AAD Tutorial 03/09/2025

Greedy Algorithms, MSTs

Problem-1

- In a movie festival, **n** movies will be shown.
- There are **k** members in our movie club, who will be all attending the festival.
- You know the starting and ending time of each movie.
- What is the maximum total number of movies the club members can watch entirely if they act optimally?
- Input
  - The first input line has two integers **n** and **k**: the number of movies and club members.
  - After this, there are n lines that describe the movies. Each line has two integers **a** and **b**: the starting and ending time of a movie.
- Output
  - One integer: the maximum total number of movies that the club members can watch entirely.

Solution

1. **Sort the Movies**: Read all n movies and store them as pairs of (start time, end time). Sort this list of movies in **ascending order based on their end times**.
2. **Initialize**:
    - Create a counter for the total movies watched, `watched_count`, and set it to 0.
    - Create a multiset, let's call it `busy_members_end_times`, to store the times when the currently busy members will become free. A multiset is used because it keeps elements sorted and allows for efficient searching and removal.
3. **Iterate and Assign**: Go through each movie `(start_time, end_time)` from the sorted list. For each movie, we need to find a member who is free. A member is free if their last movie ended *before* the current movie's `start_time`.
    - We search in our `busy_members_end_times` multiset for the end time `t` that is largest but still less than the current movie's `start_time`. This corresponds to finding the member who finished most recently but is still available.

- **If we find such a member**:
  - This means a member can watch the current movie.
  - We assign the movie to them. To do this, we **remove their old end time** from the multiset and **insert the end time of the new movie**.
  - Increment `watched_count`.
- **If we don't find such a member, BUT we still have fewer than k members busy**:
  - This means one of the initially free members can watch the movie.
  - We assign the movie to a free member. We **insert the movie's end time** into the multiset.
  - Increment `watched_count`.
- If all k members are busy and none of them are free before the current movie starts, we must skip this movie.

Proof of Correctness

- Let $M = \{m_1, m_2, \ldots, m_n\}$ be the set of all available movies, sorted by their **end times** in non-decreasing order. So, for any $i < j$, $end(m_i) \leq end(m_j)$.
- Let $G = \{g_1, g_2, \ldots, g_p\}$ be the set of p movies chosen by our **greedy algorithm**, also sorted by end time. By the nature of the algorithm, the movies in G are selected in this order from the sorted list M.
- Let $O = \{o_1, o_2, \ldots, o_q\}$ be the set of q movies in an **arbitrary optimal solution**, also sorted by end time. By definition of optimal, q is the maximum possible number of movies, so $|G| \leq |O|$, or $p \leq q$.

Our goal is to prove that $p = q$. To do this, we will prove that $p \geq q$. Combined with $p \leq q$, this will establish that $p = q$.

Lemma: For every index i from 1 to p, the end time of the i-th movie chosen by the greedy algorithm is less than or equal to the end time of the i-th movie in any optimal solution. That is, $end(g_i) \leq end(o_i)$ for all $i \in \{1, \ldots, p\}$.

Proof of Lemma (by Induction):

Base Case (i=1):

The greedy algorithm considers all movies in order of their end times. It selects the very first movie in this sorted list, m_1, because with $k \geq 1$ members, at least one is always free at the beginning. Thus, $g_1 = m_1$.

The optimal solution O selects some movie o_1. Since m_1 is the movie with the universally earliest end time, it must be that $end(m_1) \leq end(o_1)$. Therefore, $end(g_1) \leq end(o_1)$. The base case holds.

Inductive Hypothesis:

Assume that for some $i < p$, the lemma holds for all preceding indices. That is, assume $end(g_j) \leq end(o_j)$ for all $j \leq i$.

Inductive Step:

We need to prove that $end(g_{i+1}) \leq end(o_{i+1})$.

1. Let's consider the state after the first i movies, $o_1, \ldots, o_i$, have been scheduled in the optimal solution O. These i movies are assigned to some subset of the k members. The times at which these members become free are $\{end(o_1), \ldots, end(o_i)\}$ (assuming for simplicity one movie per member so far; if a member watches multiple, it's their latest end time that counts).
2. Similarly, after the first i movies, $g_1, \ldots, g_i$, are scheduled by the greedy algorithm G, the members become free at times $\{end(g_1), \ldots, end(g_i)\}$.
3. From our inductive hypothesis, we know $end(g_j) \leq end(o_j)$ for all $j \leq i$. This means that for every member in the optimal schedule, there is a corresponding member in the greedy schedule who becomes free **at or before** them. Therefore, the set of available members for the greedy algorithm is in a "better" or at least equal state to the optimal algorithm's set of members. Any movie that is schedulable by the optimal schedule's members at this point is also schedulable by the greedy schedule's members.

4. The optimal solution now schedules its next movie, o_{i+1}. This movie must be compatible with the schedule so far; i.e., there must be some member who finished their previous movie o_j (where $j \leq i$) such that $end(o_j) \leq start(o_{i+1})$.
5. As established in point 3, the greedy algorithm also has a member free at or before $end(o_j)$, so the movie o_{i+1} is a **valid candidate** for the greedy algorithm to pick as its $(i+1)$-th movie.
6. The greedy algorithm, by its definition, scans through the master list M and picks the first valid movie it finds. Let this movie be g_{i+1}. Since o_{i+1} is a valid candidate, and g_{i+1} is the *first* one the algorithm picks from the list sorted by end times, it must be that $end(g_{i+1}) \leq end(o_{i+1})$.

This completes the inductive step. Thus, we have proven that $end(g_i) \leq end(o_i)$ for all $i = 1, \ldots, p$.

Proof (by Contradiction):

Assume for the sake of contradiction that the greedy algorithm is not optimal. This would mean $p < q$.
If $p < q$, the optimal solution O contains at least one more movie than the greedy solution G.
Let's consider the $(p+1)$-th movie in the optimal solution, o_{p+1}.

1. To schedule o_{p+1}, the optimal solution must assign it to a member who is free at or before $start(o_{p+1})$. This implies that there is some movie o_j in the set $\{o_1, \ldots, o_p\}$ such that $end(o_j) \leq start(o_{p+1})$ (or the member was free from the start).
2. From our lemma, we know that $end(g_j) \leq end(o_j)$.
3. Combining these two inequalities, we get: $end(g_j) \leq end(o_j) \leq start(o_{p+1})$.

4. This result, $end(g_j) \leq start(o_{p+1})$, shows that after scheduling its first p movies, the greedy algorithm had a member available to watch movie o_{p+1}.
5. The greedy algorithm only terminates at p movies if, after scheduling g_p, it scans all remaining movies in M (which would include o_{p+1}) and finds that none of them can be scheduled.
6. But we just proved in step 4 that movie o_{p+1} *was* a valid candidate to be scheduled by the greedy algorithm. This is a **contradiction**. The algorithm would have scheduled either o_{p+1} or another valid movie with an even earlier end time, resulting in a schedule of size at least $p + 1$.

Since our assumption ($p < q$) leads to a contradiction, it must be false. Therefore, $p \geq q$.
Given that O is an optimal solution, we know $p \leq q$.
The only way for both $p \leq q$ and $p \geq q$ to be true is if $p = q$.

Time and Space Complexity

- **Time Complexity:** $O(n \log n)$.
  - Sorting the n movies takes $O(n \log n)$.
  - Iterating through each of the n movies involves operations on the multiset (insertion, deletion, search). Since the multiset size is at most k, these operations take $O(\log k)$ time.
  - The total time is $O(n \log n + n \log k)$. Since $k \leq n$, this simplifies to $O(n \log n)$. This is very efficient and will pass within the time limits.
- **Space Complexity:** $O(n)$.
  - We need to store the n movies, which takes $O(n)$ space. The multiset stores at most k elements, taking $O(k)$ space. The total is $O(n + k)$, which simplifies to $O(n)$.