Graphical user interface, text, application, email

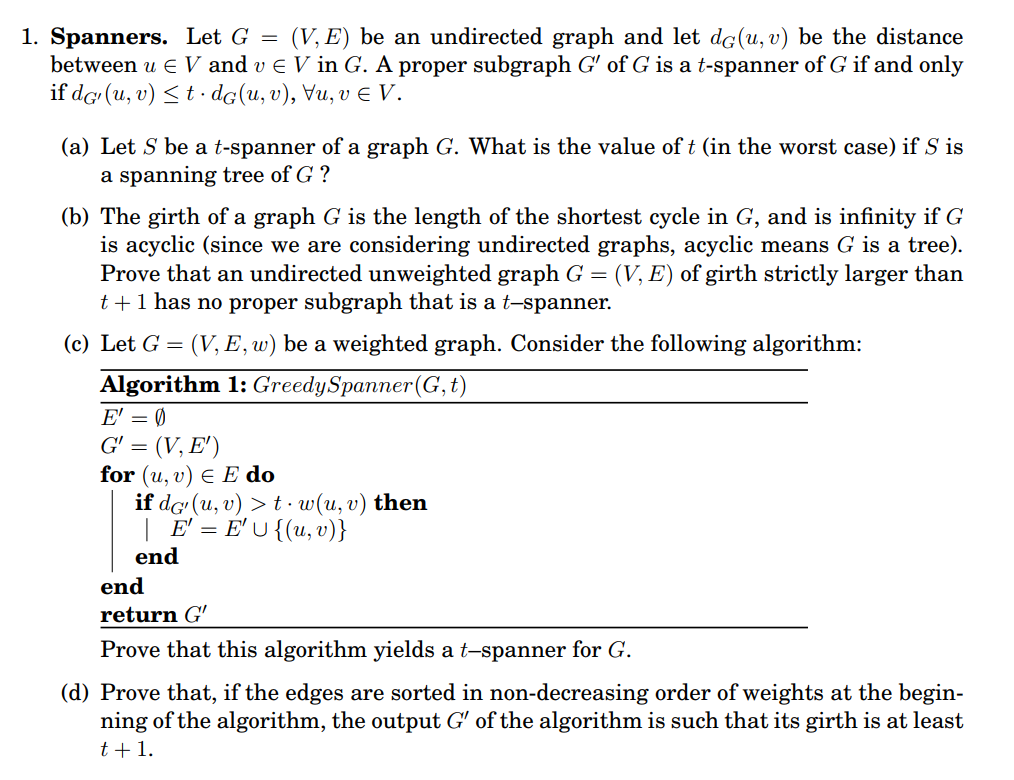
Description automatically generated

In an undirected graph, the distance  between two vertices u and v in the graph G is the number of edges in the shortest possible path out of all paths that connect them together

We can now consider the upper bound of t. Let  and . Graph S is a spanner of G => and , since S is a tree. Suppose that there exists a pair of node u,v such that its shortest connecting path in G is  and its shortest connecting path in S, in worst case, contains all edges in S =>   
=> The upper bound for t is . To prove that this upper bound can be achieved, we must find a graph and its subgraph that satisfy this property.

Let G be a graph containing only one cycle, having n vertices and m edges. Pick two arbitrary adjacent vertices and we know that their distance in G is 1. Now, we construct a subgraph S such that the edge connecting these two vertices are cut off. Since this is a close cycle, the spanner S turns into a line of vertices: two end nodes having degree 1 and other nodes having degree 2 and the end nodes happen to be the two vertices we have chosen earlier. The only path connecting them is along the line and this line is also a spanning tree => The distance between these two vertices in spanner S is n – 1 => We have found the graph and the subgraph that satisfies this property

The value of t (in the worst case) is n – 1, where n is number of vertices in graph G



First, we assume that graph G has many cycles and the length of the shortest cycle is called the girth. A proper subgraph means that the subgraph is a spanning tree and its number of edges is strictly smaller than number of edges of graph G. In other words, at least one edge must be removed from graph G to obtain the proper subgraph. If this edge makes the graph disconnected, t would be infinity and it will never have a proper subgraph that is a t-spanner.   
=> We can only remove an edge from a cycle.   
Observation: consider an edge E(u, v) belongs to any cycles in G. Then  and let denote the length of this cycle as k. Of course, by definition. When the edge is removed, the shortest distance between u and v is to go around the cycle in the opposite direction   
=>   
Another condition we are given is =>  
=> The graph G has  => according to part (a), the spanner property is not satisfied for the vertices (u,v) when the edge E(u,v) is removed => There does not exist any proper subgraph that is a t-spanner (proven)

Graphical user interface, text, application, email

Description automatically generated

First of all, the subgraph G’ is initialized with all the vertices and no edges. If there are no paths connecting u and v in G’, then. We can notice that for all edges in the graph, if there are not any other paths connecting u and v except .   
We can analyze the three cases:  
(1) when u and v has not been connected  
Edges are gradually added to the subgraph. Because of that, when u has been connected to the subgraph by some edges and v is not connected to the subgraph, then is always correct. When E(u,v) is added into E’, v is connected to the subgraph, making G’ a spanning graph of G

(2) is false when u and v has been connected before. In other words, we have, which already satisfies the t-spanner condition, so this edge doesn’t need to be added to the graph at all.

(3)  is true when u and v has been connected before. This breaks the t-spanner condition so the edge is added to the subgraph. After the edge is added to the subgraph, we have updated a new shorter path between u and v, which is => . It happens that  for all integers and. This satisfies the t-spanner condition  
=> The algorithm yields a t-spanner for G.

Graphical user interface, text, application, email

Description automatically generated  
We can use proof by contradiction here.  
First, we assume  => The smallest cycle in this graph has at most t edges.   
We denote the vertices that are connected by the edge with the maximum weigh is and. Due to the non-decreasing ordering of the edge weights, we see that is processed at the last stage, where the smallest cycle has at most t – 1 edges. According to the algorithm, this last edge is added only if . However, because the edge has not been added yet, we have to consider the opposite direction =>  because we know that  is the largest weight =>  (contradiction)  
=> The largest weight edge is not added to the cycle so the smallest cycle in this graph can have more than t edges => (proven)