Búsqueda Binaria: Magic Ship

(https://codeforces.com/problemset/problem/1117/C)

Carlos Luis Aguila Fajardo

1 Introducción:

1.1 Problema análogo:

Se puede observar que el problema subyacente a éste es el siguiente: Sean $(x_1, y_1), (x_2, y_2)$ dos posiciones o puntos de entrada, y defínanse

$$D = \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$$
$$D^* = D \cup \{(0, 0)\}$$

el conjunto de transformaciones que se pueden aplicar a cualquier punto, o por simplicidad, que se pueden aplicar a (x_1, y_1) para llegar a (x_2, y_2). Entonces, sean:

$$\{d_1, d_2 \dots d_m\},\ d_i \in D$$

transformaciones predefinidas de entrada, encontrar, si existe, el mínimo entero k tal que existe el conjunto $T_k = \{t_1, \dots t_k\}$ de parejas t_i tales que:

$$t_i = (d_r, x_i), \quad i \equiv r(m),\ x_i \in D^* \tag{1}$$

$$(x_1, y_1) + \sum_{i=1}^{k} d(t_i) + x(t_i) = (x_2, y_2) \tag{2}$$

Es decir, el conjunto T_k de parejas de transformaciones tal que aplicando todas las transformaciones de T_k sobre (x_1, y_1) se obtiene (x_2, y_2). Nótese que toda pareja de $t_i = (d_r, x_i)$ se compone de una transformación de entrada y una incógnita x_i, no es objetivo de este problema determinar las x_i, solo el mínimo tamaño k para dicho conjunto.

Problemas como este, con esta posibilidad limitada de movimiento por cuadrículas son usualmente referidos como problemas de geometría del taxista, o de distancia *Manhattan*.

2 Solución general

2.1 Definiciones y proposiciones

Definición: Llamamos distancia *Manhattan* entre dos puntos $(x_1, y_1), (x_2, y_2)$ a la suma de las diferencias absolutas de sus coordenadas:

$$M((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

Proposición: La distancia *Manhattan* entre dos puntos $(x_1, y_1), (x_2, y_2)$ también define la cantidad mínima de transformaciones normales $d_i \in D$ que se necesitan para llegar a (x_2, y_2) desde (x_1, y_1).

Definición: Sea $T = \{t_1, t_2, \dots\}$ el conjunto infinito de parejas $t_i = (d_r, x_i)$ que cumplen (1) y (2), denotamos T_k un prefijo tamaño k de T.

Definición: Un valor k se dice válido, aunque no necesariamente mínimo, si es posible llegar de (x_1, y_1) a (x_2, y_2) con T_k.

Definición: Un valor k es solución del problema si es válido y mínimo.

2.2 Teoremas

Lema: Sea T_k aplicado sobre un punto (x_1, y_1) para llegar a (x_2, y_2), k es válido si y solo si existe un punto (x_p, y_p) tal que es posible llegar de (x_1, y_1) a (x_p, y_p) con $\{d_1 \dots d_k\}$ y de (x_p, y_p) a (x_2, y_2) con $\{x_1 \dots x_k\}$.

Demostración: Es fácil ver que si k es válido entonces se cumple que:

$(\implies)$

$$(x_1, y_1) + \sum_{i=1}^{k} \Big(d(t_i) + x(t_i) \Big) = (x_2, y_2)$$

$$(x_1, y_1) + \sum_{i=1}^{k} d(t_i) + \sum_{i=1}^{k} x(t_i) = (x_2, y_2)$$

sea $(x_p, y_p) = (x_1, y_1) + \sum_{i=1}^{k} d(t_i)$, entonces:

$$(x_p, y_p) + \sum_{i=1}^{k} x(t_i) = (x_2, y_2)$$

$(\impliedby)$ Por otro lado, si existe (x_p, y_p) se cumple que:

$$(x_1, y_1) + \sum_{i=1}^{k} d(t_i) = (x_p, y_p)$$

$$(x_1, y_1) + \sum_{i=1}^{k} d(t_i) + \sum_{i=1}^{k} x(t_i) = (x_p, y_p) + \sum_{i=1}^{k} x(t_i)$$

$$(x_1, y_1) + \sum_{i=1}^{k} \Big((t_i) + x(t_i) \Big) = (x_2, y_2)$$

entonces k es válido.

Teorema 1: Sea $(x_p, y_p) = (x_1, y_1) + \sum_{i=1}^{k} d(t_i)$, k es válido si se cumple:

$$dM((x_p, y_p), (x_2, y_2)) \leq k$$

Demostración: Ampliando la ecuación anterior obtenemos:

$$\begin{aligned} dM((x_p, y_p), (x_2, y_2)) &\leq k \\ |x_p - x_2| + |y_p - y_2| &\leq k \\ |x_p - x_2| + |y_p - y_2| &= k - \alpha, \quad 0 \leq \alpha \leq k \end{aligned}$$

de forma que existen x_i tal que:

$$\begin{aligned} (x_p, y_p) + \sum_{i=1}^{k-\alpha} x(t_i) + \sum_{i=1}^{\alpha} (0,0) &= (x_2, y_2) \\ (x_p, y_p) + \sum_{i=1}^{k} x(t_i) &= (x_2, y_2) \\ (x_1, y_1) + \sum_{i=1}^{k} d(t_i) + \sum_{i=1}^{k} x(t_i) &= (x_2, y_2) \end{aligned}$$

entonces k es válido.

Nota: Naturalmente a la hora de implementar los algoritmos, basta con tener en cuenta que k es solución si es el primer entero válido.

Teorema 2: Si existe una solución k al problema, esta nunca supera el valor $m \cdot dM((x_1, y_1), (x_2, y_2))$

Demostración: Consideremos las transformaciones de entrada $d = \{d_1, \ldots d_m\}$: dada su naturaleza cíclica (las transformaciones se repiten para un T_k con $k > m$), en el peor de los casos existe una solución, pero por cada ciclo de m transformaciones solo se reduce la distancia entre $(x_1, y_1), (x_2, y_2)$ en 1. De tal forma que la solución sería realizar tantos ciclos como la distancia entre estos puntos, y cada ciclo está compuesto por m transformaciones. Por tanto la peor solución posible que pueda existir es $m \cdot dM((x_1, y_1), (x_2, y_2))$.

3 Algoritmos

Notemos que para encontrar una solución basta con encontrar el menor entero no negativo k que es válido. Conocemos además por el **Teorema 2** que la solución no supera nunca $m \cdot d$ donde d es la distancia *Manhattan* entre los puntos de entrada.

3.1 Búsqueda entera *naive*

Un acercamiento a solución utilizando las demostraciones anteriores sería utilizar fuerza bruta para obtener el k mínimo posible. Iterando todo k desde 0 hasta $n \cdot d$ comprobamos si se cumple:

$$dM((x_p, y_p), (x_2, y_2)) \leq k, \quad (x_p, y_p) = (x_1, y_1) + \sum_{i=1}^{k} d(t_i)$$

3.1.1 Complejidad temporal

En el peor caso, no existe un k válido, por tanto el algoritmo iteraría todos los enteros del intervalo $[0, dM((x_1, y_1), (x_2, y_2))]$. Sea $n = max\{x_1, y_1, x_2, y_2\}$, el algoritmo realizaría $O(nm)$ iteraciones.

Por cada iteración sin embargo, se calcula la suma de a lo sumo m transformaciones, por lo que la complejidad temporal final es $O(nm^2)$.

3.2 Búsqueda entera binaria

3.2.1 Observaciones

Como consecuencia del **Teorema 1**, si existe un k válido, todo $k' > k$ es también válido. Por tanto si utilizamos un predicado $P(k)$:

$$P(k) = dM((x_p, y_p), (x_2, y_2)) \leq k, \quad (x_p, y_p) = (x_1, y_1) + \sum_{i=1}^{k} d(t_i)$$

la lista hipotética de enteros de 0 a $n \cdot d$ es de la forma:

```
[False, False, ... False, True, True, ... True]
```

por lo cual se puede realizar una búsqueda binaria con predicado sobre la misma.

3.2.2 Mejoras en complejidad temporal

Considerando que ahora se recorren los enteros de forma logarítmica, y los enteros se pueden acotar a $O(nm)$, entonces la Complejidad temporal del algoritmo es $O(m \log(nm))$, donde la m fuera del logaritmo es el costo del predicado.

3.3 Mejoras utilizando sumas de prefijos

Consideremos el cálculo de la suma $\sum_{i=1}^{k} d(t_i)$, donde siempre se cumple que $k = m \cdot \left\lfloor \frac{k}{m} \right\rfloor + r,\ k \equiv r(m)$. Entonces:

$$\sum_{i=1}^{k} d(t_i) = \cdot \left\lfloor \frac{k}{m} \right\rfloor \sum_{i=1}^{m} d(t_i) + \sum_{i=1}^{r} d(t_i)$$

de forma tal que cualquier suma de $d(t_i)$ se puede expresar en sumas de ciclos de m transformaciones mas un resto, es decir, un ciclo no completado.

Sea S la lista de sumas de prefijos, precalculada en $O(m)$:

$$S = \left[\sum_{i=1}^{1} d(t_i), \sum_{i=1}^{2} d(t_i), \cdots \sum_{i=1}^{m} d(t_i), \right]$$

entonces hallar (x_p, y_p) en las búsquedas naive y binaria se convertiría en una operación $O(1)$, por lo que la complejidad temporal de los algoritmos se puede reducir a $O(nm)$ y $O(m + \log(nm))$ respectivamente.

3.4 Implementación y comprobación de casos

En el archivo `scripts/magic_ship.py` se encuentra la implementación de la búsqueda binaria mejorada.

Una versión mas simple, el *naive* con mejoras de sumas de prefijo, fue implementada en `scripts/tester.py` para comprobar los resultados de la búsqueda binaria sobre los casos prueba generados. Para ejecutar la generación de casos basta con ejecutar los comandos:

```
cd scripts
python tester.py n
```

donde n representa la cantidad de pruebas aleatorias a generar.