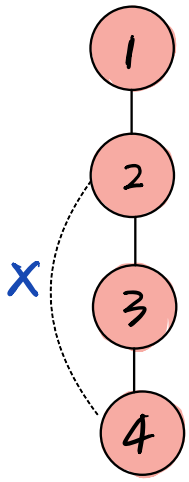
3-2 3-4 3-7 3-9 4-1 4-8 4-9 4-29

3-2. Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(m+n)$ for a graph with n nodes and m edges.




```
Bool flag = false
For each node u in Graph is not marked "Explored"
DFS(u){
   Mark u as "Explored" and add u to R
   For each edge (u, v) incident to u
      If v is not marked "Explored" then
         Recursively invoke DFS(v)
      Else
         flag = true
      Endif
   Endfor
}
Endfor
return flag
```

→ v is marked means both u and v belong to R, thus there is a path from v to u. But now there is also a edge (u, v), so these can form a circle.

3-4

Assuming that all the specimens can be labeled, we can create an indirect graph consisting n nodes (specimens) and m edges (judgments), and the graph must be connected.

Set s as starting node. R will consist of nodes that s has a path to , initially R = {s}.

```
While there is a (u, v) where u is in R but v is not in R
   Add v to R
   If (u, v) == "same" then label v as same as u
   else label v as opposite of u
   Endif
Endwhile
For each edge (u, v) in graph
   If (u, v) == "same" then
      If u and v has different labels,
            there is an inconsistency
      Endif
   Else
      If u and v has same labels,
            there is an inconsistency
      Endif
   Endif
Endfor
```

4. Inspired by the example of that great Cornellian, Vladimir Nabokov, some of your friends have become amateur lepidopterists (they study butterflies). Often when they return from a trip with specimens of butterflies, it is very difficult for them to tell how many distinct species they've caught—thanks to the fact that many species look very similar to one another.

One day they return with n butterflies, and they believe that each belongs to one of two different species, which we'll call A and B for purposes of this discussion. They'd like to divide the n specimens into two groups—those that belong to A and those that belong to B—but it's very hard for them to directly label any one specimen. So they decide to adopt the following approach.

For each pair of specimens i and j, they study them carefully side by side. If they're confident enough in their judgment, then they label the pair (i, j) either "same" (meaning they believe them both to come from the same species) or "different" (meaning they believe them to come from different species). They also have the option of rendering no judgment on a given pair, in which case we'll call the pair *ambiguous*.

So now they have the collection of n specimens, as well as a collection of m judgments (either "same" or "different") for the pairs that were not declared to be ambiguous. They'd like to know if this data is consistent with the idea that each butterfly is from one of species A or B. So more concretely, we'll declare the m judgments to be *consistent* if it is possible to label each specimen either A or B in such a way that for each pair (i, j) labeled "same," it is the case that i and j have the same label; and for each pair (i, j) labeled "different," it is the case that i and j have different labels. They're in the middle of tediously working out whether their judgments are consistent, when one of them realizes that you probably have an algorithm that would answer this question right away.

Give an algorithm with running time $O(m+n)$ that determines whether the m judgments are consistent.

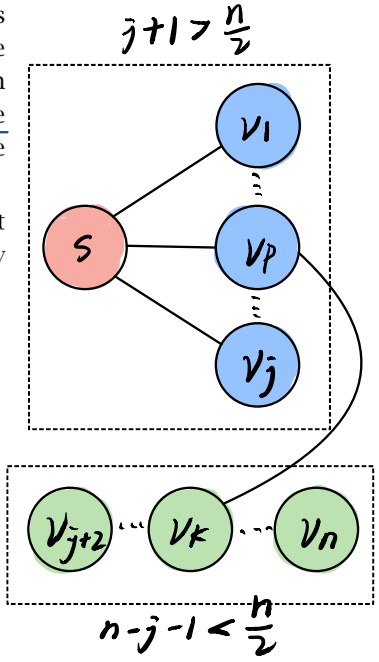
3-7. Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around (actually, as their human owners move around), they define a graph at any point in time as follows: there is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $n/2$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

Here's a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least n/2, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.



True.
Proof:
For each node s in G, R(s) will consist of nodes to which s has a path.
  Because s has degree at least n/2, there are more than n/2 edges(s, *vi*), i = 1,2,...,j (j>=n/2)
  Add these j nodes to R(s)
  For each of the rest n-j-1 nodes in G but not belong to R(s)
    According to Drawer Theory, this node *vk* must have an edge (*vk*, *vp*), *vp* is in R(s)
    Add *vk* to R(s)
  Endfor
  Thus, all nodes are in R(s). It means s has a path to each node in G.
Endfor
For every pair of node u and v in G, there is a path from u to v. So G is connected.

3 – 9. There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running time $O(m + n)$ to find such a node v.

Run BFS starting from s can build a series of layers. Assuming that t is on the layer d, we can obtain d>n/2.

Proof1: there is at least one layer has only one node.
If all the layers have more than one node, there would be a cycle has greater than 2*n/2=n nodes, which contradicts with G has n nodes.

Proof 2: there exists some nodes v, such that deleting v from G destroys all s-t paths.
Find a layer *Li* (i<d) has only one node v.
Let R consist of node s and all the nodes in *L1~Li-1.*
According to BFS, any edge out of R must in *Li*.
Since any path from s to t must leave R at some points, it must contain a node in *Li*.
Because there is only one node in *Li*, so deleting it will destroys all s-t paths.

4-1. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary connected, undirected graph with a distinct cost $c(e)$ on every edge e. Suppose e^* is the cheapest edge in G; that is, $c(e^*) < c(e)$ for every edge $e \neq e^*$. Then there is a minimum spanning tree T of G that contains the edge e^*.

True.
e* is the first edge that would be included in the minimum spanning tree according to Kruskal's algorithm.

4-8. Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

Assuming that there are two distinct minimum spanning trees T and T' of G.
Because a n-node tree has exactly n-1 edges, T and T' have the same number m of edges.
Let E(T)={e1, e2, ..., em}, E(T')={e'1, e'2, ..., e'm}
Let ek be the first edge meets ek != e'k, suppose that weight(ek)<weight(e'k)
Add ek to T' can get a cycle C (this cycle do not contain e'1, e'2, ..., e'k-1, otherwise exists a cycle containing ek in T)
If we delete any edge (not ek) in C, we can obtain a new spanning tree with lighter total weight, which contradicts the assumption that T' is a minimum spanning tree.
Thus, G has a unique minimum spanning tree.

4-9. One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum *total* cost. Here we explore another type of objective: designing a spanning network for which the *most expensive* edge is as cheap as possible.

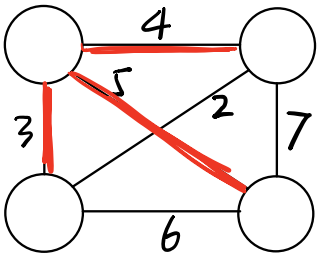
Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the *bottleneck edge* of T to be the edge of T with the greatest cost.

A spanning tree T of G is a *minimum-bottleneck spanning tree* if there is no spanning tree T' of G with a cheaper bottleneck edge.

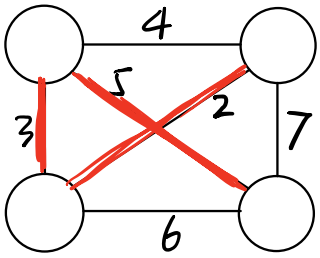
(a) Is every minimum-bottleneck tree of G a minimum spanning tree of G? Prove or give a counterexample.

(b) Is every minimum spanning tree of G a minimum-bottleneck tree of G? Prove or give a counterexample.

(a) False.



minimum-bottleneck spanning tree



minimum spanning tree

(b) True.

Assuming that T is a minimum spanning tree and T' is a spanning tree with lighter bottleneck edge. Thus, T contain an edge ek that is heavier than every edges in T'.
If we delete ek in T, it divides into two tree T1, T2.
Connect T1 and T2 with the edge used in T' to connect T1 and T2.
We can obtain a new tree which has a lighter total weight than original T.
This contradicts the fact that T is a minimum spanning tree.

4-29. Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$. (That is, if $V = \{v_1, v_2, \ldots, v_n\}$, then the degree of v_i should be exactly d_i.) G should not contain multiple edges between the same pair of nodes, or "loop" edges with both endpoints equal to the same node.

If one of the degrees d_i equals zero, it must be an isolated node. Thus we can delete d_i from the list.

Method: Sort the degree list let $d_1 \geq d_2 \geq \cdots \geq d_m$ $(m \leq n)$

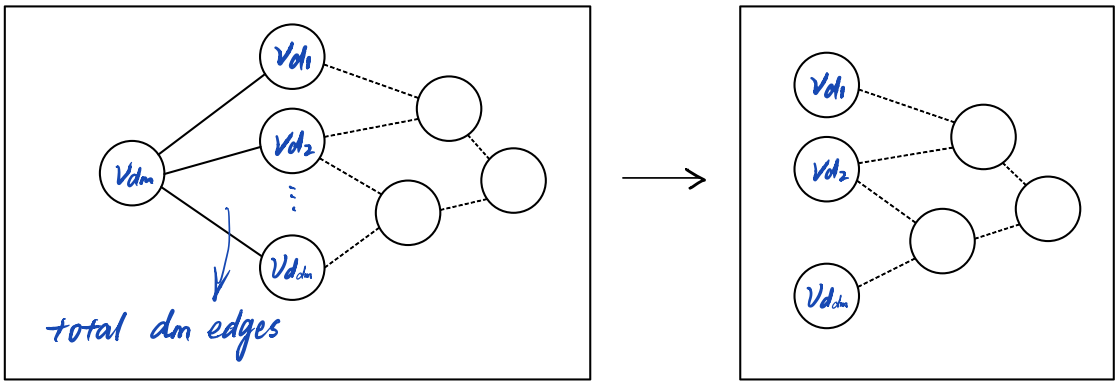
Then we drop the last number and subtract 1 from the first d_m number.

The new list is $L = \{d_1-1, d_2-1, \cdots, d_{d_m}-1, d_{d_m+1}, \cdots, d_{m-1}\}$

Repeat above method until all the degrees become zero.

If a degree turns to a negative number, it can be an indirect graph.

This method uses Havel-Hakimi Theory:



The result whether these two lists of degrees can form an indirect graph is same.

Otherwise, there exsit $i<j$ so that v_n is joined to v_j but not v_i. Since $d_i \geq d_j$, there must be some v_k meets (v_i, v_k) is an edge but (v_j, v_k) isn't. If we replace these two edges by (v_i, v_n) and (v_j, v_k).