CS 344 Homework #2

Austin Bennett, Mark Barkalow

October 20, 2019

1 Question #1

Given an undirected graph G (which consists of a set of vertices and edges) and a particular edge e we want to determine whether G has a cycle containing e.
A cycle is defined by some number of vertices (at least 3) connected in a closed chain. Any way you traverse the closed chain should result with the same traversed vertices.
A way to test if a particular edge is within a closed chain like this is to remove the edge from the graph G and to see if you can still reach one of the vertices in edge e from the other vertice in edge e.
Thus the algorithm is fairly simple: We start by removing edge e from G.
Then explore(one vertice of e)   (explore(v) is defined in lecture 7)
if(visited(other vertice of e) == true) a cycle is present in graph G.
else no cycle is present in graph G.

2 Question #2

An algorithm that computes the minimum number of semesters necessary to complete the CS curriculum is simple. Since a student is assumed to able to complete as many classes as they have the prerequisites for in a single semester; the total number of semesters required to complete the curriculum is just the course with the most prerequisites.
The algorithm could then be as simple as searching through each connected component and taking note of the maximum vertices in each component.
The highest returned value is the number of required semesters to complete the CS curriculum.

3 Question #3

To determine whether a graph G has a cycle of at least length four can be calculated with the following algorithm:
Select two distinct vertices (x and y) such that x $\notin$ adj(y) and thus y $\notin$ adj(x)
For all z $\in$ adj(x), check if z $\in$ adj(y).
If two vertices z are in common between adj(x) and adj(y) then there is a cycle of length $= 4$

4 Question #4

Show that if an undirected graph with n vertices has k connected components, then it has at least n - k edges.
Due to the context of the problem at hand we can safely ignore "lone vertices" because a vertice that has no edge connecting itself to another vertice counts as both a vertice and a connected component in the undirected graph.
For example, if we had an undirected graph G = {V, E} where E = {} and V = {1, 2, 3, 4}: we then have 4 connected components and 4 vertices.
Plugging into our equation we get n-k == 4 - 4 = 0. Thus adding any "lone vertices" to an undirected graph has no impact on the problem at hand.
We will now consider a connected component to be at least 2 vertices, and thus at least 1 edge. (since as shown above connected components with only 1 vertice are irrelevant)

If our minimum connected component has 2 vertices and 1 edge we see that the addition of this connected component to our undirected graph would result in n += 2 and k += 1 while the requirement of our edges increases by 1. If we were to add vertices to this connected graph however, no matter where the vertice is added: at least one additional edge is required (more edges can occur during cycles without the addition of vertices). (n += 1 and e += 1).

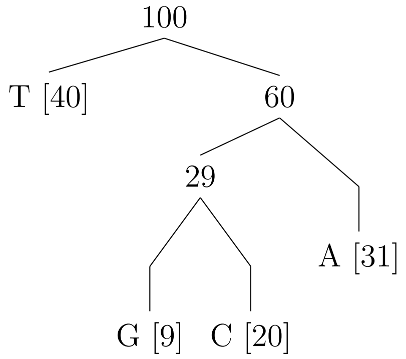
It is fairly straightforward to deduce from here that for each connected component the edges required is at least the number of vertices in the connected component - 1. Again, this is because the connected component is at a minimum 2 vertices and 1 edge, but expanding on this connected component will never again give you 2 vertices at the cost of 1 edge. The vertice gain for each edge is at most 1.
Thus we have that an undirected graph G with n vertices and k connected components must have at least k + n - 2k edges.
(+ k) because we know that each connected component must have at least 1 edge.
(+ n - 2k) this gives us the number of vertices we have that are in addition to our minimum connected components (1 new vertice = 1 new edge)
k + n - 2k = n - k
n - k = n - k   q.e.d.

5 Question #5

In class we discussed an algorithm, Kruskal's algorithm, for finding the minimum spanning tree of a graph. To find the maximum spanning tree we can use the exact same algorithm with a minor tweak to the graph to instead find a maximum spanning tree. By negating all values/costs in the graph in question we transform all of the previous maximums of the graphs to the new minimums of the graph. By applying Kruskal's algorithm to find "the minimum spanning tree" to the altered graph we have effectively found a maximum spanning tree of the graph. After the algorithm has produced a minimum spanning tree, again negate the values/costs and you will have the maxmium spanning tree of the original graph.

6 Question #6

For a string containing the letters {A, C, G, T} occuring at frequencies {31%, 20%, 9%, 40%} the huffman encoding graph would look something like:



Letting left tree branches = 0
and right tree branches = 1
We obtain the huffman encoding of:
T - 0
A - 11
G - 100
C - 101

If huffman encoding was not used and every codeword had length 2, for 100 million symbols it would require 200 Mb.
With the use of huffman encoding we have:
T: 40 million * 1 bit = 40 Mb
A: 31 million * 2 bits = 62 Mb
G: 9 million * 3 bits = 27 Mb
C: 20 million * 3 bits = 60 Mb
Total = 189 Mb