Solutions to question set for Chapter 2

1. Suppose that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are connected graphs. Let x be a vertex of G_1 and let y be a vertex of G_2. Form a new graph $G = (V_1 \cup V_2, E_1 \cup E_2 \cup \{xy\})$.

   (a) Draw the graph G if G_1 and G_2 are specified by:

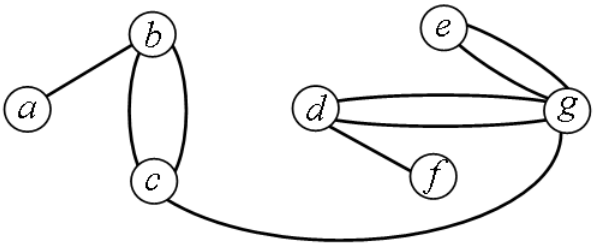
   $$V_1 = \{a, b, c\},\ E_1 = \{ab, bc, bc\},\ V_2 = \{d, e, f, g\},\ E_2 = \{df, dg, dg, eg, eg\},$$

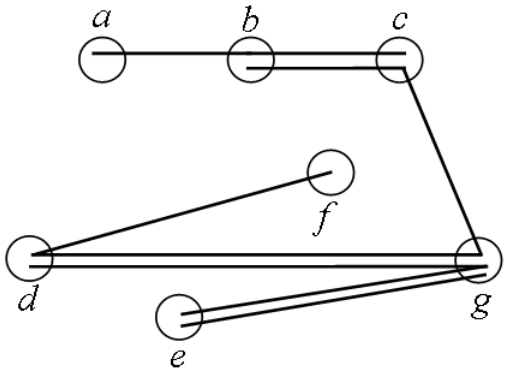
   and if $x = c$ and $y = g$.

   (b) Prove that the new graph G is connected (i.e. not just the one you constructed in part (a) but *any* G you construct in this way).

SOLUTION:

(a) The drawing might be either of the two shown below, or anything which depicts the structure of G_1 and G_2 and the connection which G makes between them:





(b) This is similar to the proof that the Maximal_Subtree algorithm produces a connected subgraph: we must show that, for any vertices u and v in G, there is a walk joining u and v. We observe that if W is a walk in G_1 or in G_2 then this is still a walk in G. (We could prove this but we will assume that it is obvious). Now there are two cases to consider:

Case 1: $u, v \in V(G_1)$, or $u, v \in V(G_2)$. Without loss of generality assume both u and v are vertices of G_1. Then there is a walk from u to v in G_1, and hence in G, because G_1 was assumed to be connected.

Case 2: $u \in V(G_1)$ and $v \in V(G_2)$. Recall that G contains an edge xy, $x \in V(G_1)$, $y \in V(G_2)$. Connectivity of G_1 means there is a walk, say W_1, joining u and x. Connectivity of G_2 means there is a walk, say W_2, joining y and v. Now W_1, xy, W_2 forms a walk joining u and v in G.

We conclude that G is connected. □

---

2. It is required to design an algorithm

st_Path(G,s,t)
Input: graph G, vertices $s, t \in V(G)$
Output: True if there is a path from s to t in G; False otherwise

(a) Specify suitable pseudocode for st_Path.

[Hint: adapt the pseudocode for Maximal_Subtree given in the lecture notes. You will **NOT** be asked to answer a question like this in the final exam, but it is good practice for thinking about algorithms.]

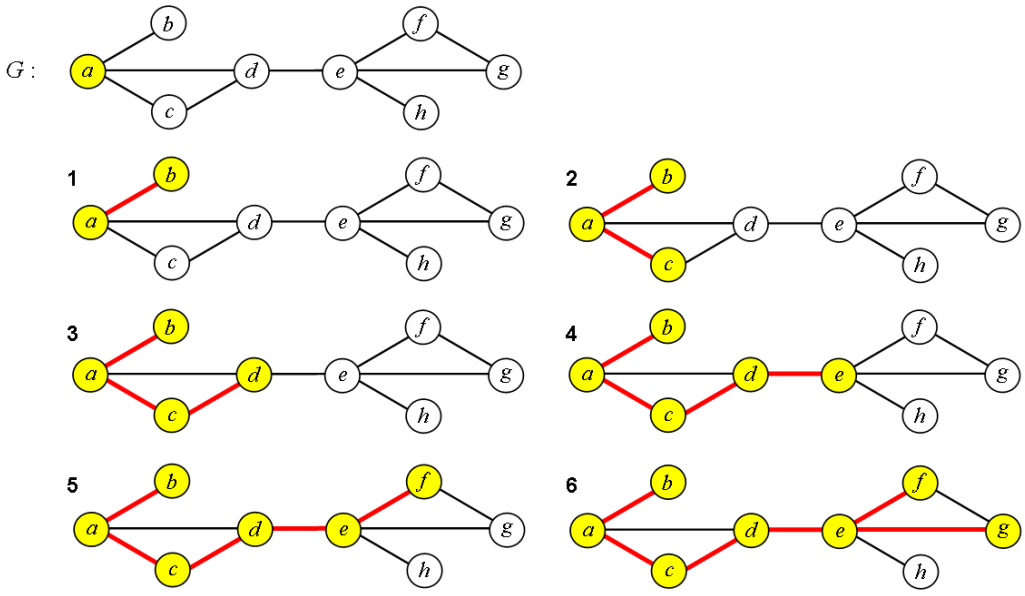
(b) Show how your algorithm runs on the following input, showing **diagrammatically** each step which the algorithm makes:

$$\begin{aligned} V(G) &= \{a, b, c, d, e, f, g, h\}; \\ E(G) &= \{ab, ac, ad, cd, de, ef, eg, eh, fg\}; \\ s &= a,\ t = g. \end{aligned}$$

SOLUTION: (a) We can adapt the Maximal_Subtree algorithm as follows:

st_Path(G,s,t)
Input: graph G, vertices $s, t \in V(G)$
Output: True if there is a path from s to t in G; False otherwise
$T := (\{s\}, \emptyset)$ *# we will grow T as a tree to try and reach t*
while there is some edge xy leaving T **do**
  if $y = t$ then **return**(True) **fi**;
  $V(T) := V(T) \cup \{y\}$; $E(T) := E(T) \cup \{xy\}$
od;
return(False) *# if we reach this point we know we cannot reach t*

(b) In order to show how this algorithm performs on the input graph diagrammatically we need to draw it and then indicate how edges are added until vertex $t = g$ is reached. This is shown below (note that an actual path from s to t is not required by the specification of the algorithm, and is not given):



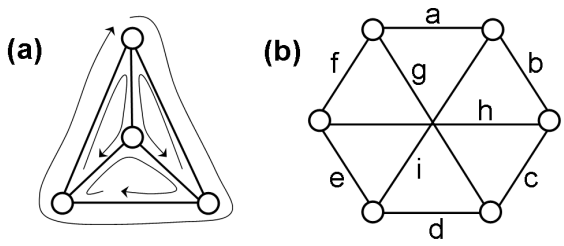
---

3. A **bridge** in a connected graph G is an edge whose deletion will disconnect the graph. One of the most famous open questions in graph theory, posed independently in the 1970's by Paul Seymour and George Szekeres, is the following:

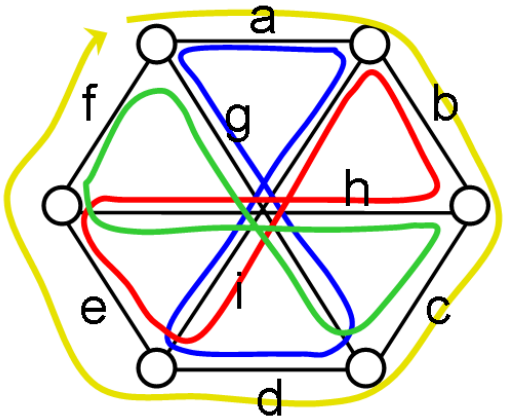
   The Cycle Double Cover Conjecture If G is a graph with no bridges then there is a list of cycles in G such that every edge of G lies in exactly two of these cycles.

   For example in the graph (a) on the right (which is called K_4) four cycles may be chosen as shown.

   Specify a list of cycles which provides a cycle double cover for the graph in (b).



SOLUTION: The following image shows a cycle double cover for the graph:



To specify the cover more formally, an incidence matrix is convenient:

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} a & b & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} 1 & & & 1 & & & 1 & & 1 \\ & 1 & & & 1 & & & 1 & 1 \\ & & 1 & & & 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & & & \end{pmatrix} \end{array}$$

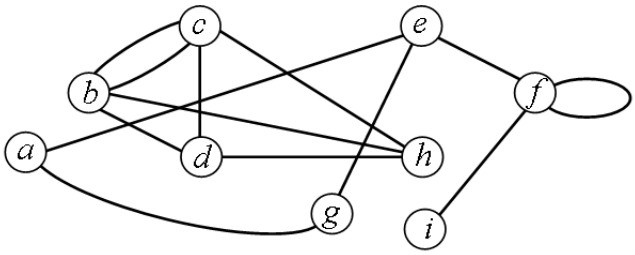
There is a row for each cycle and the columns are indexed by the edges. Since each column sum is 2 this confirms that we have a double cover.

---

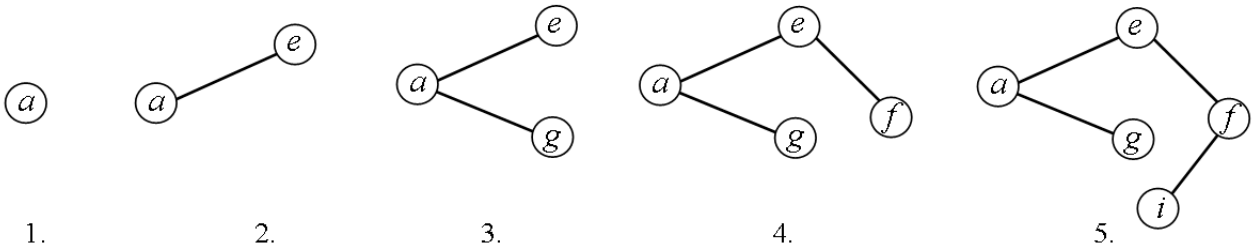
4. **[FOR MARKING]** Run the Maximal_Subtree algorithm on the graph shown on the right, starting at

(i) vertex *a*; (ii) vertex *b*;

showing each iteration your algorithm makes to construct a maximal subtree. In each case, give the number of vertices and edges contained in the output subtree and state whether or not the subtree is maximum.



SOLUTION: (i) Starting at vertex *a* we have the following sequence of construction steps:



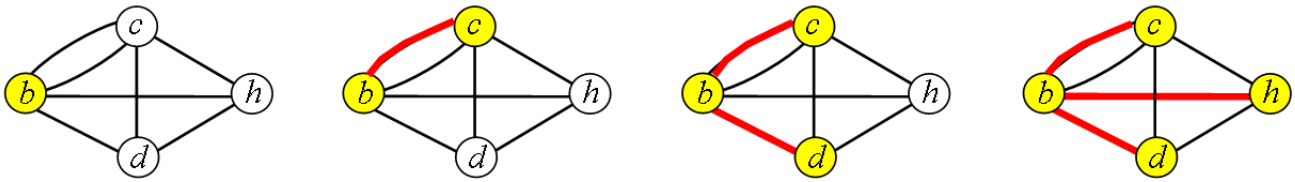
1. 2. 3. 4. 5.

*[***NOTE:** *there is some choice here: e.g. at step 2. edge ag could have been chosen instead of edge ae.]*

I do **not** *have to redraw the edges as they are drawn in the question—any convenient redrawing will do.*

This subtree has 5 vertices and 4 edges and is maximum: the graph contains no greater subtree.

(i) Starting at vertex *b* we have the following sequence of construction steps:



1. 2. 3. 4.

*[***NOTE:** *this time there is a choice of four edges at step 2. I have consistently chosen the vertices in alphabetical order.*

I have given the answer here displayed as a series of subtrees *instead of a series of* trees *(as in part (i)). Either is acceptable for this algorithm. Drawing trees is obviously much less work; however, for some algorithms it is required to show the whole graph at each step.]*

This time the subtree has 4 vertices and 4 edges and is maximal: it cannot be extended further without creating a cycle or a disconnected subgraph; but the graph contains a larger subtree.
