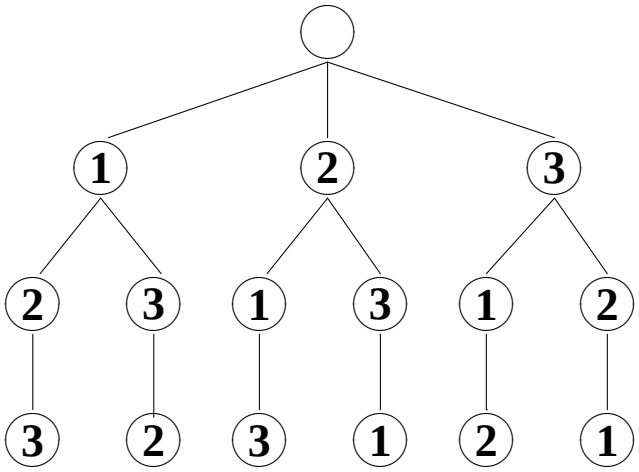
Visszalépéses módszer (Backtracking) – folytatás

Permutáció

$n = 3$ esetében:



Eredmény:

123 132 213 231 312 321

permutációk száma: $P_n = n!$

románul: ***permutări*****, angolul:** ***permutation***

n **elem permutációja,** $M = \{1, 2, \ldots, n\}$

Megoldás: $(x_1, x_2, \ldots, x_n) \in \underbrace{M \times M \times \ldots \times M}_{n\text{–szer}}$

(belső) feltétel: az elemek különbözzenek

PERMUTÁCIÓ(k)

```
1. for i := 1 to n
2.     do X_k := i
3.        if MEGFELEL(k)
4.          then if k < n
5.                 then PERMUTÁCIÓ(k + 1)
6.                 else kiír X
```

MEGFELEL(k)

```
1. OK := igaz
2. i := 1
3. while (i ≤ k − 1) és OK
4.       do if (X_k = X_i)
5.            then OK := hamis
6.            else i := i + 1
7. return OK
```

eljárás hívása: PERMUTÁCIÓ(1), adott n-re

Variáció

n elem m-ed osztályú variációja ($n \geq m$)

$n = 4, m = 2$ esetében:

12	**13**	**14**
21	**23**	**24**
31	**32**	**34**
41	**42**	**43**

variációk száma: $V_n^m = n(n-1)\ldots(n-m+1)$
románul: *aranjamente*, angolul: *permutation (!)*

VARIÁCIÓ(k)

```
1.  for i := 1 to n
2.      do X_k := i
3.          if MEGFELEL(k)
4.              then if k < m
5.                       then VARIÁCIÓ(k + 1)
6.                       else kiír X
```

MEGFELEL(k) — ugyanaz, mint a permutációnál

eljárás hívása: VARIÁCIÓ(1), adott n-re és m-re

Kombináció

n elem m-ed osztályú kombinációja ($n \geq m$)

$n = 4, m = 2$ esetében:

12	**13**	**14**
~~**21**~~	**23**	**24**
~~**31**~~	~~**32**~~	**34**
~~**41**~~	~~**42**~~	~~**43**~~

kombinációk száma: $C_n^m = \dfrac{n(n-1)\ldots(n-m+1)}{m!} = \dfrac{n!}{m!(n-m)!}$

Más jelölés: $C_n^m = \binom{n}{m}$ **rom.:** ***combinări*, ang.:** ***combination***

KOMBINÁCIÓ(k) ugyanaz, mint a variációnál

MEGFELEL(k)

1. $OK := igaz$
2. $i := 1$
3. **while** $(i \leq k-1)$ **és** OK
4. **do if** $(X_k \leq X_i)$
5. **then** $OK := hamis$
6. **else** $i := i + 1$
7. **return** OK

eljárás hívása: KOMBINÁCIÓ(1), adott n-re és m-re

Visszalépéses módszer — iteratív változat

jelölés: m_k **az** M_k **következő vizsgálandó eleme**

Királynős feladat nem rekurzívan:

KIRÁLYNŐ_ITERATÍVAN

```
1.  for i := 1 to n
2.      do m_i := 1
3.  k := 1
4.  while k > 0
5.        do OK := hamis
6.           i := m_k
7.           while (i ≤ n) és nem OK
8.                 X_k := i
9.                 m_k := i + 1
10.                OK := MEGFELEL(k)
11.                i := i + 1
12.          if OK
13.                then if k = n
14.                        then KIÍR(X)
15.                        else k := k + 1
16.                else m_k := 1
17.                     k := k − 1
```

ITERATÍV_BACKTRACKING

```
1.  for i := 1 to n
2.      do m_i := 1                          ▷ kezdőértékek beállítása
3.  k := 1
4.  while k > 0
5.        do OK := hamis
6.           i := m_k
7.           while (i ≤ n) és nem OK          ▷ létezik köv. elem M_k-ban
8.                 X_k := i                   ▷ következő elem M_k-ból
9.                 m_k := i + 1               ▷ köv. érték beállítása M_k-ban
10.                OK := MEGFELEL(k)
11.                i := i + 1
12.          if OK                            ▷ megfelelő elem
13.                then if k = n
14.                        then KIÍR(X)       ▷ eredmény kiírása
15.                        else k := k + 1    ▷ nincs vége, továbblépés
16.                else m_k := 1              ▷ új kezdőérték beállítása M_k-ban
17.                     k := k − 1            ▷ visszalépés az előző X elemre
```

Egyéb példák

Labirintus

A köv. labirintusban az 1-esek jelzik a folyosót, ahol haladni lehet vízszintesen és függőlegesen.

A feladat: egy adott pozícióból indulva kijutni a labirintusból, ha ez lehetséges.

0	1	0	0	0
0	1	1	0	0
0	0	1	1	0
0	1	1	1	0
0	1	0	1	0

Egy adott helyről 4 irányba indulhatunk:

		1		
	4	■	2	
		3		

Hasonlóképpen járunk el, mint a *lóugrás* feladatnál. A lépéseket a köv. vektorokkal oldjuk meg:

$a = (-1, 0, 1, 0)$

$b = (\ 0, 1, 0, -1)$

Az labirintust a X mátrix tartalmazza. A megoldás lépéseit az E vektorban őrizzük meg (1-estől számozva).

LABIRINTUS(i, j, s)

```
1.  for k := 1 to 4
2.      do p := i + a_k
3.         r := j + b_k
4.         if MEGFELEL(p, r)
5.            then E_pr := s
6.                 if (p = 1) vagy (p = n) vagy  (r = 1) vagy (r = n)
7.                    then KIÍR(E)
8.                         E_pr := 0
9.                    else LABIRINTUS(p, r, s + 1)
10.                        E_pr := 0
```

MEGFELEL(p, r)

```
1.  if (1 ≤ p ≤ n) és (1 ≤ r ≤ n) és (X_pr = 1) és (E_pr = 0)
2.     then OK := igaz
3.     else OK := hamis
4.  return OK
```

Az eljárás hívása, adott i **és** j **értékre:**

$E_{i,j} := 1$ ▷ **Feltételezzük, hogy** $X_{ij} = 1$**.**

$s := 2$

LABIRINTUS(i, j, s)**;**

Labirintus — másképpen

LABIRINTUS′(i, j, s)

```
 1. if MEGFELEL(i, j)
 2.    then E_ij := s
 3.         if (i = 1) vagy (i = n) vagy  (j = 1) vagy (j = n)
 4.            then KIÍR(E)
 5.                 E_ij := 0
 6.            else LABIRINTUS′(i − 1, j, s + 1)
 7.                 LABIRINTUS′(i, j + 1, s + 1)
 8.                 LABIRINTUS′(i + 1, j, s + 1)
 9.                 LABIRINTUS′(i, j − 1, s + 1)
10.                 E_ij := 0
```

A MEGFELEL(i, j) ugyanaz, mint az előző megoldásnál.

Az eljárás meghívása:

LABIRINTUS′$(i, j, 1)$**;**

Példák:

A labirintus:

0	1	0	0	0
0	1	1	0	0
0	0	1	1	0
0	1	1	1	0
0	1	0	1	0

$i = 4$, $j = 4$ pozícióból indulva, a megoldások:

0	6	0	0	0
0	5	4	0	0
0	0	3	2	0
0	0	0	1	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	3	2	0
0	5	4	1	0
0	6	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	1	0
0	0	0	2	0

0	6	0	0	0
0	5	4	0	0
0	0	3	0	0
0	0	2	1	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	3	2	1	0
0	4	0	0	0

Legrövidebb utak labirintusban

Az előbbi feladat kibővítése azzal, hogy még találja meg a labirintusból kivezető legrövidebb út hosszát is.

```
LABIRINTUS''(i, j, s)
 1. if MEGFELEL(i, j)
 2.    then E_ij := s
 3.         if (i = 1) vagy (i = n) vagy  (j = 1) vagy (j = n)
 4.            then KIÍR(E)
 5.                 E_ij := 0
 6.                 return 0
 7.            else min :=MINIMUM(LABIRINTUS''(i − 1, j, s + 1),
 8.                               LABIRINTUS''(i, j + 1, s + 1))
 9.                 min :=MINIMUM(min, LABIRINTUS''(i + 1, j, s + 1))
10.                 min :=MINIMUM(min, LABIRINTUS''(i, j − 1, s + 1))
11.                 E_ij := 0
12.                 return min + 1
13.    else return ∞
```

```
MINIMUM(a, b)
1. if a > b
2.    then return b
3.    else return a
```

A MEGFELEL(*i*, *j*) ugyanaz, mint az előző esetben.

Az eljárás hívása:

m := **LABIRINTUS″**(*i*, *j*, 1)**,**

megfelelő *i* **és** *j* **értékekre, és** *m* **lesz a legrövidebb út hossza.**

Természetes szám felbontása prímszámok összegére

Feladat: Írjunk fel egy adott n természetes számot prímszámok összegére, az összes lehetséges módon.

Egy $P_1, P_2, \ldots$ sorozatban őrizzük a prímszámokat. Feltételezzük, hogy elegendőek a megadott természetes szám felbontására.

PRÍMEK(k)

```
 1. i := 1
 2. repeat
 3.         X_k := P_i
 4.         if MEGFELEL(k)                    ▷ itt s is értéket kap
 5.           then if s < n
 6.                   then PRÍMEK(k + 1)
 7.                        s := 0
 8.                   else  if s = n
 9.                           then KIÍR(X)
10.         i := i + 1
11. until s > n
```

```
MEGFELEL(k)
 1. OK := hamis
 2. if k = 1
 3.    then OK := igaz
 4.         s := X_1
 5.    else if X_k ≥ X_{k-1}
 6.            then s := 0
 7.                 for i := 1 to k
 8.                     do s := s + X_i
 9.                 if   s ≤ n
10.                     then OK := igaz
11  return OK
```

Hívás: PRÍMEK(1).

Példák:

$n = 10$ **felbontásai:**

```
2  2  2  2  2
2  2  3  3
2  3  5
3  7
5  5
```

$n = 15$ **felbontásai:**

2 2 2 2 2 2 3
2 2 2 2 2 5
2 2 2 2 7
2 2 2 3 3 3
2 2 3 3 5
2 2 11
2 3 3 7
2 3 5 5
2 13
3 3 3 3 3
3 5 7
5 5 5

Legrövidebb lefedő szavak

Az A ábécé $a_1, a_2, \ldots, a_m$ betűivel képezzük az összes k hosszúságú szót.

Melyek azok a legrövidebb szavak, amelyek tartalmazzák mindezeket részszóként.

Be lehet látni, hogy a lehető legrövidebb ilyen szó nem lehet rövidebb $(m^k + k - 1)$-nél.

Például, ha $m = 2$ és $k = 3$, akkor a 0001011100 és 0001110100 szavak mindegyike legrövidebb lefedő szó, mivel tartalmazzák a 000, 001, 010, 011, 100, 101, 110, 111 szavakat.

- **Az M_k halmazok mindegyike egyenlő A-val.**
- **A (belső) feltétel: az utoljára képzett k hosszúságú szó először szerepeljen a lefedő szóban.**
- **Megállási feltétel: ha a lefedő szó hossza $(m^k + k - 1)$.**

LEFEDŐ_SZÓ(i)

1. **for** $j := 1, 2, \ldots, m$
2. **do if** $S_{i-k+1}S_{i-k+2}\ldots S_{i-1}a_j$ **nem része** $S_1S_2\ldots S_{i-1}$**-nek**
3. **then** $S_i := a_j$
4. **LEFEDŐ_SZÓ($i+1$)**
8. **else if** $hossz[S_1\ldots S_{i-1}] = m^k + k - 1$
9. **then írd** $S_1\ldots S_{i-1}$
10. **exit for**

Az eljárás hívása, adott m és k értékekre:

for $i = 1, 2, \ldots, k$
do $S_i := a_1$
LEFEDŐ_SZÓ $(k+1)$**.**

Más megoldás:

Betűk helyett vegyük az $a_i = i - 1$ ($i = 1, 2, \ldots, m$) értékeket, amelyek egy m-alapú számrendszer számjegyei.

Az algoritmusban az $S = (S_1, S_2, \ldots)$ vektor a lefedő szó "betűit" tartalmazza.

A $B = (B_1, B_2, \ldots)$ vektor komponensei jelzik, hogy egy adott részszó szerepel már (a komponens 1), vagy még nem (a komponens 0) a lefedő szóban.

Az aktuális betűnek a szóhoz való illesztése után kiszámítjuk az utolsó k betűből képzett $S_{i-k+1}S_{i-k+2}\ldots S_i$ részszónak megfelelő számot: val($S_{i-k+1}S_{i-k+2}\ldots S_i$), ha ezt r-rel jelöljük, akkor B_r-t 1-re állítjuk, amennyiben 0 volt.

Kezdetben:

$S_i := 0$, $i = 1, 2, \ldots, k$,

$B_0 := 1$, **és minden más** i**-re** $B_i := 0$.

LEFEDŐ_SZÓ′(i)

```
1.  for j := 1, 2, ..., m
2.      do S_i := j − 1
3.          r := val(S_{i−k+1} S_{i−k+2} ... S_i)      ▷ if MEGFELEL helyett
4.          if B_r = 0                                  ▷ if MEGFELEL helyett
5.            then B_r := 1
6.                 LEFEDŐ_SZÓ′(i + 1)
7.                 B_r := 0
8.            else if hossz[S_1 ... S_{i−1}] = m^k + k − 1
9.                   then írd S_1 ... S_{i−1}
10.                       exit for
```

Az eljárás hívása, adott m és k értékekre:

```
for i = 1, 2, ..., k
    do S_i := 0
B_0 := 1
for i = 1, 2, ..., m^k − 1
    do B_i := 0
LEFEDŐ_SZÓ′ (k + 1).
```

$m = 3, k = 2$ esetében az algoritmus eredménye:

0010211220
0010221120
0011021220
0011022120
0011202210
0011210220
0011220210
0011221020
0012022110
0012110220
0012202110
0012211020
0020112210
0020122110
0021011220
0021101220
0021122010
0021220110
0022011210
0022012110
0022101120
0022110120
0022112010
0022120110

Feladatok:

1. Írjunk ki minden helyesen nyitó és csukó n zárójelet tartalmazó karakterláncot!

2. Ismert egy $n \times n$-es sakktáblán egy ló és egy király pozíciója. A tábla egyes helyei égnek, ide nem lehet lépni. Állítsuk elő az összes lehetőséget, ahogy a ló a királyért mehet, majd onnan visszatér – hátán a királlyal – eredeti helyére. Ahova lép, az a négyzet felgyúl. Természetesen, a ló és a király eredeti pozíciója nem ég.

3. Adott egy $n \times m$ méretű mátrix, amelynek elemei egész számok. Írjuk ki azt a legkisebb összeget, amelynek elemeit különböző sorokból és oszlopokból vettük!

4. Állítsuk elő az összes $f : A \to B$ szürjektív függvényt, ha $A = \{1, 2, \ldots, n\}$ és $B = \{1, 2, \ldots, m\}$!

5. Legyen n tárgy 1-től n-ig számozva és n doboz. Ha egy dobozba legfennebb m tárgy fér bele, állítsuk elő az összes lehetőséget, ahogy a tárgyak elhelyezhetők a dobozokban!