CSC111 Assignment 3: Graphs, Recommender Systems, and Clustering

Jamie Yi

March 24, 2022

Part 1: The book review graph and simple recommendations

1. Complete this part in the provided `a3_part1.py` starter file. Do **not** include your solution in this file.

2. Let m be the number of lines in the `reviews_file` file.
   Let n be the number of lines in the `book_names_file` file.

   The first two assignment statements are both constant time, 1 step.
   `open` and `csv.reader` are both constant time, 1 step.
   The first for loop iterates exactly n times and only contains an assignment statement which is constant time, so the whole loop will take $n(1)$ or just n steps.
   `open` and `csv.reader` are both constant time, 1 step.
   The comprehension will iterate exactly m times, and creating a set and appending it to a list are both constant time, so the comprehension will take $m(1)$ or just m steps.

   The second for loop will iterate exactly m times, and all the statements inside it are constant time, so the whole loop will take $m(1)$ or just m steps.
   `Graph.add_vertex` only has one assignment statement and its if statement condition are both constant time, 1 step.
   `Graph.add_edge` has assignment statements, `set.add`(both of which are constant time) and its if statement condition is both constant time, 1 step.

   The final return statement is constant time, 1 step.

   The function will run $1+1+n+1+m+m+1$ steps per call. Which is equal to $n+2m+4$ Therefore we have a running time of $\Theta(m+n)$

3. Complete this part in the provided `a3_part1.py` starter file. Do **not** include your solution in this file.

4. Complete this part in the provided `a3_part1.py` starter file. Do **not** include your solution in this file.

Part 2: Weighted graphs, recommendations, review prediction

Complete this part in the provided `a3_part2_recommendations.py` and `a3_part2_predictions.py` starter files. Do **not** include your solution in this file.

Part 3: Finding book clusters

1. Complete this part in the provided `a3_part3.py` starter file. Do **not** include your solution in this file.

2. Complete this part in the provided `a3_part3.py` starter file. Do **not** include your solution in this file.

3. (a) Let m_1 be the size of `cluster1`
   Let m_2 be the size of `cluster2`

   The first two assignment statements are both constant time, 1 step.
   The outer for loop will iterate exactly m_1 times.
   The inner for loop will iterate exactly m_2 times, the assignment statement inside the inner for loop takes 1 step constant time, since the `Graph.get_weight` method is constant time (it contains assignment statements and dictionary indexing, both of which are always constant time).
   Therefore the whole loop will take $m_1(m_2(1))$ steps, or just $m_1 m_2$ steps.
   The final return statement is constant time, 1 step.

   The function will run $1 + m_1 m_2 + 1$ steps per call. Which is equal to $m_1 m_2 + 2$ Therefore we have a running time of $\Theta(m_1 m_2)$

   (b) Let n be the number of vertices in `graph`.
   For any fixed iteration of the outer loop, `clusters` will be at most length n, containing the clusters $c_1, c_2, c_3, ..., c_n$.
   Let the notation $size(c_0)$ represent the size of c_0.
   Fix an arbitrary $c1$, such that we can treat $size(c1)$ as a constant.

   The for loop will iterate at most n times.
   For every iteration except when `c1 is c2`, the loop will run $size(c1) \cdot size(c2) + 1$ steps (we know the running time of `cross_cluster_weight` from part a). Since sum of all cluster sizes is equal to n and `c1 is not c2`, then at most $size(c2)$ will equal $(n - s(c1))$.
   When `c1 is c2`, the loop will run 1 step to check the if statement.
   Assuming the loop iterates the maximum amount of n times, it will take at most $1+(n-1)(n-s(c1))$ steps.

   The loop will run at most $1+(n-1)(n-size(c1))$ steps per call at most, which can be rewritten as $n^2 - size(c1)n - n + size(c1)$ steps. Therefore we have a running time of $\mathcal{O}(n^2)$ (since we treat $size(c1)$ as a constant)

   (c) Let n be the number of vertices in `graph`.
   Let k be the is the value of `num_clusters`.

   The first comprehension is creating a set and appending it to a list which is constant time (the method `Graph.get_all_vertices` is also constant time if it has no arguments passed to it). It will iterate exactly n times so the comprehension will take $n(1)$ or just n steps.

   Assuming no clusters are ever merged in the for loop, the outer for loop will iterate at most $(n - k)$ times.
   The first 4 statements (print, `random.choice`, and assignment) are all 1 step constant time.

As shown in part b, the inner for loop will run for at most $n^2 - size(c1)n - n + size(c1)$ steps. The value for any given `c1` is at most $n - size(c1)$ (since there should be at least 2 clusters in `clusters` while the loop is running), so the `set.update` statement will take at most $n - size(c1)$ steps.
`set.remove` is constant time, so it's 1 step constant time.
Therefore, the loop will run $4 + n^2 - size(c1)n - n + size(c1) + (n - size(c1)) + 1$ at most per iteration.

The final return statement is constant time, 1 step.

The function will run $n + (n-k)(4 + n^2 - size(c1)n - n + size(c1) + (n - size(c1)) + 1)$ steps per call at most. Therefore we have a running time of $\mathcal{O}(n^2(n-k))$

(d) *Not to be handed in.*