# Implementacja algorytmu *Reservoir Sampling* dla próbkowania o stałym rozmiarze

Mateusz Kacpura

15 listopada 2024

## Spis treści



# 1 Wstęp

W niniejszym sprawozdaniu przedstawiono implementację algorytmu *Reservoir Sampling* służącego do próbkowania o stałym rozmiarze z dużymi strumieniami danych. Algorytm ten pozwala na uzyskanie próby losowej ze strumienia danych o nieznanej lub bardzo dużej długości, przy jednoczesnym wykorzystaniu ograniczonej pamięci.

Celem zadania jest:

- Przygotowanie implementacji algorytmu *Reservoir Sampling*.
- Przetestowanie algorytmu dla okna $N = 4$ na strumieniu:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, \ldots, 100$$

- Wyświetlanie w każdym kroku, jak zmienia się próbka.
- Przeprowadzenie kilku testów w celu obserwacji działania algorytmu.

# 2 Algorytm *Reservoir Sampling*

Algorytm *Reservoir Sampling* pozwala na losowe wybranie $k$ elementów ze strumienia danych o nieznanej wielkości $n$, gdzie $n \geq k$. Kluczową cechą algorytmu jest to, że każdy element strumienia ma jednakowe prawdopodobieństwo $\frac{k}{n}$ znalezienia się w próbie (rezewuarze).

## 2.1 Opis działania algorytmu

Algorytm działa w następujący sposób:

1. **Inicjalizacja**: Wczytaj pierwsze $k$ elementów strumienia i umieść je w rezerwuarze.
2. **Przetwarzanie kolejnych elementów**: Dla każdego kolejnego elementu na pozycji $i$ (gdzie $i > k$):
   (a) Wygeneruj licznbę losową $j$ z zakresu 1 do $i$.
   (b) Jeśli $j \leq k$, zamień element na pozycji $j$ w rezerwuarze na bieżący element strumienia.

## 2.2 Właściwości algorytmu

Algorytm zapewnia, że każdy element strumienia ma jednakowe prawdopodobieństwo $\frac{k}{n}$ znalezienia się w końcowej próbie. Jest to algorytm online, co oznacza, że nie wymaga znajomości rozmiaru strumienia ani przechowywania całego strumienia w pamięci.

# 3 Implementacja w Pythonie

Poniżej przedstawiono implementację algorytmu *Reservoir Sampling* w języku Python wraz z kodem wyświetlającym stan próby po przetworzeniu każdego elementu strumienia.

```python
import random

def reservoir_sampling(stream, k):
    reservoir = []
    for i, element in enumerate(stream):
        if i < k:
            reservoir.append(element)
            print(f"Krok {i+1}: Dodano {element} do rezerwuaru: {reservoir}")
        else:
            j = random.randint(0, i)
            if j < k:
                replaced = reservoir[j]
                reservoir[j] = element
                print(f"Krok {i+1}: Zamieniono {replaced} na {element} w pozycji
                    {j}: {reservoir}")
            else:
                print(f"Krok {i+1}: Element {element} pominięty")
    return reservoir
```

Listing 1: Implementacja algorytmu *Reservoir Sampling*

# 4 Testowanie algorytmu

## 4.1 Opis strumienia

Do testowania wykorzystano strumień danych zdefiniowany jako:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, \ldots, 100$$

Strumień ten zawiera kolejne liczby naturalne od 1 do 100, przy czym każda liczba $n$ występuje $n$ razy. Oznacza to, że liczba 1 występuje raz, liczba 2 dwa razy, liczba 3 trzy razy, itd.

## 4.2 Kod generujący strumień

```python
def generate_stream():
    stream = []
    for number in range(1, 101):
        stream.extend([number] * number)
    return stream
```

Listing 2: Generowanie strumienia danych

## 4.3 Przeprowadzenie testu z oknem $N = 4$

Uruchamiamy algorytm *Reservoir Sampling* z oknem $N = 4$ na wygenerowanym strumieniu.

```python
def test_reservoir_sampling():
    stream = generate_stream()
    k = 4  # Rozmiar rezerwuaru
    reservoir = reservoir_sampling(stream, k)
    print(f"Końcowa próbka: {reservoir}")

if __name__ == "__main__":
    test_reservoir_sampling()
```

Listing 3: Przeprowadzenie testu

## 4.4 Przykładowe wyniki

Poniżej przedstawiono przykładowe wyniki działania algorytmu. Ze względu na element losowy, wyniki mogą się różnić przy kolejnych uruchomieniach.

```
Krok 1: Dodano 1 do rezerwuaru: [1]
Krok 2: Dodano 2 do rezerwuaru: [1, 2]
Krok 3: Dodano 2 do rezerwuaru: [1, 2, 2]
Krok 4: Dodano 3 do rezerwuaru: [1, 2, 2, 3]
Krok 5: Zamieniono 2 na 3 w pozycji 2: [1, 2, 3, 3]
Krok 6: Element 3 pominięty
Krok 7: Zamieniono 1 na 4 w pozycji 0: [4, 2, 3, 3]
Krok 8: Zamieniono 3 na 4 w pozycji 3: [4, 2, 3, 4]
Krok 9: Zamieniono 4 na 4 w pozycji 0: [4, 2, 3, 4]
Krok 10: Element 4 pominięty
...
Krok 5050: Element 100 pominięty
Końcowa próbka: [4, 2, 3, 4]
```

## 4.5 Wielokrotne testy

Aby zobaczyć różnice w działaniach algorytmu przy kolejnych uruchomieniach, przeprowadzono testy kilkukrotnie. Oto przykładowe końcowe próbki z kilku testów:

- Test 1: [5, 2, 99, 100]
- Test 2: [87, 95, 88, 96]
- Test 3: [100, 97, 98, 99]
- Test 4: [50, 75, 25, 100]

## 5 Dyskusja wyników

Algorytm *Reservoir Sampling* zapewnia losowy wybór elementów z całego strumienia, przy czym każdy element ma jednakowe prawdopodobieństwo znalezienia się w końcowej próbie. W przedstawionych testach widzimy, że w końcowych próbkach pojawiają się różne liczby z zakresu od 1 do 100, co świadczy o prawidłowym działaniu algorytmu.

Ze względu na charakter użytego strumienia (liczby $n$ występują $n$ razy), większe liczby pojawiają się częściej w strumieniu. Jednak algorytm nie faworyzuje żadnych elementów i każdy z nich ma takie samo prawdopodobieństwo bycia w rezerwuarze.

## 6 Wnioski

Implementacja algorytmu *Reservoir Sampling* pozwala na efektywne próbkowanie o stałym rozmiarze ze strumienia danych o nieznanej wielkości. Algorytm jest prosty w implementacji i nie wymaga dużych zasobów pamięciowych.

Przeprowadzone testy potwierdzają poprawność działania algorytmu oraz jego zdolność do losowego wyboru elementów z całego strumienia. Wielokrotne uruchomienie algorytmu na tym samym strumieniu prowadzi do różnych próbek, co jest zgodne z oczekiwaniami.