Matematikai és valószínűségszámítási ismeretek

2. tétel

1. rész

Valószínűség fogalma és kiszámításának kombinatorikus módszerei (permutációk, variációk, kombinációk). Feltételes valószínűség, függetlenség, Bayes-formula.

Valószínűség

Elemi események: egy kísérlet lehetséges kimenetelei

Esemény: elemi eseményekből álló halmazok, jele: A, B, C …

Eseménytér: egy kísérlethez tartozó összes elemi esemény, jele: **Ω**

Valószínűség: tekintsünk egy kísérletet, és ehhez kapcsolódva egy **A** eseményt. Hajtsuk végre a kísérletet *n*-szer egymástól függetlenül, azonos körülmények között. Jelölje k_A az **A** bekövetkezései számát. Ha a k_A/n relatív gyakoriság nagy *n* esetén egy fix szám körül ingadozik, akkor ezt az **A** -ra jellemző számot **P(A)**-val jelöljük és **A** valószínűségének nevezzük.

Axiómák:

1. $\boxed{P(A) \geq 0 \text{ minden } A \text{ eseményre.}}$
2. $\boxed{P(\Omega) = 1.}$ (A biztos esemény mindig bekövetkezik)
3. $\boxed{P(A+B) = P(A) + P(B)}$ (Ha A és B egymást kizáró események)

Klasszikus kiszámítási módja:

$$\boxed{P(A) = \frac{k}{N} = \frac{\text{kedvező esetek száma}}{\text{összes esetek száma}}}$$

Kombinatorikus kiszámítási módszerek

<u>Permutáció:</u> egy A halmaz önmagára vett bijektív leképezése, vagy A elemeinek valamilyen sorrendben való felsorolása (= sorba rendezés)

- ismétlés nélküli

Tétel

n különböző elem lehetséges sorbarendezéseinek a száma $P_n = n!$.

- ismétléses

Tétel

Ha n elemünk van k különböző fajtából, az 1. fajtából ℓ_1, a 2.-ból ℓ_2, stb. (azaz $\ell_1 + \ell_2 + \cdots + \ell_k = n$), akkor az n elem lehetséges sorrendjeinek a száma

$$P_n^{\ell_1,\ldots,\ell_k} = \frac{n!}{\ell_1!\ldots\ell_k!}$$

Variáció: n elemű halmazból kiválasztott k hosszúságú sorozatok (= kiválasztás és sorba rendezés)

- ismétlés nélküli

Definíció és tétel

Egy n elemű halmaz k-ad osztályú ismétlés nélküli variációi alatt a halmaz elemeiből kiválasztott k hosszúságú sorozatokat értjük. Ezek száma:

$$V_{n,k} = \frac{n!}{(n-k)!} = n\cdot(n-1)\ldots(n-k+1).$$

Itt szükségképpen $n \geq k$.

- ismétléses

Definíció és tétel

Egy n elemű halmaz k-ad osztályú ismétléses variációi alatt a halmaz elemeiből *visszatevéssel* kiválasztott k hosszúságú sorozatokat értjük. Ezek száma:

$$V_{n,k}^i = n^k.$$

Kombináció: n elemű halmaz k elemű részhalmazai (= kiválasztás)

- ismétlés nélküli

Definíció és tétel

Egy n elemű halmaz k elemű részhalmazait a halmaz k-ad osztályú ismétlés nélküli kombinációinak nevezzük. Számuk:

$$C_{n,k} = \frac{n!}{k!(n-k)!} =: \binom{n}{k}.$$

Definíció szerint $0! = 1$.
Itt szükségképpen $n \geq k$.

- ismétléses

Definíció és tétel

Ha egy n elemű halmaz elemeiből úgy képezünk k elemű halmazt, hogy egy elemet többször is választhatunk (azaz visszatevéssel), akkor az n elem k-ad osztályú ismétléses kombinációjáról beszélünk. Számuk:

$$C_{n,k}^i = \left(\!\binom{n}{k}\!\right) = \binom{n+k-1}{k}.$$

Feltételes valószínűség

Legyen A és B esemény, P(B) > 0. Ekkor az A esemény B-re vonatkozó feltételes valószínűségén a mennyiséget értjük.

$$\boxed{P(A|B) = P(AB)/P(B)}$$

Függetlenség

Azt mondjuk, hogy A és B független események, ha

Jelentése: Egyik esemény bekövetkezése sem befolyásolja a másik bekövetkezési esélyét.

$$P(AB) = P(A)P(B)$$

Bayes-formula

A Bayes-tétel a valószínűségszámításban egy feltételes valószínűség és a fordítottja között állít fel kapcsolatot. A valamiféle hipotézis, B egy megfigyelhető esemény és a tétel azt adja meg, hogyan erősíti vagy gyengíti az esemény megfigyelése a hipotézis helyességébe vetett hitünket.

A formula: $\boxed{P(B|A) = P(A|B)P(B)/P(A)}$

Legyen A egy esemény, B_1, B_2, … teljes eseményrendszer, P(A) > 0, $P(B_i) > 0$, i = 1,2.... Ekkor

$$\boxed{P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{j=1}^{\infty} P(A|B_j)P(B_j)}}$$

minden j-re.

2. rész

Algoritmusok lépésszáma: beszúrásos rendezés, összefésüléses rendezés, keresések lineáris és logaritmikus lépésszámmal. Gyorsrendezés, az összehasonlítások minimális száma. Rendezés lineáris lépésszámmal: radix rendezés, vödör rendezés.

Beszúrásos rendezés

A rendezés lényege: a tömb második elemétől indulva lépked végig az elemeken, ellenőrizzük, hogy az adott elem kisebb-e az előtte lévő elemnél. Ha kisebb, akkor egyesével addig léptetjük a tömbben az elemet, amíg előtte kisebb, utána nagyobb szám lesz. Ha nagyobb, akkor nem történik helycsere. Nagy tömbök esetén nem hatékony, viszont kis tömböknél a leghatékonyabb

Lépésszám, időbonyolultság:

- Legrosszabb eset: $O(n^2)$ ha pont fordítva van rendezve a kiindulási tömb
- Legjobb eset: $O(n)$ ha a kiindulási tömb eleve rendezve van

```
procedure BESZÚRÁSOS_RENDEZ(A)
 - - beszúrásos rendezés

for i ← 2 to méret(A) do
  kulcs ← A[i]
  j ← i - 1
  while j ≥ 1 és A[j] > kulcs do
    A[j + 1] ← A[j]
    j ← j - 1
  end while
  A[j + 1] ← kulcs
end for

end procedure
```

Összefésüléses rendezés

A rendezés lényege: a tömböt felosztjuk két részre, a részeket külön rendezzük, majd összefésüljük. Ez rekurzívan történik, tehát egészen addig osztjuk 2 részre a résztömböket, amíg egy elemű tömbök maradnak. Ezeket kell párosával összefésülni. Ennek lényege, hogy a két résztömb soron következő elemeit hasonlítja össze, így készítve egy új összefésült tömböt. Ezt egészen addig ismételve, míg az eredeti tömbünk rendezett változatát kapjuk vissza. Példa:

```
 5   2   4   6   1   3   2   6
[5   2   4   6] [1   3   2   6]
[5   2] [4   6] [1   3] [2   6]
[5] [2] [4] [6] [1] [3] [2] [6]
[2   5] [4   6] [1   3] [2   6]
[2   4   5   6] [1   2   3   6]
 1   2   2   3   4   5   6   6
```

Lépésszám, időbonyolultság: O(n * log(n)) ahol log(n) a felosztások/szintek száma; bizonyos helyzetekben gyorsabb is lehet, mint a gyorsrendezés, viszont hátránya a magas tárterület igénye a felosztások miatt (nem helyben rendez)

```
procedure ÖSSZEFÉSÜLVE_RENDEZ(A)
 - - egy vektor összefésülésen alapuló rendezése

rendezett ← 1
méret(B) ← méret(A)
while rendezett < méret(A) do
  ÖSSZEFÉSÜL_1(A, B, rendezett)
  ÖSSZEFÉSÜL_1(B, A, rendezett)
end while

end procedure
```

```
procedure ÖSSZEFÉSÜL_1(A, B, rendezett)
 - - egy vektor összefésülésen alapuló rendezésének egy fázisa

k ← 1
repeat
  i ← k
  j ← a ← k + rendezett
  b ← a + rendezett
  if a > méret(A) then
    a ← méret(A) + 1
  end if
  if b > méret(A) then
    b ← méret(A) + 1
  end if
  while i < a és j < b do
    if A[i] > A[j] then
      B[k] ← A[j]
      j ← j + 1
    else
      B[k] ← A[i]
      i ← i + 1
    end if
    k ← k + 1
  end while
  while i < a do
    B[k] ← A[i]
    i ← i + 1
    k ← k + 1
  end while
  while j < b do
    B[k] ← A[j]
    j ← j + 1
    k ← k + 1
  end while
until k > méret(A)
rendezett ← rendezett + rendezett

end procedure
```

Keresések

Lineáris keresés: a tömb elemeinek iterálása az elejétől egészen a keresett elem megtalálásáig

Rendezetlen tömbön is működik.

```
function TELJES_KERES1(A, érték)
 - - teljes keresés while ciklussal

i ← 1
while i ≤ méret(A) és A[i] ≠ érték do
  i ← i + 1
end while
if i > méret(A) then
  KIVÉTEL "nincs ilyen értékű elem"
else
  return i
end if

end function
```

```
function TELJES_KERES2(A, érték)
 - - teljes keresés for ciklussal

for i ← 1 to méret(A) do
  if A[i] = érték then
    return i
  end if
end for
KIVÉTEL "nincs ilyen értékű elem"

end function
```

```
function TELJES_KERES_REK(A, érték)
 - - teljes keresés rekurzívan

if méret(A) = 0 then
  KIVÉTEL "nincs ilyen értékű elem"
else if A[1] = érték then
  return 1
else
  return TELJES_KERES_REK(A[2..méret(A)], érték) + 1
end if

end function
```

Lépésszám, időbonyolultság: O(n) ezért is hívják lineáris keresésnek, mert a lépészám lineárisan függ a tömb elemszámától

Bináris keresés: csak rendezett tömbön! Megvizsgálja a középső elemet, ha nem az a keresett, akkor, ha annál nagyobb, akkor a középső elem utáni résztömbben keres, ha kisebb, akkor a középső elem előtti résztömbben, ugyanilyen elven.

```
function BINÁRIS_KERES1(A, érték)
 - - bináris keresés iteratívan

alsó ← 1
felső ← méret(A)
while alsó ≤ felső do
  középső ← ⌊(alsó + felső) / 2⌋
  if A[középső] = érték then
    return középső
  else if A[középső] > érték then
    felső ← középső - 1
  else
    alsó ← középső + 1
  end if
end while
KIVÉTEL "nincs ilyen értékű elem"

end function
```

```
function BINÁRIS_KERES2(A, érték)
 - - bináris keresés rekurzívan

if méret(A) = 0 then
  KIVÉTEL "nincs ilyen értékű elem"
end if
középső ← ⌊(1 + méret(A)) / 2⌋
if A[középső] = érték then
  return középső
else if A[középső] > érték then
  return BINÁRIS_KERES2(A[1..középső - 1], érték)
else
  return középső + BINÁRIS_KERES2(A[középső + 1..méret(A)], érték)
end if

end function
```

```
function BINÁRIS_KERES3(A, érték, alsó, felső)
 - - bináris keresés rekurzívan, részvektorok nélkül

if alsó > felső then
  KIVÉTEL "nincs ilyen értékű elem"
end if
középső ← ⌊(alsó + felső) / 2⌋
if A[középső] = érték then
  return középső
else if A[középső] > érték then
  return BINÁRIS_KERES3(A, érték, alsó, középső - 1)
else
  return BINÁRIS_KERES3(A, érték, középső + 1, felső)
end if

end function
```

Lépésszám, időbonyolultság: O(log(n)) nagy elemszámú tömbök esetén lényegesen gyorsabb lehet, mint a lineáris

Gyorsrendezés

A rendezés lényege: Kiválasztunk egy kitüntetett elemet (pivot), majd a tőle kisebb vagy egyenlő elemeket tőle jobbra, a nagyobb elemeket tőle balra helyezzük el. Ekkor a pivot elem a végleges sorrendet tekintve a helyén van. Ezt követően a pivot előtti és utána résztömbön is elvégezzük ezt az eljárást (rekurzív).

```
procedure GYORS_RENDEZ2(A, alsó, felső)
 - - gyorsrendezés, 2. változat

1. if alsó < felső then
2.    határ ← FELOSZT(A, alsó, felső)
3.    GYORS_RENDEZ2(A, alsó, határ - 1)
4.    GYORS_RENDEZ2(A, határ + 1, felső)
5. end if

end procedure
```

```
function FELOSZT(A, alsó, felső)

1. kulcs ← A[felső]
2. i ← alsó - 1
3. for j ← alsó to felső - 1 do
4.   if A[j] ≤ kulcs then
5.     i ← i + 1
6.     A[i] és A[j] felcserélése
7.   end if
8. end for
9. A[i + 1] és A[felső] felcserélése
10. return i + 1

end function
```

Itt a kitüntetett elem a tömb utolsó eleme. Az ’alsó’ és a ’felső’ változók a paraméterként adott tömb első és utolsó indexei. Ezen eljárás során a tömb elejére kerülnek a pivotnál kisebb vagy egyenlő elemek. a ’FELOSZT’ eljárás 9. sorában az utolsó helyen álló pivotot cseréljük fel a sorrendben első pivotnál nagyobb elemmel. Ennek a helyére került pivotnak az indexét kapja vissza a ’GYORS_RENDEZ’ eljárás, majd az rekurzívan hívja magát a pivot előtt és utáni résztömbökre.

(Kicsit másik módszer, de részletes, vizuális magyarázattal: https://www.youtube.com/watch?v=Hoixgm4-P4M)

Bonyolultság:

- legrosszabb eset: $O(n^2)$ például akkor, ha a pivot mindig a legnagyobb eleme a tömbnek
- legjobb eset: $O(n * \log(n))$ egyenlő, vagy közel egyenlő felosztás esetén

Radix(számjegyes) rendezés

Feltételezzük, hogy a rendezni kívánt tömbünk minden eleme ugyanannyi számjegyből áll, majd a legkisebb helyiértéktől haladva a legnagyobb felé, helyiértékenként rendezzük a tömböt egy választott stabil algoritmussal (pl.: leszámláló rendezés).

Legyenek a számaink

4 9 13 15 5 2 10 7 1 8

Ezek kettes számrendszerben a következőek:

0100 1001 1101 1111 0101 0010 1010 0111 0001 1000

Az utolsó bit szerint szétválasztva az előbbi listát a következőt kapjuk:

0100 0010 1010 1000 1001 1101 1111 0101 0111 0001

A harmadik bit szerint

0100 1000 1001 1101 0101 0001 0010 1010 1111 0111

A második bit szerint

1000 1001 0001 0010 1010 0100 1101 0101 1111 0111

S végül az első bit szerint rendezve

0001 0010 0100 0101 0111 1000 1001 1010 1101 1111

Amelyek tízes számrendszerben

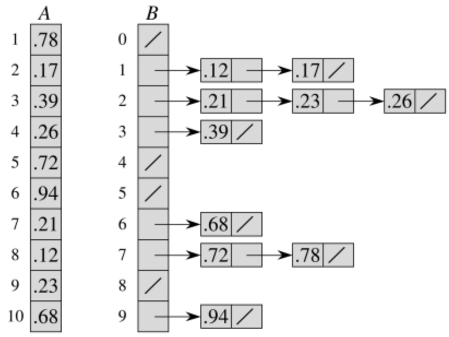
1 2 4 5 7 8 9 10 13 15

tehát valóban rendeztük a sorozatot.

Bonyolultság: $O(d * (n + k))$, ahol d a számjegyek száma, n a rendezni kívánt elemek száma, k pedig a lehetséges számjegyek száma (lineáris idejű)

Vödör(edény) rendezés

Feltételezzük, hogy a rendezni kívánt n értékekre igaz, hogy: $0 \leq n < 1$ és az értékek egyenletes eloszlásból származnak. A vödrök láncolt listák lesznek. Ezekben helyezzük el az elemeket az első tizedes jegy alapján, majd az egyes vödrökben beszúrásos rendezéssel rendezzük az



(a) (b)

elemeket. Az eljárás végén pedig összefűzzük a rendezett vödrök tartalmát.

```
procedure EDÉNY_RENDEZÉS(A)
  n ← méret(A)
  for i ← 1 to n do
      szúrjuk be az A[i] elemet a B[⌊nA[i]⌋] listába.
  end for
  for i ← 0 to n-1 do
      rendezzük a B[i] listát beszúrásos rendezéssel.
  end for
  sorban összefűzzük a B[0], B[1], . . ., B[n – 1] listákat.
end procedure
```

Bonyolultság: a bonyolultság a vödrök rendezési algoritmusától függ

- Legrosszabb eset: $O(n^2)$ ha vödrök elemszáma nagyban eltér, és a szétválogatást követően a vödrökben lévő elemek fordított sorrendben vannak
- Legjobb eset: $O(n)$ egyenletes eloszlású számokkal, és ha a szétválogatást követően már eleve rendezett vödröket kapunk

Összességében a vödör rendezés lineáris, egészen addig, amíg az edényméretek négyzeteinek összege lineáris a teljes elemszámban.