CS2023 - Aula de Ejercicios Nº 13

Brenner H. Ojeda Rios

Semestre 2024-0

Se sugiere que cada estudiante trate de resolver los ejercicios de forma **individual** y luego los discuta en grupo.

Ejercicios

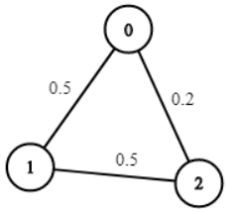
1. (5 pts) Usar Bellman Ford

   Se le proporciona un grafo ponderado no dirigido de n nodos (con índice 0), representado por una lista de aristas donde `edge[i]` = $[a, b]$ es una arista no dirigida que conecta los nodos a y b con una probabilidad de éxito al atravesarlo de `succProb[i]`.

   Dado dos nodos `start` y `end`, encuentre el camino con la máxima probabilidad de éxito para ir de `start` a `end` y devuelva su probabilidad de éxito.

   Si no hay una ruta de `start` a `end`, devuelve 0. Su respuesta será aceptada si difiere de la respuesta correcta en como máximo 1e-5. Hay como máximo un arista entre cada dos nodos.

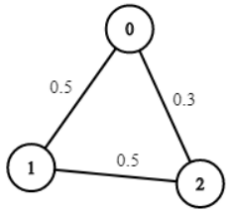
   - **Ejemplo 1:**


   ```
   Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2
   Output: 0.25000
   ```

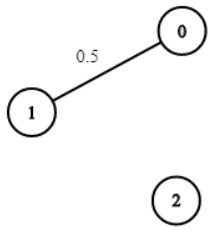
   Explicación: Hay dos caminos de principio a fin, uno tiene una probabilidad de éxito = 0,2 y el otro tiene 0,5 * 0,5 = 0,25.

   - **Ejemplo 2:**


   ```
   Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.3], start = 0, end = 2
   Output: 0.30000
   ```

- **Ejemplo 3:**




```
Input: n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2
Output: 0.00000
Explicación:
```
No hay un camino de 0 a 2.

Restricciones:

- $1 \leq n \leq 10^4$
- $0 \leq$ `start, end` $< n$
- `start` $\neq$ `end`
- $0 \leq$ `a, b` $< n$
- $0 \leq$ `len(succProb)` $=$ `len(edges)` $\leq 2 \times 10^4$
- $0 \leq$ `succProb[i]` ≤ 1

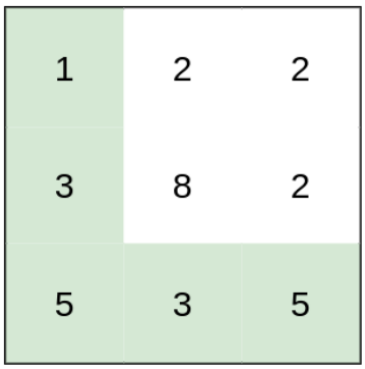
2. (6 pts) Usar Dijkstra

   Eres un excursionista que se prepara para una próxima caminata. Se le dan una matriz de alturas, de tamaño `rows x columns`, donde `alturas[fil][col]` representa la altura de la celda (fila, columna). Está situado en la celda superior izquierda (0, 0) y espera viajar a la celda inferior derecha (filas-1, columnas-1) (es decir, indexado en 0). Puede moverte hacia arriba, abajo, izquierda o derecha y deseas encontrar una ruta que requiera el mínimo esfuerzo.

   El esfuerzo de una ruta es la máxima diferencia absoluta de cotas entre dos celdas consecutivas de la ruta.

   Devuelve el esfuerzo mínimo necesario para viajar desde la celda superior izquierda a la celda inferior derecha.

   - **Ejemplo 1:**

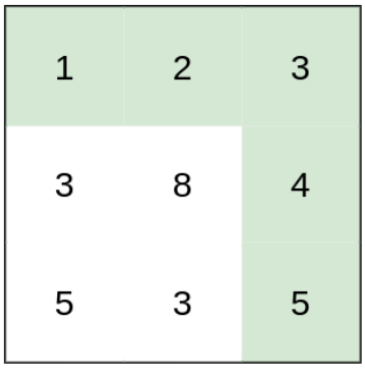


   `Input: heights = [[1,2,2],[3,8,2],[5,3,5]]`

   `Output: 2`

   `Explicación:` La ruta de [1,3,5,3,5] tiene una diferencia absoluta máxima de 2 en celdas consecutivas. Esto es mejor que la ruta de [1,2,2,2,5], donde la diferencia absoluta máxima es 3.

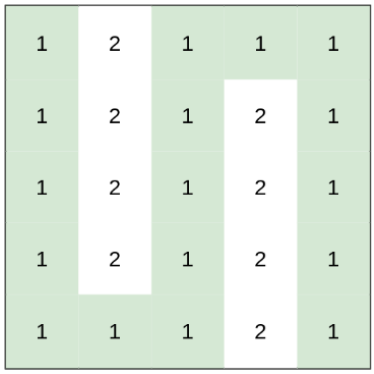
   - **Ejemplo 2:**



   `Input: heights = [[1,2,3],[3,8,4],[5,3,5]]`

   `Output: 1` (La ruta de [1,2,3,4,5] tiene una diferencia absoluta máxima de 1 en celdas consecutivas, que es mejor que la ruta [1,3,5,3,5].)

- **Ejemplo 3:**



`Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]`
`Output: 0` (Esta ruta no requiere ningún esfuerzo.)

Restricciones:

- `rows` = `heights.length`
- `columns` = `heights[i].length`
- $1 \leq$ `rows, columns` ≤ 100
- $1 \leq$ `heights[i][j]` $\leq 10^6$

3. (9 pts) Usar Floyd-Warshall

Estás organizando un juego extraño para un ejercicio de formación de equipos. En este juego hay ciertos lugares en los que las personas pueden pararse, y desde cada lugar hay caminos que conducen a otros lugares, pero no necesariamente hay caminos que conducen directamente de regreso. Ya tienes todo configurado, pero necesitas saber dos números importantes. Es posible que haya algunas ubicaciones desde las que se pueda llegar a cualquier otra ubicación. También puede haber ubicaciones a las que se pueda llegar desde cualquier otra ubicación. Necesitas saber cuántos de cada una de estas posiciones hay.

Cree una clase TeamBuilder con un método specialLocations que toma rutas que describen la forma en que se han conectado las ubicaciones y devuelve exactamente dos elementos, el primero es el número de ubicaciones que pueden llegar a todas las demás ubicaciones y el segundo es el número de ubicaciones a las que pueden acceder todas las demás ubicaciones. Cada elemento de las rutas contendrá tantos caracteres como elementos haya en las rutas. El i-ésimo elemento de las rutas (indexado en 0) contendrá un '1' en la posición j si hay una ruta que conduce directamente de i a j, y un '0' si no hay un camino que conduzca directamente de i a j.

- **Ejemplo 1:**
  `Input: edges = ["010","000","110"]`
  `Output: [ 1, 1 ]`
  `Explicación:` Las ubicaciones 0 y 2 pueden llegar a la ubicación 1 y la ubicación 2 puede llegar a las otras dos ubicaciones, por lo que devolvemos 1,1.
- **Ejemplo 2:**
  `Input: ["0010","1000","1100","1000"]`
  `Output: [1, 3]`
  `Explicación:` Solo la ubicación 3 puede llegar a todas las demás ubicaciones, pero debe tomar más de un camino para llegar a las ubicaciones 1 y 2. Todas las demás ubicaciones pueden llegar a las ubicaciones 0, 1 y 2. El método devuelve [1,3].
- **Ejemplo 3:**
  `Input: ["01000","00100","00010","00001","10000"]`
  `Output: [5, 5]`
- **Ejemplo 4:**
  `Input: ["0110000","1000100","0000001","0010000","0110000","1000010","0001000"]`
  `Output: [1, 3]`

Restricciones:

- `rutas` contendrán entre 2 y 50 elementos, inclusive.
- Cada elemento de `rutas` contendrá N caracteres, donde N es el número de elementos de las rutas.
- Cada elemento de `rutas` contendrá sólo los caracteres '0' y '1'.
- El i-ésimo elemento de `rutas` contendrá un cero en la i-ésima posición.