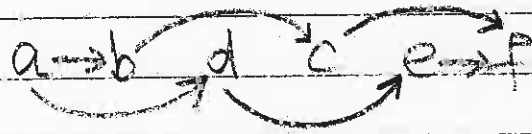
CMPT 307 (hw2)

1. Consider one topological order:



a must come first & f must come last. b must be to the left of c & d must be to the left of e.

- b comes first, either d or c comes next
  - c comes next; The ordering is a b c d e f
  - d comes next; The orderings are a b d c e f & a b d e c f

- d comes first, either b or e comes next
  - e comes next; The ordering is a d e b c f
  - b comes next; The orderings are a d b c e f & a d b e c f.

There are 6 orderings in total.

Note In the algorithm for topological sort, the # of orderings can be determined by keeping track of the # of nodes whose remaining edges are all outgoing. (Try yourself)

3.2. Discussed in the class

3.5. Induction proof:

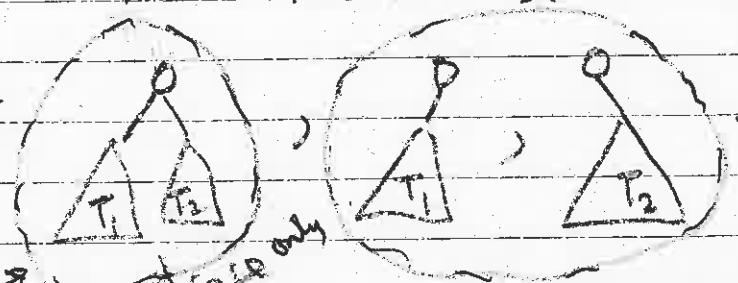
First consider basis: For $n = 3$, evt can have the following binary trees

 + other symmetric cases!

In each of the above cases, # of nodes with two children is exactly one less than the number of leaves. $\Rightarrow$ basis is true

Induction hypothesis: Suppose the property is true for any binary tree with nodes less than n.

Consider a binary tree with n nodes. The tree could be one of the following types:



→ you try to prove.

Considering this case only

Let n_i be the # of nodes in T_i.

Let m_i be the # of nodes with degree 2 in T_i.

$\therefore$ # of leaves in T_1 & T_2 = $(m_1 - 1) + (m_2 - 1)$

$= m_1 + m_2 - 2$. (hypothesis)

$\therefore$ # of leaves in $T = (m_1 + m_2 - 2)$

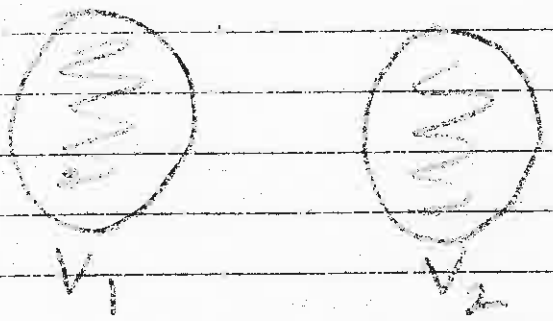
# of nodes with degree 2 = $m_1 + m_2 + 1$

Thus the statement is true by induction

3.7 Proof by contradiction:

Suppose $G = (V, E)$ is not connected.

∴ We have

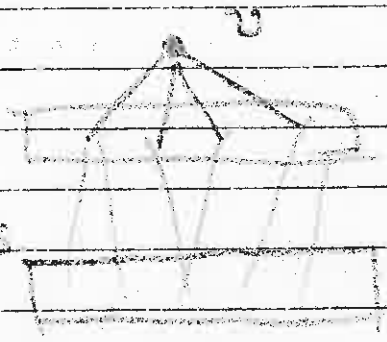


There is no edge between V_1 & V_2.

Since $|V| = n$, $\min\{|V_1|, |V_2|\} \leq \frac{n}{2}$.

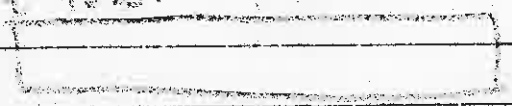
This implies that nodes of V_1 or V_2 have degrees at most $n/2 - 1$ ⟶ a contradiction.

3.10 First we compute BFS from v & suppose ω appears in layer L_i.



Note: All the shortest paths from v to ω is of length i.

Let $S_q(\omega)$ = # of shortest paths of length $i+\omega$.



Let $v_1, v_2, \ldots, v_k$ be the nodes in layer L_{i-1} which are adjacent to w.

Let $S_{i-1}(v_j)$ be the # of shortest paths of length $i-1$ from u to v_j. We can therefore say that # of shortest paths to w is $S_{i-1}(v_1) + S_{i-1}(v_2) + \cdots + S_{i-1}(v_k)$

$\therefore S_i(w) = \sum_{j=1}^{k} S_{i-1}(v_j)$.

We now determine each $S_{i-1}(v_j)$ by looking at the nodes in layer L_{i-2} adjacent to v_j.

The process is repeated.

2. Straightforward to reverse the adjacency list of G.

3.a) Clearly, if the graph G has an Euler tour, the degree of each node of G must be even.

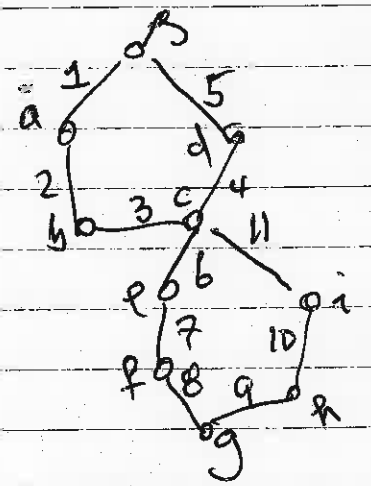
We now show that if the degree of each node is even, there exists an Euler tour of G. Let s be an arbitrary starting vertex of G from where we start the traversing. We observe that

(i) Every time a node is visited from s, the # (remaining) of unvisited edges incident to s is odd.

(ii) Every time a node v ($v \neq s$) is visited the number of (remaining) unvisited edges incident to v is odd.

The above observations tell us that when we leave s, there is an unvisited edge incident to s & therefore, s will be visited again. Moreover, when we enter v, $v \neq s$, there is an unvisited edge incident to v, and therefore, ~~it will be visited again~~ it is always possible to leave v.

(b) We will use a traversal of G starting from an arbitrary vertex s. It is possible to end the search at s with no uncovered edge incident on s, but still not all edges covered. Consider the following graph:



The traversal starts from s. We traverse the edges labeled 1, 2, 3, 4, 5 & then I am in s & can't go anywhere

Sol?: Let C be the cycle (traversed). Look at all the nodes on the cycle and check if there is any vertex which has an uncovered edge. In our example, C = <1, 2, 3, 4, 5> and Vertex c is such a node with uncovered edges. We therefore, restart the walk starting from c. Every time we are stuck in the walk, we check for a vertex with uncovered edge. The process gets repeated & eventually all the edges will be traversed. Since each edge needs to be traversed once, as soon as it is covered, the edge must be removed from the list storing the graph. How will you do it? Can you do it if it is stored in an adjacency matrix?