**`תיאור מבני הנתונים:**

מבני הנתנוים שלנו מורכב מ 8 עצי AVL שכל אחד מהם אחראי על תפקיד מסויים:

1. All Movies ID: העץ הזה מכיל בתוכות את כל הסרטים במערכת שלנו, והולכים להיות מסודרים לפי מספר ה ID של כל סרט (העץ הזה הולך להכיל את כל הסרטים מכל ה Genres).
2. All Movies Rating: העץ הזה מכיל גם את כל הסרטים במערכת שלנו אבל דרך הסידור היא לפי הדירוג של העץ (ובמקרים שיש את איתו דירוג סידרנו לפי ההנחיות למצב הזה).
3. Comedy/Drama/Action/Fantasy: ארבעת העצים הם כמו העץ הקודם אבל שכל עץ יכיל רק את הסרטים מה Genre המתאים והם מסודרים לפי הדירוג שלהם.
4. All Users: העץ הזה מכיל את כל המשתמשים במערכת שלנו והם מדורגים לפי ה ID שלהם.
5. All Groups: העץ הזה מכיל את כל הקבוצות של משתמשים במערכת שלנו, והקבוצו מדורגות לפי ה ID שלהן.

העצים האלו מבוססים על AVLNodes שבנינו, הנודים שלנו הם גינירים (בנויים עם Templates ) על מנת שיתאימו להרבה סוגים של מידע, הם מקבלים שני דברים:

Key – שהוא אחראי על סידור הנודים בתוך העץ.

Data- שהוא מכיל את המידע בתוך הנוד.

**תיאור לדברים כללים:**

**Node Class:**

הנודים הוא הם האיברים שהולכים לאחסן בתוך העץ שלנו, לכל נוד יש לו 8 משתנים:

מידע מסוג DATA , מפתח מסוג KEY, סוג הנוד (leaf, has two sons, only left, only right) , מצביע להאב שלו ומצבעים לבנים שלו, הגובה שלו וה BalanceFactor. יש לנו פונקציה בשם updateParameters שהיא אחרראית על עידכון ה סוג של הנוד ואת גובהו וה BalanceFactor שלו, שאר המימושים הם טריוויאלים.

**Tree Class:**

הקלאס הזה הוא העץ שהולכים להשתמש בהם, בתוכו הוא הולך לאחסן את הנודים שלנו, יש לו שדה שמאחסן את גודל העץ (כמות הנודים), מצביע על ה root של העץ ומצביע על הנוד הכי גדול בעץ.

עשינו מימושים של כל הסיבובים (RR,RL,LR,LL) לפי שלמדנו וכלם בסיבוכיות , בנוסף לזה מימשנו פונקציה בשם MakeBalance שהיא בודקת אם הנוד מאיזון, במקרה שלו היא מבצעת את הסיבוב המתאים שבחרנו איתו לפי שלמדנו זהיא גם בסיבוכיות , מימשנו עוד 3 פונקציות שהולכים להשתמש בהם הרבה והן:

1. **Find**: הפונקציה הזאת מקבל מפתח מסוג KEY והיא מחפשת עליו בעץ, החיפוש שלנו מתבסס על קריאות ריקורסיבות שכל פעם מתקדמים יותר בעץ, הפונקציה תחילה קוראת לפונקציה ריקורסיבית בשם FindNode ושולחת אילו את ה Root ו המפתח, הפונקציה הזאת בודקת אם המפתח שווה להנוד שנשלח לה (שבהתחלה הוא רוט) היא מחזירה את הנוד, אחרת היא בודקת אם המפתח גדול יותר היא קוראת לעצמה עוד פעם אבל עם נוד חדש שהוא הבן הימני של הנוד הזה אחרת קוראת לעצמה עם הבן השמאלי, עושים את הקראות האלה עד או שנגיע למפתח שווה או ל nullptr שזה אומר שהנוד לא קיים בעץ.

סיבוכיות הפונקציה הזאת היא כאשר n הוא מספר הנודים בעץ (זה לפי שהוכחנו בהרצאה בעץ AVL).

1. **Insert**: הפונקציה הזאת היא אחראית על הכנסת נוד חדש להעץ שלנו, היא מקבלת מפתח ומידע, דבר ראשון מחפשת אם כבר קיים נוד כזה בעץ, אם כן היא מחזירה SatutsType::Failure , אחרת אנחנו קוראים לפונקציה ריקורסיבית בשם InserNode , הפונקציה הזאת ממשיכה לקראת בעצמה בדרך ריקורסיבית (כל פעם מתקדמת שלבן שמאלי או ימיני תלוי בההשוואות עם המפתח כפי שלמדנו) עד שתגיע להמקום המתאים בעץ , עושים אלוקציה חדשה לנוד עם המפתח והמידע הנתון ודואגים למקרה קצה, ועכשיו מתחילים לעלות בעץ שלנו כל פעם שעולים דרגה למעלה מעדכנים את הנתונים של הנוד ובודקים במקרה שצריך איזון נאזן עד שיוצאים מהנוד, ובסוף חוזרים לפונקציה node ושם קוראים לפונקציה setmax שהיא עוברת על העץ שלנו ורק הולכת ימינה ומעדכנת את הMAX שיהיה האיבר הכי גדול בעץ, ומגדילים את גודל העץ ב 1.

סיבוכיות הזמן של הפונקציה היא: יש לנו

כלומר הסיבוכיות היא כאשר n הוא מספר הנודים בעץ

1. **Remove**: הפונקציה הזאת אחראית על מחיקת אביר מהעץ שלנו, היא מקבל מפתח מסוג KEY, קוראת לפונקציה FindNode אם הנוד לא קיים במערכת אז מחזירים StatusType::Failure , אחרת קוראים לפונקציה ריקורסיבית בשם DeleteNode, הפונקציה מתחילה לרדת בעץ כל פעם הולכת שמאלה או ימינה בקריאות ריקורסיבית (לפי השוואות המפתח) עד שתגיע להנוד המתאים, יש לנו 3 מקרים של נודים:
2. נוד מסוג Leaf אז רק נמחק איתו.
3. יש לו בן אחד אז נמחק איתו ונעלה את הבן שלו למקומו
4. יש לו 2 בנים, אז הצרך למצוא לו מחליף לכן נילך פעם אחד ימינה ואחרי זה רק שמאלה עד שנגיע ל LEAF ונעשה Swap הין שני הנודים האלו ואחרי זה נמחק את נוד המטרה.

בשלושת המקרים האלו אנחנו שומרים על מבנה העץ.

אחרי שמוחקים מתחילים לחזור בקריאות הרקורסיבית שלנו ומעדכנים את הנודים ודואגים שהכל מאיזון.

וכשחוזרים לפונקציה remove קוראים עוד פעם לפונקציה setMax וגם מקטינים גודל העץ ב 1.

סיבוכית הזמן של הפונקציה היא כאשר n הוא גודל העץ (מאותו הסבר כמו לפני).

1. **BuildInOrderArray:** הפונקציה הזאת בונת לנו מערך שמכיל את כל הנודים שלהעץ בסדר יורד, בנינו איתו כפי שלמדנו (הולכים שמאלה עד הסוף שמים את האיבר אחרי זה עולים דרגה למעלה שמים את האיבר, הולכים ימינה וממשיכים ככה...)

סיבוכיות הזמן של הפונקציה הזאת היא וסיבוכיות המקום אותו דבר כאשר n הוא גודל העץ.

**בנינו Classes שיתאימו למידע שנצרך לשמור ואיך לסדר איתו, תיאור לה Classes שלנו:**

1. **Movies Ranking Key:** הוא מכיל את ה ID, דירוג, צפיות של הסרט, בתוך הקלאס עשינו העמסה של אופירטורים על מנת שנוכל לעשות השוואה בין הסרטים השונים כשההשוואה מתבססת על ה דירוג.
2. **Movie Data:** הוא מכיל את כל המידע על הסרט: ID, דירוג, Genre, צפיות, VIP/Not VIP.

* כל העצים שמכילים את הסרטונים הולכים להכיל את הקלאס של Movie Data וה Key שלהם או שהולך להיות int רגיל במקרה של All Movies ID או הקלאס Movies Ranking Key בשאר העצים של הסרטים.

1. **User Data:** קלאס שמכיל את המידע שלהמשתמשים שלנו, הוא מכיל את ה ID של המשתמש, את הצפיות שלו שעשה לבד בכל Genre בתוך Array בגודל 5 שארבעת התאים הראשונים מתאר Genre מסויים לפי הסדר שנתון לנו בתרגיל הבית, והתא האחרון מתאר את הצפיות בכל סוגי הסרטים לבד, בנוסף לזה יש לו עוד Array בגודל 5 עם אותו מבנה שהולך להחזיק את הצפיות שלו בתוך קבוצה, בנוסף לזה יש לנו פרמטר אם המשתמש

VIP/Not VIP, ומפסר קבוצה במקרה שהוא שייך לאחת (ואם לא המפסר הזה הוא אפס, כי נתון שאין מספר קבוצה 0 לכן השתמשנו במספר הזה כדי לעשות אינדיקציה שהמשתמש לא בקבוצה).

1. **Group Data:** קלאס שמכיל את המידע של הקבוצה, הוא מכיל את המספר של הקבוצה, וגם אם הקבוצה VIP/Not VIP, כמות האנשים בקבוצה, כמות האנשים ה VIP בקבוצה אם יש, מערך בגודל 5 שיכיל את מפסר הצפיות של הקבוצה בכל Genre ובכל הסרטים (אותו מבנה של מערך כמו לפני) , בנוסף לזה הוא מכיל עוד עץ של משתמשים שבתוכו מאחסנים את המשתמשים ששייכים להקבוצה הזאת והם מסודרים לפי ה ID שלהם.

**הסבר להפעולות המבוקשות עם ניתוח סיבוכיות זמן:**

**\*\* בכל הפונקציות שיש התעסקות עם זיכרון דינמי בדקנו אם הכל עבר כפי שרוצים ואם לא החזרנו StatusType של Allocation\_error**

1. **Streaming\_database():**

הפעולה הזאת מאתחלת המבני שלנו של השמונה עצים, כל העצים ריקים ואתחות כל עץ הוא בסיבוכיות .

1. **Virtual ~streaming\_database()**
2. **Add\_movie:**

בפונקציה הזאת מקבלים מספר של סרט, את ה Genre שלו, צפיות ואם הוא VIP, דבר ראשון בודקים אם ה Input חוקי או לא, אחרי זה משתמשים בפונקציה insert עך העץ של כל הסרטים וגם על העץ עם Genre מתאים להסרט.

והפונקציה הזאת תחזיר StatusType שהוא Success אם הצלחנו, Failure אם לא הצלחנו, Invalid input אם מקבלים input לא חוקי.

סיבוכיות הזמן של הפונקציה היא סיבוכיות הזמן של פעמים insert במקרה הגרוע שניהם בגודל K כאשר K הוא מספר הסרטים בעץ ולכן .

1. **Removie\_movie:**

הפונקציה הזאת מקבלת מספר סרטון, בודקת שהוא חוקי אחרי זה קוראים לפונקציה remove על העץ של כל הסרטים והעץ עם ה Genre המתאימה לסרט שלנו, והפונקציה הזאת תחזיר StatusType שהוא Success אם הצלחנו, Failure אם לא הצלחנו, Invalid input אם מקבלים input לא חוקי.

סיבוכיות הזמן של הפונקציה היא סיבוכיות הזמן של פעמים remove במקרה הגרוע שניהם בגודל K כאשר K הוא מספר הסרטים בעץ ולכן .

1. **Add\_user:**

הפונקציה מקבלת ID ואם המשתמש VIP וקוראת לפונקציה Insert עם העץ של כל המשתמשים, והפונקציה הזאת תחזיר StatusType שהוא Success אם הצלחנו, Failure אם לא הצלחנו, Invalid input אם מקבלים input לא חוקי.

סיבוכיות הזמן של הפונקציה היא כאשר n הוא מספר המשתמשים במערכת שלנו.

1. **Remove\_user:**

הפונקציה הזאת היא קצת שונה ממחקית סרט, דבר ראשון מחפשים אם המשתמש קיים ע"י Find , אם כן אנחנו בונים מצביע חדש להמידע הזה, אחרי זה קוראים כרגיל ל remove, ואחרי זה בודקים אם המשתמש היה שייך לקבוצה מסויימת או לא, אם כן נשתמש בהמצביע על הקבוצה הזאת שנמצא בתוך המידע של המשתמש (שהוספנו איתו כאשר הוספנו המשתמש הזה לקבוצה) ונמחק איתו מקבוצה (נסביר את זה אחרי) ונדאג לעדכן את הקבוצה ( גודלה, השמתמשים שמכילה ואם היא עדיין VIP), ואחרי זה נשחרר הזיכרון.

תחזיר StatusType שהוא Success אם הצלחנו, Failure אם לא הצלחנו, Invalid input אם מקבלים input לא חוקי.

סיבוכיות הזמן של הפונקציה שלנו היא כאשר n הוא מספר השמתמשים במערכת שלנו.

1. **Add \_group:**
2. הפונקציה מקבלת ID וקוראת לפונקציה Insert עם העץ של כל הקבוצות, והפונקציה הזאת תחזיר StatusType שהוא Success אם הצלחנו, Failure אם לא הצלחנו, Invalid input אם מקבלים input לא חוקי.

סיבוכיות הזמן של הפונקציה היא כאשר m הוא מספר הקבוצות במערכת שלנו.

1. **Remove\_group:**

מחיקת קבוצה היא כמו מחיקת משתמש אבל בנוסף לזה לפני מחיקת הקבוצה נעבור על כל המשתמשים בה ונעדכן שהם עכשיו לא שייכים לקבוצה, נכול לעשות את זה כי יש לנו מצביעים על המידע של כל משתמש והמידע הזה הוא גם המידע שמוצבע עליו מתוך העץ של המשתמשים (ולכן "גם ישנה שם"), סביוכיות הזמן של הפונקציה הזאת היא הסיבוכיות הרגילה של מחיקה בעת AVL כמו שהיסברנו לפני בנוסף לזה יש צורך לעבור על משתמש מתשמש בתוך הקבוצה לכן סכ"כ יהי לנו כאשר m הוא מפסר הקבוצות במערכת שלנו, ומחזירה StatusType כפי שהוסבר בפונקציה לפני.

1. **Add\_user\_to\_group:**

הפונקציה מקבל כ input את מספר המשתמש ומפסר קבוצה, בודקת שהם חוקים, אחרי זה היא מחפשת על המשתמש בעץ המשתמשים ובודקת שהוא לא שייך לקבוצה, אם הוא לא שייך לקבוצה היא מחפשת על הקבוצה בעץ הקבוצות ומוסיפה איתו להקבוצה (בתוך הקבוצה יש לנו עץ שמכיל מצבעים להמשתמשים ששייכים לקבוצה הזאת), במקרה שהמשתמש כבר שייך לקבוצה או שהקבוצה לא קיימת הפונקציה מחזירה Failure , ואם הצלחנו להוסיף המשתמש לקבוצה תחזיר success, בנוסף דאגנו לבדוק אם מצב הקבוצה השתנה מ not vip ל vip,

הסיבוכיות שלנו היא החיפוש על המשתמש בעץ המשתמשים,חיפוש על הקבוצה בעץ הקבוצות ולעושת insert להמשתמש לתוך עת המשתמשים השייכים לקבוצה, ומקרה הגרוע זה יעלה לנו

כאשר m מספר הקבוצות ו n מספר המשתמשים זה סיבוכיות זמן של , ומחזירה StatusType כפי שהוסבר בפונקציה לפני.

1. **User\_Watch**: