CS 529: Advanced Data Structures & Algorithms
Assignment 8: Number Theory and NP Algorithms

Andrew Struthers

April 26, 2024

Contents

1 Number Theory - Section 11.1 & 11.2

Number theory is a branch of mathematics that deals with the properties and relationships of integer numbers. It encompasses various concepts and theorems that form the foundation for understanding the structure of numbers. Some of the key concepts covered in the textbook are prime and composite numbers, greatest common divisor (GCD), prime factorization, least common mulitple (LCM), and Euclid's algorithm.

1.1 Prime and Composite Numbers

A prime number k is prime if and only if $k \in \mathbb{Z}, k > 1$, and k has no positive divisors other than 1 and k. Some examples of prime numbers are 2, 3, 4, 5, and 11. Composite numbers are integers greater than 1 that can be factored into smaller positive integers other than 1 and themselves. For example, 4 is composite because $4 = 2 \cdot 2$. More examples include 6, 8, 9, and 10. Prime numbers are fundamental in number theory because of the **fundamental theorem of arithmetic**, that states every natural number greater than 1 is either a prime number or can be factorized as a product of prime numbers, and this factorization is unique. This becomes very important in areas of cybersecurity and cryptography, because prime factors are used frequently when encoding and decoding messages. Composite numbers can be factored into prime numbers. For example, $10 = 5 \cdot 2$, and both 2 and 5 are prime numbers.

1.2 Euclidean Division

It can be shown that for any pair of positive integers $a, b \in \mathbb{Z}^+$, there exists positive integers $q, r \in \mathbb{Z}^+$ such that

$$a = qb + r \tag{1}$$

where $0 \leq r < b$. q is called the "quotient" and r is called the "remainder". Because this is true for any pair, the same holds for every pair of positive integers. A well known algorithm for calculating the remainder is long division.

1.3 Greatest Common Divisor (GCD)

The **greatest common divisor (GCD)** of two integers is the largest positive integer that divides each of the integers without a remainder. An example of the GCD is $\text{GCD}(8, 12)$. The largest positive integer that divides both 8 and 12 is 4, so $\text{GCD}(8, 12) = 4$. The GCD can be calculated directly using various methods. One method that can calculate the GCD directly is prime factorization. Some other algorithms include Euclid's algorithm, which will be discussed later, and the binary algorithm. The GCD is used in various ares of math and computer science, such as simplifying fractions and in generating random numbers. Euclid's algorithm is one of the widely used methods for finding the GCD, and this algorithm involves repeatedly applying the division algorithm until the remainder is zero. The GCD of two numbers is invariant under multiplication by any nonzero constant.

1.4 Prime Factorization

As we mentioned earlier, every composite number an be expressed as a unique product of prime numbers, known as its prime factorization. The **fundamental theorem of arithmetic** states that every integer greater than 1 can be factored into prime numbers in exactly one way, discarding the order of the factors. For instance, the prime factorization of 38 is $2^1 \cdot 19^1$, where both 2 and 19 are primes. This tells us that 38 can be represented by the product of one 2 and one 19. For prime numbers, the prime factorization is $1^1 \cdot k^1$ where k is the prime number. Prime factorization is used in many areas, where some of the simpler use cases are for computing the greatest common divisor and least common multiple. The prime factorization can also help simplify radicals as well as factor polynomials. Prime factorization is also a key part of many cryptographic algorithms such as RSA encryption.

1.5 Least Common Multiple (LCM)

The **least common multiple (LCM)** of two integers a and b is the smallest positive integer that is divisible by both a and b. This is very similar in concept to the GCD, but here we are finding a number that both a and b divide, instead

of a number that a and b are both divisible by. For example, $\text{LCM}(5, 12) = 60$. The LCM is useful in various places. Specifically, it is useful when listing multiples and performing prime factorization. We can take the highest power of each prime factor that appears in either number to find the LCM Interestingly, the $\text{LCM}(a, b) = \frac{|a \cdot b|}{\text{GCD}(a,b)}$. The LCM is used when we want to add or subtract fractions, when we want to compute the period of a repeating decimal, and in the solution of Diophantine equations.

1.5.1 Diophantine Equations

Diophantine equations are a specific type of equations in mathematics, named after the ancient mathematician Diophantus of Alexandria. These equations are usually polynomial equations with two or more unknowns and integer coefficients. One part of Diophantine equations that makes them unique above other polynomial equations is that we are only interested in integer solutions. For example, consider the equation $3x + 7y = 1$. This is a Diophantine equation because we are looking for integer values of x and y that satisfy the equation.

Diophantine equations can be linear or exponential. A linear Diophantine equation equates to a constant the sum of two or more monomials, each of degree one. An exponential Diophantine equation is one in which unknowns can appear in exponents. Diophantine problems often have fewer equations than unknowns and involve finding integers that solve simultaneously all equations. Because these equations and systems of equations usually algebraic curves, surfaces, and sets, the study focused on this type of problem inside of algebraic geometry is called Diophantine geometry.

The mathematical study of Diophantine problems that Diophantus initiated is now called Diophantine analysis. While individual equations present a kind of puzzle and have been considered throughout history, the formulation of general theories of Diophantine equations was an achievement of the twentieth century. One of the most famous examples of a Diophantine equation is Fermat's Last Theorem, which states that there are no three positive integers a, b, and c that satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 21. Another famous example is Pell's equation, $x^2 - ny^2 = \pm 1$, named after the mathematician John Pell. This equation is a specific type of Diophantine equation and has been studied extensively due to its appearance in various branches of mathematics. Diophantine equations have a wide range of applications in areas such as cryptography, number theory, and algebraic geometry.

1.6 Euclid's Algorithm

Euclid's algorithm is a method for finding the greatest common divisor of two integers. It is based on the principle that the GCD of two numbers remains the same if the large number is replaced by the difference between the two numbers. This means that the greatest common divisor of two numbers also divides their difference. We can iteratively apply the division algorithm until the remainder becomes zero. The GCD is then found to be the last non-zero remainder. Euclid's algorithm is very efficient and allows areas of computer science and math to operate quickly. This is especially useful in cryptography when encoding or decoding large messages.

1.7 Summary

Number theory is very important in various fields including cryptography. The concepts of prime and composite numbers, GCD and LCM, prime factorization, and Euclid's algorithm provide a good foundation for understanding properties and relationships of integer numbers. Prime numbers are very significant in cryptography, where they are used in encryption algorithms to hide messages when transmitting over public airways. Prime factorization also forms the basis for many cryptographic systems, such as RSA encryption, where the difficulty of factoring large numbers into their prime factorization allows high levels of security when encrypting messages. Euclid's algorithm is very efficient and simple to implement, especially on modern computers, and is very useful for finding modular inverses, solving linear Diophantine equations, and simplifying rationals.

2 Theorem 11.6

Prove that the $\gcd(n, m)$ is a product of primes that are common to n and m, where the power of each prime in the product is the smaller of its orders in n and m.

Proof. The **Fundamental Theorem of Arithmetic** states that every integer greater than 1 is either a prime number itself or can be represented as a product of prime numbers, and that this representation is unique up to the order of the factors. By the FToA, let the prime factorization of n be denoted by:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

and the prime factorization of m be denoted by:

$$m = p_1^{b_1} \cdot p_2^{b_2} \cdot \ldots \cdot p_k^{b_k}$$

where p_i are the prime factors and a_i and b_i are their respective powers in n and m. Notice that if a prime p_i does not divide n or m, we can say that its power is 0 in n or m. The **greatest common divisor** of n and m is the product of the primes that are common to n and m, where the power of each prime in the product is the smaller of its orders in n and m. Therefore, we can write: Let d be the greatest common divisor (GCD) of n and m. Then, d is given by:

$$d = p_1^{\min(a_1,b_1)} \cdot p_2^{\min(a_2,b_2)} \cdot \ldots \cdot p_k^{\min(a_k,b_k)}$$

We need to show that $d = \gcd(n, m)$.

First, we need to show that d is a common divisor of both n and m. By construction, the exponents of each prime in the factorization of d are at most the exponents in the factorizations of n and m. Therefore, each term in the expression of d divides the corresponding term in the factorizations of both n and m. Hence, d is a common divisor of both n and m.

Next, we need to show that d is the greatest common divisor of n and m. Suppose there exists another common divisor c of n and m. Then, c can be expressed as:

$$c = p_1^{x_1} \cdot p_2^{x_2} \cdot \ldots \cdot p_k^{x_k}$$

where $x_1 \leq a_1$, $x_1 \leq b_1$, $x_2 \leq a_2$, $x_2 \leq b_2$, and so on. Since $\min(a_i, b_i) \leq a_i$ and $\min(a_i, b_i) \leq b_i$ for all i, it follows that $x_i \geq \min(a_i, b_i)$ for all i. Therefore, c is a multiple of d, and thus d is greater than or equal to any common divisor of n and m.

Hence, $d = \gcd(n, m)$, which completes the proof. By the definition of the GCD and by using the FToA, the GCD of n and m is a product of primes that are common to both n and m, with the power of each prime in the product being the smaller of its orders in n and m.

□

3 Number Theory - Section 11.3, 11.4, & 11.5

Modular arithmetic and group theory are fundamental concepts in mathematics with applications spanning various fields, including cryptography, computer science, and number theory. This summary provides an overview of modular arithmetic, group theory, and their applications in solving modular equations, computing modular powers, and understanding modular congruence.

3.1 Modular Arithmetic

Modular arithmetic is a branch of arithmetic that deals with operations on integers modulo a fixed integer called the modulus. In modular arithmetic, two numbers are considered equivalent if their difference is divisible by the modulus. For example, $7 \equiv 1 \pmod 6$ because $7 - 1 = 6$ is divisible by 6. Modular arithmetic is often denoted using the notation $a \equiv b \pmod n$, where a and b are integers, and n is the modulus.

If, as in Euclidean division, $a = qb + r$ we define the *modulo* operation as

$$a \mod b = r \tag{2}$$

For example

$$3 = 1 * 2 + 1 \implies 3 \mod 2 = 1 \tag{3}$$
$$15 = 1 * 12 + 3 \implies 14 \mod 12 = 3 \tag{4}$$
$$90 = 1 * 60 + 30 \implies 90 \mod 60 = 30 \tag{5}$$
$$99 = 0 * 100 + 99 \implies 99 \mod 100 = 99 \tag{6}$$

Equation (3) is an example of how modular arithmetic to test if a number is even or odd. Equation (4) shows how modular arithmetic determines how to use a 12-hour based cycle of time. Equation (5) furthermore determines how to use a 60-minute based cycle of time. Equation (6) is an example of a case where the quotient can be zero.

3.2 Group Theory

Group theory is the branch of mathematics that studies algebraic structures known as groups. A group is a set equipped with a binary operation that satisfies certain properties, including closure, associativity, identity element, and invertibility. In the context of modular arithmetic, the set of integers modulo n forms a group under addition modulo n, denoted as $(\mathbb{Z}_n, +)$.

3.3 Modular Exponentiation

Modular exponentiation is the process of computing $a^b \mod n$ efficiently. This operation arises frequently in cryptography, particularly in public-key encryption schemes like RSA. One common method for computing modular exponentiation is the repeated squaring algorithm, which reduces the number of multiplications required to compute $a^b \mod n$ by exploiting the properties of modular arithmetic and exponentiation.

A particularly interesting case is the modulus of an exponentiated positive integer as

$$c = b^e \mod m \tag{7}$$

When $e < 0$, modular exponentiation can be defined using the modular multiplicative inverse d

$$b^e \mod m = d^{-e} \mod m \tag{8}$$

and $bd \equiv 1(\mod m)$.

3.4 Modular Congruence

Modular congruence is an equivalence relation defined on integers modulo a fixed modulus n. Two integers a and b are said to be congruent modulo n if their difference is divisible by n. In other words, a and b have the same remainder when divided by n. Modular congruence is denoted as $a \equiv b \pmod{n}$ and is fundamental in solving modular equations and proving properties of modular arithmetic.

Two positive integers $a, b \in \mathbb{Z}^+$ are said to be "congruent mod m", denoted $a \equiv b(\mod m)$, if

$$a \mod m = b \mod m \tag{9}$$

For example

$$3 \mod 2 = 1 = 5 \mod 2 \implies 3 \equiv 5 (\mod 2) \tag{10}$$

$$15 \mod 12 = 3 = 27 \mod 12 \implies 15 \equiv 27 (\mod 12) \tag{11}$$

$$90 \mod 60 = 30 = 210 \mod 60 \implies 90 \equiv 210 (\mod 60) \tag{12}$$

$$99 \mod 100 = 99 = 1000000099 \mod 100 \implies 99 \equiv 1000000099 (\mod 100) \tag{13}$$

Similar to what we showed in the Modular Arithmetic section, (10) is another example of how modular arithmetic to test if a number is even or odd. Equation (11) shows how modular congruency shows us how to use a 12-hour based cycle of time. Equation (12) furthermore determines how to use a 60-minute based cycle of time. Equation (13) is an example of a case where the quotient can be zero.

3.5 Solving Modular Linear Equations

Solving modular linear equations involves finding integer solutions to equations of the form $ax \equiv b \pmod{n}$, where a, b, and n are given integers, and x is the unknown variable. This problem arises in various applications, including cryptography and number theory. Techniques for solving modular linear equations include the extended Euclidean algorithm, which finds the multiplicative inverse of a modulo n when a and n are coprime.

3.6 Computing Modular Powers

Computing modular powers involves efficiently computing $a^b \mod n$ for given integers a, b, and n. This operation is essential in cryptographic algorithms like RSA and Diffie-Hellman key exchange. Efficient algorithms for computing modular powers include exponentiation by squaring, which reduces the number of multiplications required to compute $a^b \mod n$ by exploiting the properties of modular arithmetic and exponentiation.

3.7 Summary

Modular arithmetic and group theory provide powerful tools for solving problems in various mathematical and computational domains. Understanding these concepts and their applications in modular exponentiation, modular congruence, solving modular equations, and computing modular powers is crucial for cryptography, computer science, and number theory. Further research and development in this field continue to advance the state-of-the-art in modular arithmetic and group theory, paving the way for new discoveries and applications in mathematics and beyond.

4 Theorem 11.12

Proof. First we must define congruence modulo n. We say that m is congruent to k modulo n if and only if m and k have the same remainder when divided by n, or equivalently, if $m \mod n = k \mod n$. Now, we must prove both directions of the statement.

Direction 1: m is congruent to k modulo n implies $m \mod n = k \mod n$:

Assume that m is congruent to k modulo n. This means that $m \equiv k \pmod{n}$. By definition, this implies that $m - k$ is divisible by n. Therefore, there exists an integer q such that $m - k = qn$. Rearranging this equation, we get $m = qn + k$.

Now, let's find the remainder when m is divided by n. We have:

$$\begin{aligned} m \mod n &= (qn + k) \mod n \\ &= (qn \mod n + k \mod n) \mod n \\ &= (0 + k \mod n) \mod n \\ &= k \mod n \end{aligned}$$

Hence, $m \mod n = k \mod n$, as required.

Direction 2: $m \mod n = k \mod n$ implies m is congruent to k modulo n:

Conversely, assume that $m \mod n = k \mod n$. This means that when m and k are divided by n, they have the same remainder. In other words, there exist integers p and q such that:

$$\begin{aligned} m &= pn + (m \mod n) \\ k &= qn + (k \mod n) \end{aligned}$$

Since $m \mod n = k \mod n$, we can substitute this into the equations above:

$$\begin{aligned} m &= pn + (k \mod n) \\ k &= qn + (k \mod n) \end{aligned}$$

Thus, we have $m = pn + k - qn$, which simplifies to $m = qn + k$. This means that $m - k = qn$, indicating that m is congruent to k modulo n, as required.

Therefore, we have shown both directions of the statement, concluding the proof.

□

5 Number Theory - Section 11.6 & 11.7

Prime numbers play a crucial role in various cryptographic protocols, hashing functions, and other computational tasks. Prime numbers, as we said above, are integers greater than 1 with no positive divisors other than 1 and themselves. Prime numbers are very significantly important in computer science. They form the building blocks of many cryptographic systems, such as RSA encryption, and are essential in various algorithms and data structures. However, as the size of prime numbers increases, the computational challenges associated with finding and verifying them also grow. There are many number theoretic algorithms tailored to address these challenges.

5.1 Finding Large Prime Numbers

One of the fundamental challenges in number theory and computer science is finding large prime numbers efficiently. Various algorithms tackle this problem, including the Sieve of Eratosthenes, which efficiently generates all prime numbers up to a specified limit. However, for extremely large primes, probabilistic algorithms like the Miller-Rabin primality test and deterministic algorithms like the AKS primality test are preferred. These algorithms leverage different techniques such as modular arithmetic, polynomial factorization, and probabilistic methods to identify prime numbers within a given range.

5.2 Checking Primality

Verifying whether a number is prime is a crucial step in many applications. While deterministic algorithms like AKS offer certainty, they can be computationally expensive for large numbers. Probabilistic algorithms like Miller-Rabin provide a probabilistic guarantee of primality, with adjustable accuracy based on the number of iterations. Other methods, such as the Lucas-Lehmer test for Mersenne primes or the elliptic curve primality proving (ECPP) algorithm, offer alternative approaches with varying trade-offs in terms of efficiency and certainty.

5.3 Runtime Efficiencies

The runtime efficiency of prime number algorithms is paramount in practical applications. Probabilistic algorithms often provide a balance between efficiency and accuracy, making them suitable for many real-world scenarios. Deterministic algorithms, while offering certainty, may suffer from high computational complexity, especially for extremely large numbers. Therefore, understanding the runtime complexities of different algorithms and selecting the appropriate one based on the specific requirements of the application is crucial for optimizing performance.

The RSA public key cryptosystem is a widely used encryption and digital signing technique that relies on the difficulty of factoring large composite numbers into their prime factors. Named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman, RSA provides a secure method for communication and data protection over insecure networks. This summary explores how RSA works, its underlying principles, and some important theorems associated with the RSA cryptosystem.

5.4 How RSA Works

RSA encryption involves the generation of public and private key pairs, each consisting of a modulus n and an exponent e for the public key, and a modulus n and an exponent d for the private key. The security of RSA relies on the fact that it is computationally infeasible to factorize the modulus n into its prime factors without knowing the private key.

The encryption process involves converting the plaintext message M into a numerical representation m and then raising it to the power of the public exponent e modulo n:

$$C \equiv m^e \pmod{n}$$

where C is the ciphertext.

Decryption, on the other hand, involves raising the ciphertext C to the power of the private exponent d modulo n:

$$m \equiv C^d \pmod{n}$$

which recovers the original plaintext message M.

5.5 Theorems about the RSA Cryptosystem

5.5.1 The RSA Encryption Theorem

Theorem: Let M be a plaintext message, C be the corresponding ciphertext, and e and d be the public and private exponents, respectively. Then, for any plaintext message M, the ciphertext C obtained through RSA encryption and the plaintext message M obtained through RSA decryption satisfy:

$$M \equiv (M^e)^d \pmod{n}$$

Proof: The proof follows from the decryption process of RSA, where $M \equiv (C^d) \pmod{n}$. Since $C \equiv M^e \pmod{n}$, substituting $C = M^e$ into the decryption equation yields $M \equiv (M^e)^d \pmod{n}$.

5.5.2 The RSA Homomorphic Property

Theorem: RSA encryption possesses a homomorphic property, meaning that the multiplication of two ciphertexts results in the encryption of the product of the corresponding plaintexts.

Proof: Let C_1 and C_2 be ciphertexts corresponding to plaintexts M_1 and M_2, respectively. Then, $C_1 \cdot C_2 \equiv (M_1^e \cdot M_2^e) \pmod{n}$. By the homomorphic property of modular exponentiation, $M_1^e \cdot M_2^e \equiv (M_1 \cdot M_2)^e \pmod{n}$, which implies that $C_1 \cdot C_2$ is the encryption of the product of M_1 and M_2.

5.6 Summary

Number theoretic algorithms play a critical role in computer science applications, particularly in cryptography, data security, and optimization. The efficient generation and verification of large prime numbers are essential for ensuring the security and integrity of digital communication and computational systems. By exploring and understanding the various algorithms for finding and verifying prime numbers, we can balance the performance and reliability of our applications. The RSA public key cryptosystem offers a robust method for secure communication and data protection. By leveraging the difficulty of factoring large composite numbers, RSA ensures confidentiality and integrity in digital communication. Understanding the underlying principles and theorems associated with RSA is crucial for its proper implementation and usage in real-world applications.

Primality testing and RSA encryption are intimately connected within the realm of cryptography and number theory. The RSA public key cryptosystem relies on the difficulty of factoring large composite numbers into their prime factors for its security. Primality testing plays a crucial role in RSA, as it ensures the selection of large prime numbers for generating secure RSA keys. Efficient primality testing algorithms, such as the Miller-Rabin primality test, enable the rapid identification of large prime numbers, essential for RSA key generation. Moreover, the security of RSA encryption relies on the assumption that it is computationally infeasible to factorize the product of two large primes back into its constituent primes, a problem closely related to primality testing. Therefore, primality testing algorithms and RSA encryption are intricately intertwined, forming the foundation of secure digital communication and data protection in modern cryptography.

6 Presburger and Halting Complexity

The Halting Problem asks if an algorithm exists which is always capable of determining whether a given program will ever terminate, provided that said algorithm exists within the program it is attempting to terminate, and the program will perform the opposite action to what the algorithm outputs. For example: if the algorithm determines that a given program will terminate, the system will never terminate, but given our assumption that the algorithm will always be right and the algorithm will perform the opposite to what the algorithm produces, it has effectively disproved itself. Because of this dilemma, the problem is non-decidable, and therefore intractable.

Deciding the outcome of a Presburger arithmetic function is one that remains decidable, yet has been proven impossible to solve in polynomial time. Presburger arithmetic seeks to determine outcome of a statement consisting of some series of inequalities with various undefined variables. Because each of these variables must be evaluated before making a decision, the evaluation of each variable increases the runtime by a factor equal to the number of variables. For example, in the following trivial statement, variables x, y, and z must each be evaluated in order to solve the inequality.

$$2x + 4y - 3z < 7 \wedge 3x - y + 2z < -4 \tag{14}$$

The solution for this trivial case is easily calculable, however as the number of variables increase, the complexity for calculating the statements value grows exponentially. Because of this the complexity for Presburger Arithmetic functions is impossible to solve in polynomial time.

7 Polynomial-time Algorithms

Sorting an unsorted array is solvable using various polynomial-time algorithms. This is because the problem size of length n is traverse in n time, and requires traversing the graph for every index, which can be performed in $O(n^2)$ time. There do exist algorithms however which solve this problem in $O(nlogn)$ time including Merge Sort and Quick Sort.

For the fake coin problems (which can be used for identifying anomalies in data) involves the identification of a single coin which produces a heavier weight in a pile of $n-1$ coins which all share the same weight. With the ability to compare two separate piles of coins, this problem can be solved in $O(logn)$ time, as the coins are split into equal piles, and weighed, and then splitting the heavier pile in two and weighing until only one coin remains as the fake coin. Because the search space is being cut in half after every weigh, the problem size is decreasing by half, therefore representing a logarithmic solution.

Another problem with a polynomial-time algorithm includes string matching. Where a length of n characters exist, and the location of a given set of characters must be located. The brute force approach to this algorithm involves iteratively searching for a matching string. However, this approach can take $O(mn)$ where m is the length of the search term. To avoid this, approaches have been implemented which can skip chunks of the problem space (up to the length of m) depending on how values are read in the line. These approaches include the Boyer-Moore algorithm and the Aho-Corasick algorithm, which produce near linear runtime.

8 Encoding Schemes

8.1 Problem: Traveling Salesman Problem (TSP)

The Traveling Salesman Problem is an NP-complete problem in which the goal is to find the shortest possible route that visits each city exactly once and returns to the origin city.

8.2 Encoding Schemes

8.2.1 Encoding Scheme 1: Adjacency Matrix

In this encoding scheme, we represent the input as an $n \times n$ matrix, where n is the number of cities. Each element (i, j) in the matrix represents the distance between city i and city j.

8.2.2 Encoding Scheme 2: Edge List

In this encoding scheme, we represent the input as a list of edges, where each edge is represented by a tuple $(\text{city}_1, \text{city}_2, \text{distance})$, indicating the connection between city city_1 and city city_2 with the given distance.

8.3 Performance Analysis

8.3.1 Performance using Encoding Scheme 1 (Adjacency Matrix)

- **Space Complexity**: $O(n^2)$ to store the entire adjacency matrix, where n is the number of cities.
- **Time Complexity**:
  - Computing the distance between two cities: $O(1)$ (constant time lookup in the matrix).
  - Checking the existence of an edge: $O(1)$.
  - Total time complexity for distance computations and edge checks: $O(n^2)$.

8.3.2 Performance using Encoding Scheme 2 (Edge List)

- **Space Complexity**: $O(n^2)$ to store the list of edges, where n is the number of cities in the worst-case scenario where every pair of cities is connected.
- **Time Complexity**:
  - Computing the distance between two cities: $O(n)$ in the worst case (linear time search through the list of edges).
  - Checking the existence of an edge: $O(n)$ in the worst case.
  - Total time complexity for distance computations and edge checks: $O(n^2)$.

8.4 Summary

We considered two encoding schemes for representing the input of the Traveling Salesman Problem: Adjacency Matrix and Edge List. Both encoding schemes have their advantages and disadvantages in terms of space and time complexity.

The Adjacency Matrix encoding scheme offers constant time lookup for computing distances between cities and checking the existence of edges, resulting in a time complexity of $O(n^2)$. However, its space complexity of $O(n^2)$ may become prohibitive for large instances of the TSP.

On the other hand, the Edge List encoding scheme has the same space complexity as the Adjacency Matrix but may require slightly higher time complexity ($O(n^2)$) for distance computations and edge checks due to linear time searches through the list of edges. Nonetheless, it offers flexibility and can be more memory-efficient for sparse graphs.

Therefore, the choice between these encoding schemes depends on the specific requirements of the problem instance and the algorithms employed for solving it. Both encoding schemes provide viable approaches for tackling the Traveling Salesman Problem and other related optimization problems.