Assignment 3

Due 23:59, Thursday, April 09, 2020

CS ID 1:	
CS ID 2:	

Instructions:

1. Do not change the problem statements we are giving you. Simply add your solutions by editing this latex document.
2. Take as much space as you need for each problem. You'll tell us where your solutions are when you submit your paper to gradescope.
3. Export the completed assignment as a PDF file for upload to gradescope.
4. On gradescope, upload only **one** copy per partnership. (Instructions for uploading to gradescope will be posted on the HW3 page of the course website.)

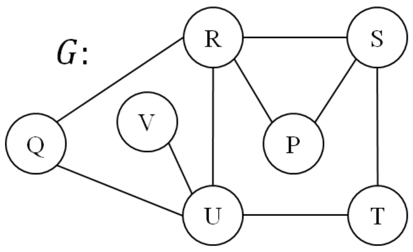
1. **Go Eulers! (25 points).**

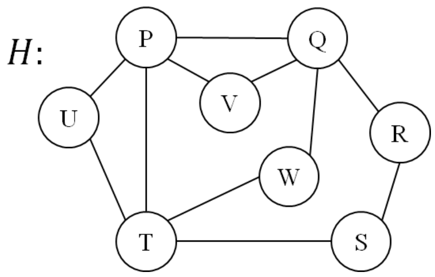
   [Hedayat: I think it would be helpful to have an example that clarifies the difference between path and trail, and cycle and circuit.]

   Some terminology for this problem. Assume we are working with connected graphs:

   - **path**: a sequence of edges which joins a sequence of vertices.
   - **simple path**: a path whose vertices are all distinct.
   - **cycle**: a path whose first and last vertices are the same.
   - **trail**: a path whose edges are all distinct.
   - **circuit**: a trail whose first and last vertices are the same.
   - **Eulerian trail**: a trail which uses every edge in the graph exactly once
   - **Eulerian circuit**: a circuit which uses every edge in the graph exactly once
   - **Eulerian graph**: a graph which has an Eulerian trail or circuit.
   - **bridge**: an edge which, if removed from the graph, increases the number of connected components in the graph.

   Consider the following undirected graphs:



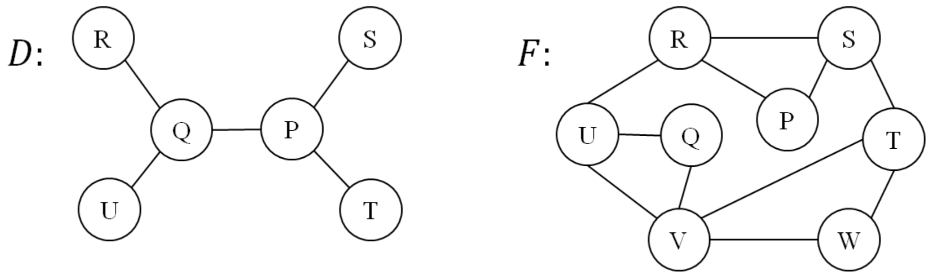


   (3 points) In the spaces here, write a sequence of vertices describing an Eulerian trail or circuit in each of the graphs G and H.

   Eulerian trail in graph G: $\boxed{\text{V-U-R-P-S-R-Q-U-T-S}}$

   Eulerian circuit in graph H: $\boxed{\text{U-T-W-Q-V-P-T-S-R-Q-P-U}}$

The following undirected graphs do not have Eulerian trails:



(2 points) What is the common characteristic of these graphs D and F that prevent the existence of an Eulerian trail? Complete the sentence:

A graph has no Eulerian path or circuit if...

More than two vertices have odd degree

(2 points) Referring to graphs G and H above, one has an Eulerian circuit; the other has an Eulerian trail but not an Eulerian circuit. What must be the conditions for an Eulerian circuit to exist? Complete the sentence:

A graph has an Eulerian circuit if...

Every vertex has even degree

(4 points) Based on your findings in the previous sections, use pseudocode to describe an algorithm $IsEulerian(G)$ which takes a connected graph $G = (V, E)$ as input, and outputs $true$ or $false$ indicating whether an Eulerian circuit exists in G.

```
function IsEulerian(G = (V, E))

    for all vertices v in V do
        count the number of neighbours of v
        if v has an odd number of neighbours then
            return FALSE
        end if
    end for
    return TRUE
end function
```

(2 points) What is the worst-case complexity of your implementation of $IsEulerian(G)$, assuming your graph is represented using an adjacency list?

$$\boxed{\sum_{v \in V} deg(v) = O(|E|)}$$

Assuming we know that a graph $G = (V, E)$ has an Eulerian trail, arrange the statements below, adding parenthesization to complete the algorithm $FindETrail(G)$ which will output an Eulerian trail in G. There may be statements that are used more than once.

(A) consider a neighbour u, such that edge e connects $current$ and u

(B) Set $current = u$

(C) remove e from E

(D) Set $current = v$

(E) If e is not $\boxed{}$

(F) While $|E| > 0$

(G) Find a vertex that $\boxed{}$, call this vertex v

(H) Output $current$

(4 points) Rewrite your properly ordered and parenthesized code here. It will be helpful to the grader if you retain the enumerated labels next to each line of code.

```
function FindETrail(G = (V, E))

    G. Find a vertex that has odd degree, or if none exists, choose any vertex, call this
  vertex v
    D. Set current = v
    H. Output current
    while F. $|E| > 0$ do
        A. Consider a neighbour $u$, such that edge $e$ connects $current$ and $u$
        if E. $e$ is not a bridge then
            C. remove $e$ from $E$
            B. Set $current = u$
            H. Output current
        end if
    end while
end function
```

(3 points) Assuming that determining whether or not a single edge e has the conditional check property (line E) in the algorithm can be performed in $O(|V|+|E|)$ time on an adjacency list, what is the worst-case complexity of $FindETrail(G)$? Briefly justify your answer.

Find a vertex of odd degree: $O(|E|)$

While loop iterates $|E|$ times. Each iteration of the while loop has worst case cost $O(|V|+|E|)$.

The cost of performing the bridge check may decrease as the algorithm progresses, but a pessimistic upper bound can be estimated as $O(|E| \cdot (|V| + |E|))$

OPTIONAL BONUS (1 point) Very briefly describe in 2-5 lines, how to determine in $O(|V| + |E|)$ time, whether or not an edge e has the conditional check property (line E) in the algorithm above.

Perform DFS from v and count the number of reachable vertices.
Remove e from E.
Perform DFS from v again and count the number of reachable vertices.
If the two counts of reachable vertices are different, then e is a bridge.

(3 points) The algorithm above employs an expensive conditional check for a particular substructure, so we wonder if perhaps we can find an Eulerian Trail more efficiently! Below is pseudo-code for a simplified version of depth-first-search. Augment and adapt the code so that it produces the *reverse* of an Eulerian Trail in a given graph. You may assume that such a path exists.

```
DFS( $G$, $v$ )
  if $v$ has degree $> 0$
    let $w$ be some neighbour of $v$
    remove the edge $(v, w)$
      DFS($G$, $w$)
  otherwise
    output $v$
```

(2 points) What is the running time of the algorithm in the previous part?

Assuming degree can be looked up in constant time, this has complexity $O(|V| + |E|)$ (same as basic DFS)

2. **DNA sequencing (16 points).**

[Hedayat: I think it is nice that students see an example of how graph theory be used in a real world application, but the questions that are being asked are not asking for much graph theory/algorithm understanding (e.g., listing all possible genome sequences may not be an interesting question. I think students do not take much from it.]

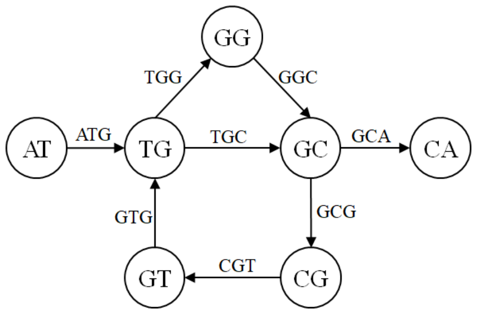
DNA is a molecule consisting of two intertwined polynucleotide chains and encodes the genetic information of (almost) all known organisms. The chains are formed from four types of nucleotide molecules (abbreviated as A, T, C, and G) which appear in the two strands as complementary base pairs. The COVID-19 genome was recently sequenced and determined to be approximately 29900 base pairs in length. However, sequencing equipment are unable to read the entire length of the viral genome to produce the sequence, and are only able to read much shorter sequences at a time. Thus, the genome is sequenced by first producing many copies of the genome to be sequenced, physically/chemically breaking them into many short fragments, and then sequencing the fragments and joining the sequenced fragments together to form the full genome sequence.

One of the difficulties in reconstructing the full genomic sequence arises from the fact that there will be many overlapping fragments. For example, many copies of the sequence ACTTGACGCTAC may produce the fragments { TGAC, TTG, TAC, CGC, CTAC, CTTG, ACG, ACT}. Starting from only the fragments, it is not at all obvious which one starts the sequence.

For simplicity, suppose that the fragmentation process always produces fragments of length $k = 3$. A directed graph is constructed as follows:

- For each fragment of length k, create a vertex for each of the two sequences of length $k-1$ in the fragment, if such vertices do not already exist. For example, a fragment xyz will create a vertex xy and a vertex yz.
- For each fragment xyz, draw a directed edge from vertex xy to yz (if $k = 3$, for example).

For example, the fragments { GTG, GCA, ATG, CGT, GGC, TGC, GCG, TGG } will form the following directed graph below:



(4 points) Study the graph above carefully. Using the knowledge that the original sequence produced *all* of the fragments used to construct the graph, briefly describe how to reconstruct a genome sequence from the graph.

Find an Eulerian trail in the graph.
Begin the genome sequence using the label from the first vertex.

For each vertex in the sequence of the Eulerian trail, append to the genome sequence the "overhang" of the vertex label when overlapped with the previous vertex label. In this case, it is the last character of the current vertex label.

(3 points) The graph and algorithm on the previous page could produce more than one full genome sequence. Write all of the full sequences here:

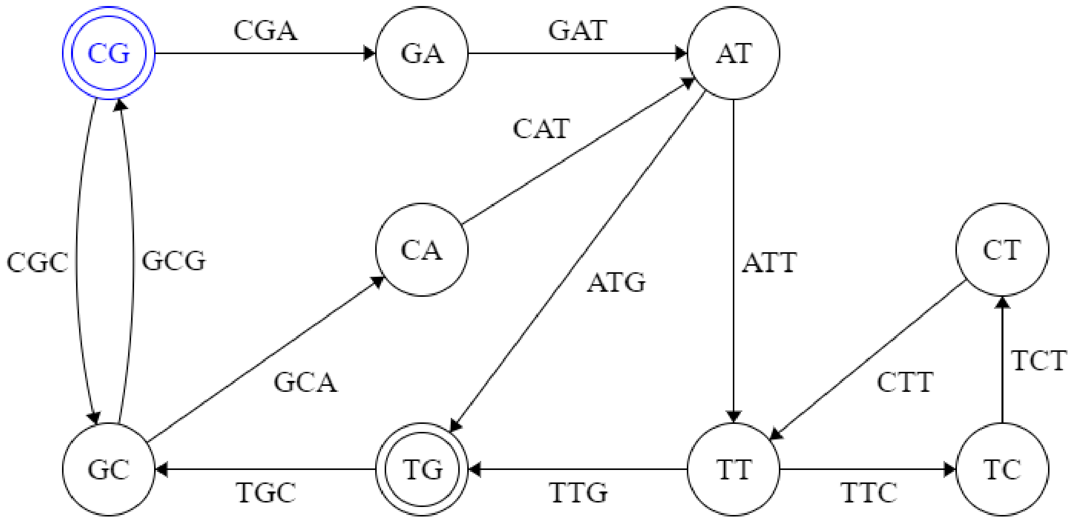
Sequence 1: ATGGCGTGCA

Sequence 2: ATGCGTGGCA

Thus, there may be multiple candidates produced for the "correct" full genome sequence, and determination of the genome sequence is then probabilistic. Counting the number of solutions in such a graph is beyond the scope of this assignment.

Given the fragments { TGC, GAT, CGA, ATT, GCA, TTG, CAT, TCT, GCG, TTC, CGC, ATG, CTT }, construct the directed graph which will be used to find the genome sequence. There will be 9 vertices in the graph.

(5 points) Insert or draw your graph here:



(4 points) List all the possible genome sequences that can be determined from your graph.

Sequence 1: CGATTCTTGCGCATG

Sequence 2: CGATGCGCATTCTTG

Sequence 3: CGCGATTCTTGCATG

Sequence 4: CGCGATGCATTCTTG

Sequence 5: CGCATTCTTGCGATG

Sequence 6: CGCATGCGATTCTTG

3. **It can't be that hard (5 points).**

[Hedayat: I liked how computing Eulerian trail/circuit is compared with Hamiltonian path/-cycle. Maybe it would be helpful to have this as the second question so that the comparison is easier.]

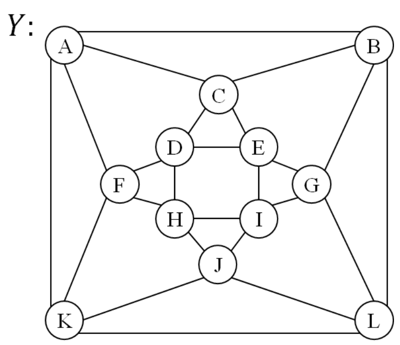
Recall the definitions from the first problem in this homework:

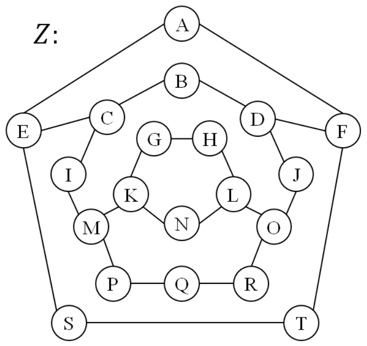
- **Eulerian trail**: a trail which uses every edge in the graph exactly once
- **Eulerian circuit**: a circuit which uses every edge in the graph exactly once

Consider these very similar definitions:

- **Hamiltonian path**: a path which uses every vertex in the graph exactly once
- **Hamiltonian cycle**: a path which uses every vertex in the graph exactly once and returns to the starting vertex

The graph Y below has several Hamiltonian cycles. The graph Z below does not have a Hamiltonian path or cycle.





(2 points) Write the sequence of vertices for one such Hamiltonian cycle for the graph Y. It will be helpful to the grader if you visually highlight the edges in the figure above.

Example Hamiltonian cycle in Y: A-K-J-L-G-B-C-E-I-H-D-F-A

(3 points) Briefly describe how you could algorithmically determine if a graph has a Hamiltonian cycle, and estimate with justification the asymptotic complexity of your algorithm. Contrast this with the complexity of your best solution for the Eulerian circuit problem.

Produce all permutations of vertices, call this collection P.

For each permutation p in P, check that the sequence forms a valid path in the graph (i.e. an edge exists between each pair of vertices in the sequence of p). If p is a valid path, check that there is an edge between the last and first vertices in the sequence. If there is, return true.

If no permutation is a valid path and has an edge from the last vertex to the first vertex, return false.

Complexity: Let $|V| = n$. There are $n!$ permutations of vertices. Checking a single permutation has a cost of $\Theta(n)$, so the overall complexity of this algorithm is $\Theta(n \cdot n!)$ which is significantly less efficient than the cost of finding an Eulerian circuit.