Lab 05

Adjacency Matrix conversion

while we know that an adjacency matrix requires more memory to store a graph than an adjacency List ($\Theta(|V|^2)$ compared to $\Theta(|E|+|V|)$ where V is vertex and E is edge) the adjacency matrix does one thing well, looks up edges. Looking up an edge in an adjacency matrix can be done in constant time or $\Theta(1)$. This is helpful for converting a matrix to a list becase if we know what edges exist we can just push them to our list. If we break down the runtime of the code I created to do the conversion, it isn't as fast as $\Theta(1)$. In my conversion I have an outer for loop iterating over the rows of the matrix, i, and this has a runtime of $\Theta(|V|)$. Inside this for loop is another for loop iterating over the columns of the matrix, j, and has a runtime of $\Theta(|V|)$. Within both these for loops is where I check if an edge(i,j) exists, $\Theta(1)$ and if yes I push it to an array previously declared. While the lookup of edge(i,j) is $\Theta(1)$, the pushing to the array would be dependant on the amount of edges we have or $\Theta(|E|)$. This leaves us with a run time of $\Theta(2|V|) + |E|)$ or $\Theta(|V| + |E|)$.

For a best case scenario we would see a $\Theta(|V|)$ complexity meaning the matrix, and graph have no edges or no vertexes so you check everything once and dont do anything else.

For the average and worst case we would see the $\Theta(|V|^2 + |E|)$ runtime. This is becase for most cases we will have to iterate over everything and return a number of edges, if no edges we have best case. for the worst case we would have a completely full matrix, or a complete graph, which would still only iterate over i and j to have $2|V|$ and push the number of edges $|E|$.

If we flipped the two and now wanted to convert an adjacency list to an adjacency matrix we would see a runtime of $\Theta(|E|^2 + |V|^2)$ this is becase we would have to first iterate over our edges, $\Theta(|E|)$, and then check whether an edge exists while iterating, which gives us $\Theta(|E|)$. Now that we have iterated over our list and have the edges that exist, we need to add the vertexes to our matrix, and then add the edges.

Continued on Back ⟶

Because my original implementation created an empty matrix and added to it for this Hypothetical reversal I will also add entries to an empty matrix. Once we get our edge(i,j) we would have to create the vertex i and create the vertex j, and then add an edge. Now, what if we already have the vertex in our matrix? We would still have to be iterating over the matrix to check so it wouldnt affect our runtime. This addition of vertex i runs in $\Theta(|V|^2)$ becase the iteration and the addition to our matrix. Now we repeat the process for j vertex and get the same $\Theta(|V|^2)$. Luckily, to add an edge in an adjacency matrix can be done in $\Theta(1)$. Therefore, all ~~together~~ we have $\Theta((|E| \cdot |E|) + (2|V|^2) + \Theta(1))$ or $\Theta(|E|^2 + |V|^2)$

For this we would see a best case runtime of $\Theta(|E|^2)$ becaus we have to iterate over all the edges, $\Theta(|E|)$, and then check if the edge exists, $\Theta(|E|)$, but we don't have to do anything with our matrix.

For the average and worst cases we would see the $\Theta(|E|^2 + |V|^2)$ because if any edge exists in the list it is $\Theta(|E|^2 + |V|^2)$ to convert it to the matrix, with a worst case again having a full list or complete graph and having to check every edge and insert every vertex.

For my original implementation it is very dependant on both the verticies and edges because we have to iterate over the verticies and add the edges to our list.