INDEX

PREFACE

Advanced algorithms form the backbone of modern computing, enabling efficient solutions to complex problems in areas like optimization, networks, and data processing. This lab manual for M.Tech CSE I Year II Semester equips students with hands-on skills to implement and analyze key techniques—from brute-force enumeration to sophisticated string matching and flow algorithms—fostering deep insight into performance trade-offs.

The experiments align with core texts like S. Sridhar's *Design and Analysis of Algorithms* and references such as Cormen et al.'s *Introduction to Algorithms*, emphasizing practical coding in C/C++ alongside theoretical rigor. Through these 10 structured labs, students will not only code solutions but also evaluate time/space complexities, preparing them for real-world applications in IT infrastructure and systems design.

ACKNOWLEDGEMENT

It has been a rewarding experience working on the Advance algorithms Laboratory Manual. We would like to express our sincere gratitude to **Dr.K.Abdul Basith**, Professor and Head of the Department of Computer Science and Engineering, Marri Laxman Reddy Institute of Technology & Management, for his constant encouragement, valuable guidance, and continuous support in the preparation of this manual.

We are deeply indebted and gratefully acknowledge the support and motivation provided by the **Dr.P.Sridhar** Director, Marri Laxman Reddy Institute of Technology & Management, for granting us the opportunity and necessary resources to develop this laboratory manual.

Our heartfelt thanks to **Dr. R. Murali Prasad**, Principal, Marri Laxman Reddy Institute of Technology & Management, for his insightful suggestions, timely feedback, and academic guidance throughout the preparation of this document.

Finally, we extend our sincere appreciation to all the faculty members of the CSE Department whose encouragement, cooperation, and constructive inputs have played a significant role in helping us accomplish this work successfully.

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL
GENERAL INSTRUCTIONS

- Students are instructed to attend the laboratory on time. Late comers will not be entertained in the lab.
- Students must be punctual. Experiments conducted during the session will not be repeated for those who are absent or late.
- Students are expected to come prepared with the theory and procedure of the experiment scheduled for the day.
- The use of mobile phones inside the lab is strictly prohibited.
- Any damage or loss of system components such as keyboard, mouse, or other peripherals during the lab session will be the responsibility of the student, and a fine or penalty will be imposed accordingly.
- Students must update their lab records and observation books session-wise. Before leaving the lab, the student should get their observation book signed by the concerned faculty member.
- Lab records must be submitted in the next lab session to the respective faculty members in the staffroom for correction and return.
- Students should not move around or disturb others during the lab session.
- In case of any emergency, students must obtain written permission from the concerned faculty member before leaving the laboratory.
- Faculty members reserve the right to suspend any student from the lab session on disciplinary grounds.

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL
INSTITUTE VISION AND MISSION

Vision:

To be as an ideal academic institution by graduating talented engineers to be ethically strong, competent with quality research and technologies.

Mission:

- Utilize rigorous educational experiences to produce talented engineers
- Create an atmosphere that facilitates the success of students
- Programs that integrate global awareness, communication skills and Leadership qualities
- Education and Research partnership with institutions and industries to prepare the students for interdisciplinary research

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL
DEPARTMENT VISION AND MISSION

Vision:

To empower the students to be technologically adept, innovative, self-motivated and responsible global citizen possessing human values and contribute significantly towards high quality technical education with ever changing world.

Mission:

- To offer high-quality education in the computing fields by providing an environment where the knowledge is gained and applied to participate in research, for both students and faculty.
- To develop the problem solving skills in the students to be ready to deal with cutting edge technologies of the industry.
- To make the students and faculty excel in their professional fields by inculcating the communication skills, leadership skills, team building skills with the organization of various co-curricular and extra- curricular programmes.
- To provide the students with theoretical and applied knowledge, and adopt an education approach that promotes lifelong learning and ethical growth.

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL
PROGRAM OUTCOMES

PO No.	NBA Statement / Vital Features	No. of Vital Features
PO1.	**An ability to independently carry out research /investigation and development work to solve practical problems.** 1. Research problems in Computer Science and Engineering are clearly identified and defined. 2. Literature review highlights research gaps and suitable methods. 3. Experiments or simulations are conducted using appropriate tools. 4. Data collection, analyses, and interpretation systematically. 5. Innovative approaches are applied to engineering problem-solving. **6.** Results are validated against established theories and standards.	6
PO 2.	**An ability to write and present a substantial technical report/document** 1. Technical reports, dissertations, and papers are well-structured. 2. Referencing and academic integrity practices are properly maintained. 3. Content is presented with clarity, precision, and logical flow. 4. Oral communication and presentation skills are effectively demonstrated. 5. Digital tools are used for documentation and visualization. **6.** Research findings are communicated to both technical and non-technical audience.	6
PO 3.	**Students should be able to demonstrate advanced proficiency in Computer Science and allied emerging areas of Engineering.** i. In-Depth Technical Knowledge ii. Advanced Problem-Solving and Algorithmic Skills. iii. Hands-On Practical Expertise iv. Research and Innovation Aptitude **v.** Interdisciplinary Integration	5
PO 4.	**Students should be able to identify, analyze, and effectively solve complex real-world problems by applying advanced computing concepts, while considering solutions from a global perspective** i. Ability to break down multifaceted problems into manageable parts and develop effective solutions using advanced computing techniques. ii. Analytical Thinking and Critical Reasoning iii. Advanced Computing Knowledge.	

	iv. Capability to understand and model complex systems, considering interdependencies and dynamic behaviors within global contexts. v. Global and Societal Awareness. vi. Aptitude for developing novel approaches and innovative solutions to address complex challenges. vii. Ability to work effectively in diverse teams and integrate knowledge from multiple disciplines to solve global problems. viii. Skill in clearly presenting problem analyses, solutions, and their implications to technical and non-technical global audience.	**8**
PO 5.	**An ability to acquire and apply advanced technical knowledge, professional skills, and modern computing tools to develop sustainable solutions** i. Ability to continuously learn and apply cutting-edge technical knowledge in computing and related fields. ii. Proficiency with Modern Computing Tools. iii. Capability to design and implement solutions that are environmentally, and economically feasible. iv. Understanding of professional ethics and responsibility in creating technology solutions with long-term positive impact. v. Integration of Multidisciplinary Knowledge. vi. Adaptability to sustainable practices.	**6**
PO 6.	**An Ability to recognize the significance of lifelong learning and actively pursue continuous professional development by adapting technologies in emerging areas.** i. Self-Directed Learning. ii. Skill in quickly learning and integrating new technologies and tools as they emerge in the field. iii. Capability to assess the relevance and impact of emerging technologies and decide on their applicability. iv. Continuous Professional Development Planning. v. Ability to engage with professional communities, attend workshops, and collaborate to stay updated. vi. Habit of regularly reflecting on personal growth, learning experiences, and professional competencies to improve continuously.	**6**

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL
PROGRAM EDUCATIONAL OBJECTIVES

Sl. No.	PEOs Name	Program Education Objective Statements
1	PEO – 1	Graduates will achieve professional excellence and success in the field of Computer Science and Engineering by applying strong technical foundations and problem-solving skills to contribute effectively to industry, academia, and entrepreneurship.
2	PEO – 2	Graduates will demonstrate a commitment to lifelong learning by continuously enhancing their knowledge and skills through professional development and self-directed learning to effectively adapt to evolving global challenges.
3	PEO – 3	Graduates of the Computer Science and Engineering program will actively pursue advanced research, contributing to the development of solutions for complex problems and the generation of new knowledge to effectively address real-world challenges.
4	PEO – 4	Graduates will exhibit professionalism, effective communication, leadership skills, and ethical responsibility while working in multidisciplinary teams to deliver computing solutions that address societal needs and contribute to sustainable development.

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL

PROGRAM SPECIFIC OUTCOMES

PSO1: Applications of Computing: Ability to use knowledge in various domains to provide solution to new ideas and innovations.

PSO2: Programming Skills: Identify required data structures, design suitable algorithms, develop and maintain software for real world problems.

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL

Course Structure

Advanced Algorithms Lab will have a continuous evaluation during II semester for 40 sessional marks and 60 end semester examination marks.

Out of the 40 marks for internal evaluation, day-to-day work in the laboratory shall be evaluated for 20 marks and internal practical examination shall be evaluated for 10 marks conducted by the laboratory teacher concerned.

The end semester examination shall be conducted with an external examiner and internal examiner. The external examiner shall be appointed by the principal / Chief Controller of examinations

COMPUTER SCIENCE AND ENGINEERING
ADVANCED ALGORITHMS LAB MANUAL

OBJECTIVE:

- To be able to introduce core programming basics and program design with functions using Python programming language.
- To understand a range of Object-Oriented Programming, as well as in-depth data and information processing techniques.
- To understand the high-performance programs designed to strengthen the practical expertise.

OUTCOMES:

Upon the completion of Operating Systems practical course, the student will be able to:

- Student should be able to understand the basic concepts scripting and the contributions of scripting language
- Ability to explore python especially the object oriented concepts, and the built in objects of Python.

2425835: ADVANCED ALGORITHMS LAB (Lab - III)

M.Tech CSE I Year II Sem. **L T P C**
0 0 4 2

Course Objective: The student can able to attain knowledge in advanced algorithms.

Course Outcomes: The student can able to analyze the performance of algorithms

List of Experiments

1. Implement assignment problem using Brute Force method
2. Perform multiplication of long integers using divide and conquer method.
3. Implement a solution for the knapsack problem using the Greedy method.
4. Implement Gaussian elimination method.
5. Implement LU decomposition
6. Implement Warshall algorithm
7. Implement the Rabin Karp algorithm.
8. Implement the KMP algorithm.
9. Implement Harspool algorithm
10. Implement max-flow problem.

TEXT BOOK:

1. Design and Analysis of Algorithms, S.Sridhar, OXFORD University Press

REFERENCES:

1. Introduction to Algorithms, second edition, T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, PHI Pvt. Ltd./ Pearson Education.
2. Fundamentals of Computer Algorithms, Ellis Horowitz, Satraj Sahni and Rajasekharam, Universities Press.
3. Design and Analysis of algorithms, Aho, Ullman and Hopcroft, Pearson education.

Experiment-1

Implement assignment problem using Brute Force method

```
import itertools
def assignment_bruteforce(cost_matrix):
n = len(cost_matrix)
min_cost = float('inf')
best_assignment = None
for perm in itertools.permutations(range(n)):
cost = sum(cost_matrix[i][perm[i]] for i in range(n))
if cost <min_cost:
min_cost = cost
best_assignment = perm
return best_assignment, min_cost
cost_matrix = [
[9, 2, 7, 8],
[6, 4, 3, 7],
[5, 8, 1, 8],
[7, 6, 9, 4]
]
assignment, cost = assignment_bruteforce(cost_matrix)
print("Best Assignment:", assignment)
print("Minimum Cost:", cost)
```

OUTPUT:-

```
Best Assignment: (1, 2, 3, 0)
Minimum Cost: 21
```

Viva Questions

1. What is the assignment problem, and how is it modeled mathematically?
2. Explain the brute force approach for solving the assignment problem.
3. What is the time complexity of the brute force method for n workers and n jobs?
4. Why is brute force impractical for large n (e.g., n > 10)?
5. How do you generate all possible permutations in your code (e.g., using std::next_permutation in C++)?
6. Describe the steps to compute the total cost for a given permutation.
7. What data structure do you use for the cost matrix (e.g., 2D vector or array)?
8. How do you ensure each job is assigned to exactly one worker and vice versa?
9. Provide a sample input cost matrix for n=3 and trace the optimal assignment found.
10. What is the role of factorial (n!) in the brute force enumeration?
11. How does the Hungarian algorithm differ from brute force in efficiency?
12. What edge cases should your code handle (e.g., all costs equal, zero costs)?
13. How do you output the minimum cost and the corresponding assignment?
14. Compare space complexity of brute force vs. dynamic programming for assignment.
15. Why group permutations by fixing one assignment (e.g., job 1 to person 1) to reduce redundancy?

Experiment-2

Perform multiplication of long integers using divide and conquer method

```
def karatsuba(x, y):
# Base case for recursion
if x < 10 or y < 10:
return x * y
else:
m = min(len(str(x)), len(str(y))) // 2
high_x, low_x = divmod(x, 10**m)
high_y, low_y = divmod(y, 10**m)
z0 = karatsuba(low_x, low_y)
z1 = karatsuba(low_x + high_x, low_y + high_y)
z2 = karatsuba(high_x, high_y)
return z2 * 10**(2 * m) + (z1 - z2 - z0) * 10**m + z0
# Example
x = 1234
y = 5678
result = karatsuba(x, y)
print("Product:", result)
```

OUTPUT:-

```
Product: 7006652
```

Viva Questions

1. What is Divide and Conquer method?
2. Which famous algorithm is used for long integer multiplication using divide and conquer?
3. Who proposed the Karatsuba algorithm?
4. What is the time complexity of the normal multiplication method?
5. What is the time complexity of Karatsuba algorithm?
6. How do you split a long integer in divide and conquer method?
7. What is the base case in long integer multiplication?
8. What is the recurrence relation of Karatsuba algorithm?
9. Why is divide and conquer faster than traditional multiplication?
10. What is the main idea behind Karatsuba multiplication?
11. How many recursive calls are made in Karatsuba method?
12. What is the space complexity of divide and conquer multiplication?
13. Can divide and conquer method be used for very large numbers?
14. What is the difference between classical multiplication and Karatsuba multiplication?
15. In which type of problems is divide and conquer strategy commonly used?

Experiment-3

Implement a solution for the knapsack problem using the Greedy method.

```
def fractional_knapsack(weights, values, capacity):
n = len(values)
ratio = [(values[i] / weights[i], weights[i], values[i]) for i in range(n)]
ratio.sort(reverse=True, key=lambda x: x[0])
total_value = 0
for r, w, v in ratio:
if capacity - w >= 0:
capacity -= w
total_value += v
else:
total_value += v * (capacity / w)
break
return total_value
# Example
weights = [10, 20, 30]
values = [60, 100, 120]
capacity = 50
print("Maximum value:", fractional_knapsack(weights, values, capacity))
```

OUTPUT:-

Maximum value: 240.0

Viva Questions

1. What is the Knapsack problem?
2. Which type of Knapsack problem uses the Greedy method?
3. What is the time complexity of the Greedy Knapsack algorithm?
4. On what basis are items selected in the Greedy method?
5. What is meant by profit-to-weight ratio?
6. Can Greedy method solve 0/1 Knapsack optimally?
7. What is the first step in the Greedy Knapsack algorithm?
8. Why is sorting required in Fractional Knapsack?
9. What is the base condition in the Knapsack algorithm?
10. What happens when the remaining capacity is less than the item weight?
11. What is the space complexity of the Greedy approach?
12. Is Fractional Knapsack a dynamic programming problem?
13. Why does Greedy work for Fractional Knapsack?

14. What data structure is commonly used to sort items?
15. What is the difference between 0/1 Knapsack and Fractional Knapsack?

Experiment-4

Implement Gaussian elimination method.

```
import numpy as np
def gaussian_elimination(a, b):
n = len(b)
a = np.array(a, dtype=float)
b = np.array(b, dtype=float)
for i in range(n):
max_row = np.argmax(np.abs(a[i:, i])) + i
a[[i, max_row]] = a[[max_row, i]]
b[i], b[max_row] = b[max_row], b[i]
for j in range(i + 1, n):
factor = a[j][i] / a[i][i]
a[j, i:] -= factor * a[i, i:]
b[j] -= factor * b[i]
x = np.zeros_like(b)
for i in range(n - 1, -1, -1):
x[i] = (b[i] - np.dot(a[i, i + 1:], x[i + 1:])) / a[i][i]
```

```
return x
# Example
a = [[2, -1, 1], [-3, -1, 2], [-2, 1, 2]]
b = [8, -11, -3]
print("Solution:", gaussian_elimination(a, b))
```

OUTPUT:-

```
Solution: [2. 3. -1.]
```

Viva Questions

1. What is Gaussian Elimination method?
2. For what type of equations is Gaussian Elimination used?
3. What is an augmented matrix?
4. What is the main goal of Gaussian Elimination?
5. What is meant by row operations?
6. How many types of elementary row operations are there?
7. What is forward elimination?
8. What is back substitution?
9. What is a pivot element?
10. What is meant by a singular matrix?
11. What is the time complexity of Gaussian Elimination?
12. What happens if a pivot element is zero?
13. What is partial pivoting?
14. What is the difference between Gauss Elimination and Gauss-Jordan method?
15. Can Gaussian Elimination be used for non-linear equations?

Experiment-5

Implement LU decomposition.

```
import numpy as np
def lu_decomposition(matrix):
n = len(matrix)
L = np.zeros_like(matrix, dtype=float)
U = np.zeros_like(matrix, dtype=float)
for i in range(n):
L[i][i] = 1
for j in range(i, n):
U[i][j] = matrix[i][j] - sum(L[i][k] * U[k][j] for k in range(i))
for j in range(i + 1, n):
L[j][i] = (matrix[j][i] - sum(L[j][k] * U[k][i] for k in range(i))) / U[i][i]
return L, U
# Example
A = np.array([[4, 3], [6, 3]])
L, U = lu_decomposition(A)
print("L:", L)
print("U:", U)
```

OUTPUT:-

```
L: [[1. 0.]
 [1.5 1.]]
U: [[4. 3.]
 [0. 0.]]
```

Viva Questions

1. What is LU Decomposition?
2. Into which two matrices is a matrix decomposed in LU method?
3. What does L represent in LU decomposition?
4. What does U represent in LU decomposition?
5. For which type of matrices is LU decomposition applicable?
6. What is the main purpose of LU decomposition?
7. What is the time complexity of LU decomposition?
8. What is meant by a triangular matrix?
9. What is a unit lower triangular matrix?
10. What happens if a pivot element is zero during LU decomposition?

11. What is partial pivoting in LU decomposition?
12. How is LU decomposition related to Gaussian elimination?
13. Can LU decomposition be used to find the determinant of a matrix?
14. What is the difference between LU decomposition and Cholesky decomposition?
15. Is LU decomposition suitable for solving multiple systems with the same coefficient matrix?

Experiment-6

Implement Warshall algorithm

```
def warshall_algorithm(graph):
n = len(graph)
reach = [row[:] for row in graph]
for k in range(n):
for i in range(n):
for j in range(n):
reach[i][j] = reach[i][j] or (reach[i][k] and reach[k][j])
return reach
# Example adjacency matrix
graph = [
[0, 1, 0],
[0, 0, 1],
[1, 0, 0]
]
transitive_closure = warshall_algorithm(graph)
print("Transitive Closure:", transitive_closure)
```

OUTPUT:-

```
Transitive Closure: [[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

Viva Questions

1. What is Warshall algorithm?
2. For which type of graph is Warshall algorithm used?
3. What problem does Warshall algorithm solve?
4. What is meant by transitive closure?
5. What type of matrix is used in Warshall algorithm?
6. What is the time complexity of Warshall algorithm?
7. Is Warshall algorithm applicable to weighted graphs?
8. What is the space complexity of Warshall algorithm?
9. Which technique is used in Warshall algorithm?
10. How many nested loops are used in Warshall algorithm?
11. What is the base matrix in Warshall algorithm?
12. What is the difference between Warshall and Floyd algorithm?
13. Can Warshall algorithm detect reachability?
14. What is the order of vertices considered in Warshall algorithm?
15. Is Warshall algorithm a greedy method?

Experiment-7

Implement the Rabin Karp algorithm.

```
def rabin_karp(text, pattern):
d = 256  # number of characters in the input alphabet
q = 101  # a prime number
m = len(pattern)
n = len(text)
h_pattern = 0  # hash value for pattern
h_text = 0  # hash value for text
h = 1  # value of d^(m-1) % q
for i in range(m - 1):
h = (h * d) % q
for i in range(m):
h_pattern = (d * h_pattern + ord(pattern[i])) % q
h_text = (d * h_text + ord(text[i])) % q
for i in range(n - m + 1):
if h_pattern == h_text:
if text[i:i+m] == pattern:
return i
if i< n - m:
h_text = (d * (h_text - ord(text[i]) * h) + ord(text[i + m])) % q
if h_text< 0:
h_text += q
return -1
# Example
text = "ABABDABACDABABCABAB"
pattern = "ABABCABAB"
print("Pattern found at index:", rabin_karp(text, pattern))
```

OUTPUT:-

```
Pattern found at index: 10
```

Viva Questions

1. What is the Rabin–Karp algorithm?
2. For what type of problem is Rabin–Karp mainly used?
3. What technique is used in Rabin–Karp algorithm?
4. What is meant by a rolling hash?
5. What is the average time complexity of Rabin–Karp algorithm?
6. What is the worst-case time complexity of Rabin–Karp algorithm?

7. What is a spurious hit in Rabin–Karp?
8. Why is modulus operation used in Rabin–Karp algorithm?
9. What is the purpose of rehashing?
10. Which type of string matching does Rabin–Karp support efficiently?
11. How does Rabin–Karp differ from Naïve string matching?
12. What happens after a hash match is found?
13. What are the main parameters used in hash calculation?
14. Can Rabin–Karp be used for multiple pattern matching?
15. Is Rabin–Karp a deterministic or randomized algorithm?

Experiment-8

Implement the KMP algorithm.

```
def kmp_search(text, pattern):
def lps_array(pattern):
lps = [0] * len(pattern)
j = 0
for i in range(1, len(pattern)):
while j > 0 and pattern[i] != pattern[j]:
j = lps[j - 1]
if pattern[i] == pattern[j]:
j += 1
lps[i] = j
return lps
lps = lps_array(pattern)
i = 0
j = 0
while i<len(text):
if text[i] == pattern[j]:
i += 1
j += 1
if j == len(pattern):
return i - j
else:
if j > 0:
j = lps[j - 1]
else:
i += 1
return -1
# Example
text = "ABABDABACDABABCABAB"
pattern = "ABABCABAB"
print("Pattern found at index:", kmp_search(text, pattern))
```

OUTPUT:-

```
Pattern found at index: 1
```

Viva Questions

1. What is the KMP algorithm?
2. Who developed the KMP algorithm?
3. What problem does KMP solve?
4. What is the time complexity of KMP algorithm?
5. What is the LPS array in KMP?
6. What does LPS stand for?
7. Why is the LPS array needed?
8. What is the preprocessing step in KMP?
9. What is the worst-case time complexity of the naïve string matching algorithm?
10. How does KMP improve over the naïve approach?
11. What happens when a mismatch occurs in KMP?
12. Is KMP a dynamic programming algorithm?
13. What is the space complexity of KMP algorithm?
14. Can KMP be used for multiple pattern matching?
15. Is KMP suitable for large text searching?

Experiment-9

Implement Harspool algorithm

```
def heapify(arr, n, i):
largest = i
left = 2 * i + 1
right = 2 * i + 2
if left < n and arr[left] >arr[largest]:
largest = left
if right < n and arr[right] >arr[largest]:
largest = right
if largest != i:
arr[i], arr[largest] = arr[largest], arr[i]
heapify(arr, n, largest)
def heap_sort(arr):
n = len(arr)
for i in range(n // 2 - 1, -1, -1):
heapify(arr, n, i)
for i in range(n - 1, 0, -1):
arr[i], arr[0] = arr[0], arr[i]
heapify(arr, i, 0)
# Example
arr = [12, 11, 13, 5, 6, 7]
heap_sort(arr)
print("Sorted array:", arr)
```

OUTPUT:-

Sorted array: [5, 6, 7, 11, 12, 13]

Viva Questions

1. What is Horspool algorithm?
2. Which type of problem does Horspool algorithm solve?
3. On which algorithm is Horspool based?
4. What technique is used in Horspool algorithm?
5. What is the purpose of the shift table in Horspool algorithm?
6. How is the shift value calculated in Horspool method?
7. What is the average time complexity of Horspool algorithm?
8. What is the worst-case time complexity of Horspool algorithm?
9. From which side does Horspool start comparison?

10. What happens when a mismatch occurs in Horspool algorithm?
11. Does Horspool use preprocessing?
12. What is the space complexity of Horspool algorithm?
13. Is Horspool better than the naïve string matching algorithm?
14. Can Horspool be used for large text searching?
15. What is the main advantage of Horspool over brute-force method?

Experiment-10

Implement max-flow problem

```
from collections import deque
def bfs(capacity, flow, source, sink, parent):
visited = [False] * len(capacity)
queue = deque([source])
visited[source] = True
parent[source] = -1
while queue:
u = queue.popleft()
for v in range(len(capacity)):
if not visited[v] and capacity[u][v] - flow[u][v] > 0:
queue.append(v)
visited[v] = True
parent[v] = u
if v == sink:
return True
return False
def ford_fulkerson(capacity, source, sink):
n = len(capacity)
flow = [[0] * n for _ in range(n)]
parent = [-1] * n
max_flow = 0
while bfs(capacity, flow, source, sink, parent):
path_flow = float('Inf')
s = sink
while s != source:
path_flow = min(path_flow, capacity[parent[s]][s] - flow[parent[s]][s])
s = parent[s]
max_flow += path_flow
v = sink
while v != source:
u = parent[v]
flow[u][v] += path_flow
flow[v][u] -= path_flow
v = parent[v]
return max_flow
# Example
```

```
capacity = [
[0, 16, 13, 0, 0, 0],
[0, 0, 10, 12, 0, 0],
[0, 4, 0, 0, 14, 0],
[0, 0, 9, 0, 0, 20],
[0, 0, 0, 7, 0, 4],
[0, 0, 0, 0, 0, 0]
]
source = 0
sink = 5
print("Maximum Flow:", ford_fulkerson(capacity, source, sink))
```

OUTPUT:-

Maximum Flow: 23

Viva Questions:

1. What is the Max-Flow problem?
2. In which type of graph is the Max-Flow problem defined?
3. What is meant by source and sink in a flow network?
4. What is capacity in a flow network?
5. What is flow conservation law?
6. What is residual capacity?
7. What is an augmenting path?
8. Which famous algorithm is used to solve the Max-Flow problem?
9. What is the time complexity of Ford–Fulkerson algorithm?
10. What is the difference between Ford–Fulkerson and Edmonds–Karp algorithm?
11. What is a residual graph?
12. Can flow exceed capacity in a network?
13. What is the Max-Flow Min-Cut theorem?
14. Is Max-Flow problem a greedy algorithm?
15. What are the real-world applications of Max-Flow problem?