CSC111 Assignment 3: Graphs, Recommender Systems, and Clustering

Stewart Chandler, Katherine Luo

March 27, 2021

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. Analysis of the `Graph.add_vertex` fucntion.

```
0  |def add_vertex(self, item: Any, kind: str) -> None:
1  |    if item not in self._vertices:
2  |        self._vertices[item] = _Vertex(item, kind)
```

Analysis.
Let $RT_{g.av} \in \mathbb{R}^+$ denote the runtime in steps of `Graph.add_vertex`.

- (Line 1): 1 step for the if statement, dictionary lookup is constant time so 1 step for the in operator. (2 Steps)
- (Line 2): Adding an item to a dictionary is constant time so 1 step. (1 Step)

Totaling up the steps we get $RT_{g.av} = 2 + 1 = 3$. The asymptotic runtime of the `Graph.add_vertex` function is $\Theta(3) = \Theta(1)$.

Analysis of the `Graph.add_edge` fucntion.

```
0  |def add_edge(self, item1: Any, item2: Any) -> None:
1  |    if item1 in self._vertices and item2 in self._vertices:
2  |        v1 = self._vertices[item1]
3  |        v2 = self._vertices[item2]
4  |
5  |        v1.neighbours.add(v2)
6  |        v2.neighbours.add(v1)
7  |    else:
8  |        raise ValueError
```

Analysis.
Let $RT_{g.ae} \in \mathbb{R}^+$ denote the runtime in steps of `Graph.add_edge`.

- (Line 1): 1 step for the if statement, dictionary lookup is constant time so 1 step for each in operator. (3 Steps)
- (Line 2-3): Adding an item to a dictionary is constant time so 1 step for each insertion. (2 Steps)
- (Line 5-6): Adding an item to a set is constant time so 1 step for each insertion. (2 Steps)

- (Line 8): This will not be reached as a result of calling the `load_review_graph` function as it is only called directly after adding the vertices to `self._vertices`, thus we can consider it to be 0 steps. (0 Steps)

Totaling up the steps we get $RT_{g.ae} = 3+2+2+0 = 7$. The asymptotic runtime of the `Graph.add_edge` function is $\Theta(7) = \Theta(1)$.

Analysis of the `load_review_graph` fucntion.

```
0  |def load_review_graph(reviews_file: str, book_names_file: str) -> Graph:
1  |    graph = Graph()
2  |    with open(book_names_file) as book_names_csv:
3  |        names = csv.reader(book_names_csv)
4  |        book_names = {name[0]: name[1] for name in names}
5  |
6  |    with open(reviews_file) as reviews_file_csv:
7  |        reviews = csv.reader(reviews_file_csv)
8  |
9  |        for review in reviews:
10 |            graph.add_vertex(review[0], 'user')
11 |            graph.add_vertex(book_names[review[1]], 'book')
12 |
13 |            graph.add_edge(review[0], book_names[review[1]])
14 |
15 |    return graph
```

Analysis.
Let $m \in \mathbb{Z}^+$ be the number of lines in `reviews_file` and let $n \in \mathbb{Z}^+$ be the number of lines in `book_names_file`.
Let $RT_{lrg} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$ denote the runtime in steps of `load_review_graph` as a function of m and n.

- (Line 1): Initializing an empty graph is constant time. (1 Step)
- (Line 2): Opening a file is constant time. (1 Step)
- (Line 3): Initializing a `csv.reader` object is constant time. (1 Step)
- (Line 4): Comprehension iterating through names and assigning them to values in a dictionary will take m (`len(names)`) steps. (m Steps)
- (Line 6): Opening a file is constant time. (1 Step)
- (Line 7): Initializing a `csv.reader` object is constant time. (1 Step)
- (Lines 9-13): For loop with n (`len(reviews)`) iterations:
  - (Line 10-11): Calls `Graph.add_vertex` for a total of $RT_{g.av} = 3$ steps each line. (6 Steps)
  - (Line 13): Calls `Graph.add_vertex` for a total of $RT_{g.ae} = 7$ steps. (7 Steps)

  Adding it up we get $6 + 7 = 13$ steps eahc iteration.

  This totals to $\sum_{n=1}^{n} 13 = 13n$ steps. ($13n$ Steps)
- (Line 15): Return statement is constant time. (1 Step)

Totaling up the steps we get $RT_{lrg} = 1+1+1+m+1+1+13n+1 = 6+m+13n$. The asymptotic runtime of the `load_review_graph` function is $\Theta(6 + m + 13n) = \Theta(m + n)$.

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. Analysis of the `cross_cluster_weight` function

```
0  |def load_review_graph(reviews_file: str, book_names_file: str) -> Graph:
1  |    total_weight = 0
2  |    for v1 in cluster1:
3  |        for v2 in cluster1:
4  |            total_weight += book_graph.get_weight(v1, v2)
5  |
6  |    return total_weight / (len(cluster1) * len(cluster2))
```

Analysis.
Let $m_1 \in \mathbb{Z}^+$ be the size of `cluster1` and let $m_2 \in \mathbb{Z}^+$ be the size of `cluster2`.
Let $RT_{ccw} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$ denote the runtime in steps of `cross_cluster_weight` as a function of m_1 and m_2.

(Line 1): Assignment of a varible is constant time. (1 Step)

- (Lines 2-4): For loop m_1 (`len(cluster1)`) iterations:
  - (Lines 3-4): For loop m_2 (`len(cluster2)`) iterations:
    - (Line 4): `WeightedGraph.get_weight` is constant time so 1 step, and += is also 1 step. (2 Steps)

    Hence, we have 2 steps per iteration.

    Totaling to $\sum_{n=1}^{m_2} 2 = 2m_2$ steps. ($2m_2$ Steps)

  Thus, there are $2m_2$ steps per iteration.

  This gives us a total of $\sum_{n=1}^{m_1} 2m_2 = 2m_1m_2$ steps for the for loop. ($2m_1m_2$ Steps)
- (Line 6) Return is constant time, 1 step, the len operator takes constant time or 1 step for each use, and the multiplication takes constant time adding another step. (4 Steps)

Totaling up all the steps we get $RT_{ccw} = 1 + 2m_1m_2 + 4 = 2m_1m_2 + 5$. The asymptotic runtime of the `cross_cluster_weight` function is $\Theta(2m_1m_2 + 5) = \Theta(m_1m_2)$.

Analysis of the upper bound of the inner loop of `find_cluster_random`.

(b)

```
0  |for c2 in clusters:
1  |    if c1 is not c2:
2  |        score = cross_cluster_weight(graph, c1, c2)
3  |        if score > best:
4  |            best = score
5  |            best_c2 = c2
```

Analysis.
Let $n \in \mathbb{Z}^+$ be the number of verticies in `graph`.
For all $i \in \mathbb{Z}^+$, let $m_i \in \mathbb{Z}^+$ denote the size of `clusters[i]`.
Let $j \in \mathbb{Z}^+$ denote the size of `c1`.
Let $k \in \mathbb{Z}^+$ be the size of `clusters`.
Let $RT_{il} : \mathbb{N} \to \mathbb{R}^+$ denote the runtime in steps of the inner loop as a function of n.
To find an upper bound on $RT_{il}(n)$.

- (Line 0): For loop k (`len(clusters)`) iterations:
  Let $i \in \mathbb{Z}^+$ be the iteration of k
  - (Line 1): If statement is constant time 1 step, `is not` is constant time 1 step. As we are looking for an upper bound we can assume this if statement will always be true. (2 Steps)
  - (Line 2): $RT_{ccw} \in \Theta(jm_i)$ so we can assume that it will take jm_i steps, plus 1 step for variable assignment. ($jm_i + 1$ Steps)
  - (Line 3): If statement is constant time 1 step, comparison is constant time, 1 step. As we are finding an upper bound we can assume that the if statement will be true. (2 Steps)
  - (Lines 4-5): Variable assignment is constant time so 1 step for each line. (2 Steps)

  Thus, the runtime for an interation of the loop is at most $2 + jm_i + 1 + 2 + 2 = jm_i + 7$ steps.

  Summing it up, the total runtime of the loop is at most $\displaystyle\sum_{i=1}^{k}(jm_i + 7) = \sum_{i=1}^{k} jm_i + 7k = j\sum_{i=1}^{k} m_i + 7k$ steps.

Therefore we can say that $RT_{il}(n) \leq j\displaystyle\sum_{i=1}^{k} m_i + 7k$.
Furthermore, as `clusters` is a set of disjointed subsets of `graph._vertices` that's union is `graph._vertices`, $\displaystyle\sum_{i=1}^{k} m_i = n$, thus, $RT_{il}(n) \leq jn + 7k$.
Additionally, as `c1` is a subset of `clusters`, $j \leq n$, thus, $RT_{il}(n) \leq jn + 7k \leq n^2 + 7k$.
Furthermore, as `clusters` is a partition of `graph._vertices`, we know that $k \leq n$, thus $RT_{il}(n) \leq n^2 + 7k \leq n^2 + 7n$.
Therefore, as $RT_{il}(n) \leq n^2 + 7n$, $RT_{il}(n) \in \mathcal{O}(n^2 + 7n) = \mathcal{O}(n^2)$.

(c) Analysis of the upper bound of `find_cluster_random`.

```
0   |def find_clusters_random(graph: WeightedGraph, num_clusters: int) -> list[set]:
1   |    clusters = [{book} for book in graph.get_all_vertices()]
2   |
3   |    for _ in range(0, len(clusters) - num_clusters):
4   |        print(f'{len(clusters)} clusters')
5   |
6   |        c1 = random.choice(clusters)
7   |        # Pick the best cluster to merge c1 into.
8   |        best = -1
9   |        best_c2 = None
10  |
11  |        for c2 in clusters:
12  |            if c1 is not c2:
13  |                score = cross_cluster_weight(graph, c1, c2)
14  |                if score > best:
15  |                    best = score
16  |                    best_c2 = c2
17  |
18  |        best_c2.update(c1)
19  |        clusters.remove(c1)
```

```
20 |
21 |     return clusters
```

Analysis.
Let $n \in \mathbb{Z}^+$ be the number of verticies in `graph`.
Let $k \in \mathbb{Z}^+$ be the value of `num_clusters`.
Let $RT_{fcr} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$ denote the runtime in steps of `find_cluster_random` as a function of n and k.
To find an upper bound on $RT_{fcr}(n, k)$.

- (Line 1): Comprehension takes n (`len(graph._vertices)`) steps. (n Steps)
- (Lines 3-19): For loop, $n - k$ iterations:
  - (Line 4): `print` function takes constant time, 1 step. `len` takes constant time, 1 step. (2 Steps)
  - (Line 6): `random.choice` takes constant time, 1 step. Variable assignment takes constant time, 1 step. (2 Steps)
  - (Lines 8-9): Variable assignment takes constant time, 1 step for each line. (2 Steps)
  - (Lines 11-16): As we have shown the inner loop is $\mathcal{O}(n^2)$ so we may assume that it takes at most n^2 steps. (n^2 Steps)
  - (Line 18): `set.update` takes `len(c1)` steps, however as `len(c1)`$\leq n$, takes at most n steps. (n Steps)
  - (Line 19): `set.remove` takes constant time, 1 step. (1 Step)

  Thus, for each iteration the for loop takes at most $2+2+2+n^2+n+1 = n^2+n+7$ steps. As such, the loop takes at most $\sum_{i=1}^{n-k}(n^2+n+7) = (n-k)(n^2+n+7)$. ($(n-k)(n^2+n+7)$ Steps)
- (Line 21): Return statment takes constant time 1 step. (1 Step)

Totaling it up we get that `find_cluster_random` takes at most $n + (n-k)(n^2+n+7)+1$ steps.
Therefore we can say that $RT_{fcr}(n, k) \leq n + (n-k)(n^2+n+7)+1$.
As such, $RT_{fcr} \in \mathcal{O}(n + (n-k)(n^2+n+7)+1) = \mathcal{O}(n^2(n-k))$.

(d) *Not to be handed in.*