גיליון רטוב 1 – חלק יבש

תיאור מבני הנתונים:

להלן תיאור מבני הנתונים העיקריים בהם נשתמש בפתרון:

Song:

אובייקט המייצג שיר.

מאפיינים:

- int id: מזהה שיר.
- int playsCount: מספר הפעמים ששמעו את השיר.
- int playlistsCount: מספר רשימות ההשמעה אליהן שיר זה שייך.

AVLAllSongs:

אובייקט המייצג צומת בעץ AVL של שירים.

מאפיינים:

- int songId: מזהה השיר המשוייך לצומת
- Song *song_ptr: מצביע לאובייקט השיר אליו צומת זה משוייך.
- SongNode* left: תת העץ השמאלי.
- SongNode* right: תת העץ הימני.

Playlist:

אובייקט המייצג פלייליסט.

מאפיינים:

- int playlistId: מספר מזהה.
- int numOfSongs: מספר השירים בפלייליסט.
- SongTreePlaylist *songsByIdTree: עץ AVL של שירי הפלייליסט, כאשר הסדר נקבע לפי מזהה השיר.
- PlayCountNode *AVLPlayCount: עץ AVL של שירי הפלייליסט, כאשר הסדר נקבע לפי מספר ההשמעות של השיר.

צריכת מקום:

- במקרה הגרוע כל n השירים של DSpotify שייכים לפלייליסט זה. כלומר העצים ש-songsByIdTree ו-AVLPlayCount הם השורשים שלהם מכילים n צמתים בעל מקום אחסון קבוע לכל אחד, כלומר סיבוכיות המקום של עצים אלו היא O(n).
- במקרה הגרוע, הרשימה המקושרת ש-songListHead שייכת אליה מכילה n שירים, לכן סיבוכיות המקום של רשימה זו היא $O(n)$.
- שאר המאפיינים תופסים מקום קבוע בזכרון לכל פלייליסט.

לכן סך הכל סיבוכיות המקום של playlist היא O(n).

AVLPlaylist:

אובייקט המייצג צומת בעץ AVL של פלייליסטים, כאשר הסדר נקבע לפי מזהה הפלייליסט.

מאפיינים:

- int playlistId: מזהה הפלייליסט.
- Playlist *playlist_ptr: מצביע לפלייליסט המשוייך לצומת.
- AVLPlaylist *right: מצביע לבן הימני של הצומת.
- AVLPlaylist *left: מצביע לבן השמאלי של הצומת.
- int height: גובה העץ.

SongNodeList:

אובייקט המייצג צומת ברשימה מקושרת של שירים.

מאפיינים:

- Song *song_ptr: מצביע לאובייקט השיר המשוייך לצומת.
- SongNodeList* next: מצביע לצומת הבאה ברשימה.
- SongNodeList* prev: מצביע לצומת הקודמת ברשימה.

DSpotify:

מבנה הנתונים הראשי המכיל את השירים והפלייליסטים.

מאפיינים:

- AVLAllSongs *songs: השורש של עץ ה-AVL המכיל את כל שירי המערכת, כאשר הסדר נקבע לפי מזהה השיר.
- AVLPlaylist *playlists: השורש של עץ ה-AVL המכיל את כל הפלייליסטים במערכת, כאשר הסדר נקבע לפי מזהה הפלייליסט.

צריכת מקום:

בכל רגע נתון האובייקט DSpotify מורכב מהמבנים הבאים:

- האובייקט songs מטיפוס AVLAllSongs. זהו עץ AVL המורכב מ-n אובייקטים מטיפוס AVLAllSongs, כאשר כל אובייקט מטיפוס זה צורך כמות מקום קבועה. לכן סיבוכיות המקום של מבנה זה היא $O(n)$.
- האובייקט playlist מטיפוס AVLPlaylist. זהו עץ AVL המורכב מ-m אובייקטים מטיפוס AVLPlaylist, כאשר כל אובייקט מטיפוס זה מכיל:
  - האובייקט playlist, המכיל את העצים songsByIdTree, AVLPlayCount המורכבים מ-$n_{playlistId}$ צמתים. כלומר צריכת המקום של חלק זה של כל פלייליסט לינארית במספר השירים השייכים לפלייליסט, כלומר סיבוכיות המקום היא

$$O(3 * n_{playlistId})$$

  - מאפייני העץ (playlistId, height וכו...) אשר תופסים מקום קבוע. כלומר חלק זה של כל פלייליסט תופס מקום בסיבוכיות $O(1)$.

לכן סיבוכיות המקום של AVLPlaylist היא

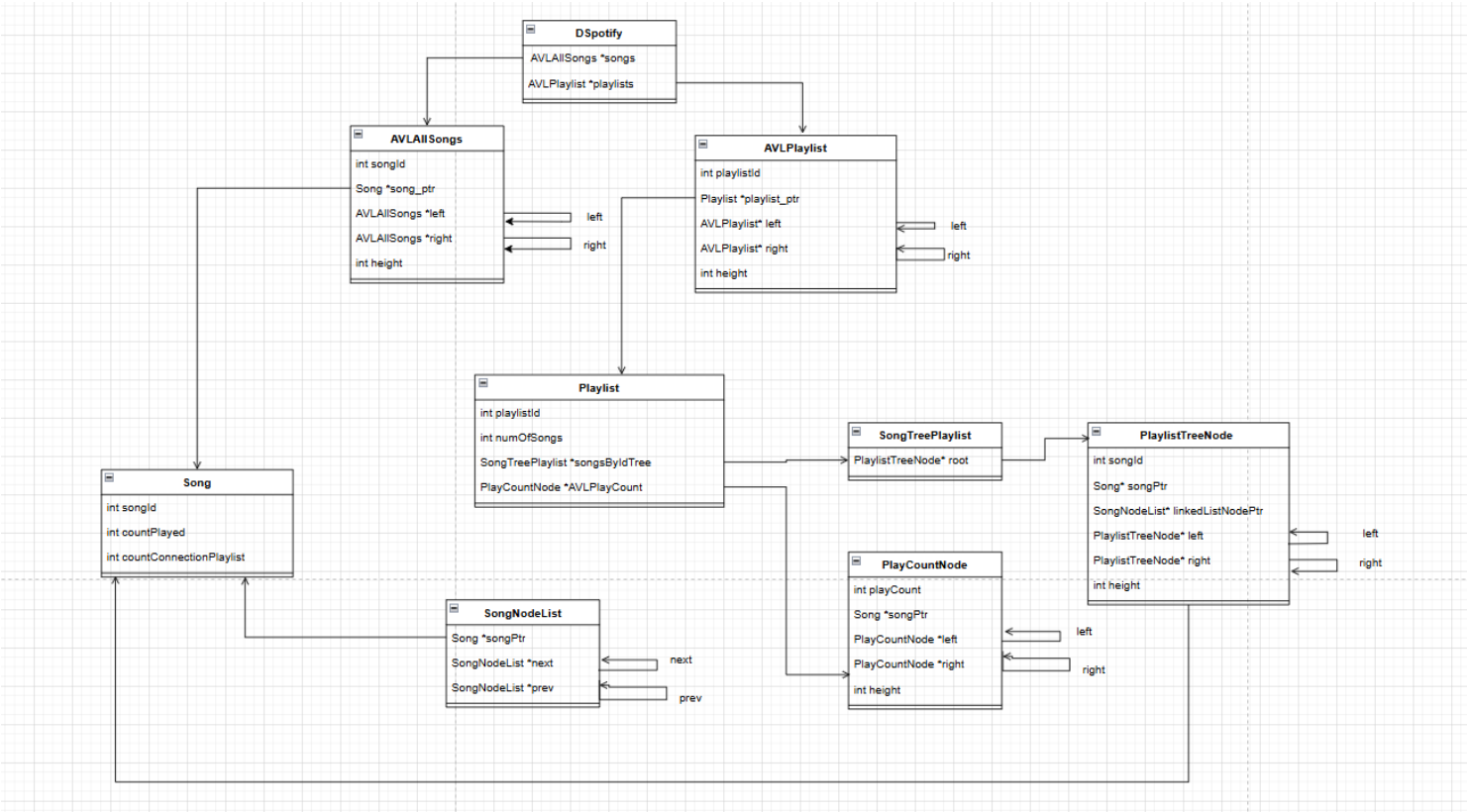
$$O(1 + 3n_1 + 1 + 3n_2 + \cdots + 1 + 3n_m)$$
$$= O\left(\underbrace{(1 + 1 + .. + 1)}_{m\ times} + 3(n_1 + n_2 + \cdots + n_m)\right)$$

$$= O\left(m + 3\sum_{i\ is\ a\ playlist} n_i\right)$$
$$= O(m + \sum_{i\ is\ a\ playlist} n_i)$$

כלומר, סך הכל סיבוכיות המקום של מבנה הנתונים שתיארנו היא

$$O(n + m + \sum_{i\ is\ a\ playlist} n_i)$$

שרטוט UML של מבני הנתונים:



מימוש הפעולות:

- הערה לגבי סיבוכיות הזמן והמקום: נשים לב כי עבור כלל הפעולות, שגיאה כלשהי מקצרת את זמן הריצה של הפעולה, לכן זמן הריצה הארוך ביותר מתקבל במקרה של הצלחה, לכן בכלל הפעולות נתייחס למקרה של הצלחה בתור המקרה הגרוע.

DSpotify()

תיאור: בפעולה זו אנו מאתחלים את songs ע"י אתחול עץ AVL ריק ואת playlists ע"י אתחול עץ AVL ריק.

סיבוכיות זמן: כפי שראינו בהרצאה, סיבוכיות הזמן של אתחול עץ היא $O(1)$, לכן סיבוכיות הזמן של פעולה זו היא

$$O(1+1) = O(1)$$

סיבוכיות מקום: הפעולה יוצרת שני עצים ריקים, כלומר שני העצים צורכים כמות קבועה של זיכרון, לכן סיבוכיות המקום של הפעולה היא

$$O(1+1) = O(1)$$

Virtual ~Dspotify

תיאור: בפעולה זו אנו משחררים את כל הזיכרון שהוקצה לעץ playlists ע"י שחרור כל הצמתים בעץ, לאחר מכן אנו משחררים את כל הזיכרון שהוקצה לעץ songs ע"י שחרור כל הצמתים בעץ זה.

סיבוכיות זמן: במקרה הגרוע, כל הפלייליסטים מכילים את כל השירים במערכת, לכן על מנת לשחרר את כל הצמתים בכל העצים עלינו לשחרר את כל m הצמתים ב-playlists, כאשר בכל צומת אנו משחררים את הזיכרון שהוקצה ל-n צמתים בכל אחד משני העצים המשוייכים לפלייליסט (songsByIdTree ו-AVLPlayCount). כמו כן, עלינו לשחרר את הזיכרון שהוקצה ל-n הצמתים בעץ songs. את מחיקת העצים נעשה באמצעות סיור postorder, כלומר קודם נמחק את הצומת השמאלית, לאחריה את הצומת הימנית ולבסוף את השורש. לכן סיבוכיות הזמן של פעולה זו היא

$$O(m*2n) = O(2mn) = O(mn)$$

סיבוכיות מקום: הפעולה מוחקת שני עצי AVL באופן רקורסיבי (באמצעות סיור postorder), לכן כפי שראינו בהרצאות סיבוכיות המקום עבור מחיקת העץ songs היא $O(\log n)$ וסיבוכיות המקום של מחיקת העץ playlists היא $O(\log m)$, לכן סיבוכיות המקום של פעולה זו היא

$$O(\log m + \log n)$$

StatusType add_playlist(int playlistId)

תיאור: בפעולה זו נבדוק אם קיים PlaylistNode בעל המזהה המבוקש בעץ playlistTree, ואם לא קיים PlaylistNode כזה ניצור אובייקט PlaylistNode חדש עם המאפיינים המתאימים ונוסיף אותו ל-playlistTree.

סיבוכיות זמן: סיבוכיות הזמן של יצירת אובייקט PlaylistNode היא $O(1)$, וכיוון ש-playlistTree הוא עץ AVL, סיבוכיות הזמן של חיפוש צומת בעץ והוספת צומת לעץ היא $O(\log m)$.

לכן סיבוכיות הזמן של פעולה זו היא

$$O(1+2*\log m) = O(\log m)$$

סיבוכיות מקום: הפעולה יוצרת אובייקט PlaylistNode בגודל קבוע ומכניסה אותו לעץ AVL באופן רקורסיבי, לכן סיבוכיות המקום של הפעולה היא

$$O(1+\log m) = O(\log m)$$

StatusType delete_playlist(int playlistId)

תיאור: בפעולה זו אנו מוצאים את ה-PlaylistNode המתאים ל-playlistId ב-playlistTree, בודקים האם ניתן למחוק את ה-playlist (ע"י בדיקה שמספר השירים בפלייליסט הוא 0) ומוציאים את ה-PlaylistNodeבמידה וניתן למחוק את ה-playlist.

סיבוכיות זמן: כפי שראינו בהרצאה, חיפוש צומת בעץ AVL היא פעולה בסיבוכיות זמן $O(\log m)$, הבדיקה נעשית בזמן קבוע, כלומר $O(1)$, וסיבוכיות הזמן של הוצאת צומת מעץ AVL היא $O(\log m)$, לכן סיבוכיות הזמן של פעולה זו היא

$$O(\log m + 1 + \log m) = O(2\log m + 1) = O(\log m)$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוציאים צומת בעץ AVL באופן רקורסיבי, לכן כפי שראינו בהרצאות סיבוכיות המקום של פעולה זו היא

$$O(\log m)$$

StatusType add_song(int songId, int plays)

תיאור: בפעולה זו אנו מחפשים שיר בעל מזהה songId בעץ songs, ואם הוא אינו קיים בעץ יוצרים אובייקט SongNode חדש ומוסיפים אותו ל-songs .

סיבוכיות זמן: יצירת אובייקט מסוג SongNode נעשית בזמן קבוע, כלומר בסיבוכיות זמן $O(1)$, וכן songs הוא עץ AVL לכן הוספת צומת אליו נעשית בסיבוכיות זמן $O(\log m)$. לכן סיבוכיות הזמן של פעולה זו היא

$$O(1 + \log m) = O(\log n)$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוסיפים צומת לעץ AVL באופן רקורסיבי, לכן כפי שראינו בהרצאות סיבוכיות המקום של פעולה זו היא

$$O(\log n)$$

StatusType add_to_playlist(int playlistId, int songId)

תיאור: בפעולה זו אנו מחפשים את הפלייליסט המתאים ב-playlists ואת השיר המתאים ב-songs (כדי לודא שהשיר נמצא במערכת) וב-songsByIdTree של הפלייליסט. במקרה הגרוע השיר אינו נמצא בפלייליסט ולכן ניצור אובייקט SongTreePlaylist חדש ואובייקט SongNode חדש ונוסיף אותם ל-songsByIdTree.

סיבוכיות זמן: העצים playlists, songs ו-songsByIdTree הינם עצי AVL, לכן סיבוכיות הזמן של חיפוש ב-playlists היא $O(\log m)$, סיבוכיות הזמן של חיפוש ב-songs היא $O(\log n)$ וסיבוכיות הזמן של חיפוש השיר ב-songsByIdTree והוספת צומת אליו היא $O(\log n_{playlistId})$ לכל פעולה. כמו כן סיבוכיות הזמן של יצירת האובייקטים היא $O(1)$ לכל פעולה, לכן סך הכל סיבוכיות הזמן של הפעולה היא

$$O\left(\log m + \log n + 2\log n_{playlistId} + 2\right) \underbrace{=}_{(1)} O(\log m + 3\log n + 2) = O(\log m + \log n)$$

(1) לפי הגדרתם מתקיים $n_{playlistId} \le n$ לכן $\log n_{playlistId} = O(\log n)$.

סיבוכיות מקום: בפעולה זו אנו מבצעים חיפוש בעצי AVL בגובה n ו-m, ומוסיפים צומת לעץ AVL בגובה n, לכן סיבוכיות המקום של הפעולה היא

$$O(\max(\log n, \log m))$$

StatusType delete_song(int songId)

תיאור: בפעולה זו תחילה נחפש את הצומת המתאימה לשיר ב-songsTree, בודקים שמספר הפלייליסטים אליהם השיר משוייך הוא 0 ובמידה שכן מוציאים את הצומת מהעץ.

סיבוכיות זמן: songTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש צומת והוצאת צומת ב-songsTree היא $O(\log n)$ לכל פעולה. סיבוכיות הזמן של הבדיקה של מספר הפלייליסטים אליהם השיר משוייך היא $O(1)$, לכן סיבוכיות הזמן של הפעולה היא

$$O(2\log n + 1) = O(\log n)$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוציאים צומת בעץ AVL בגובה n, לכן סיבוכיות המקום של הפעולה היא

$$O(\log n)$$

StatusType remove_from_playlist(int playlistId, int songId)

תיאור: בפעולה זו אנו מחפשים את הפלייליסט המתאים ב-playlistTree, מחפשים את הצומת המתאימה ב-songsByIdTree של הפלייליסט, מורידים את ה-playlist_count של השיר ומסירים את הצומת מ-playlistTree ומה-songNodeList של הפלייליסט ולבסוף מורידים את ה-numOfSongs של הפלייליסט ב-1.

סיבוכיות זמן: playlistTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש הפלייליסט ב-playlistTree היא $O(\log m)$. songsByIdTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש הצומת של השיר ב-songsByIdTree היא $O(\log n_{plalistId})$. סיבוכיות הזמן של הסרת השיר מ-songNodeList ועדכון המשתנים המתאימים היא $O(1)$ לכל פעולה, לכן סיבוכיות הזמן של הפעולה היא

$$O\left(\log m + \log n_{playlistId} + 3\right) = O(\log m + \log n_{playlistId})$$

סיבוכיות מקום: בפעולה זו אנו מחפשים צומת בעץ AVL בגובה m ומחפשים ומוציאים צומת מעץ AVL בגובה $n_{playlistId}$, לכן סיבוכיות המקום של הפעולה היא

$$O(\max(\log m, \log n_{playlistId}))$$

output_t <int> get_plays(int songId)

תיאור: בפעולה זו אנו מחפשים את השיר ב-songsTree ומחזירים את ערך המאפיין countPlayed.

סיבוכיות זמן: songsTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש צומת ב-songsTree היא $O(\log n)$. סיבוכיות הזמן של החזרת countPlayed היא $O(1)$, לכן סיבוכיות הזמן של הפעולה היא

$$O(\log n + 1) = O(\log n)$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוציאים צומת בעץ AVL בגובה n, לכן סיבוכיות המקום של הפעולה היא

$$O(\log n)$$

output_t <int> get_num_songs(int playlistId)

תיאור: בפעולה זה אנו מחפשים את הפלייליסט המתאים ב-playlistTree ומחזירים את ערך המאפיין numOfSongs.

סיבוכיות זמן: playlistTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש צומת ב-playlistTree היא $O(\log m)$. סיבוכיות הזמן של החזרת המאפיין numOfSongs היא $O(1)$, לכן סיבוכיות הזמן של הפעולה היא

$$O(\log m + 1) = O(\log m)$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוציאים צומת בעץ AVL בגובה n, לכן סיבוכיות המקום של הפעולה היא

$$O(\log m)$$

output_t <int> get_by_plays(int playlistId, int plays)

תיאור: בפעולה זה אנו מחפשים את הפלייליסט המתאים ב-playlistTree. לאחר מכן אנו מחפשים את השיר המתאים בעץ AVLPlayCount של הפלייליסט.

החיפוש מתחיל בצומת השורש ב-AVLPlayCount ונעשה באופן הבא

- נגדיר מצביע לצומת הנוכחית בשם current.
- נגדיר מצביע לצומת הנוכחית בשם bestCandidate.
- כל עוד current אינו מצביע ל-nullptr נבצע את הפעולות הבאות בלולאה
  - אם מספר ההשמעות בצומת הנוכחית גדול או שווה ל-plays נבצע את הפעולות הבאות
    - אם מספר ההשמעות בצומת הנוכחית קטן ממספר ההשמעות ב-bestCandidate, או אם מספר ההשמעות בשתי הצמתים זהה אך המזהה של השיר המשוייך לצומת הנוכחית קטן מהמזהה של השיר המשוייך ל-bestCandidate, נשנה את הכתובת ב bestCandidate לכתובת של הצומת הנוכחית.
    - נמשיך את החיפוש בתת העץ השמאלי ע"י שינוי הכתובת ב-current לכתובת של הבן השמאלי של העץ.
  - אם מספר ההשמעות בצומת הנוכחית קטן מ-plays נמשיך את החיפוש בתת העץ הימני ע"י שינוי הכתובת ב-current לכתובת של הבן הימני של העץ.

סיבוכיות זמן: playlistTree הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש ב-playlistTree היא $O(\log m)$. AVLPlayCount הוא עץ AVL, לכן סיבוכיות הזמן של חיפוש ב-AVLPlayCount היא $\log n_{playlistId}$. לכן סיבוכיות הזמן של הפעולה היא

$$O(\log m + \log n_{playlistId})$$

סיבוכיות מקום: בפעולה זו אנו מחפשים ומוציאים צומת בעץ AVL בגובה m, ולאחר מכן מבצעים חיפוש איטרטיבי בעץ AVL, לכן סיבוכיות המקום של הפעולה היא

$$O(\log m + 1) = O(\log m)$$

StatusType unite_playlists(int playlistId1,int playlistId2)

תיאור: ראשית נמצא את הפלייליסטים בעץ playlists. לאחר מכן נמזג את עצי ה-songsByIdTree של שני הפלייליסטים לפי האלגוריתם שראינו בתרגול:

- נמיר את העצים לרשימות מקושרות ממוינות באמצעות סיור inorder.
- נמזג את הרשימות לרשימה מקושרת ממוינת אחת המכילה את כל השירים משני הפלייליסטים כאשר לכל שיר צומת אחת ברשימה (כלומר הרשימה אינה מכילה כפילויות)
- מכן נבנה עץ כמעט ריק בגודל (כפי שראינו בתרגול).
- נבצע סיור inorder על העץ החדש ונשים בו את איברי הרשימה המקושרת מראש הרשימה עד לסופה.
- נשנה את המצביע songsByIdTree של אובייקט הפלייליסט הראשון להצביע על העץ החדש.

באותו אופן נמזג את עצי ה-AVLPlayCount של שני העצים.

לבסוף נשחרר את כל הזיכרון שהוקצה לאובייקט playlist2 על ידי שחרור הזכרון שהוקצה לעצים songsByIdTree ו-AVLPlayCount המשוייכים לאובייקט של הפלייליסט. כמו כן נשחרר את הזיכרון שהוקצה עבור הרשימות המקושרות אשר יצרנו בפעולה.

סיבוכיות זמן: playlists הוא עץ AVL, לכן חיפוש הצמתים של הפלייליסטים בעץ playlists נעשה בסיבוכיות זמן $O(2\log m)$. במקרה הגרוע כל שירי המערכת נמצאים בשני הפלייליסטים הממוזגים, כלומר מתקיים $n_1 + n_2 = n + n = 2n$. במקרה זה, המרת עצי ה-songsByIdTree ו-AVLPlayCount למערכים ממויינים נעשית בסיבוכיות זמן $O(n_1 + n_2) = O(2n)$ לכל עץ. מיזוג הרשימות המקושרות, וכן בניית עץ ואכלוסו נעשים ב-$O(n_1 + n_2) = O(2n)$ לכל פעולה. שחרור הזיכרון שהוקצה ל-n הצמתים של העצים songsByIdTree ו-AVLPlayCount ו-n הצמתים של הרשימה המקושרת החדשה נעשה בסיבוכיות זמן של $O(n)$ לכל פעולה.

לכן סך הכל סיבוכיות הזמן של הפעולה היא:

$$O(2\log m + 2 * (2n + 2n + 2n + 2n) + 3n) = O(2\log m + 19n)$$
$$= O(\log m + n)$$

סיבוכיות מקום: תחילה אנו מחפשים 2 צמתים בעץ AVL בגובה m. לאחר מכן, במקרה הגרוע, שני הפלייליסטים מכילים את כל n השירים במערכת. לכן במהלך הפעולה אנו יוצרים 2 רשימות מקושרות באורך n כל אחת. לאחר מכן אנו יוצרים רשימה מקושרת עם הערכים המשותפים לשתי הרשימות המקושרות הקודמות באורך n (כיוון שהיא מכילה צומת עבור כל שיר במערכת לכל היותר פעם אחת). לאחר מכן אנו יוצרים עץ כמעט ריק בעל n צמתים, כמו כן פעולות היצירה והאכלוס של העץ נעשות באופו רקורסיבי, ולכן סיבוכיות המקום שלהן היא $O(\log n)$. נשים לב שאנו מבצעים את הפעולות הללו פעמיים (פעם עבור songsByIdTree ופעם עבור AVLPlayCount), לכן סיבוכיות המקום של הפעולה היא

$$O\big(2\log m + 2 * (2n + n + n + \log n)\big) = O(2\log m + 8n + 2\log n) = O(n)$$

סיבוכיות מקום של המבנה DSpotify: כפי שהראינו בתיאור מבני הנתונים, סיבוכיות המקום של המבנה DSpotify היא

$$O(n + m + \sum\nolimits_{i\ is\ a\ playlist} n_i)$$

נשים לב שבכל הפעולות סיבוכיות המקום היא לכל היותר $O(n)$ (עבור הפעולה unite_playlists), לכן סיבוכיות המקום של מבנה הנתונים שהצענו במקרה הגרוע היא

$$O\left(n + m + \sum\nolimits_{i\ is\ a\ playlist} n_i + n\right) = O(n + m + \sum\nolimits_{i\ is\ a\ playlist} n_i)$$

כנדרש.