
Enunciado 5
Elección algorítmica 5
Alternativas descartadas y motivos 6
Análisis de la complejidad teórica 6
Comparación de tiempos de ejecución reales (tablas y gráficos) 7
Reflexión sobre escalabilidad y eficiencia 7
Propuestas de mejora para futuros cartógrafos reales de la Tierra Media 8
Problema 3: Los Senderos del Druadan 8
Enunciado 8
Elección algorítmica 8
Alternativas descartadas y motivos 9
Análisis de la complejidad teórica 9
Comparación de tiempos de ejecución reales (tablas y gráficos) 9
Reflexión sobre escalabilidad y eficiencia 10
Propuestas de mejora para futuros cartógrafos reales de la Tierra Media 11
Requisitos de visualización 11
Restricciones de programación 12
Sistema de Control de Versiones 12

Enunciado: Cartografía de la Tierra Media

Enunciado para sabios estudiantes de Algorítmica en la EPS de Ingeniería Informática de la Universidad Pablo de Olavide. En los antiguos archivos de Rivendel se ha descubierto un conjunto de mapas energéticos creados por los Dúnedain. Estos mapas describen los costes de atravesar regiones de la Tierra Media, desde las tierras nevadas de Forochel hasta las llanuras devastadas de Gorgoroth. El rey Elessar ha encargado a la Hermandad de Sabios que analice estos mapas para determinar rutas, regiones y estructuras de paso que puedan utilizarse en futuras expediciones de exploración, guerra y comercio. Tú eres el cartógrafo designado. Tu misión: analizar el mapa, identificar patrones y resolver varios problemas aplicando el algoritmo más adecuado en cada caso.

Tú eres el cartógrafo designado.

Tu misión: analizar el mapa, identificar patrones y resolver varios problemas aplicando el algoritmo más adecuado en cada caso.

Representación del Mapa Energético

El mapa se proporciona como una matriz cuadrada $N\ x\ N$ leída desde un archivo *CSV*. Cada celda es un entero que indica el "costo energético" de avanzar por ese terreno:

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

Puedes imaginar, que terrenos con mayor coste representan:

- cenizas volcánicas que queman las botas,
- bosques espesos que exigen rodeos,
- ciénagas donde los pasos se hunden,
- desfiladeros traicioneros donde cada roca pesa como un juramento.

Objetivos de la práctica

El consejo necesita resolver varios problemas estratégicos basados en el mapa. Para cada uno de ellos:

1. Escoge el algoritmo más adecuado, entre:
   o *Backtracking* recursivo con poda
   o Búsqueda Voraz (*Greedy*)
   o *Divide y Vencerás*
   o Combinaciones justificadas de los anteriores

2. Justifica rigurosamente tu elección.

3. Implementa la solución conforme a las restricciones de programación.

4. Analiza la complejidad temporal (teórica y empírica) de tu algoritmo.

5. Incluye comparaciones experimentales entre alternativas.

Problemas a resolver

A continuación, se presentan las tareas que el Consejo espera que resuelvas. Cada una representa una situación distinta en el viaje por la Tierra Media.

Problema 1: La Marcha Silenciosa

Enunciado

Encontrar una ruta desde la entrada del mapa $(0,0)$ hasta la salida $(N-1, N-1)$ que minimice el coste total.

Este es el problema raíz, pero debes elegir la estrategia óptima según el tamaño y las características del mapa.

Se debe analizar si corresponde:

• *Backtracking*

• Algoritmo voraz que escoja siempre la celda más "prometedora"

• Una descomposición *Divide y Vencerás* del mapa

• Una mezcla híbrida, si es justificable

Incluye visualización *ASCII* del camino encontrado.

Elección Algorítmica

El algoritmo elegido para este ejercicio es *backtracking* para un valor de N pequeño. ($N \approx< 10$), para valores grandes de N usaremos un híbrido entre *backtracking* y *voraz*, ya que el coste del algoritmo se dispara y se hace inviable.

Alternativas descartadas y motivos

El algoritmo voraz puro sí encuentra solución, pero encontraría la primera que pille y seguramente no sería la óptima, y eso no es lo que pide el enunciado.

Divide y vencerás queda descartado porque será demasiado complejo y costoso para el cálculo de todos los caminos, y, además, es posible que ni siquiera encuentre la solución, por eso no es la opción óptima.

Análisis de la complejidad teórica

Peor de los casos:

Los algoritmos *backtracking* se caracterizan por tener una estructura de árbol. Las hojas corresponden a soluciones completas.

El algoritmo sería exponencial porque habría que explorar todos los caminos, $O(2^n)$. Para $n = 10$, habría $2^{10} = 1024$ combinaciones distintas.

La demostración de que es exponencial viene a través de la siguiente operación:

$$T(n) = \prod_{k=1}^{n} \frac{n+k}{k} = 1 + \prod_{k=1}^{n} \frac{n}{k} \geq 2^n \epsilon\, O(2^n)$$

Caso promedio:

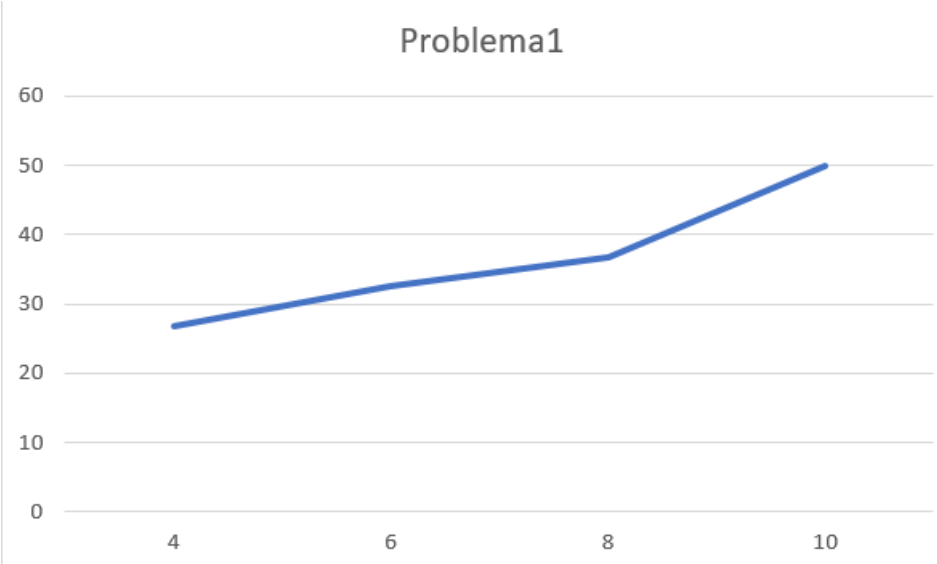
El caso promedio es igual que el peor de los casos, con la diferencia de que va a ir multiplicado n por una constante de probabilidad $p \in [0,1]$. Para el caso promedio, $(n = 0.5)$ tenemos lo siguiente: $T(\mathrm{n}) = 2^{0.5n} = (\sqrt{2})^n \in \mathrm{O}((\sqrt{2})^n)$

Mejor de los casos:

Para el mejor de los casos, la complejidad es un valor constante C, de modo que queda: $\mathrm{T(n)} \in \mathrm{O(C)}$.

Comparación de tiempos de ejecución reales (tablas y gráficos)

La siguiente gráfica muestra el estudio temporal:



Las siguientes tablas muestran el valor promedio para distintos tamaños de la matriz.

Tamaño	Problema1
4	26,73014
6	32,64126
8	36,62448
10	49,82928

Interpretando los resultados, hemos probado a cargar el programa cinco veces con la misma matriz de y anotar los tiempos, lo mismo para 4×4, 8×8, y 10×10: Lo cual corresponde con la segunda captura adjunta en el estudio temporal. Luego, hemos obtenido las medias de dichos valores y hemos generado una gráfica lineal donde en el eje de abscisas se representa el tamaño y en el eje de ordenadas se representan los tiempos. En esta gráfica, podemos observar que crece de una manera mucho más rápida en comparación con los otros problemas.

Reflexión sobre escalabilidad y eficiencia

El algoritmo de *backtracking* presenta las siguientes características:

- Si existe una solución, la obtiene.
- Simple de implementar
- Adaptable a las características de cada problema.

Pero presenta limitaciones:

- No escala bien para valores grandes de N, debido al crecimiento combinatorio de rutas posibles. Coste exponencial en la mayoría de los casos.
- Si el espacio de búsqueda es muy grande, puede que no se encuentre la solución.
- Consume mucha memoria debido a la recursividad.

Aun así, para los tamaños de mapa típicos del ejercicio, el enfoque ofrece un equilibrio razonable entre **entendimiento, claridad y tiempo de ejecución**, siempre que se aplique una poda eficiente.

Propuestas de mejora para futuros cartógrafos reales de la Tierra Media

1. Para mapas muy grandes, el algoritmo actual podría tardar demasiado, quizás una propuesta sería combinar *backtracking* con un algoritmo voraz inicial que proporcione un coste mínimo aproximado, y usar ese valor para podar aún más rutas desde el inicio.
2. Para detectar cuellos de botella, podría ser útil guardar todas las rutas de coste mínimo en lugar de solo guardar un camino mínimo.

Problema 2: El ojo de Cirith Ungol

Enunciado

Detectar todas las celdas cuya travesía es obligatoria en todas las rutas de coste mínimo.

Se requiere identificar auténticos cuellos de botella estratégicos.

Se puede emplear:

• enumeración completa mediante *backtracking*,

• enfoque voraz,

• análisis de distancias parciales combinando *Divide y Vencerás*,

• o cualquier enfoque justificado.

Debe analizarse la complejidad del número total de rutas posibles.

Elección algorítmica

El algoritmo usado para este problema es *Divide y Vencerás* combinado con *Programación Dinámica*, ya que tiene mismo nivel de exactitud que un *Divide y Vencerás* simple, es mucho más eficiente, y escalable a mapas de tamaño realista.

Backtracking solo es útil para mapas muy pequeños donde las combinaciones no sean un problema.

Alternativas descartadas y motivos

Descartamos algoritmo voraz porque queremos todas las rutas de coste mínimo para identificar las celdas obligatorias, y el algoritmo voraz podrá encontrar una ruta, pero no todas. Por ejemplo, si existen dos caminos distintos de igual coste, el algoritmo voraz solo encontrará uno.

Descartamos divide y vencerás como algoritmo único, porque no sabes cuantas rutas mínimas pasan por cada celda dentro de la subregión. Una celda puede parecer obligatoria en una subregión, pero al combinar subregiones pueden aparecer rutas mínimas que la evitan, por eso, solo dividir y combinar distancias parciales no garantiza que detectemos todos los cuellos de botella.

Análisis de la complejidad teórica

Peor de los casos:
Nuestro algoritmo es *Divide y Vencerás* con *Programación Dinámica*, por lo que lo podemos expresar la complejidad en función de sumas.

La demostración de la complejidad viene a través de la siguiente operación:

$$T(n) = 3\sum_{i=0}^{n}\sum_{j=0}^{n} 1 = 3n^2 \epsilon\, O(n^2)$$

Caso promedio:

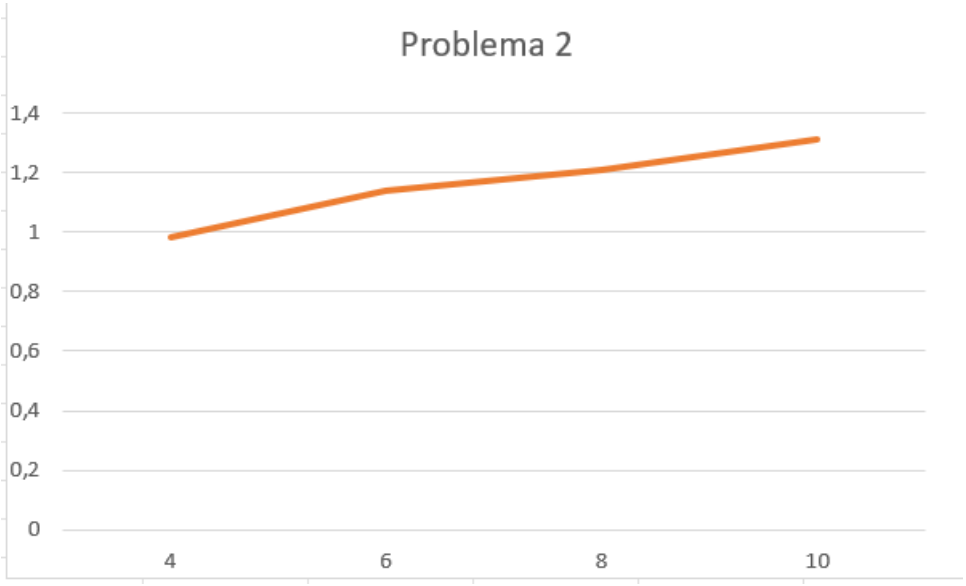
El caso promedio es igual que el peor de los casos, con la diferencia de que va a ir multiplicado n por una constante de probabilidad $\mathrm{p} \in [0,1]$. Para el caso promedio $(\mathrm{n} = 0.5)$, tenemos lo siguiente: $T(\mathrm{n}) = 3 \cdot (0.5n)^2 = 0.75n^2 \in O(n^2)$

Mejor de los casos:

Para el mejor de los casos, la complejidad es un valor constante C, de modo que queda: $T(n) \in O(C)$

Comparación de tiempos de ejecución reales (tablas y gráficos)

La siguiente gráfica muestra el estudio temporal:



Las siguientes tablas muestran el valor promedio para distintos tamaños de la matriz.

Tamaño	Problema 2
4	0,98112
6	1,13648
8	1,20792
10	1,310084

Interpretando los resultados, hemos probado a cargar el programa cinco veces con la misma matriz de y anotar los tiempos, lo mismo para 4×4, 8×8, y 10×10: Lo cual corresponde con la segunda captura adjunta en el estudio temporal. Luego, hemos obtenido las medias de dichos valores y hemos generado una gráfica lineal donde en el eje de abscisas se representa el tamaño y en el eje de ordenadas se representan los tiempos. En esta gráfica, podemos observar que cuando el algoritmo tiene valores de N, (en este caso $N_{máx} = 10$), crece más rápido el tiempo, y puede llegar a ser incontrolable para valores muy grandes de N.

Reflexión sobre escalabilidad y eficiencia

En este problema teníamos que ver qué celdas siempre se usan en todas las rutas de coste mínimo. Una forma muy sencilla de hacerlo sería probar todas las rutas posibles desde la entrada hasta la salida, pero eso se vuelve imposible rápidamente porque el número de rutas crece muy rápido a medida que el mapa es más grande. Básicamente sería exponencial y tardaría demasiado incluso con mapas medianos.

Por eso usamos un enfoque más inteligente: calculamos distancias parciales desde el inicio y desde la salida, y así podemos saber si una celda es obligatoria sin tener que recorrer todos los caminos. Es como dividir el mapa en partes más pequeñas, resolver cada parte y luego combinar la información. Esto es lo que se llama *Divide y Vencerás*, y nos ahorra muchísimo trabajo.

dividimos el problema en dos, resolvemos cada mitad, y luego hacemos un trabajo lineal para combinarlas. Según el *Teorema Maestro*, esto nos da una complejidad de $O(n \log n)$,que es bastante buena. No crece de forma exponencial, así que funciona incluso con mapas grandes.

En resumen, usar este método es mucho más rápido y escalable que intentar mirar todas las rutas. Es eficiente y nos permite resolver el problema sin que se nos vuelva imposible con mapas grandes.

Propuestas de mejora para futuros cartógrafos reales de la Tierra Media

1. A nivel de código, actualmente usamos dos matrices completas: `costedeInicio` y `costeHastaSalida`, para mapas muy grandes podría ocupar demasiada memoria y se podría evaluar el uso de una sola matriz.
2. Se podría evaluar añadir más movimientos como diagonales o movernos hacia atrás.
3. Para mejorar la visualización se podría añadir un mapa *ASCII* de las rutas como hicimos en el Problema 1.

Problema 3: Los Senderos del Druadan

Enunciado

Enumerar todos los caminos desde $(0,0)$ a $(N-1, N-1)$ cuyo coste total sea $\leq C$, siendo C una constante.

Aquí se modelan rutas secretas de los hombres del bosque.

Se debe decidir entre:

• *backtracking* con memoización,

• un enfoque voraz,

• estrategias de partición del mapa.

Elección algorítmica

La mejor opción es *backtracking* con poda y memoización, porque garantiza enumerar todas las rutas válidas con $coste \leq C$, poda ramas, evitando recorrer caminos inútiles.

Evita recomputaciones: memoización guarda resultados intermedios, reduciendo el trabajo redundante. Funciona en mapas de tamaño pequeño/medio mejor que la partición del mapa o *backtracking* puro.

Alternativas descartadas y motivos

Voraz sirve solo para encontrar un ejemplo rápido de ruta, no para enumeración completa, por eso se descarta. Además, no garantiza cumplimiento del límite C en todas las rutas, por lo que queda descartado.

En las estrategias de partición del mapa, cada subruta tiene un coste parcial, y para formar rutas completas $\leq C$, se necesita combinar todas las subrutas posibles cuyas sumas de costes respeten el límite C.

Esto significa que, si hay muchas subrutas en cada subregión, el número de combinaciones crece muy rápido.

Análisis de la complejidad teórica

Peor de los casos:

Este algoritmo se caracteriza por tener una estructura de árbol.
Las hojas corresponden a soluciones completas.

El algoritmo sería exponencial porque habría que explorar todos los caminos, $O(2^n)$. Para $n = 10$, habría $2^{10} = 1024$ combinaciones distintas.

La demostración de que es exponencial viene a través de la siguiente operación:

$$T(n) = \prod_{k=1}^{n} \frac{n+k}{k} = 1 + \prod_{k=1}^{n} \frac{n}{k} \geq 2^n \epsilon\, O(2^n)$$

Caso promedio:

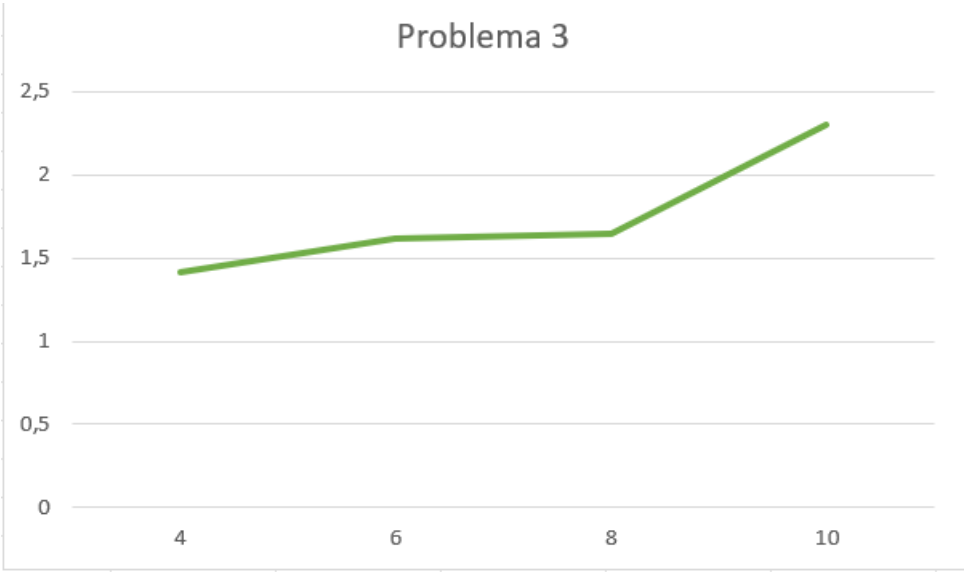
El caso promedio es igual que el peor de los casos, con la diferencia de que va a ir multiplicado n por una constante de probabilidad $\mathrm{p} \in [0,1]$. Para el caso promedio $(\mathrm{n} = 0.5)$, tenemos lo siguiente: $\mathrm{T(n)} = 2^{0.5\mathrm{n}} = \left(\sqrt{2}\right)^{\mathrm{n}} \in \mathrm{O}\left(\left(\sqrt{2}\right)^{\mathrm{n}}\right)$.

Mejor de los casos:

Para el mejor de los casos, la complejidad es un valor constante C, de modo que queda: $T(n) \in O(C)$

Comparación de tiempos de ejecución reales (tablas y gráficos)

La siguiente gráfica muestra el estudio temporal:



Las siguientes tablas muestran el valor promedio para distintos tamaños de la matriz:

Tamaño	Problema 3
4	1,4125
6	1,6163
8	1,6448
10	2,30252

Interpretando los resultados, hemos probado a cargar el programa cinco veces con la misma matriz y anotar los tiempos, lo mismo para 4×4, 8×8, y 10×10: Lo cual corresponde con la segunda captura adjunta en el estudio temporal. Luego, hemos obtenido las medias de dichos valores y hemos generado una gráfica lineal donde en el eje de abscisas se representa el tamaño y en el eje de ordenadas se representan los tiempos. En esta gráfica podemos observar que ha aumentado bastante el tiempo en $N = 10$, con respecto a los anteriores. Además es mucho más rápido que el algoritmo del Problema 1, y se acerca a los tiempos del Problema 2.

Reflexión sobre escalabilidad y eficiencia

En este problema tenemos que enumerar todas las rutas desde la entrada hasta la salida cuyo coste total sea menor o igual a C. Usar un backtracking simple sería muy ineficiente porque el número de rutas posibles crece de manera exponencial con el tamaño del mapa. Incluso mapas medianos se volverían imposibles de calcular si exploráramos todas las combinaciones.

Por eso se implementó backtracking con memoización, que permite guardar resultados intermedios y no recalcular subproblemas idénticos. Esto reduce bastante la cantidad de cálculos redundantes, aunque la complejidad sigue creciendo rápido cuando el mapa es grande o el valor de C es alto.

Además, se aplican podas tempranas: si el coste acumulado de una ruta supera C, se deja de explorar esa rama inmediatamente. Esto ayuda mucho a que el algoritmo sea más rápido y evita recorrer caminos inútiles.

En términos de escalabilidad, el algoritmo funciona bien para mapas pequeños o medianos, pero su rendimiento disminuye para mapas muy grandes o cuando C permite muchas rutas. Por eso se podría mejorar el algoritmo usando técnicas más avanzadas o simplificando la información que guardamos, como contar rutas en vez de almacenar cada una.

En resumen, con las técnicas actuales, el algoritmo es eficiente para tamaños moderados, pero su escalabilidad tiene límites debido al crecimiento exponencial de rutas posibles.

Propuestas de mejora para futuros cartógrafos reales de la Tierra Media

1. Como propusimos en las propuestas de mejora del problema 1, aquí también podríamos combinar *backtracking* con un algoritmo voraz al inicio, de tal forma que use ese valor como referencia para acotar rutas durante el *backtracking* posterior.
2. Podría ser interesante mostrar las rutas que no están dentro del umbral $coste \leq C$, para saber cuántos caminos inútiles existen, quizás sea información útil para tomar alguna decisión al respecto.

Requisitos de visualización

Para todos los problemas que involucren caminos: Se debe imprimir una versión visual del mapa marcando los pasos utilizando iconos.

Es necesario crear una leyenda adecuada. Se mostrará el coste final y, si aplica, el número de rutas encontradas.

Leyenda:

```
String[] emojis = {
    "🟧",      // 0
```

```
            "😀",       // 1
            "😅",       // 2
            "😂",       // 3
            "😍",       // 4
            "😎",       // 5
            "🤔",       // 6
            "😡",       // 7
            "😭",       // 8
            "👺"        // 9
    };
```

Restricciones de programación

El estilo de programación debe ajustarse a:

• uso de *arrays* primitivos,

• métodos *static*,

• uso de clases auxiliares (más allá de la clase contenedora) hay que indicarlo en la memoria mediante un gráfico de relaciones de métodos y clases.

Sistema de Control de Versiones

Como extra hemos utilizado Github para un control de versiones y facilitamos el enlace del repositorio público para su revisión:

https://github.com/itsmakodev/13_DanielSanchez_VelazquezMarcos

Adicionalmente, el proyecto se entrega en un .zip en la parte habilitada para ello.