Master of Data Science Online Programme
Course: Algorithms and Data Structures - Part II
Week #6: Film Recommendation: Algorithm

Student: Andrei Batyrov (Fall 2022)
arbatyrov@edu.hse.ru

Higher School of Economics (HSE)
National Research University
Faculty of Computer Science

April 5, 2023

Contents

List of Figures

Listings

List of Algorithms

Problem

You are tasked with developing a movie recommendation system. You are given a list of movies (their names) and a list of similarities between movies (pairs of movies that are similar). You are also given a list of user's friends and for each friend a list of movies that he has already seen.

Your system should recommend one movie with the highest discussability and uniqueness. Discussability is the number of friends of user, who have already seen that movie. Uniqueness is 1 divided by the mean number of similar movies that the user's friends have already seen. So you should return the film with the highest number: F/S, where F is the number of friends who have seen this movie, and S is the mean of the number of similar movies seen for each friend. Exclude the movies with $S = 0$.

If (a, b) and (b, c) are pairs of similar movies, then (a, c) is a pair of similar movies too. Each movie is not counted in its Uniqueness.

Input example. Basically it is up to you to come up with data structure you like or you think easy to work with. In a nutshell you have as an input these parameters (they can be in form of a list/dict etc):

```
movies = ["Parasite", "1917", "Ford v Ferrari", "Jojo Rabbit", "Joker"]
similarities = [["Parasite", "1917"],
                ["Parasite", "Jojo Rabbit"],
                ["Joker", "Ford v Ferrari"]]
friends = [["Joker"],
           ["Joker","1917"],
           ["Joker"],
           ["Parasite"],
           ["1917"],
           ["Jojo Rabbit", "Joker"]]
```

Listing 1: Input example

Solution

Let $Movs$ be an array of n movies (movie names), $Sims$ be an array of k similarity pairs of movies, and $Frnds$ be an array of l friends, where each friend is an array of seen movies by that friend.

$$\begin{aligned} Movs &= [m_1, \ldots, m_n], n = |Movs|, \\ Sims &= [s_1, \ldots, s_k], k = |Sims|, |s_i| = 2, s_i = [m_i, m_j] \mid m_i, m_j \in Movs \wedge i \neq j, \\ Frnds &= [f_1, \ldots, f_l], l = |Frnds|, f_i = [m_1, \ldots, m_i, \ldots,] \mid m_i \in Movs. \end{aligned} \tag{1}$$

The whole algorithm can be divided into three major steps:

1. Construct Discussability array F:
   - 1.1 Convert arrays $Movs$ and $Frnds$ to hash tables;
   - 1.2 Construct array F using those hash tables;
2. Construct hash table F/S:
   - 2.1 Construct graph $\mathcal{G}$ from array $Sims$ and hash table $Movs$;
   - 2.2 Find connected components of $\mathcal{G}$;
   - 2.3 Construct hash table F/S using array F and finding numerators for array S (reciprocal Uniqueness);
3. Recommend movie:
   - 3.1 Find the key of the first occurrence of $max(F/S)$;
   - 3.2 Return movie at that key as index in array $Movs$.

See the main Algorithm 8. This algorithm is optimized for speed, i.e. indexing, search, and insertion operations are done in $\mathcal{O}(1)$ time, where possible and reasonable. For all algorithms in the solution section, time complexity and auxiliary space are shown under each algorithm. For detailed time and space complexity analysis, please refer to the corresponding sections. Let's consider all steps of this algorithm.

Algorithm 1 Convert array $Movs$ to hash table

```
1: function MovsToHashTable(Movs)
2:     H_Movs ← Empty hash table with keys as movie names and values as indices
3:     n ← |Movs|
4:     for all i ∈ [1, n] do
5:         m_i ← Movs_i
6:         if m_i ∉ H_Movs then
7:             H_Movs(m_i) ← i                ▷ insert key = m_i, value = i into H_Movs
8:         end if
9:     end for
10:    return H_Movs
11: end function
```

Time complexity: $\mathcal{O}(n)$.
Auxiliary space: $\mathcal{O}(|H_{Movs}|)$.

Algorithm 2 Convert array $Frnds$ to hash table

```
1: function FrndsToHashTable(Frnds)
2:     H_Frnds ← Empty hash table with keys as movie names and values as movie counts
3:     for all f_i ∈ Frnds do
4:         for all m_i ∈ f_i do
5:             if m_i ∉ H_Frnds then
6:                 H_Frnds(m_i) ← 1                ▷ insert key = m_i, value = 1 into H_Frnds
7:             else
8:                 H_Frnds(m_i) ← H_Frnds(m_i) + 1   ▷ increase value at key = m_i by 1 in H_Frnds
9:             end if
10:        end for
11:    end for
12:    return H_Frnds
13: end function
```

Time complexity: $\mathcal{O}(l \times \overline{|f_i|})$, where $\overline{|f_i|}$ is the average number of movies watched by all friends.
Auxiliary space: $\mathcal{O}(|H_{Frnds}|)$.

1. Construct Discussability array F. First we need to convert arrays $Movs$ and $Frnd$ into hash table to reduce the search time complexity from $\mathcal{O}(n)$ for arrays to $\mathcal{O}(1)$ for hash tables. See Algorithms 1 and 2. For hash tables the expression $H(key) = val$ means inserting (if absent) or updating the key with val.

   Now, we can construct array F. Each element in F corresponds to a movie in $Movs$ and is the total number of friends who have watched this movie, so the size of F is $|Movs| = n$. Since, we store the movie counts in the friends hash table H_{Frnds}, we simply need to iterate over the keys of the movies hash table H_{Movs} and search for all movies in the friends hash table H_{Frnds} to get corresponding movie counts and construct F. If some movie was not watched by friends, then push zero to F. See Algorithm 3.

2. Construct hash table F/S. In array S each element corresponds to a movie in $Movs$ and is the mean of the number of similar movies seen for each friend. Also, it is important that if (a, b) and (b, c) are pairs of similar movies, then (a, c) is a pair of similar movies too. For describing such relations of similarities, we can construct a graph $\mathcal{G}$, where each vertex corresponds to the index of a movie in $Movs$ and edges connect similar movies. Let the graph be stored as a hash table with keys as movie indices and values as adjacency arrays [3]. To construct such a hash table, we loop through the array $Sims$ and insert $key =$ vertex with $val =$ adjacency array for each vertex. Since each similarity s_i is a pair, we check, if our movie is equal to either the left (first) or the right (second) movie in the pair. If our movie (vertex) is equal to the left (first) movie in the pair, we add the index of the right (second) similar movie to the adjacency array for that vertex. Likewise, if our movie is equal to the right (second) movie in the pair, we add the index of the left (first) similar movie. See Algorithm 4.

   Since the denominator of S is the number of friends l and is the same for all movies, we only need to construct the array of numerators of S for each movie. We might traverse the graph

Algorithm 3 Construct array F

function ConstructF(H_{Movs}, H_{Frnds})
    $F \leftarrow [\,]$
    for all $m_i \in H_{Movs}$ **do**
        if $m_i \in H_{Frnds}$ **then**
            $F \leftarrow$ **push** $H_{Frnds}(m_i)$
        else
            $F \leftarrow$ **push** 0
        end if
    end for
    return F
end function
Time complexity: $\mathcal{O}(n)$.
Auxiliary space: $\mathcal{O}(|F| = n)$.

Algorithm 4 Convert array $Sims$ to hash table

function SimsToHashTable($Sims, H_{Movs}$)
    $\mathcal{G} \leftarrow$ Empty hash table with keys as movie indices (vertices) and values as adjacency arrays (neighbors of vertex)
    for all $s_i \in Sims$ **do**
        if $H_{Movs}(s_{i_1}) \notin \mathcal{G}$ **then** ▷ *checking left (first) movie in similarity pair* s_i
            $\mathcal{G}(H_{Movs}(s_{i_1})) \leftarrow [H_{Movs}(s_{i_2})]$
        else
            $\mathcal{G}(H_{Movs}(s_{i_1})) \leftarrow$ **push** $H_{Movs}(s_{i_2})$
        end if
        if $H_{Movs}(s_{i_2}) \notin \mathcal{G}$ **then** ▷ *checking right (second) movie in similarity pair* s_i
            $\mathcal{G}(H_{Movs}(s_{i_2})) \leftarrow [H_{Movs}(s_{i_1})]$
        else
            $\mathcal{G}(H_{Movs}(s_{i_2})) \leftarrow$ **push** $H_{Movs}(s_{i_1})$
        end if
    end for
    return $\mathcal{G}$
end function
Time complexity: $\mathcal{O}(2 \times |Sims| = 2k)$.
Auxiliary space: $\mathcal{O}(|\mathcal{G}|)$.

$\mathcal{G}$ starting from an arbitrary vertex (movie) v and recursively calculate SN_v for vertex v:

$$SN_v = F_{u_i} + SN_{u_i}, \tag{2}$$

where u_i is the i-th neighbor of v. However, this approach is not optimal, since we would traverse the same graph for $V(\mathcal{G})$ times, each time starting from the next vertex and repeatedly accessing the same values F_{u_i} to find SN_v. It is more reasonable to find SN not for each vertex, but for each connected component, since it is easy to see that

$$SN_{cc} = \big(\sum_i F_{u_i}\big), \tag{3}$$

which is calculated only once for each connected component. Since we know nothing about the structure of $\mathcal{G}$: deep or wide, we can use either Depth-First Search (DFS) or Breadth-First Search (BFS) algorithm for graph traversal. Both algorithms have the same time and space complexity, but DFS has a simple recursive implementation, so let's choose DFS [1], [2] to find connected components: array of size equal to the number of found connected components, where each element is again an array of vertices belonging to that connected component. See Algorithm 5.

Next we can populate our hash table of connected components by calling 5 for each vertex in $\mathcal{G}$. See Algorithm 6.

Now, we're ready to construct the array of ratios F_v/S_v for each vertex (movie). For that, first we iterate over all found connected components to find its $\big(\sum_i F_{u_i}\big)$, and then we iterate

Algorithm 5 Find vertices of a connected component of $\mathcal{G}$

function FINDCCVERTICES($\mathcal{G}, V_{cc}, v, H_{visited}$)
  $H_{visited}(v) \leftarrow true$
  $V_{cc} \leftarrow$ **push** v
  for all $u_i \in \mathcal{G}(v)$ **do** ▷ *for all neighbours of* v
    if $u_i \notin H_{visited}$ **then**
      $V_{cc} \leftarrow$ FINDCCVERTICES($\mathcal{G}, V_{cc}, v, H_{visited}$)
    end if
  end for
  return V_{cc} ▷ *array of all vertices in one connected component*
end function
Time complexity: $\mathcal{O}(|V(\mathcal{G})| + |E(\mathcal{G})|)$.
Auxiliary space: $\mathcal{O}(|H_{visited}| = 2|V(\mathcal{G})| = 2n) + \mathcal{O}(|V_{cc}|)$.

Algorithm 6 Construct hash table of connected component of $\mathcal{G}$

function CONSTRUCTCC($\mathcal{G}, H_{visited}$)
  $CC \leftarrow$ Empty hash table with keys as the starting vertices of each component (kind of ids of components) and values as arrays of all vertices in that component
  for all $v_i \in V(\mathcal{G})$ **do**
    if $v_i \notin H_{visited}$ **then**
      $V_{cc} \leftarrow [\,]$
      $CC(v_i) \leftarrow$ FINDCCVERTICES($\mathcal{G}, V_{cc}, v_i, H_{visited}$)
    end if
  end for
  return CC ▷ *array of all found connected components*
end function
Time complexity: $\mathcal{O}(|V(\mathcal{G})| + |E(\mathcal{G})|)$.
Auxiliary space: $\mathcal{O}(|CC|)$.

over all found connected components once again to eventually find F/S. Since each movie is not counted in its Uniqueness, we need to exclude its F_v value from the sum SN_{cc}. Also, if none of the friends watched movie(s) in a connected component, we should exclude, or better mark such movie(s) by -1, for example.

$$\frac{F_v}{S_v} = \begin{cases} \frac{F_v \times l}{SN_{cc} - F_v}, & SN_{cc} \neq F_v, \\ -1, & SN_{cc} = F_v. \end{cases} \tag{4}$$

See Algorithm 7.

3. Recommend movie. Finally, we need to return the movie with highest ratio F/S. For that, we first find the first maximal value in the hash table F/S, then search in F/S for that maximal value to get its key, and then return the element of the array $Movs$ at that index. If the search returns more than one element with the maximal value in F/S, use the key/index of the first found element. See Algorithm 8.

Proof of Solution

Let's solve the problem, i.e. recommend the movie for the given sample input 1 using the main algorithm 8. The arrays $Movs, Sims, Frnds$ are as follows.

$$\begin{aligned} Movs &= [\text{Parasite}, 1917, \text{Ford v Ferrari}, \text{Jojo Rabbit}, \text{Joker}], \\ Sims &= [[\text{Parasite}, 1917], [\text{Parasite}, \text{Jojo Rabbit}], [\text{Joker}, \text{Ford v Ferrari}]], \\ Frnds &= [[\text{Joker}], [\text{Joker}, 1917], [\text{Joker}], [\text{Parasite}], [1917], [\text{Jojo Rabbit}, \text{Joker}]]. \end{aligned} \tag{5}$$

Algorithm 7 Construct hash table F/S

```
function ConstructFS(G, F, l)
    H_visited ← Empty hash table with keys as vertices and Boolean values
    CC ← ConstructCC(G, H_visited)
    FS ← Empty hash table with keys as vertices and values as ratios F_v/S_v
    SN_cc ← Empty hash table with keys as starting vertices (kind of ids of components) and
    values as sums of F_{u_i}, as per (3)
    for all cc_i ∈ CC do
        sn ← 0
        for all v_j ∈ CC(cc_i) do
            sn ← sn + F_{v_j}
        end for
        SN_cc(cc_i) ← sn
    end for
    for all cc_i ∈ CC do
        for all v_j ∈ CC(cc_i) do
            if SN_cc(cc_i) ≠ F_{v_j} then
                FS(v_j) ← F_{v_j} × l/(SN_cc(cc_i) − F_{v_j})
            else
                FS(v_j) ← −1
            end if
        end for
    end for
    return FS
end function
```

Time complexity: $\mathcal{O}(|V(\mathcal{G})| + |E(\mathcal{G})|)$.
Auxiliary space: $\mathcal{O}(|FS|) + \mathcal{O}(|SN_{cc}|)$.

1. Construct a hash table H_{Movs} from array $Movs$:

$$\begin{aligned} H_{Movs} = \{&\text{Parasite} : 1,\\ &1917 : 2,\\ &\text{Ford v Ferrari} : 3,\\ &\text{Jojo Rabbit} : 4,\\ &\text{Joker} : 5\}. \end{aligned} \tag{6}$$

   For example, the movie Parasite's index is 1, as per (5), so $H_{Movs}(Parasite) = 1$.

2. Construct a hash table H_{Frnds} from array $Frnds$:

$$\begin{aligned} H_{Frnds} = \{&\text{Joker} : 4,\\ &1917 : 2,\\ &\text{Parasite} : 1,\\ &\text{Jojo Rabbit} : 1\}. \end{aligned} \tag{7}$$

   For example. the movie 1917 was watched 2 times in total by all friends, as per (5), so $H_{Frnds}(1917) = 2$.

3. Construct array F. Search each movie (6) in (7), and push the found value (count) into F. Push 0 for movies not found in (7).

$$F = [1, 2, 0, 1, 4]. \tag{8}$$

   For example, the movie Joker has count 4, as per (7), and it's index is 5, as per (6), so $F_5 = 4$. At the same time, the movie Ford v Ferrari has count 0, since it is not found in (7), and it's index is 3, as per (6), so $F_3 = 0$.

4. Construct a graph (hash table) for $Sims$. As per (5), we have 3 similarity pairs, which after converting to indices, as per (6), will look as follows:

$$\begin{aligned} Sims = [&[1, 2],\\ &[1, 4],\\ &[5, 3]]. \end{aligned} \tag{9}$$

Algorithm 8 Recommend movie main algorithm

function RECOMMENDMOVIE($Movs, Sims, Frnds$)
  $H_{Movs} \leftarrow$ MOVSTOHASHTABLE(($Movs$)
  $H_{Frnds} \leftarrow$ FRNDSTOHASHTABLE($Frnds$)
  $F \leftarrow$ CONSTRUCTF(H_{Movs}, H_{Frnds})
  $\mathcal{G} \leftarrow$ SIMSTOHASHTABEL($Sims, H_{Movs}$)
  $l \leftarrow |Frnds|$
  $FS \leftarrow$ CONSTRUCTFS($\mathcal{G}, F, l$)
  $best \leftarrow \text{argmax}_{m_i \in Movs} FS$
  return $Movs_{best}$
end function
Time complexity: $\sum_i \mathcal{O}_{A_i} \sim max(\mathcal{O}_{A_i})$, where $A_i \in \{$ algorithms 1, 2, 3, 4, 7 $\}$.
Auxiliary space: $\mathcal{O}(1)$.

As per the Algorithm 4, we take each similarity pair s_i in (9) and insert (if absent) or update the key s_{i_1} with the value s_{i_2} and do the same for the key s_{i_2} with the value s_{i_1}, thus creating adjacency arrays for all movies in $Sims$. After this process, $Sims$ (9) will be represented by a graph (hash table):

$$\begin{aligned} \mathcal{G} = \{&1 : [2, 4], \\ &2 : [1], \\ &4 : [1], \\ &5 : [3], \\ &3 : [5]\}. \end{aligned} \tag{10}$$

5. Number of friends $l = |Frnds| = 6$.

6. Construct hash table F/S. We traverse the graph (10) for each vertex once to find all connected components with their vertices: $CC = \{1 : [1, 2, 4], 5 : [5, 3]\}$, where keys are the starting vertices of each component (kind of ids of components). Here we have two connected components: the first has the vertices $V_{cc} = [1, 2, 4]$ and the second has the vertices $V_{cc} = [5, 3]$. Then we find the array of numerators of S (sums of F_{u_i}), as per (3) for each of the connected components: $SN_{cc} = \{1 : 4, 5 : 4]\}$, where keys, as for CC, are the starting vertices of each component (kind of ids of components). In this case for both connected components numerators of S are equal to 4. Finally, we can find values F_v/S_v for each vertex, as per (4), inserting all such ratios to one hash table F/S.

$$\begin{aligned} &\text{first connected component:} \\ &F_1/S_1 = F_1 \times l/(SN_1 - F_1) = 1 \times 6/(4-1) = 2, \\ &F_2/S_2 = F_2 \times l/(SN_1 - F_2) = 2 \times 6/(4-2) = 6, \\ &F_4/S_4 = F_4 \times l/(SN_2 - F_4) = 1 \times 6/(4-1) = 2, \Rightarrow \\ &\text{second connected component:} \\ &F_5/S_5 = F_5 \times l/(SN_2 - F_5) = 4 \times 6/(4-4) = \infty \Rightarrow -1, \\ &F_3/S_3 = F_3 \times l/(SN_1 - F_3) = 0 \times 6/(4-0) = 0. \end{aligned} \qquad \begin{aligned} F/S = \{&1 : 2, \\ &2 : 6, \\ &4 : 2, \\ &5 : -1, \\ &3 : 0\}. \end{aligned} \tag{11}$$

7. Find $\text{argmax}_{m_i \in Movs} FS$, i.e. search for the maximal element in (11) and obtain its key. It is clearly seen that the maximal element is 6 at key 2, so $\text{argmax}_{m_i \in Movs} FS = 2$. As per (3), $Movs_2 =$ 1917. So, the algorithm recommends the movie **1917**.

Time Complexity Analysis

Let's analyze time complexity of the whole algorithm 8. Since the main algorithm calls all functions in sequence, the overall time complexity is the sum of time complexities of each function. So, let's analyze each function.

1. Convert array $Movs$ to hash table 1. Since it iterates over all movies in $Movs$ and all other operations including hash table search and insertion are $\mathcal{O}(1)$, the overall time complexity is $\mathcal{O}(|Movs|) = \mathcal{O}(n)$.

2. Convert array $Frnds$ to hash table 2. This algorithm has two iterations: one over all friends in $Frnds$ and another over all movies for each friend. All other operations including hash table search and insertion are $\mathcal{O}(1)$. So, the overall time complexity is $\mathcal{O}(|Frnds| = l) \times \mathcal{O}(\overline{|f_i|}) = \mathcal{O}(l \times \overline{|f_i|})$, where $\overline{|f_i|}$ is the average number of movies watched by all friends. Also, we can express its time complexity as $\mathcal{O}(\sum_i F_i)$, i.e. it is bounded by the total number of movies in $Frnds$. In the best case scenario, each friend has watched just one movie: $\mathcal{O}(l)$. In the worst case, each friend has watched all n movies: $\mathcal{O}(l \times n) \sim \mathcal{O}(n^2)$.

3. Construct array F 3. Since it iterates over all movies in $Movs$ and all other operations including hash table search and array push are $\mathcal{O}(1)$, the overall time complexity is $\mathcal{O}(|Movs|)) = \mathcal{O}(n)$.

4. Convert array $Sims$ to hash table 4. Since it iterates over all similarity pairs in $Sims$ and all other operations including hash table search and insertion, and array push are $\mathcal{O}(1)$, the overall time complexity is $\mathcal{O}(2 \times |Sims|)) = \mathcal{O}(2k) \sim \mathcal{O}(k)$. The coefficient 2 is taken because for each s_i we repeat operations two times for each movie in the similarity pair. For trees $k < n$, so $\mathcal{O}(k) < \mathcal{O}(n)$. For complete graphs, the number of similarity pairs $k = \binom{n}{2} = n(n-1)/2$ which means that the algorithm for converting $Sims$ into $\mathcal{G}$ will be bounded by $\mathcal{O}(n(n-1)/2) \sim \mathcal{O}(n^2)$.

5. Find vertices of a connected component of $\mathcal{G}$ 5. This is the classical recursive realization of Depth-First Search algorithm, where each vertex is visited only once and processed in $\mathcal{O}(1)$ time. Its time complexity is $\mathcal{O}(|V(\mathcal{G})_{cc}| + |E(\mathcal{G})_{cc}|)$ for each connected component, where $|V_{cc}|, |E_{cc}|$ are the number of vertices and edges in each connected component of $\mathcal{G}$. To guarantee $\mathcal{O}(1)$ time of vertex processing, the operations with the sequence which stores the visited vertices should be bounded by $\mathcal{O}(1)$ time, so a hash table $H_{visited}$ is used to to keep track of the visited vertices. Array push operations are also $\mathcal{O}(1)$. Parameters V and E are determined by the structure of the similarities graph $\mathcal{G}$. See next algorithm for a more detailed analysis.

6. Construct hash table of connected component of $\mathcal{G}$ 6. Since this algorithm needs to find all connected components of the graph ("split" the graph), it internally calls the graph traversal DFS algorithm 5 for all not visited vertices and does it as many times as the number of connected components there are in the graph. All other operations including hash table insertions are $\mathcal{O}(1)$. So, the overall time complexity is $\mathcal{O}(\sum_{i=1}^{n_{cc}}(|V_i| + |E_i|) = |V| + |E|)$, where n_{cc} is the number of found connected components and $max(n_{cc}) = |V(\mathcal{G})|/2$, since at least two movies must be in any one component to form a similarity pair. Let's consider several cases:

   6.1 Connected component(s) of $\mathcal{G}$ are tree(s). In this case, $|E| = |V| - 1$. The time complexity is bounded by

   $$\mathcal{O}(|V| + |E|) = \mathcal{O}(n + n - 1) = \mathcal{O}(2n - 1) \sim \mathcal{O}(n). \tag{12}$$

   This is the theoretical minimum boundary of the algorithm.

   6.2 $\mathcal{G}$ is a complete graph with $|V|$ vertices and $\binom{|V|}{2}$ edges. The time complexity is bounded by

   $$\mathcal{O}(|V| + |E|) = \mathcal{O}\left(n + \frac{n(n-1)}{2}\right) = \mathcal{O}\left(\frac{n^2}{2} + \frac{n}{2}\right) \sim \mathcal{O}(n^2). \tag{13}$$

   This scenario is not expected and should be avoided. However, it is possible that some of connected components in real $\mathcal{G}$ may have cycles due to errors in array $Sims$ or other reasons, so such connected components may be quasi-complete graphs. This is the theoretical maximum boundary of the algorithm.

   So, we can expect that constructing hash table F/S is bounded by

   $$\mathcal{O}(n^\alpha), \alpha = f(\mathcal{G}) \in [1, 2]. \tag{14}$$

7. Construct hash table F/S 7. The algorithm has two nested loops called one after another. In each nested loop, the outer loop iterates over all connected components n_{cc}, and the internal loop iterates over all vertices V_{cc} in each component. So, the total number of operations is $\sum_{i=1}^{n_{cc}}(|V_i| + |E_i|) = |V| + |E|$, i.e. is bounded by $\mathcal{O}(|V| + |E|) \sim \mathcal{O}(n^\alpha)$, as per (14). All other operations including hash table insertions and array indexing are $\mathcal{O}(1)$.

8. Final operations. Finding maximal value in hash table F/S is $\mathcal{O}(n)$, since it requires at worse case n iterations (max element is the last one). We can use the max binary heap data structure for F/S to get max in $\mathcal{O}(1)$ time, but it is not necessary, since it is already clear that the overall time complexity is greater than $\mathcal{O}(1)$. Indexing in hash table F/S is performed in $\mathcal{O}(1)$ time. Finally, indexing array $Movs$ is done in $\mathcal{O}(1)$. So, the overall time complexity of these final operations is bounded by $\mathcal{O}(n)$.

Now it is clear that the whole main algorithm is bounded by the slowest parts with the worst time $\mathcal{O}(n^2)$. So, we should expect the overall time complexity for recommending one movie out of n movies with k similarity pairs for a user having l friends to be bounded by

$$\mathcal{O}(n^\alpha), \alpha = f(\mathcal{G}, l) \in [1, 2]. \tag{15}$$

See the plots of running time of the main algorithm versus the number of movies for the best, average, and worst cases in Figures 1, 2, and 3.

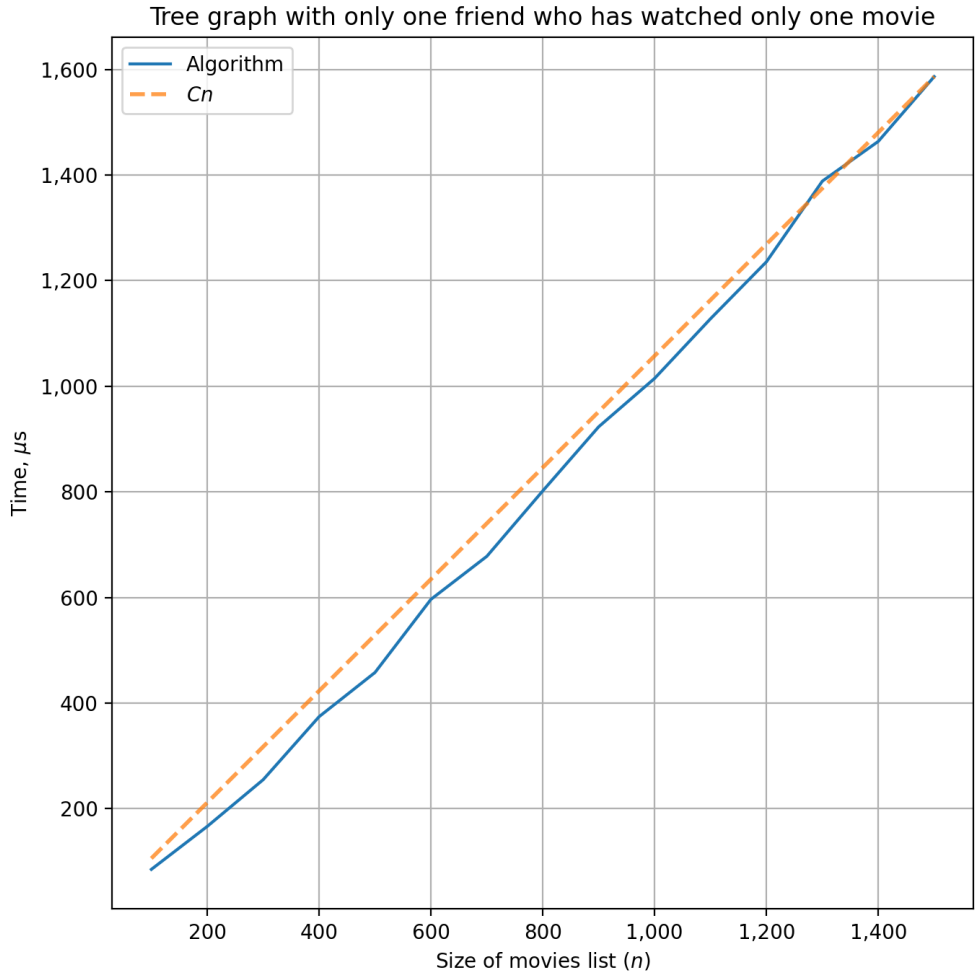

Figure 1: Main algorithm running time: best case

Space Complexity Analysis

As with time complexity, let's first analyze auxiliary space for each function, and then estimate the total space complexity.

1. Convert array $Movs$ to hash table 1. The function creates a new hash table with $|Movs| = n$ keys (movie names), each storing one value (index), so the total auxiliary space is $\mathcal{O}(n + n = 2n) \sim \mathcal{O}(n)$.

2. Convert array $Frnds$ to hash table 2. The function creates a new hash table with $|Movs| = n$ or fewer keys (movie names), each storing one value (count), so the total auxiliary space is $\mathcal{O}(n + n = 2n) \sim \mathcal{O}(n)$.

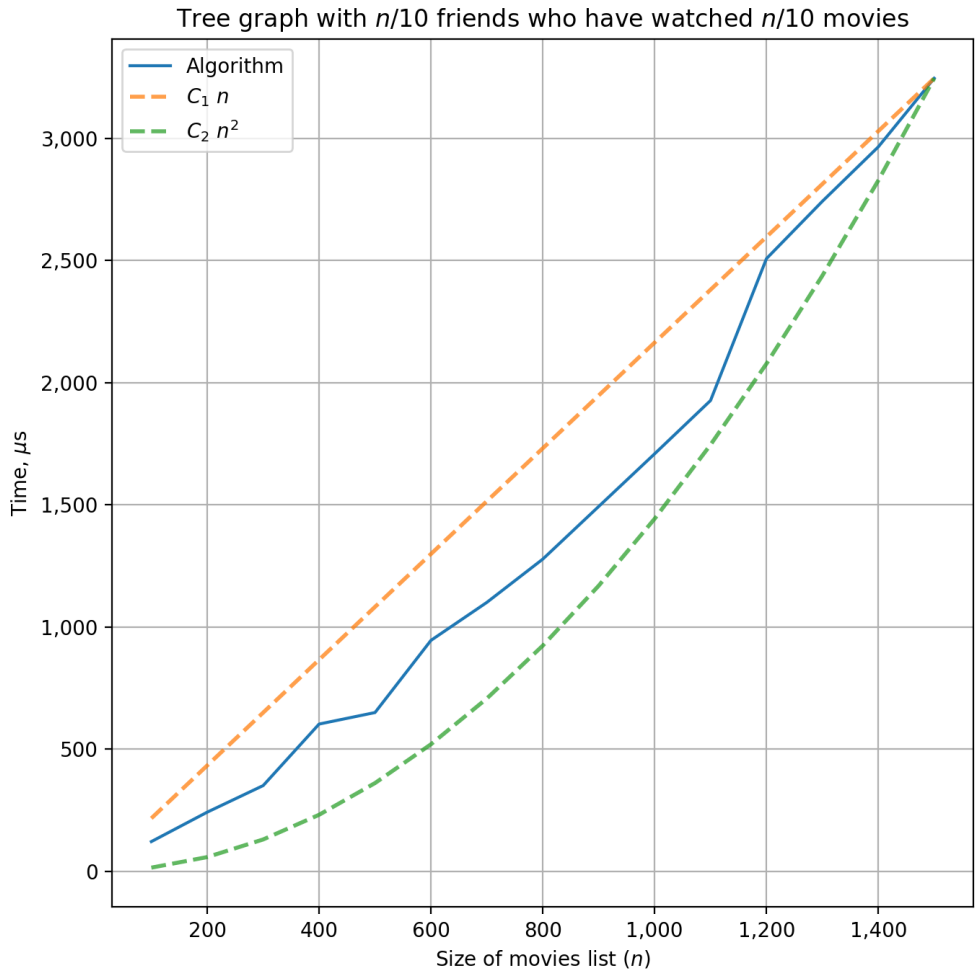

Figure 2: Main algorithm running time: average case

3. Construct array F 3. The function creates a new array with with $|F| = |Movs| = n$ elements, so the total auxiliary space is $\mathcal{O}(n)$.

4. Convert array *Sims* to hash table 4. The function creates a new hash table with keys as vertices (indices), each storing an adjacency array of neighboring vertices (indices). To be able to traverse the graph in any direction starting from any vertex, we need to store a pair of outgoing and incoming edges, i.e. the total size of graph is

$$|\mathcal{G}| = |V| + 2|E|. \tag{16}$$

   4.1 Tree case. Let n_{cc} be the number of connected components in $\mathcal{G}$. Then the total size of the graph is

$$|\mathcal{G}| = n_{cc}\left(\frac{|V|}{n_{cc}} + 2\left(\frac{|V|}{n_{cc}} - 1\right)\right) = 3|V| - 2n_{cc}. \tag{17}$$

   For one connected component the auxiliary space is $\mathcal{O}(3n-2) \sim \mathcal{O}(n)$. For $n/2$, i.e. the maximal possible number of connected components, the auxiliary space is $\mathcal{O}(2n) \sim \mathcal{O}(n)$.

   4.2 Complete graph case. $|E| = \binom{|V|}{2}$. Then the total size of the graph is

$$|\mathcal{G}| = n + 2\left(\frac{n(n-1)}{2}\right) = n^2. \tag{18}$$

   And the auxiliary space is thus $\mathcal{O}(n^2)$.

5. Find vertices of a connected component of $\mathcal{G}$ 5. The algorithm populates the hash table $H_{visited}$ of size $2n$ and also creates an array of all vertices in a given connected component V_{cc}. If $n_{cc} = 1$, then $|V_{cc}| = |V| = n$. If $n_{cc} > 1$, then $|V_{cc}| < |V| = n$. So, the total auxiliary space is $\mathcal{O}(2n) + \mathcal{O}(n) \sim \mathcal{O}(n)$.

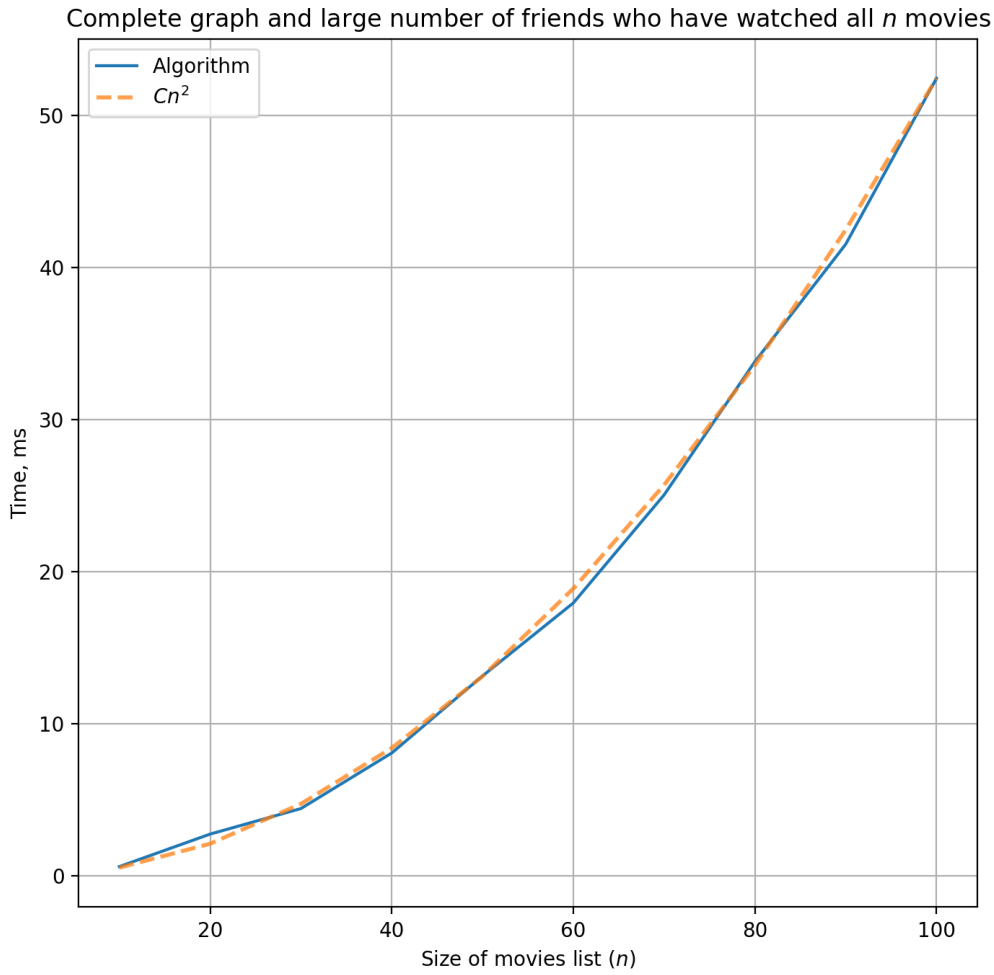

Figure 3: Main algorithm running time: worst case

6. Construct hash table of connected component of $\mathcal{G}$ 6. The algorithm creates hash table CC the number of keys equal to n_{cc}, each key storing all vertices grouped by connected components they belong to. So its size is n_{cc} for keys and $|V|$ for values, and the auxiliary space is $\mathcal{O}(n_{cc} \leq n/2) + \mathcal{O}(|V| = n) \sim \mathcal{O}(n)$.

7. Construct hash table F/S 7. The algorithm creates two hash tables: FS and SN_{cc}. FS stores values F_v/S_v, so its size is $2n$. SN_{cc} stores sums (3) for each connected component, so its size is n_{cc}, and $max(n_{cc}) = |V(\mathcal{G})|/2$, as discussed above. So, the auxiliary space is $\mathcal{O}(2n + n/2) \sim \mathcal{O}(n)$.

8. Final operations. The main algorithm creates only two single-value variables l and $best$, so its auxiliary space is $\mathcal{O}(1)$.

9. The total space complexity is the sum of the size of the input data and auxiliary space of each function. The input array of movies has size $|Movs| = n$, i.e. $\mathcal{O}(n)$. The input array of friends has size $|Frnds| = l$ times the average number of movies seen by all friends, which is in the range $[1, n]$, so the space complexity is from $\mathcal{O}(l) \sim \mathcal{O}(n)$ to $\mathcal{O}(l \times n) \sim \mathcal{O}(n^2)$. The input array of similarities depends on relations between movies, and in the best case require to store $2k < n$ elements, and in the worst case may require to store $n(n-1)/2$ elements, as discussed above for graph $\mathcal{G}$, so the space complexity is from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$.

   As we have shown above, the auxiliary space required for all functions is from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$. Thus we should expect the overall space complexity for recommending one movie out of n movies with k similarity pairs for a user having l friends to be bounded by

$$\mathcal{O}(n^\alpha), \alpha = f(\mathcal{G}, l) \in [1, 2]. \tag{19}$$

References

[1] Andrey Kharatyan. *Depth First Search algorithm*. Faculty of Computer Science, Higher School of Economics. URL: https://smartedu.hse.ru/mod/page/0/697722.

[2] Andrey Kharatyan. *DFS implementation*. Faculty of Computer Science, Higher School of Economics. URL: https://smartedu.hse.ru/mod/page/0/697723.

[3] Andrey Kharatyan. *Graph representations*. Faculty of Computer Science, Higher School of Economics. URL: https://smartedu.hse.ru/mod/page/0/697718.