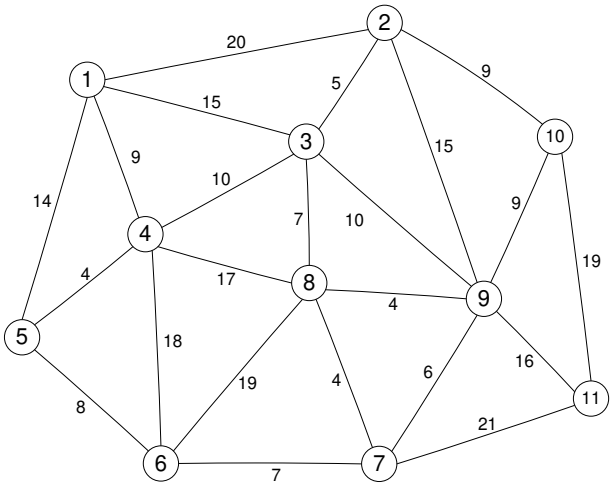
2.1 Minimum-cost spanning tree

Find the minimum-cost spanning tree in the graph given in the figure by using Prim's algorithm, starting from the node 3.



2.2 Kruskal's algorithm

In 1956 Joseph Kruskal proposed the following greedy algorithm to find a minimum-cost spanning tree in an arbitrary connected undirected graph $G = (N, E)$ with a cost c_e attached to each edge $e \in E$.

1) Sort the edges of E as $\{e_1, \ldots, e_m\}$ where $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m}$
2) Let $i = 1$ and initialize the subgraph $G' = (N, F)$ of G with $F = \emptyset$ (G' consists of n connected components[1] corresponding to the isolated nodes)
3) WHILE $|F| < n - 1$ DO

   IF the two endpoints of the edge e_i belong to different connected components of the current subgraph G' THEN $F := F \cup \{e_i\}$ and merge the two connected components

   $i := i + 1$

   END
4) Return the spanning tree $G' = (N, F)$

In other words, we order the edges by increasing (non-decreasing) cost, we consider the edges in that order and, at each step, we select the current edge (which is one of the cheapest edges still available) only if it does not create a cycle with the previously selected edges. The algorithm terminates when $n - 1$ edges have been selected.

[1] A connected component of an undirected graph is a subgraph in which any two nodes are connected, and which is connected to no other nodes.

a) Describe an efficient way to identify/keep track of the connected components of the subgraph G' and to check that a new edge is creating a cycle with the previously selected edges (is connecting two distinct connected components of G').

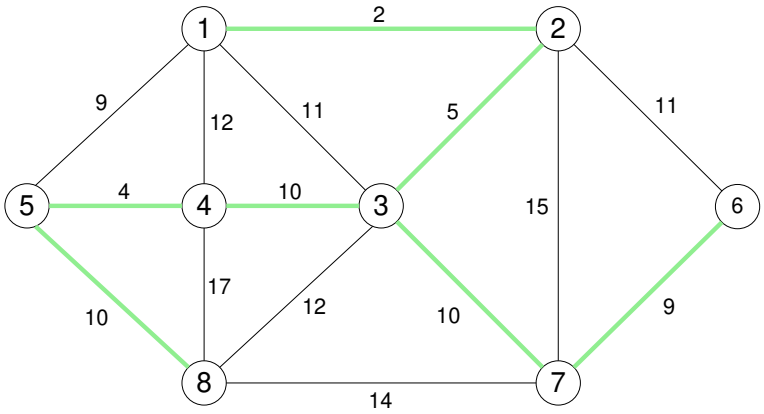
b) Determine the overall computational complexity of this simple implementation of Kruskal's algorithm.

c) By invoking the optimality condition for minimum-cost spanning trees, verify that Kruskal's algorithm is exact, i.e., is guaranteed to provide an optimal solution for any undirected graph with costs on the edges.

d) Find the maximum-cost spanning tree in the graph of the previous exercise by using a straightforward adaptation of Kruskal's algorithm.

2.3 Optimality check

Without applying any one of Prim's and Kruskal's algorithms, verify whether the following spanning tree is of minimum cost.



2.4 Compact storage of similar sequences

Consider the problem of storing a large set of strings, i.e., sequences of characters from a finite alphabet. We assume that the strings have many similar entries (they differ only in a small number of positions) and we wish to store them in a compact way. This problem arises in several contexts such as when storing DNA sequences, where the characters correspond to the four DNA bases. In this exercise, we consider the simplified version of the problem with only two characters.

Given a set of k sequences of M bits, we compute for each pair i, j, with $1 \leq i, j \leq k$, the Hamming distance between the sequences i and j, i.e., the number of bits that need to be flipped in sequence i to obtain sequence j. This function clearly satisfies the three usual properties of a distance: nonnegativity, symmetry and triangle inequality.

Consider the following set of 6 sequences and the corresponding matrix $D = \{d_{ij}\}$ of Hamming distances

1) `011100011101`
2) `101101011001`
3) `110100111001`
4) `101001111101`
5) `100100111101`
6) `010101011100`

	1	2	3	4	5	6
1	0	4	4	5	4	3
2		0	4	3	4	5
3			0	5	2	5
4				0	3	6
5					0	5
6						0

where, due to symmetry, only the upper triangle of the matrix is shown.

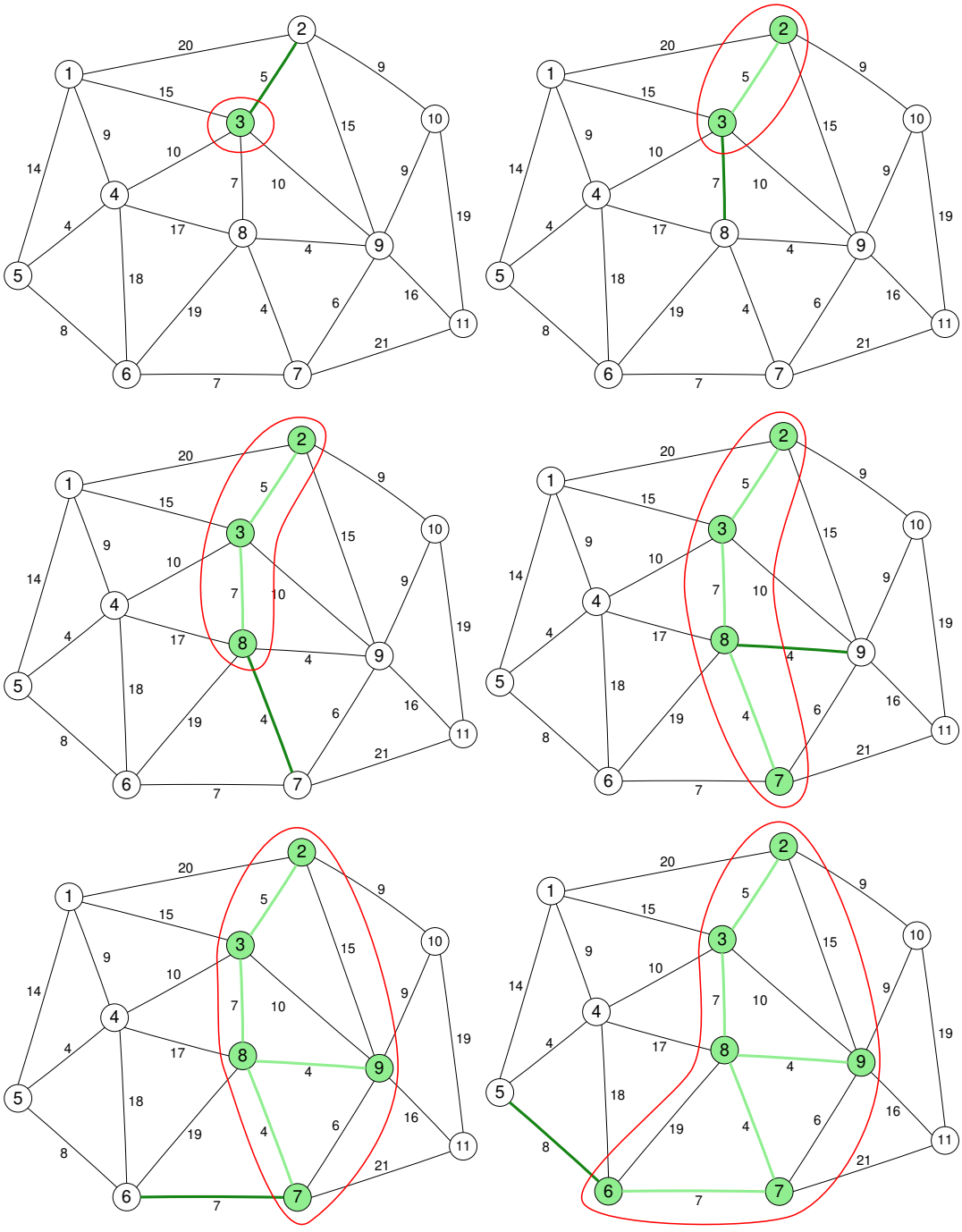
In order to exploit redundancies between sequences and to save memory, we can store: i) one of the sequences, called the reference sequence, completely and ii) for every other sequence, only the set of bit flips that allow us to retrieve it either directly from the reference sequence or from another sequence.

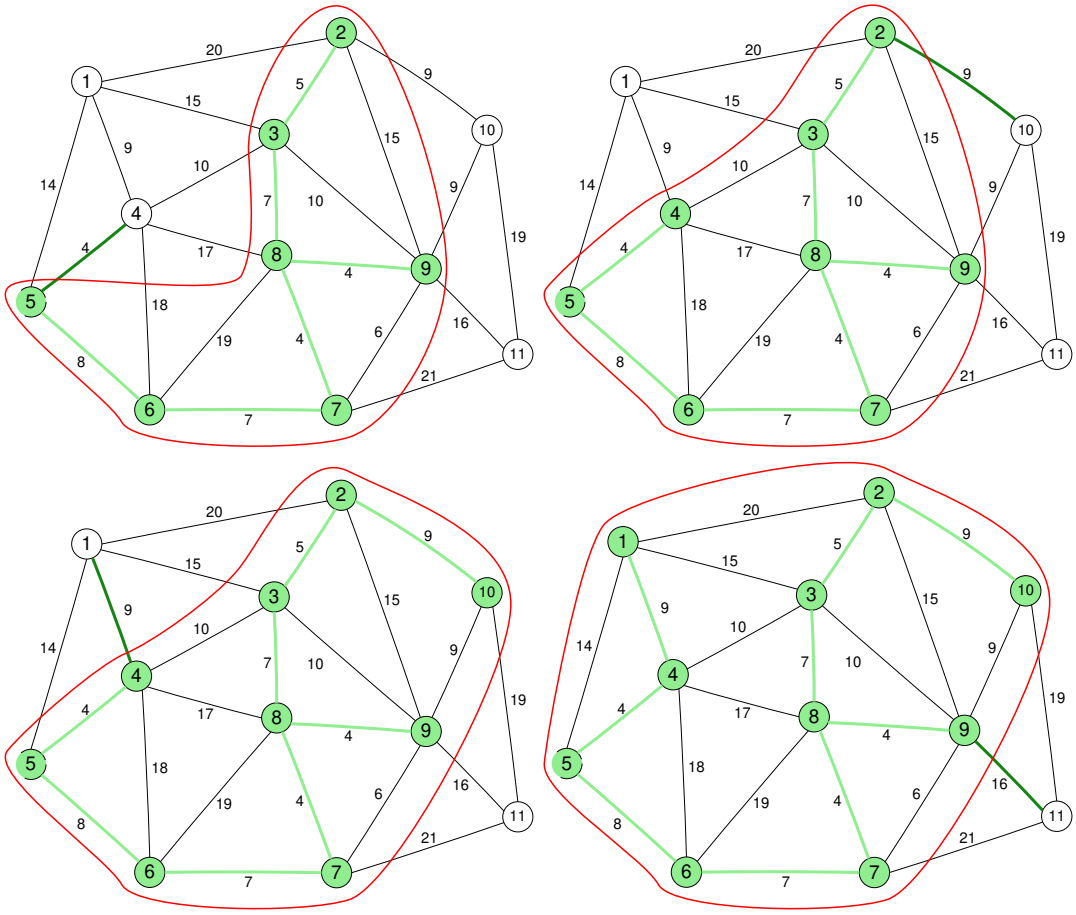
Show how the problem of deciding which differences to memorize, so as to minimize the total number of bits used for storage, can be reduced to the problem of finding a minimum-cost spanning tree in an appropriate graph. Solve the problem for the given instance.

[*Hint*: Given any two sequences (the i-th one and j-th one), how many bits are needed to store the positions in which they differ?]

SOLUTION

2.1 **Minimum-cost spanning tree**. We apply Prim's algorithm, starting at node 3. At each iteration, the set S of the nodes in which the edges selected so far are incident is highlighted in red. The partial spanning tree is highlighted in light green. Among all edges in the cut $\delta(S)$, induced by S, the edge, among those of minimum cost, that is going to be added to the partial spanning tree is highlighted in dark green.

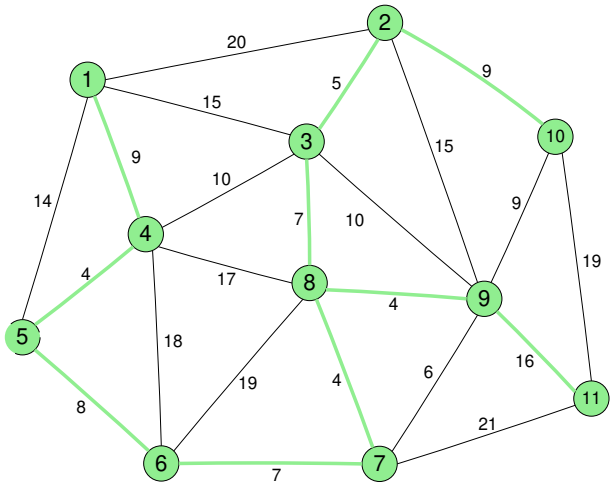




The edges are added in the following order

reached nodes	edge	cost	iteration
3	(2,3)	5	1
2,3	(3,8)	7	2
2,3,8	(7,8)	4	3
2,3,7,8	(8,9)	4	4
2,3,7,8,9	(6,7)	7	5
2,3,6,7,8,9	(5,6)	8	6
2,3,5,6,7,8,9	(4,5)	4	7
2,3,4,5,6,7,8,9	(2,10)	9	8
2,3,4,5,6,7,8,9,10	(1,4)	9	9
1,2,3,4,5,6,7,8,9,10	(9,11)	16	10=n-1 → HALT

Since $S = V$, i.e., every node has been reached, the algorithm halts. The minimum-cost spanning tree that has been found has total cost 73. It is shown in the following figure.



2.2 **Kruskal's algorithm**.

a) To identify/keep track of the connected components (subtrees) of the subgraph G', we use a vector v with as many components as vertices in the graph, where $v[i]$ indicates the index of the connected component containing node i. At the beginning of the algorithm, we start with $v[i] = i$ for $i = 1, \ldots, n$. When an edge $e = \{i, j\}$ is considered for addition to G', we compare the values $v[i]$ and $v[j]$. If $v[i] \neq v[j]$, then we can add the edge e to G' because it does not create a cycle. Since the two connected components of indices $v[i]$ and $v[j]$ are merged, the indices are updated as follows: in the vector v we substitute each occurrence of the index of node i with that of node j. If $v[i] = v[j]$, edge e is skipped because it would create a cycle.

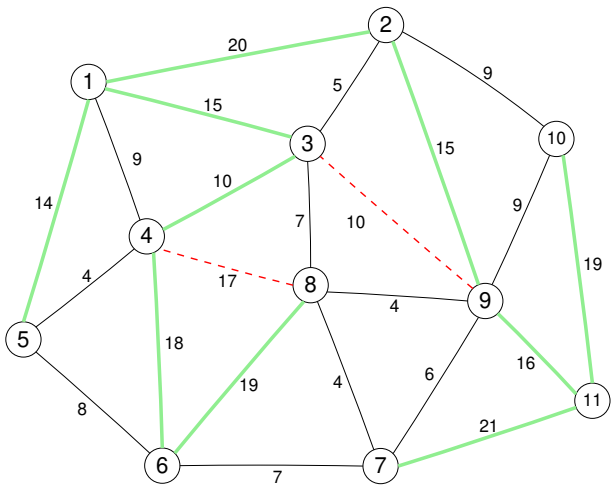
b) The m edges can be ordered by non-decreasing cost in $O(m \log m)$, which is $O(m \log n)$ since $m \log m \leq m \log n^2 = 2m \log n$. At most m edges are considered for addition to the current subgraph G'. At each iteration, an edge $e = \{i, j\}$ is considered and the vector v is updated (in $O(n)$) only if $v[i] \neq v[j]$. Since a merging operation occur exactly $n - 1$ times (a spanning tree contains $n - 1$ edges), the overall complexity is $O(m \log n + m + n^2) = O(m \log n + n^2)$.

d) To determine a maximum-cost spanning tree in the given undirected graph $G = (V, E)$, we sort all the edges of G by nonincreasing cost and consider them one by one in that order. Let e be the edge considered at the current iteration and F be the set of edges selected so far. If adding e to F creates a cycle, e is dropped. Otherwise, e is added to F, and a new iteration is performed. The algorithm terminates when $n - 1$ edges have been selected, namely, when $|F| = n - 1$.

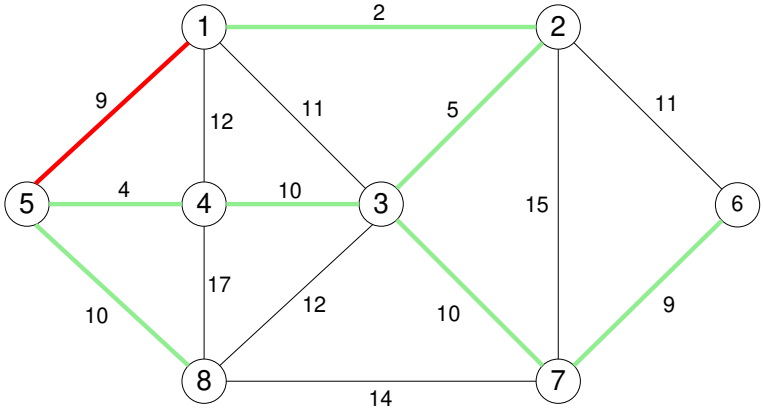
The edges are considered in the following order

connected components	edge	cost	iteration
	(7,11)	21	1
{7,11}	(1,2)	20	2
{1,2}, {7,11}	(6,8)	19	3
{1,2}, {7,11}, {6,8}	(10,11)	19	4
{1,2}, {7,10,11}, {6,8}	(4,6)	18	5
{1,2}, {7,10,11}, {4,6,8}	(4,8)	NO	(introduces a cycle)
{1,2}, {7,10,11}, {4,6,8}	(9,11)	16	6
{1,2}, {7,9,10,11}, {4,6,8}	(1,3)	15	7
{1,2,3}, {7,9,10,11}, {4,6,8}	(2,9)	15	8
{1,2,3,7,9,10,11}, {4,6,8}	(1,5)	14	9
{1,2,3,5,7,9,10,11}, {4,6,8}	(3,9)	NO	(introduces a cycle)
{1,2,3,4,5,6,7,8,9,10,11}	(3,4)	10	10=n-1 → HALT

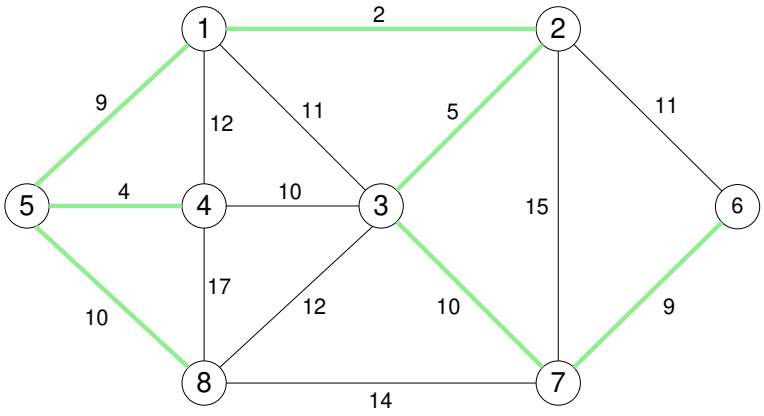
A spanning tree of maximum cost, of value 167, is shown in green in the following figure. The dropped edges are highlighted in red.



2.3 **Optimality proofs**. It suffices to verify that there exists a diminishing edge. By inspection, we observe that, by adding edge $(1, 5)$ to the tree, the cycle $(1, 5, 4, 3, 2, 1)$ is introduced. In such cycle, edge $(4, 3)$ has a strictly larger cost than $(1, 5)$.



Therefore, by removing edge $(4, 3)$ and adding edge $(1, 5)$, a spanning tree of strictly smaller total cost is obtained. It is shown in the following figure.



2.4 **Compact storage of similar sequences**. We construct a complete graph G with a node for each sequence and an edge for each pair of sequences. Moreover, each edge $[i, j]$ is assigned a cost d_{ij}.

The problem is then to look for a subgraph G' of G of minimum total cost. Since only one sequence is completely stored, G' must be connected to be able to construct any sequence. Since a subgraph of minimal cost is sought, G' will be acyclic. Therefore a minimum-cost spanning tree in G provides an optimal solution to the problem under consideration.

The following minimum cost spanning tree has been found using Prim's algorithm.