Kombinatorikai algoritmusok

Tartalom

Az összes … előállítása 4
Ismétlés nélküli permutációk 4
A cél: az összes permutáció előállítása 4
Egy naiv megvalósítás 4
Backtrack-alapú megvalósítás 5
Egy rekurzív megoldás 6
Feladatok 6
1. feladat 6
Megoldási ötlet 6
2. feladat 7
Megoldási ötlet 7
Ismétlés nélküli kombinációk 7
A cél: az összes kombináció előállítása 7
Backtrack-alapú megvalósítás 7
Részhalmazok 8
A cél: az összes részhalmaz előállítása 8
1. megoldásötlet 8
2. megoldásötlet 9
Feladatok 9
1. feladat 9
Megoldásötlet 9
Feladatok 9
1. feladat 9
Megoldásötlet 9
2. feladat 10
Az i. … előállítása 11
Ismétlés nélküli permutáció 11
A permutációk indexelése 11
Rendezettség-alapú ötlet 11
Rendezés-alapú ötlet 12
Az i. permutáció előállítása 15
Megoldás 15
Feladatok 15
1. feladat 15
Megoldási ötlet 15
Ismétléses variáció 16
A variációk indexelése 16
Az i. variáció előállítása 16
Feladatok 16
1. feladat 16

Az összes … előállítása

Ebben a részben eltöprengünk néhány kombinatorikai szempontból érdekes probléma összes megoldásának előállításán.

ISMÉTLÉS NÉLKÜLI PERMUTÁCIÓK

A cél: az összes permutáció előállítása

Állítsuk elő az összes N-elemű ismétlés nélküli permutációt! A permutáció elemei legyenek 1..N között természetes számok!

> Ez a megszorítás nyilvánvalóan nem lényegi megszorítás, ha a permutáció alaphalmazának egyes elemeihez annak sorszámát feleltetjük meg, azaz bijekciót hoztunk létre a felhasznált természetes szám intervallum és az eredeti alaphalmaz között.

Egy naiv megvalósítás

Mivel igen egyszerű N elem összes N-elemű ismétléses variációját előállítani, ezért kiindulunk ennek algoritmusából. Ha ezt megoldottuk, akkor már csak egy szűrőt kell beillesztenünk az algoritmusba, amely a permutációk közé nem való N-eseket számításon kívül hagyja.

N elem összes N-ed rendű ismétléses variációját előállítása valójában az összes N-jegyű N-alapú szám előállítását jelenti. Persze ez az N-egyű szám „furcsa”, hiszen számjegyei között 0- nincs, cserében az N igen. Azaz az 11…1-től az NN…N-ig kell generálnunk a variációkat.

Az algoritmus az alábbi. A permutációt „helyben” felhasználjuk (nem gyűjtjük tömbben), meghívva a most kidolgozatlanul hagyott PermFeldolg eljárást:

```
Konstans
  MaxN:Egész(?)
Típus
  TPerm=Tömb(1..MaxN:Egész)
Eljárás ÖsszesPermutáció(Konstans N:Egész):
  Konstans
    P1:TPerm(1,…,1) [MaxN darab 1-es]
  Változó
    permSorsz,i:Egész
    P:TPerm
  permSorsz:=0
  P:=P1; i:=N
  Ciklus amíg i>0
    Ha PermutációE(P) akkor
      permSorsz:+1; PermFeldolg(P,N,permSorsz)
    Elágazás vége
    Ciklus amíg i>0 és P(i)=N
      P(i):=1; i:-1
    Ciklus vége
    Ha i>0 akkor P(i):+1; i:=N
  Ciklus vége
Eljárás vége.
```

A „főszereplő” típus definiálása.

A lényegi eljárás szignatúrája és lokális adatainak deklarálása.

P1 az első permutáció.

A ciklus előkészítése.

A sorozat szűrése és a –permutáció esetén– feldolgozása.

A következő variáció előállítása

A ciklusból kiléptünk, amikor az NN…N-t is túlhaladtuk.

Fel kellett használnunk az előállított variáció permutáció voltát eldöntő függvényt. Ezt definiáljuk most. Lényegében azt kell eldöntenie, hogy van-e a sorozatban elemismétlődés.

```
Függvény PermutációE(Konstans P:TPerm):Logikai
  Változó
    i,j:Egész
    mégOK:Logikai
  i:=2; mégOK:=Igaz
  Ciklus amíg i≤N és mégOK
    j:=1
    Ciklus amíg j<i és P(j)≠P(i)
      j:+1
    Ciklus vége
    mégOK:=j=i
    i:+1
  Ciklus vége
  PermutációE:=mégOK
Függvény vége.
```

A „permutáció”-ságot vizsgáló függvény szignatúrája és lokális adatainak deklarációja.

Van-e olyan P(i:2..N), amely előtt már volt belőle?

Az i. valamelyik előzőnek a duplikátuma-e?

Ennek a naiv megoldásnak nyilvánvaló baja a rossz hatékonyság, hiszen végig menni az összes variáción: N^N elem előállítását jelenti, amíg számunkra érdekes elem „mindössze” N! van. Ha a „permutáció”-ság vizsgálatát nem is vesszük számításba (amit mind az N^N-szer elvégzünk), a többlet elemvizsgálat száma így is borzasztóan alakul: $1–N!/N^N$. (A hányados alakulása N=2..10-re: 0,5000; 0,7778; 0,9063; 0,9616; 0,9846; 0,9939; 0,9976; 0,9991; 0,9996.) Gyakorlatilag csupa felesleges elemet vizsgálgatunk.

Backtrack-alapú megvalósítás

Érdemes a felesleges variációk előállítását még ideje korán (azaz az első ismétlődő elem felbukkanásakor) abbahagyni. Ez az alapgondolata a jól ismert **backtrack**-nek, vagy más néven a visszalépéses keresésnek. Most persze nem keresünk vele –a programozási tételek szokásos terminológiája szerint–, hanem az összes megfelelőt **kiválogat**juk. Az alábbi algoritmusban valójában ismét rögvest felhasználjuk a generált permutációt (a PermFeldolg eljárásban). Az alap backtrack változathoz képest a kiválogatás miatti „újdonságokat” pirossal emeltem ki.

```
Eljárás ÖsszesPermutáció(Konstans N:Egész):
  Konstans
    X0:TPerm(0,…,0) [MaxN darab 0-ás]
  Típus
    TKeresett=Rekord(van:Logikai; melyik:Egész)
  Változó
    i:Egész
    X:TPerm [aktuális permutáció]
    ker:TKeresett
    permSorsz:Egész
  permSorsz:=0
  i:=1; X:=X0
  Ciklus amíg i≥1
    Ha i>N akkor
      permSorsz:+1
      PermFeldolg(X,N,permSorsz)
      i:-1
    Elágazás vége
    ker:=JóElem(i)
    Ha ker.van akkor
      X(i):=ker.melyik; i:+1
    különben
      X(i):=0; i:-1
    Elágazás vége
  Ciklus vége
Eljárás vége.
```

A lényegi eljárás fejsora és lokális adatainak deklarációja.

A permSorsz változó kódolásánál ügyelni kell arra, hogy viszonylag nagy egész számok fognak belekerülni.

A megszokott backtrack ciklus inicializáló része. A permSorsz a „kiválogatásos” variációhoz tartozik.

Ha összeállt egy helyes permutáció, akkor következhet a feldolgozása (pl. a megjegyzése, most a PermFeldolg eljárás hívása).

Továbblépünk a következő permutáció felé, vagy visszalépünk, a backtrack logikája szerint.

A backtrack mindenkori részeként jön a JóElem függvény, amely megvizsgálja, hogy i. elemként van-e választható (tehát eddig még elő nem forduló) elem.

```
Függvény JóElem(Konstans i:Egész):TKeresett
  Változó
    j:Egész
  j:=X(i)+1
  Ciklus amíg j≤N és nem LehetségesE(i,j)
    j:+1
  Ciklus vége
  JóElem.van:=j≤N
  Ha JóElem.van akkor JóElem.melyik:=j
Függvény vége.
```

A legutóbbi függvényben felhasznált függvény maradt hátra, amely arra hivatott, hogy eldöntse: i. elemként j értékű elem választható-e, azaz egyedi-e.

```
Függvény LehetségesE(Konstans i,j:Egész):Logikai
  Változó
    k:Egész
  k:=1
  Ciklus amíg k<i és X(k)≠j
    k:+1
  Ciklus vége
  LehetségesE:=k=i
Függvény vége.
```

Egy rekurzív megoldás

```
Típus
  TPermek=Tömb(1..MaxPerm:TPerm)
Változó
  pk:TPermek [az összes permutáció]
  pDb:Egész  [a permutációk száma]
Eljárás ÖsszesPermutáció(Konstans aktP:TPerm,
                                 i,n:Egész):
  Változó j:Egész
          P:TPerm(aktP)  [kezdőértékadás]
  Ha i>n akkor
    pDb:+1; pk(pDb):=P   [~PermFeldolg(…)]
  különben
    P(i):=i; ÖsszesPermutációk(P,i+1,n)
    Ciklus j=i-1-től 1-ig -1-esével
      Csere(P,j,j+1); ÖsszesPermutáció(P,i+1,n)
    Ciklus vége
  Elágazás vége
Eljárás vége.
```

Az összes permutációt gyűjtő globális pk tömb (definiálása és) deklarálása.

Az aktP n-ed rendű permutációjából kiindulva, generáljuk a következő i. komponensét.

Nincs n-nél nagyobb komponens, elkészült a permutáció, feljegyezhető.

Még nincs kész. Generáljuk az összes olyant, amelyek i. komponense i,

majd cserékkel permutáljuk a kezdőszeleteket, és a hozzájuk tartozó összes permutációt generáljuk.

Feladatok

1. feladat

Egy jól ismert fejtörő egy aritmetikai művelet kapcsol egybe szavakat. A pontos feladat az, hogy a szavak egyes betűinek feleltessünk meg egy számjegyet úgy, hogy a művelet helyes eredményt szolgáltasson a szavakon. Pl. SEND + MORE = MONEY.

Megoldási ötlet

A szavakban –összességben– előforduló jelekhez (SENDMORY) keressük 0..9 számjegyek egyértelmű hozzárendelését. A „klasszikus” algebrai megoldás nehézségekbe ütközik. Az alapötlet szerint adott annyi egyenlet, ahány „helyiérték”: „D+E=Y”, „N+R=E”,… „M=?” (?=1). Továbbgondolva más egyenletrendszer is szolgálhat megoldással. Ebben átvitel képződését feltételezzünk: „D+E=10+Y”,

„N+R+1=E”, … És még jó néhány egyenletrendszer. Ezek egyike, esetleg közülük több is megoldható a peremfeltételek mellett: minden ismeretlen 0..9 között, sőt S,M>0. Hát ez így túl bonyodalmas lenne.

Keressünk más megoldást! Ehhez az előzőekben taglalt összes permutáció algoritmusát átvehetjük, csak éppen módosítani kell a PermFeldolg eljárást, amely ez esetben ellenőrzi a permutáció, a feladat szempontjából való helyességét, és gondoskodik az esetleges megoldás gyűjtéséről vagy kiírásáról.

2. feladat

Végezzük el a hatékonysági összevetést a naiv (összes variáció módosításával kapott) és a backtrack alapú megoldás között!

Megoldási ötlet

Mivel mindkettőben meghatározó szerepű a sorozat elemeinek összehasonlítása, ezért ezek végrehajtási számát kellene számlálni, majd összevetni. Ilyen elemhasonlítást találunk a naiv megoldás legfelsőbb szintű eljárásában is, és a PermutációE függvényben is, ill. a backtrack megoldás LehetségesE függvényében.

ISMÉTLÉS NÉLKÜLI KOMBINÁCIÓK

A cél: az összes kombináció előállítása

Adott N elemhez soroljuk föl az elemek összes K-ad osztályú kombinációját, azaz adjuk meg az összes pontosan K elemet tartalmazó olyan sorozatot, amelyben az eredeti sorozat elemei szerepelnek és legfeljebb egy-egy példányban.

Backtrack-alapú megvalósítás

Egyetlen kérdés vetődik föl: miként érjük el azt, hogy a csak az elemek sorrendjében különböző K-asakat **egyszer** vegyük figyelembe. Egyik szokásos trükköt alkalmazzuk: csak (mondjuk: növekvően) **rendezett sorozat**okat generálunk, így eleve kizárt, hogy ugyanolyan elemű rész-sorozatok kijöhessenek.

Legkézenfekvőbbnek a backtrack előbbiekben elemzett permutációhoz igazított megoldásból kiindulni. Ebben a legfelsőbb szinten lényegi változás alig van:

- az azonosítókat értelemszerűen átírjuk;
- az eljárás fejsora bővül a K paraméterrel;
- a TKomb típus most is MaxN elemű, de természetesen csak az első K elemére lesz szükség;
- a KombFeldolg eljárásnak átadjuk a K paramétert is;
- az egyedüli „érdemi” módosítás a K-ad osztályúság felismerésében van.

```
Eljárás ÖsszesKombináció(Konstans N,K:Egész):
  Konstans
    X0:TKomb(0,…,0)
  Típus
    TKeresett=Rekord(van:Logikai; melyik:Egész)
  Változó
    i:Egész
    X:TKomb [aktuális kombináció]
    ker:TKeresett
    kombSorsz:Egész
```

A lényegi eljárás fejsora és lokális adatainak deklarációja.

A kombSorsz változó kódolásánál ügyelni kell arra, hogy viszonylag nagy egész számok fognak belekerülni.

```
kombSorsz:=0
i:=1; X:=X0
Ciklus amíg i≥1
  Ha i>K akkor
    kombSorsz:+1
    KombFeldolg(X,N,K,kombSorsz)
    i:-1
  Elágazás vége
  ker:=JóElem(i)
  Ha ker.van akkor
    X(i):=ker.melyik; i:+1
  különben
    X(i):=0; i:-1
  Elágazás vége
Ciklus vége
Eljárás vége.
```

A megszokott backtrack ciklus inicializáló része. A kombSorsz a „kiválogatásos" változathoz tartozik.

Ha összeállt egy helyes kombináció, akkor következhet a feldolgozása (pl. a megjegyzése, most a KombFeldolg eljárás hívása).

Tovább lépünk a következő kombináció felé, a backtrack logikája szerint.

A JóElem függvény változatlanul jó úgy, hiszen most is N elem közül kell egy „lehetségest" kiválasztani.

A LehetségesE függvény feladata: „döntse el, hogy választható-e (azaz egyedi-e és növekvő-e) az i. elemként kijelölt j érték". Ez drasztikus módosítást, mégpedig –hihetetlen, de– egyszerűsítést jelent:

```
Függvény LehetségesE(Konstans i,j:Egész):Logikai
  LehetségesE:=X(i-1)<j
Függvény vége.
```

RÉSZHALMAZOK

A cél: az összes részhalmaz előállítása

Adott egy halmaz N darab elemének felsorolásával. Állítsuk elő az összes részhalmazát! Most is megengedjük, hogy a sorozat elemei 1..N között természetes számok legyenek.

1. megoldásötlet

Igen szoros kapcsolat van egy halmaz elemeit felsoroló vektor és egy kombinációt tartalmazó vektor között. Mindkettőre igaz, hogy **nincs elemismétlődés**. Pontosíthatunk: az N-elmű bázishalmazból képzett K-elemű (rész)halmazok bijektív kapcsolatban állnak az N elem K-ad osztályú kombinációkkal. Hiszen az előbbi közös tulajdonság mellett az is teljesül mindkettőre, hogy belőlük kettő csak akkor azonos, ha ugyanazokat az elemeket tartalmazzák.

Ekkor már is adódik egy algoritmus az összes részhalmaz előállítására: képezzük az üres halmazt, az összes 1-elemű, 2-elemű, … K-elemű, … N-elemű részhalmazt az előző részben taglalt backtrack-es algoritmussal. Kihasználjuk, hogy a halmaznak 0 nem eleme, így a generált kombinációbeli 0-k nem tartoznak a kombinációhoz, most a részhalmazhoz. Az üres részhalmaz tehát a kezdőállapotot jelentő X0 maga. (L. az ÖsszesKombináció algoritmusában.)

```
Eljárás ÖsszesRészhalmaz(Konstans N:Egész):
  …
  Változó
    K:Egész
    rhk:TRészHalmazok
    kombSorsz:Egész [≈részhalmaz-sorszám]
    …
  Eljárás ÖsszesKombináció(Konstans N,K:Egész):
    …
  Eljárás vége.
```

A lényegi eljárás fejsora és lokális adatainak deklarációja.

Ez a lokális eljárás előállítja N elem összes K-elemű részhalmazát. A kombSorsz-ot nem inicializálja.

```
    rhk(1):=X0   [üres halmaz]
    kombSorsz:=1
    Ciklus K=1-től N-1-ig
      ÖsszesKombináció(N,K)
    Ciklus vége
    rhk(kombSorsz):=(1,2,…,N)   [„teli” halmaz]
  Eljárás vége.
```

Első az üres részhalmaz.
A kombSorsz inicializálása.
Majd megszervezi minden K-ra a K-elemű részhalmazok generálását.
Végül a „teli” részhalmaz.

Igenám, de –jobban belegondolva– ez nem túl hatékony hozzáállás, ugyanis a K-eleműek képzése során minden K-nál kisebb elemszámú részsorozat újrakészül.

2. megoldásötlet

A halmaztípusok szokásos ábrázolása a logikai vektoros, azaz a halmaz potenciális elemeit indexeljük (1..N), és bentlétüket jegyezzük föl: THalmaz=Tömb(1..N:Logikai).

Értelmezése nyilvánvaló: legyen h:THalmaz, h(i)=Igaz ↔ az i. elem a h-nak eleme.

A feladat tehát az, hogy soroljuk föl összes lehetséges módon az N darab Igaz/Hamis értékből álló **ismétléses variáció**t. Ha Igaz értéket 1-gyel, a Hamis-at 0-val kódoljuk, akkor az N elemű tömb helyett egy N számjegyű bináris számmal írjuk le a halmazt. A feladat tovább egyszerűsödött: soroljuk föl az összes $0..2^N–1$ közötti számot (amelyek bináris alakjából rekonstruálható, hogy az adott szám mely elemek bentlétét írja elő).

Vegyük észre a két megoldásötlet kombinatorikai „furcsaságát”: az első algoritmus N elem összes K-ad osztályú kombinációt állította elő, míg a második 2 elem összes ismétléses variációját… és mindketten ugyanazt állították elő! De emlékezzünk a matematika jól ismert összefüggésére:

$$\sum_{K=0}^{N} \binom{N}{K} = 2^N$$

Feladat

1. feladat

Módosítsa az előbbi megoldást úgy, hogy figyelembe vesz egy újabb paramétert: a K-t, amely a részhalmazok maximális elemszámát korlátozza.

FELADATOK

Gondolja meg az eddigi feladatoknak azt a változatát, amelyben szerepel az „ismétléses” jelző! Ismét kézenfekvő ötlet a backtrack alkalmas módosítása.

1. feladat

Adjon algoritmust N elem összes K-ad rendű ismétléses permutációinak a felsorolására! Tudjuk (ilyen az N-es), hogy az elemek benne K_1, K_2, …K_M–szer ismétlődnek, de természetesem teljesül:

$$K_1+K_2+…+K_M=N.$$

Megoldásötlet

Úgy kell elképzelni a feladatot, hogy eleve adott az N-elemű sorozat, amelyben K_1-szer ismétlődve szerepel az e_1, K_2-ször az e_2 é.í.t. Észre kell vennünk, hogy megszűnt a kölcsönösen egyértelmű kapcsolat a sorozat és az indexeik (1..N) között.

Ha backtrack-alapú megoldást készítünk, akkor a legsúlyosabb változást a LehetségesE függvényt szenvedi. Ennek kell megvizsgálnia, hogy i.-ként a j. elem (nem érték, hanem index!) elfogadható-e. Akkor elfogadható, ha

a) még nem választottuk (ez az ismétlés nélkülinél is így volt);

b) az ugyanilyen értékűek közül ez a legnagyobb indexű.

A b) feltétel magyarázata: az azonos elemekből álló „részpermutációk” közül csak egyet szabad elfogadni, akárcsak a kombinációk esetében. (Ott a kombináció lényegéből fakadó ok miatt, itt a megkülönböztethetetlenség miatt.)

---

2. feladat

Adjon algoritmust N elem összes K-ad rendű ismétléses kombinációinak a felsorolására!

Az i. … előállítása

ISMÉTLÉS NÉLKÜLI PERMUTÁCIÓ

Adott egy N elemű sorozat. Adjuk meg az i. olyan sorozatot, amely elemei a bemeneti sorozat elemei, de –természetesen– különbözőnek tekintjük az eltérő sorrendűeket!

A permutációk indexelése

A permutáció elemei legyenek ismét 1..N között természetes számok.

Az i.-séghez az kell, hogy a permutációkat felsoroljuk. Azaz létezzék egy PermIndex „index-függvény” az alábbi tulajdonságokkal:

(PI1) PermIndex: Perm→[1..**N!**],
ahol Perm⊂[1..N]^N, ∀p∈Perm: Halmazfelsorolás(p)
(azaz a Perm az N hosszú, [1..N]-elemű sorozatok közül csak a permutációkat tartalmazza)

(PI2) PermIndex **bijekció**

Mivel a PermIndex-nek bijekciónak kell lennie, azaz tetszőleges P permutáció minden egyes elemének helyétől függnie kell, logikusnak látszik olyan összefüggést keresni, amelyben az egyes P-elemek helye, mint az index egy valamilyen számrendszerbeli számjegyeként legyen jelen.

Emlékezzünk: egy $s_N...s_2s_1$ számjegysorozattal adott A-alapú számrendszer decimális értékét az alábbi formula határozza meg:

(A2Dec1) $$A2Dec(s_N...s_2s_1):=\sum_{i=1}^{N} s_i * A^{i-1}$$

Elvárás a szemjegyekkel szemben:

(A2Dec2) $s_i \in [0..A-1]\ \forall i \in [1...N]$.

Rendezettség-alapú ötlet

(Ö1) Legyen az 1 permutációindexű a növekvően rendezett P1 permutáció; azaz P1=(1,2,…,N):
PermIndex((1,2,…,N)):=1

(Ö2) Minden permutációt a P1 a permutációhoz viszonyítunk: azaz a P permutáció **j.** eleméhez rendeljük hozzá a P1-beli helyétől (azaz j-től) való *eltérését*: P_j–$P1_j$-t, vagyis **P_j–j**-t. Ez legyen a P permutációhoz rendelt indexbeli számnak, mint A-alapú számnak a **j. számjegye**.

Folyományok:

F1. Az eltérés értéke –N+1..N–1 között fut, pontosabban a j. elem értéke valójában **1–j..N–j** közötti,

F2. ezért a számrendszer **A alap**ja (a j. helyi értéken) (N–j)–(1–j)+1=N, így minden helyi értéken ugyanaz az **N** lehet.

F3. Így kapjuk a keresett összefüggést a PermIndex függvényre, amelyet **(A2Dec1)**-ből kiindulva **1**-gyel megnövelünk, hogy az értékkészlet alsóhatárára **(PI1)**-ben megfogalmazott elvárást teljesítsük:

(PI3) $$PermIndex((P_1,P_2,\ldots,P_{N-1},P_N)):=\sum_{j=1}^{N}(P_{N-j+1}-N+j-1)*N^{j-1}\ \ \mathbf{+1}$$

Elemezzük a formulát! Belátható, hogy a **(PI3)** formula

F3. „rendezéstartó” abban az értelemben, hogy
P<Q∈Perm → PemIndex(P)<PemIndex(Q),
ahol a < lexikografikus rendezés;

F5. **injektív**, azaz
$\forall P \neq Q \in Perm \rightarrow PemIndex(P) \neq PemIndex(Q)$; de

F6. **nem szürjektív**,
ui. a Perm-beli elemeket nem „hézag nélkül” képezi le az $\mathcal{R}_{PermIndex}$-ra:
észrevehetjük, hogy nemcsak permutációkhoz rendel index értékeket, hanem
pl. az alábbiakhoz is:

$$PermIndex((1,2,\ldots,N,N)) = (N-N+1-1)*N^0+(N-N+2-1)*N^1+\sum_{j=3}^{N}(N-j+1-N+j-1)*N^{j-1} =$$

$$=0*N^0+1*N^1+\sum_{j=2}^{N}0*N^{j-1}+1=N+1$$

$$PermIndex((2,2,\ldots,N)) = (2-1)*N^{N-1}+\sum_{j=1}^{N-1}(N-j+1-N+j-1)*N^{j-1}+1=N^{N-1}+1$$

Érdemes látni, hogy minden permutációra igaz, hogy ha a PermIndex-beli tagokban szereplő N^{j-1} szorzótényezőit összeadjuk, akkor 0-t kapunk; vagyis

$$\sum_{j=1}^{N}P_{N-j+1}-N+j-1=\sum_{j=N}^{1}P_j-j=0 \quad \text{és –persze–} \quad P_j \in [1-j..N-j].$$

Más példa: a fordított rendezettségű (N,N–1,…,1) permutációhoz tartozó perm-index:

$$PermIndex((N,N-1,\ldots,1)) = \sum_{j=1}^{N}(j-N+j-1)*N^{j-1}+1=\sum_{j=1}^{N}(2*j-N-1)*N^{j-1}+1=$$

$$=2*\sum_{j=1}^{N}j*N^{j-1}-\sum_{j=1}^{N}(N+1)*N^{j-1}+1,$$

amiről belátható, hogy >N!

Lássuk be! Két állításból következik: az előbbi F3-ban megfogalmazott „rendezéstartó”-ságból, valamint az alábbiból:

Állítás: $N \geq 3$ esetén $PermIndex((2,P_2,\ldots,P_N)) > N!$ $\forall (2,P_2,\ldots,P_N) \in Perm$

Bizonyítás:

$$PermIndex((2,P_2,\ldots,P_N)) = \sum_{j=1}^{N-1}(P_{N-j+1}-N+j-1)*N^{j-1}+2*N^{N-1}$$

Ennek legkisebb értéke annál a permutációnál van (F3 miatt), amelyben $P_2=1$, $P_3=3$, …, $P_N=N$, vagyis az első permutáció, ami 2-vel kezdődik (magyarán amelyben a P_1 és a P_2-ben csere van ($1 \leftrightarrow 2$)). Tehát

$$PermIndex((2,P_2,\ldots,P_N)) \geq PermIndex((2,1,3,\ldots,N)) =$$

$$=\sum_{j=1}^{N-2}(P_{N-j+1}-N+j-1)*N^{j-1}-1*N^{N-2}+2*N^{N-1}=$$

$$=\sum_{j=1}^{N-2}0*N^{j-1}-N^{N-2}+2*N^{N-1}=(-N^{N-2}+N^{N-1})+N^{N-1}>N^{N-1}>N!$$

Az utóbbi egyenlőtlenség belátása teljes indukcióval gyerekkáték.

F7. azaz $\exists pI \in [1..N!]$: $PermIndex(X)=pI$ és $X \notin Perm$ (a kívánt értékkészlet értékei közt van olyan, amely nem permutációhoz tartozik).

F8. Tehát **nem bijekció** az 1..N! halmazra nézve!

Rendezés-alapú ötlet

Az előbbi PermIndex definíció hibája, hogy nemcsak a permutációkhoz rendel folytonosan értékeket, s ezért illesztődik be „oda nem való” értékek is az 1..N! közé. Ezen próbálunk most segíteni.

Megtartjuk a „számrendszeres” ötletet, de feladjuk a „hagyományos” „egy-alapúság” feltevését. Úgy próbáljuk folytonosan feltölteni a kívánt értékkészletet, hogy olyan számrendszerre térünk át, amely egyes jegyeihez nem A^i szorzótényező tartozik, hanem i! . A hozzátartozó „számjegy” persze már csak 0..i–1 között lehet.

Emlékezzünk: egy $s_N...s_2s_1$ számjegysorozattal adott faktoriális-alapú számrendszer[1] decimális értékét az alábbi formula határozza meg:

(Fakt2Dec1) $\text{Fakt2Dec}(s_N...s_2s_1) := 1 + \sum_{i=1}^{N} s_i * i!$

Elvárás a szemjegyekkel szemben:

(Fakt2Dec2) $s_i \in [0..i-1] \ \forall i \in [1...N]$.

Másrészről azt a „logikus” elképzelést is meg szeretnénk tartani, hogy a permutációhoz rendelt indexérték fejezze ki a viszonyítási alaptól (az **(Ö1)**-ben definiált P1-től) való eltérést.

A két elképzelés együtt tartásában az a nehézség, hogy a permutáció i. jegyéhez rendelhető eltérésértéknek 0..N–i közöttinek kell lennie. Ugyanis így használható fel a faktoriális számrendszerben felírt szám (elölről sorszámozott) i. jegyeként.

Ez bajosnak látszik, hiszen a fordítottan rendezett permutáció esetében az eltérés pl. az utolsó elem esetén N–1, ami sokkal nagyobb a kívánatosnál.

Ezért a következőt tesszük:

(Ö3) választunk egy rendezési algoritmust (pl. a minimum kiválasztásosét),

(Ö4) az i. elemet „helyre tevő” ciklusléptésben a cserében szereplő elemek indexkülönbségét tekintjük eltérésnek.

Folyományok:

F1. A rendezés során a rendezendő $P \neq Q \in Perm$ permutációkra különbözőek lesznek a cserékben szereplő indexpárok, tehát a kiinduló permutációnak egyedi „ujjlenyomatának” tekinthetők. Azaz a leképezés **injektív**.

F2. A ciklusléptésekben előrehaladva a cserében szereplő indexpárok maximális különbségei szükségképpen egyre kisebbek lesznek. Kezdetben (i=1-nél) N–1, i=2-nél N–2, az utolsó lépésnél (i=N–1-nél) már csak 1. Így teljesítjük a **(Fakt2Dec2)**-ben megfogalmazott elvárást.

Ezek után egy lehetséges megvalósítás algoritmusa, amely kódolásánál meg kell gondolni, hogy az egyes egész típusú adatokat mennyi memóriát igénylő változataikkal kell kódolni:

```
Változó
  permInd_Fakt:Szöveg
```

Ez a változó fogja tartalmazni az éppen generált permutáció indexének faktoriális számrendszerbeli alakját. Mint „mellékszereplőt” nem tüntettük föl a paraméterlistában, hanem „orvul” globálisan elérhető adatként deklaráltuk.

```
Függvény PermIndex_MKR(Változó P:TPerm;
                       Konstans N:Egész):Egész
  Változó
    i,j,mini:Egész
    pI:Egész
```

A minimum kiválasztásos rendezés (MKR) végrehajtása közben képezzük a P permutáció pI indexét.

„Balkézről” megadjuk a faktoriális számrendszerbeli alakját is.

[1] http://www.hiradastechnika.hu/data/upload/file/2005/2005_3/HT_0503-11.pdf

```
  permInd_Fakt:=''
  pI:=0
  Ciklus i=1-től N-1-ig
    mini:=i
    Ciklus j=i+1-től N-ig
      Ha P(j)<P(mini) akkor mini:=j
    Ciklus vége
    Csere(P(i),P(mini))
    pI:=pI*(N-i+1)+(mini-i)
    permInd_Fakt:=permInd_Fakt+Karakter(mini-i+48)
  Ciklus vége


  [sorszám-korrekció: 1→0]
  permInd_Fakt:=permInd_Fakt+'1'
  PermIndex_MKR:=pI+1
Függvény vége.
```

A minimum kiválasztásos rendezés szokásos algoritmusa a permutációindexet előállító betétekkel megspékelve.

Figyeljük meg, miként veszi figyelembe az adott helyi értéknél alkalmazandó számrendszer-alapot! (Horner-elrendezés)

A végső értékadás gondoskodik arról, hogy a permutációindex 1-gyel, nem 0-val kezdődjék; és a faktoriális számrendszer definíciójának (és a céljainknak) megfelelően 1-re végződjék.

Nem részletezzük a Csere eljárást.

A kódban pirossal emeltük ki a permutációindexet, zölddel a faktoriális számrendszerben felírt alakját számító részeket.

Végezetül a futás néhány eredményét összegezzük a mellékelt ábrákon.

```
N:3
1.: ( 1 2 3 ), permIndex:  1=  1 permIndex-Fakt: 001
2.: ( 1 3 2 ), permIndex:  2=  2 permIndex-Fakt: 011
3.: ( 2 1 3 ), permIndex:  3=  3 permIndex-Fakt: 101
4.: ( 2 3 1 ), permIndex:  6=  5 permIndex-Fakt: 211
5.: ( 3 1 2 ), permIndex:  4=  6 permIndex-Fakt: 111
6.: ( 3 2 1 ), permIndex:  5=  7 permIndex-Fakt: 201
```

1. ábra: A futás N=3-ra.

```
 1.: ( 1 2 3 4 ), permIndex:   1=   1 permIndex-Fakt: 0001
 2.: ( 1 2 4 3 ), permIndex:   2=   2 permIndex-Fakt: 0011
 3.: ( 1 3 2 4 ), permIndex:   3=   4 permIndex-Fakt: 0101
 4.: ( 1 3 4 2 ), permIndex:   6=   6 permIndex-Fakt: 0211
 5.: ( 1 4 2 3 ), permIndex:   4=   8 permIndex-Fakt: 0111
 6.: ( 1 4 3 2 ), permIndex:   5=   9 permIndex-Fakt: 0201
 7.: ( 2 1 3 4 ), permIndex:   7=  13 permIndex-Fakt: 1001
 8.: ( 2 1 4 3 ), permIndex:   8=  14 permIndex-Fakt: 1011
 9.: ( 2 3 1 4 ), permIndex:  15=  19 permIndex-Fakt: 2101
10.: ( 2 3 4 1 ), permIndex:  24=  22 permIndex-Fakt: 3211
11.: ( 2 4 1 3 ), permIndex:  16=  23 permIndex-Fakt: 2111
12.: ( 2 4 3 1 ), permIndex:  23=  25 permIndex-Fakt: 3201
13.: ( 3 1 2 4 ), permIndex:   9=  28 permIndex-Fakt: 1101
14.: ( 3 1 4 2 ), permIndex:  12=  30 permIndex-Fakt: 1211
15.: ( 3 2 1 4 ), permIndex:  13=  31 permIndex-Fakt: 2001
16.: ( 3 2 4 1 ), permIndex:  20=  34 permIndex-Fakt: 3011
17.: ( 3 4 1 2 ), permIndex:  17=  39 permIndex-Fakt: 2201
18.: ( 3 4 2 1 ), permIndex:  22=  40 permIndex-Fakt: 3111
19.: ( 4 1 2 3 ), permIndex:  10=  44 permIndex-Fakt: 1111
20.: ( 4 1 3 2 ), permIndex:  11=  45 permIndex-Fakt: 1201
21.: ( 4 2 1 3 ), permIndex:  14=  47 permIndex-Fakt: 2011
22.: ( 4 2 3 1 ), permIndex:  19=  49 permIndex-Fakt: 3001
23.: ( 4 3 1 2 ), permIndex:  18=  51 permIndex-Fakt: 2211
24.: ( 4 3 2 1 ), permIndex:  21=  52 permIndex-Fakt: 3101
```

2. ábra: A futás N=4-re.

Magyarázat az ábrákhoz: Az összes permutáció előállításához felhasználtuk az előző részben megtervezett backtrack-alapú algoritmust. Ez magyarázza a permutációk sorrendjét. Az egyes permutációk mellett megjelenik a permutációindex (permIndex: folytonosan tölti ki az 1..N! tartományt); az előző részben ecsetelt P1-hez viszonyított eltérések alapján számított érték (az egyenlőség jel mögött: jól látszik az alkalmatlansága); valamint a permutációindex faktoriális-alapú számrendszerben felírt alakja (permIndex-Fakt után).

F3. **Nem „rendezéstartó”** abban az értelemben, hogy
P<Q∈Perm → PemIndex(P)<PemIndex(Q),
ahol a < lexikografikus rendezés.
Lásd pl. a 4. és 5.-et: permIndex((1342))=6>4=permIndex((1423)), pedig (1342)<(1423).

F4. Minden index 1..N! közötti és nincs ismétlődő. Ezekből következően 1..N! közötti minden szám előfordul permutációindexként. Azaz a leképezés **szürjektív**.

F5. Tehát **bijekció** 1..N! tartományra vonatkozóan.

Az i. permutáció előállítása

Elérkeztünk a kitűzött feladathoz: az i. permutáció generálása indexe alapján. Feltéve tehát, hogy a permutációkat az előző részben definiált módon értelmezzük, állítsuk elő egy 1..N! között értékhez az annyiadik permutációt.

Megoldás

Az előbb alkalmazott számjegysorozattá alakítást kell most csupán „visszafelé” végrehajtanunk. Ez a következő algoritmussal végezhető. Az N (=a permutáció rendje), a P1 (=a rendezett permutáció) globálisan elérhető paraméterek.

```
Függvény Index2Perm(Konstans pI:Egész):TPerm
  Változó
    i,j,k,t:Egész
    x:TPerm
  x:=P1; i:=pI-1 [sorszám-korrekció: 1→0]
  k:=2
  Ciklus j=N-1-től 1-ig -1-esével
    t:=i Mod k; i:=i Div k; Csere(x(j),x(j+t))
    k:+1
  Ciklus vége
  Index2Perm:=x
Függvény vége.
```

Megdöbbentően egyszerűen sikerült! Különösen, ha figyelembe vesszük, hogy az korábbi megfontolások algoritmusai lényegében „feleslegesek”, hiszen csak a helyesség belátását szolgálták.

Feladatok

1. feladat

Jól ismert fejtörő, amelyben egy aritmetikai művelet kapcsol egybe szavakat. A feladat az, hogy a szavak egyes betűinek feletessünk meg egy számjegyet úgy, hogy a művelet helyes eredményt szolgáltasson a szavakon. Pl. SEND + MORE = MONEY.

Megoldási ötlet

A szavakban –összességben– előforduló jelekhez (SENDMORY) keressük 0..9 számjegyek egyértelmű hozzárendelését. A permutációkon haladunk végig, amíg megoldást nem találunk. A hatékonyság növelése érdekében a konkrét betűegyenlet alapján könnyen ellenőrizhető feltételeket állítunk föl.

Az eredeti egyenlet helyett az alábbi megoldását keressük:

(*) SEND + MORE – MONEY = 0

Természetesen most is számjegyesen. Az egyenletből leolvashatjuk a legfontosabb összefüggéseket. Íme:

„S” ↔ s	s*1000	„M” ↔ m	m*(1000-10000)=m*(-9000)
„E” ↔ e	e*(100+1-10)=e*91	„O” ↔ o	o*(100-1000)=o*(-900)
„N” ↔ n	n*(10-100)=n*(-90)	„R” ↔ r	r*(10)=r*(10)
„D” ↔ d	d*(1)	„Y” ↔ y	y*(-1)

Az egyes jelekhez rendelt számot behelyettesítve kapjuk a **(*)**-beli egyenlet bal oldalát, ha az 0, akkor találtunk jó hozzárendelést.

ISMÉTLÉSES VARIÁCIÓ

Adott egy N elemű sorozat. Adjuk meg az i. olyan sorozatot, amely elemei a bemeneti sorozat elemei, és pontosan K hosszúságú! A sorozat elemei legyenek –a kialakult „hagyománynak” megfelelően– 1..N között természetes számok!

A variációk indexelése

Az „ismétléses”-ség kifejezett szerencse forrása, hiszen ez azt jelenti, hogy minden generálandó sorozat minden eleme gyanánt az összes bemeneti elem (1..N) előfordulhat. Így tekinthetjük a sorozatokat egy N-alapú számrendszerben felírt K-jegyű számnak: a $v_K v_{K-1}..v_1$=11..1_N-től a $w_K w_{K-1}..w_1$=NN..N_N-ig.

A fenti K-jegyű számokat decimálissá alakítva a természetes számok egy folytonos intervallumát kapjuk. Az indexeléssel szembeni szokásos elvárás (ti., hogy 1-gyel kezdődjék) kielégítése érdekében egy konstans értékkel toljuk „helyre”:

(IVI1) IsmVarIndex: $\mathbb{N}^* \times \mathbb{N} \times \mathbb{N} \rightarrow [1..N^K]$;

(IVI2) IsmVarIndex **bijekció** az N-elemű K-hosszúságú sorozatok halmaza és a $[1..N^K]$ között;

(IVI3) $\text{IsmVarIndex}((V_N,V_{N-1},\ldots,V_2,V_1),N,K) := \sum_{j=1}^{K} V_j * N^{j-1} - v$, ahol $v := \sum_{j=1}^{K} 1 * N^{j-1} - 1 = (N^K-1)/(N-1)-1$

Az i. variáció előállítása

Az előbbi inverzét kell megvalósítanunk.

```
Konstans
  MaxN:Egész(?)
Típus
  TIsmVar=Tömb(1..MaxN:Egész)
Függvény Index2IsmVar(Konstans i,N,K:Egész):TIsmVar
  Változó
    ii,j:Egész
    x:TIsmVar
  ii:=i-1  [korrekció az 1-gyel kezdődő sorszámozás miatt]
  Ciklus j=K-tól 1-ig -1-esével
    x(j):=(ii Mod N)+1; ii:=ii Div N
  Ciklus vége
  Index2IsmVar:=x
Függvény vége.
```

Feladatok

1. feladat

Adjon algoritmust N elem K-ad rendű ismétlés nélküli variáció i-dikjének az előállítására!