Rayat Shikshan Sanstha's
KARMAVEER BHAURAO PATIL COLLEGE, VASHI
[Autonomous College]
Reaccredited by NAAC with Grade 'A+' (CGPA 3.53) | ISO 9001 : 2008 Certified Institute
'Best College' Award by University of Mumbai

Analysis of Algorithm

Name: - Vinit Rajendra Ghodekar.

Class: - MSC-CS I

Roll No: - 240524

Subject: - Analysis of Algorithm.

INDEX

SR.NO	PRACTICAL	DATE	SIGN
1	Write a program to implement insertion sort and find the running time of the algorithm.	25-07-2024	
2	Write a program to implement a merge sort algorithm. Compare the time and memory complexity.	01-08-2024	
3	Write a program to implement Longest Common Subsequence (LCS) algorithm	22-08-2024	
4	Write a program to implement Huffman’s code algorithm	29-08-2024	
5	Write a program to implement Kruskal’s algorithm	19-09-2024	
6	Write a program to implement Dijkstrass’s algorithm	26-09-2024	
7	write a program to implement prim’s algorithm	03-10-2024	
8	Write a program to implement Euclid’s algorithm to implement gcd of two non negative integers a and b. Extend the algorithm to find x and y such that gcd(a,b) = ax+by. Compare the running time and recursive calls made in each case.	10-10-2024	
9	Write a program to verify (i) Euclid’s theorem (ii) Fermat’s theorem	17-10-2024	
10	Write a program to implement greedy set cover	24-10-2024	

Practical No 1

AIM: Write a program to implement insertion sort and find the running time of the algorithm.

CODE:

```
def insertionSort(a):
    for i in range(1, len(a)):  # Start from the second element
        temp = a[i]
        j = i - 1
        while j >= 0 and temp < a[j]:
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = temp
def printArr(a):
    for i in range(len(a)):
        print(a[i], end=" ")
    print()
def main():
    a = [70, 15, 2, 51, 60]
    print("Before sorting algorithm:")
    printArr(a)
    insertionSort(a)
    print("\nAfter sorting algorithm:")
    printArr(a)
if__name__== "__main__":
    main()
```

OUTPUT:

```
= RESTART: C:/Users/ADMIN/Desktop/algo pr 1.py
Before sorting algorithm:
70 15 2 51 60

After sorting algorithm:
2 15 51 60 70
```

Practical No 2

AIM: Write a program to implement a merge sort algorithm. Compare the time and memory complexity.

CODE:

```
def mergeSort(arr):
   if len(arr) <= 1:
      return arr
   mid = len(arr) // 2
   leftHalf = arr[:mid]
   rightHalf = arr[mid:]
   sortedLeft = mergeSort(leftHalf)
   sortedRight = mergeSort(rightHalf)
   return merge(sortedLeft, sortedRight)
def merge(left, right):
   result = []
   i = j = 0
   while i < len(left) and j < len(right):
      if left[i] < right[j]:
         result.append(left[i])
         i += 1
      else:
         result.append(right[j])
         j += 1
   result.extend(left[i:])
   result.extend(right[j:])
   return result
unsortedArr = [3, 7, 6, 10, 15, 23, 55, 13]
sortedArr = mergeSort(unsortedArr)
print("Sorted array:", sortedArr)
```

OUTPUT:

```
= RESTART: C:/Users/ADMIN/Desktop/algo pr 2.py
Sorted array: [3, 6, 7, 10, 13, 15, 23, 55]
```

Practical No 3

AIM: Write a program to implement Longest Common Subsequence (LCS) algorithm

CODE:

```
def lcs_algo(S1, S2, m, n):
    L = [[0 for x in range(n+1)] for x in range(m+1)]
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0:
                L[i][j] = 0
            elif S1[i-1] == S2[j-1]:
                L[i][j] = L[i-1][j-1] + 1
            else:
                L[i][j] = max(L[i-1][j], L[i][j-1])
    index = L[m][n]
    lcs_algo = [""] * (index+1)
    lcs_algo[index] = ""
    i = m
    j = n
    while i > 0 and j > 0:
        if S1[i-1] == S2[j-1]:
            lcs_algo[index-1] = S1[i-1]
            i -= 1
            j -= 1
            index -= 1
        elif L[i-1][j] > L[i][j-1]:
            i -= 1
        else:
            j -= 1
    print("S1 : " + S1 + "\nS2 : " + S2)
    print("LCS: " + "".join(lcs_algo))
S1 = "ABCDEF"
S2 = "CDGEABE"
m = len(S1)
n = len(S2)
lcs_algo(S1, S2, m, n)
```

OUTPUT:

```
= RESTART: C:/Users/ELE.LAB 03 PC NO 01/AppData/Local/Programs/Python/Python312/
LCS.py
S1 : ABCDEF
S2 : CDGEABE
LCS: CDE
```

Practical No 4

AIM: Write a program to implement Huffman's code algorithm

CODE:

```
import heapq
class node:
    def_init_(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''
    def__lt__(self, nxt):
        return self.freq < nxt.freq
def printNodes(node, val=''):
    newVal = val + str(node.huff)
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")
chars = ['a', 'b', 'c', 'd', 'e', 'f']
freq = [ 5, 9, 12, 13, 16, 45]
nodes = []
for x in range(len(chars)):
    heapq.heappush(nodes, node(freq[x], chars[x]))
while len(nodes) > 1:
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)
    left.huff = 0
    right.huff = 1
    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
    heapq.heappush(nodes, newNode)
printNodes(nodes[0])
```

OUTPUT:

```
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

Practical No 5

AIM: Write a program to implement Kruskal's algorithm

CODE:

```
class Graph:
	def __init__(self, vertices):
		self.V = vertices
		self.graph = []
	def addEdge(self, u, v, w):
		self.graph.append([u, v, w])
	def find(self, parent, i):
		if parent[i] != i:
			parent[i] = self.find(parent, parent[i])
		return parent[i]
	def union(self, parent, rank, x, y):
		if rank[x] < rank[y]:
			parent[x] = y
		elif rank[x] > rank[y]:
			parent[y] = x
		else:
			parent[y] = x
			rank[x] += 1
	def KruskalMST(self):
		result = []
		i = 0
		e = 0
		self.graph = sorted(self.graph,
						key=lambda item: item[2])
		parent = []
		rank = []
		for node in range(self.V):
			parent.append(node)
			rank.append(0)
		while e < self.V - 1:
			u, v, w = self.graph[i]
			i = i + 1
			x = self.find(parent, u)
			y = self.find(parent, v)
			if x != y:
				e = e + 1
				result.append([u, v, w])
				self.union(parent, rank, x, y)
```

```
        minimumCost = 0
        print("Edges in the constructed MST")
        for u, v, weight in result:
            minimumCost += weight
            print("%d -- %d == %d" % (u, v, weight))
        print("Minimum Spanning Tree", minimumCost)
if__name__== '__main__':
    g = Graph(4)
    g.addEdge(0, 1, 10)
    g.addEdge(0, 2, 6)
    g.addEdge(0, 3, 5)
    g.addEdge(1, 3, 15)
    g.addEdge(2, 3, 4)
    g.KruskalMST()
```

OUTPUT:

```
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19

=== Code Execution Successful ===
```

Practical No 6

AIM: Write a program to implement Dijkstrass's algorithm

CODE:

```
class Graph():
	def __init_(self, vertices):
		self.V = vertices
		self.graph = [[0 for column in range(vertices)]
					for row in range(vertices)]
	def printSolution(self, dist):
		print("Vertex \t Distance from Source")
		for node in range(self.V):
			print(node, "\t\t", dist[node])
	def minDistance(self, dist, sptSet):
		min = 1e7
		for v in range(self.V):
			if dist[v] < min and sptSet[v] == False:
				min = dist[v]
				min_index = v
		return min_index
	def dijkstra(self, src):
		dist = [1e7] * self.V
		dist[src] = 0
		sptSet = [False] * self.V
		for cout in range(self.V):
			u = self.minDistance(dist, sptSet)
			sptSet[u] = True
			for v in range(self.V):
				if (self.graph[u][v] > 0 and
				sptSet[v] == False and
				dist[v] > dist[u] + self.graph[u][v]):
					dist[v] = dist[u] + self.graph[u][v]
		self.printSolution(dist)
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
		[4, 0, 8, 0, 0, 0, 0, 11, 0],
		[0, 8, 0, 7, 0, 4, 0, 0, 2],
		[0, 0, 7, 0, 9, 14, 0, 0, 0],
		[0, 0, 0, 9, 0, 10, 0, 0, 0],
		[0, 0, 4, 14, 10, 0, 2, 0, 0],
		[0, 0, 0, 0, 0, 2, 0, 1, 6],
		[8, 11, 0, 0, 0, 0, 1, 0, 7],
		[0, 0, 2, 0, 0, 0, 6, 7, 0]
		]
g.dijkstra(0)
```

OUTPUT:

```
Vertex   Distance from Source
0        0
1        4
2        12
3        19
4        21
5        11
6        9
7        8
8        14

=== Code Execution Successful ===
```

Practical No 7

AIM: write a program to implement prim's algorithm

CODE:

```
import sys

class Graph():
    def__init_(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    def minKey(self, key, mstSet):

        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    def primMST(self):

        key = [sys.maxsize] * self.V
        parent = [None] * self.V

        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1

        for cout in range(self.V):

            u = self.minKey(key, mstSet)
```

```
            mstSet[u] = True

            for v in range(self.V):

                if self.graph[u][v] > 0 and mstSet[v] == False \
                and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

        self.printMST(parent)

if__name__== '__main__':
    g = Graph(5)
    g.graph = [[0, 2, 0, 6, 0],
               [2, 0, 3, 8, 5],
               [0, 3, 0, 0, 7],
               [6, 8, 0, 0, 9],
               [0, 5, 7, 9, 0]]

    g.primMST()
```

OUTPUT:

```
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Practical No 8

AIM: Write a program to implement Euclid's algorithm to implement gcd of two non negative integers a and b. Extend the algorithm to find x and y such that gcd(a,b) = ax+by. Compare the running time and recursive calls made in each case.

CODE:

```
def gcd_iterative(a,b):
    while b != 0:
        a,b =b, a % b
        return a

def extended_gcd_iterative(a,b):
    x0,x1,y0,y1 = 1,0,0,1
    while b !=0:
        q, a, b = a // b, b, a % b
        x0, x1 =x1, x0 - q * x1
        y0, y1 =y1, y0 - q * y1
        return a, x0, y0

def gcd_recursive(a,b):
    if b == 0:
        return a
    return gcd_recursive(b, a % b)

def extended_gcd_recursive(a,b):
    if b == 0:
        return a, 1, 0
    gcd, x1, y1 = extended_gcd_recursive(b, a % b)
    x = y1
    y = x1 - (a//b)*y1
    return gcd, x ,y

a = 56
b = 98
gcd_iter = gcd_iterative(a, b)
gcd_ext_iter, x_iter, y_iter = extended_gcd_iterative(a,b)

gcd_rec = gcd_recursive(a, b)
gcd_ext_rec, x_rec, y_rec = extended_gcd_recursive(a,b)

print("iterative gcd: ", gcd_iter)
print("Extended iterative gcd: ", gcd_ext_iter, "x:", x_iter,  "y:", y_iter)
```

```
print("recursive gcd: ", gcd_rec)
print("Extended recursive gcd: ",gcd_ext_rec, "x:", x_rec,  "y:", y_rec)

import time

def timed_gcd (func, a, b) :
    start_time = time.time()
    result = func (a,b)
    end_time = time.time()
    return result, end_time - start_time

gcd_iter_time = timed_gcd(gcd_iterative, a, b)
gcd_rec_time = timed_gcd(gcd_recursive, a, b)

print("iterative Gcd time: ", gcd_iter_time[1])
print("recursive Gcd time: ", gcd_rec_time[1])
```

OUTPUT:

```
iterative gcd:  98
Extended iterative gcd:  98 x: 0 y: 1
recursive gcd:  14
Extended recursive gcd:  14 x: 2 y: -1
iterative Gcd time:  0.0
recursive Gcd time:  0.0
```

Practical No 9

AIM: Write a program to verify (i) Euclid’s theorem (ii) Fermat’s theorem

CODE:

```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def generate_primes(limit):
    primes = []
    for num in range(2, limit):
        if is_prime(num):
            primes.append(num)
    return primes

def euclids_theorem(limit):
    primes = generate_primes(limit)
    product = 1
    for prime in primes:
        product *= prime
    new_prime_candidate = product + 1

    if is_prime(new_prime_candidate):
        return f"New prime found: {new_prime_candidate}"
    else:
        return f"{new_prime_candidate} is not a prime number, but Euclid's method finds
infinitely many primes."

# Example usage
limit = 10
result = euclids_theorem(limit)
print(result)
```

OUTPUT:

```
==================== RESTART: C:/Users/admin/Desktop/10.py
New prime found: 211
```

(ii) Fermat's theorem

```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
def fermats_little_theorem(a,p):
    if is_prime(p):
        if pow(a,p-1,p)==1:
            return f"Fermet's theorem holds for a = {a} and prime p = {p}"
        else:
            return f"Fermet's theorem does not hold for a = {a} and prime p = {p}"
    else:
        return f"{p} is not a prime number."
a=3
p=7
result= fermats_little_theorem(a,p)
print(result)
```

OUTPUT:

```
Fermet's theorem holds for a = 3 and prime p = 7
```

Practical No 10

AIM: Write a program to implement greedy set cover

CODE:

```
def greedy_set_cover(universe, sets):
    covered = set()
    cover = []

    while covered != universe:
        best_set = max(sets, key=lambda s: len(s - covered), default=None)
        if best_set is None:
            break
        cover.append(best_set)
        covered.update(best_set)
        sets.remove(best_set)

    return cover

# Example usage
if __name__ == "__main__":
    universe = {1, 2, 3, 4, 5}
    sets = [{1, 2}, {2, 3}, {3, 4}, {4, 5}, {1, 5}]

    selected_sets = greedy_set_cover(universe, sets)
    print("Selected sets to cover the universe:", selected_sets)
```

OUTPUT:

```
Selected sets to cover the universe: [{1, 2}, {3, 4}, {4, 5}]

=== Code Execution Successful ===
```