Práctica 3: Búsqueda con retroceso

Myrtille Knockaert, Maryne Comblat - ERASMUS

Marzo 2025

Índice

1. Introduction

El objetivo de este proyecto es trazar una ruta que pase por todos los lugares posibles una sola vez, validando al mismo tiempo los puntos de control. La implementación en forma de "búsqueda con retroceso"permite comprobar cada posibilidad, deteniéndose cada vez si no se cumplen determinadas condiciones.

2. Tarea 1. Diseño

El primer paso consistió en definir las reglas fundamentales que deben verificarse en cada paso del recorrido:

- No se debe volver a visitar una casilla ya recorrida.
- Verificar que la casilla candidata no esté demasiado lejos del checkpoint (usando la distancia de Manhattan).
- Asegurarse de que, desde la casilla actual, sea posible alcanzar la casilla inicial (1,1) sin quedar aislado.
- Comprobar que el checkpoint se alcance exactamente en el momento previsto.

El algoritmo global para cada solución se puede describir de la siguiente manera:

```
Para cada vecino del último punto de la solución:
    Si el vecino no ha sido recorrido:
        Si el vecino es el checkpoint:
            Si se alcanza en el momento correcto y se puede completar el
            recorrido:
                Agregar el vecino a la solución.
        Sino:
            Si es posible llegar al checkpoint en el tiempo restante y completar
            el recorrido:
                Agregar el vecino a la solución.
```

3. Tarea 2. Implementación

Para la implementación, se decidió usar la biblioteca `Numpy` para manejar las matrices que representan la cuadrícula. Se representa una casilla recorrida con un 1 y una no recorrida con un 0, añadiendo además un borde de 1 para facilitar el manejo de los límites.

Para almacenar las soluciones, se construyen listas que incluyen los puntos recorridos (en orden) junto con la matriz asociada. Optamos por una representación iterativa (donde el inicio se repite varias veces) en lugar de una estructura arbórea, ya que, aunque las estructuras en forma de árbol podrían ser más eficientes en una estrategia *meet in the middle*, resultaban más complejas de gestionar para asegurar que cada punto apareciera una única vez en la solución final.

Nuestras dos funciones principales son:

- Una función que itera la búsqueda de soluciones a lo largo de todas las etapas necesarias para cubrir la cuadrícula.
- Una función que verifica, para cada vecino potencial, si cumple con las condiciones (distancia, viabilidad de completar el recorrido, etc.) para ser agregado a la solución.

Además, se han desarrollado funciones auxiliares para:

- Leer y procesar los datos de entrada del archivo.
- Construir y modificar la matriz en función del punto por el que se pasa.
- Verificar las condiciones de la distancia de Manhattan y la posibilidad de retorno a la casilla (1,1).

4. Tarea 3. Experimentación

Para evaluar la corrección y la eficiencia de nuestro algoritmo de búsqueda con retroceso, utilizamos un conjunto de casos de prueba especificados en `test.txt`. El programa principal (`retroceso.py`) lee estos casos y genera, para cada uno, el número total de recorridos válidos y el tiempo de ejecución (en milisegundos). Dichos resultados se guardan en el archivo `results.txt`.

Adicionalmente, implementamos una versión *meet in the middle* en el archivo `man_in_the_middle.py`. Esta variante también produce un fichero de salida (`results_man_in_the_middle.txt`) que incluye, para cada caso de prueba, la cantidad de recorridos encontrados y el tiempo de ejecución.

En la siguiente tabla se muestran algunos de los resultados obtenidos:

rueba	**Búsqueda retroceso**	**Meet in the middle**
1	1 recorrido, 0.000186 ms	1 recorrido, 0.187874 ms
2	2 recorridos, 0.002533 ms	2 recorridos, 2.538919 ms
3	0 recorridos, 0.004181 ms	0 recorridos, 0.449896 ms
4	24 recorridos, 0.205741 ms	24 recorridos, 209.938049 ms
5	67 recorridos, 14.654014 ms	67 recorridos, 177.428007 ms

Cuadro 1: Comparación de resultados y tiempos de ejecución.

Observamos que, en la mayoría de los casos, la versión *meet in the middle* produce la misma cantidad de recorridos (como era de esperar, pues resuelve el mismo problema), pero el tiempo de ejecución no siempre es menor. Para ciertos tamaños de cuadrícula y configuraciones de puntos de control, la estrategia *meet in the middle* puede resultar más costosa debido a la necesidad de combinar y filtrar las soluciones parciales para garantizar que no haya solapamientos en los recorridos.

5. Bola extra

La estrategia *meet in the middle* se basa en dividir los checkpoints y etapas en dos grupos y realizar dos búsquedas separadas, que posteriormente se combinan mediante una función de comparación para formar la solución final. Esta aproximación, aunque mantiene la estructura global del algoritmo, requiere una verificación adicional para asegurar que las soluciones parciales sean compatibles (cada punto debe aparecer únicamente una vez en la solución global).

6. Conclusiones

En esta práctica se ha implementado un algoritmo de búsqueda con retroceso para contar el número de recorridos válidos en una cuadrícula, cumpliendo con la restricción de visitar tres puntos de control en pasos específicos. La solución propuesta:

- Verifica la validez de cada movimiento mediante la distancia de Manhattan y la conectividad de las celdas restantes.
- Emplea estructuras de datos sencillas (listas y matrices con bordes) para facilitar la gestión de los límites y la marca de celdas visitadas.
- Ofrece resultados correctos para los casos de prueba, confirmando que el número de recorridos y los tiempos de ejecución coinciden con lo esperado en cada configuración.

Además, se desarrolló una variante *meet in the middle* para optimizar la búsqueda en teoría, separando la exploración en dos mitades que se combinan al final. Sin embargo, la experimentación muestra que, si bien ambas versiones hallan la misma cantidad de soluciones, la variante *meet in the middle* no siempre resulta más rápida, especialmente para configuraciones más grandes o complejas.

Como trabajo futuro, podrían explorarse técnicas de memoización o heurísticas de búsqueda más avanzadas para reducir el tiempo de ejecución. También sería interesante comparar distintas estrategias de *pruning* (eliminación temprana de ramas inviables) o integrar estructuras de datos más eficientes para manejar los recorridos parciales.

En definitiva, la práctica confirma la viabilidad del enfoque de backtracking para este tipo de problema, al tiempo que pone de relieve sus limitaciones de rendimiento en casos de gran tamaño.