Chapter 4
Algorithms and Conjectures

ABSTRACT

The chapter deals with combinatorial algorithms and their role in analyzing discrete structures such as graphs, sets, and permutations. The sieve of Eratosthenes is introduced to find prime numbers, highlighting the inclusion-exclusion principle to count the elements of a set without certain divisors. Next, we analyze the Atkin sieve, which uses modular congruences to identify prime numbers more efficiently. Another part of the chapter explores the distribution of twin primes and proposes a statistical analysis of their distances through combinatorial models. The Ulam spiral is discussed, revealing interesting patterns in the former distribution. Finally, mathematical conjectures are explored, such as the Erdős–Szekeres conjecture on the presence of convex polygons between sets of points and hypotheses on the distribution of permutations with prime jumps, and a simple example starting from the Erdős–Ginzburg–Ziv theorem is exposed. The chapter highlights the link between combinatorics, algorithms and conjectures in mathematical progress.

ALGORITHMS

Programming computer languages allows you to reflect on the ability to think

Nicholas Negroponte

Previously, we emphasised the importance of combinatorial algorithms. These algorithms are designed to solve problems involving discrete structures, such as graphs, sets, strings, or permutations, and they are essential in many fields of computer science and applied mathematics (Graham et al., 1989; Cameron, 1994).

A simple example may clarify.

The sieve of Eratosthenes, one of the oldest methods for identifying prime numbers, exemplifies an algorithm rooted in arithmetic logic yet closely intertwined with combinatorics. More advanced extensions of this sieve have been developed, revealing deep connections to principles of enumerative combinatorics. In general, combinatorial sieves operate through a structured process of inclusion and exclusion, closely reflecting the principle of inclusion-exclusion.

The fundamental idea is to begin with a universal set of elements and progressively eliminate those that fail to satisfy specific desired properties. Combinatorics, for its part, is primarily concerned with counting such elements, and sieve methods contribute meaningfully in this context by offering systematic tools for filtering and selecting subsets—such as in the generation of prime numbers (Berman & Fryer, 2014). It is worth emphasizing that the problem of counting prime numbers is of significant interest in both combinatorics and number theory.

The general sieve constitutes a powerful generalisation of the sieve of Eratosthenes, relying, as previously mentioned, on the inclusion-exclusion principle, a foundational concept in enumerative combinatorics that will be discussed in greater detail later. Instead of removing multiples directly, it focuses on counting the number of elements in a set that are not divisible by any of a given set of prime numbers P. In summary, given a set of integers and a set of primes P, the inclusion-exclusion principle is applied to determine how many elements are not divisible by any prime in P.

DOI: 10.4018/979-8-3373-3089-1.ch004

Program 14 General Sieve

```
from math import sqrt, prod
def general_sieve(limit):
def count_multiples(primes):
return limit // prod(primes)
# Calculate the square root of the limit
root = int(sqrt(limit))
# Initialize the list of prime numbers up to the square root of the limit
primes = [True] * (root + 1)
primes[0] = primes[1] = False # 0 and 1 are not prime numbers
# Implement the Sieve of Eratosthenes to find prime numbers up to the square root
for i in range(2, int(sqrt(root)) + 1):
if primes[i]:
for j in range(i*i, root + 1, i):
primes[j] = False
base_primes = [i for i in range(len(primes)) if primes[i]]
# Apply the principle of inclusion-exclusion
count = limit
for k in range(1, len(base_primes) + 1):
for combination in combinations(base_primes, k):
if k % 2 == 1:
count -= count_multiples(combination)
else:
count += count_multiples(combination)
return count - 1 # Subtract 1 to exclude the number 1
def combinations(iterable, r):
pool = tuple(iterable)
n = len(pool)
if r > n:
return
indices = list(range(r))
yield tuple(pool[i] for i in indices)
while True:
for i in reversed(range(r)):
if indices[i] != i + n - r:
break
else:
return
indices[i] += 1
for j in range(i + 1, r):
indices[j] = indices[j - 1] + 1
yield tuple(pool[i] for i in indices)
limit = int(input("Upper limit"))
prime_count = general_sieve(limit)
print(f"The number of prime numbers up to {limit} is: {prime_count}")
```

Related Content