

# רטוב 1 – חלק יבש

## עץ AVL:

נממש את כל העצים בתרגיל בעזרת עצי AVL. עצי AVL כפי שהוכח בהרצאות הם עצים בינאריים המשמרים על סיבוכיות זמן של פעולות הוצאה, הכנסה ומציאת איברים בעץ של  $O(\log(n))$ , וסיבוכיות מקום של  $O(n)$  כלומר נוכל באופן חופשי לבצע פעולות אלה על העץ בלי לדאוג שהסיבוכיות של העץ תחרוג מזה.

נבצע את השינויים הבאים למבנה העץ AVL שלנו

1. העץ ישמור את מספר הצמתים בעץ - כל מה שזה דורש לתחזק זה להוסיף למספר +1 בכל הכנסת ו -1 בכל הוצאה ולכן זה לא משפיע על סיבוכיות הפעולות
2. עץ יוכל לכלול מפתחות מטיפוסים שונים - המפתח של העץ לא יהיה בהכרח מספר אלא יוכל להיות גם כל טיפוס אחר, כל עוד זה טיפוס שיש לו סדר ושניתן להשוות בינו לבין איבר אחר ב  $O(1)$  כיוון שפעולת ההשוואה נשארת  $O(1)$  זה לא משנה את הסיבוכיות של אף פעולה בעץ
3. העץ ישמור מצביע לערך המינימלי בעץ - לאחר כל פעולת הצבעה העץ ימצא את הערך המינימלי בעץ וישמור מצביע אליו. מציאת האיבר המינימלי נעשה בסיבוכיות של  $O(\log(n))$  כיוון שבעץ AVL הגעה לתחתית העץ היא מסיבוכיות כזאת. חוץ מהשינויים האלה המימוש הזה למימוש הסטנדרטי שהועבר בשיעור ולכן לא יפורט כאן.

n - מספר המשתמשים, k - מספר הסרטים, m - מספר הקבוצות.

עבור כל אחד מהפרמטרים הוקצה טיפוס בעזרת מחלקה: *class Movie, class User, class Group*

## streaming database

### סיבוכיות מקום:

את כל הסרטים במערכת נשמור בעזרת 6 עצים שונים -

*AVLtree < int, Movie > movies*: עץ אחד יכלול את כל הסרטים בשירות סטרימינג ממויינים לפי ID של הסרט. כלומר המפתח לכל איבר בעץ יהיה ID והערך יהיה טיפוס שמכיל את כל הפרטים של הסרט

*AVLtree < Movie, int > genreMovies[5]*: ניצור עץ עבור כל אחד מהז'נרים של הסרטים ועץ עבור כל הסרטים שכולם יהיו ממויינים לפי טיפוס הסרט באופן הבא: סידור בסדר יורד לפי דירוג ממוצע של סרטים, במקרה של שוויון בדירוג אז בסדר יורד לפי צפיות ובמקרה של שוויון בצפיות אז בסדר עולה לפי ID. כלומר אם לסרט יש דירוג גבוה יותר מסרט אחר אז הסרט יהיה "מוקדם יותר" בעץ כלומר משמאלה בעץ. כל עץ של ז'נר מסוים יכיל אך ורק את הסרטים של הז'נר הזה. מספר הצמתים בכל אחד מה 6 עצים חסום על ידי מספר הסרטים בסטרימר סך הכל. כל סרט לוקח כמות קבועה של מקום אז לכן הסיבוכיות מקום הכוללת של ששת העצים היא  $O(k)$

*AVLtree < int, User > users*: את כל המשתמשים במערכת נשמור בעזרת עץ אחד. המפתח של כל צומת יהיה ID של המשתמש והערך יהיה טיפוס שמכיל את כל הפרטים של המשתמש. כיוון שמספר הצמתים שווה למספר המשתמשים וכמות הזיכרון לצומת קבועה סיבוכיות המקום של העץ  $O(n)$

*AVLtree < int, shared\_ptr < Group >> groups*: את כל הקבוצות במערכת נשמור בעזרת עץ אחד. המפתח של כל צומת יהיה ID של הקבוצה והערך יהיה טיפוס שמכיל את כל הפרטים של הקבוצה. הטיפוס של כל קבוצה לא יכיל מצביעים למשתמשים בו אלא המשתמשים יכילו מצביעים לקבוצה שלהם.

ולסיכום: סיבוכיות המקום של מבנה הנתונים היא  $O(n + k + m)$  כנדרש.

### סיבוכיות זמן:

*streaming\_database*:

אתחול עצים ריקים, כמות קבועה של מצביעים ולכן הפעולה מתבצעת ב- $O(1)$

*~streaming\_database*:

שחרור כל הזיכרון שהוקצה, כל יחידת זיכרון משתחררת ב- $O(1)$  ובמקרה הגרוע קיימות  $n + k + m$  יחידות זיכרון ולכן כל התהליך יתבצע ב- $O(n + k + m)$

*add\_movie*:

נכניס טיפוס המכיל את כל הפרטים של הסרט לתוך עץ הסרטים הרגיל, עץ הסרטים של הז'אנר של הסרט ועץ כל הסרטים עם המיון המיוחד. מספר הצמתים בכל אחד מהעצים האלה חסום על ידי מספר הסרטים בסטרימר  $k$  וכל אחד מפעולות ההוספה האלה בסיבוכיות  $O(\log(k))$

*remove\_movie*

קודם נשיג את הפרטים של הסרט מהעץ הרגיל ואז נשתמש בהם על מנת למצוא ולהוציא את הצומת של הסרט מתוך שלושת העצים בו הוא נמצא (רגיל, ז'נר מדורג, כולם מדורג). כל אחת מה 4 פעולות הנתונות חסומות על ידי  $O(\log(k))$  ובסה"כ  $O(\log(k)) = O(5\log(k))$

*add\_user, remove\_user*

בכל אחד מהפעולות הללו נוציא או נכניס את האיבר המתאים לעץ המתאים לפי id המתאים כמפתח בעץ. מכיוון שהסיבוכיות של כל פעולה כזאת בעץ היא  $O(\log(x))$  כאשר  $x$  הוא מספר האיברים בעץ. לכן בסה"כ נקבל  $O(\log(n))$  במקרה הגרוע אז עבור משתמשים, סרטים, וקבוצות נקבל:  $O(\log(n))$ ,  $O(\log(k))$ ,  $O(\log(m))$  בהתאמה.

*add\_group*:

נכניס את האיבר המתאים לעץ המתאים לפי id המתאים כמפתח בעץ. מכיוון שהסיבוכיות של כל פעולה כזאת בעץ היא  $O(\log(x))$  כאשר  $x$  הוא מספר האיברים בעץ. לכן בסה"כ נקבל  $O(\log(m))$  במקרה הגרוע

*Remove\_group*

ראשית נמצא מעץ הקבוצות את הקבוצה שאותה נרצה להסיר, פעולה זאת לוקחת  $O(\log(m))$  במקרה הגרוע. נשנה דגל בוליאני בטיפוס הקבוצה שמסמן שהקבוצה היא "סגורה" ב- $O(1)$ . בכל פעולה של גישה לאחד המשתמשים תעשה בדיקה שתוודא שהקבוצה של המשתמש סגורה ואז המשתמש יפסיק להצביע לקבוצה והקבוצה תצא מהזיכרון רק כשכל המשתמשים של הקבוצה יתעדכנו. דבר זה אינו חורג ממגבלות הזיכרון כיוון שלכל היותר יכולים להיות  $n$  קבוצות סגורות כאלה, אחת לכל משתמש. הקבוצה "הסגורה" תוסר מעץ הקבוצות מה שישאיר את גודלו להיות  $m$  וישמר את סיבוכיות הפעולות עליו תקינה.

*add\_user\_to\_group*

ראשית נמצא מעץ המשתמשים את המשתמש אותו נרצה להוסיף לקבוצה, לאחר מכן נמצא מעץ הקבוצות את הקבוצה שאליה נרצה להוסיף את המשתמש, פעולות אלה אורכות  $O(\log(n))$ ,  $O(\log(m))$  בהתאמה. נעת נעדכן את השדה של המשתמש שתפקידו לסמן באיזו קבוצה נמצא המשתמש (עדכון מצביע)  $O(1)$ . ונעדכן את השדות הרלוונטיים של הקבוצה גם ב- $O(1)$  ולכן בסה"כ:  $O(\log(n) + \log(m)) = O(\log(n) + \log(m))$

*User\_watch*

נוציא את טיפוס המשתמש מעץ המשתמשים בסיבוכיות  $O(\log(n))$  נעדכן את הערכים ואת המיקום בעץ של טיפוס הסרטים הנמצאים בשלושת העצים הרלוונטיים לסרט. נעשה זאת בעזרת סדרה הוצאות והכנסות כאשר כל פעולה כזאת אורכת  $O(\log(k))$ . ולכן בסה"כ: סיבוכיות כוללת היא  $O(\log(n) + \log(k))$

### group\_watch

ראשית נמצא מעץ הסרטים את הסרט בו תרצה הקבוצה לצפות, לאחר מכן נמצא מעץ הקבוצות את הקבוצה הצופה. פעולות אלה אורכות  $O(\log(k))$ ,  $O(\log(m))$  בהתאמה. כעת נותר לבצע את הפעולות הבאות:

- 1) עדכון הסרט שצפו בו במספר הצפיות ובעוד נתוני מספרים נוספים -  $O(1)$
- 2) הוצאה מעצי הסרטים הכנסה אליהם מחדש עם הסרט המעודכן -  $O(X \cdot \log(m))$  כאשר  $X$  מספר קבוע של פעולות כאלה. ולכן בסה"כ נקבל סיבוכיות זמן:  $O(\log(m) + \log(k)) = O(\log(m) + (X + 1) \cdot \log(k))$

### get\_all\_movies\_count

נחזיר את מספר הצמתים בעץ הסרטים הרגיל. כיוון שהגדרנו שעצים שומרים את מספר הצמתים בו כשדה, הגישה למידע זה תהיה  $O(1)$

### get\_all\_movies

נבצע סיור inorder בעץ הממוין לפי דירוג המתאים ל-genre שהתקבל ונכניס את id של הסרטים למערך לפי הסדר. נממש את הסיור inorder באופן רקורסיבי בדומה לאלגוריתם שהודגם בתרגול שבו סיבוכיות זמן היא  $O(x)$  כאשר  $x$  הוא מספר הצמתים בעץ. ולכן עבור genre=NONE נעבור על כל הסרטים במערכת פעולה שאורכת  $O(k)$ , ובכל מקרה אחר העץ יכיל  $k_{genre}$  סרטים והפעולה אורכת  $O(k_{genre})$

**\*\* כיוון שקריאות הפונקציות הרקורסיביות תופסות מקום בstack כיחס לעומק העץ, אין פגיעה בסיבוכיות המקום.**

### get\_num\_view

נמצא את המשתמש המתאים מתוך עץ המשתמשים בסיבוכיות  $O(\log(n))$  ואז נשיג מהטיפוס המשתמש את ערך הצפיות המתאים. ערך זה תמיד מעודכן כי כל פעם שהמשתמש צופה בסרט או שהקבוצה שהמשתמש חלק ממנה צופה בסרט אז טיפוס המשתמש מתעדכן בהתאם. הפעולה אורכת  $O(\log(n))$  במקרה הגרוע, כנדרש.

### rate\_movie

נמצא את המשתמש ואת הסרט המתאים מעצי המשתמשים והסרטים, פעולות אלה אורכות  $O(\log(n))$ ,  $O(\log(k))$  בהתאמה נוודא שהמשתמש הוא vip אם נדרש ונעדכן את ממוצע הדירוגים של טיפוס הסרט. בנוסף נצטרך לעדכן באופן דומה גם את טיפוס הסרט 2 ההופעות שלו בעצים אחרים ונעשה זאת על ידי פעולות הוצאה והכנס לעץ ככה שהסדר של הסרטים גם יתעדכן. כיוון שפעולות ההוצאה וההכנסה הן כמות חסומה של פעולות עם סיבוכיות  $O(\log(k))$  אז בסה"כ נקבל:  $O(\log(n) + \log(k))$

### get\_group\_recommendation

ראשית נמצא מעץ הקבוצות את הקבוצה שתרכזה המלצה, פעולה זאת לוקחת  $O(\log(m))$  במקרה הגרוע. לאחר מכן נמצא מהו הז'אנר האהוב ע"י חברי הקבוצה בעזרת מעבר בלולאה בעלת 4 איטרציות שקוראת 4 ערכי int השמורים בתוך אובייקט הקבוצה. ערכים אלה שומרים את כמות הצפיות שהקבוצה וחבריה צפו בכל ז'אנר. ערכים אלה קלים לתחזוק מכיוון שדורשים כמות סופית של פעולות חשבון במספרי int בכל מאורע של צפייה בסרט -  $O(1)$ . לבסוף נשאר למצוא מתוך עץ הסרטים של הז'אנר את הסרט בעל הדירוג הממוצע הגבוה ביותר – עבור עץ ה AVL שלנו הוגדר סרט זה להיות "בעל הערך המינימאלי ביותר", שאליו העץ מחזיק מצביע עדכני וגישה אליה היא ב  $O(1)$ . ולכן בסה"כ נקבל סיבוכיות זמן  $O(\log(m))$  כנדרש.

**\*\* הערה - מימושי האלגוריתמים הנ"ל מכילים פעולות נוספות שסיבוכיות הזמן שלהם היא  $O(1)$  ולא הוצגו למען פשטות ההסבר. דוגמאות לפעולות ב  $O(1)$  שהושמטו: גישה לשדות int פנימיים של מחלקות (ע"י פונקציות get כמובן), כמות סופית של פעולות חשבון בין int, גישה למצביעים ועדכון מצביעים, בדיקות nullptr ובדיקות נכונות בעזרת משתני booli int.**