CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6651: Algorithm Design Techniques

Winter 2022

Final - Close book exam - 3 hours

Instructor: Professor B. Jaumard

First Name	**Last Name**	**ID#**

- In order for your answer (in particular yes/no answer) be considered, you need to provide a justification.

	a.	**b.**	**c.**	
Question 1				
Question 2				
Question 3				
Question 4				
Question 5				
Total				

Question 1. (30 points.)

Assuming we are considering a minimization problem.

a. Describe the various reduction tests (we discuss 4 during the lecture on B&B) which allow the pruning of a node in the search tree of a branch-and-bound method.

b. The traveling salesman problem (TSP) is defined as follows. Given a number of cities and the costs of traveling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city? Propose a branch-and-bound method to solve the TSP. In order to do so, you need to design:

- An algorithm for computing a lower bound on the optimal value
- An algorithm for computing an upper bound on the optimal value
- A branching (separation) scheme
- An exploration scheme

You need to provide a general description of your algorithms or schemes, an illustration on an example cannot replace a general algorithm description.

c. Develop the first 4 nodes of the search tree of your algorithm on the example described in the table below.

	Berlin	London	Paris	Moscow	Rome	Seville
Berlin		8	12	7	10	15
London	8		18	3	12	11
Paris	12	18		19	20	27
Moscow	7	3	19		9	8
Rome	10	12	20	9		13
Seville	15	11	27	8	13	

a. 4 points.

Pruning a node k in the search tree of a B&B algorithm.
Lower bound $\underline{z}$ is associated with the optimal solution of a relaxation
Upper bound $\overline{z} = z^{\text{BEST}}$ is associated with a heuristic (feasible) solution

Test 1. 1 point. The lower bound is equal to the upper bound: It implies we have solved to optimality the current subproblem (but not the initial problem).

Test 2. 1 point. The current subproblem is infeasible.

Test 3. 1 point. The vector associated with the lower bound is feasible AND the objective function of the relaxation is equal to the objective function of the optimization problem: It implies that the lower bound is indeed the optimal value of the current subproblem.

Test 4. 1 point. The lower bound of a subproblem is greater than the value of the incumbent value: No need to solve exactly the current subproblem as its optimal value cannot better than the incumbent one.

b. 6 points.

b1. 2 points. An algorithm for computing a lower bound on the optimal value
For each node v, compute the sum s_v of the distances to the two closest cities to v. Lower bound: $(1/2) \times \sum_{v \in V} s_v$.

b2. 2 points. An algorithm for computing an upper bound on the optimal value
Select a city, say v_1. Look for the closest, yet unvisited city, say v_2. Look for the closest, yet unvisited city, to either v_1 or v_2, say v_3. And so. We get a tour v_1, v_2, v_3, ..., v_n, and back to v_1 whose length defines an upper bound. To improve the upper bound, we can do the same computations, starting each time from a different city, and then keep the best upper bound.

b3. 1 point. A branching (separation) scheme
Look for the smallest distance linking either two unvisited cities, or a pair of two cities, one visited and one unvisited. Say the shortest distance links v and v'. First branch: v follows v'. Second branch: v does not follow v'.

b4. 1 point. An exploration scheme.
Best first search, the next node to be visited is the node with the worst lower bound.

c. 20 points.

Development of the first 4 nodes of the search tree of your algorithm on the example.

4 points for the computation of the bounds in each node: 2 for the upper bound, 2 for the lower bound.
An overall 4 points for the branching and the pruning processes.

First lower bound.

City	Two closest cities	Sum of the distances to the two closest cities
Berlin	London, Moscow	8 + 7 = 15
London	Berlin, Moscow	8 + 3 = 11
Paris	Berlin, London	12 + 18 = 29
Moscow	Berlin, London	7 + 3 = 10
Rome	Berlin, Moscow	10 + 9 = 19
Seville	London, Moscow	11 + 8 = 19
Total		104

Lower bound = $\underline{z_0} = 104/2 = 52$

First upper bound

- Starting city: Berlin
- Closest city to Berlin: Moscow. Tour: Berlin $\mapsto$ Moscow. Current length of the tour: 7
- Closest city to either Berlin or Moscow: London.

$$\text{London} \xleftrightarrow{\text{Choice 1: 8}} \text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \xleftrightarrow{\text{Choice 2: 3}} \text{London}$$

  Tour: Berlin $\leftrightsquigarrow$ Moscow $\leftrightsquigarrow$ London. Current length of the tour: 10

- Closest city to either Berlin or London: Rome.

$$\text{Rome} \xleftrightarrow{\text{Choice 1: 10}} \text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \overset{3}{\longleftrightarrow} \text{London} \xleftrightarrow{\text{Choice 2: 11}} \text{Seville}$$

  Tour: Rome $\leftrightsquigarrow$ Berlin $\leftrightsquigarrow$ Moscow $\mapsto$ London. Current length of the tour: 20.

- Closest city to either Rome or London: Seville.

$$\text{Seville} \xleftrightarrow{\text{Choice 1: 13}} \text{Rome} \overset{10}{\longleftrightarrow} \text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \overset{3}{\longleftrightarrow} \text{London} \xleftrightarrow{\text{Choice 2: 11}} \text{Seville}$$

  Tour: Rome $\leftrightsquigarrow$ Berlin $\leftrightsquigarrow$ Moscow $\leftrightsquigarrow$ London $\leftrightsquigarrow$ Seville. Current length of the tour: 31.

- Closest city to either Rome or Seville: Moscow.

$$\text{Paris} \xleftrightarrow{\text{Choice 1: 20}} \text{Rome} \overset{10}{\longleftrightarrow} \text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \overset{3}{\longleftrightarrow} \text{London} \overset{11}{\longleftrightarrow} \text{Seville} \xleftrightarrow{\text{Choice 2: 27}} \text{Paris}$$

  Tour: Paris ⇝ Rome ⇝ Berlin ⇝ Moscow ⇝ London ⇝ Seville. Current length of the tour: 51.

- Link the two extreme cities. Length of the tour: 78.

Upper bound $= \overline{z}_0 = z^{\text{BEST}} = 78$.

Upper bound: an alternate way (not to be used in the sequel)

- Starting city: Berlin
- Closest city to Berlin: Moscow.
  Tour: Berlin $\mapsto$ Moscow. Length of the partial tour: 7
- Closest city to Moscow: London.

$$\text{Berlin} \quad \overset{7}{\longleftrightarrow} \quad \text{Moscow} \quad \overset{3}{\longleftrightarrow} \quad \text{London}$$

  Tour: Berlin ⇝ Moscow ⇝ London.
  Length of the partial tour: 10
- Closest city to London: Seville.

$$\text{Berlin} \quad \overset{7}{\longleftrightarrow} \quad \text{Moscow} \quad \overset{3}{\longleftrightarrow} \quad \text{London} \quad \overset{11}{\longleftrightarrow} \quad \text{Seville}$$

  Tour: Berlin ⇝ Moscow ⇝ London ⇝ Seville.
  Length of the partial tour: 21.
- Closest city to Seville: Rome.

$$\text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \overset{3}{\longleftrightarrow} \text{London} \overset{11}{\longleftrightarrow} \text{Seville} \overset{13}{\longleftrightarrow} \quad \text{Rome}$$

  Tour: Berlin ⇝ Moscow ⇝ London ⇝ Seville ⇝ Rome.
  Length of the partial tour: 34.
- Closest city to Rome: Paris.

$$\text{Berlin} \overset{7}{\longleftrightarrow} \text{Moscow} \overset{3}{\longleftrightarrow} \text{London} \overset{11}{\longleftrightarrow} \text{Seville} \overset{13}{\longleftrightarrow} \text{Rome} \overset{13}{\longleftrightarrow} \text{Rome} \overset{20}{\longleftrightarrow} \text{Paris}$$

  Tour: Berlin ⇝ Moscow ⇝ London ⇝ Seville ⇝ Rome ⇝ Paris. Length of the partial tour: 54.
- Link the two extreme cities. Length of the tour: 66.

Two branches: consider the two closest cities, i.e., Moscow and London
Branch 1: London follows immediately Moscow, i.e., Moscow $\mapsto$ London
Branch 2: London does not follow Moscow (be careful that "London $\mapsto$ Moscow" is not the complement of "Moscow $\mapsto$ London".

Node 1: Node 0 + Branch 1

Lower bound (does not change): 52

Upper bound

- Starting city: Moscow
- Branch 1 imposes: Moscow $\mapsto$ London.

- Closest city to either Moscow or London: Berlin. Tour: Berlin - Moscow - London. Current length of the tour: 10
- Closest city to either Berlin or London: Rome. Tour: Rome - Berlin - Moscow - London. Current length of the tour: 20
- Closest city to either Rome or London: Seville. Tour: Rome - Berlin - Moscow - London - Seville. Current length of the tour: 31.
- Closest city to either Rome or Seville: Moscow. Tour: Paris - Rome - Berlin - Moscow - London - Seville. Current length of the tour: 51.
- Link the two extreme cities. Length of the tour: 78.

Upper bound = $\overline{z}_1 = 78$. No improvement of the incumbent value.

Node 2: Node 0 + Branch 2

City	Two closest cities	Sum of the distances to the two closest cities	
Berlin	London, Moscow	$8 + 7 = 15$	
London	Berlin, Moscow	$8 + 3 = 11$	
Paris	Berlin, London	$12 + 18 = 30$	
Moscow	Berlin, London	$7 + 3 = 10$	London is not excluded as Branch 2 $\equiv$ London $\not\mapsto$ Moscow Therefore it allows: London $\mapsto$ Moscow
Rome	Berlin, Moscow	$10 + 9 = 19$	
Seville	London, Moscow	$11 + 8 = 19$	
Total		104	

Lower bound = $\underline{z_2} = 104 = 52$.

- Starting city: Moscow
- Branch 2 imposes: Moscow $\not\mapsto$ London.
- Closest city to Moscow: London.
  Tour: London $\mapsto$ Moscow. Current length of the tour: 3
- Closest city to either Moscow or London: Berlin.
  Tour: London $\mapsto$ Moscow $\mapsto$ Berlin. Current length of the tour: 10
- Closest city to either Berlin or London: Rome.
  Tour: London $\mapsto$ Moscow $\mapsto$ Berlin $\mapsto$ Rome. Current length of the tour: 20.
- Closest city to either Rome or London: Seville.
  Tour: Seville $\mapsto$ London $\mapsto$ Moscow $\mapsto$ Berlin $\mapsto$ Rome. Current length of the tour: 31.
- Closest city to either Rome or Seville: Moscow.
  Tour: Seville $\mapsto$ London $\mapsto$ Moscow $\mapsto$ Berlin $\mapsto$ Rome $\mapsto$ Moscow. Current length of the tour: 51.
- Link the two extreme cities. Length of the tour: 78.

Upper bound = $\overline{z}_2 = 78$. No improvement of the incumbent value.

Next node to partition?
Select Node 1 (as the node with the worst lower bound) and consider the the closest city to either Moscow or London as we already have "Moscow $\mapsto$ London".

Branch 3: we have Berlin $\overset{7}{\mapsto}$ Moscow $\mapsto$ London $\overset{8}{\mapsto}$ Berlin,
therefore we choose "Berlin $\mapsto$ Moscow" so that we get: Berlin $\mapsto$ Moscow $\mapsto$ London

Branch 4: Moscow does not follow Berlin.

Node 3: Node 1 + (Branch 3 ≡ Berlin $\mapsto$ Moscow $\mapsto$ London)

City	Two closest cities	Sum of the distances to the two closest cities	
Berlin	~~London~~, Moscow, Rome	10 + 7 = 17	London is excluded by the conditions of the branches 1 & 3
London	~~Berlin~~, Moscow, Seville	8 + 11 = 19	Berlin is excluded by the conditions of the branches 1 & 3
Paris	Berlin, London	11 + 18 = 29	
Moscow	Berlin, London	7 + 3 = 10	
Rome	Berlin, ~~Moscow~~, London	10 + 12 = 22	
Seville	London, ~~Moscow~~, Rome	11 + 13 = 19	
Total		116	

Lower bound = $\underline{z_2}$ = 116/2 = 58.

Computation of the upper bound

Starting point: conditions of the various branching conditions
Berlin $\mapsto$ Moscow $\mapsto$ London

- Starting point: Berlin $\mapsto$ Moscow $\mapsto$ London
- Closest city to either Moscow or Berlin (London excluded): Rome.
  Tour: Rome $\overset{8}{\mapsto}$ Berlin $\overset{7}{\mapsto}$ Moscow $\overset{3}{\mapsto}$ London.
  Current length of the tour under construction: 18
- Closest city to either Rome or London: Seville.
  Rome $\overset{8}{\mapsto}$ Berlin $\overset{7}{\mapsto}$ Moscow $\overset{3}{\mapsto}$ London $\overset{11}{\mapsto}$ Seville.
  Current length of the tour under construction: 29
- Closest city to either Rome or Seville: Paris (only choice).
  Tour: Paris $\overset{20}{\mapsto}$ Rome $\overset{8}{\mapsto}$ Berlin $\overset{7}{\mapsto}$ Moscow $\overset{3}{\mapsto}$ London $\overset{11}{\mapsto}$ Seville.
  Current length of the tour: 49.
- Link the two extreme cities. Length of the tour: 49 + 27 = 76.

Upper bound = $\overline{z}_3$ = 76. Improvement of the incumbent value.

Question 2. Graph problems. (20 points.)

Q2.1 Consider a graph $G = (V, E)$ that has both directed and undirected edges. There are no cycles in G that use only directed edges. Propose an algorithm to assign each undirected edge a direction so that the completely directed graph has no cycles. What is the complexity of your algorithm (you need to justify your answer).

Solution: 7 points.

Several of you proposed to enumerate cycles and then assign directions to the edges opposite to the directed links.
Comment 1: Tarjan's algorithm does not enumerate all cycles.
Comment 2: A cycle may contain more than one edge, and in such a case, not always true that the right assignment is opposite directions for all edges.
Comment 3: You need to provide explanations on how you define and search for "cycles" in a mixed graph.

Taking care of the (directed) links. First topologically sort the graph using only the directed links. Create an array so that for each vertex you store its number in the sorted list.

Defining an orientation of the undirected links (edges). Then, for each undirected edge, draw the edge in the direction that goes from the vertex with the lower sorted number to the higher sort number.

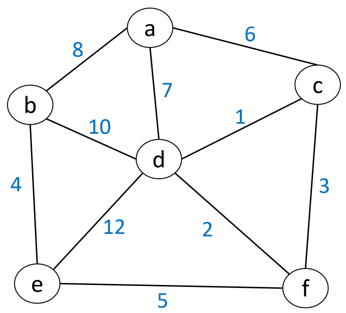
Q2.2 Given a directed graph G, you would like to get from s to t stopping at u if not too inconvenient where "too inconvenient" means the shortest path that stops at u is more than 10% longer than the shortest path from s to t. Give an algorithm for returning the shortest path from s to t that stops at u if convenient. What is the complexity of your algorithm (you need to justify your answer).

Solution: 7 points. - Writing the details of Dijkstra's algorithm: 3 points Run Dijkstra's algorithm once from s and once from u. The shortest path from s to t is found in doing Dijkstra's algorithm from s. The shortest path from s to t through u is the shortest path from s to u plus the shortest path from u to t.

3 points if you provide the description of Dijkstra's algorithm and its complexity.

1 point for the complexity.

Q2.3 Consider the following graph



Q2.3a Provide the order in which Kruskal's algorithm adds the edges to the MST.

3 points. Order of Kruskal's algorithm: $\{c, d\}, \{d, f\}, \{b, e\}, \{e, f\}, \{a, c\}$.

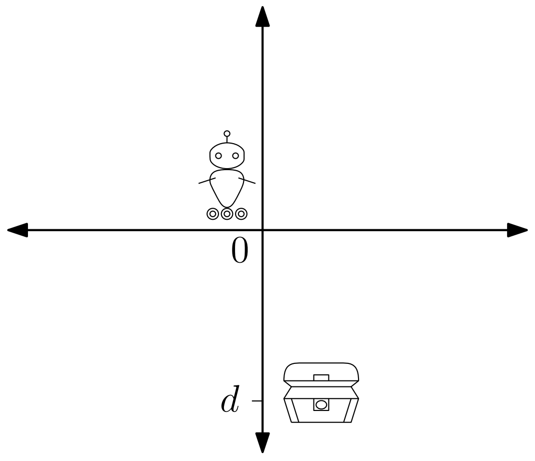
Q2.3b Provide the order in which Prim's algorithm (starting at vertex d) adds the edges to the MST.

3 points. Order of Prim's algorithm: $\{c, d\}, \{d, f\}, \{e, f\}, \{b, e\}, \{a, c\}$.

Question 3. **Online algorithms.** (20 points)

In the standard Line Search problem a robot moves along a line to find a treasure. Consider the following modification of this problem. The robot starts out at the crossroads connecting 4 roads. The treasure is located at unknown distance d along one of the roads. See the figure below.

Consider the following algorithm for finding the treasure. The algorithm works in phases labelled by $i = 0, 1, 2, \ldots$ In phase i the robot walks along each of the 4 roads in some predetermined order to distance 2^i and returns back to the crossroads if the treasure is not found. In this question, your goal is to compute the competitive ratio of this algorithm.



Q3.1 What is the worst case total distance travelled by the robot when $d = 5$? Show your work.

5 points.

Solution: The treasure is not found during phases $0, 1, 2,$ meanwhile the robot travels the total distance $8(1 + 2 + 4) = 56$ during those phases. 2 points

During the third phase, the robot travels distance $6 \cdot 8 = 48$ while checking the first three roads. 2 points

The treasure is found while checking the last road during the third phase, incurring additional 5 units of distance. The overall distance travelled is $56 + 48 + 5 = 109$.

$$\underbrace{8(1+2+4)}_{56} + \underbrace{6 \cdot 8}_{48} + 5$$

1 point

Q3.2 What is the worst case total distance travelled by the robot when $d = 0.2$? Show your work.

5 points.

Solution: The treasure will be found while checking the last road during phase 0. 2 points

The robot travels the total distance of 6 while checking the first three roads, 2 points

and additional 0.2 while checking the last road. The overall distance travelled by the robot is 6.2 1 point
.

Q3.3 Assume that $d = 2^k + \epsilon$ for some small $\epsilon > 0$ and $k \in \mathbb{N}$. What is the worst case total distance travelled by the robot in this case? What is the competitive ratio achieved by the algorithm?

5 points.

Solution: The treasure will be found during phase $k+1$ and while examining the last road. The first k phases incur the total cost of $8\sum_{j=0}^{k} 2^j = 8(2^{k+1}-1)$. During phase $k+1$, the robot travels distance $6 \cdot 2^{k+1} + d$. 2 points
The overall distance travelled by the robot is 2 points for the upper bounding calculation

$$8(2^{k+1}-1) + 6 \cdot 2^{k+1} + d \leq 16 \cdot 2^k + 12 \cdot 2^k + d \leq (16+12+1)d.$$

Therefore the competitive ratio is 29. 1 point

Q3.4 Instead of 4 roads meeting at the crossroads, now there are m roads meeting at the crossroads. Now, in phase i the robot walks along each of the m roads in some predetermined order to distance 2^i and returns back to the crossroads if the treasure is not found. Use similar analysis to part (b) to compute the competitive ratio in this case. Your competitive ratio should be expressed as a function of m. Assume $d \geq 1$.

5 points.

Solution: The distance travelled in phase i becomes $(2m)2^i$ if the robot doesn't find the treasure, and $(2m-2)2^i + d$ if it finds the treasure. 2 points
Plugging this into the previous calculation for $d = 2^k + \varepsilon$ we obtain that the total distance travelled by the robot is 2 points for the upper bound calculation

$$(2m)(2^{k+1}-1) + (2m-2)2^{k+1} + d \leq (4m)2^k + (4m-4)2^k + d \leq (8m-3)d.$$

Therefore, the competitive ratio is $8m-3$ 1 point
.

Question 4. (20 points)

Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis or accounting method to determine the amortized cost per operation.

Aggregate method: summing up the totals and dividing by n. In this case, the powers of 2 give at most

$$1 + 2 + 4 + 8 + ... + n = \sum_{k=0}^{\log_2 n} 2^k = 2^{\log_2 n + 1} - 1 < 2n,$$

and the non-powers-of-2 give $n - \log(n) < n$, so the total is $< 3n$, or amortized 3 per operation.

Accounting method: if every operation since the last power of 2 pays \$2 extra, that produces enough money to pay for the expensive operation. So, in this accounting, the cheap operations pay \$3, and the expensive operations pay \$2, for an amortized cost $<$ \$3.

Question 5. In relation with the project. (25 points)

Consider an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, and assume you would like to find an orientation for that graph.

Question 5.1 A strong orientation is an orientation that results in a strongly connected graph G. Robbins' theorem states that a graph has a strong orientation if and only if it is 2-edge-connected, i.e., if the graph has no bridge assuming the graph is connected. Provide an algorithm that checks that the graph is connected and that it contains no bridge. Provide the complexity of your algorithm: you need to provide a clear justification of the complexity in order for your complexity answer to be considered.

10 points: 2 points for DFS, 6 points for detecting bridges, 2 points for complexity

Comment 1. It is not enough to cite BFS for detecting bridges, you need to explain how it has to be modified with the identification of the back edges.

Comment 2. Do not confuse connected components (relate to undirected graph) and strongly connected components (relate to directed graph). In particular, you cannot compute the strongly connected components and look for the (directed) links connecting the strongly connected components as we deal with an undirected graph.

General Comment. Tarjan has written many algorithms, so not enough to say that you will use Tarjan's algorithm: you need to describe it. Similar comment for DFS: it can be used for many purposes, sometimes as is, sometimes with some minor modifications. You need to clarify/explain the input/output as well as how you plan to use it.

Checking connectivity: Use DFS and make sure you can reach all nodes.

Detecting bridges. Consider DFS. While we are looking for vertices adjacent to vertex v, then $v, u)$ will be a bridge if and only if none of the vertices u or any of its descendants in DFS traversal tree has a back edge to vertex v or any of its ancestors.

We can check the existence of back edges in $O(V + E)$ time. See, e.g., https://cp-algorithms.com/graph/bridge-searching.html

So, let $\text{TIN}(v)$ denote entry time for node v. We introduce an array LOW which will let us check the fact for each vertex v. $\text{LOW}(v)$ is the minimum of $\text{TIN}(v)$, the entry times $\text{TIN}(p)$ for each node p that is connected to node v via a back-edge (v, p) and the values of $\text{LOW}(w)$ for each vertex w which is a direct descendant of v in the DFS tree:

$$\text{LOW}(v) = \min \begin{cases} \text{TIN}(v) & \\ \text{TIN}(p) & \text{for all } p \text{ such that } (v,p) \text{ is a back edge} \\ \text{TIN}(v) & \text{for all } w \text{ such that } (v,w) \text{ is a tree edge} \end{cases} \tag{1}$$

Now, there is a back edge from vertex v or one of its descendants to one of its ancestors if and only if vertex v has a child w for which $\text{LOW}(w) \leq \text{TIN}(v)$. If $\text{LOW}(w) = \text{TIN}(v)$, the back edge comes directly to v, otherwise it comes to one of the ancestors of v.

Thus, the current edge (v, w) in the DFS tree is a bridge if and only if $\text{LOW}(w) > \text{TIN}(v)$.

Now, there is a back edge from vertex or one of its descendants to one of its ancestors if and only if vertex has a child for which . If , the back edge comes directly to , otherwise it comes to one of the ancestors of .

Thus, the current edge in the DFS tree is a bridge if and only if .

You had to provide the definition of a back edge + how to detect a bridge in detail

Question 5.2 What is the complexity result that is proved in the paper of Omran *et al.* (2013), entitled *Finding Paths with Minimum Shared Edges*? Does the complexity result apply for directed or undirected graphs?

5 points: 2 points for the problem/question statement, 2 points for complexity, 2 points for answering directed graph
Answer. Paper studies the so-called MSE problem:

Minimum Shared Edges (MSE) Problem:. Given a **directed** graph $G = (V, E)$, two special nodes $s, t \in V$, and an integer $k > 0$, find a set P of k paths from s to t in G so as to minimize $c(P) = \sum_{e \in E} \lambda(e)$, where $\lambda(e) = 0$ if e is used in at most one path of P, and $\lambda(e) = 1$ otherwise.

An edge e with $\lambda(e) = 1$ is called a shared edge.

First main complexity result of the paper:

The MSE problem is NP-hard.

Comment: there are several complexity results in the paper. The above one was the main one. Answer to the question was considered only if the complexity was provided with the problem statement. Answers with other results of the paper with the problem/question statement were accepted.

Question 5.3 Recall the definition of the one-way street problem

5 points
Answer. The one-way street problem is a **graph orientation problem**, meaning that you need to define the orientation of edges (undirected links) subject to some conditions.

Definition of the one-way street problem: Consider an undirected graph. Is it possible to choose a direction for each edge, turning it into a directed graph that has a path from every vertex to every other vertex.

Question 5.4 One possible difference between the one-street problem and the problem of defining an orientation of corridors lies in that we do not need a path between every node pair in the orientation of the corridors. Does this change the condition under which we can find a strong orientation of an undirected graph?

5 points
Answer. Yes, it changes the condition. Indeed, even if there is a bridge but no requested path between nodes on each side of the bridge, then problem may have a solution.

Robbins, H. E., A theorem on graphs, with an application to a problem on traffic control, *American Mathematical Monthly*, 1939, 46 (5): 281–283,