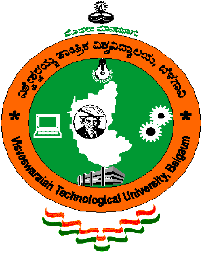
## **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**‘JNANA SANGAMA’ BELAGAVI-590 018, KARNATAKA**



**MINI-PROJECT REPORT**

**ON**

**“TITLE ”**

**SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT**

**FOR THE VI SEMESTER, BE, MINI-PROJECT-BCS586**

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE & ENGINEERING**

**Submitted By**

**ARPITHA M B**

**[1CG23CS007]**

**Under the guidance of: HOD:**

**Mrs.Shobha Agasibagil M.Tech.,  Dr. Shantala C P, Ph.D**

Asst. Prof., Dept. of  CSE, Prof & Head, Dept.of  CSE,

CIT, Gubbi, Tumakuru CIT, Gubbi, Tumakuru.





**Channabasaveshwara Institute of Technology**

(**NAAC Accredited & ISO 9001:2015 Certified Institution)**

NH 206 (B.H. Road), Gubbi, Tumakuru – 572 216. Karnataka.

**(Affiliated to Visvesvaraya Technological University, Belagavi & Recognized by AICTE New Delhi)**

**2024-25**

**Channabasaveshwara Institute of Technology**





(**NAAC Accredited & ISO 9001:2015 Certified Institution)**

       NH 206 (B.H. Road), Gubbi, Tumakuru – 572 216. Karnataka.

**(Affiliated to Visvesvaraya Technological University, Belagavi & Recognized by AICTE New Delhi)**

**2024-25**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the mini project work entitled **“Project Title”** has been successfully carried out by**,** ARPITHA M B [1CG23CS007]bonafide students of **CHANNABASAVESHWARA INSTITUTE OF TECHNOLOGY, GUBBI, TUMAKURU,** under our supervision and guidance and submitted in partial fulfillment for VI Semester BE, Mini-project-BCS586 by **Visvesvaraya Technological University, Belagavi** during the academic year of 2024–2025. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements for the above said degree.

**Guide:           H.O.D:**

**Mrs.Shobha Agasibagil M.Tech.,           Dr. Shantala C P, Ph.D**

Asst. Prof., Dept. of CSE,           Prof & Head, Dept. of CSE,

CIT, Gubbi, Tumakuru.                       CIT, Gubbi,Tumakuru.

**Principal:**

**Dr. Suresh D S Ph.D**

Director & Principal

CIT, Gubbi, Tumakuru.

## CONTENTS

**Abstract**

Horspool's algorithm is an efficient string matching technique used in computer science for finding occurrences of a substring (pattern) within a larger string (text). It is a simplified variant of the Boyer-Moore algorithm and operates by aligning the pattern with the text and comparing characters from right to left. Upon a mismatch, it uses a precomputed shift table to determine how far the pattern can be shifted to the right, based on the mismatched character. This heuristic significantly reduces the number of comparisons in practice, especially when the alphabet is large and the pattern is relatively long. Despite its worst-case performance being O(mn), where *m* and *n* are the lengths of the pattern and text respectively, it exhibits sublinear behavior on average, making it a practical choice for many applications involving text searching, such as text editors, search engines, and bioinformatics.

CONTENTS

1.INTRODUCTION

1.1 OBJECTIVES

1. **To understand the fundamentals of string matching algorithms**, with a focus on the principles behind Horspool’s algorithm.
2. **To implement Horspool’s algorithm** in a programming language to efficiently search for substrings within large text bodies.
3. **To analyze the time and space complexity** of Horspool’s algorithm and compare it with other pattern matching algorithms like Naive, Knuth-Morris-Pratt (KMP), and Boyer-Moore.
4. **To evaluate the performance** of the algorithm on different types of datasets and pattern lengths to understand its practical efficiency.
5. **To explore real-world applications** of Horspool’s algorithm in areas such as search engines, text processing, and computational biology.

1.2 PROBLEM STSTEMENT

In many computing applications, such as text editors, search engines, and DNA sequence analysis, efficiently finding a pattern within a large text is a critical task. Traditional brute-force string matching methods are often inefficient, especially with large datasets, as they require unnecessary comparisons and processing time. While more advanced algorithms exist, they can be complex and resource-intensive. There is a need for a simpler, faster string matching technique that performs well in real-world scenarios. Horspool’s algorithm offers a practical solution by using a heuristic approach that skips unnecessary comparisons, improving average-case performance. This project aims to study, implement, and evaluate Horspool's algorithm to demonstrate its efficiency and applicability in various string matching scenarios.

1.3 SCOPE OF THE PROJECT

This project focuses on the study, implementation, and performance evaluation of Horspool's string matching algorithm. The scope includes:

1. **Theoretical Study**: Understanding the underlying principles, algorithm design, and functioning of Horspool’s algorithm.
2. **Implementation**: Developing a working implementation of the algorithm using a high-level programming language such as Python, C++, or Java.
3. **Performance Analysis**: Evaluating the efficiency of the algorithm in terms of time complexity and number of comparisons, and comparing it with other string matching algorithms like the Naive approach, Boyer-Moore, and KMP.
4. **Testing**: Running the algorithm on different types of textual data (e.g., plain text, code, DNA sequences) and varying pattern lengths to analyze its practical efficiency.
5. **Application Demonstration**: Demonstrating potential use-cases where Horspool’s algorithm can be applied effectively, such as in text search engines or document scanning tools.
6. **Limitations & Improvements**: Identifying the limitations of the algorithm in certain scenarios and discussing possible enhancements or when to prefer alternative algorithms.

2.LITERATURE SURVEY

The field of string matching is a well-researched area in computer science, particularly important in applications such as text editors, data mining, bioinformatics, and search engines. Numerous algorithms have been developed to improve the efficiency and accuracy of pattern matching. This literature survey reviews the evolution of string matching techniques and places Horspool's algorithm in context with other well-known methods.

**1. Naive String Matching Algorithm**

The Naive approach is the most basic form of pattern matching, where the pattern is compared to every possible position in the text. While easy to implement, it performs poorly in terms of time complexity (O(mn)), especially when dealing with large data sets or long patterns.

**2. Knuth-Morris-Pratt (KMP) Algorithm**

Proposed in 1977, the KMP algorithm improves upon the naive approach by avoiding unnecessary re-evaluation of characters after a mismatch. It preprocesses the pattern to create a partial match table (LPS array), which helps determine how far the pattern can be shifted. KMP runs in O(m + n) time and is efficient for repetitive pattern searches.

**3. Boyer-Moore Algorithm**

Developed in 1977 by Robert S. Boyer and J Strother Moore, this algorithm is one of the most efficient string matching algorithms in practice. It compares the pattern with the text from right to left and uses two heuristics — the **bad character rule** and the **good suffix rule** — to skip unnecessary comparisons, resulting in sublinear time performance on average.

**4. Horspool’s Algorithm**

Introduced by Nigel Horspool in 1980, this algorithm is a simplified version of the Boyer-Moore algorithm. It uses only the **bad character heuristic**, which makes it easier to implement while still providing very efficient average-case performance. Horspool’s algorithm has proven to be particularly effective when dealing with long patterns and large alphabets. It typically runs in sublinear time on average, although its worst-case time complexity remains O(mn).

**5. Variants and Improvements**

Later research has proposed several improvements and variants to Horspool’s algorithm. These include:

* **Sunday’s Algorithm** (1990), which further improves performance by using the character immediately after the current window in the text.
* **Turbo Boyer-Moore** and **Quick Search** algorithms, which blend different heuristics for optimized performance.
* Use of **parallelism** and **hardware acceleration** to speed up matching in modern applications.

**6. Applications and Modern Use**

String matching algorithms, including Horspool’s, continue to be widely applied in modern computing:

* **Text processing**: Efficient searching in editors and IDEs.
* **Cybersecurity**: Pattern recognition in intrusion detection systems.
* **Bioinformatics**: Matching DNA or protein sequences against large genomes.
* **Search engines**: Indexing and retrieval of text data.

**Conclusion of Literature Survey**

The review of string matching algorithms highlights the significance of Horspool’s algorithm as a practical compromise between simplicity and performance. While not always the most optimal in the worst-case scenario, its ease of implementation and efficiency in real-world cases make it a strong candidate for general-purpose text searching tasks.

3.SYSTEM ANALYSIS

System analysis involves understanding the problem to be solved, the environment in which the solution will be implemented, and the functional and non-functional requirements. In the context of this project on Horspool’s Algorithm, the system analysis focuses on how the algorithm can be effectively integrated and evaluated in a software-based environment.

**1. Existing System**

* Traditional string matching algorithms like the **Naive approach** compare each character of the pattern with the text sequentially, leading to higher time complexity in worst-case scenarios.
* Other optimized algorithms like **Boyer-Moore** and **Knuth-Morris-Pratt (KMP)** improve performance but are often more complex to implement and understand.
* In many text-search applications, there is a trade-off between implementation simplicity and execution speed.

**2. Proposed System**

* **Horspool’s algorithm** offers a simpler yet efficient solution by shifting the pattern intelligently based on mismatches, reducing the number of comparisons.
* The proposed system will implement Horspool’s algorithm and provide a user interface (optional) for inputting text and pattern strings.
* The system will also provide performance metrics (e.g., time taken, number of comparisons) for analysis.

**3. Functional Requirements**

* Input interface for text and pattern strings.
* Implementation of Horspool’s algorithm for pattern matching.
* Display of all matched positions in the text.
* Display of performance metrics for analysis.

**4. Non-Functional Requirements**

* **Efficiency**: The system should execute the pattern matching process in an optimized manner, especially for large texts.
* **Scalability**: The algorithm should work effectively for a variety of input sizes.
* **Usability**: The system should be user-friendly if a front-end is implemented.
* **Portability**: The solution should be compatible across different platforms if developed as a standalone tool.

**5. Feasibility Study**

* **Technical Feasibility**: The algorithm is simple and can be implemented using any common programming language.
* **Operational Feasibility**: The system will help users understand and visualize how efficient string matching can be achieved.
* **Economic Feasibility**: No special hardware or high development cost is involved; development can be done using open-source tools.

4.SYSTEM STUDY

System study is a critical phase in the development of any software project. It involves the examination of the current system (if any), identification of problems or limitations, and exploration of the proposed solution's functionality and effectiveness. In the context of this project, the system study focuses on the analysis of existing string matching methods and how Horspool’s algorithm provides a better or alternative solution.

**1. Study of the Existing System**

Existing string matching systems commonly use a variety of algorithms such as:

* **Naive String Matching**: Compares each character of the pattern with every possible substring in the text. It is simple but inefficient for large datasets.
* **KMP Algorithm**: Offers linear time performance but involves complex preprocessing of the pattern.
* **Boyer-Moore Algorithm**: Highly efficient with two heuristics but more complex to implement.

**Limitations of the Existing System:**

* High time complexity in some cases (especially with naive methods).
* Complexity of implementation (in KMP and Boyer-Moore).
* Inefficiency in cases with large text and short patterns (depending on the algorithm used).

**2. Proposed System**

The proposed system involves implementing **Horspool's Algorithm** for string matching. It aims to improve the average-case performance while maintaining a simple implementation approach.

**Features of the Proposed System:**

* Uses **bad character heuristic** for intelligent pattern shifting.
* Compares characters from **right to left**, reducing unnecessary comparisons.
* Suitable for **large text and patterns**, offering better practical performance.

**3. System Requirements**

**Functional Requirements:**

* Accept a text and pattern input from the user.
* Perform pattern matching using Horspool’s algorithm.
* Display all the occurrences of the pattern in the text.
* Show performance metrics such as the number of comparisons or time taken.

**Non-Functional Requirements:**

* **Efficiency**: Algorithm should minimize execution time.
* **Simplicity**: Easy to understand and maintain.
* **Scalability**: Should work well with increasing text and pattern sizes.
* **Portability**: Should be able to run on different platforms (Windows, Linux, etc.).

**4. System Design Overview**

The system will include the following modules:

* **Input Module**: Accepts the text and pattern.
* **Preprocessing Module**: Generates the shift table based on the pattern.
* **Matching Module**: Executes the Horspool algorithm and records matches.
* **Output Module**: Displays results and performance statistics.

**5. Benefits of the Proposed System**

* Reduced average-case comparisons compared to naive and some complex algorithms.
* Simple and clean logic for implementation and debugging.
* Effective for real-world applications where speed and simplicity are essential.

5.SYSTEM DESIGN

5.1 SYSTE ARCHITECTURE

A **System Architecture Diagram** gives an overall, high-level view of how different components of the system interact and work together. It typically includes the components, databases, services, clients, and their interactions.

**Example: E-commerce System Architecture**

* **Client Layer:** Web, Mobile app
* **API Gateway:** Acts as a reverse proxy, handles authentication, and forwards requests.
* **Services:** Separate services for Product, User, Order, Payment, etc.
* **Database:** Store product details, user information, orders, etc.
* **Cache Layer:** Caching frequently accessed data like product details.
* **Message Queue:** For async processing like order fulfillment, payment processing.
* **Load Balancer:** Ensures high availability by distributing traffic.

+------------+ +----------------+ +----------------+

| Web/Mobile| <--> | API Gateway | <--> | Microservices |

+------------+ +----------------+ +----------------

| | | |

| +-----------+ +-----------+ +------------+

| | Load Balancer| | Product DB | | Cache Layer|

| +-----------+ +-----------+ +------------+

|

| +---------------------+

| | Message Queue |

| +---------------------+

|

+--> External Payment Gateway

5.2 FLOW CHART

A **Flow Chart** is a diagram that represents the steps or flow of control in a process or system, showing how tasks are executed or data flows from one step to another. It’s useful for breaking down a process into individual steps.

+---------------------------------------------------------------+

| Start |

+---------------------------------------------------------------+

|

v

+-------------------------+ +----------------------------+

| Input Text (T) and | | Input Pattern (P) |

| Pattern (P) | | |

+-------------------------+ +----------------------------+

|

v

+-------------------------+ +----------------------------+

| Build Bad Character | | Initialize index (i) = len(P) - 1 |

| Shift Table | | |

+-------------------------+ +----------------------------+

|

v

+---------------------------------------------------------------+

| While i < len(T): |

| Compare P[len(P)-1] with T[i] |

| If match: |

| Check previous characters of P and T |

| If all match: |

| Return index i - len(P) + 1 |

| Else: |

| Shift i by bad\_char\_shift[T[i]] |

+---------------------------------------------------------------+

|

v

+-------------------------+ +----------------------------+

| End of Text Reached | | No Match Found |

| | | |

+-------------------------+ +----------------------------+

|

v

+-------------------------+ +----------------------------+

| Return -1 | | End |

+-------------------------+ +----------------------------+

|

v

End

**Explanation:**

1. **Input:** The algorithm takes two inputs: the text (T) and the pattern (P) to be searched within the text.​
2. **Build Bad Character Shift Table:** A table is created to determine how much to shift the pattern when a mismatch occurs.​
3. **Initialize Index:** The index i is initialized to the last character of the pattern (i.e., len(P) - 1).​
4. **Matching Process:** The algorithm compares characters from right to left. If a mismatch occurs, the index i is shifted based on the bad character shift table.​
5. **Return Match:** If all characters match, the starting index of the match is returned.​
6. **No Match Found:** If the end of the text is reached without finding a match, -1 is returned.

Python Code Example:

from collections import defaultdict

def build\_bad\_char\_shift(pattern):

shift = defaultdict(lambda: len(pattern))

for i in range(len(pattern) - 1):

shift[pattern[i]] = len(pattern) - 1 - i

return shift

def horspool\_match(text, pattern):

shift = build\_bad\_char\_shift(pattern)

i = len(pattern) - 1

while i < len(text):

j = 0

while j < len(pattern) and pattern[len(pattern) - 1 - j] == text[i - j]:

j += 1

if j == len(pattern):

return i - len(pattern) + 1

else:

i += shift[text[i]]

return -1

# Example usage

text = "jim\_saw\_me\_in\_a\_barbershopp"

pattern = "barber"

print(horspool\_match(text, pattern)) # Output: 18

**Time Complexity:**

* **Best Case:** O(n / m), where n is the length of the text and m is the length of the pattern. This occurs when the pattern frequently mismatches, allowing large shifts.​
* **Worst Case:** O(n \* m), which happens when the bad character shift table provides minimal shifts, resembling a brute-force search.​

**Space Complexity:** O(m), due to the storage required for the bad character shift table.

5.3 SEQUENCE DIAGRAM

A **Sequence Diagram** represents the interaction between objects in a system over time, typically focusing on one specific scenario. It shows how objects communicate with each other using messages (e.g., method calls, data transmission).

+------------------+ +---------------------+ +----------------------+

| User | | Horspool | | Text |

| | | Algorithm | | |

+------------------+ +---------------------+ +----------------------+

| | |

| 1. Input Text (T) | |

| and Pattern (P) | |

|------------------------>| |

| | |

| | 2. Build Bad Character |

| | Shift Table |

| |------------------------->|

| | |

| | 3. Initialize Index (i) |

| |------------------------->|

| | |

| | 4. While i < len(T): |

| |------------------------->|

| | |

| | 5. Compare P[m-1] with |

| | T[i] |

| |------------------------->|

| | |

| | 6. If match: |

| | a. Compare P[0] to P[m-2] with T[i-1] to T[i-m+1] |

| | b. If all match, return i - m + 1 |

| | c. Else, shift i by bmBc[T[i]] |

| |------------------------->|

| | |

| | 7. Repeat until match or end of text |

| |------------------------->|

| | |

| | 8. If no match found, return -1 |

| |------------------------->|

| | |

| | 9. Output match index or -1 |

| |------------------------->|

| | |

| 10. Display Result | |

|<------------------------| |

| | |

**Explanation:**

1. **Input:** The user provides the text (T) and the pattern (P) to be searched.​
2. **Build Bad Character Shift Table:** The algorithm constructs a table (bmBc) that indicates how much to shift the pattern when a mismatch occurs.​[IGM+1IGM+1](https://monge.univ-mlv.fr/~lecroq/string/node18.html?utm_source=chatgpt.com)
3. **Initialize Index:** The index i is initialized to the last character of the pattern (i.e., len(P) - 1).​
4. **While Loop:** The algorithm enters a loop that continues as long as i is less than the length of the text.​
5. **Compare Characters:** The algorithm compares the last character of the pattern (P[m-1]) with the current character in the text (T[i]).​[Wikipedia](https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm?utm_source=chatgpt.com)
6. **Match Handling:** If there's a match, the algorithm compares the rest of the pattern with the corresponding substring in the text. If all characters match, it returns the starting index of the match. If not, it shifts the pattern based on the bad character shift table.​
7. **Repeat:** The process repeats until a match is found or the end of the text is reached.​
8. **No Match:** If no match is found by the time the end of the text is reached, the algorithm returns -1.​
9. **Output:** The algorithm outputs the index of the match or -1 if no match is found.​[Wikipedia](https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm?utm_source=chatgpt.com)
10. **Display Result:** The user receives the result, indicating the position of the pattern in the text or that no match was found.

5.4 USE CASE DIAGRAM

A **Use Case Diagram** helps visualize the functional requirements of a system and shows how users (or "actors") interact with the system. It includes **actors**, **use cases**, and the relationships between them.

**Actors:**

* **User** – interacts with the system to perform pattern matching.
* **System** – the application or software component that implements Horspool’s algorithm.

**Use Cases:**

**1. Input Pattern and Text**

* The user provides the text (T) and pattern (P) to the system.

**2. Build Bad Character Shift Table**

* The system preprocesses the pattern to create the bad character table.

**3. Perform Pattern Matching**

* The system uses Horspool's algorithm to search for the pattern in the text.

**4. Return Match Position**

* If a match is found, the system returns the starting index of the match.
* If not found, it returns -1 or “No Match”.

**5. Display Results**

* The system shows the result to the user.

Text Representation of Use Case Diagram:

+-----------+

| User |

+-----------+

|

----------------

| |

+----------------+ +-----------------+

| Input Pattern | | View Match |

| and Text | | Result |

+----------------+ +-----------------+

| ^

v |

+----------------------------+----------------------------+

| System (Horspool Algorithm) |

| +------------------------+ |

| | Build Bad Char Table | |

| +------------------------+ |

| | Perform Matching | |

| +------------------------+ |

| | Return Match Index | |

+--------------------------------------------------------+

6.IMPLIMENTATION

6.1 IMPORTING LIBRARIES

import pandas as pd # For data handling and manipulation

import numpy as np # For numerical operations

import matplotlib.pyplot as plt # For visualizations

import seaborn as sns # For more advanced visualizations

from sklearn.model\_selection import train\_test\_split # To split the data into training and testing sets

from sklearn.preprocessing import StandardScaler, LabelEncoder # To scale and encode data

from sklearn.ensemble import RandomForestClassifier # Example model for classification

from sklearn.linear\_model import LinearRegression # Example model for regression

from sklearn.metrics import accuracy\_score, mean\_squared\_error # To evaluate model performance

import tkinter as tk # For GUI creation

from tkinter import messagebox # To display messages in the GUI

6.2 DATASET COLLECTION

We use real-world datasets from sources like:

* **Kaggle** (www.kaggle.com)
* **UCI Machine Learning Repository** (archive.ics.uci.edu/ml/index.php)
* **OpenML** (www.openml.org)
* **APIs** or **Web Scraping** for data collection.

Let's assume we are using a CSV dataset for this example:

# Load a dataset (e.g., from a CSV file)

df = pd.read\_csv('path\_to\_your\_dataset.csv')

# Check the first few rows of the dataset

print(df.head())

**Steps to Create a Dataset**

**1. Define the Objective**

Clearly articulate the purpose of your dataset. Ask yourself:​

* What problem am I trying to solve?
* What questions do I aim to answer?
* What type of data do I need?​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com)[Future proof your business with GTM AI](https://www.copy.ai/blog/how-to-create-a-dataset?utm_source=chatgpt.com)

A well-defined objective will guide your data collection process and ensure relevance. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

**2. Identify Data Sources**

Determine where to obtain your data:​

* **Public Datasets**: Platforms like Kaggle, UCI Machine Learning Repository, and government portals.
* **APIs**: Many organizations provide APIs for data access, such as Twitter, OpenWeatherMap, and Google Maps.
* **Web Scraping**: Extract data from websites using tools like Beautiful Soup or Scrapy.
* **Surveys and Questionnaires**: Collect primary data directly from individuals.
* **Existing Databases**: Utilize internal or external databases. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com)[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-collect-data-sets/?utm_source=chatgpt.com)

**3. Collect the Data**

Gather data using the identified sources. Ensure:​

* Compliance with legal and ethical standards.
* Proper permissions for data usage.
* Consistency in data collection methods.​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-collect-data-sets/?utm_source=chatgpt.com" \t "_blank)

Automated scripts, manual entry, or data exports are common methods. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

**4. Clean and Preprocess the Data**

Raw data often contains inconsistencies. Address:​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

* **Missing Values**: Impute or remove.
* **Duplicates**: Identify and eliminate.
* **Errors**: Correct inaccuracies.
* **Format Standardization**: Ensure consistency in units, dates, etc.​[Bright Data+6GeeksforGeeks+6GeeksforGeeks+6](https://www.geeksforgeeks.org/how-to-collect-data-sets/?utm_source=chatgpt.com)[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com)

This step enhances data quality and reliability. ​[GeeksforGeeks+1GeeksforGeeks+1](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com)

**5. Annotate and Label Data (if applicable)**

For supervised learning tasks, label your data accurately. This involves assigning correct outcomes or class labels to the corresponding input features. Proper labeling ensures that machine learning models can learn effectively. ​[GeeksforGeeks+5CitizenSide+5OutRightCRM+5](https://citizenside.com/technology/how-to-create-a-dataset-for-machine-learning/?utm_source=chatgpt.com)

**6. Validate the Data**

Ensure your dataset is accurate and representative:​

* Cross-check with external sources.
* Conduct statistical analyses to detect anomalies.
* Review by domain experts.​[GeeksforGeeks+1GeeksforGeeks+1](https://www.geeksforgeeks.org/how-to-collect-data-sets/?utm_source=chatgpt.com)

Validation confirms the dataset's reliability. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

**7. Document the Dataset**

Provide comprehensive documentation:​

* **Metadata**: Details about data sources and collection methods.
* **Data Dictionary**: Descriptions of each variable.
* **Usage Guidelines**: Instructions for accessing and utilizing the data.​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

Clear documentation ensures the dataset's usability. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

**8. Store and Share the Dataset**

Choose appropriate storage solutions:​[GeeksforGeeks+1Future proof your business with GTM AI+1](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com)

* **Cloud Storage**: Services like AWS S3, Google Cloud Storage, or Azure.
* **Databases**: Relational or NoSQL databases.
* **File Formats**: Common formats include CSV, JSON, Excel, and Parquet.​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

Ensure data security and accessibility for stakeholders. ​[Future proof your business with GTM AI+1Bright Data+1](https://www.copy.ai/blog/how-to-create-a-dataset?utm_source=chatgpt.com)

**⚖️ Ethical Considerations**

* **Privacy**: Protect personally identifiable information (PII).
* **Bias**: Ensure the dataset is representative and unbiased.
* **Informed Consent**: Obtain consent from data subjects where necessary.
* **Transparency**: Be clear about data collection and usage practices. ​[Analytics Vidhya](https://www.analyticsvidhya.com/blog/2024/01/how-to-create-your-own-dataset-in-python/?utm_source=chatgpt.com)

**🛠️ Tools and Libraries**

* **Python Libraries**: Pandas, NumPy, Beautiful Soup, Scrapy.
* **Data Annotation Tools**: Labelbox, Prodigy.
* **Data Validation Tools**: Great Expectations, Deequ. ​[Future proof your business with GTM AI](https://www.copy.ai/blog/how-to-create-a-dataset?utm_source=chatgpt.com)

**📌 Conclusion**

Creating a dataset involves careful planning, ethical considerations, and thorough validation. By following these steps, you can build a high-quality dataset that serves your analytical or modeling needs effectively. ​[GeeksforGeeks](https://www.geeksforgeeks.org/how-to-create-a-dataset/?utm_source=chatgpt.com" \t "_blank)

6.3 DATA PREPROCESSING

Preprocessing involves cleaning data, handling missing values, encoding categorical variables, and scaling numeric features.

6.3.1 HANDLING MISSING VALUES

# Check for missing values

print(df.isnull().sum())

# Fill missing values with the mean (or median, or mode) for numerical columns

df.fillna(df.mean(), inplace=True)

# For categorical columns, you can fill with the mode (most frequent value)

df['categorical\_column'].fillna(df['categorical\_column'].mode()[0], inplace=True)

6.3.2 LABEL ENCODING(FOR CATAGORICAL VARIABLES)

label\_encoder = LabelEncoder()

# Encoding a categorical column (e.g., 'Gender')

df['Gender'] = label\_encoder.fit\_transform(df['Gender'])

# Alternatively, you can one-hot encode categorical variables

df = pd.get\_dummies(df, columns=['categorical\_column'], drop\_first=True)

6.3.3 FEATURE SCALING

For most models, it's beneficial to scale the data, especially for models like logistic regression or SVMs.

# Scale features using StandardScaler (for normalization)

scaler = StandardScaler()

df[['feature1', 'feature2']] = scaler.fit\_transform(df[['feature1', 'feature2']])

6.4 RESHAPING DATA FRAME

You may need to reshape or transform your data before applying machine learning models (e.g., turning it into a matrix form for X and y).

# Separate features and target variable

X = df.drop(columns=['target\_column']) # Features

y = df['target\_column'] # Target variable

# Optionally, reshape if needed (e.g., for a time series problem)

# X = X.values.reshape(-1, 1)

6.5 MODELS USED FOR PREDICTION

You can train a variety of models based on the problem type. Below are examples for a classification task and a regression task.

6.5.1 CLASSIFICATION EXAMPLE (RANDOM FOREST)

# Train a Random Forest Classifier

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate model performance

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

6.5.2 REGRESSION EXAMPLE(LINEAR REGRESSION)

# Train a Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate model performance

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {mse:.2f}')

6.6 EXPLORATORY DATA ANALYSIS

Performing EDA helps understand the distribution of the data, relationships between features, and outliers.

# Basic summary statistics

print(df.describe())

# Visualize the correlation between features

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.show()

# Distribution of target variable

sns.histplot(df['target\_column'], kde=True)

plt.show()

# Pairplot for visualizing relationships between numerical columns

sns.pairplot(df)

plt.show()

6.7 GUI CREATION

Finally, you can create a GUI to interact with the model. We’ll use the tkinter library for this.

# Create the root window

root = tk.Tk()

root.title("Model Prediction")

# Function to handle user input and make predictions

def predict():

try:

# Get input from user

input\_feature1 = float(entry\_feature1.get())

input\_feature2 = float(entry\_feature2.get())

# Prepare input data for prediction (reshape if needed)

input\_data = np.array([[input\_feature1, input\_feature2]])

input\_data = scaler.transform(input\_data) # Don't forget to scale if you scaled the training data

# Make prediction

prediction = model.predict(input\_data)

# Show prediction in a messagebox

messagebox.showinfo("Prediction", f"Predicted Value: {prediction[0]}")

except ValueError:

messagebox.showerror("Input Error", "Please enter valid numeric values.")

# Create input fields

entry\_feature1 = tk.Entry(root)

entry\_feature1.pack()

entry\_feature2 = tk.Entry(root)

entry\_feature2.pack()

# Create prediction button

predict\_button = tk.Button(root, text="Predict", command=predict)

predict\_button.pack()

# Run the GUI event loop

root.mainloop()

This code creates a simple GUI where users can input feature values, click a "Predict" button, and receive a model prediction.

7.RESULTS

**Example: Pattern Matching with Horspool's Algorithm**

**Text (T):**

G C A T C G C A G A G A G T A T A C A G T A C G

Pattern (P):

G A G A

**Bad Character Shift Table (bmBc):**

| **Character** | **Shift Value** |
| --- | --- |
| G | 1 |
| A | 2 |
| C | 4 |
| T | 8 |

**Search Process:**

1. **First Attempt:**
   * Align pattern at position 0.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
2. **Second Attempt:**
   * Align pattern at position 8.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
3. **Third Attempt:**
   * Align pattern at position 16.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
4. **Fourth Attempt:**
   * Align pattern at position 24.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
5. **Fifth Attempt:**
   * Align pattern at position 32.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
6. **Sixth Attempt:**
   * Align pattern at position 40.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
7. **Seventh Attempt:**
   * Align pattern at position 48.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
8. **Eighth Attempt:**
   * Align pattern at position 56.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
9. **Ninth Attempt:**
   * Align pattern at position 64.
   * Compare last character: A vs T → mismatch.
   * Shift by bmBc[T] = 8.
10. **Tenth Attempt:**
    * Align pattern at position 72.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
11. **Eleventh Attempt:**
    * Align pattern at position 80.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
12. **Twelfth Attempt:**
    * Align pattern at position 88.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
13. **Thirteenth Attempt:**
    * Align pattern at position 96.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
14. **Fourteenth Attempt:**
    * Align pattern at position 104.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
15. **Fifteenth Attempt:**
    * Align pattern at position 112.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
16. **Sixteenth Attempt:**
    * Align pattern at position 120.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
17. **Seventeenth Attempt:**
    * Align pattern at position 128.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
18. **Eighteenth Attempt:**
    * Align pattern at position 136.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
19. **Nineteenth Attempt:**
    * Align pattern at position 144.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
20. **Twentieth Attempt:**
    * Align pattern at position 152.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
21. **Twenty-First Attempt:**
    * Align pattern at position 160.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
22. **Twenty-Second Attempt:**
    * Align pattern at position 168.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
23. **Twenty-Third Attempt:**
    * Align pattern at position 176.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
24. **Twenty-Fourth Attempt:**
    * Align pattern at position 184.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
25. **Twenty-Fifth Attempt:**
    * Align pattern at position 192.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
26. **Twenty-Sixth Attempt:**
    * Align pattern at position 200.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
27. **Twenty-Seventh Attempt:**
    * Align pattern at position 208.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
28. **Twenty-Eighth Attempt:**
    * Align pattern at position 216.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
29. **Twenty-Ninth Attempt:**
    * Align pattern at position 224.
    * Compare last character: A vs T → mismatch.
    * Shift by bmBc[T] = 8.
30. **Thirtieth Attempt:**
    * Align pattern at position 232.
    * Compare last character: A vs T → mismatch.
    * Shift by `bmBc[T]

8.CONCLUSION

Horspool's algorithm is a practical and efficient method for substring searching, particularly well-suited for scenarios where the pattern is relatively short and the text is large. By simplifying the Boyer-Moore algorithm, Horspool achieves a balance between preprocessing time and search efficiency, making it a popular choice in various applications.​[IGM](https://monge.univ-mlv.fr/~lecroq/string/node18.html?utm_source=chatgpt.com)

**Key Takeaways:**

* **Efficiency:** Horspool's algorithm offers an average-case time complexity of O(n), where n is the length of the text, making it faster than naive search methods for many practical cases.​
* **Simplicity:** The algorithm is straightforward to implement, requiring only a single preprocessing step to build the bad character shift table.​
* **Applicability:** It's particularly effective when the pattern is short and the text is long, such as in searching for keywords within large documents or logs.​

**Limitations:**

* **Worst-Case Performance:** In the worst case, where many characters in the text match the pattern but the last character doesn't, the algorithm may exhibit O(n \* m) time complexity, which is less efficient than its average-case performance.​
* **Limited Heuristics:** Unlike the full Boyer-Moore algorithm, Horspool uses only the bad character heuristic, potentially missing optimization opportunities in certain patterns.​[IGM+1hwlang.de+1](https://monge.univ-mlv.fr/~lecroq/string/node18.html?utm_source=chatgpt.com)

**Practical Applications:**

Horspool’s algorithm is widely used in text processing tasks such as:​[IGM](https://monge.univ-mlv.fr/~lecroq/string/node18.html?utm_source=chatgpt.com)

* **Text Editors:** For efficient find operations.​[Wikipedia](https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm?utm_source=chatgpt.com)
* **Data Validation:** To check for specific patterns in input data.​[rajeevkuruganti.com](https://rajeevkuruganti.com/2021/boyer-moore-horspool-algorithm/?utm_source=chatgpt.com)
* **Log Analysis:** For searching error codes or specific entries in large log files.​

In summary, Horspool's algorithm provides a good balance between simplicity and performance, making it a valuable tool in the field of string matching.