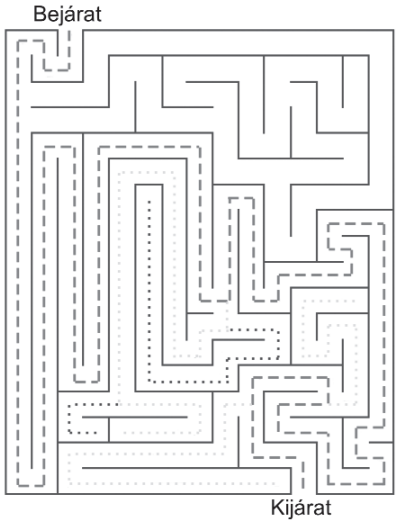
29. Visszalépéses keresés – 1.

A visszalépéses keresés algoritmusa

Az eddig megismert algoritmusok bizonyos értelemben „nyílegyenesen” haladtak előre. Tudtuk, hogy merre kell mennünk, és minden egyes lépéssel közelebb kerültünk a feladat megoldásához. Sokszor azonban nem látjuk előre az utat, csak próbálkozásokkal tudjuk elérni a célt. Tipikus példa egy labirintus felderítése. Ha zsákutcába jutottunk, akkor bizony vissza kell lépni az utolsó elágazásig, és másfelé indulni.

A **visszalépéses keresést** (backtrack) azokban a feladatokban alkalmazzuk, melyekben a megoldás során gyakran kerülünk döntési helyzetbe. Egy-egy döntés meghatározza a megoldás további folyamatát. Lehetséges azonban, hogy így megakadunk, ezért meg kell változtatnunk korábbi döntéseinket. Ehhez visszalépünk egy előző döntéshez, majd másik irányban keressük tovább a megoldást.

Visszalépéses keresés rekurzióval



Útkeresés a labirintusban. Van rövidebb út is a bejárattól a kijáratig?

Példaként tekintsük a kijárat keresését egy labirintusban. Minden egyes elágazásnál kiválasztjuk, hogy melyik irányba haladjunk tovább. Ha zsákutcába jutunk (vagy körbeértünk), akkor visszatérünk egy előző elágazáshoz, ahonnan másik irányba tudunk indulni.

Az egyszerűség kedvéért a labirintust egy *MaxSor* sorból és *MaxOszlop* oszlopból álló, négyzetrácsos táblának tekintjük, melynek bizonyos mezőit falak foglalják el. A bal felső cellából kell indulnunk, és a jobb alsó cellánál van a kijárat. Közben csak jobbra vagy lefelé léphetünk. Megoldásunk könnyen általánosítható tetszőleges labirintusra.

A táblát kétdimenziós, karakter típusú tömbben tároljuk. A falat *X*-szel, a szabad utat szóközzel jelezzük. Egyszerűbb lesz a lehetséges lépésirányok kiválasztása, ha a 0. sorba és oszlopba, illetve a *MaxSor*+1 sorba és a *MaxOszlop*+1 oszlopba szintén *X*-et írunk. A felderített úthoz tartozó cellákat pedig *O*-val jelöljük.

A táblázat egy cellájára érkezve a következő algoritmust követjük. Megpróbálunk jobbra lépni, ameddig csak lehet, majd megpróbálunk lefelé lépni, ameddig csak lehet. Ha megakadtunk, akkor visszalépünk az első olyan cellára, ahonnan még tudunk ki nem próbált irányba továbbhaladni. Az út kiírása miatt a visszalépésnél szabaddá tesszük azokat a cellákat, melyek zsákutcába vezettek.

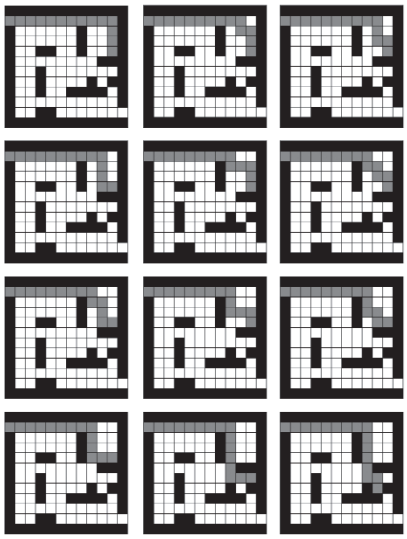
A táblát és a *MaxSor*, *MaxOszlop* változókat modulszinten deklaráljuk. Így csökkentjük az útkereső eljárás paramétereinek számát. A továbblépést az útkereső eljárás rekurzív hívásával végezzük. Ügyeljünk arra, hogy a paramétereket cím szerint adjuk át, mert az eljárás módosítja az értéküket:

```
ELJÁRÁS Útkeresés(VÁLTOZÓ Sor, Oszlop MINT Egész, VÁLTOZÓ Kész MINT Logikai)
    Tábla(Sor, Oszlop) = "O"   ' bejelöljük, hogy elértük ezt a cellát
    Kész = (Sor == MaxSor ÉS Oszlop == MaxOszlop)
    HA NEM Kész ÉS Tábla(Sor, Oszlop+1) == "  " AKKOR   ' jobbra lépünk
        Útkeresés(Sor, Oszlop+1, Kész)
    ELÁGAZÁS VÉGE
    HA NEM Kész ÉS Tábla(Sor+1, Oszlop) == "  " AKKOR   ' lefelé lépünk
        Útkeresés(Sor+1, Oszlop, Kész)
    ELÁGAZÁS VÉGE
    HA NEM Kész AKKOR   ' visszalépünk
        Tábla(Sor, Oszlop) = "  "   ' felszabadítjuk a cellát (a kiírás miatt)
        HA Tábla(Sor, Oszlop–1) == "O" AKKOR Oszlop = Oszlop – 1   ' balról jöttünk
        EGYÉBKÉNT Sor = Sor – 1   ' fentről jöttünk
        ELÁGAZÁS VÉGE
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

A rekurziót a *Sor* = 1, *Oszlop* = 1, *Útkeresés*(*Sor*, *Oszlop*, *Kész*) utasításokkal indítjuk el.

1. gyakorlat. Írjuk meg az útkeresést végző programot! Jelenítsük meg a labirintust és a kijárathoz vezető utat! Teszteléshez felhasználhatjuk a forrásfájlok között található *Útkeresés1-4.txt* állományokat.

Gondoljuk át alaposan az algoritmust! Mi történik egy rekurzív hívásból való visszatérés után? Honnan tudjuk a végén, hogy eljutottunk-e a kijárathoz? Módosul-e a *Kész* változó értéke a feltételes elágazásokban? Mi történne, ha a három elágazást egy többágú szelekcióba vonnánk össze (*Egyébként* ágakkal)? Miért kell a visszalépésnél felszabadítani a cellát? A program módosításával és futtatásával ellenőrizzük válaszainkat! Futtassuk a programot úgy is, hogy az eljárás elején (tehát minden egyes hívásnál) kirajzoljuk az addig bejárt utat! Így figyelemmel kísérhetjük a visszalépéseket.



Az útkeresés első néhány próbálkozása (Útkeresés1.txt)

Visszalépéses keresés iterációval

Készítsük el az útkeresés iteratív algoritmusát! A rekurzív hívások egymásutánját ciklussal helyettesítjük. A ciklusmagot addig ismételjük, amíg el nem jutunk a célig, vagy már nincs lehetőségünk a további visszalépésre (a legelső celláról kellene visszalépnünk). Ez utóbbi feltételt a rekurzív algoritmusban azért nem vizsgáltuk, mert a hívásokból való visszatérés automatikusan leállította a rekurziót, ha nem találtunk megoldást.

Mivel elöltesztelő ciklust használunk, a *Kész* változó értékét a ciklus előtt és a ciklus végén is meghatározzuk. A visszalépések során pedig ügyelnünk kell arra, hogy tiltsuk le az aktuális cellát, többet ide ne lépjünk. Egyébként végtelen ciklusba kerülnénk. A tiltást egy speciális karakterrel, például egy "-" jellel fejezzük ki. Az algoritmus egyébként pontosan megfelel a rekurzív megoldásnak:

```
ELJÁRÁS Útkeresés(VÁLTOZÓ Sor, Oszlop MINT Egész, VÁLTOZÓ Kész MINT Logikai)
    Kész = (Sor == MaxSor ÉS Oszlop == MaxOszlop)
    CIKLUS AMÍG NEM Kész ÉS Sor ≥ 1 ÉS Oszlop ≥ 1
        ' A következő lépés keresése:
        Tábla(Sor, Oszlop) = "O"   ' bejelöljük, hogy elértük ezt a cellát
        HA Tábla(Sor, Oszlop+1) == "  " AKKOR   ' jobbra lépünk
            Oszlop = Oszlop + 1
        EGYÉBKÉNTHA Tábla(Sor+1, Oszlop) == "  " AKKOR   ' lefelé lépünk
            Sor = Sor + 1
        EGYÉBKÉNT   ' visszalépünk
            Tábla(Sor, Oszlop) = "-"   ' letiltjuk a cellát, hogy még egyszer ne jöjjünk erre
            HA Tábla(Sor, Oszlop–1) == "O" AKKOR Oszlop = Oszlop – 1   ' balról jöttünk
            EGYÉBKÉNT Sor = Sor – 1   ' fentről jöttünk
            ELÁGAZÁS VÉGE
        ELÁGAZÁS VÉGE
        Kész = (Sor == MaxSor ÉS Oszlop == MaxOszlop)
    CIKLUS VÉGE
    HA Kész AKKOR Tábla(Sor, Oszlop) = "O"
ELJÁRÁS VÉGE
```

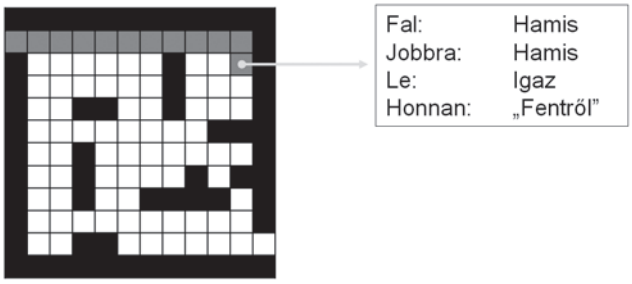
Az eljárást a *Sor* = 1, *Oszlop* = 1, *Útkeresés*(*Sor*, *Oszlop*, *Kész*) utasításokkal hívhatjuk.

2. gyakorlat. Írjuk meg az útkeresés iteratív programját! Az 1. gyakorlathoz hasonlóan jelenítsük meg a megoldást!

Hasonlítsuk össze az algoritmust a rekurzív megoldással! Miért nem vizsgáltuk meg az elágazások feltételeiben a *Kész* változó értékét? Mi történne, ha többágú szelekció helyett egymás utáni feltételes elágazásokat alkalmaznánk (a rekurzív megoldáshoz hasonlóan)? Miért volt szükség a ciklus után a *Tábla*(*Sor*, *Oszlop*) cella bejelölésére? Mi történne, ha nem írnánk "-" jelet a visszalépés előtt az aktuális cellába? A program módosításával és futtatásával ellenőrizzük válaszainkat!

A keresés általánosítása

Az előzőekben nagyon egyszerű adatszerkezettel ábrázoltuk a labirintust. Az útkeresés általánosításához célszerű a karakterként kezelt cellák jellemzőit struktúrában (rekordban) tárolni. Logikai változókban feljegyezzük, hogy fal van-e a cella helyén, továbbá, hogy jobbra, illetve lefelé léphetünk-e a celláról. Sztringként tároljuk azt az irányt, ahonnan a cellára léptünk (balról, illetve fentről):



A labirintus egy cellájához tartozó mezők értéke (Útkeresés1.txt)

```
STRUKTÚRA TCella
    VÁLTOZÓ Fal, Jobbra, Le MINT Logikai
    VÁLTOZÓ Honnan MINT Sztring
STRUKTÚRA VÉGE
```

A labirintust most is egy kétdimenziós tömb tartalmazza, minden szélén egy-egy többletsorral, illetve -oszloppal, melyek celláit falként kezeljük. A tömbelemek *TCella* típusúak. A mezők értékét a labirintus beolvasásánál kell megadni. Az algoritmus apró változtatásoktól eltekintve megfelel az előző szakaszban bemutatott eljárásnak:

```
ELJÁRÁS Útkeresés(VÁLTOZÓ Sor, Oszlop MINT Egész, VÁLTOZÓ Kész MINT Logikai)
    Kész = (Sor == MaxSor ÉS Oszlop == MaxOszlop)
    CIKLUS AMÍG NEM Kész ÉS Sor ≥ 1 ÉS Oszlop ≥ 1
        ' A következő lépés keresése:
        HA Tábla(Sor, Oszlop).Jobbra AKKOR   ' jobbra lépünk
            Tábla(Sor, Oszlop).Jobbra = Hamis ' többet innen ne lépjünk jobbra
            Oszlop = Oszlop + 1
            Tábla(Sor, Oszlop).Honnan = "Balról"
        EGYÉBKÉNTHA Tábla(Sor, Oszlop).Le AKKOR   ' lefelé lépünk
            Tábla(Sor, Oszlop).Le = Hamis ' többet innen ne lépjünk lefelé
            Sor = Sor + 1
            Tábla(Sor, Oszlop).Honnan = "Fentről"
        EGYÉBKÉNT   ' visszalépünk
            HA NEM Tábla(Sor, Oszlop+1).Fal AKKOR Tábla(Sor, Oszlop).Jobbra = Igaz
            HA NEM Tábla(Sor+1, Oszlop).Fal AKKOR Tábla(Sor, Oszlop).Le = Igaz
            HA Tábla(Sor, Oszlop).Honnan == "Balról" AKKOR
                Oszlop = Oszlop – 1   ' balról jöttünk
            EGYÉBKÉNT
                Sor = Sor – 1   ' fentről jöttünk
            ELÁGAZÁS VÉGE
        ELÁGAZÁS VÉGE
        Kész = (Sor == MaxSor ÉS Oszlop == MaxOszlop)
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Ezt az eljárást is a *Sor* = 1, *Oszlop* = 1, *Útkeresés*(*Sor*, *Oszlop*, *Kész*) utasításokkal indíthatjuk.

Az algoritmus felírásánál ügyeljünk az utasítások sorrendjére a feltételes elágazásokban! Hasonlítsuk össze az eljárást az előző szakaszban bemutatott algoritmussal! Melyik utasítást helyeztük át és miért? Szükség van-e a *Jobbra*, illetve a *Le* mezők hamisra állítására a feltételes elágazásokban?

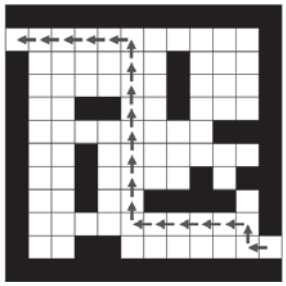
3. gyakorlat. Írjuk meg struktúra alkalmazásával az útkeresés programját!

A keresés után a bejárattól a kijáratig vezető utat a *Honnan* mezők segítségével kapjuk meg. A kijárattól indulva a mezők értéke szerint lépegetünk visszafelé, amíg csak el nem jutunk a bejáratig:

```
Sor = MaxSor, Oszlop = MaxOszlop
CIKLUS AMÍG NEM (Sor == 1 ÉS Oszlop == 1)
    ELÁGAZÁS Tábla(Sor, Oszlop).Honnan SZERINT
        "Balról" ESETÉN Ki: "Lépés jobbra", Oszlop = Oszlop – 1
        "Fentről" ESETÉN Ki: "Lépés lefelé", Sor = Sor – 1
    ELÁGAZÁS VÉGE
CIKLUS VÉGE
```

A ciklus fordított sorrendben írja ki a lépéseket. Egy verem felhasználásával a sorrend könnyen visszafordítható.

4. gyakorlat. Bővítsük a 3. gyakorlat programját a lépések kiírásával!



Az út felderítése a céltól a startig (Útkeresés1.txt)

Útkereső algoritmusunkat némileg egyszerűsíthettük volna, de fenti formájában könnyen általánosítható. Bővítésével például engedélyezhetjük a négyirányú mozgást. Ha egy celláról négyféle irányba is továbbléphetünk, akkor az irányok logikai értékét célszerű tömbben tárolni. Ekkor a következő lépésirányt ciklussal kereshetjük meg.

Az összes út meghatározása

A visszalépéses keresésre vonatkozó feladatok egy része az összes lehetséges megoldás meghatározását igényli. Ehhez „becsapjuk" a fenti algoritmust. Ha elérkeztünk egy megoldáshoz, akkor még a ciklusson belül kiíratjuk, majd visszalépünk. Ezzel arra kényszerítjük a programot, hogy egy további megoldást keressen. A ciklust csak akkor állítjuk le, ha már a legelső mezőről kellene visszalépnünk:

```
ELJÁRÁS ÖsszesÚt(VÁLTOZÓ Sor, Oszlop MINT Egész, VÁLTOZÓ Kész MINT Logikai)
    CIKLUS AMÍG Sor ≥ 1 ÉS Oszlop ≥ 1
        ' A következő lépés keresése:
        HA Sor == MaxSor ÉS Oszlop == MaxOszlop AKKOR ' találtunk egy megoldást
            Ki: Út
            Visszalép(Sor, Oszlop)
        EGYÉBKÉNTHA Tábla(Sor, Oszlop).Jobbra AKKOR
            ' jobbra lépünk (lásd az előző szakaszt)
        EGYÉBKÉNTHA Tábla(Sor, Oszlop).Le AKKOR
            ' lefelé lépünk (lásd az előző szakaszt)
        EGYÉBKÉNT
            Visszalép(Sor, Oszlop)
        ELÁGAZÁS VÉGE
    CIKLUS VÉGE
ELJÁRÁS VÉGE
```

Mivel a visszalépés két helyen is előfordul, külön eljárással végezzük. Az utasítások pontosan megfelelnek az előző szakasz algoritmusának:

```
ELJÁRÁS Visszalép(VÁLTOZÓ Sor, Oszlop MINT Egész)
    HA NEM Tábla(Sor, Oszlop+1).Fal AKKOR Tábla(Sor, Oszlop).Jobbra = Igaz
    HA NEM Tábla(Sor+1, Oszlop).Fal AKKOR Tábla(Sor, Oszlop).Le = Igaz
    HA Tábla(Sor, Oszlop).Honnan == "Balról" AKKOR
        Oszlop = Oszlop – 1  ' balról jöttünk
    EGYÉBKÉNT
        Sor = Sor – 1  ' fentről jöttünk
    ELÁGAZÁS VÉGE
ELJÁRÁS VÉGE
```

5. gyakorlat. Írjunk programot, amely meghatározza az összes lehetséges utat a bejárattól a kijáratig!

A visszalépéses keresés algoritmusa természetesen nem csak a labirintusokra alkalmazható. Különböző játékok, stratégiai feladatok és más problémák is megoldhatók a segítségével.