KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

ADVANCED ALGORITHMS LABORATORY MANUAL

Subject Code : KGR2358203

Regulation : KGR23

Academic Year : 2023-2024

M .Tech CSE/CS I Year / II Sem

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with governmentorganizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

• To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
• To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
• To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societalissues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.
PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO II: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO III: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO IV: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO V: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO VI: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO VII: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
PO VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO X: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO XI: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO XII: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areasof computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi-disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED ALGORITHMS

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

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

S.NO	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.

1. **Implement assignment problem using Brute Force method**

   The brute force method involves generating all possible permutations of task-worker assignments and calculating the cost of each permutation. The permutation with the lowest cost is then selected as the optimal solution.

Here's an implementation of the Brute Force method for solving the Assignment Problem in Python:

```
import itertools


def brute_force_assignment(cost_matrix):
    # Generate all possible permutations of task-worker assignments
    n = len(cost_matrix)
    assignments = itertools.permutations(range(n))

    # Calculate the cost of each permutation
    min_cost = float('inf')
    optimal_assignment = None
    for assignment in assignments:
        cost = sum(cost_matrix[i][j] for i, j in enumerate(assignment))
        if cost < min_cost:
            min_cost = cost
            optimal_assignment = assignment

    # Return the optimal assignment and its cost
    return optimal_assignment, min_cost
```

2. Perform multiplication of long integers using divide and conquer method.

The Divide and Conquer method is a popular technique used in computer science to solve problems by recursively breaking them down into smaller subproblems, solving the subproblems independently, and then combining the solutions to solve the original problem. This method can be used for multiplication of long integers as well.

The basic idea behind the Divide and Conquer method for multiplication is to break the two long integers into smaller halves, recursively compute the product of these halves, and then combine these smaller products to form the final product.

Here's an implementation of the Divide and Conquer method for multiplication of long integers in Python:

python

```
def multiply(x, y):
    # Base case: If the input is a single digit, return the product
    if len(str(x)) == 1 or len(str(y)) == 1:
        return x*y

    # Split the input integers into smaller halves
    n = max(len(str(x)), len(str(y)))
    n2 = n // 2

    a = x // 10**n2
    b = x % 10**n2
    c = y // 10**n2
    d = y % 10**n2

    # Recursively compute the product of the smaller halves
    ac = multiply(a, c)
    bd = multiply(b, d)
    ad_bc = multiply(a+b, c+d) - ac - bd

    # Combine the smaller products to form the final product
    return ac * 10**(2*n2) + ad_bc * 10**n2 + bd
```

The function multiply(x, y) takes two integers x and y as input and returns their product. The base case of the recursion is when either x or y is a single digit. Otherwise, the function splits the input integers into smaller halves and recursively computes the product of these halves. Finally, the smaller products are combined to form the final product.

Note that the time complexity of the Divide and Conquer method for multiplication is O(n^log2(3)), where n is the number of digits in the input integers. This is faster than the naive approach of multiplying each digit in one integer by each digit in the other integer, which has a time complexity of O(n^2). However, for very large integers, more efficient algorithms, such as the Karatsuba algorithm or the Schonhage-Strassen algorithm, may be needed to achieve reasonable performance.

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

The Knapsack problem is a classic optimization problem in which a set of items must be selected for inclusion in a knapsack, subject to a weight constraint, in order to maximize the total value of the selected items. The Greedy method is a simple and intuitive approach to solving this problem, which involves selecting the items with the highest value-to-weight ratio first, until the weight constraint is reached.

Here's an implementation of the Greedy method for solving the Knapsack problem in Python:

python

```
def knapsack_greedy(values, weights, capacity): # Calculate the value-to-weight
ratios for each item ratios = [(value/weight, value, weight) for value, weight in
zip(values, weights)] # Sort the items by value-to-weight ratio in decreasing
orderratios.sort(reverse=True) # Initialize the total value and weight of the
selected itemstotal_value = 0 total_weight = 0 # Select items with highest value-
to-weight ratio until capacity is reached for ratio, value, weight in ratios: if
total_weight + weight <= capacity: total_value += value total_weight += weight #
Return the total value of the selected items return total_value
```

The function `knapsack_greedy(values, weights, capacity)` takes three lists as input: `values`, which contains the value of each item, `weights`, which contains the weight of each item, and `capacity`, which is the maximum weight that the knapsack can hold. The function first calculates the value-to-weight ratio for each item and sorts the items in decreasing order of this ratio. It then iterates over the sorted items, selecting each item with the highest value-to-weight ratio that can be added to the knapsack without exceeding the weight capacity. Finally, the function returns the total value of the selected items.

Note that the Greedy method does not always produce an optimal solution to the Knapsack problem. In some cases, selecting items with the highest value-to-weight ratio first can lead to a suboptimal solution, since it does not take into account the possibility of selecting a combination of items that yields a higher total value. However, the Greedy method can be a useful heuristic for solving the Knapsack problem when a quick and simple solution is needed.

4. Implement Gaussian elimination method.

Gaussian elimination is a popular method used in linear algebra to solve a system of linear equations. The method involves transforming the system of equations into an equivalent system that has the same solutions, but which is in a simpler form that is easier to solve. This is achieved by performing a series of row operations on an augmented matrix that represents the system of equations.

Here's an implementation of the Gaussian elimination method in Python:

python

```
def gaussian_elimination(augmented_matrix):
    # Get the number of rows and columns in the augmented matrix
    n_rows, n_cols = augmented_matrix.shape

    # Perform row operations to transform the augmented matrix into row echelon form
    for i in range(n_rows):
        # Find the row with the largest absolute value in the i-th column
        max_row = i
        for j in range(i+1, n_rows):
            if abs(augmented_matrix[j,i]) > abs(augmented_matrix[max_row,i]):
                max_row = j

        # Swap the current row with the row with the largest absolute value
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

        # Subtract multiples of the current row from the rows below it to eliminate the i-th column
        for j in range(i+1, n_rows):
            factor = augmented_matrix[j,i] / augmented_matrix[i,i]
            augmented_matrix[j,i+1:] -= factor * augmented_matrix[i,i+1:]
            augmented_matrix[j,i] = 0

    # Perform back substitution to solve for the unknowns
    solutions = [0] * n_rows
    for i in range(n_rows-1, -1, -1):
        solutions[i] = (augmented_matrix[i,-1] - sum(augmented_matrix[i,i+1:-1] * solutions[i+1:])) /
augmented_matrix[i,i]

    return solutions
```

The function gaussian_elimination(augmented_matrix) takes an augmented matrix as input, where the last column of the matrix represents the constants in the system of equations. The function first performs row operations on the augmented matrix to transform it into row echelon form. This involves finding the row with the largest absolute value in each column and swapping the current row with that row, then subtracting multiples of the current row from the rows below it to eliminate the corresponding column. Once the augmented matrix is in row echelon form, the function performs back substitution to solve for the unknowns.

Note that the Gaussian elimination method can fail if the matrix is singular, meaning that it has no unique solution. In practice, it's a good idea to check for singularity before applying Gaussian elimination. Additionally, for very large matrices, more efficient algorithms such as LU decomposition may be needed to achieve reasonable performance.

5. Implement LU decomposition

LU decomposition is a popular method used in linear algebra to factorize a square matrix into a lower triangular matrix and an upper triangular matrix. The factorization is useful in many applications, including solving systems of linear equations and calculating determinants.

Here's an implementation of the LU decomposition method in Python, using the Doolittle algorithm:

python

```
import numpy as np

def lu_decomposition(matrix):
    # Get the number of rows and columns in the matrix
    n = matrix.shape[0]

    # Initialize the L and U matrices
    L = np.eye(n)
    U = np.zeros((n, n))

    # Perform LU decomposition using the Doolittle algorithm
    for i in range(n):
        # Calculate the i-th row of U
        for j in range(i, n):
            U[i,j] = matrix[i,j] - sum(L[i,k] * U[k,j] for k in range(i))

        # Calculate the i-th column of L
        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
```

The function lu_decomposition(matrix) takes a square matrix as input and returns the L and U matrices of the LU decomposition. The function first initializes the L matrix to the identity matrix and the U matrix to all zeros. It then performs the LU decomposition using the Doolittle algorithm, which involves calculating the i-th row of U and the i-th column of L for each i. The calculation of U[i,j] involves subtracting the sum of the products of corresponding elements in L and U matrices, while the calculation of L[j,i] involves dividing the difference between matrix[j,i] and the sum of the products of corresponding elements in L and U matrices by U[i,i]. Once the LU decomposition is complete, the function returns the L and U matrices.

6. Implement Warshall algorithm

The Warshall algorithm is a popular method used to find the transitive closure of a directed graph. The transitive closure of a graph is another graph that contains all the pairs of vertices for which there is a path from the first vertex to the second vertex in the original graph. The Warshall algorithm is a dynamic programming algorithm that works by computing the transitive closure of the graph in stages, starting with the vertices that have a direct connection and working outward.

Here's an implementation of the Warshall algorithm in Python:

python

```
def warshall_algorithm(graph):
    n = len(graph)
    closure = [[False] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            closure[i][j] = graph[i][j]

    for k in range(n):
        for i in range(n):
            for j in range(n):
                closure[i][j] = closure[i][j] or (closure[i][k] and closure[k][j])

    return closure
```

The function warshall_algorithm(graph) takes an adjacency matrix representing a directed graph as input and returns the transitive closure of the graph as another adjacency matrix. The function first initializes the closure matrix to the same values as the input graph. It then performs the Warshall algorithm by iterating over all possible intermediate vertices k and updating the closure matrix for each pair of vertices (i, j) by setting closure[i][j] to True if either there is a direct connection between i and j or there is a path from i to j through k.

Note that the Warshall algorithm has a time complexity of O(n^3), where n is the number of vertices in the graph. For very large graphs, more efficient algorithms such as Floyd-Warshall may be needed to achieve reasonable performance.

7. Implement the Rabin Karp algorithm,

The Rabin-Karp algorithm is a string searching algorithm that works by using a hash function to compare substrings of the text and pattern. The algorithm works by computing the hash value of the pattern and then sliding a window of the same length over the text, computing the hash value of each window, and comparing it to the hash value of the pattern. If the hash values match, the algorithm checks whether the actual characters in the window match the characters in the pattern.

Here's an implementation of the Rabin-Karp algorithm in Python:

python

```
def rabin_karp(text, pattern):
    # Define a prime number for hash function
    prime = 101

    # Get the length of the text and pattern
    n = len(text)
    m = len(pattern)

    # Calculate the hash value of the pattern
    pattern_hash = 0
    for i in range(m):
        pattern_hash += ord(pattern[i]) * (prime ** i)

    # Calculate the hash value of the first window in the text
    text_hash = 0
    for i in range(m):
        text_hash += ord(text[i]) * (prime ** i)

    # Slide the window over the text and compare hash values
    for i in range(n - m + 1):
        # If hash values match, check for actual character match
        if pattern_hash == text_hash:
            if text[i:i+m] == pattern:
                return i

        # Update the hash value for the next window
        if i < n - m:
            text_hash = (text_hash - ord(text[i]) + ord(text[i+m])) * prime

    # If pattern not found, return -1
    return -1
```

8. Implement the KMP algorithm/ Implement Harspool algorithm

Knuth-Morris-Pratt (KMP) algorithm

The KMP algorithm is a string searching algorithm that works by precomputing a prefix table for the pattern to avoid redundant comparisons when a mismatch occurs.

python

```
def kmp_search(text, pattern):
    n = len(text)
    m = len(pattern)

    # Build prefix table for pattern
    prefix = [0] * m
    i, j = 0, 1
    while j < m:
        if pattern[i] == pattern[j]:
            i += 1
            prefix[j] = i
            j += 1
        elif i > 0:
            i = prefix[i-1]
        else:
            j += 1

    # Perform string search
    i, j = 0, 0
    while i < n:
        if text[i] == pattern[j]:
            i += 1
            j += 1
            if j == m:
                return i - m
        elif j > 0:
            j = prefix[j-1]
        else:
            i += 1

    return -1
```

Boyer-Moore-Horspool algorithm

The Horspool algorithm is a simplified version of the Boyer-Moore algorithm that only considers the rightmost character of the current window when a mismatch occurs.

python

```
def horspool_search(text, pattern):
    n = len(text)
    m = len(pattern)
```

```
# Build bad character table for pattern
bad_char = {}
for i in range(m-1):
    bad_char[pattern[i]] = m - i - 1

# Perform string search
i = m - 1
while i < n:
    j = m - 1
    while text[i] == pattern[j]:
        if j == 0:
            return i
        i -= 1
        j -= 1
    i += max(bad_char.get(text[i], m), 1)

return -1
```

9. Implement max-flow problem.

The maximum flow problem is a classic optimization problem in which we want to find the maximum flow that can be sent through a network from a source node to a sink node. Here's one way to implement the Edmonds-Karp algorithm, which is an efficient algorithm for solving the maximum flow problem:

python

```
from collections import deque

def edmonds_karp(graph, source, sink):
    # Initialize residual graph
    residual = {u: {v: graph[u][v] for v in graph[u]} for u in graph}

    # Initialize flow and parents
    flow = 0
    parents = {u: None for u in graph}

    while True:
        # Use BFS to find an augmenting path
        queue = deque([source])
        parents[source] = source
        while queue:
            u = queue.popleft()
            for v in residual[u]:
                if parents[v] is None and residual[u][v] > 0:
                    parents[v] = u
                    queue.append(v)
                    if v == sink:
                        break
        if parents[sink] is None:
            break
        # Find bottleneck capacity
        bottleneck = float('inf')
        v = sink
        while v != source:
            u = parents[v]
            bottleneck = min(bottleneck, residual[u][v])
            v = u

        # Update flow and residual graph
        flow += bottleneck
        v = sink
        while v != source:
            u = parents[v]
            residual[u][v] -= bottleneck
            residual[v][u] += bottleneck
            v = u
        # Reset parents
        parents = {u: None for u in graph}
    return flow
```