Advanced Data Structures Project - Documentation

Team Members:
Gunanicaa Arun 205002035
Mayuri Balajee 205002047

Problem Statement

One of the main difficulties faced by many people is planning a proper trip. To solve this issue, the proposed project is a trip planner which gives people the optimised travel plan.

The input data given by the users are the following:

- Number of passengers
- Date of Journey
- Origin location
- Destination location

From the given input data, we formulate the most efficient itinerary, which includes most important places to visit keeping in mind the constraints like time availability, date of journey, number of people. To implement this project, the following data structures will be used.

1. Graphs:

Graph is a non-linear data structure consisting of nodes and edges. Here in this case, we use graphs to compute the shortest path between two locations. The optimum plan with what places to visit will be suggested (taking into account shortest distance between places) , however if the user wants to change the places to visit, the most efficient path will be suggested to the user. The algorithm for computing the distance uses graphs as we calculate the value for each path using weighted graphs.

2. Hash Table:

The list of places to visit will be stored in a hash table. The hash table keys will hence be stored in a priority queue. This is done as each place will have its corresponding key value, and hence easier to objectively choose the more important place to visit.

3. Priority Queue:

Priority queue is used to store all the nodes in the graph along with their distance to the starting node. And hence we get the node with the minimum distance as until the priority queue is not empty we extract the node with the current shortest known distance to our starting node.

Another usage of the priority queue will be storing the hash keys in priority queue. These hash keys correspond to different places, and hence giving the hash key as priority queue elements, each hash key will have a priority value. This priority value will help to find out the higher priority place to visit.

Hence, this project will act as a solution to a wide range of audience, ranging from school or college students to elderly people. Our application implements various data structures to effectively store data and use the same and computes an efficient itinerary.

Implementation:

Program Code:

Module 1:

```
# Function to print an array
def printArray(arr, n):

     #printing the hashtable(Array)
     for i in range(n):
          print(arr[i], end = " ")

# quadratic probing
def hashing(table, tsize, arr, N):

     for i in range(N):

          hv = len(arr[i]) % tsize

          if (table[hv] == -1):
               table[hv] = arr[i]
```

```
        else:

            for j in range(tsize):

                t = (hv + j * j) % tsize

                if (table[t] == -1):

                    table[t] = arr[i]
                    break

    printArray(table, L)



# Driver code
arr = [ "Marina-Beach","VGP-Snow-Kingdom","Mahabalipuram","EA-
Mall","Santhome-Church",
    "Pondy_Bazar","Kabali_Temple","Senmozhi_Poonga",
"Vandaloor_Zoo","Muttukadu-Boat-House"]

N = 10

# Size of the hash table
L = 2*N

hash_table = [0] * L

# Initializing the hash table
for i in range(L):
    hash_table[i] = -1

hashing(hash_table, L, arr, N)

index_arr=[None]*L
j=0
```

```
for i in range(0,L):
     if (hash_table[i]!=-1):
          index_arr[j]=i
          j+=1

print("\n\nIndex Array: ")
for i in range(0,len(index_arr)):
     if (index_arr[i]!=None):
          print(index_arr[i], end = " ")
#print(index_arr)'''

class PriorityQueueNode:

 def _init_(self, value, pr):

    self.data = value
    self.priority = pr
    self.next = None

# Implementation of Priority Queue
class PriorityQueue:

    def _init_(self):

        self.front = None

    def isEmpty(self):

        return True if self.front == None else False

    def push(self, value, priority):

        if self.isEmpty() == True:

            self.front = PriorityQueueNode(value,
```

```
                                  priority)

        return 1

    else:
        if self.front.priority > priority:

            newNode = PriorityQueueNode(value,
                                        priority)

            newNode.next = self.front
            self.front = newNode
            return 1

        else:
            temp = self.front

            while temp.next:

                if priority <= temp.next.priority:
                    break

                temp = temp.next

            newNode = PriorityQueueNode(value,
                                        priority)
            newNode.next = temp.next
            temp.next = newNode

            return 1

def pop(self):


    if self.isEmpty() == True:
```

```
            return

        else:


            self.front = self.front.next
            return 1



    def peek(self):

        if self.isEmpty() == True:
            return
        else:
            return self.front.data

    def traverse(self):



        if self.isEmpty() == True:
            return "Queue is Empty!"
        else:
            temp = self.front
            while temp:
                print(temp.data, end = "\n")
                temp = temp.next

# Driver code
if _name_ == "_main_":
    print("\nTop 10 places to visit in Chennai( based on priority
queue)")

    pq = PriorityQueue()

    pq.push(arr[0], 1,)
```

```
    pq.push(arr[1], 2)
    pq.push(arr[4], 3)
    pq.push(arr[5], 4)
    pq.push(arr[8], 5)
    pq.push(arr[9], 6)
    pq.push(arr[2], 7)
    pq.push(arr[3], 8)
    pq.push(arr[7], 9)
    pq.push(arr[6], 10)

    pq.traverse()

    pq.pop()
```

Module 2:

```
#RANDOM NUMBERS
import random
# Add a vertex to the set of vertices and the graph
def add_vertex(v):
  global graph
  global vertices_no
  global vertices
  if v in vertices:
    print("Vertex ", v, " already exists")
  else:
    vertices_no = vertices_no + 1
    vertices.append(v)
    if vertices_no > 1:
        for vertex in graph:
            vertex.append(0)
    temp = []
    for i in range(vertices_no):
        temp.append(0)
```

```
    graph.append(temp)

# Add an edge between vertex v1 and v2 with edge weight e
def add_edge(v1, v2, e):
    global graph
    global vertices_no
    global vertices
    # Check if vertex v1 is a valid vertex
    if v1 not in vertices:
        print("Vertex ", v1, " does not exist.")
    # Check if vertex v1 is a valid vertex
    elif v2 not in vertices:
        print("Vertex ", v2, " does not exist.")
    # Since this code is not restricted to a directed or
    # an undirected graph, an edge between v1 v2 does not
    # imply that an edge exists between v2 and v1
    else:
        index1 = vertices.index(v1)
        index2 = vertices.index(v2)
        graph[index1][index2] = e

# Print the graph
def print_graph():
  global graph
  global vertices_no
  for i in range(vertices_no):
    for j in range(vertices_no):
      if graph[i][j] != 0:
        print(vertices[i], " -> ", vertices[j], \
        " edge weight: ", graph[i][j])

# Driver code
# stores the vertices in the graph

def final():
```

```
    vertices = []
    # stores the number of vertices in the graph
    vertices_no = 0
    graph = []
    # Add vertices to the graph
    add_vertex(1)
    add_vertex(2)
    add_vertex(3)
    add_vertex(4)
    add_vertex(5)
    add_vertex(6)
    add_vertex(7)
    add_vertex(8)
    add_vertex(9)
    add_vertex(10)
    # Add the edges between the vertices by specifying
    # the from and to vertex along with the edge weights.
    k =
[3,7,9,4,3,2,3,4,5,6,7,11,34,66,89,34,55,10,9,2,34,6,2,95,38,24,20
]
    for i in range(11):
        for j in range(11):

            add_edge(i,j,random.choice(k))



    print_graph()
    return graph
    #print("Internal representation: ", graph)

#-------------------------



# Function to print an array
def printArray(arr, n):
```

```
     #printing the hashtable(Array)
     for i in range(n):
          print(arr[i], end = " ")

# quadratic probing
def hashing(table, tsize, arr, N):


     for i in range(N):


          hv = len(arr[i]) % tsize


          if (table[hv] == -1):
               table[hv] = arr[i]


          else:


               for j in range(tsize):


                    t = (hv + j * j) % tsize


                    if (table[t] == -1):


                         table[t] = arr[i]
                         break
     print("\nList of places are stored in a Hash Table: \n")
     print("Hash Table Contents: \n")
     printArray(table, L)



# Driver code for hash table
arr = [ "Marina-Beach","VGP-Snow-Kingdom","Mahabalipuram","EA-
Mall","Santhome-Church",
     "Pondy_Bazar","Kabali_Temple","Senmozhi_Poonga",
"Vandaloor_Zoo","Muttukadu-Boat-House"]
```

```
N = 10

# Size of the hash table
L = 2*N

hash_table = [0] * L

# Initializing the hash table
for i in range(L):
     hash_table[i] = -1

hashing(hash_table, L, arr, N)

index_arr=[None]*L
j=0
for i in range(0,L):
     if (hash_table[i]!=-1):
          index_arr[j]=i
          j+=1

print("\n\nList of Hash-Keys: (Basically list of Indices where the
values are stored)\n")
for i in range(0,len(index_arr)):
     if (index_arr[i]!=None):
          print(index_arr[i], end = " ")
#print(index_arr)

#TSP
from sys import maxsize
from itertools import permutations
V = N

def travellingSalesmanProblem(graph, s):

     # store all vertex apart from source vertex
```

```
vertex = []
for i in range(V):
     if i != s:
          vertex.append(i)

# store minimum weight Hamiltonian Cycle
min_path = maxsize
next_permutation=permutations(vertex)

for i in next_permutation:
     #print(next_permutation)

     # store current Path weight(cost)
     current_pathweight = 0

     # compute current path weight
     k = s
     for j in i:
          current_pathweight += graph[k][j]
          #print(i)
          k = j
     current_pathweight += graph[k][s]

     # update minimum
     min_path = min(min_path, current_pathweight)
     perm_arr.append(i)
     min_vals.append(min_path)

#print(min_vals)
#print(perm_arr)
x=0
for a in range(0,len(min_vals)):
     if min_vals[a]==min_path:
          x=a
          break
```

```
    #printing the path
    print("\n\n",s,end=" ")
    for y in range(0,V-1):
        print(" -> ",perm_arr[x][y],end=" ")
    print(" ->",s,end=" ")

    '''#storing the path in an array
    path_arr=[0]*(V+1)
    path_arr[0]=0
    for i in range(0,V-1):
        path_arr[i+1]=perm_arr[x][i]
        #print("\n\n",perm_arr[x][i])
    print(path_arr)

    #accessing the hash table contents
    final_arr=[]
    for p in range(0,len(path_arr)):
        q=path_arr[p]
        index_arr[q]!=None
        r=index_arr[p]
        final_arr.append(hash_table[r])'''

    return min_path



# matrix representation of graph
graph = [[6, 3, 34, 4, 3, 66, 4, 34, 6, 10],
          [20, 34, 34, 34, 2, 9, 7, 6, 2, 5],
          [38, 11, 3, 4, 89, 34, 34, 3, 34, 10],
          [7, 34, 3, 66, 7, 2, 89, 7, 55, 34],
          [9, 2, 38, 7, 24, 9, 3, 34, 11, 2],
          [11, 4, 20, 9, 4, 95, 34, 34, 10, 6],
          [9, 38, 66, 3, 4, 20, 3, 11, 5, 55],
          [3, 7, 55, 2, 9, 6, 2, 3, 9, 95],
          [10, 20, 4, 66, 2, 6, 34, 7, 89, 55],
```

```
            [20, 34, 2, 66, 89, 9, 38, 55, 3, 95]]

z=index_arr[0]
s = 0
min_vals=[]
perm_arr=[]

print("\n\nThe final order of places to visit such that it has the minimum cost is: ")
travellingSalesmanProblem(graph, s)

print("\n\nThe numbers here represent the following places: \n")

for x in range(0,len(index_arr)):
        if (index_arr[x]!=None):
            y=index_arr[x]
            print(x,hash_table[y])
```

Executed Output:

```
List of places are stored in a Hash Table:

Hash Table Contents:

Muttukadu-Boat-House -1 -1 -1 -1 -1 -1 EA-Mall -1 -1 -1 Pondy_Bazar Marina-Beach Mahabalipuram Kabali_Temple Santhome-Church VGP-Snow-Kingdom Vandaloor_Zoo -1 Senmozhi_Poonga

List of Hash-Keys: (Basically list of Indices where the values are stored)

0 7 11 12 13 14 15 16 17 19
```

```
List of factors which will afect your trip:
1.Distance
2.Budget
3.Familiy friendly
Enter Choice: 1

Top 10 places to visit in Chennai( based on priority queue)
Marina-Beach
VGP-Snow-Kingdom
Santhome-Church
Pondy_Bazar
Vandaloor_Zoo
Muttukadu-Boat-House
Mahabalipuram
EA-Mall
Senmozhi_Poonga
Kabali_Temple
```

```
List of factors which will afect your trip:
1.Distance
2.Budget
3.Familiy friendly
Enter Choice: 2

Top 10 places to visit in Chennai( based on priority queue)
Kabali_Temple
Pondy_Bazar
Santhome-Church
EA-Mall
Vandaloor_Zoo
Muttukadu-Boat-House
Marina-Beach
Mahabalipuram
Senmozhi_Poonga
VGP-Snow-Kingdom
```

```
List of factors which will afect your trip:
1.Distance
2.Budget
3.Familiy friendly
Enter Choice: 3

Top 10 places to visit in Chennai( based on priority queue)
Mahabalipuram
Vandaloor_Zoo
Santhome-Church
Marina-Beach
Muttukadu-Boat-House
VGP-Snow-Kingdom
Senmozhi_Poonga
Kabali_Temple
EA-Mall
Pondy_Bazar
```

```
The final order of places to visit such that it has the minimum cost is:


 0  ->  6  ->  3  ->  5  ->  1  ->  8  ->  4  ->  9  ->  2  ->  7  -> 0

The numbers here represent the following places:

0 Muttukadu-Boat-House
1 EA-Mall
2 Pondy_Bazar
3 Marina-Beach
4 Mahabalipuram
5 Kabali_Temple
6 Santhome-Church
7 VGP-Snow-Kingdom
8 Vandaloor_Zoo
9 Senmozhi_Poonga
PS C:\Users\Mayuri\Desktop\ITA2\ads proj>
```

Conclusion:

The list of places to visit in Chennai have been stored in a **hash table**. This project displays the top 10 places to visit in Chennai based on the factor chosen by the user. Hence here **priority queues** has been implemented.

Further more, based on weights, the best and shortest path to visit these places have also been computed with the help of **Graphs** using the Travelling Salesman Algorithm and the path has been suggested to the user.