## **PROBLEM 6**

## Implement All Pair Shortest paths problem using Floyd's algorithm.

## **ALGORITHM**

```
Afforsilhm:

April: grouph [v][v] - adjaceny makes of grouph

The of mether

INE - large number.

Output. drif [v][v] -> make x of should distance broadle rain

Instratize drif malrix:

for [=0 to v-1

for j=0 to v-1

drif [i][j] = grouph [i][j]

2. For each intermediate needen k from 0 to v-1:

for each source needen i from 0 to v-1:

for each destination needen j from 0 to v-1:

of drif [i][j] = drif [i][j] from 0 to v-1:

Af drif [i][j] = drif [i][j] from 0 to v-1:
```

```
CODE
```

```
printf("%7s", "INF");
        else
          printf("%7d", dist[i][j]);
     }
     printf("\n");
// Floyd-Warshall algorithm
void floydWarshall(int graph[V][V]) {
  int dist[V][V];
  // Initialize the solution matrix with input graph
  for (int i = 0; i < V; i++)
     for (int j = 0; j < V; j++)
       dist[i][j] = graph[i][j];
  // Update dist[i][j] to the shortest path
  for (int k = 0; k < V; k++) {
     for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
          if (dist[i][k] + dist[k][j] < dist[i][j])
             dist[i][j] = dist[i][k] + dist[k][j];
        }
  printSolution(dist);
```

```
// Driver code
int main() {
    // Example graph with 4 vertices
    int graph[V][V] = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };

floydWarshall(graph);

return 0;
}
```

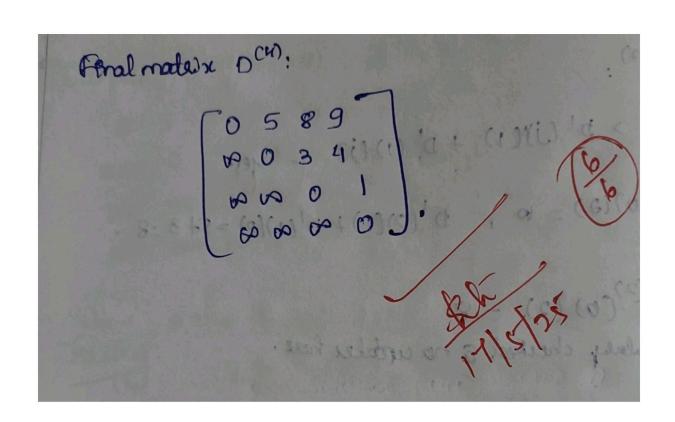
# OUTPUT:

```
Shortest distances between every pair of vertices:
            5
                  8
     0
                         9
            0
                  3
   INF
                         4
   INF
         INF
                  0
                         1
   INF
          INF INF
                         0
```

#### **TRACING**

```
Jearing:
Instral graph:
                       0
                               1 1000
                    00 .
                            10
                        0 00 00
    000 (1)(1) = min (DE-1 (1)(1), DE-1 (1)(E) + DE-1 (1)(1)
9=11.j=3 har a [M] [3] - 17 181 3ab
D°(11(2)=3
00(1)(0)+00(0)(0)= w+ m= m = no upadale
1=0 , 3=0.
            A LOW THE THE WAY
 D°(0)(2)= 50
 0°(0)(0) + 0°(0)(0) =0+0= 60 -100 apoleb
  100 uposte unce going strough o doesn't impose any defaut
```

```
update mater to oth).
 check of D' (1)(1) > D' (1)(1) + D' (1)(1) - yes
 1=0 j= > p(1)(0)(a) = w , p'(0)(1) = 0'(1)(0) = 5+3 =8 .
 8< 00, ardale p(0)(0) (0) = 8.
  nee other pains, umber cheeked to no captales have.
9 E 2
 entable to o'3):
 : i=0 j=3
   D2 (0)(3) =10, D2(0)(1) + D2 (1)(3) =2+1=9.
   9210 =) p3(0)(9=9.
e = 1, j=3:
         0°(1)(3) = 6 , D2 (1)(2) + 0°(2)(3) =3+1=4
              D3 (1)(3) = 4
54: K= 3 .
cupden to bear.
  when checked p3(i)(j) > p3(1)(3) + p3(3)(j)
   i=0, j=3 p(0)(3) = 9 p(0)(1)+p(3)(3)=9+0 = 9 - 10m2
   Similarly when cheeted for remaining weeker, the condition
```



#### LEETCODE 5

## NUMBER OF WAYS TO ARRIVE AT DESTINATION

## **ALGORITHM**

```
* Number of ways to assend out a destination
 Rendo ade:
 Initialle graph ar adjacency list
Initialle enentime [n] = 10 for all nodes
Initialle ways [n] = 0 for all nodes
 Set men time to J= 0
 set ways to = 1 = the state of the or and the state
 Crade min-heap paronty queue pa & insert (0,0) - (time, node)
 while pop is not empty:
      time, u = pr. poper a co tobb says we at a con a col tob
      new time + t
        Il newtime < mintime [v]:
            12. puch (Cnew Time, vi)
              min time [v] = newtime
         Elee of newtine == montime [v].
               Laye PVJ = ( ways FVJ + ways FUT) mod (1ea+ +)
Return ways [n-1]
```

#### **CODE**

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define MOD 1000000007

typedef struct {
   int to, weight;
} Edge;
```

```
typedef struct {
  Edge* edges;
  int size, capacity;
} AdjList;
typedef struct {
  long long time;
  int node;
} HeapNode;
typedef struct {
  HeapNode* arr;
  int size;
} MinHeap;
void swap(HeapNode* a, HeapNode* b) {
  HeapNode tmp = *a;
  *a = *b;
  *b = tmp;
}
void heapify_up(MinHeap* h, int i) {
  while (i > 0 \&\& h-\arr[i].time < h-\arr[(i - 1) / 2].time) {
    swap(\&h->arr[i], \&h->arr[(i-1)/2]);
    i = (i - 1) / 2;
  }
}
void heapify_down(MinHeap* h, int i) {
  int left, right, smallest;
```

```
while (2 * i + 1 < h > size) {
     left = 2 * i + 1;
     right = 2 * i + 2;
     smallest = i;
     if (left < h->size && h->arr[left].time < h->arr[smallest].time)
       smallest = left;
     if (right < h->size && h->arr[right].time < h->arr[smallest].time)
       smallest = right;
     if (smallest == i) break;
     swap(&h->arr[i], &h->arr[smallest]);
     i = smallest;
  }
}
void push(MinHeap* h, long long time, int node) {
  h->arr[h->size++] = (HeapNode){time, node};
  heapify up(h, h->size - 1);
}
HeapNode pop(MinHeap* h) {
  HeapNode top = h->arr[0];
  h\rightarrow arr[0] = h\rightarrow arr[--h\rightarrow size];
  heapify down(h, 0);
  return top;
}
int countPaths(int n, int** roads, int roadsSize, int* roadsColSize) {
  // Step 1: Build graph
```

```
AdjList* graph = malloc(n * sizeof(AdjList));
for (int i = 0; i < n; ++i) {
  graph[i].edges = malloc(100 * sizeof(Edge));
  graph[i].size = 0;
  graph[i].capacity = 100;
}
for (int i = 0; i < roadsSize; ++i) {
  int u = roads[i][0], v = roads[i][1], w = roads[i][2];
  graph[u].edges[graph[u].size++] = (Edge)\{v, w\};
  graph[v].edges[graph[v].size++] = (Edge)\{u, w\};
}
// Step 2: Initialize minTime and ways
long long* minTime = malloc(n * sizeof(long long));
int* ways = malloc(n * sizeof(int));
for (int i = 0; i < n; ++i) {
  minTime[i] = LLONG_MAX;
  ways[i] = 0;
}
minTime[0] = 0;
ways[0] = 1;
// Step 3: Min-heap priority queue
MinHeap heap;
heap.arr = malloc(100000 * sizeof(HeapNode));
heap.size = 0;
push(&heap, 0, 0); // (time, node)
while (heap.size > 0) {
```

```
HeapNode curr = pop(&heap);
  long long time = curr.time;
  int u = curr.node;
  if (time > minTime[u]) continue;
  for (int i = 0; i < graph[u].size; ++i) {
    int v = graph[u].edges[i].to;
    int t = graph[u].edges[i].weight;
     long long newTime = time + t;
     if (newTime < minTime[v]) {</pre>
       minTime[v] = newTime;
       ways[v] = ways[u];
       push(&heap, newTime, v);
     } else if (newTime == minTime[v]) {
       ways[v] = (ways[v] + ways[u]) \% MOD;
int result = ways[n - 1];
// Clean up
for (int i = 0; i < n; ++i) free(graph[i].edges);
free(graph);
free(minTime);
free(ways);
free(heap.arr);
```

```
return result;
```

## **OUTPUT**

```
Accepted Runtime: 0 ms
Accepted Runtime: 0 ms
                                                • Case 1 • Case 2
• Case 1 • Case 2
                                                Input
Input
                                                 n =
 n =
                                                 2
 roads =
                                                [[1,0,10]]
 [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3
 ,1],[6,5,1],[2,5,1],[0,4,5],[4,6,2]]
                                                Output
Output
                                                 1
                                                Expected
Expected
                                                 1
```

# **TRACING**

```
Trockno
                    8n: n= 4
                      roods [(0.1.7),
 [00,00,00,00] - tello
                               (O,2, D,
  heep = [0.07
                       (a, 3, n),
(c, a, a)
 1) por(0,0)
 = neghouse : 1 (w+ 2), 0 (c+2)
 cheek na :
  dist FIJ = 10 > 0 + 1 - update det FIJ = 1 ways DT = ways to ] = )
    put cities and a ser and process and and
 check me:
   do (0) = 00 > 0 +0 - apple do (0) = 0, way (0) = 1
   bmp (3'3)
Now: dut = [0,1,2,6] 2) Pop (1,1)
                             no: d(0) = 0<1+1(2) -> 7
  heap = [ (1.17 , (2.2 )]
                    03/d(3)=+0>(+>(3)-)
n: 0 (12), 1 (12), 3 (14) dis [01,12,3)
37 pop (2,5)
                            waye = (4,2,17
teap = (412) (0,2)]
 * dto7 = 0 < 2 + 2 cm - no expolate
check n 3:
   d (3)=7= 2+1 (3) - equal defances - mays (3)+= ways (3) = 120
```

```
Now: dist = [0,1,2,3]

eagle = [1,1,2,3]

heap = (3,3)

4 n: 2(M n), 1 (M s)

n e: d(2) = 2 < 3+1 (4) - no update

n 2: d(1) = 1 < 3+2 (5) - no update

Heep - emply - rend.

Result:

Shorted defane from 0 to 3 is 3

Number of shorters paths ways (3) = 3
```