Téli kerékpárverseny feladatmegoldás

Függvény: max_teli_kerekparverseny

```
C: > Zsuzsa > SZTE digit > algoritmusok > github > telikerekparverseny.py > ...
  1    def max_teli_kerekparverseny(N, M, K, magassagok):
```

Bemeneti paraméterek: N: A négyzetrács sorainak száma (magasság); M: A négyzetrács oszlopainak száma (szélesség); K: Az engedélyezett maximális szintkülönbség két egymást követő lépés között; magassagok: Egy N × M méretű lista, amely minden kereszteződés magasságát tartalmazza.

Lépések a függvényen belül: lehetséges irányok meghatározása:

```
  2        iranyok = [(0, 1, 'J'), (1, 0, 'L')]  # Jobbra és lefelé mozgás
```

Itt definiáljuk, hogy milyen irányokban mozoghatunk: (0, 1, 'J'): Ez jobbra lépést jelent (J), azaz az aktuális oszlopból a következő oszlopba; (1, 0, 'L'): Ez lefelé lépést jelent (L), azaz az aktuális sorból a következő sorba. DP mátrix és szülő mátrix inicializálása:

```
  3        dp = [[0] * M for _ in range(N)]  # Maximális hossz tárolása
  4        szulo = [[None] * M for _ in range(N)]  # Útvonal visszakövetése
  5
```

dp mátrix: Minden cellában azt tároljuk, hogy az adott cellába lépve mennyi a maximális lépések száma (azaz az adott cellából indulva hány kereszteződés érhető el). szulo mátrix: Ez segít visszakövetni, hogy az adott cellába honnan érkeztünk, és milyen lépéssel (jobbra vagy lefelé). Dinamikus programozás a táblázat feltöltéséhez:

```
  6        # Dinamikus programozás a táblázat feltöltéséhez
  7        for i in range(N):
  8            for j in range(M):
  9                for di, dj, lep in iranyok:
 10                    ni, nj = i - di, j - dj
 11                    if 0 <= ni < N and 0 <= nj < M and abs(magassagok[i][j] - magassagok[ni][nj]) <= K:
 12                        if dp[ni][nj] + 1 > dp[i][j]:
 13                            dp[i][j] = dp[ni][nj] + 1
 14                            szulo[i][j] = (ni, nj, lep)
```

Ez a rész végigmegy minden cellán, és megvizsgálja, hogy az adott cellába lépve milyen hosszú út érhető el: Külső két ciklus (for i, for j): soronként és oszloponként végigmegyünk a mátrixon. Belső ciklus (for di, dj, lep): az

aktuális cella minden lehetséges "szülőjét" ellenőrizzük, ahonnan az adott cellába érkezhettünk (jobbra vagy lefelé lépés). Határok ellenőrzése: csak olyan cellát vizsgálunk, amely a rácson belül van (0 <= ni < N és 0 <= nj < M). Szintkülönbség ellenőrzése: csak akkor léphetünk a cellába, ha a szintkülönbség a két kereszteződés között nem haladja meg a K értéket. Maximális hossz frissítése: ha a szülő cellából érkezve hosszabb út érhető el, frissítjük a dp mátrixot és a szulo mátrixot. Maximum hosszú út keresése:

```
16      # Maximum hossz keresése
17      max_hossz = 0
18      vegpont = None
19      for i in range(N):
20          for j in range(M):
21              if dp[i][j] > max_hossz:
22                  max_hossz = dp[i][j]
23                  vegpont = (i, j)
```

Ebben a lépésben meghatározzuk, hogy melyik cellából indul a leghosszabb lehetséges út: max_hossz: a maximális lépések száma, amely egy cellából indulva érhető el; vegpont: az a cella (sor, oszlop), ahonnan a leghosszabb út elérhető. Útvonal visszakövetése:

```
25  # Útvonal visszakövetése
26  utvonal = []
27  jelenlegi = vegpont
28  while jelenlegi:
29      ni, nj, lep = szulo[jelenlegi[0]][jelenlegi[1]] if szulo[jelenlegi[0]][jelenlegi[1]] else (None, None, None)
30      if lep:
31          utvonal.append(lep)
32      jelenlegi = (ni, nj) if ni is not None and nj is not None else None
```

Ez a rész visszaköveti a maximális hosszú út lépéseit: a szulo mátrix segítségével megtaláljuk, hogy az adott cellába honnan érkeztünk; a lépéseket (J vagy L) hozzáadjuk az útvonalhoz; visszalépünk a szülő cellába, amíg el nem érjük az útvonal kezdőpontját. Út kezdőpontjának meghatározása:

```
34      # Kiírás
35      kezdo_pont = vegpont
36      for lep in utvonal[::-1]:
37          if lep == 'L':
38              kezdo_pont = (kezdo_pont[0] - 1, kezdo_pont[1])
39          elif lep == 'J':
40              kezdo_pont = (kezdo_pont[0], kezdo_pont[1] - 1)
```

Miután visszakövettük az útvonalat, az eredeti vegpont alapján kiszámítjuk az útvonal kezdőpontját. Eredmények visszaadása:

```
42      return max_hossz, (kezdo_pont[0] + 1, kezdo_pont[1] + 1), ''.join(utvonal[::-1])
```

Az eredmények: max_hossz: a maximális út hossza (az érintett kereszteződések száma); kezdo_pont: az útvonal kezdőpontjának sor- és oszlopindexe (1-alapú indexelésben); utvonal: az útvonalat leíró karakterlánc, amely a lépések sorrendjét tartalmazza.

Főprogram: bemenet beolvasása:

```
45  # Bemenet beolvasása
46  N, M, K = map(int, input().split())
47  magassagok = [list(map(int, input().split())) for _ in range(N)]
```

Beolvassuk a mátrix méreteit és a kereszteződések magasságait. Megoldás meghívása:

```
49  # Megoldás meghívása
50  max_hossz, kezdo_pont, utvonal = max_teli_kerekparverseny(N, M, K, magassagok)
```

Meghívjuk a függvényt, és eltároljuk az eredményeket. Eredmények kiírása:

```
52  # Kimenet
53  print(max_hossz)
54  print(kezdo_pont[0], kezdo_pont[1])
55  print(utvonal)
56
```

Az eredményeket a megadott formátumban írjuk ki: Maximális út hossza; Kezdőpont (sor, oszlop); Útvonal (lépések).