DINAMIKUS PROGRAMOZÁS

MINIMÁLIS KÖLTSÉGŰ ÚT

Adott egy **m**x**n**-es költségmátrix (`koltseg[][]`) és benne egy (`m, n`) pozíció. **Adjuk meg az adott pozíció eléréshez szükséges minimális „költségút" költségét!**

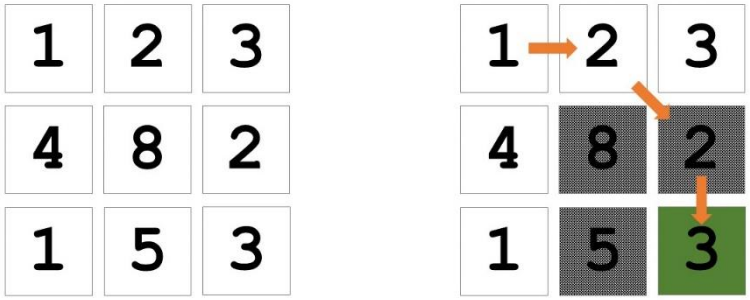
Indulás a (0, 0) pozícióról. A mátrix minden egyes eleme az adott elemen való áthaladás költségét jelenti. Az **elérési útvonal teljes költsége** az adott útvonalon felmerülő **összes költség** összege, beleértve a **forrást** és a **célt** is.

A mozgás szabályai:

Egy adott helyről *csak lefelé*, *jobbra* és **átlósan jobbra lefelé** lehet lépni, vagyis egy (`i`, `j`) indexű cellából az (`i+1`, `j`), (`i`, `j+1`) és (`i+1`, `j+1`) indexű cellákba léphetünk.

Egy egyszerű példa:

Mi a (2, 2) indexű elem minimális költségútja?



A minimális költségű utat a fenti ábra mutatja. Az elérési út: (0, 0) ➡ (0, 1) ➡ (1, 2) ➡ (2, 2).

Az út ára így **8** (1 + 2 + 2 + 3).

Az (`m`, `n`) elérési útnak a sötét kitöltésű három cella valamelyikén biztosan keresztül kell haladnia: (`m-1`, `n-1`) vagy (`m-1`, `n`) vagy (`m`, `n-1`). Tehát az (`m`, `n`) cella minimális elérési költsége: minimum költség a három cellából és még a cél költsége (`koltseg[m][n]`).

`mK(m, n)` : az `(m, n)` indexű pozíció elérésének **minimális költsége**.

```
mK(m,n)=min(mK(m-1,n-1),mK(m-1,n),mK(m,n-1))+koltseg[m][n]
```

Tehát a **minimális költség útvonala** problémának egyszerű **rekurzív** megvalósítását adtuk meg.

(Megjegyzendő, hogy a fenti függvény újra és újra kiszámolja ugyanazokat a részproblémákat. Sok olyan csomópont van, amely többször is feldolgozásra kerül. Ennek a rekurzív megoldásnak az „időbonyolultsága" exponenciális, és rettenetesen lassú.)

Más dinamikus programozási problémákhoz hasonlóan, *ugyanazon részproblémák újraszámítása* elkerülhető egy ideiglenes (`temp[][]`) mátrix felépítésével. *Lásd a kódot!*