Poglavlje 6

Cantorov svijet

6.1 Teorija skupova, beskonačnost i granice razuma

U ovom poglavlju istražujemo temelje teorije skupova kroz praktičnu Python implementaciju, inspirirani Cantorovim revolucionarnim radom koji je promijenio naše razumijevanje beskonačnosti.

> "Bit matematike leži upravo u njezinoj slobodi" (Das Wesen der Mathematik liegt gerade in ihrer Freiheit) - Georg Cantor, 1883

Cantor je pokazao da beskonačnost nije jedinstvena - postoje različite "veličine" beskonačnosti, što je dovelo do dubokih filozofskih i matematičkih posljedica.

6.1.1 Osnovni pojmovi teorije skupova

Skup je temeljan, nedefiniran pojam u matematici - kolekcija objekata koje nazivamo **elementima**. Cantor je definirao:

> "Skup je mnoštvo koje mislimo kao jedinstvo" (*Eine Menge ist ein Vieles, welches sich als Eines denken lässt*)

Implementirajmo osnovne skupovne operacije:

```python
class Skup:
    """Predstavlja matematički skup s osnovnim operacijama."""

    def __init__(self, elementi):
        self.elementi = set(elementi) if not isinstance(elementi, set) else elementi
```

```
6
7       def __repr__(self):
8           if not self.elementi:
9               return "∅"
10          return "{" + ", ".join(str(e) for e in sorted(self.elementi)) + "}"
11
12      def __contains__(self, element):
13          return element in self.elementi
14
15      def unija(self, drugi):
16          """Unija skupova: A ∪ B"""
17          return Skup(self.elementi | drugi.elementi)
18
19      def presjek(self, drugi):
20          """Presjek skupova: A ∩ B"""
21          return Skup(self.elementi & drugi.elementi)
22
23      def razlika(self, drugi):
24          """Razlika skupova: A \\ B"""
25          return Skup(self.elementi - drugi.elementi)
26
27      def simetricna_razlika(self, drugi):
28          """Simetrična razlika: A △ B"""
29          return Skup(self.elementi ^ drugi.elementi)
30
31      def je_podskup(self, drugi):
32          """Provjera je li skup podskup drugog: A ⊆ B"""
33          return self.elementi <= drugi.elementi
34
35  # Primjeri skupova
36  A = Skup([1, 2, 3, 4, 5])
37  B = Skup([3, 4, 5, 6, 7])
38
39  print("Osnovni skupovi:")
40  print(f"  A = {A}")
41  print(f"  B = {B}")
42  print("\nSkupovne operacije:")
43  print(f"  A ∪ B = {A.unija(B)}")
44  print(f"  A ∩ B = {A.presjek(B)}")
45  print(f"  A \\ B = {A.razlika(B)}")
46  print(f"  A △ B = {A.simetricna_razlika(B)}")
47  print("\nRelacije:")
48  print(f"  3 ∈ A: {'⊤' if 3 in A else '⊥'}")
49  print(f"  8 ∈ A: {'⊤' if 8 in A else '⊥'}")
50  print(f"  {{3, 4}} ⊆ A: {'⊤' if Skup([3, 4]).je_podskup(A) else '⊥'}")
```

Output

```
Osnovni skupovi:
  A = {1, 2, 3, 4, 5}
  B = {3, 4, 5, 6, 7}
```

```
Skupovne operacije:
  A ∪ B = {1, 2, 3, 4, 5, 6, 7}
  A ∩ B = {3, 4, 5}
  A \ B = {1, 2}
  A △ B = {1, 2, 6, 7}

Relacije:
  3 ∈ A: ⊤
  8 ∈ A: ⊥
  {3, 4} ⊆ A: ⊤
```

6.1.2 Partitivni skup i hijerarhija skupova

Partitivni skup P(A) je skup svih podskupova skupa A. Cantorov teorem pokazuje fundamentalnu činjenicu:

Za svaki skup A vrijedi: $\|P(A)\| > \|A\|$

Ovo vodi u beskonačnu hijerarhiju sve većih beskonačnosti:

6.1.3 Partitivni skup i hijerarhija skupova

Partitivni skup P(A) je skup svih podskupova skupa A. Cantorov teorem pokazuje fundamentalnu činjenicu:

Za svaki skup A vrijedi: $|P(A)| > |A|$

Ovo vodi u beskonačnu hijerarhiju sve većih beskonačnosti:

```python
def partitivni_skup(skup):
    """Generira partitivni skup (skup svih podskupova)."""
    elementi = list(skup.elementi)
    n = len(elementi)
    rezultat = []

    # Generiraj sve podskupove pomoću binarnog brojanja
    for i in range(2**n):
        podskup = set()
        for j in range(n):
            if i & (1 << j):
                podskup.add(elementi[j])
        rezultat.append(Skup(podskup))

    return rezultat

# Demonstracija partitivnog skupa
```

```python
S = Skup([1, 2, 3])
P_S = partitivni_skup(S)

print(f"Skup S = {S}")
print(f"Kardinalnost |S| = {len(S.elementi)}")
print("\nPartitivni skup P(S):")
for podskup in sorted(P_S, key=lambda x: (len(x.elementi), str(x))):
    print(f"  {podskup}")

print(f"\nKardinalnost |P(S)| = {len(P_S)} = 2^{len(S.elementi)}")
print(f"\nCantorov teorem: |P(S)| > |S| ✓")

# Iterirana primjena partitivnog skupa
print("\nRast kardinalnosti kroz iteracije:")
trenutni = Skup([])
for i in range(6):
    kard = len(trenutni.elementi)
    print(f"  P{''.join(['⁰¹²³⁴⁵⁶⁷⁸⁹'[int(d)] for d in str(i)])}(∅): ", end="")
    print(f"|{'P(' * i}∅{')'*i}| = {kard}")
    if i < 5:
        trenutni = Skup(range(2**kard))
```

Output

```
Skup S = {1, 2, 3}
Kardinalnost |S| = 3

Partitivni skup P(S):
  ∅
  {1}
  {2}
  {3}
  {1, 2}
  {1, 3}
  {2, 3}
  {1, 2, 3}

Kardinalnost |P(S)| = 8 = 2^3

Cantorov teorem: |P(S)| > |S| ✓

Rast kardinalnosti kroz iteracije:
  P⁰(∅): |∅| = 0
  P¹(∅): |P(∅)| = 1
  P²(∅): |P(P(∅))| = 2
  P³(∅): |P(P(P(∅)))| = 4
  P⁴(∅): |P(P(P(P(∅))))| = 16
  P⁵(∅): |P(P(P(P(P(∅)))))| = 65536
```

6.1.4 Bijektivna funkcija i kardinalnost

Cantor je definirao **jednakost kardinalnosti** kroz postojanje **bijekcije** (1-1 korespondencije) između skupova:

> Dva skupa imaju istu kardinalnost ako i samo ako postoji bijekcija između njih

Ovo omogućava usporedbu "veličina" beskonačnih skupova:

```python
class Bijekcija:
    """Predstavlja bijektivnu funkciju između skupova."""

    def __init__(self, naziv, funkcija, inverzna=None):
        self.naziv = naziv
        self.funkcija = funkcija
        self.inverzna = inverzna

    def je_bijekcija_na_uzorku(self, domena, kodomena):
        """Provjerava bijekciju na konačnom uzorku."""
        # Provjeri injektivnost
        slike = {}
        for x in domena:
            y = self.funkcija(x)
            if y in slike:
                return False  # Nije injektivna
            slike[y] = x

        # Za konačne skupove, provjeri surjektivnost
        if len(kodomena) <= len(domena):
            for y in kodomena:
                if y not in slike:
                    return False  # Nije surjektivna

        return True

# Primjeri bijekcija između beskonačnih skupova

# 1. ℕ → Parni brojevi
bij_parni = Bijekcija(
    "f: ℕ → 2ℕ",
    lambda n: 2 * n,
    lambda m: m // 2
)

# 2. ℕ → ℤ (cijeli brojevi)
def nat_to_int(n):
    """Mapira prirodne brojeve na cijele brojeve."""
    if n == 0:
        return 0
```

```
    elif n % 2 == 1:
        return -((n + 1) // 2)
    else:
        return n // 2

bij_cijeli = Bijekcija("g: ℕ → ℤ", nat_to_int)

# 3. ℕ → ℚ⁺ (pozitivni racionalni - Cantorov zigzag)
def cantor_zigzag(n):
    """Cantorov zigzag kroz racionalne brojeve."""
    # Jednostavna verzija za demonstraciju
    dijagonala = 1
    pozicija = n

    while pozicija >= dijagonala:
        pozicija -= dijagonala
        dijagonala += 1

    brojnik = pozicija + 1
    nazivnik = dijagonala - pozicija
    return (brojnik, nazivnik)

bij_racionalni = Bijekcija("h: ℕ → ℚ⁺", cantor_zigzag)

# Testiranje bijekcija
print("Provjera bijekcija:\n")

print("1. f: ℕ → 2ℕ (parni brojevi), f(n) = 2n")
uzorak_nat = list(range(10))
uzorak_parni = [bij_parni.funkcija(n) for n in range(5)]
print(f"   Bijekcija? ⊤")
print(f"   Primjeri: f(0)={bij_parni.funkcija(0)}, f(1)={bij_parni.funkcija(1)}, "
      f"f(2)={bij_parni.funkcija(2)}, f(3)={bij_parni.funkcija(3)},
      ↪  f(4)={bij_parni.funkcija(4)}")

print("\n2. g: ℕ → ℤ (cijeli brojevi)")
print(f"   Bijekcija? ⊤")
print(f"   Primjeri: g(0)={nat_to_int(0)}, g(1)={nat_to_int(1)}, "
      f"g(2)={nat_to_int(2)}, g(3)={nat_to_int(3)}, g(4)={nat_to_int(4)}")

print("\n3. h: ℕ → ℚ⁺ (pozitivni racionalni - Cantorov zigzag)")
print(f"   Bijekcija? ⊤")
print("   Prva mapiranja:")
for i in range(5):
    b, n = cantor_zigzag(i)
    print(f"     {i} → {b}/{n}")

print("\nZaključak: ℕ, 2ℕ, ℤ i ℚ⁺ imaju istu kardinalnost (ℵ₀)")
```

Output

```
Provjera bijekcija:

1. f: ℕ → 2ℕ (parni brojevi), f(n) = 2n
   Bijekcija? ⊤
   Primjeri: f(0)=0, f(1)=2, f(2)=4, f(3)=6, f(4)=8

2. g: ℕ → ℤ (cijeli brojevi)
   Bijekcija? ⊤
   Primjeri: g(0)=0, g(1)=-1, g(2)=1, g(3)=-2, g(4)=2

3. h: ℕ → ℚ⁺ (pozitivni racionalni - Cantorov zigzag)
   Bijekcija? ⊤
   Prva mapiranja:
     0 → 1/1
     1 → 1/2
     2 → 2/1
     3 → 1/3
     4 → 2/2

Zaključak: ℕ, 2ℕ, ℤ i ℚ⁺ imaju istu kardinalnost (ℵ₀)
```

6.1.5 Cantorova dijagonalna metoda

Cantorov najslavniji rezultat je dokaz da **realni brojevi nisu prebrojivi** - njihova kardinalnost je veća od prirodnih brojeva. Dijagonalna metoda je elegantan dokaz kontradikcijom:

> "Vidim to, ali ne vjerujem!" - pisao je Cantor Dedekindu o ovom otkriću

```python
import math

def dijagonalna_metoda_demo():
    """Demonstracija Cantorove dijagonalne metode."""

    # Simuliraj "popis" realnih brojeva između 0 i 1
    # (u stvarnosti je ovo nemoguće!)
    realni_popis = [
        math.pi - 3,        # 0.14159...
        math.e - 2,         # 0.71828...
        0.5,                # 0.50000...
        1/3,                # 0.33333...
        (math.sqrt(5)-1)/2, # 0.61803... (zlatni rez)
        math.sqrt(2) - 1,   # 0.41421...
        math.sqrt(3) - 1,   # 0.73205...
        math.pi/12,         # 0.26179...
        0.123456789012345,  # 0.12345...
        0.987654321098765   # 0.98765...
```

```
    ]

    print("Cantorova dijagonalna metoda")
    print("="*30)
    print("\nPretpostavimo da možemo popisati sve realne brojeve u [0,1]:\n")

    # Prikaži popis
    for i, broj in enumerate(realni_popis):
        print(f"r{chr(0x2080+i)} = {broj:.14f}...")

    # Izvuci dijagonalne elemente
    print("\nDijagonalni elementi (označeni):")
    dijagonala = []
    for i, broj in enumerate(realni_popis):
        # Pretvori u string decimala
        decimale = str(broj).split('.')[1] if '.' in str(broj) else '0'
        if i < len(decimale):
            digit = decimale[i]
            dijagonala.append(int(digit))
            print(f"  r{chr(0x2080+i)}[{i}] = {digit}")

    # Konstruiraj novi broj mijenjanjem dijagonale
    print("\nKonstrukcija novog broja d mijenjanjem dijagonale:")
    print(f"  Dijagonala: {''.join(map(str, dijagonala))}...")

    novi_broj_cifre = []
    for d in dijagonala:
        # Mijenjaj svaku cifru (npr. d → d+1 mod 10, izbjegni 9→0)
        nova_cifra = (d + 1) if d < 9 else 0
        novi_broj_cifre.append(nova_cifra)

    novi_broj_str = "0." + ''.join(map(str, novi_broj_cifre))
    print(f"  Novi broj d = {novi_broj_str}...")

    # Pokaži da se razlikuje od svakog broja na popisu
    print("\nProvjera: d se razlikuje od svakog r_i na i-toj poziciji:")
    for i in range(min(5, len(dijagonala))):
        print(f"  d ≠ r{chr(0x2080+i)} jer d[{i}]={novi_broj_cifre[i]} ≠ "
              f"r{chr(0x2080+i)}[{i}]={dijagonala[i]} ✓")

    print("\nKONTRADIKCIJA! Broj d ∈ [0,1] ali d nije na popisu.")
    print("Zaključak: Realni brojevi nisu prebrojivi.")

dijagonalna_metoda_demo()
```

Output

```
Cantorova dijagonalna metoda
==============================

Pretpostavimo da možemo popisati sve realne brojeve u [0,1]:
```

```
r_0 = 0.14159265358979...
r_1 = 0.71828182845905...
r_2 = 0.50000000000000...
r_3 = 0.33333333333333...
r_4 = 0.61803398874989...
r_5 = 0.41421356237310...
r_6 = 0.73205080756888...
r_7 = 0.26179938779915...
r_8 = 0.12345678901234...
r_9 = 0.98765432109877...

Dijagonalni elementi (označeni):
  r_0[0] = 1
  r_1[1] = 1
  r_3[3] = 3
  r_4[4] = 3
  r_5[5] = 3
  r_6[6] = 8
  r_7[7] = 8
  r_8[8] = 9
  r_9[9] = 0

Konstrukcija novog broja d mijenjanjem dijagonale:
  Dijagonala: 113338890...
  Novi broj d = 0.224449901...

Provjera: d se razlikuje od svakog r_i na i-toj poziciji:
  d ≠ r_0 jer d[0]=2 ≠ r_0[0]=1 ✓
  d ≠ r_1 jer d[1]=2 ≠ r_1[1]=1 ✓
  d ≠ r_2 jer d[2]=4 ≠ r_2[2]=3 ✓
  d ≠ r_3 jer d[3]=4 ≠ r_3[3]=3 ✓
  d ≠ r_4 jer d[4]=4 ≠ r_4[4]=3 ✓

KONTRADIKCIJA! Broj d ∈ [0,1] ali d nije na popisu.
Zaključak: Realni brojevi nisu prebrojivi.
```

6.1.6 Hijerarhija beskonačnosti

Cantor je otkrio **transfinitne kardinalne brojeve** - hijerarhiju beskonačnosti:

- $\aleph_0$ (alef-nula): kardinalnost prirodnih brojeva
- $2^{\aleph_0} = c$: kardinalnost realnih brojeva (kontinuum)
- $\aleph_1, \aleph_2, ...$: veće beskonačnosti

Hipoteza kontinuuma: Ne postoji kardinalnost između $\aleph_0$ i c.

```python
class KardinalniBroj:
    """Predstavlja kardinalni broj (konačan ili transfinitan)."""

    def __init__(self, simbol, opis, primjeri=None):
        self.simbol = simbol
        self.opis = opis
        self.primjeri = primjeri or []

    def __repr__(self):
        return self.simbol

# Definiraj kardinalne brojeve
alef_0 = KardinalniBroj("ℵ₀", "Prebrojiva beskonačnost",
                        ["ℕ", "ℤ", "ℚ", "Parni", "Prosti"])

kontinuum = KardinalniBroj("c", "Kardinalnost kontinuuma",
                           ["ℝ", "[0,1]", "P(ℕ)", "ℝ²"])

print("Hijerarhija kardinalnih brojeva")
print("="*32)

print("\nKonačne kardinalnosti:")
print("  |∅| = 0")
print("  |{a}| = 1")
print("  |{a,b,c}| = 3")

print("\nPrebrojive beskonačnosti (kardinalnost ℵ₀):")
prebrojivi = [
    ("ℕ", "prirodni brojevi"),
    ("ℤ", "cijeli brojevi"),
    ("ℚ", "racionalni brojevi"),
    ("Parni", "parni brojevi"),
    ("Prosti", "prosti brojevi")
]
for skup, opis in prebrojivi:
    print(f"  |{skup}| = ℵ₀           ({opis})")

print("\nNeprebrojive beskonačnosti:")
neprebrojivi = [
    ("ℝ", "$2^ℵ₀ = c$", "realni brojevi"),
    ("[0,1]", "$c$", "interval [0,1]"),
    ("P(ℕ)", "$2^ℵ₀$", "partitivni skup od ℕ"),
    ("ℝ²", "$c$", "ravnina"),
    ("ℝ^ℝ", "$2^c$", "funkcije ℝ→ℝ")
]
for skup, kard, opis in neprebrojivi:
    print(f"  |{skup}| = {kard:<10} ({opis})")


print("\nHipoteza kontinuuma (CH):")
print("  CH tvrdi: ℵ₁ = 2^ℵ₀ = c")
```

```
52 print("  Status: NEZAVISNA od ZFC aksioma (Cohen & Gödel)")
```

Output

```
Hijerarhija kardinalnih brojeva
================================

Konačne kardinalnosti:
  |∅| = 0
  |{a}| = 1
  |{a,b,c}| = 3

Prebrojive beskonačnosti (kardinalnost ℵ0):
  |ℕ| = ℵ0            (prirodni brojevi)
  |ℤ| = ℵ0            (cijeli brojevi)
  |ℚ| = ℵ0            (racionalni brojevi)
  |Parni| = ℵ0           (parni brojevi)
  |Prosti| = ℵ0           (prosti brojevi)

Neprebrojive beskonačnosti:
  |ℝ| = $2^ℵ0 = c$ (realni brojevi)
  |[0,1]| = $c$        (interval [0,1])
  |P(ℕ)| = $2^ℵ0$     (partitivni skup od ℕ)
  |ℝ²| = $c$         (ravnina)
  |ℝ^ℝ| = $2^c$       (funkcije ℝ→ℝ)

Hipoteza kontinuuma (CH):
  CH tvrdi: ℵ1 = 2^ℵ0 = c
  Status: NEZAVISNA od ZFC aksioma (Cohen & Gödel)
```

6.1.7 Russellov paradoks i kriza osnova

Cantorova naivna teorija skupova dovela je do paradoksa. Najpoznatiji je **Russellov paradoks** (1901):

> Neka je $R = x : x \notin x$ skup svih skupova koji ne sadrže sami sebe. Pitanje: Je li $R \in R$?

Ovaj paradoks pokazuje da ne možemo slobodno formirati skupove - potrebni su aksiomi!

```
1 def russellov_paradoks_demo():
2     """Demonstracija Russellovog paradoksa."""
3
4     print("Russellov paradoks - simulacija")
5     print("="*32)
```

```
    print("\nPokušajmo definirati skup R = {x : x ∉ x}")

    # Primjeri običnih skupova
    print("\nObični skupovi (ne sadrže sami sebe):")
    obicni = [
        ("Skup brojeva {1,2,3}", "ne sadrži sebe"),
        ("Skup slova {a,b,c}", "ne sadrži sebe"),
        ("Prazan skup ∅", "ne sadrži sebe")
    ]
    for skup, status in obicni:
        print(f"  {skup}: {status} ✓")

    print("\nNeobični skupovi (hipotetski sadrže sami sebe):")
    neobicni = [
        ("Skup svih skupova", "sadrži sebe (?)"),
        ("Skup apstraktnih koncepata", "možda sadrži sebe (?)")
    ]
    for skup, status in neobicni:
        print(f"  {skup}: {status}")

    # Analiza paradoksa
    print("\nAnaliza paradoksa:")
    print("  Pretpostavka 1: R ∈ R")
    print("    → Po definiciji R, ako R ∈ R tada R ∉ R")
    print("    → KONTRADIKCIJA! ⊥")

    print("\n  Pretpostavka 2: R ∉ R")
    print("    → Po definiciji R, ako R ∉ R tada R ∈ R")
    print("    → KONTRADIKCIJA! ⊥")

    print("\nZaključak: R ne može postojati kao skup!")

    # Rješenja
    print("\nRješenja paradoksa:")
    print("  1. Teorija tipova (Russell): hijerarhija tipova")
    print("  2. ZFC aksiomi (Zermelo-Fraenkel): ograničena konstrukcija")
    print("  3. NBG teorija (von Neumann): razlika skup/klasa")

    print("\nFilozofske implikacije:")
    print("  • Granice samoreferencijalnosti")
    print("  • Nemogućnost \"skupa svih skupova\"")
    print("  • Potreba za formalnim aksiomima")
    print("  • Gödel: inherentna nepotpunost formalnih sustava")

russellov_paradoks_demo()
```

Output

```
Russellov paradoks - simulacija
================================
```

```
Pokušajmo definirati skup R = {x : x ∉ x}

Obični skupovi (ne sadrže sami sebe):
  Skup brojeva {1,2,3}: ne sadrži sebe ✓
  Skup slova {a,b,c}: ne sadrži sebe ✓
  Prazan skup ∅: ne sadrži sebe ✓

Neobični skupovi (hipotetski sadrže sami sebe):
  Skup svih skupova: sadrži sebe (?)
  Skup apstraktnih koncepata: možda sadrži sebe (?)

Analiza paradoksa:
  Pretpostavka 1: R ∈ R
    → Po definiciji R, ako R ∈ R tada R ∉ R
    → KONTRADIKCIJA! ⊥

  Pretpostavka 2: R ∉ R
    → Po definiciji R, ako R ∉ R tada R ∈ R
    → KONTRADIKCIJA! ⊥

Zaključak: R ne može postojati kao skup!

Rješenja paradoksa:
  1. Teorija tipova (Russell): hijerarhija tipova
  2. ZFC aksiomi (Zermelo-Fraenkel): ograničena konstrukcija
  3. NBG teorija (von Neumann): razlika skup/klasa

Filozofske implikacije:
  • Granice samoreferencijalnosti
  • Nemogućnost "skupa svih skupova"
  • Potreba za formalnim aksiomima
  • Gödel: inherentna nepotpunost formalnih sustava
```

6.1.8 Filozofske implikacije beskonačnosti

Ontološki status beskonačnosti

Cantorova otkrića pokrenula su duboka filozofska pitanja:

1. **Platonizam**: Postojanje matematičkih objekata nezavisno od uma
2. **Konstruktivizam**: Samo konstruktivno definirani objekti postoje
3. **Formalizam**: Matematika kao igra simbola bez ontološkog značenja

Aktualna vs. potencijalna beskonačnost

- **Aristotel**: Samo potencijalna beskonačnost (proces bez kraja)
- **Cantor**: Aktualna beskonačnost kao završen totalitet

"Beskonačnost je ponor u koji tone sve naše misli" - Musil

```python
def filozofija_beskonacnosti():
    """Ilustracija filozofskih aspekata beskonačnosti."""

    print("Ilustracija razlike između potencijalne i aktualne beskonačnosti")
    print("="*65)

    # Potencijalna beskonačnost
    print("\nPOTENCIJALNA BESKONAČNOST (Aristotel):")
    print("  Proces brojanja: ", end="")
    for i in range(1, 11):
        print(f"{i}, ", end="")
    print("...")
    print("  → Uvijek možemo dodati još jedan")
    print("  → Nikad ne dostižemo \"kraj\"")
    print("  → Beskonačnost kao mogućnost")

    # Aktualna beskonačnost
    print("\nAKTUALNA BESKONAČNOST (Cantor):")
    print("  Skup ℕ = {0, 1, 2, 3, ...} postoji kao cjelina")
    print("  → Možemo govoriti o |ℕ| = ℵ₀")
    print("  → Možemo uspoređivati beskonačnosti")
    print("  → Beskonačnost kao objekt")

    # Zenov paradoks
    print("\nZenov paradoks - Ahilej i kornjača:")
    print("="*37)
    print("Kornjača ima prednost od 100m, Ahilej trči 10x brže.\n")

    print("Koraci:")
    ahilej_poz = 0
    kornjaca_poz = 100
    brzina_omjer = 10

    suma = 0
    for korak in range(1, 6):
        udaljenost = kornjaca_poz - ahilej_poz
        ahilej_poz = kornjaca_poz
        kornjaca_poz += udaljenost / brzina_omjer
        suma += udaljenost

        print(f"  Korak {korak}: Ahilej na {ahilej_poz:.2f}m, "
              f"kornjača na {kornjaca_poz:.2f}m")
```

```
    print("  ...")

    # Matematička analiza
    print(f"\nSuma beskonačnog reda: 100 + 10 + 1 + 0.1 + ... = 111.11...")
    granica = 100 * (1 / (1 - 1/brzina_omjer))
    print(f"Konvergira na: {granica:.2f}m")

    print("\nCantorovo rješenje: Aktualna beskonačnost omogućava")
    print("tretiranje beskonačnog procesa kao završene cjeline.")

filozofija_beskonacnosti()
```

Output

```
Ilustracija razlike između potencijalne i aktualne beskonačnosti
======================================================================

POTENCIJALNA BESKONAČNOST (Aristotel):
  Proces brojanja: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ...
  → Uvijek možemo dodati još jedan
  → Nikad ne dostižemo "kraj"
  → Beskonačnost kao mogućnost

AKTUALNA BESKONAČNOST (Cantor):
  Skup ℕ = {0, 1, 2, 3, ...} postoji kao cjelina
  → Možemo govoriti o |ℕ| = ℵ₀
  → Možemo uspoređivati beskonačnosti
  → Beskonačnost kao objekt

Zenov paradoks - Ahilej i kornjača:
====================================
Kornjača ima prednost od 100m, Ahilej trči 10x brže.

Koraci:
  Korak 1: Ahilej na 100.00m, kornjača na 110.00m
  Korak 2: Ahilej na 110.00m, kornjača na 111.00m
  Korak 3: Ahilej na 111.00m, kornjača na 111.10m
  Korak 4: Ahilej na 111.10m, kornjača na 111.11m
  Korak 5: Ahilej na 111.11m, kornjača na 111.11m
  ...

Suma beskonačnog reda: 100 + 10 + 1 + 0.1 + ... = 111.11...
Konvergira na: 111.11m

Cantorovo rješenje: Aktualna beskonačnost omogućava
tretiranje beskonačnog procesa kao završene cjeline.
```

6.1.9 Praktična primjena: Moć skupova u programiranju

Implementirajmo sustav za rad s beskonačnim skupovima kroz "lijene" evaluacije:

```python
class BeskonacniSkup:
    """Predstavlja beskonačni skup kroz generator funkciju."""

    def __init__(self, generator_func, naziv):
        self.generator = generator_func
        self.naziv = naziv
        self._cache = {}

    def element(self, n):
        """Vraća n-ti element skupa (s cachiranjem)."""
        if n not in self._cache:
            for i, val in enumerate(self.generator()):
                if i not in self._cache:
                    self._cache[i] = val
                if i == n:
                    return val
        return self._cache[n]

    def prvih(self, n):
        """Vraća prvih n elemenata."""
        rezultat = []
        for i, val in enumerate(self.generator()):
            if i >= n:
                break
            rezultat.append(val)
        return rezultat

# Generatori za različite beskonačne skupove

def prirodni_brojevi():
    """Generator prirodnih brojeva."""
    n = 0
    while True:
        yield n
        n += 1

def prosti_brojevi():
    """Generator prostih brojeva (Eratostenovo sito)."""
    yield 2
    prosti = [2]
    kandidat = 3
    while True:
        je_prost = True
        for p in prosti:
            if p * p > kandidat:
                break
            if kandidat % p == 0:
```

```
                je_prost = False
                break
        if je_prost:
            prosti.append(kandidat)
            yield kandidat
        kandidat += 2

def fibonacci():
    """Generator Fibonaccijevih brojeva."""
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

def racionalni_pozitivni():
    """Generator pozitivnih racionalnih (Cantorov zigzag)."""
    from math import gcd

    dijagonala = 1
    while True:
        for brojnik in range(1, dijagonala + 1):
            nazivnik = dijagonala + 1 - brojnik
            if gcd(brojnik, nazivnik) == 1:  # Samo reducirani razlomci
                yield (brojnik, nazivnik)
        dijagonala += 1

# Demonstracija
print("Rad s beskonačnim skupovima")
print("="*28)

nat = BeskonacniSkup(prirodni_brojevi, "ℕ")
print("\nPrirodni brojevi:")
print(f"  Prvih 10: {nat.prvih(10)}")
print(f"  100. element: {nat.element(99)}")

primes = BeskonacniSkup(prosti_brojevi, "Prosti")
print("\nProsti brojevi:")
print(f"  Prvih 10: {primes.prvih(10)}")
print(f"  50. prosti: {primes.element(49)}")

fib = BeskonacniSkup(fibonacci, "Fibonacci")
print("\nFibonaccijev niz:")
print(f"  Prvih 15: {fib.prvih(15)}")

rationals = BeskonacniSkup(racionalni_pozitivni, "ℚ⁺")
print("\nCantorova dijagonala kroz ℚ⁺:")
prvih_10_rac = rationals.prvih(10)
print(f"  Prvih 10: [{', '.join(f'{b}/{n}' for b, n in prvih_10_rac)}]")

# Vizualizacija hipoteze kontinuuma
print("\nHipoteza kontinuuma vizualizacija:")
print("  $|ℕ| = ℵ₀$")
print("  $|P(ℕ)| = 2^ℵ₀ = c$")
```

```
print("  Pitanje: Postoji li $X$ takav da $ℵ₀ < |X| < c$?")
print("  CH kaže: NE - između nema ništa!")
```

Output

```
Rad s beskonačnim skupovima
============================

Prirodni brojevi:
  Prvih 10: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  100. element: 99

Prosti brojevi:
  Prvih 10: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
  50. prosti: 229

Fibonaccijev niz:
  Prvih 15: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]

Cantorova dijagonala kroz ℚ⁺:
  Prvih 10: [1/1, 1/2, 2/1, 1/3, 3/1, 1/4, 2/3, 3/2, 4/1, 1/5]

Hipoteza kontinuuma vizualizacija:
  $|ℕ| = ℵ₀$
  $|P(ℕ)| = 2^ℵ₀ = c$
  Pitanje: Postoji li $X$ takav da $ℵ₀ < |X| < c$?
  CH kaže: NE - između nema ništa!
```

6.1.10 Prijedlozi za daljnje istraživanje

Za produbljivanje razumijevanja teorije skupova kroz praktične Python zadatke, predlažemo sljedeće vježbe prikladne za studente:

Implementacija aksioma ZFC

Napišite klase koje simuliraju osnovne aksiome Zermelo-Fraenkel teorije skupova: aksiom ekstenzionalnosti, aksiom para, aksiom unije, aksiom partitivnog skupa. Pokažite kako svaki aksiom ograničava konstrukciju skupova.

Ordinalni brojevi

Implementirajte ordinalne brojeve koristeći von Neumannovu konstrukciju ($0 = \emptyset, 1 = \emptyset, 2 = \emptyset, \emptyset, ...$). Definirajte ordinalno zbrajanje i množenje. Ilustrirajte razliku između kardinalnih i ordinalnih brojeva.

Schröder-Bernsteinov teorem

Dokažite kroz kod: ako postoji injekcija $f : A \to B$ i injekcija $g : B \to A$, tada postoji bijekcija između A i B. Testirajte na konkretnim primjerima skupova.

Hilbertov hotel

Simulirajte Hilbertov paradoks beskonačnog hotela. Implementirajte scenarije: novi gost u punom hotelu, beskonačno novih gostiju, beskonačno autobusa s beskonačno gostiju. Vizualizirajte preslagivanja soba.

Cantorova funkcija (Vražje stepenice)

Konstruirajte Cantorovu funkciju - kontinuiranu funkciju koja je gotovo svugdje konstantna ali ipak raste od 0 do 1. Vizualizirajte je pomoću matplotlib.

Fraktalna dimenzija

Implementirajte Cantorov skup (iterativno uklanjanje srednje trećine). Izračunajte njegovu Hausdorffovu dimenziju (log 2 / log 3). Generirajte Cantorovu prašinu u 2D.

Aksiom izbora - simulacija

Simulirajte situacije gdje je aksiom izbora potreban: Banach-Tarskijev paradoks (konceptualno), well-ordering princip, Zornova lema. Ilustrirajte kontroverznost aksioma.

Gödel brojevi

Implementirajte Gödelovo numeriranje - preslikavanje formula u prirodne brojeve. Pokažite kako se meta-matematika svodi na aritmetiku. Ilustrirajte ideju nepotpunosti.

Transfinitna indukcija

Napišite funkciju koja koristi transfinitnu indukciju za definiranje funkcija na ordinalima. Primjer: Ackermanova funkcija generalizirana na ordinale.

Forsing metoda - konceptualna simulacija

Stvorite jednostavnu simulaciju Cohen forcing metode. Pokažite kako se mogu "dodati" novi skupovi postojećem modelu teorije skupova. Ilustrirajte nezavisnost hipoteze kontinuuma.

Svaki zadatak postupno gradi razumijevanje dubokih koncepata teorije skupova kroz praktično programiranje, omogućavajući studentima da eksperimentiraju s apstraktnim idejama i razviju intuiciju za rad s beskonačnostima.

6.1.11 Zaključak

Kroz ovu implementaciju istražili smo temeljne koncepte Cantorove teorije skupova:

1. **Osnovne skupovne operacije** kao temelj matematike
2. **Hijerarhiju beskonačnosti** kroz kardinalne brojeve
3. **Dijagonalnu metodu** koja otkriva neprebrojivost realnih brojeva
4. **Paradokse** koji pokazuju granice naivnog pristupa

Cantorova naslijeđe transformiralo je matematiku i filozofiju:

> "Nitko nas neće istjerati iz raja koji je Cantor stvorio za nas" - David Hilbert

Ipak, Gödelovi teoremi nepotpunosti pokazuju da čak ni u ovom "raju" ne možemo dokazati sve istine. Teorija skupova otkriva da je beskonačnost ne samo matematički koncept, već prozor u fundamentalne granice ljudskog razumijevanja.

Kroz praktično programiranje otkrivamo da rad s beskonačnošću zahtijeva pažljivo balansiranje između intuicije i formalizma, između filozofske kontemplacije i matematičke strogosti. Cantorova vizija beskonačnosti ostaje jedna od najdubljih intelektualnih avantura čovječanstva.

> "Vidim to, ali ne vjerujem!" - možda je upravo ta nevjerica pred beskonačnošću ono što nas čini ljudima.