Šta znači nabrojati i prebrojati elemente nekog skupa?

U kontekstu skupova, izrazi "nabrojati" i "prebrojati" imaju različita značenja:

1. **Nabrojati elemente skupa** znači navesti sve elemente koji pripadaju skupu, jedan po jedan. Na primer, ako imamo skup A={1,2,3}, onda nabrajanje elemenata tog skupa podrazumeva reći: "1, 2, 3".
2. **Prebrojati elemente skupa** znači utvrditi koliko elemenata ima u tom skupu. Na primer, za skup A={1,2,3}, prebrojavanje bi rezultiralo brojem 3, jer skup A sadrži tri elementa.

Dakle, "nabrojati" se odnosi na navođenje pojedinačnih elemenata, dok "prebrojati" znači izračunati njihov ukupan broj.

Zadatak 1:

Dat je skup A={2,4,6,8,10,12,14,16}

Nabroj sve elemente skupa A.

1. **Prebroj** koliko elemenata ima skup A.
2. Formiraj novi skup B, koji se sastoji od svih parnih brojeva iz skupa A koji su veći od 6.
3. **Nabroj** elemente skupa B.
4. **Prebroj** elemente skupa B.

Resenje:

- **Nabrajanje elemenata skupa** A:
  - Elementi skupa A su: 2, 4, 6, 8, 10, 12, 14, 16.
- **Prebrojavanje elemenata skupa** A:
  - Skup A ima ukupno 8 elemenata.
- Formiraj novi skup B, koji se sastoji od svih parnih brojeva iz skupa A koji su veći od 6:
  - Parni brojevi veći od 6 iz skupa A su: 8, 10, 12, 14, 16.
  - Dakle, skup B={8,10,12,14,16}.
- **Nabrajanje elemenata skupa** B:
  - Elementi skupa B su: 8, 10, 12, 14, 16.
- **Prebrojavanje elemenata skupa** B:
  - Skup B ima ukupno 5 elemenata.

Zadatak 2:

Zamislimo da radiš na sistemu za upravljanje korisnicima u softverskoj aplikaciji. Imaš listu korisnika sa različitim statusima:

Skup svih korisnika je U={K1,K2,K3,K4,K5,K6,K7,K8}, gde su K1,K2,...,K8 korisnici. Svaki korisnik može imati jedan od dva statusa: **aktivni** ili **neaktivni**.

Statusi korisnika su sledeći:

- Aktivni korisnici: A={K1,K3,K5,K7}
- Neaktivni korisnici: N={K2,K4,K6,K8}

1. **Nabroj** sve aktivne korisnike.
2. **Prebroj** koliko ima aktivnih korisnika.
3. Formiraj skup korisnika koji su **aktivni** i čiji ID (broj u oznaci Ki) je **neparan**.
4. **Nabroj** te korisnike i **prebroj** koliko ih ima.
5. Kako bi ovaj proces mogao da bude korisno implementiran u aplikaciji za praćenje korisničkog statusa? Na primer, kako bi mogao da koristiš algoritme ili baze podataka za ovakvu funkcionalnost?

Skup svih korisnika je U={K1,K2,K3,K4,K5,K6,K7,K8}

- Aktivni korisnici su: A={K1,K3,K5,K7}
- Neaktivni korisnici su: N={K2,K4,K6,K8}

1. Nabrajanje aktivnih korisnika:

- Aktivni korisnici su: K1,K3,K5,K7

2. Prebrojavanje aktivnih korisnika:

- Skup aktivnih korisnika A ima 4 elementa, tj. 4 aktivna korisnika.

3. Formiraj skup korisnika koji su aktivni i čiji ID je neparan:

- Aktivni korisnici koji imaju neparan ID su K1,K3,K5,K7, jer su svi njihovi ID brojevi neparni.
- Dakle, skup ostaje isti: B={K1,K3,K5,K7}

4. Nabrajanje i prebrojavanje korisnika iz skupa B:

- Korisnici sa neparnim ID-jem su: K1,K3,K5,K7.
- Broj takvih korisnika je 4.

5. Kako bi se ovo moglo primeniti u softverskoj aplikaciji?

U kontekstu softverskog inženjerstva, ovaj zadatak se može lako primeniti na sistem upravljanja korisnicima u aplikaciji. Evo kako bi to moglo da se implementira:

Koraci implementacije:

1. **Baza podataka**:
    - Možeš imati tabelu u bazi podataka koja sadrži informacije o korisnicima, uključujući njihov status (aktivni/neaktivni).
    - Na primer, tabela "Korisnici" može imati kolone: ID, ime, status (aktivni/neaktivni), i druge relevantne informacije.
2. **Algoritmi**:
    - Ako radiš sa skupovima u programskom jeziku kao što je Python, možeš koristiti liste i skupove. Na primer:

```python
svi_korisnici = { 'K1', 'K2', 'K3', 'K4', 'K5', 'K6', 'K7', 'K8' }
aktivni_korisnici = { 'K1', 'K3', 'K5', 'K7' }
neaktivni_korisnici = { 'K2', 'K4', 'K6', 'K8' }

# Aktivni korisnici sa neparnim ID-jem
aktivni_neparni = { korisnik for korisnik in aktivni_korisnici if int(korisnik[1]) % 2 != 0 }
print(aktivni_neparni)
print(len(aktivni_neparni))  # Broj aktivnih korisnika sa neparnim ID
```

Ovo je korisno kada se želi upravljati korisničkim statusima u realnom vremenu ili kad se prate specifične grupe korisnika.

Prebrojavanje elemenata konačnog skupa

Kardinalnost skupa

- Broj koji predstavlja mjeru veličine skupa i definiše se uz pomoć bijektivnih preslikavanja.
- Po kardinalnosti skup može biti:
  1. Konačan
  2. Beskonačan
     - 2.1. Prebrojiv
     - 2.2. Neprebrojiv

Za skup S kažemo da je konačan ako važi:

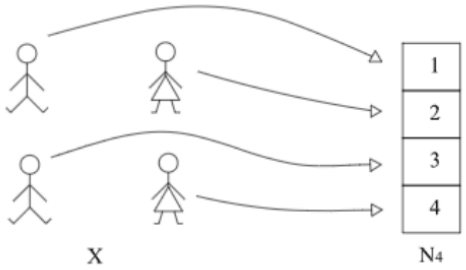
- *Ako je $S = \emptyset$, važi da je broj elemenata skupa jendak 0, tj. $|S| = 0$;*
- *Ako je $S \neq \emptyset$ i postoji bijektivno preslikavanje skupa S u skup N_n (od n elemenata) onda važi da je kardinalnost skupa S jednaka n ($|S| = n$).*

Prebrojavanje konacnog skupa X je odredivanje broja n za koji

postoji bijekcija

$$f : X \rightarrow Nn.$$

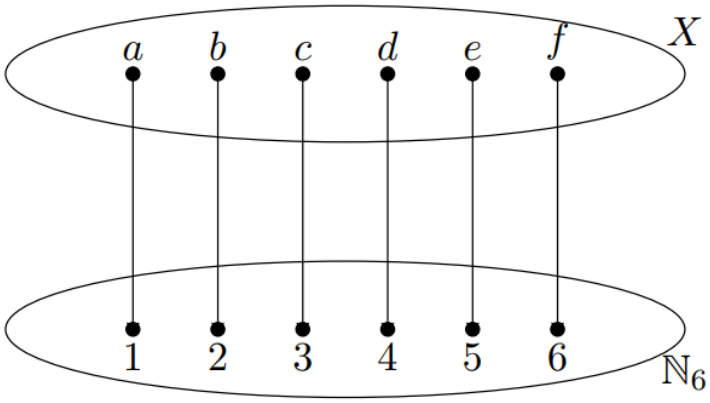
Jasno je da je funkcija f bijekcija, jer ukoliko nismo pogriješili pri brojanju, svaki element X dobija različiti broj i svaki broj iz Nn se dodeljuje nekom elementu iz X.



Slika 2.1: Prebrojavanje studenata.

Ako je skup konačan, možemo jednostavno nabrojati sve njegove elemente i tako dobiti ukupan broj. Označimo konačan skup s A, tada se broj elemenata u skupu A naziva kardinalnost skupa i označava se sa $|A|$.

Na primjer, ako imamo skup X={1,2,3,4,5,6}, kardinalnost ovog skupa je $|X|=6$, jer skup sadrži 6 elemenata.



Prebrojavanje elemenata ima ključnu ulogu u kombinatorici. Potrebno je prebrojavati objekte sa odredjenom vrijednošću da bi se riješili različiti problemi.

Primjene prebrojavanja:

1. Utvrđivanje kompleksnosti algoritama,
2. Utvrđivanje da li ima dovoljno telefonskih brojeva ili IP adresa da zadovolje potražnju
3. Matematička biologija - sekvenciranje DNA

Štaviše, tehnike prebrojavanja se široko koriste da bi se izračunavale vjerovatnoće događaja.

Primjer korišćenja tehnika prebrojavanja: Recimo da se lozinka koju je potrebno napraviti može sastojati od 6, 7 ili 8 karaktera i da svaki od tih karaktera može biti cifra ili slovo alfabeta. Takođe, svaka lozinka mora sadržati barem jednu cifru. Koliko takvih mogućih lozinki postoji?

Zadaci:

1. Koliko elemenata ima skup F={0,5,10,15,...,50}?
2. Ako je skup G skup svih slova u riječi "MATEMATIKA", koliko različitih elemenata ima skup G?

Rješenja:

1. Skup F ima 11 elemenata, jer elementi kreću od 0 sa korakom 5.
2. Skup G ima 6 elemenata obzirom da skupovi nemaju duplikate.

Kako se prebrojavaju elementi prebrojivo beskonačnog skupa?

Kardinalnost skupa

Između bilo koja dva konačna skupa sa istim brojem elemenata postoji bijekcija.

Definicija 1: Skupovi A i B imaju istu kardinalnost ako i samo ako postoji bijektivna funkcija između A i B. Kada A i B imaju istu kardinalnost, pišemo $|A| = |B|$.

Za beskonačne skupove, definicija kardinalnosti pruža relativnu mjeru veličine dva skupa, umjesto apsolutne veličine jednog određenog skupa.

Definicija 2: Ako postoji funkcija koja je injektivna (jedan na jedan) iz skupa A u skup B, tada je kardinalnost skupa A manja ili jednaka kardinalnosti skupa B i pišemo $|A| \leq |B|$. Odnosno kada je $|A| \leq |B|$ i A i B imaju različitu kardinalnost, kažemo da je kardinalnost skupa A manja od kardinalnosti skupa B i pišemo $|A| < |B|$.

Definicija prebrojivo beskonačnog skupa

- Skup A je prebrojivo beskonačan ako postoji bijekcija između A i $\mathbb{N}$, $f : \mathbf{N} \rightarrow \mathbf{A}$. To znači da možemo elemente skupa A poređati kao niz $a_1, a_2, a_3, \ldots$ tako da svaki element iz A ima odgovarajući indeks iz $\mathbb{N}$ i svaki prirodni broj odgovara jednom elementu iz A.

Prebrojavanje elemenata prebrojivo beskonačnog skupa

Prebrojavanje elemenata prebrojivo beskonačnog skupa se zasniva na ideji da svaki element tog skupa možemo urediti tako da odgovara nekom broju iz skupa $\mathbb{N}$, odnosno za takav skup postoji bijekcija između njegovih elemenata i elemenata skupa prirodnih brojeva.

Primjeri prebrojivo beskonačnih skupova

a) Skup prirodnih brojeva N

Skup prirodnih brojeva N={1,2,3,... } je trivijalno prebrojivo beskonačan jer možemo očigledno napraviti bijekciju f(n)=n, gde je f : N→N

b) Skup cijelih brojeva Z

Skup cijelih brojeva Z={...,−3,−2,−1,0,1,2,3,... } je također prebrojivo beskonačan. Iako se na prvi pogled čini da je skup "veći" od N, možemo napraviti odgovarajuću bijekciju između Z i N.

Jedan način je sljedeći:

$$f(n) = \{ n/2,\ ako\ je\ n\ paran,\ \ -(n-1)/2,\ ako\ je\ n\ neparan \}$$

Ova funkcija mapira:

- n=1→f(1)=0
- n=2→f(2)=1
- n=3→f(3)=−1
- n=4→f(4)=2
- n=5→f(5)=−2 itd.

Dakle, za svaki prirodan broj n, postoji tačno jedan element f(n) iz skupa Z, i obrnuto.

c) Skup racionalnih brojeva Q

Skup racionalnih brojeva Q je prebrojivo beskonačan, iako se čini da bi mogao biti "gušći" nego N ili Z, jer između svaka dva cijela broja postoji beskonačno mnogo racionalnih brojeva. Postoji metoda pomoću koje možemo urediti sve racionalne brojeve tako da ih možemo prebrojati. Ova metoda je poznata kao **Cantorova dijagonalna metoda**.

Koraci:

1. Prvo predstavimo sve racionalne brojeve kao parove $\frac{p}{q}$ gde su p∈Z , a q∈N. Npr. $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{2}{2}$...
2. Zatim možemo rasporediti te parove u oblik dijagonalne matrice i prebrojavati ih dijagonalno, izbjegavajući ponavljanje istih vrijednosti (npr.$\frac{2}{2} = 1$).

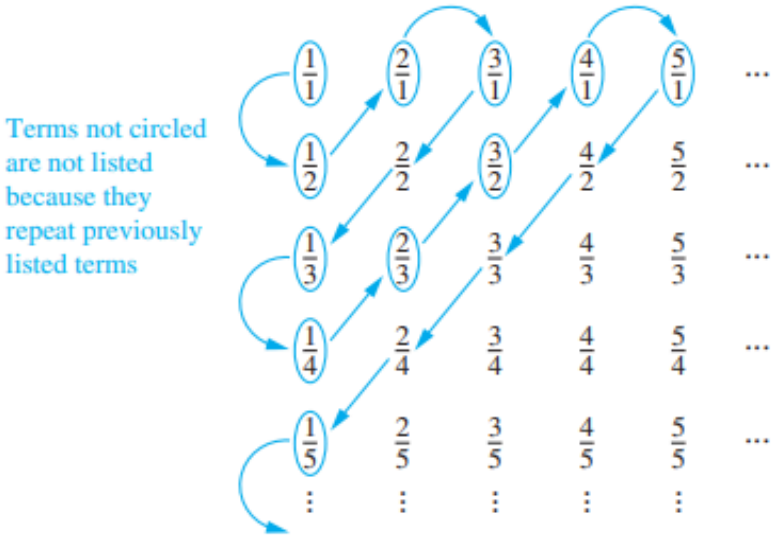
Tako dobijamo beskonačan niz racionalnih brojeva, čime dokazujemo da je skup Q prebrojivo beskonačan.

Zadaci

1. Pokaži da je skup pozitivnih racionalnih brojeva prebrojivo beskonačan.

Rešenje:

Pozitivne racionalne brojeve možemo navesti kao niz $r_1, r_2, r_3, \ldots, r_n \ldots$. Primetimo da je svaki pozitivan racionalni broj količnik p/q dva pozitivna cela broja. Brojeve ovog skupa možemo urediti tako da ćemo navesti one sa imeniocem $q = 1$ u prvom redu, one sa imeniocem $q = 2$ u drugom redu i tako redom kao što je prikazano na slici ispod.



Kod navođenja brojeva u nizu bitno je da se prvo navedu pozitivni racionalni brojevi p/q sa $p + q = 2$, zatim oni sa $p + q = 3$ i tako dalje, prateći putanju prikazanu na gornjoj slici. Kad god naiđemo na broj p/q koji je već naveden, nećemo ga ponovo navoditi. Na primer, kada dođemo do $2/2$, to je 1, pa ga nećemo navoditi jer smo već naveli $1/1$. Početni članovi ovog niza zaokruženi su na gorepomenutoj slici, to su: $1, 1/2, 2, 3, 1/3, 1/4, 2/3, 3/2, 4, 5$, i tako dalje. S obzirom da su svi pozitivni racionalni brojevi navedeni samo jednom, pokazano je da je skup pozitivnih racionalnih brojeva prebrojivo beskonačan, tj. postoji bijekcija između skupa prirodnih brojeva $\boldsymbol{N}$ i skupa $\boldsymbol{Q}^+$.

2. Pretpostavimo da sistem treba da kreira jedinstvene korisničke ID-ove za beskonačan broj korisnika. Kako da osiguramo da svaki korisnik dobije jedinstven identifikator?

Rešenje:

U softverskim sistemima koji obrađuju mnogo korisnika (potencijalno beskonačno mnogo), možemo koristiti činjenicu da je skup prirodnih brojeva prebrojivo beskonačan kako bismo svakom korisniku dodelili ID. Na taj način, bez obzira na broj korisnika, uvek imamo sledeći dostupan ID. Sistem može automatski dodeljivati ID-ove bez potrebe za ručnim unosom, što olakšava upravljanje velikim brojem korisnika (npr. društvene mreže).

```python
class SistemKorisnika:
    def __init__(self):
        self.baza_korisnika = []  # Lista koja čuva podatke o korisnicima
        self.sledeci_id = 1  # Inicijalni ID kreće od 1

    def dodaj_korisnika(self, ime, prezime):
        # Svakom korisniku dodeljujemo jedinstveni ID iz prebrojivog skupa prirodnih brojeva
        korisnik = {
            'id': self.sledeci_id,
            'ime': ime,
            'prezime': prezime
        }
        self.baza_korisnika.append(korisnik)
        print(f"Dodat novi korisnik: {korisnik}")

        # Povećavamo sledeći ID za novog korisnika
        self.sledeci_id += 1

    def prikazi_korisnike(self):
        print("Svi korisnici u sistemu:")
        for korisnik in self.baza_korisnika:
            print(f"ID: {korisnik['id']}, Ime: {korisnik['ime']} {korisnik['prezime']}")
```

Na slici je prikazana jednostavna implementacija ove ideje u Python-u.

Koji principi se mogu prepozna? prilikom prebrojavanja elemenata konačnog skupa

Prebrojavanje elemenata konačnih skupova zasniva se na nekoliko ključnih matematičkih principa koji su temelj kombinatorike. Ovi principi se primenjuju kako u teorijskim zadacima, tako i u praktičnim problemima.

1. Princip zbira

Princip zbira koristi se kada se elementi skupa mogu podeliti u disjunktne podskupove. Ukupan broj elemenata u skupu jednak je zbiru elemenata tih podskupova:

$$|A| = |A_1| + |A_2| + \cdots + |A_n|.$$

Na primer, ako student bira ispitno pitanje iz jedne od tri grupe pitanja, broj različitih pitanja je suma elemenata tih grupa.

Princip zbira se često koristi u problemima klasifikacije i raspodele. Ovaj princip se može proširiti na složenije slučajeve kada imamo više od dva skupa. Takođe, kada su skupovi koji se prebrojavaju disjunktni (ne preklapaju se), možemo jednostavno sabirati njihove veličine. U slučajevima gde skupovi nisu disjunktni, potrebna je primena principa uključenja-isključenja.

Ovaj princip se često koristi u analizi složenosti algoritama, gde se algoritam sastoji iz više disjunktnih koraka. Ukupna složenost je zbir složenosti pojedinačnih koraka.

Ako skupovi nisu disjunktni, princip zbira ne može biti direktno primenjen jer dolazi do duplog prebrojavanja elemenata u preseku.

Zadatak: Na fakultetu, jedan student može izabrati jedno ispitno pitanje iz tri različite grupe. Prva grupa sadrži 5 pitanja, druga 8 pitanja, a treća 7 pitanja. Koliko ukupno različitih ispitnih pitanja može student odabrati ako može birati iz bilo koje grupe?

Rešenje: Pošto su grupe disjunktne (ne preklapaju se), broj različitih pitanja koja student može izabrati je suma elemenata u sve tri grupe:

Ukupan broj pitanja=5+8+7=20\text{Ukupan broj pitanja} = 5 + 8 + 7 =
20Ukupan broj pitanja=5+8+7=20

Student može birati iz ukupno 20 različitih pitanja.

Ovaj softverski primer potvrđuje **princip zbira** jer se pokazuje da se ukupan broj elemenata (ispitnih pitanja) dobija sabiranjem elemenata disjunktnih skupova (grupa pitanja).

```
#include <iostream>


int main() {

   // Skupovi pitanja koristeći osnovne statičke nizove

   const int velicinaGrupa1 = 5;

   const int velicinaGrupa2 = 8;

   const int velicinaGrupa3 = 7;


   // Ukupan broj pitanja

   int ukupanBrojPitanja = velicinaGrupa1 + velicinaGrupa2 + velicinaGrupa3;


   std::cout << "Ukupan broj ispitnih pitanja koje student može odabrati: " << ukupanBrojPitanja <<
std::endl;


   return 0;

}
```

Primena **principa zbira** u programiranju i kompjuterskoj nauci:

1. **Analiza složenosti algoritama**:
    - Zbir vremenskih složenosti disjunktnih faza algoritma daje ukupnu složenost.

2. **Pretraga podataka**:
    - Ukupan broj rezultata pretrage je zbir rezultata iz različitih nezavisnih izvora podataka.
3. **Paralelno programiranje**:
    - Ukupan broj zadataka ili operacija je zbir zadataka koje izvršava svaka paralelna nit.
4. **Raspodela resursa**:
    - Ukupan broj resursa koji koriste različiti procesi je zbir resursa svakog pojedinačnog procesa.
5. **Baze podataka (SQL)**:
    - Kada koristimo UNION na disjunktnim tabelama, ukupan broj rezultata je zbir rezultata iz svake tabele.
6. **Sistemi za preporuku**:
    - Ako se preporuke generišu iz disjunktnih kategorija, ukupan broj preporuka je zbir rezultata iz svake kategorije.

Svaka od ovih situacija primenjuje **princip zbira** kako bi se izračunao ukupan broj elemenata iz disjunktnih skupova.

2. Princip proizvoda

Princip proizvoda primenjuje se kada se vrši izbor iz dva ili više konačnih skupova. Ukupan broj mogućih izbora jednak je proizvodu broja elemenata tih skupova:

$$|A \times B| = |A| \cdot |B|.$$

Na primer, broj nizova dužine *8* sa elementima 0 i 1 je $2^8=256$ različitih nizova. Ako biramo

Zadatak: U školi se organizuje takmičenje u kojem učenici mogu izabrati jedan od tri predmeta: Matematiku, Hemiju i Fiziku. Pored predmeta, svaki učenik može izabrati jednu od četiri različite boje za svoju zastavu. Koliko različitih kombinacija predmeta i boja mogu izabrati učenici?

Rešenje: Da bismo izračunali ukupan broj kombinacija, primenjujemo princip proizvoda:

- Broj predmeta = 3 (Matematika, Hemija, Fizika)
- Broj boja = 4 (Crvena, Plava, Zelena, Žuta)

Ukupan broj kombinacija =broj predmeta×broj boja=3×4=12

Dakle, učenici mogu izabrati **12 različitih kombinacija** predmeta i boja.

```
Ovaj program jasno ilustruje primenu principa proizvoda u situaciji kada se vrši izbor iz više
nezavisnih skupova.

#include <iostream>

int main() {

   // Broj predmeta i boja

   const int brojPredmeta = 3; // Matematika, Hemija, Fizika

   const int brojBoja = 4;     // Crvena, Plava, Zelena, Žuta


   // Ukupan broj kombinacija

   int ukupanBrojKombinacija = brojPredmeta * brojBoja;


   // Prikaz rezultata

   std::cout << "Ukupan broj kombinacija predmeta i boja: " << ukupanBrojKombinacija << std::endl;


   return 0;

}
```

automobil i njegovu boju, gde imamo n modela automobila i m boja, ukupan broj izbora je n×m.

Primena principa proizvoda u programiranju i kompjuterskoj nauci:

1. **Kombinacije ulaznih podataka**: Kada razvijate softver koji uzima višestruke ulaze od korisnika, princip proizvoda se može koristiti za izračunavanje ukupnog broja kombinacija ulaza. Na primer, ako korisnik može izabrati veličinu (S, M, L) i boju (Crvena, Plava, Zeleno), ukupan broj kombinacija biće 3×3=9.
2. **Generisanje korisničkih imena**: U aplikacijama gde se korisničkim imenom može kombinovati više elemenata (na primer, ime, prezime, broj), princip proizvoda se može koristiti za izračunavanje broja mogućih korisničkih imena. Na primer, ako je broj imena 5, prezimena 4, i brojeva 10, ukupan broj korisničkih imena biće 5×4×10=200.
3. **Testiranje softvera**: Kada se testiraju aplikacije sa različitim kombinacijama ulaznih vrednosti, princip proizvoda se koristi za izračunavanje ukupnog broja test slučajeva. Na primer, ako se testira aplikacija sa 3 različita uređaja i 4 različite verzije softvera, ukupan broj test slučajeva biće 3×4=12.
4. **Razvoj interfejsa**: Kada se dizajnira korisnički interfejs sa različitim komponentama (npr. dugmad, liste, formi), princip proizvoda se koristi za izračunavanje ukupnog broja mogućih rasporeda. Na primer, ako imate 3 dugmeta i 2 liste, ukupan broj rasporeda će biti 3×2=6.

5. **Kombinacije opcija u pretraživačima**: Kada se korisnicima pružaju različite opcije za filtriranje rezultata pretrage (npr. kategorija, cena, ocena), princip proizvoda se koristi za izračunavanje ukupnog broja mogućih kombinacija. Na primer, ako postoje 4 kategorije, 3 opsega cena, i 2 nivoa ocena, ukupan broj kombinacija za pretragu biće $4 \times 3 \times 2 = 24$.

Ovi primeri jasno ilustruju kako se princip proizvoda može primeniti u raznim oblastima programiranja i kompjuterske nauke, olakšavajući analizu i rešavanje problema koji uključuju više nezavisnih izbora.

3. Dirihleov princip

Dirihleov princip (poznat i kao princip fioke) kaže da ako rasporedimo više objekata nego što ima fioka, najmanje jedna fioka mora sadržati više od jednog objekta. Ako raspodelimo n objekata u k fioka, i ako je $n>k$, tada najmanje jedna fioka mora sadržati više od jednog objekta.

Na primer, u grupi od 13 ljudi, bar dvoje će imati rođendan u istom mesecu.

Dirihleov princip je egzistencijalni princip — on nam samo kaže da neki objekat mora postojati, ali ne pruža način da taj objekat identifikujemo ili pronađemo.

Zadatak:
U jednoj školi, 30 učenika je raspoređeno u 12 različitih klasa. Da li je sigurno da će bar dvoje učenika biti u istoj klasi? Koristite Dirihleov princip da dokažete svoje tvrdnje.

Rešenje:
Prema Dirihleovom principu, ako imamo nnn objekata (u ovom slučaju učenike) i kkk fioka (klase), i ako je n>kn > kn>k, onda bar jedna fioka mora sadržavati više od jednog objekta. U ovom primeru imamo:

- n=30 (učenici)
- k=12 (klase)

Pošto je 30>12, može se zaključiti da najmanje jedna klasa mora sadržavati više od jednog učenika.

Dakle, bar dvoje učenika će biti u istoj klasi.

```
#include <iostream>

int main() {
    const int brojUcenika = 5; // Broj učenika
    int ocene[brojUcenika];

    // Unos ocena
    std::cout << "Unesite ocene za " << brojUcenika << " učenika (1-5):" << std::endl;
    for (int i = 0; i < brojUcenika; i++) {
        std::cin >> ocene[i];
    }

    // Proveravamo da li se ocene ponavljaju
    for (int i = 0; i < brojUcenika; i++) {
        for (int j = i + 1; j < brojUcenika; j++) {
            if (ocene[i] == ocene[j]) {
                std::cout << "Bar dvoje učenika ima istu ocenu: " << ocene[i] << std::endl;
                return 0; // Izlazimo iz programa
            }
        }
    }

    std::cout << "Sve ocene su jedinstvene." << std::endl;

    return 0;
}
```

Primena Dirihleovog principa u programiranju i kompjuterskoj nauci:

1. **Haširanje**: Kada se podaci haširaju u haš tabeli, ako broj unikatnih ključeva premašuje broj dostupnih pozicija u tabeli, mora doći do sudara (tj. dva ključa će se dodeliti istoj poziciji).
2. **Raspodela resursa**: Kada se resursi (npr. serveri, procesi) raspodeljuju među korisnicima ili aplikacijama, ako je broj korisnika veći od broja dostupnih resursa, barem jedan resurs će biti dodeljen više od jednom.
3. **Projekti i zadaci**: U sistemima za upravljanje projektima, ako se dodeli više zadataka nego što ima članova tima, barem jedan član tima će morati da preuzme više od jednog zadatka.
4. **Kombinacije boja**: Ako u grafičkom interfejsu imamo više od četiri različite boje, a samo četiri boje su dozvoljene za popunjavanje dijagrama, barem jedna boja će se ponavljati.
5. **Distribucija podataka**: U sistemima za skladištenje podataka, kada se više datoteka pohranjuje u manje od potrebnog broja fioka, barem jedna fioka će sadržati više od jedne datoteke.

4. Princip uključenja-isključenja

Princip uključenja-isključenja koristi se za prebrojavanje elemenata više skupova koji se preklapaju. Za dva skupa A i B, broj elemenata u uniji je:

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

Ovaj princip je koristan ne samo za dva skupa, već i za više skupova, i igra ključnu ulogu u problemima koji uključuju preklapanje ili preseke skupova.

Zadatak:U jednoj školi, učitelji su podelili zadatke učenicima iz dva različita predmeta: matematike i hemije.

- 20 učenika je dobilo zadatak iz matematike.
- 15 učenika je dobilo zadatak iz hemije.
- 5 učenika je dobilo zadatak iz oba predmeta.

Koliko je ukupno učenika dobilo zadatke iz barem jednog od ova dva predmeta?

Rešenje: Koristeći princip uključenja-isključenja, možemo izračunati broj učenika koji su dobili zadatke iz barem jednog predmeta.

Neka je:

- $|A|=20$ (učenici koji su dobili zadatak iz matematike)
- $|B|=15$ (učenici koji su dobili zadatak iz hemije)
- $|A \cap B|=5$ (učenici koji su dobili zadatak iz oba predmeta)

Ukupan broj učenika koji su dobili zadatke iz barem jednog predmeta $|A \cup B||A \cup B|$ izračunava se kao:

$$|A \cup B|=|A|+|B|-|A \cap B|$$

Ubacujući vrednosti:

$$|A \cup B|=20+15-5=30$$

Dakle, ukupno 30 učenika je dobilo zadatke iz barem jednog od ova dva predmeta.

Primena principa uključenja-isključenja u programiranju i kompjuterskoj nauci:

1. **Prebrojavanje jedinstvenih elemenata**: Kada se rade upiti nad bazama podataka, princip uključenja-isključenja se koristi za prebrojavanje jedinstvenih elemenata u više tabela. Na primer, ako želite da znate koliko jedinstvenih korisnika je registrovano u različitim tabelama korisnika.
2. **Statistika o korisnicima**: U analizi podataka, kada se prate korisnici koji koriste više servisa, princip se može koristiti za izračunavanje ukupnog broja jedinstvenih korisnika koji su koristili jedan ili više servisa.
3. **Istraživanje preklapanja skupova**: U aplikacijama koje prate upotrebu resursa, kao što su sistemski resursi ili mrežni resursi, princip uključenja-isključenja može se koristiti za identifikaciju koliko korisnika koristi više resursa i koliko korisnika koristi samo jedan resurs.
4. **Planiranje događaja**: Kada organizujete događaje i želite da znate koliko jedinstvenih učesnika će prisustvovati, princip se može primeniti ako su učesnici registrovani za više događaja, kako bi se izbeglo preklapanje.
5. **Pretraga i upiti u grafovima**: U grafovima, kada se pretražuju čvorovi koji pripadaju različitim grupama, princip uključenja-isključenja se može koristiti za određivanje broja jedinstvenih čvorova koji se nalaze u više grupa.

6. **Teorija skupova i kombinatorika**: U kombinatorici, kada se analiziraju problemi koji se tiču kombinacija elemenata iz više skupova, princip uključenja-isključenja se koristi za izračunavanje broja kombinacija koje uključuju ili isključuju određene elemente.

Ovi primeri pokazuju kako se princip uključenja-isključenja može koristiti u raznim situacijama u programiranju i analizi podataka, posebno kada se radi sa skupovima i preklapanjima između njih.

```
#include <iostream>

int main() {

   // Broj učenika koji su dobili zadatke iz matematike i hemije

   int brojMatematika = 20;

   int brojHemija = 15;

   int brojOba = 5;

   // Računanje ukupnog broja učenika koji su dobili zadatke iz barem jednog predmeta

   int ukupnoUcenika = brojMatematika + brojHemija - brojOba;

   std::cout << "Ukupan broj učenika koji su dobili zadatke iz barem jednog predmeta: "

         << ukupnoUcenika << std::endl;

   return 0;

}
```