Uređeni izbori elemenata

Definicija: Načini biranja i raspoređivanja elemenata iz skupa ili multiskupa gde je redosled izbora važan.

Razlikujemo četiri vrste uređenih izbora:

- Varijacije sa ponavljanjem
- Varijacije bez ponavljanja
- Permutacije sa ponavljanjem
- Permutacije bez ponavljanjem

Varijacija sa ponavljanjem

Definicija: Varijacije sa ponavljanjem opisuju način izbora elemenata iz skupa, pri čemu je važan redosled izbora i dozvoljeno je ponavljanje elemenata. To znači da možemo birati isti element više puta, a različiti redosledi istih elemenata se računaju kao različite varijacije.

Izvođenje formule varijacija sa ponavljanjem

Formula za broj varijacija sa ponavljanjem kada biramo k elemenata iz skupa od n elemenata izračunavamo formulom:

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

Gde je:

- n ukupan broj elemenata u skupu,
- k broj elemenata koje biramo,
- $V_{n,k}$ broj varijacija sa ponavljanjem.

- Pretpostavimo da biramo k elemenata iz skupa koji ima n elemenata, pri čemu se elementi mogu ponavljati.
    1. **Prvi izbor:** Za prvi element u varijaciji imamo n mogućnosti (jer biramo bilo koji element iz skupa n).
    2. **Drugi izbor:** Za drugi element, takođe imamo n mogućnosti jer se elementi mogu ponavljati.
    3. **Treći izbor:** Za treći element, opet imamo n mogućnosti, jer se elementi ponavljaju, i tako dalje.
- Dakle, za svaki od k izbora, imamo n mogućih elemenata koje možemo izabrati. Pošto se elementi mogu ponavljati, svaki izbor je nezavistan od prethodnih.
- Broj različitih varijacija je proizvod svih mogućnosti, pa imamo ukupno:

$$n \times n \times n \times \ldots \times n = n^k$$

Primeri: Imamo skup S = { jabuka, banana, trešanja, grožđe}. Treba da se pronađu sve moguće varijacije sa ponavljanjem kada biramo 3 voćke iz onog skupa, pri čemu je redosled biranja bitan i voćke mogu biti izabrane više puta.

Ukupan broj varijacija sa ponavljanjem jednak je:

$$V_{n,k} = 4^3 = 64$$

Nekoliko primera varijacija sa ponavljanjem za k=3:

- jabuka, jabuka, jabuka
- jabuka, jabuka, banana
- jabuka, jabuka, trešnja
- jabuka, jabuka, grožđe
- jabuka, banana, jabuka
- jabuka, banana, banana
- jabuka, banana, trešnja
- jabuka, banana, grožđe
- trešnja, jabuka, banana
- grožđe, grožđe, grožđe

  ...

Tako nastavljamo dok ne izlistamo svih 64 kombinacija.

Primer: Imamo skup brojeva S={1,2}. Pronađi sve moguće varijacije sa ponavljanjem biranjem 2 broja iz ovog skupa, pri čemu je redosled bitan i brojevi mogu biti izabrani više puta.

Evo svih mogućih varijacija sa ponavljanjem za k=2:

1. 1,1
2. 1,2
3. 2,1
4. 2,2

Varijacije bez ponavljanja

Definicija: Način izbora elemenata iz skupa gde je redosled važan, ali ponavljanje elemenata nije dozvoljano. To znači da svaki element može biti izabran samo jednom.

Izvođenje formule varijacija bez ponavljanja

Formula za broj varijacija sa ponavljanjem kada biramo k elemanata iz skupa od n elemanata izračunava se formulom:

$$V_{n,k} = \frac{n!}{(n-k)!}$$

Gde je:

- n ukupan broj elemenata u skupu,
- k broj elemenata koje biramo,
- $V_{n,k}$ broj varijacija bez ponavljanja.

Pretpostavimo da biramo k elemenata iz skupa koji ima n elemenata, pri čemu se elementi ne smeju ponavljati.

1. **Prvi izbor:** Za prvi element u varijaciji imamo n mogućnosti (biramo bilo koji element iz skupa n).
2. **Drugi izbor:** Nakon što smo izabrali prvi element, on više nije dostupan, pa za drugi izbor ostaje n−1 mogućnost.
3. **Treći izbor:** Nakon što smo izabrali prva dva elementa, za treći izbor ostaje n−2 mogućnosti, i tako dalje.

Broj mogućnosti za svaki izbor se smanjuje jer se elementi ne mogu ponavljati.

Dakle, ukupan broj varijacija je proizvod mogućnosti za svaku poziciju:

$$n \times (n-1) \times (n-2) \times \ldots \times (n-k+1)$$

Ovaj izraz je zapravo deo faktorijela n! — samo biramo prvih k elemenata:

$$V_{n,k} = \frac{n!}{(n-k)!}$$

Primer: Imamo skup S = { narandža, jagoda, kivi, lubenica}. Treba da se pronađu sve moguće varijacije bez ponavljanjem kada biramo 3 voćke iz onog skupa, pri čemu je redosled biranja bitan.

$$V_{4,3} = \frac{4!}{(4-3)!} = 24$$

Nekoliko primera varijacija bez ponavljanja za k=3k = 3k=3:

1. narandža, jagoda, kivi
2. narandža, jagoda, lubenica
3. narandža, kivi, lubenica
4. jagoda, narandža, kivi
5. jagoda, kivi, lubenica
6. kivi, lubenica, narandža

Primer: Ako imamo skup brojeva S={1,2} i biramo 2 broja bez ponavljanja, onda imamo:

$$V_{2,2} = \frac{2!}{(2-2)!} = 2$$

Sve moguće varijacije bez ponavljanja za k=2 su:

1. 1, 2
2. 2, 1

Permutacije bez ponavljanja

Definicija: Permutacija bez ponavljanja je poseban način izbora i rasporeda svih elemenata iz jednog skupa, gde je redosled bitan, a nijedan element se ne može ponoviti.

Izvođenje formule permutacije bez ponavljanja

Broj permutacija bez ponavljanja računa se po formuli:

$$P_n = n!$$

Gde je:

- n ukupan broj elemenata u skupu,
- P_n broj permutacija bez ponavljanja.

1. **Prvi izbor:** Kada biramo prvi element u permutaciji, imamo na raspolaganju n mogućnosti, jer nijedan element još nije odabran.
2. **Drugi izbor:** Nakon što smo odabrali prvi element, ostaje nam n−1 mogućnost za izbor drugog elementa, jer smo jedan element već iskoristili.
3. **Treći izbor:** Nakon što smo odabrali prva dva elementa, za treći izbor ostaje n−2 mogućnosti, i tako dalje.
4. **Ukupan broj mogućnosti:** Nastavljajući ovaj proces za sve n elemenata, dolazimo do toga da broj mogućih rasporeda svih n elemenata izračunavamo kao proizvod svih mogućnosti:

$$P_n = n \times (n-1) \times \ldots \times 1 = n!$$

Primer: Pretpostavimo da imamo skup S={A,B,C}. Ukupan broj elemenata je n=3, pa se broj permutacija bez ponavljanja računa kao:

$$P_3 = 3! = 3 \times 2 \times 1 = 6$$

Sve moguće permutacije bez ponavljanja su:

1. ABC
2. ACB
3. BAC
4. BCA
5. CAB
6. CBA

Permutacije sa ponavljanjem

Definicija: Permutacija je poseban slučaj varijacije gde biramo sve elemente iz skupa, a redosled je važan. Kod permutacija sa ponavljanjima, neki elementi u skupu mogu se ponoviti više puta.

Izvođenje formule permutacije sa ponavljanjem

Broj permutacija sa ponavljanjima računa se po formuli:

$$P_n = \frac{n!}{n_1! \times n_2! \times \ldots \times n_k!}$$

Gde je:

- n ukupan broj elemenata
- $n_1, n_2, \ldots, n_k$ brojevi ponavljanja različitih vrsta elemenata

Formula se izvodi na osnovu formule za permutacije bez ponavljanja P(n) = n!

1. **Uvođenje ponavljanja**: Ako neki elementi nisu različiti, to znači da ćemo dobiti manje jedinstvenih permutacija, jer će se neki rasporedi činiti istim zbog ponavljanja tih elemenata.Na primer, ako imamo element koji se ponavlja n1 puta, različite permutacije koje sadrže te elemente na različitim pozicijama više ne treba smatrati različitim. Zbog toga, moramo "ukloniti" one permutacije koje se ne razlikuju zbog ponavljanja. Ako se neki element ponavlja n1 puta, onda bi broj načina na koji se ti elementi mogu rasporediti međusobno bio $n_1!$. Isto važi za svaki drugi ponovljeni element.
2. **Ukupan broj permutacija sa ponavljanjem**: Ako imamo n elemenata, i neki se ponavljaju n_1, n_2, ..., n_k puta, ukupan broj permutacija je jednak broju permutacija bez ponavljanja (n!) podeljen sa brojem permutacija ponovljenih elemenata ($n_1!$, $n_2!$, ..., $n_k!$)

Primer: Zamislimo da imamo skup A={A,A,B}. Ovde imamo ukupno 3 elementa, od kojih se element A ponavlja 2 puta.

1. Ukupan broj elemenata (n): 3 (dva A i jedan B)
2. Ponavljanja:

   - $n_1 = 2$ (za A)
   - $n_2 = 1$ (za B)

Sada možemo izračunati broj permutacija:

$$P_{3;2,1} = \frac{3!}{2! \times 1!} = \frac{6}{2 \times 1} = 3$$

Dakle, postoji 3 različite permutacije skupa A:

1. AAB
2. ABA
3. BAA

Veze između osobina funkcija i prethodnih objekata

Kako bismo uspostavili veze između osobina funkcija i objekata, ponovo ćemo izdvojiti pravila svake funkcije i povezati ih sa skupovima, primenjujući pravila svake funkcije na svaki objekat, pri čemu su funkcije ranije spomenute varijacije i permutacije, a objekti elementi skupova.

1. Varijacije sa ponavljanjem

Osobina: redosled je važan, a elementi mogu da se ponavljaju

Primer: Skup voćki {jabuka,banana,trešnja,grožđe}, biranje 3 voćke sa ponavljanjem.

Veza: Pošto se elementi mogu ponavljati, broj mogućnosti za svaki izbor ostaje isti, *n*. Zato ukupni broj mogućih kombinacija za biranje 3 voćke iz skupa od 4 je $4^3 = 64$.
Svaka kombinacija gde se jabuka pojavljuje dva ili više puta smatra se drugačijom (npr., "jabuka, jabuka, banana" i "jabuka, banana, jabuka" su različite).

2. **Varijacije bez ponavljanja**

Osobina: Redosled je važan, ali elementi ne smeju da se ponavljaju.

Primer: Skup voćki {narandža,jagoda,kivi,lubenica}, biranje 3 voćke bez ponavljanja.

Veza: Pošto se elementi ne ponavljaju, broj mogućnosti se smanjuje sa svakim izborom. Formula je $V_{n,k} = \frac{n!}{(n-k)!}$, gde n = 4 i k = 3. Dakle, ukupni broj mogućih kombinacija je $V_4^3 = 24$. Ovde kombinacija "narandža, jagoda, kivi" nije ista kao "jagoda, narandža, kivi", jer je redosled važan.

3. **Permutacije bez ponavljanja**

Osobina: Redosled je važan, svi elementi se biraju, i nema ponavljanja.

Primer: Skup slova {A,B,C}.

Veza: S obzirom da biramo sve elemente i da se ne ponavljaju, broj mogućih kombinacija je $P_n=n!$, gde je n=3. Ovde je broj mogućih permutacija 3! = 6, jer je redosled važan, a svaki element može biti izabran samo jednom.

4. **Permutacije sa ponavljanjem**

Osobina: Redosled je važan, ali se neki elementi mogu ponavljati.

Primer: Skup slova {A,A,B}.

Veza: Pošto se jedan element ponavlja (slovo "A"), koristimo formulu $\frac{n!}{n_1! \times n_2!}$, gde je $n_1!$ broj ponavljanja za prvi element. Dakle, imamo 3! Podeljeno sa 2!, što daje 3 različite kombinacije: "AAB," "ABA," i "BAA".

Algoritmi i programi za generisanje prethodnih kombinatornih objekata

1.Varijacije sa ponavljanjem

Najlakši način za prikazivanje varijacija sa ponavljanjem u programu je rekurzija. Nakon svake varijacije sa ponavljanjem skupa {0, 1, … n-1} koja će biti dužine k-1 biramo sledeći element.

```
def write(niz):
    print(niz)                                          //ispis jedne varijacije

2 usages
def variate(niz, n, k):
    if k==0:
        write(niz)                                      //ako smo dosli do kraja, ispisujemo
    else:
        for i in range(n):                              //pozivanje rekurzije za jedan el. manje
            niz[k-1] = i
            variate(niz, n, k-1)


if __name__ == "__main__":
    n = 4                                               //zadajemo n i k
    k = 5
    niz = [0 for i in range(k)]
    variate(niz, n, k)
```

2.Varijacije bez ponavljanja

Opet, i ovde nam je rekurzija glavni element našeg programa. U ovom konkretnom primeru, prvih k elemenata niza menjamo sa ostalima kao da su jedan element.

```
def write(niz, k):
    print(niz[0:k])
```
//ispis prvih k elemenata

```
2 usages
def switch(niz, a, b):
    niz[a], niz[b] = niz[b], niz[a]
```
//zamena elemenata sa m i trenutne pozicije

```
2 usages
def variate(niz, n, k, m):
    if m == k:
        write(niz, k)
    else:
        for i in range(m, n):
            switch(niz, m, i)
            variate(niz, n, k, m+1)
            switch(niz, m, i)
```
//poziv rekurzije

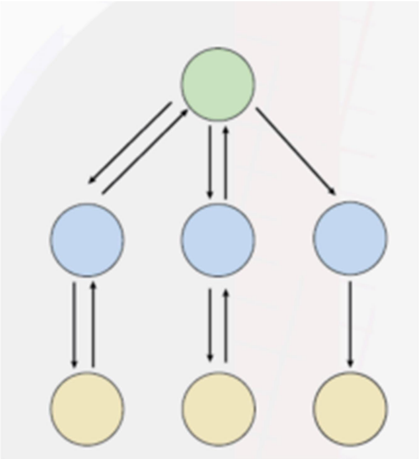
```
if __name__ == "__main__":
    n = 4
    k = 2
    niz = [i for i in range(n)]
    variate(niz, n, k,  m: 0)
```
//inicijalizacija n i k

3.Permutacije bez ponavljanja

Da bi generisali sve permutacije bez ponavljanja, možemo koristiti „backtracking“ algoritam koji rekurzivno traći sve moguće permutacije I vraća se unazad kada istraži sve mogućnosti.

Slikovito možemo objasniti rad ovog algoritma koristeći strukturu stabla.(Slika 3.1)

```
def swap(string, i, j):
    string[i], string[j] = string[j], string[i]
    return string


def permute(string, index):

    if index == len(string) - 1:
        print(string)
        return

    for i in range(index, len(string)):

        string = swap(string, index, i)

        permute(string, index + 1)

        string = swap(string, index, i)


def permutations(string):
    permute(string, 0)

if __name__ == "__main__":
    permutations([1, 2, 3])
```



Slika 3.1

4.Permutacije sa ponavljanjem

Kod generisanja permutacija sa ponavljanjem moramo uzeti u obzir da elementi skupa nisu jedinstveni I da izbacimo duple permutacije iz konačnog rešenja

```
def findCeil(str1, first, l, h):
    ceilIndex = l

    for i in range(l + 1, l + h):
        if (str1[i] > first and str1[i] < str1[ceilIndex]):
            ceilIndex = i

    return ceilIndex

def sortedPermutations(str1):
    size = len(str1)
    str1 = list(str1)

    str1.sort()

    isFinished = False
    x = 1
    while (not isFinished):

        print(x, "".join(str1))
        x += 1

        i = len(str1) - 2
        while i >= 0:
            if (str1[i] < str1[i + 1]):
                break
            i -= 1

        if (i == -1):
            isFinished = True
        else:

            ceilIndex = findCeil(str1, str1[i], i + 1, size - 1)

            temp = str1[i]
            str1[i] = str1[ceilIndex]
            str1[ceilIndex] = temp

            str1 = str1[0: i + 1] + sorted(str1[i + 1:])

string = "AABBB"
sortedPermutations(string)
```