1. (a) To prove a relation to be equivalence, we need to prove reflexive, symmetric and transitive.

—Reflexive: Since $e_1 = e_2$ gives us $e_1 \approx e_2$, we can say $\approx$ is reflexive.

—Symmetric: Since $e_1 = e_2$ implies $e_1 \approx e_2$, we can also say that $e_2 = e_1$ implies $e_2 \approx e_1$ and considering $e_1 = e_2$ implies $e_2 = e_1$, $e_1 \approx e_2$ implies $e_2 \approx e_1$ stands true. If $e_1 \approx e_2$ implies e_1 and e_2 are in the same simple cycle, it's fair to say that e_2 and e_1 are in the same simple cycle, which can be implied by $e_2 \approx e_1$. This shows that $e_1 \approx e_2$ can implies $e_2 \approx e_1$ and this relation is symmetric.

— Transitive: Assue that $e_3 \in \bar{E}$. Since we have $e_1 = e_2$ implies $e_1 \approx e_2$, if $e_2 = e_3$. we can say that $e_2 \approx e_3$. Then we will have $e_1 = e_3$, which implies $e_1 \approx e_3$. Considering e_1 and e_2 are two edges in cycle C_1 and e_2 and e_3 are two edges in cycle C_2. Assume that C_1 and C_2 are both simple cycles, we have $e_1 \approx e_2$ and $e_2 \approx e_3$. At this time, e_2 is the common edge of C_1 and C_2, if we remove e_2 here and C_1 and C_2 will be joined a new bigger simple cycle, C_3. Within this C_3, we have e_1 and e_3 as it's edges and it implies $e_1 \approx e_3$. Therefore, we can say that $\approx$ is transitive.

Overall, $\approx$ relation is reflexive, symmetric and transitive, which makes it a equivalence relation.

(b) Since we are talking about edges here, it can either disjoint or one or two shared vertex. If the edges are disjoint, imagining that in a simple cycle, there're two edges and these two edges are considered bi-connected. If the edges have one vertex in common, imagine two edges connected and they are all in a simple cycle, in this situation, we can say that these two edges are bi-connected. If the edges have two vertexs in common, it can not happen in undirected graph because in that situation, these two edges will be considered as one. Therefore, two edges are either disjoint or shave one vertex.

(c) If there is no child in the tree, the root can also be considered as a leaf and removing it will not actually separate the tree because there will be no vertex left. If there is only one child, removing the root will left only one vertex and still, it's not a separation. If there are more than one child, then removing root will cause the tree separated into sub-trees and since this is undirected graph we are talking about, there are no cross edges, or at least no other edges can connect sub-trees once the root is removed. Therefore, if and only if a root has more than one child in the tree can it be a separating vertex.

(d) Similiar to question c up there. If we need a non-root vertex to be a separating vertex, it has to have at least one child to separate with. Also, since it's undirected graph we are talking about, there will be no edges except tree edges and back edges. To separate vertexs means if we remove one vertex, there should be more sub-parts unconnected and as I explained, the only chance here is to have a back edge from one part connecting to another, which can also been expressed as having a child v' none of whose descendents has a back edge to a proper ancestor of v.

(e) This array can be computed by single explore of the tree and if we use DFS, whose time complexity is $O(|E|+|V|)$, we can make sure the array computation will take linear time.

(f) If $pre(v) < low(u)$, where v is a descendant of u, then we can say there is a back edge from v and that u is not a separating vertex. If not, u is a separating vertex. By doing DFS, we can identify all separating vertexs which have a simple cycle and it's a biconnected component. In other words, we can identify biconnected componene by simply one-time DFS, which is linear time.

2. Since T is a MST of graph G, it contains all vertexs and minimal weight edges from G. If there's no cycle in H, then the MST of H must contain T∩H cause some part of T is also the MST of H part. If there is a cycle, the H will have lots of MST, but one of them will contain T∩H. Therefore, T∩H is definitely contained in some MST of H.

3. {1} {2} {3} {4} {5} {6} {7} {8}

union (1, 2)

{1.2} {3} {4} {5} {6} {7} {8}

union (3, 4)

{1.2} {3.4} {5} {6} {7} {8}

union (5, 6)

{1.2} {3.4} {5.6} {7} {8}

union (7, 8)

{1.2} {3.4} {5.6} {7.8}

union (1, 4)

{1.2.3.4} {5.6} {7.8}

union (6, 7)

{1.2.3.4} {5.6.7.8}

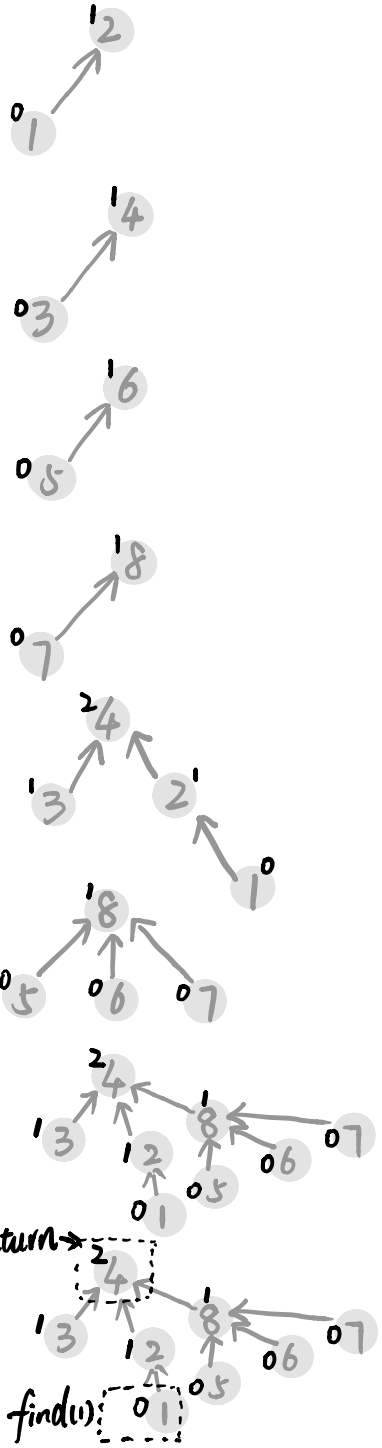
union (4, 5)

{1.2.3.4.5.6.7.8}

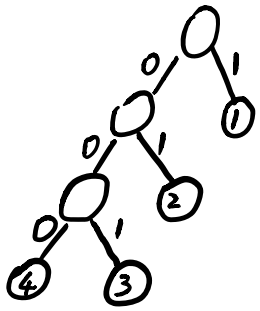
find (1)

set → {1.2.3.4.5.6.7.8}

return 4



4. decreasing frequencies: $f_1 . f_2 . f_3 . f_4$



The longest codeword is 000, (001) and it's 3 bits.

For n symbols of frequencies, the longest code will be n-1 bits.