Practice 4

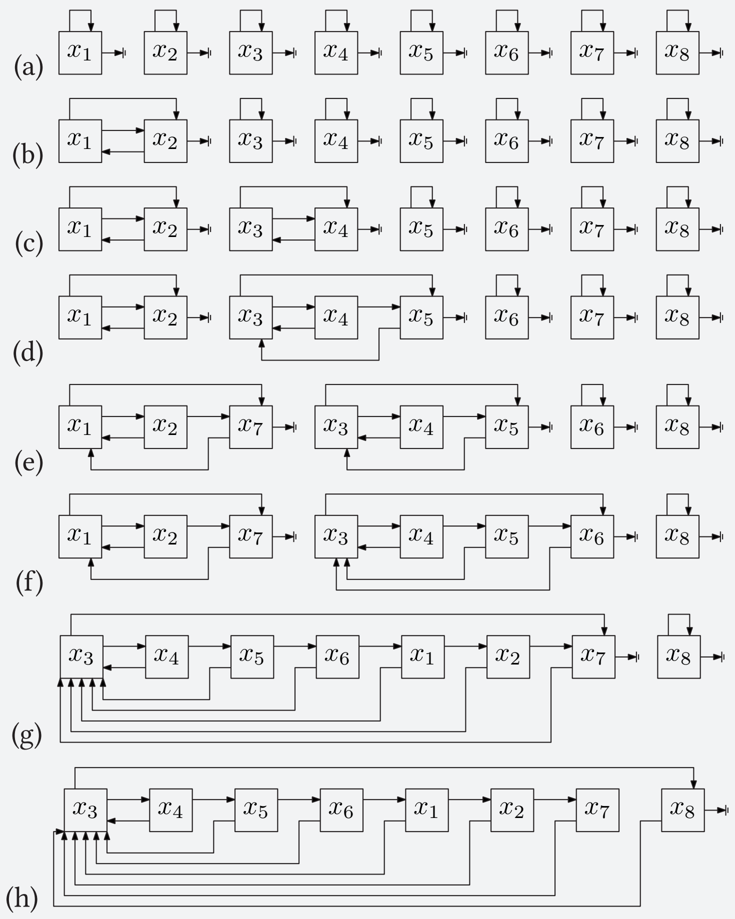
Exercise levels:
(L1) Reproduce: Reproduce basic facts or check basic understanding.
(L2) Apply: Follow step-by-step instructions.
(L3) Reason: Show insight using a combination of different concepts.
(L4) Create: Prove a non-trivial statement or create an algorithm or data structure of which the objective is formally stated.

▶ Lecture 10 Union-Find

Exercise 1 *(L2)* Consider the elements $\{x_1, x_2, \ldots, x_8\}$. We call MAKE-SET($x$) once for each element. Show the status of the data structure after performing each of the following instructions in sequence: UNION(x_1, x_2), UNION(x_3, x_4), UNION(x_3, x_5), UNION(x_1, x_7), UNION(x_3, x_6), UNION(x_1, x_3), UNION(x_1, x_8) ...

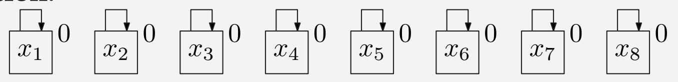
(a) ...when using Union-by-size.

Solution:

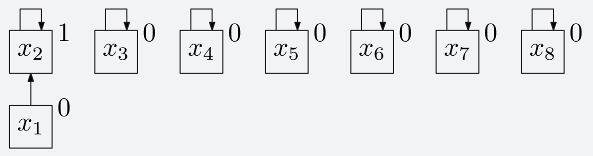


(b) ...when using Union-by-rank with path compression.

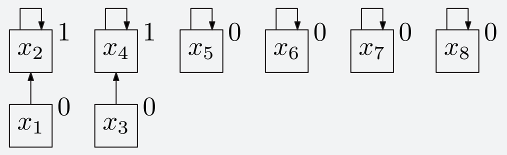
Solution:



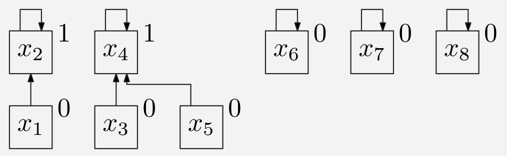
(a)



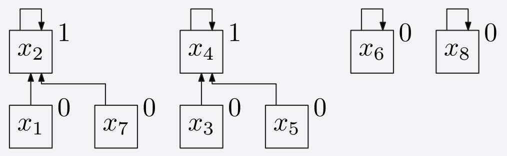
(b)



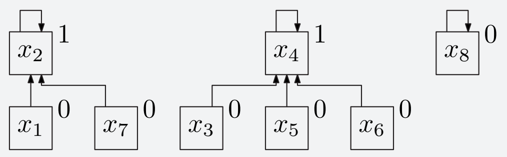
(c)



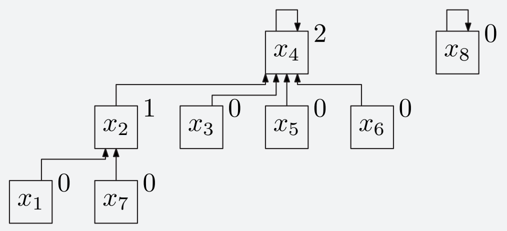
(d)



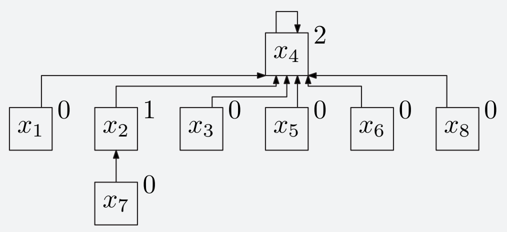
(e)



(f)



(g)



(h)

Exercise 2

(a) *(L1)* What is the maximum number of UNION operations performed during Union-Find?

Solution:
The sets can only merge and can never split. There are n sets at the start, and after every UNION the number of sets decreases by one. Thus there can be at most $n-1$ UNION operations.

(b) *(L3)* Consider union-by-size. Let $S = \{x_1, x_2, \dots, x_{n-1}, x_n\}$ be a set of $n = 2^k$ elements, for $k \in \mathbb{N}$. Describe a sequence of m operations on S (of which n are MAKE-SET) that achieves the worst-case runtime of $\Theta(m + n \log n)$.

Solution:
Our example repeatedly takes the union of two sets of the exact same size until there is only one set left.
We make a set for each of the different elements using MAKE-SET. Consider all the different single-elements sets as leaves of a tree. For every two sets of the same size that we take the union of let their parent node be the union of the sets. All together all union-operations form a binary tree where at the root there is a single set containing all elements. The sets at height h each contain 2^h elements. The binary tree has n leaves and thus has $2n-1$ nodes is total. As the tree is perfectly balanced it has height $\log(2n-1) = O(\log n)$.
Consider all UNION operations performed on sets at height h. There are $\frac{n}{2^h}$ nodes at height h and each represents a set with 2^h elements. For half of these sets we will need to update all rep pointers. Thus in total for all sets at height h we need to update $\frac{1}{2} \cdot \frac{n}{2^h} \cdot 2^h = \frac{n}{2}$. Thus in total there are $\sum_{h=0}^{\log(2n-1)} \frac{n}{2} = \Theta(\frac{n}{2} \log(2n-1)) = \Theta(n \log n)$.
The n MAKE-SET operations take $\Theta(n)$ time in total.
The remaining $m-2n$ FIND-SET operations each take $\Theta(1)$ time and thus in total $\Theta(m-2n)$ time.
Thus this example achieve the worst-case runtime of $O(m + n \log n)$. We conclude that the worst case is $\Theta(m + n \log n)$.

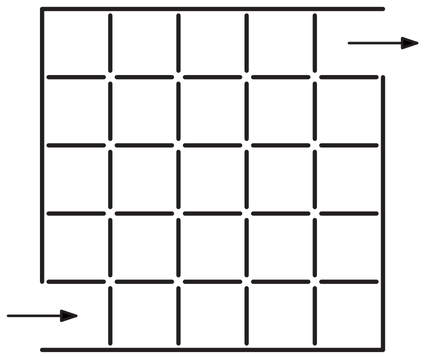
(c) *(L3)* Consider union-by-rank with path compression. What is the maximum height of a node with rank k? What is the minimum height?

Solution:
The maximum height is k. If we never compress any path then the rank is equal to the height. The minimum height is 0. If all paths are compressed then we have a root element with all other elements as children. In this way a node with rank k can be a leaf of the tree.

(d) *(L3)* Explain, in your own words, why there are at most $\log^* n$ rank groups when using union-by-rank, when $n \geq 1$.

Solution:
Similar to union-by-size, every time the rank is increased by one, the size of the set at least doubles. This can happen at most $\log n$ times and hence the maximum rank is at most $\log n$. But then the element that is in the highest rank group can have rank at most $\log n$. By definition this element is in rank group $\log^*(\log n)$. The number of times we can take the logarithm of $\log n$ before reaching a number smaller than 2 is exactly one less than the number of times for n. Hence $\log^*(\log n) = \log^*(n) - 1$ is the highest rank group number. We have 1 more rank group as 0 is also a rank group, so in total there are $\log^*(n)$ rank groups.

Exercise 3 Let G be a grid of k cells by k cells. Between every two horizontally or vertically adjacent cells there is a wall separating the cells. On the outside there is a wall encompassing the full grid that has two openings (see figure).



We are going to build a maze using the following algorithm:

```
MAZE($G$)
1  for each cell $c$
2      MAKE-SET($c$)
3  for all internal walls $w$ in an arbitrary order
4      let $c$ and $d$ be the cells on both sides of $w$
5      if FIND-SET($c$) $\neq$ FIND-SET($d$)
6          remove wall $w$
7          UNION($c, d$)
```

(a) *(L3)* What does it mean when during execution two cells are in the same set?

Solution:
When two cells are in the same set there is a path between the two cells.

(b) *(L2)* What is the running time of the algorithm in terms of k when using Union-Find and the implementation of solution 1 from the lectures?

Solution:
There are $2 \cdot k \cdot (k-1) = \Theta(k^2)$ walls and $k \cdot k = \Theta(k^2)$ cells. Make-Set: Thus we perform $\Theta(k^2)$ Make-Set operations. Union: Once all cells are merged no more Union operations are performed, thus there are $k^2 - 1$ Union operations. Find-Set: Each Union operation gives two Find-Set operations, next to this for each for each of the $\Theta(k^2)$ walls we perform two more Find-Set operations. Thus there are also a total of $\Theta(k^2)$ Find-Set operations.
In total there are $m = \Theta(k^2)$ operations and there are $n = k^2$ elements.
The runtime for solution 1 is $O(mn) = O(k^2 * k^2) = O(k^4)$

(c) *(L2)* What is the running time of the algorithm in terms of k when using union-by-rank with path compression?

Solution:
Once again $m = \Theta(k^2)$ and $n = k^2$. We fill this in the runtime to get $O(k^2\alpha(k^2))$.

▶ Lecture 11 Elementary graph algorithms

Exercise 4 Consider the following two adjacency matrices:

$$(1)\ \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 1 \\ 3 & 0 & 1 & 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 & 1 & 0 \\ 5 & 1 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array} \qquad (2)\ \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 1 & 1 & 0 \\ 5 & 1 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

For each, answer the following questions:

(a) *(L2)* Can this matrix be the adjacency matrix of an undirected graph? Could it represent a directed graph? Why or why not?

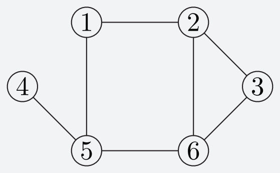
Solution:
For a matrix to be an adjacency matrix of an undirected graph, it has to be:

- symmetric, i.e., $a_{ij} = a_{ji}$ for any i and j (if there is an edge from i to j then there is an edge from j to i).
- its diagonal elements have to be 0 (there are no self-loops).

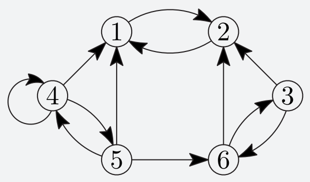
Any square matrix with 0s and 1s can be an adjacency matrix of a directed graph.
So, matrix (1) can be an adjacency matrix of both, an undirected and a directed graph, and matrix (2) is an adjacency matrix of a directed graph.

(b) *(L2)* For both adjacency matrices draw the corresponding undirected graph if possible, otherwise draw the corresponding directed graph.

Solution:



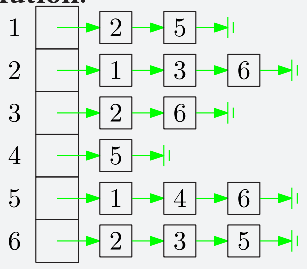
(1)



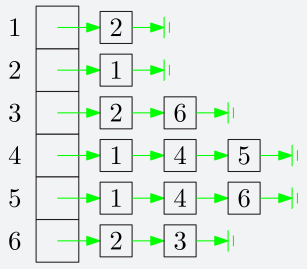
(2)

(c) *(L2)* Give the adjacency list representation for both graphs.

Solution:



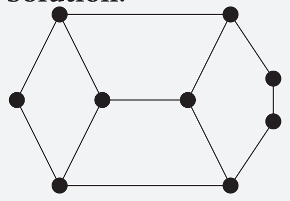
(1)



(2)

Exercise 5 *(L3)* Draw an undirected graph that has 12 edges and at least 6 vertices. Six of the vertices have to have degree exactly 3, all other vertices have to have degree at most 2. Use as few vertices as possible.

Solution:



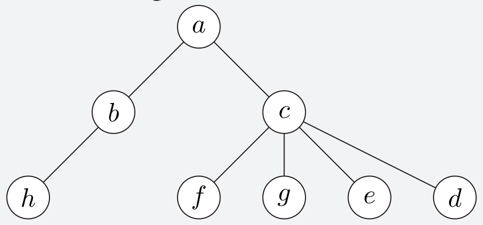
Such a graph can be constructed as follows. Start with six vertices, arrange them into two triangles, and connect the corners of the triangles pairwise. Now we have six vertices of degree three, what is left is to bring the number of edges up to twelve. We can do it by subdividing edges three times. We obtain a graph with nine vertices in total.
We will show next that nine is the minimum number of vertices needed to construct a graph satisfying the requirements....

We know that the sum of degrees is exactly two times the number of edges. There must be 12 edges, thus the sum of degrees must be 24. Exactly $6 \cdot 3 = 18$ is contributed by the vertices of degree 3. The remaining $24 - 18 = 6$ must be caused by the remaining vertices. As every remaining vertex has degree at most 2 there must be at least another $6/2 = 3$ vertices. Thus the minimum number of vertices needed is $6 + 3 = 9$.

Exercise 6 *(L3)* Let G be an undirected graph on the vertices $\{a, b, c, d, e, f, g, h\}$. We do not know which edges are in G but we know that Breadth First Search (BFS), when starting at node a, visits the following edges of the graph in the given order: (a, b), (a, c), (b, h), (c, f), (c, g), (c, e), (c, d). Can the following edges be in the graph G?

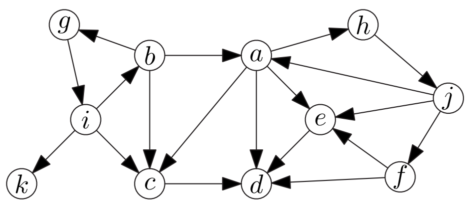
1: (b, d), 2: (d, e), 3: (d, h), 4: (a, f)

Solution:
The tree we get:



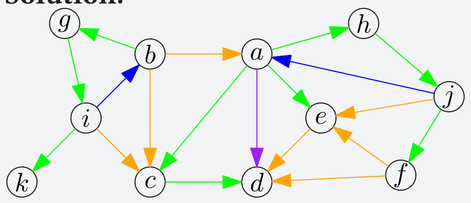
1. Edge (b, d) cannot be in G. If it were, it would be explored from the vertex b, and not from c, as vertex b gets explored before c.
2. Edge (d, e) can be in G, as d and e have the same depth.
3. Edge (d, h) can be in G, as d and h have the same depth.
4. Edge (a, f) cannot be in G. If it were, it would be explored from the vertex a, and not from c, as vertex a gets explored before c.

Exercise 7 *(L2)* Consider the following directed graph:



Perform a Depth-First-Search (DFS) from node a. Whenever there are multiple candidate nodes that could be visited next, choose the one with the smallest label in lexicographic order. Determine all tree edges, back edges, forward edges, and cross edges.

Solution:



Green edges are the tree edges, back edges are blue, forward edges are purple, and cross edges are orange.

Exercise 8 Consider the following algorithm on an unrooted tree T. (An unrooted tree has no designated root node.) Iteratively remove all the nodes with degree at most 1 until the tree is empty.

(a) *(L3)* How many nodes could have been removed in the last step? Justify your answer.

> **Solution:**
> There are either one or two nodes removed in the last step of the algorithm.
> First, observe that during the execution of the algorithm we always have one connected component, i.e., the nodes that are left form a single tree. If the number of nodes in the tree is greater than 0, then all the nodes that are removed are leaves. Removing a leaf does not make the tree disconnected as the node can only be an end-node for any path in the tree. Then, in the last step, at most two nodes are removed, as a tree with only leaves and with no interior nodes can consist of a single vertex, or two vertices connected by an edge.

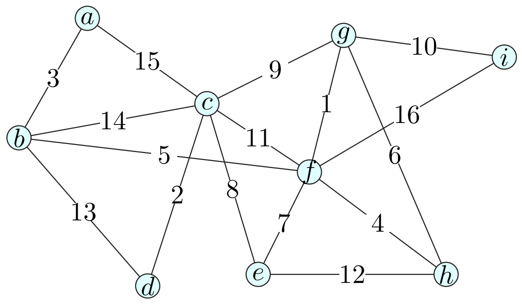
(b) *(L4)* The nodes that are removed in the last step are called the *center nodes* of the tree. Assume there is unique center point c. Prove that for this node the longest distance to any other node in the tree is minimized. That is, for any non-center node a, it holds $\max_v d(a, v) > \max_w d(c, w)$, where $d(x, y)$ is the distance between the nodes x and y.

> **Solution:**
> Consider the tree T rooted at node c, and consider the height $height(v)$ of any node v in T. The algorithm removes a node v from the tree at step $height(v) + 1$: all leaves are of height 0, and they get removed at step 1; all the nodes of height 1 get removed at step 2, etc. If the height of the root c is h, then c must have at least two children with height $h-1$. Otherwise c would be removed from T at an earlier step. Let n be an arbitrary node in T that is not c and let c_1 be child of c that contains n in the subtree rooted at c_1. Let c_2 be an child of c that is not c_1 and has $height(c_2) = h - 1$. As c_2 has height $h - 1$ there is a downwards path of length $h - 1$ to a leaf l in the subtree rooted at c_2. As the path from n to l must pass through c we have $d(n, l) > d(c, l) = h$. But then $\max_v d(n, v) \geq h + 1$. As this holds for any arbitrary node $n \neq c$, we have that for any non-center node a it holds that $\max_v d(a, v) > \max_w d(c, w)$, where $d(x, y)$ is the distance between the nodes x and y.

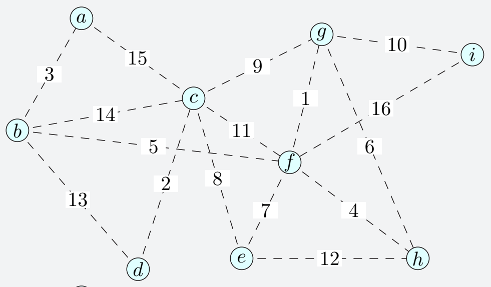
▶ Lecture 12 Minimum spanning trees

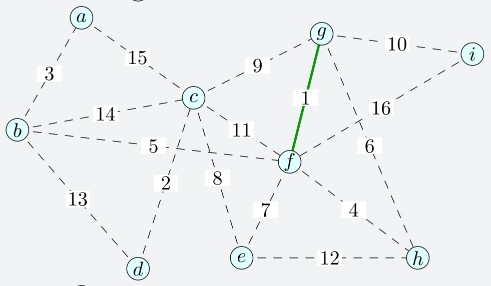
Exercise 9

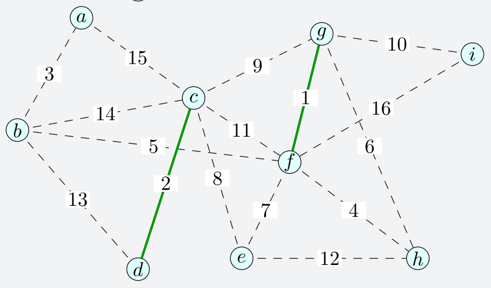
(a) *(L2)* Draw the selected edges after each step of Kruskal's algorithm for the following graph. Do the same for Prim's algorithm when starting from node *a*.

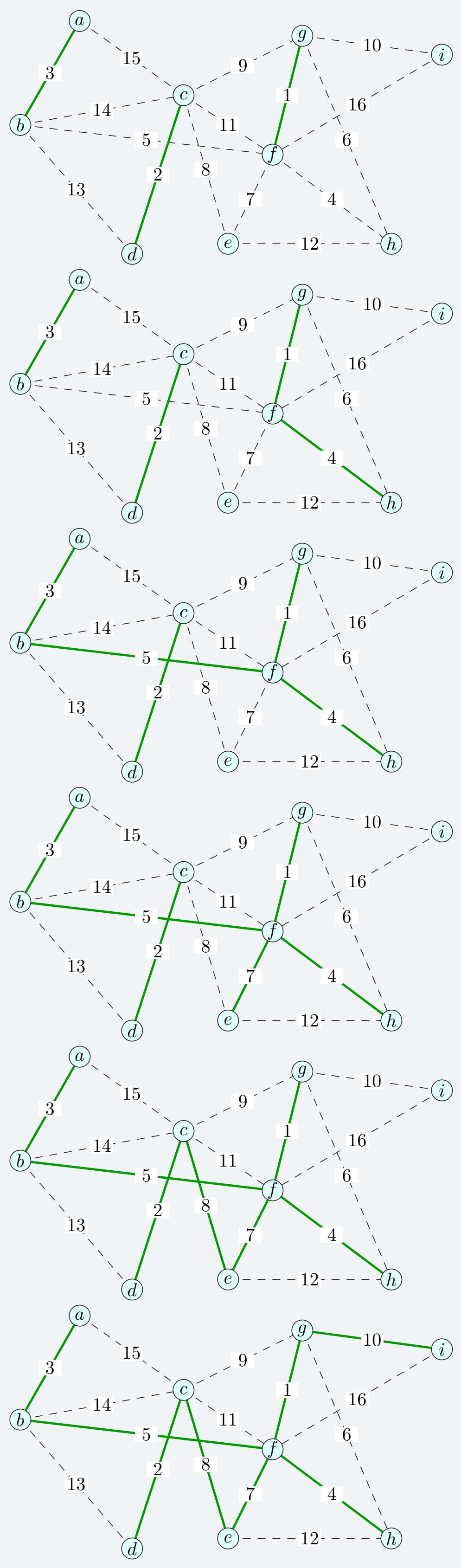


Solution:
Kruskal's:

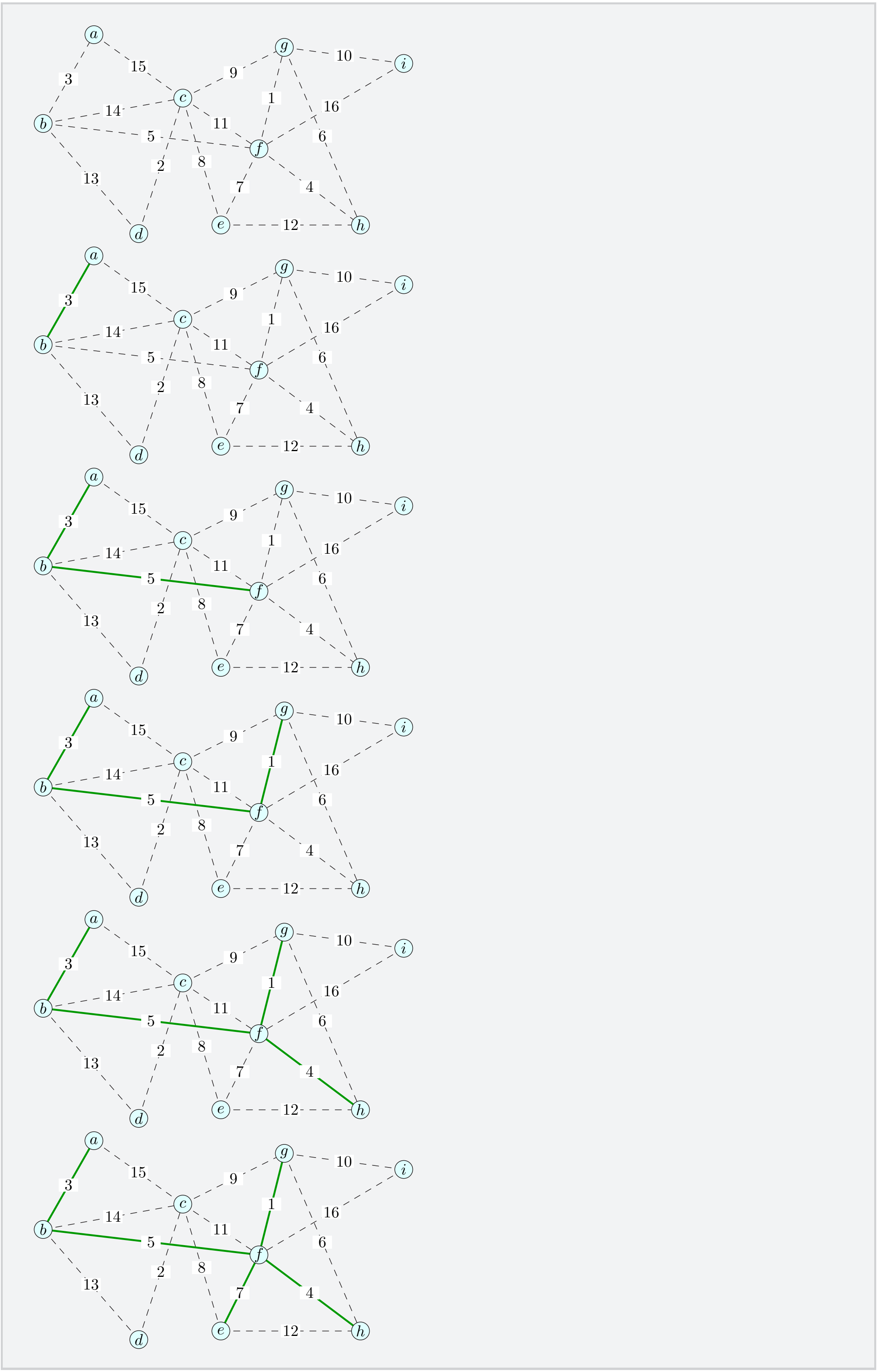


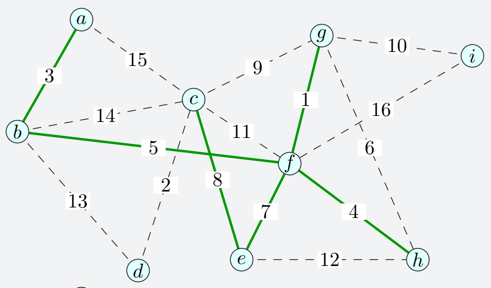


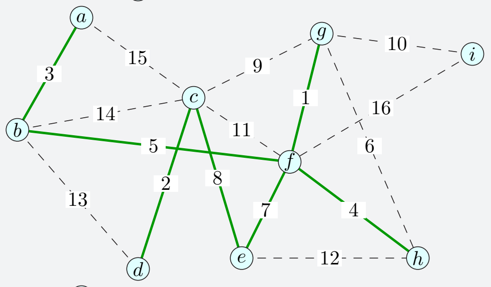


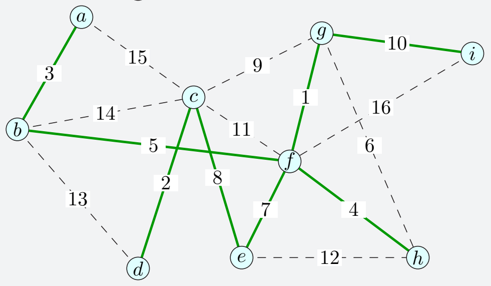


Prim’s:


a
b
c
d
e
f
g
h
i
3
15
14
5
13
2
9
11
8
1
10
16
6
7
4
12







(b) *(L2)* For each of the edges (a, b), (c, e), (b, f), give a cut of G where the edge is a light edge.

Solution:

- The edge (a, b) is light for the cut $(\{a\}, \{b, c, d, e, f, g, h, i\})$;
- The edge (c, e) is light for the cut $(\{c, d\}, \{a, b, e, f, g, h, i\})$;
- The edge (b, f) is light for the cut $(\{a, b\}, \{c, d, e, f, g, h, i\})$.

We will discuss the solution on the example of edge (b, f).
First, we check that (b, f) is indeed a light edge of the cut $(\{a, b\}, \{c, d, e, f, g, h, i\})$. The edges that cross this cut are (b, f) (weight 5), (a, c) (weight 15), (b, c) (weight 14), and (b, d) (weight 13). Out of these four edges (b, f) has the minimal weight.
One way of finding such a cut would be to consider the cut defined by the two subtrees we obtain if we delete the edge (b, f) from the MST. If (b, f) were not a light edge of this cut, there would be another edge e' with lower weight crossing the cut. Similarly as in the lecture, we could remove (b, f) from the MST and replace it with e'. The resulting graph would still be a tree spanning all vertices, but have lower weight than our original MST, which is impossible. So (b, f) must be a light edge.
This same argument can be made for any MST edge, so we get that the edge (a, b) is light for the cut $(\{a\}, \{b, c, d, e, f, g, h, i\})$, and that the edge (c, e) is light for the cut $(\{c, d\}, \{a, b, e, f, g, h, i\})$.

Exercise 10 *(L3)* Let T be a minimum spanning tree of a weighted graph $G = (V, E)$ with distinct positive edge weights. We make a new graph G' that is equal to G but where the weight for each edge $e \in E$ is defined as:

(a) $w_{new}(e) = 2 \cdot w(e)$

Solution:
Yes.
Assume for contradiction that T is not a minimum spanning tree in G'. Let T' be a minimum spanning tree of the new graph G'. Then $w_{new}(T') < w_{new}(T)$. Moreover as every edge in T has doubled in weight, we also know that $w_{new}(T_x) = 2 \cdot w(T_x)$. We have that $2 \cdot w(T') = w_{new}(T') < w_{new}(T) = 2 \cdot w(T)$. But then, as $w(T') < w(T)$, T cannot have been a minimum spanning tree in G. Contradiction, as T is a minimum spanning tree in G.
We conclude that our assumption was false and T is also a minimum spanning tree in G'.

(b) $w_{new}(e) = w(e) + 1$

Solution:
Yes.
Assume for contradiction that T is not a minimum spanning tree in G'. Let T' be a minimum spanning tree of the new graph G'. Then $w_{new}(T') < w_{new}(T)$. Moreover as every edge in T has increase by one in weight and every spanning tree has $|V| - 1$ edges, we also know that $w_{new}(T_x) = w(T_x) + (|V| - 1)$. But then in G the weight of $w(T') = w_{new}(T') - (|V| - 1) < w_{new}(T) - (|V| - 1) = w(T) + (|V| - 1) - (|V| - 1) = w(T)$. But as $w(T') < w(T)$, then T cannot have been a minimum spanning tree in G. Contradiction, as T is a minimum spanning tree in G.
We conclude that our assumption was false and T is also a minimum spanning tree in G'.

(c) $w_{new}(e) = 1/w(e)$

Solution:
No.
For example, consider graph $G = (V, E)$ with $V = \{a, b, c\}$ and $E = \{(a, b), (b, c), (a, c)\}$. That is, G is a cycle of size three. Set the weights of the edges to be $w(a, b) = 1$, $w(b, c) = 2$, and $w(a, c) = 3$. Then the minimum spanning tree consists of edges (a, b) and (b, c). After the weights change, we get $w(a, b) = 1$, $w(b, c) = 1/2$, and $w(a, c) = 1/3$, and the new minimum spanning tree will consist of edges (a, c) and (b, c). Thus, we have constructed an example where the minimum spanning tree changes if the weights of all edges get inverted.

Is T also a minimum spanning tree for G'? Prove T is a minimum spanning tree or give a counterexample to disprove it.
Note: to prove the statement holds it must be true for any (connected) graph G with any (distinct, positive) weights on the edges. To disprove the statement, a single example suffices.

Exercise 11 Given a graph $G = (V, E)$ and its minimum spanning tree T.

(a) *(L2)* We increase the weight of one edge $e \in E$. Can the minimum spanning tree of G change if e was part of the MST? What if e was not part of the MST? Argue your answers.

Solution:
If $e \notin T$ then T never changes. We can proof this by contradiction. Assume that T is no longer a minimum spanning tree. It's weight didn't change, and there is another tree T' that weighs less than T: $w(T') < w(T)$. If $e \in T'$, then the weight of T' was even less before the weight of e was changed. Therefore, originally T could not have been a minimum spanning tree. Thus, after the weight of e is increased, T is still a minimum spanning tree.
If $e \in T$ then T may change. This will depend on how much its weight was increased.

(b) *(L3)* You are given a graph $G = (V, E)$, a minimum spanning tree T of G, and an edge $e \in T$. Consider the graph G' obtained from G by removing edge e. That is $G' = (V, E \setminus \{e\})$. Assume G' is still connected. Let $(S, V - S)$ be a cut that respects $T - \{e\}$, and let e' be a light edge crossing $(S, V - S)$ in G'.

Prove that $T' = T - \{e\} + \{e'\}$ is a minimum spanning tree of G'.

Solution:
Let e' be a light edge of the cut $(S, V - S)$ in G'. Then by the theorem from the lectures (Theorem 23.1 from the book), the light edge e' is part of some MST in G', let's call it T_1'. For sake of contradiction assume $w(T_1') < w(T')$. Otherwise T' will also be a minimum spanning tree of G' and we are done.
Consider the tree T_1' in G. As T_1' is a spanning tree adding any edge creates a cycle involving that edge. Consider the cycle C formed by adding the edge e to T_1'. As e crosses the cut $(S, V - S)$, and e was not part of T_1', at least one other edge of C in T_1' must cross $(S, V - S)$. Let $f \neq e$ be an edge of C crossing the cut $(S, V - S)$. We know that the weight of edge e' was the minimum over all the edges crossing the cut $(S, V - S)$ in graph G', i.e., $w(e') \leq w(f)$.
Let $T_1 = T_1' + \{e\} - \{f\}$, then its weight is $w(T_1) = w(T_1') + w(e) - w(f) \leq w(T_1') + w(e) - w(e') < w(T') + w(e) - w(e') = w(T)$. But then T was not a minimum spanning tree in the original graph G.
We have arrived to a contradiction, and thus T' must be a minimum spanning tree in the graph G'.