Airports Management System

The problem statement is regarding a Airports Management System. There are different airports characterized by a unique ID (integer) and an Airport Name (String). These airports are represented by the structure airport as can be seen below:

```
struct airport
{
     int num_id;
     const char* airport_name;
};
typedef struct airport airport_t;
```

The array of structures (of type airport_t) represents all the airports and their respective information.

For Example:

```
airport_t airports[] = {{0,"BLR"}, {1,"MUM"}, {{2,"DEL"}};
```

In this above array of structures, airports[0] represent information about the 1st airport that has the num_id - 0 and airport_name - “BLR”.

The different airports are connected to each other by means of edges. These form a graph where the airports are the nodes and the edges between the airports represent the connections. The connection between two airports is characterized by the distance between the airports and the time taken to travel from one airport to another. The edges between the airports are represented by the structure connection as shown below:

```
struct connection
{
     int distance;
     int time;
};
typedef struct connection connection_t;
```

The graph itself will be represented by an adjacency matrix. It must be noted that the num_id field of the airport will be contiguous (It will range from 0 to the (Total number of Airports - 1)). **The num_id field acts as indices in this adjacency matrix. Please look at the table below.**

Connection from an airport to itself (self loop) will be represented by **{0,0}** in the adjacency matrix. **Self loops are not to be considered as edges.**

If a particular airport cannot be reached from an airport, that connection will be represented by **{INT_MAX, INT_MAX}**. This represents that the distance between the airports is infinite and the time taken to reach the airport is infinite. Hence, this signifies that you cannot travel from the former to the latter airport.

For Example: Consider the following adjacency matrix representing a graph where there are 3 airports. The num_id fields of the 3 airports will be 0,1,2 respectively. The first row of this adjacency matrix represents the edges of airport 0 with other airports and so on for the remaining rows. For instance, adj_mat[0][1] represents the connection from airport with id 0 to airport with id 1. The distance between the 2 is 1 unit and the time taken is 2 units. Here, {0, 0} values in the adjacency matrix represent the self loops which should not be considered as edges in the algorithms you write. It should also be noted here that it is not possible to reach airport 2 **from** airport 0 as the value of adj_mat[0][2] is {INT_MAX, INT_MAX}.

```
connection_t adj_mat[3][3] = {
                        {{0,0},{1,2},{INT_MAX,INT_MAX}},
                        {{1,1},{0,0},{2,4}},
                        {{2,3},{2,3},{0,0}}
                        };
```

To --> From | V	BLR		MUM		DEL	
	Distance	Time	Distance	Time	Distance	Time
BLR	0	0	1	2	INT_MAX	INT_MAX
MUM	1	1	0	0	2	4
DEL	2	3	2	3	0	0

We will be using the following structure as the return type for some of the questions.

```
struct pair
{
     int first;
     int second;
};
typedef struct pair pair_t
```

General Constraints for the input graph

- Edges are non negative in terms of both distance and time
- Unless explicitly stated otherwise in the constraints section of a question, you should assume that the graph is **directed, not necessarily connected, and not necessarily complete**

Problem Statements

1. Q1

People love to travel. So when designing connections between airports we need to make sure we can cater to the wanderlust of all travelers. So given the airport connections graph, write a function to check if a traveler can start at any airport and reach any other airport in the given graph, either directly or indirectly. This should be true for all the airports in the given graph.

Input

- connections: A 2-D adjacency matrix which is a graph representing the connections between the airports
- n: the number of airports in the graph

Output

If it is possible for a traveler to be able to go from any airport to any other airport, directly or indirectly, and this being true for all airports, return 1, else return 0.

Constraints

- 1 <= n <= 10^3

Interface

```
/*
 * @param n: Total number of airports
 * @param connections: Adjacency matrix which is a graph
representing the connections between the airports
 *
 * @return value: 1 if it is possible for a traveler to be able
to go from any airport to any other airport, directly or
indirectly, and this being true for all airports, else 0
 */

int q1(int n, const connection_t connections[n][n]);
```

Question Samples

A. **Sample Input**:

connection_t connections[3][3] = {

{{0,0}, {1,2}, {2,3}},

{{1,2}, {0,0}, {2,3}},

{{2,3}, {2,3}, {0,0}}

};

n = 3

Sample Output: 1

Explanation: For the given graph, it is possible to start at a given airport and reach all other airports, either directly or indirectly. This is true for all the airports shown as part of the graph. Therefore, the output is 1.

B. **Sample Input:**

connection_t connections[3][3] = {

{{0,0}, {INT_MAX, INT_MAX}, {3,3}},

{{INT_MAX, INT_MAX}, {0,0}, {2,3}},

{{INT_MAX, INT_MAX}, {2,3}, {0,0}}

};

n = 3

Sample output: 0

Explanation: For the given graph, it is not possible to go to an airport with id 0 from any airport.

2. Q2

As an airline, we need to ensure smooth travel for our customers. So we have decided to check if from a given source airport to a destination airport, the traveler will need to change more than some “k” flights. Write a function which returns if at most k flights are sufficient to travel between the source and destination airports.

Note: Flights here refer to the edges in the connection matrix.

Input

- connections: A 2-D adjacency matrix which is a graph representing the connections between the airports
- n: the number of airports in the graph
- k: The maximum number of flights
- src: The airport from where we start
- dest: The airport which we expect to reach

Output

Return a boolean value. Return 1 if you can go from source airport to destination airport using at most k flights, otherwise return 0.

Constraints

- 1 <= n <= 10^3
- 1 <= k <= n
- src is never equal to dest

Interface

```
/*
 * @param src: The source airport
 * @param dest: The destination airport
 * @param n: Total number of airports
 * @param k: The maximum number of flights
 * @param connections: Adjacency matrix which is a
   graph representing the connections between the airports
 *
 * @return value: boolean variable indicating if it is possible
to reach dest from src using at most k flights
```

```
 */

int q2(const airport_t *src, const airport_t *dest, int n, int k,
const connection_t connections[n][n]);
```

Question Samples

A. Sample Input:

connection_t connections[3][3] = {

{{0,0},{INT_MAX, INT_MAX},{1,5}},

{{1,2}, {0,0}, {4,3}},

{{2,3}, {2,3}, {0,0}}

};

n = 3

k = 2

src = {0,"BLR"}

dest = {1,”MUM”}

Sample output: 1

Explanation: For the given graph, it is possible to start at source airport “BLR” and reach the destination airport “MUM” with just 2 flights via airport {2,”DEL”}. Therefore, the output is 1.

B. Sample Input:

connection_t connections[4][4] = {

{{0,0},{INT_MAX, INT_MAX},{1,5}, {INT_MAX, INT_MAX}},

{{1,2}, {0,0}, {4,3}, {1, 4}},

{{INT_MAX, INT_MAX}, {INT_MAX, INT_MAX}, {0,0}, {2, 3}}

{{1,3}, {2,4}, {INT_MAX, INT_MAX}, {0, 0}}

};

n = 4

k = 2

src = {0,"BLR"}

dest = {1, "MUM”}

Sample output: 0

Explanation:

For the given graph, it is only possible to start at source airport “BLR” and reach the destination airport “MUM” with at least 3 flights via 0 -> 2 -> 3 -> 1, therefore, the output is 0.

3. Q3

As an airport management system, it is our duty to not only make our customers travel smoothly, but also to ensure that they can reach back to the airport from which they started. After all, everyone wants to go back home!!
Write a function that checks if for a given source airport, a traveler can return to their source airport.

Input

- connections: A 2-D adjacency matrix which is a graph representing the connections between the airports
- n: the number of airports in the graph
- src: The airport from where we start

Output

Return a boolean value. Return 1 if you can start from a given airport and come back to the same airport, otherwise return 0.

Constraints

- 1 <= n <= 10^3

Interface

```
/*
 * @param src: Start and end airport
 * @param n: Total number of airports
 * @param connections: Adjacency matrix which is a
   graph representing the connections between the airports
 *
 * @return value: boolean variable indicating if it is possible
to start at the src airport and return to the src airport
 *
 */

int q3(const airport_t *src, int n, const connection_t
connections[n][n]);
```

Question Samples

A. **Sample input:**

connection_t connections[3][3] = {

{{0,0},{1, 2},{2,3}},

{{1,2}, {0,0}, {2,3}},

{{2,3}, {2,3}, {0,0}}

};

n = 3

src = {0,"BLR"}

Sample output:

1

Explanation: For the given graph, one possible route is to start at airport “BLR” (num_id = 0), go to the airport with num_id = 1, then go to the airport with num_id = 2 and then finally go back to the airport with num_id = 0 which is the source airport “BLR”. Therefore, it is possible to start at source airport “BLR” and return to the same airport “BLR”. Hence, output is 1.

4. Q4

As a large airport management system, we will always find ourself in the need for being able to sort the airports based on various criteria. Write a function that uses a given predicate and sorts the given array of airports based on the predicate. Make sure the sort is **in-place**. Use an algorithm that takes O(n*logn) time in the average case.

Input

- airport_list: An array of type airport_t
- n: The number of airports in airports_list
- predicate_func: A function pointer that compares 2 airports. If predicate(A,B) is true(not equal to 0), then A will appear before B in the sorted array.

Output

The array should be sorted in-place.

Constraints

- 1 <= n <= 10^6
- predicate_func will be O(1)

Interface

```
/*
 * @param n: Total number of airports
 * @param predicate_func: Function pointer to the comparator
function
 * @param airport_list: list of all airports
 *
```

```
 * The array must be sorted *in-place* according to the predicate
after the function call
*/

void q4(int n, int (*predicate_func)(const airport_t*, const
airport_t*), airport_t airport_list[n]);
```

Question Samples

A. Sample Input:

airport_list = {{0, “BLR”}, {1, “SUR”}, {2, “DEL”}, {3, “NEL”}, {4, “MUM”}};
n = 5

```
int predicate_func(airport_t* x, airport_t* y)
{
     return strcmp(x->airport_name, y->airport_name) < 0;
}
```

Sample Output:

airport_list = {{0, “BLR”},{2, “DEL”}, {4, “MUM”}, {3, “NEL”}, {1, “SUR”}};

Explanation:

The predicate sorts the names of airports in ascending order.

B. Sample Input:

airport_list = {{0, “BLR”}, {1, “SUR”}, {2, “DEL”},{3, “NEL”}, {4, “MUM”}};
n = 5

```
int predicate_func(const airport_t* x, const airport_t* y)
{
     return strcmp(x->airport_name, y->airport_name) > 0;

}
```

Sample Output:

airport_list = {{1, “SUR”}, {3, “NEL”}, {4, “MUM”}, {2, “DEL”}, {0, “BLR”}};

Explanation:

The predicate sorts the names of airports in descending order.

C. **Sample Input:**

airport_list = {{0, “BLR”}, {4, “MUM”}, {10, “NEL”}, {11, “SUR”}, {22, “DEL”}}
n = 5

```
int predicate_func(const airport_t* x, const airport_t* y)
{
     return (x->num_id % 10) < (y->num_id %10);
}
```

Sample Output:

airport_list = {{0, “BLR”}, {10, “NEL”}, {11, “SUR”}, {22, “DEL”}, {4, “MUM”}}

or

airport_list = {{10, “NEL”}, {0, “BLR”}, {11, “SUR”}, {22, “DEL”}, {4, “MUM”}}

Explanation:

The predicate sorts by the least significant digit of the id in ascending order.

5. Q5

Airports around the world have different names. Air Traffic Control needs to reference airports carefully and hence avoid any possible confusion. We would like to give Air Traffic Control the functionality to know which airports share the longest common prefix. Given the entire list of airports with their names and num_id. Find the longest common prefix shared between 2 airports. It may be unwise to check every possible pair of airports.

Input

- airports: An array of type struct airport_t
- n: The number of airports

Output

Return a pair containing the num_ids of the 2 airports that have the longest common prefix. The ids may be stored in any order within the pair. If none of the names have a common prefix return {-1,-1}.

Constraints

- 1 <= n <= 10^5
- 1 <= |airports[i].airport_name| <= 10^5

Interface

```
/*
 * @param n: The number of airports in the array airports
 * @param airports: An array of type airport containing the
details of various airports
 *
 * @return value: A pair containing the num_ids of the 2 airports
that have the longest common prefix names, return {-1,-1} if no
names have a common prefix. The IDs may be in any order within
the pair.
*/

pair_t q5(int n, airport_t airports[n]);
```

Question Samples

A. **Sample Input**:

   airport_t airports[] = {{0,"KOC"}, {1,"KIA"}, {2,"KOL"}};

   n = 3

   Sample Output: 0, 2 or 2, 0

Explanation: In the given set of airports, Airport 0 (“KOC”) and Airport 1 (“KIA”) have a common prefix (“K”) of length 1. Airport 0 (“KOC”) and Airport 2 (“KOL”) have a common prefix (“KO”) of length 2. Airport 1 (“KIA”) and Airport 2 (“KOL”) have a common prefix (“K”) of length 1. Thus, “KO” is the longest common prefix of length 2. Therefore, it is expected to return a pair with first field as 0 and second field as 2 or a pair with first field as 2 and second field as 0.

B. **Sample Input:**

airport_t airports[] = {{0,"BLR"}, {1,"MUM"}, {2,"KOL"}};

n = 3

Sample Output: -1, -1

Explanation: In the given set of airports, it can be seen that no pair of airports have a common prefix. Therefore, it is expected to return a pair with the first and second field as -1 and -1.

6. Q6

To manage crowds in an airport, we have decided to introduce an entry fee for every airport. While we want to reduce crowds in airports, we would also like to make airports accessible to people. So, we want you to write a function, given a **sorted** array of entry fees and an amount of money, return the number of airports that could be visited. A brute force solution is not advised.

Note: The visitor visits only one airport. Do not count the number of airports they can simultaneously visit. Do not accumulate the costs.

Input

- entry_fee: An **sorted** array of entry fees
- n: The number of airports
- amount: The money that a visitor has

Output

The number of airports that you may be able to visit given the amount of money available to you.

Constraints

- 1 <= n <= 10^6

Interface

```
/*
 * @param n: Total number of airports
 * @param amount: Amount of money you have
 * @param entry_fee: Sorted array containing entry fee of all airports
 *
 * @return_value: Number of airports you may be able to visit given the amount of money available to you
*/

int q6(int n, int amount, const int entry_fee[n]);
```

Question Samples

A. **Sample Input**:

   int entry_fee[] = {1, 2, 5, 10, 20, 25};

   n = 6

   amount = 10

   Sample Output: 4

   Explanation: Any one Airport with entry fee 1, 2, 5 and 10 can be visited.

B. **Sample Input:**

   int entry_fee[] = {1, 2, 5, 10, 20, 25};

   n = 6

   amount = 30

   Sample Output: 6

Explanation: Any one airport can be visited.

7. Q7

Travellers need to book their air travel, and to do this they first need to search the airports they would like to be their source and destination airports. Given a string typed by a traveller, find what airport names contain that string.

NOTE: Do not use brute force substring matching. Use a more optimized string matching algorithm that you have learnt in your DAA theory class. Using Horspool's pattern matching algorithm is a good idea.

Input

- airports: An array of type struct airport_t
- n: The number of airports in the array
- pat: A string pattern to look for in the names of the airport

Output

An array of size N, *contains*, will be passed by reference, all initialized to 0. You will need to set the value to 1 if the airport does contain the pattern pat in its name. Thus, if contains[i]=1, then airport with num_id=i contains pat in its name.

Constraints

- 1 <= n <= 10^3
- 1 <= |airports[i].airport_name| <= 10^5
- 0 <= |pat| <= 10^5

Interface

```
/*
 * @param n: Total number of airports
 * @param pat: Pattern to search in the list of airport names
 * @param contains: array of size n initialized to 0
 * @param airports: list of all airports
 *
```

```
 * At the end of the function, contains[i] should be 1 if the
airport with num_id i contains pat as a substring in its name
 */

void q7(int n, const char *pat, int contains[n], const airport_t
airports[n]);
```

Question Samples

A. Sample Input:

airport_t airports[] = {{0,"KOCHI"}, {1,"KIA"}, {2,"KOLKATA"}};

n = 3

pat = “KO”

Sample Output: [1, 0, 1]

Explanation: In the given set of airports, Airport 0 (“KOCHI”) contains the pattern “KO”, Airport 1 (“KIA”) does not contain the pattern “KO”, Airport 2 (“KOLKATA”) contains the pattern “KO”. Therefore, it is expected to modify the *contains* array such that the index 0 and index 2 representing Airports 0 and 2 contain the value 1. Therefore, the resultant array should be [1, 0, 1].

B. Sample Input:

airport_t airports[] = {{0,"BLR"}, {1,"MUM"}, {2,"KOL"}};

n = 3

pat = “U”

Sample Output: [0, 1, 0]

Explanation: In the given set of airports, Airport 0 (“BLR”) does not contain the pattern “U”, Airport 1 (“MUM”) contains the pattern “U”, Airport 2 (“KOL”) does not contain the pattern “U”. Therefore, it is expected to modify the *contains* array such that the index 1 representing Airport 1 contains the value 1. Therefore, the resultant array should be [0, 1, 0].

8. Q8

International Air Transport Association (IATA), an international regulatory body, has asked the airport management systems team to provide them with answers to a few questions. Can a traveler travel to every airport in the world **once** and return to their start airport? However, there is one restriction, **one airport must be exempted** from this trip. You are expected to find the shortest **distance** that needs to be traveled to complete this trip, while following all the mentioned restrictions. **There is no restriction on which airport is exempted** (In other words, any airport can be chosen to be exempted from this). Return the **least total distance** possible for this trip. If there exists no path at all such that all but one airport can be reached, return -1.

NOTE:

- **Only one airport can be exempted, no more and no less. There is no restriction on which airport is exempted**
- **There is no restriction on which airport the traveler needs to start from**

Input

- connections: An adjacency matrix which is a graph representing the connections between the airports
- n: The number of airports

Output

An array of size n-1, *trip_order*, which will be passed to function by reference which is initialized to -1's. You must fill up this array in the order in which you will visit the airports, you need not fill the last element of the trip which would be a return to the start point.

You must return the minimum cost of this round trip. If there are multiple round trips that have the same cost, any trip is accepted. If no trip exists, return -1.

Constraints

- 1 <= n <= 7
- The given graph is connected. All other general constraints of the graph still hold.

Interface

```
/*
 * @param n: Total number of airports
 * @param connections: Adjacency matrix which is a
   graph representing the connections between the airports
 * @param trip_order: An array of size n-1 which contains the
trip order initialized to -1s.
 *
 * @return value: The minimum cost of the round trip. If no such
trip exists, return -1.
 *
 * At the end of the function, trip order must contain the trip
in the minimum sequence.
 */

int q8(int n, int trip_order[n - 1], const connection_t
connections[n][n]);
```

Question Samples

A. **Sample Input**:

connection_t connections[4][4] = {

{ {0,0}, {7,2}, {3,5}, {INT_MAX, INT_MAX} },

{ {INT_MAX, INT_MAX}, {0,0}, {INT_MAX, INT_MAX}, {1,3} },

{ {INT_MAX, INT_MAX}, {2,4}, {0,0}, {4,7} },

{ {4,5}, {INT_MAX, INT_MAX}, {6,3}, {0,0} }

};

n = 4

trip_order = [-1, -1, -1]

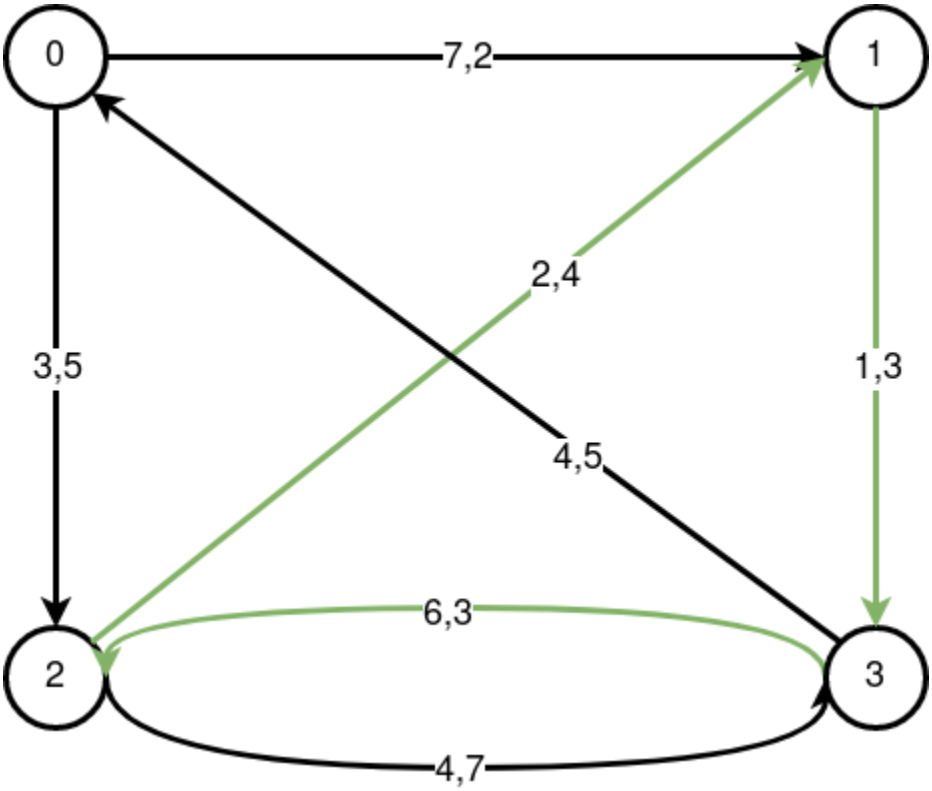
Sample Output:

Minimum Cost: 9

trip_order: [1,3,2] or [2,1,3] or [3,2,1]

Explanation:

The trip order of 1 -> 3 -> 2 -> 1 (here the numbers represent the Airport IDs) has a total distance of 9 units (1 + 6 + 2). The trip order of 2 -> 1 -> 3 -> 2 has a total distance of 9 units as well (2 + 1 + 6). The trip order of 3 -> 2 -> 1-> 3 has a total distance of 9 units as well (6 + 2 + 1). It can be seen that 9 units is the least distance that needs to be traveled such that you follow all the restrictions of the given problem statement (starting from any airport and returning to the same airport, visiting all other airports except one). As there are 3 such trips that have the same minimum distance, any of these trips are accepted as the answer.



B. Sample Input:

connection_t connections[3][3] = {

{ {0,0}, {1,2}, {INT_MAX, INT_MAX} },

{ {INT_MAX, INT_MAX}, {0,0}, {2,3} },

{ {3,4}, {INT_MAX, INT_MAX}, {0,0} }

};

n = 3

trip_order = [-1,-1]

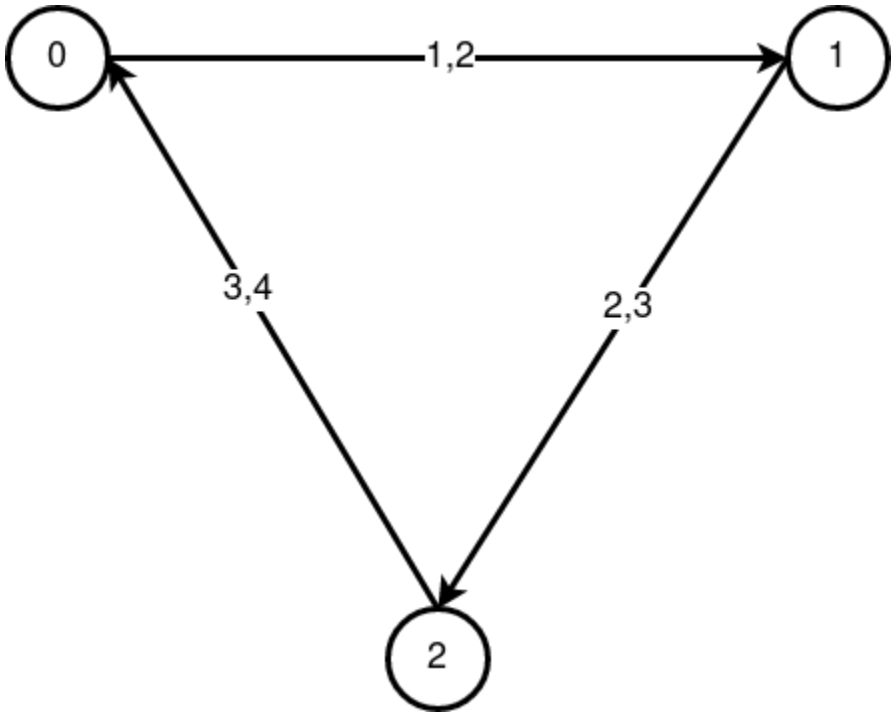
Sample Output:

Minimum Cost: -1

trip_order : [-1,-1]

Explanation:

In this graph, it is not possible to start at an airport and return to the same airport, visiting all other airports **except one**. Therefore, the minimum cost is -1 and the *trip_order* array remains unmodified.



9. Q9

International Air Transport Association (IATA), an international regulatory body, has asked the airport management systems team to provide them with answers to a few questions. They want to know if in a war scenario, where air travel is highly restricted, what would be the paths that are needed to be kept open such that **every airport is connected to one another** while **reducing the total time to be the minimum possible** to have these connections active? You are expected to return the minimum value and the list of connections that need to be kept active such that the total time of these edges is minimum.

Input

- connections: An adjacency matrix which is an **undirected graph** representing the connections between the airports.
- n: The number of airports

Output

An array of pairs of length n-1 will be passed by reference to the function, where each element contains the two vertices the edge connects. This array should contain all the edges that need to be kept active when the war scenario is in effect. If a connection from airport i to airport j must be kept active, the edges array must contain pair({i,j}). **The edges may be present in any order. All equivalently minimal solutions are also accepted.**

You must also return the minimum sum of the total time.

Constraints

- The given graph will be undirected and connected. All other general constraints hold.
- 1 <= n <= 10^3

Interface

```
/*
 * @param n: Total number of airports
 * @param edges: Array of type pair_t of length n-1 initialized
to {-1,-1}.
 * @param connections: Adjacency matrix which is an undirected
```

```
   graph representing the connections between the airports
 *
 * @return value: Minimum total time
 *
 * At the end of the function must be filled with the edges
belonging to the solution. The edges are represented by a pair
with the start node and end node of the edge. The edges may be in
any order. The start and end order in the pair
 is not important as it is undirected. All equivalent solutions
are accepted.
 */

int q9(int n, pair_t edges[n - 1], const connection_t
connections[n][n]);
```

Question Samples

A. **Sample Input:**

connection_t connections[4][4] = {

{ {0,0}, {1,2}, {2,6}, {INT_MAX, INT_MAX} },

{ {1,2}, {0,0}, {3,5}, {4,1} },

{ {2,6}, {3,5}, {0,0}, {5,7} },

{ {INT_MAX, INT_MAX}, {4,1}, {5,7}, {0,0} }

};

n = 4

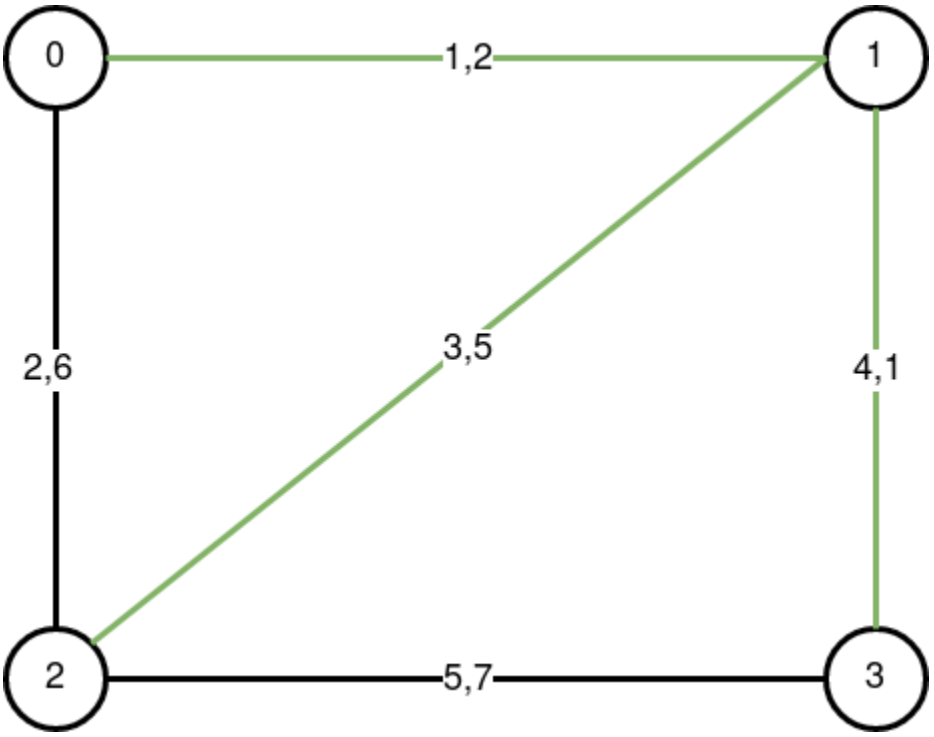
edges = { {-1, -1}, {-1,-1}, {-1,-1} }

Sample Output:

Minimum Total Time: 8
edges = { {0,1}, {1,2}, {1,3} } (Any order of edges is accepted as the solution)

Explanation:

It can be seen that the paths 0 – 1, 1 – 2 and 1 – 3 need to be kept open such that every airport is connected to one another and such that the total time taken across these connections is minimum - 8 units (2 + 5 + 1). Therefore, the minimum total time that needs to be returned is 8 units and the edges array is as shown above. Note that any ordering of the edges is accepted as a solution. For example, { {1,3}, {0,1}, {1,2} } is also accepted as a valid solution. It should also be noted that the ordering within the pair can be anything as well. That is, as it is an undirected graph, edge 0 – 1 can either be represented as {0,1} or {1,0} in the solution array. Both the orders will be accepted.



10. Q10

Our customers love to travel, and when deciding where to plan their next holiday they would like to compare the travel time for different possible destinations. We would like to aid our customers in this search for a destination. So for a given source and a set of destinations, return the minimum time required to travel to these destinations.

Input

- connections: An adjacency matrix which is a graph representing the connections between the airports.
- n: The number of airports
- destinations: An array of integers of length k which indicates the airports that must be visited with minimal cost
- k: The number of destinations
- src: A variable of type airport_t which acts as the source to the destinations

Output

An array costs of length k, initialized to INT_MAX will be passed by reference. This array must be filled with the minimum time for the corresponding airport destination.

Constraints

- 1 <= n <= 10^3
- 1 <= k <= n

Interface

```
/*
 * @param n: Total number of airports
 * @param k: Number of destination airports
 * @param src: The source of travel to the destinations
 * @param connections: Adjacency matrix which is a
   graph representing the connections between the airports
 * @param destinations: Array of airport IDs that you will need
to find the
 minimum time for.
 * @param costs: An integer array of length k that will be
initialized to
```

```
 INT_MAX
 *
 * At the end of the function, costs must be filled with the
minimum cost for
 the corresponding destination airport. costs[i] must be the
minimum time taken
 to travel to destination[i].
 */

void q10(int n, int k, const airport_t* src, const connection_t
connections[n][n], const int destinations[k], int costs[k]);
```

Question Samples

A. Sample Input

src = {3,"BLR"}

connection_t connections[4][4] = {

{ {0,0}, {1,2}, {2,6}, {INT_MAX, INT_MAX} },

{ {1,2}, {0,0}, {3,5}, {4,1} },

{ {2,6}, {3,5}, {0,0}, {5,7} },

{ {INT_MAX, INT_MAX}, {4,1}, {5,7}, {0,0} }

};

n = 4

k = 2

destinations = [0, 2]

costs = [INT_MAX, INT_MAX]

Sample Output

costs = [3, 6]

Explanation

Starting from airport 3, the fastest time to reach airport 0 is via airport 1. 3->1->0 takes a time of 3 units. To reach airport 2, the direct travel time is 7, whereas via airport 1 it is reduced to 6. 3->1->2 costs 6 time units. Hence the cost for 0 is 3 and the cost for 2 is 6.
Please understand the costs array, destination[0] is 0, hence costs[0] is the cost for destination[0](which is airport 0) and it is 3. costs[1] is the cost for destination[1](which is airport 2) and it is 6.