Klasifikacija uređenog izbora elemenata

Ponekad je poredak u kojem se elementi nalaze bitan, a ponekad nije. Naprimjer, riječ ROMAN je različita od riječi ORMAN, bez obzira što su obe riječi formirane od slova iz skupa {M,O,R,A,N}. Međutim, zbir brojeva 1 + 2 + 3 je isti kao zbir 2+3+1, bez obzira što je redoslijed ovih brojeva promijenjen.

Treba naučiti razliku:

- Kada je uređenost elemenata bitna?
- Gdje se elementi nalaze (skup, multiskup), tj. Da li je dozvoljeno ponavljanje?

Jasno je da postoje 4 kategorije:

1. Uređenje *m* elemenata iz multiskupa od *n* elemenata:
    a. m- permutacije multiskupa (m <= n) - varijacije s ponavljanjem
    b. m- permutacije multiskupa (m = n) - permutacije s ponavljanjem

2. Uređenje m elemenata iz skupa od n elemenata:
    a. m- permutacije skupa (m <= n) - varijacije bez ponavljanja
    b. m- permutacije skupa (m = n) - permutacije bez ponavljanja

M-permutacija multiskupa od n elemenata(varijacije sa ponavljanjem)

Neka je dat skup $B = \{b_1,..., b_l\}$ sa $l \geq 1$ elemenata i neka je
$M = [\, b_1,., b_l \,]_{m,...m}$
multiskup u kojem se svaki element iz B pojavljuje tano m puta.
Broj elemenata u multiskupu M je $n = m \bullet l$.
Ako izaberemo m-točlani podmultiskup od M i uredimo ga dobijamo jednu m-permutaciju multiskupa.

Primjer1 : Koliko postoji različitih riječi sa 5 slova (koristeći naše pismo sa 30 slova i uključujući i besmislene riječi kao kćndv)?

Rješenje. Pošto se svako od 5 slova može nezavisno izabrati na 30 načina, nije teško vidjeti da je odgovor 30^5

I, zaista, svaka riječ sa 5 slova se može posmatrati kao preslikavanje skupa {1, 2, . . . , 5} u skup slova {a,b,c,. . . ,ž}: za svako od 5 mjesta u riječi, sa rednim brojevima 1, 2, . . . , 5, preslikavanje odredjuje slovo na tom mjestu.
Uredjeni izbor n elemenata sa ponavljanjem iz skupa M sa m elemenata
u stvari odgovara preslikavanju skupa {1, 2, . . . , n} u skup M.

Definicija1 : m-permutacija elemenata multiskupa $M = [\, b_1,., b_l \,]_{m,...m}$ je bilo koja uredena m-torka elemenata iz M, ti. bilo koja uredena m-torka u kojoj je svaka komponenta element iz skupa B.

Broj m-permutacija multiskupa $M = [\, b_1,., b_l \,]_{m,...m}$ oznacavacemo sa $\bar{P}(l; m)$.

Jasno je da je broj nacina da se formiraju *m*-torke elemenata sa sobinom da je svaka komponenta element skupa *B* jednak broju elemenata u B^m.

Teorema1 : Broj *m*-permutacija elemenata multiskupa $M = [\, b_1, ., b_l \,]_{m,...m}$ elemenata jednak ie I^m,

$\bar{P}(l;m) = I^m$

Tj. Neka je *N* skup sa *n* elemenata (koji može da bude i prazan, tj. n>0) i neka je *M* skup sa m elemenata, *m* > 1. Broj svih mogućih preslikavanja $f : N \rightarrow M$ jednak je m^n.

Dokaz1 : Za dokaz koristimo metod matematičke indukcije po n.
Šta teorema tvrdi za n = 0? U ovom slučaju, posmatramo sva preslikavanja f skupa $N = \emptyset$ u neki skup M. Definicija preslikavanja nam kaže da takvo f mora da bude skup uredjenih parova (x, y) tako da $x \in N = \emptyset$ i $y \in M$. Pošto prazan skup nema elemenata, f ne može da sadrži nijedan takav uredjeni par, pa je jedina mogućnost da je $f = \emptyset$ (nema uredjenih parova). S druge strane, $f = \emptyset$ zadovoljava definiciju preslikavanja. Prema tome, postoji tačno jedno preslikavanje $f : \emptyset \rightarrow M$. Ovo se slaže sa formulom, jer je $m^0 = 1$ za m > 1, pa zaključujemo da smo provjerili slučaj n = 0 kao bazu indukcije.

Dalje, pretpostavimo da je teorema dokazana za sve $n <= n_0$ i sve m>1. Neka je $n = n_0 + 1$ i posmatrajmo skup N sa n elemenata i skup M sa m elemenata. Izaberimo proizvoljan element $a \in N$. Za opis preslikavanja $f : N \rightarrow M$ potrebno je znati vrednost $f(a) \in M$ i preslikavanje
$f_0 : N \setminus \{a\} \rightarrow M$.
Vrednost f(a) može da se izabere na m načina, a za izbor f0 imamo m^{n-1} mogćnosti po induktivnoj hipotezi, pa sada po principu proizvoda dobijamo da je ukupan

broj mogućnosti za f jednak $m \cdot m^{n-1} = m^n$.

m – permutacije skupa od n elemenata

(varijacije bez ponavljanja)

- m-permutacija skupa od n elemenata je bilo koji uređeni podskup koji sadrži tačno m elemenata iz tog skupa. Ključna stvar jeste da je redosled elemenata bitan, kao i da ponavljanja istih nisu moguća.
- Broj m-permutacija skupa sa n elemenata označava se sa P (n, m). Možemo naći P (n, m) koristeći pravilo proizvoda:

Primer 1.

Neka je S = {A, B, C}. Tražimo sve 2-permutacije (tj. m=2). Moguće varijacije su: (A,B), (A,C), (B,A), (B,C), (C,A), (C,B). Prema tome, postoji šest 2-permutacija ovog skupa sa tri elementa. Uvek postoji šest 2-permutacija skupa sa tri elementa. Postoji samo tri načina izbora prvog elementa rasporeda. Postoje dva načina da se izabere drugi element jer se mora razlikovati od prvog elementa. Dakle, po pravilu proizvoda vidimo da je P (3, 2) = 3 · 2 = 6. Znači, koristimo ovo pravilo da pronađemo formulu za P (n, m) kad god su n i m pozitivni celi brojevi i to 1 ≤ m ≤ n.

Teorema 1

Ako je n pozitivan ceo broj i m ceo broj sa 1 ≤ m ≤ n, onda postoji
P (n, m) = n(n − 1)(n − 2)⋯(n − m + 1) m-permutacija skupa

Dokaz:

Prvi element permutacije se može izabrati na n načina jer u skupu ima n elemenata. Postoji n – 1 načina za izbor drugog elementa permutacije, jer je ostalo n − 1 elemenata izbora za tu poziciju i tako redom, sve dok ne postoji tačno n − (m − 1) = n − m + 1 načina da se izaberite m-ti element. Shodno tome, po pravilu proizvoda postoje n(n − 1)(n − 2)⋯(n – m + 1) m-permutacije skupa.

Odavde sledi posledica:

Posledica 1

Ako su n i m celi brojevi 0 ≤ m ≤ n, onda je P (n, m) = n! / (n − r)!

Dokaz:

Kada su n i m celi brojevi 1≤m≤n, prema Teoremi 1 dobijamo:

$$P(n,m) = n(n-1)(n-2)\dots(n-m+1) = \frac{n!}{(n-m)!}$$

S obzirom da je n!/(n – 0)! *= n!/n! = 1 za bilo koje nenegativno n, gorenavedena formula takođe važi i kad je m = 0.

1. Na koliko načina postoji izbor dobitnika prve nagrade, dobitnika druge i treće nagrade pobednik od 100 različitih ljudi koji su učestvovali u takmičenju?

   (rešenje: P (100, 3) = 100 · 99 · 98 = 970200

Permutacije skupa od n elemenata

Permutacija skupa od n elemenata je specijalan slučaj m-permutacije kada je m=n, što znači da biramo i uređujemo svih n elemenata. Permutacija se odnosi na bilo koji način uređenja svih elemenata jednog skupa.

Definicija:

Bijektivno preslikavanje konačnog skupa A u samog sebe je permutacija skupa A.

Definicija:

Permutacija skupa $X_n = \{x_1, x_2, \dots, x_n\}$ je proizvoljna uređena n–torka različitih elemenata iz tog skupa.

Primjer:

Neka je skup $X_4 = \{a, b, c, d\}$. Tada je (b, d, c, a) jedna permutacija tog skupa i nju ćemo jos označavati i sa bdca, dok (a, d, b, a) i (a, b, c) nisu permutacije skupa X_4 jer moraju da imaju sve elemente različite i da su uređene četvorke.

Broj permutacija skupa A od *n* elemenata jednak je:

$$P(n) = n(n-1) \cdot \dots \cdot 2 \cdot 1 = n!$$

Primjer 2:

Za skup $\{a, b, c\}$ ukupan broj permutacija je $3! = 6$, a one su:

- *abc*
- *acb*
- *bac*
- *bca*
- *cab*
- *cba*

Uvod u permutacije multiskupa

Multiskup (ili multiskupina) je skup koji može da sadrži ponovljene elemente. U klasičnoj kombinatorici, permutacija skupa označava uređeni raspored elemenata. Kada govorimo o **permutacijama multiskupa**, bavimo se uređivanjem elemenata skupa u kojem neki elementi mogu da se ponavljaju.

Definicija permutacije multiskupa

Permutacija multiskupa je uređen raspored svih elemenata tog multiskupa, pri čemu uzimamo u obzir ponavljanja elemenata. Ako imamo multiskup od nnn elemenata, gde se neki elementi ponavljaju, broj permutacija ovog multiskupa zavisi o broju pojavljivanja svakog elementa.

Za multiskup koji sadrži n elemenata, od kojih se k različitih elemenata ponavljaju $n1,n2,\ldots,nk$ puta (gde je $n1+n2+\cdots+nk=n$), broj permutacija ovog multiskupa računa se formulom:

$$P = \frac{n!}{n_1!n_2!\cdots n_k!}$$

Ovde je n! faktorijel ukupnog broja elemenata, dok su $n1!,n2!,\ldots,nk!$ faktorijeli brojeva ponavljanja svakog od različitih elemenata.

Detaljno objašnjenje formule

1. **Numerator n!**: Ova vrednost predstavlja broj permutacija običnog skupa koji sadrži n različitih elemenata. Drugim rečima, to je ukupan broj načina na koje bi svi elementi mogli biti uređeni da su svi različiti.

2. **Denominator** $n1!,n2!,\cdots,nk!$: Svaki $n_i!$ u imenitelju kompenzuje višestruke permutacije koje nastaju usled ponavljanja elementa i. Kada se element i pojavljuje n_i puta, bez tog faktora imenitelja, računali bismo te permutacije kao različite, iako se zapravo ne razlikuju zbog istovetnosti elemenata.

Primeri permutacija multiskupa

1. **Jednostavan primer:** Pretpostavimo da imamo multiskup $\{A,A,B\}$. Ovaj multiskup sadrži 3 elementa, od kojih se A ponavlja 2 puta, a B pojavljuje jednom. Da bismo izračunali broj permutacija ovog multiskupa, koristimo formulu:

$$P = \frac{3!}{2!1!} = \frac{6}{2} = 3$$

Permutacije ovog multiskupa su: AAB,ABA,BAA.

2. **Složeniji primer:** Razmotrimo multiskup $\{1,1,1,2,2,3\}$ koji sadrži 6 elemenata, gde se 1 pojavljuje 3 puta, 2 dva puta, a 3 jednom. Broj permutacija multiskupa je:

$$P = \frac{6!}{3!2!1!} = \frac{720}{6 \cdot 2 \cdot 1} = \frac{720}{12} = 60$$

Ovaj multiskup ima ukupno 60 različitih permutacija.

Permutacije sa ponavljanjem i njihova veza sa funkcijama

Permutacije multiskupa mogu biti shvaćene i kao posebni slučajevi funkcija. Naime, permutacija multiskupa odgovara rasporedu elemenata iz multiskupa, gde funkcija treba da preslika n pozicija na n elemenata multiskupa, uz poštovanje ponavljanja.

Specijalni slučajevi:

- Ako nema ponavljanja, tj. $n1=n2=...=nk=1$, formula se svodi na n!, što je klasična formula za broj permutacija običnog skupa.
- Ako je samo jedan element prisutan n puta, tada postoji samo jedna permutacija, jer svi elementi su isti.

Algoritmi za generisanje permutacija multiskupa

Kao što postoje algoritmi za generisanje permutacija običnog skupa, tako postoje i prilagođeni algoritmi za generisanje permutacija multiskupa. Algoritmi za permutacije sa ponavljanjem koriste sledeće korake:

1. **Korak 1**: Svi elementi multiskupa se sortiraju.
2. **Korak 2**: Primenjuje se algoritam za generisanje sledeće leksikografske permutacije, koji vodi računa o ponavljanju elemenata.
3. **Korak 3**: Ponavljanje koraka dok se ne generišu sve permutacije.

Jedan od popularnih algoritama za generisanje permutacija sa ponavljanjem je modifikacija algoritma Steinhaus-Johnson-Trotter, koji generiše permutacije tako da se razlikuju samo u jednoj zameni između uzastopnih permutacija. Za multiskup, ovaj algoritam mora da uzima u obzir identične elemente.

Povezane kombinatorne strukture

Permutacije multiskupa su usko povezane sa varijacijama i kombinacijama sa ponavljanjem. Dok permutacije multiskupa omogućavaju da svi elementi budu uređeni u nizu, varijacije sa ponavljanjem fokusiraju se na izbor podskupova sa uređivanjem.

Zaključak

Permutacije multiskupa predstavljaju bitan deo kombinatorike jer omogućavaju da se elementi koji se ponavljaju urede na različite načine. Formula za broj permutacija multiskupa: $\frac{n!}{n_1!n_2!\cdots n_k!}$ osigurava da se ne prebrojavaju identične permutacije koje nastaju usled ponavljanja elemenata. Algoritmi za generisanje ovih permutacija koriste posebne tehnike za optimizovano generisanje svih mogućih uređenja elemenata multiskupa.

Šta su m-permutacije multiskupa?

Definicija:

m-permutacija multiskupa odnosi se na uređene podskupove sa ponavljanjem elemenata iz multiskupa. To znači da biramo **m** elemenata iz multiskupa od ukupno **n** elemenata, pri čemu se neki elementi mogu ponoviti više puta.

Multiskup se razlikuje od običnog skupa po tome što jedan element može biti prisutan više puta. Na primer, multiskup {a,a,b,b,c} ima 2 kopije elementa a, 2 kopije elementa b, i jednu kopiju elementa c.

Formula za prebrojavanje m-permutacija multiskupa

Za multiskup $M=\{m_1,m_2,...,m_k\}$ gde se element i-tog tipa pojavljuje k_i puta, broj m-permutacija ovog multiskupa može se izračunati koristeći sledeću formulu:

$$P_m = \frac{n!}{k_1!k_2!...k_r!}$$

gde su n svi dostupni elementi u multiskupu, a $k_1,k_2,...,k_r$ su frekvencije svakog elementa u multiskupu.

Kako klasifikovati m-permutacije multiskupa?

Klasifikacija se odnosi na razlikovanje multiskupa u odnosu na permutacije koje se generišu, uzimajući u obzir broj ponavljanja svakog elementa. Da bi se m-permutacije generisale ispravno, važno je pratiti koliko je puta svaki element korišćen.

Algoritam za generisanje m-permutacija multiskupa

1. **Inicijalizacija:** Započni sa multiskupom M i vrednošću m (broj elemenata koje želimo da permutujemo).
2. **Generisanje permutacija:** Koristi algoritam koji prolazi kroz sve elemente multiskupa, i koristi kombinatornu tehniku backtracking-a ili iterativnu tehniku za generisanje svih mogućih permutacija.

Evo osnovnog algoritma za generisanje permutacija multiskupa(Ovaj kod generiše sve moguće permutacije sa dužinom m iz datog multiskupa):

```
1 usage
def generate_m_permutations(multiskup1, m1):
    # Funkcija koja generiše sve m-permutacije iz datog multiskupa.
    # 'multiskup' je lista koja može sadržati ponovljene elemente,
    # a 'm' je broj elemenata u permutaciji.

    result = []

    # 'result' je lista koja će sadržati sve generisane m-permutacije.

    def backtrack(path, remaining1):
        # Rekurzivna funkcija koja gradi permutacije pomoću 'backtrack' tehnike.
        # 'path' je trenutni niz elemenata u permutaciji,
        # a 'remaining' je mapa koja sadrži preostali broj pojavljivanja svakog elementa.

        if len(path) == m1:
            # Ako je trenutna permutacija (path) dužine m (kada smo odabrali m elemenata),
            # dodajemo je u listu rezultata (kao novu listu da izbegnemo reference).
            result.append(list(path))
            return  # Prekidamo dalju rekurziju za ovu granu jer je permutacija završena.

        for elem in multiskup1:
            # Prolazimo kroz sve elemente multiskupa.
            if remaining1[elem] > 0:
                # Ako je preostao još barem jedan neiskorišćeni element 'elem',
                # dodajemo ga u trenutnu permutaciju ('path').
                path.append(elem)
                remaining1[elem] -= 1
                # Smanjujemo broj preostalih kopija tog elementa za 1.
                backtrack(path, remaining1)
```

```
                # Rekurzivno pozivamo 'backtrack' da nastavimo graditi permutaciju.
                remaining1[elem] += 1
                # Vraćamo broj preostalih kopija elementa kada se vratimo iz rekurzije.
                path.pop()
                # Uklanjamo poslednji dodat element iz 'path' (backtracking),
                # kako bi se istražile druge kombinacije.

    remaining = {elem: multiskup1.count(elem) for elem in set(multiskup1)}
    # Kreiramo rečnik 'remaining' koji čuva broj preostalih pojavljivanja
    # svakog elementa u multiskupu. Koristimo 'set(multiskup)' da osiguramo
    # da ne ponavljamo iste elemente više puta.

    backtrack( path: [], remaining)
    # Pozivamo rekurzivnu funkciju 'backtrack' sa praznim 'path' i preostalim elementima.

    return result
    # Vraćamo listu rezultata koja sadrži sve m-permutacije.


# Primer multiskupa
multiskup = ['a', 'a', 'b', 'b', 'c']
# Multiskup koji sadrži dva puta 'a', dva puta 'b' i jednom 'c'.

m = 3
# Želimo da generišemo sve permutacije dužine 3.

permutacije = generate_m_permutations(multiskup, m)
# Pozivamo funkciju da generiše sve 3-permutacije iz datog multiskupa.

print(permutacije)
# Ispisujemo listu svih generisanih permutacija.
```

Primena formule

Ako imamo multiskup {a,a,b,b,c} i želimo da generišemo 3-permutacije, koristićemo algoritam iznad, takođe, ako želimo da prebrojimo broj m-permutacija, koristimo odgovarajuću formulu.

Algoritmi i programi za generisanje prethodnih kombinatornih objekata

1. Varijacije sa ponavljanjem

Generisanje varijacija sa ponavljanjem dužine k, n elemenata:

```python
def varijacije_sa_ponavljanjem(niz, k):
    def generisi(tekuca_varijacija):
        if len(tekuca_varijacija) == k:
            rezultat.append(tekuca_varijacija[:])
            return
        for element in niz:
            tekuca_varijacija.append(element)
            generisi(tekuca_varijacija)
            tekuca_varijacija.pop()

    rezultat = []
    generisi([])
    return rezultat

# Primer
niz = [1, 2, 3]
k = 2
rezultat = varijacije_sa_ponavljanjem(niz, k)
print("Varijacije sa ponavljanjem:", rezultat)
```

ISPIS: Varijacije sa ponavljanjem: [[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]]

2. Varijacije bez ponavljanja

Generisanje varijacija bez ponavljanja dužine k:

```python
def varijacije_bez_ponavljanja(niz, k):
    def generisi(tekuca_varijacija, korisceni):
        if len(tekuca_varijacija) == k:
            rezultat.append(tekuca_varijacija[:])
            return
        for i in range(len(niz)):
            if not korisceni[i]:
                korisceni[i] = True
                tekuca_varijacija.append(niz[i])
                generisi(tekuca_varijacija, korisceni)
                tekuca_varijacija.pop()
                korisceni[i] = False

    rezultat = []
    generisi([], [False] * len(niz))
    return rezultat

# Primer
niz = [1, 2, 3]
k = 2
rezultat = varijacije_bez_ponavljanja(niz, k)
print("Varijacije bez ponavljanja:", rezultat)
```

ISPIS: Varijacije bez ponavljanja: [[1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2]]

3. Permutacije sa ponavljanjem

Generisanje permutacija sa ponavljanjem, gde je dozvoljeno korišćenje istih elemenata više puta:

```python
def permutacije_sa_ponavljanjem(niz, duzina):
    def generisi(tekuca_permutacija):
        if len(tekuca_permutacija) == duzina:
            rezultat.append(tekuca_permutacija[:])
            return
        for element in niz:
            tekuca_permutacija.append(element)
            generisi(tekuca_permutacija)
            tekuca_permutacija.pop()

    rezultat = []
    generisi([])
    return rezultat

# Primer
niz = [1, 2]
duzina = 3
rezultat = permutacije_sa_ponavljanjem(niz, duzina)
print("Permutacije sa ponavljanjem:", rezultat)
```

ISPIS: Permutacije sa ponavljanjem: [[1, 1, 1], [1, 1, 2], [1, 2, 1], [1, 2, 2], [2, 1, 1], [2, 1, 2], [2, 2, 1], [2, 2, 2]]

4. Permutacije bez ponavljanja

Generisanje permutacija bez ponavljanja, gde svaki element može biti korišćen samo jednom:

```python
def permutacije_bez_ponavljanja(niz):
    def generisi(tekuca_permutacija, korisceni):
        if len(tekuca_permutacija) == len(niz):
            rezultat.append(tekuca_permutacija[:])
            return
        for i in range(len(niz)):
            if not korisceni[i]:
                korisceni[i] = True
                tekuca_permutacija.append(niz[i])
                generisi(tekuca_permutacija, korisceni)
                tekuca_permutacija.pop()
                korisceni[i] = False

    rezultat = []
    generisi([], [False] * len(niz))
    return rezultat

# Primer
niz = [1, 2, 3]
rezultat = permutacije_bez_ponavljanja(niz)
print("Permutacije bez ponavljanja:", rezultat)
```

ISPIS: Permutacije bez ponavljanja: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]