תרגיל רטוב 1 במבני נתונים – חלק יבש

מגישים:

סהר כהן – 206824088
יובל נהון – 206866832

תאריך הגשה: 7/12/2017

טיפוס הנתונים של המערכת כולה הוא **Colosseum**. לצורך הגדרת מבנה זה, נציג את מבני הנתונים בהם נעשה שימוש על מנת לממש את היחידה Colosseum, ולאחר מכן את המערכת כולה:

Splay Tree:

מבנה הנתונים המרכזי אשר ישמש אותנו הוא עץ חיפוש בינארי (BST) עם תכונת splay – קרי, splay tree. עץ זה הוא עץ חיפוש בינארי, ולכן מקיים את כל תכונות העצים הבינאריים: לכל צומת בעץ ישנו מפתח ושני מצביעים לכל היותר לצמתים נוספים, כאשר מוגדר יחס סדר הקובע את מבנה העץ: מפתחות בעלי ערך קטן ממש מערך המפתח בצומת יכנסו משמאל, ואחרת ייכנסו מימין. בעץ מסוג splay קיימת תכונה נוספת: תכונת ה - splay, לפיה העץ הוא עץ אשר משנה תמידית את מבנה הצמתים שלו בכל גישה שנעשתה אליו בצורה הבאה: בכל פעם שמבקרים באחת הרשומות בעץ, הצומת עבור אותה הרשומה יהפוך להיות השורש החדש של העץ, בצורה שלא תפר את מגבלות העץ החיפוש הבינארי. לצורך כך קיימים גלגולים אשר משנים את סדר האיברים בעץ בצורה בה יתעדכן כראוי. הפעולות של splay tree הן כדלקמן (בחתימות הבאות, הסימון T מייצג את טיפוס הרשומות):

(SplayTree* t, T value) **rotate** – ביצוע גלגולים מתאימים (zig, zig-zig, zig-zag כפי שנלמד בתרגולים), עד לקבלת עץ חיפוש בינארי תקין, ובו הצומת בו רשומה value הוא השורש החדש בעץ. נציין שהפעולות zig, zig-zig, zig-zag הינן פעולות של החלפת מצביעים בין מספר קבוע של צמתים, ואינן רלוונטיות למשתמש חיצוני אלא כפריט מימוש בלבד. משום שמדובר בהחלפת מצביעים בגודל קבוע – סיבוכיות זמן ריצה של כל גלגול בודד היא O(1), לכל סוג. מספר הגלגולים חסום על ידי גובה המסלול מהשורש אל הצומת, ולכן נקבל סיבוכיות משוערכת של **O(log(n))** עבור פעולת rotate. זוהי הנחה בתרגיל שאין להוכיח, שלפיה גובה העץ h הוא בסדר גודל של log(n) לפי ניתוח משוערך, בדומה לסיבוכיות הנתונה עבור הפונקציות הבאות:

(SplayTree* t, T value) **find** – חיפוש של הרשומה value בתוך העץ t.
(SplayTree* t, T value) **insert** – הכנסה של הרשומה value לתוך העץ t.
(SplayTree* t, T value) **kick** – מחיקה של הצומת ובה מוחזקת הרשומה value מתוך העץ t.

לאחר ביצוע כל אחת מהפעולות הבאות, נבצע את הפעולה rotate על הצומת האחרונה שבה ביקרנו (לדוגמה, עבור kick, נגלגל את צומת האב לצומת אותה מחקנו, במידה וישנה), על מנת לשמור על עקרון splay של העץ. לפי ההנחה של תרגיל זה, סיבוכיות זמן ריצה משוערכת של כל אחת משלוש הפעולות הנ"ל היא **O(log(n)), וסיבוכיות worst-case היא O(n).**

נוסיף גם את הפונקציות הבאות כהרחבה ל – splay tree הסטנדרטי:

(SplayTree* line, T value) **concatLinear** – בנאי זה, בהינתן עץ line מסוג "שרוך", יוצרת צומת חדש המכיל את הרשומה value, וכבן שמאלי את כל העץ line, לקבלת עץ חדש מסוג שרוך, עם איבר אחד נוסף. הבן הימני של הצומת החדש יאותחל להיות null. הקצאה ואתחול של שדות צומת בודד מתבצעת ב – O(1), ולכן סיבוכיות זמן הריצה של הפונקציה היא **O(1) ב – worst-case.**

(SplayTree* t) **findMax** – מקבלת splay tree ומחזירה את האיבר הגדול ביותר בו, לפי יחס סדר גודל שהוגדר עבור המפתחות. לצורך כך, הפונקציה מתקדמת מהשורש באופן רקורסיבי עד לעלה הימני ביותר בעץ, בכך שתכנס בכל איטרציה לבן הימני, עד שתגיע ל – null. כפי שהצגנו קודם, באופן משוערך נקבל שגובה splay tree הוא h = O(log(n)), ומשום שהפונקציה מבצעת מסלול מהשורש לאחד העלים, ובסה"כ נקבל **סיבוכיות זמן ריצה משוערכת היא O(log(n)), וסיבוכיות worst-case היא O(n).**

ניתוח סיבוכיות מקום: בהינתן רשומה מטיפוס T בעלת סיבוכיות מקום נדרשת O(m) – סיבוכיות מקום נדרשת עבור עץ splay הינה **O(mn)**, כאשר n הוא מספר הרשומות השמורות בעץ, משום שנשמור n צמתים המכילים מספר קבוע של מצביעים (בן שמאלי ובן ימני), ואת תוכן הרשומה, לפי המקום הנדרש לה.

Gladiator:

יחידה זו מייצגת גלדיאטור. לכל גלדיאטור מוגדר המידע הבא:
- מזהה ייחודי ID (מספר שלם),
- מצביע למאמן (trainer) עבור גלדיאטור זה (מובטח כי לכל גלדיאטור יש מאמן אחד ויחיד),
- רמת הגלדיאטור אשר מיוצגת על ידי מספר שלם.
הפעולות היחידות עבור טיפוס זה הן פעולות get/set עבור כל אחד מהשדות. כמובן, קריאת ערך פרימיטיבי / דריסתו מתבצעת בסיבוכיות זמן **O(1)** במקרה הגרוע.
סיבוכיות מקום נדרשת עבור אחסון המידע של יחידה זו היא **O(1)**, משום שמדובר בשדות פרימיטיביים בלבד.

Squad:

יחידה זו הינה מבנה לאחסון נוח של גלדיאטורים. בתוך מבנה נתונים זה קיימים השדות הבאים:
- עץ splay המאחסן רשומות מסוג מצביע ל - gladiator, כאשר יחס הגודל הוא מעל ה – ID של הגלדיאטורים (המפתחות).
- עץ splay המאחסן רשומות מסוג מצביע ל - gladiator, כאשר יחס הגודל הוא מעל הרמה של הגלדיאטורים כמפתח ראשי, ומפתח משני (קרי, מיון משני של העץ) לפי המזהה הייחודי ID, במקרה של ריבוי גלדיאטורים באותה הרמה (סידור הפוך לפי ID: עבור שני גלדיאטורים באותה הרמה, האחד עם ה ID נמוך יותר ייחשב כגדול יותר),
- מספר הגלדיאטורים השמורים בעצים (הכנסה של גלדיאטור למבנה תכניסהו לשני העצים הנ"ל, ולכן מספר הגלדיאטורים בשני העצים זהה).
- מצביע לגלדיאטור ברמה הגבוהה ביותר השמור בעצים. במידה ויש מספר גלדיאטורים הנמצאים ברמה זו, המצביע יהיה לאחד בעל ה ID הגדול ביותר.
ננתח סיבוכיות מקום נדרשת עבור מבנה זה: מדובר בשני עצי splay עם רשומות מסוג מצביעים ל – gladiator ושני טיפוסים פרימיטיביים: ערך מספרי שלם, ומצביע לטיפוס gladiator. סיבוכיות מקום נדרשת עבור מצביע לטיפוס gladiator הינה O(1), וסה"כ עבור n גלדיאטורים נקבל:
O(1) + O(1) + O(n) + O(n) = **O(n)**.

Trainer:

יחידה זו מייצגת מאמן גלדיאטורים. לכל מאמן מוגדר המידע הבא:
- מזהה ייחודי ID (מספר שלם),
- מצביע ל - squad אשר מכיל את הגלדיאטורים הנמצאים תחת מאמן זה.
סיבוכיות מקום נדרשת עבור אחסון המידע של יחידה זו היא **O(1)**, משום שמדובר בשדות פרימיטיביים בלבד.

המערכת – Colosseum:

זהו מבנה הנתונים של **המערכת כולה** אשר תספק את הפונקציות הדרושות בתרגיל זה.
מבנה הנתונים זה מכיל את המידע הבא:
- squad של כל הגלדיאטורים הנמצאים במערכת,
- עץ splay של רשומות מטיפוס מצביעים למאמנים, המכיל את כל המאמנים במערכת (סדר עולה לפי ID).
ראשית, ננתח את סיבוכיות המקום הכוללת של המערכת:
בהינתן n גלדיאטורים ו – k מאמנים במערכת כולה, לפי הניתוח שלעיל, סיבוכיות מקום נדרשת עבור squad של n הגלדיאטורים תהיינה O(n). בנוסף, מתוך תכונת השייכות של גלדיאטור נתון למאמן אחד ויחיד, נקבל שסכום כל הגלדיאטורים הנמצאים בבעלותם של סך כל המאמנים הוא כל הגלדיאטורים במערכת – n. במילים אחרות: $\sum_{i=1}^{k} \mathrm{n_{trainerID}} = n$. לפי הרשום לעיל, סיבוכיות עץ splay של מאמנים במערכת תהיינה:

$$O(k + \sum_{i=1}^{k} \mathrm{n_{trainerID}}) = O(k+n)$$

במילים: יש במערכת k רשומות של מאמנים, וידוע שסדר גודל של **כל** הרשומות אשר בבעלות **כל** המאמנים יחדיו הוא O(n) מעצם בניית המערכת. כלומר, יש לנו: O(n+k).
סך הכל, קיבלנו שסיבוכיות המקום הנדרשת עבור המערכת כולה היא: O(n) + O(n+k) = **O(n+k)**.

מעל Colosseum מוגדרות הפעולות הרצויות בתרגיל. ננתח אותן ואת סיבוכיות זמן הריצה שלהן:
הערה 1: בניתוח להלן, נסמן את מספר הגלדיאטורים השונים השמורים במערכת ברגע נתון ב – **n**, מספר המאמנים יסומן ב – **k**, ומספר הגלדיאטורים השייכים למאמן המתאים ל – trainerID יסומן ב – $n_{trainerID}$.
הערה 2: בפונקציות יש התייחסות לשגיאות / מקרי קצה, לדוגמת הסרת איבר מעץ כשלא נמצא שם. משום שבמקרה של שגיאה אנו נפסיק מיד את ביצוע הפונקציה, אין טעם להתייחס למצבים אלו לצורך ניתוח סיבוכיות זמן גרידא, ועל כן נתייחס לאופן הריצה המתוכנן והתקין של כל הפונקציות.
הערה 3: ניתן לשים לב שעל מנת שנעמוד בדרישות הסיבוכיות של **פונקציות התרגיל בלבד**, מספיק לשמור במערכת עבור כל מאמן עץ בודד של גלדיאטורים מסודרים לפי הרמות (קרי, עץ רמות). לעומת זאת, במקרה מציאותי נרצה להכין מערכת שתתמוך בהוספה של פונקציות **טבעיות עבור המערכת** (פונקציות בעלות קשר מזהה של מאמן עם מזהה של גלדיאטור השייך לו), ולבצען בסיבוכיות זמן ריצה **מינימאלית**, בדומה לסיבוכיות הדרושה עבור קשר מאמן עם רמות הגלדיאטורים שלו בתרגיל. לשם כך שמרנו עבור כל מאמן squad שלם, על מנת **שהוספה של פונקציות כאלה בעתיד** תהייה נוחה, פשוטה, יעילה וטבעית, **ולא תדרוש שינוי** של כל מבנה הנתונים. כמובן, לפי ניתוח סיבוכיות המקום שהראנו, אנו עדיין עומדים בדרישות סיבוכיות המקום!

(1) *void* Init() – אתחול קולוסאום ריק.* נקרא לבנאי המערכת: יש להקצות מקום בזיכרון עבור:
- עץ המאמנים (יחד עם פונקציית השוואת ה – IDs),
- squad המערכת כולה.
סה"כ, עבור פעולות הקצאת זיכרון בלבד, נקבל כי **סיבוכיות זמן הריצה במקרה הגרוע היא O(1).**

(2) *StatusType AddTrainer(void *DS, int trainerID) – הוספת מאמן חדש לקולוסאום עם מזהה trainerID.* תחילה, ניצור מאמן חדש: יצירת מאמן נעשית במקרה הגרוע ב – O(1) לפי הרשום לעיל. נשתמש בפונקציית ההכנסה insert של עץ המאמנים על מנת להוסיף מצביע למאמן החדש שיצרנו: סיבוכיות זמן ריצה של פעולת insert על עץ המאמנים היא O(k) במקרה הגרוע, ולכן קיבלנו **שסיבוכיות זמן הריצה במקרה הגרוע היא O(k)** = O(1) + O(k).
השינוי החל על מבנה הנתונים בשל פעולה זו הוא הגדלת מספר הרשומות בעץ המאמנים. כמובן, המאמן החדש ישב בשורש העץ בגישה הבאה למבנה הנתונים בגלל תכונת ה – splay.

(3) *StatusType BuyGladiator(void *DS, int gladiatorID, int trainerID, int level) – הוספת גלדיאטור חדש לקולוסאום עם מזהה gladiatorID. הגלדיאטור נמצא ברמה level ושייך למאמן המתאים ל – trainerID.*
ראשית, נשים לב שנצטרך להוסיף מצביע לגלדיאטור החדש לשני squads שונים: האחד עבור המערכת כולה, והשני עבור המאמן הספציפי של הגלדיאטור. לצורך מציאת המאמן, ניצור מאמן זמני (כאמור, יצירת מאמן חדש נעשית במקרה הגרוע ב – O(1)), ואז נחפש אותו בעץ המאמנים באמצעות פעולת find על מנת לוודא שאכן קיים בעץ. הוספת הגלדיאטור: באמצעות המצביע למאמן שמצאנו, נקצה גלדיאטור חדש (נעשה ב – O(1) כפי שצוין), ואז נוסיף אותו ל – squad המערכת באמצעות פעולת insert, ובאופן זהה גם ל – squad של המאמן שלו. בכל squad, לאחר ההוספה, נמצא את הגלדיאטור ברמה הגבוהה ביותר השמור שם באמצעות findMax, שכן ייתכן והוא התעדכן לאחר ההוספה. בנוסף, נעלה את מספר הגלדיאטורים ב – 1.

- חיפוש המאמן בעץ המאמנים: O(log(k)) משוערך.
- יצירת גלדיאטור חדש: O(1) במקרה הגרוע.
- הכנסה ל – squad המערכת: פעולת insert על שני עצים בגודל n: 2 O(log(n)) = O(log(n)) משוערך.
- הכנסה ל – squad המאמן: פעולת insert על שני עצים בגודל $n_{trainerID}$:
2 $O(\log(n_{trainerID})) = O(\log(n_{trainerID}))$ משוערך. כמובן, $n > n_{trainerID}$ לפי אופן בניית המערכת.
- הפעלת findMax על squad המערכת: O(log(n)) משוערך.
- הפעלת findMax על squad המאמן: $O(\log(n_{trainerID}))$ משוערך.
- הגדלת מונה ב – 1: O(1) במקרה הגרוע.
סה"כ קיבלנו **שסיבוכיות הזמן המשוערכת של הפעולה היא:**

$$O(\log(k)) + O(1) + O(\log(n_{trainerID})) + O(\log(n)) + O(\log(n)) + O(\log(n_{trainerID})) = \mathbf{O(\log(n) + \log(k)) = o(\log(n) + k)}$$

השינוי החל על מבנה הנתונים בשל פעולה זו מתבטא במגוון צורות: ראשית, ארבעת העצים המפורטים התרחבו משום שהוספנו להם גלדיאטור אחד. כמובן, גלדיאטור זה נמצא כעת בשורש העץ בשל עקרון ה splay. בנוסף, שינינו מספר שדות ב squads המתאימים כמפורט: מספר הגלדיאטורים גדל ב – 1, והמצביע לגלדיאטור ברמה הגבוהה ביותר עשוי היה להתעדכן להיות הגלדיאטור הנוכחי.

(4) *StatusType FreeGladiator(void *DS, int gladiatorID) – שחרור גלדיאטור בעל המזהה gladiatorID מהקולוסאום.* נבצע חיפוש ב – squad המערכת באמצעות ה – ID הנתון. אם נמצא הגלדיאטור

– נסיר אותו מהמערכת בכך שנסיר קודם כל את כל צמתי העצים המחזיקים אותו, ולאחר מכן נשחרר את הגלדיאטור עצמו. לכן, נקרא ל – kick עם גלדיאטור זה עבור squad המערכת ועבור squad המאמן המתאים לגלדיאטור (גישה למאמן דרך המצביע השמור בגלדיאטור). בתוך כל squad, לאחר הסרת הצומת מהעץ וארגונו מחדש (חיבור הצמים ו splay), יש למצוא את הגלדיאטור ברמה הגבוהה ביותר במערכת, מאחר וייתכן שמחקנו אותו. נקרא ל – getMax בשני ה – squads הנ"ל ונעדכן את המצביע לגלדיאטור ברמה הגבוהה ביותר. בנוסף, נקטין ב – 1 את מספר הגלדיאטורים ב - squad. בתום תהליך זה נוכל לשחרר את הגלדיאטור עצמו.

- חיפוש ב – squad המערכת (שקול, כאמור, לחיפוש בעץ ה – IDs): O(log(n)) משוערך.
- מחיקת צומת הגלדיאטור מ – squad המערכת (הסרה משני עצים): 2O(log(n)) = O(log(n)) משוערך.
- מחיקת צומת הגלדיאטור מ – squad המאמן (הסרה משני עצים): $2O(\log(n_{trainerID})) = O(\log(n_{trainerID}))$.
- הפעלת findMax על squad המערכת: O(log(n)) משוערך.
- הפעלת findMax על squad המאמן: $O(\log(n_{trainerID}))$ משוערך.
- שחרור הגלדיאטור – מכיל רק שדות פרימיטיביים, ולכן: O(1) במקרה הגרוע.
- הקטנה מונה ב – 1: O(1) במקרה הגרוע.

סה"כ, קיבלנו ש**סיבוכיות זמן משוערכת של הפעולה היא:**

$$3 * O(\log(n)) + 2O(\log(n_{trainerID})) + O(1) = \boldsymbol{O(\log(n)) = o(\log(n) + k)}$$

השינוי החל על מבנה הנתונים בשל הוצאת הגלדיאטור הוא שינוי של השדות המתאימים להסרתו ב squads המתאימים לו (squad המערכת כולה ו – squad המאמן האישי שלו), וכן הוצאתו ממאגר הגלדיאטורים בעצים המתאימים להם. כמובן, מבנה הנתונים נשאר תקין: כל השדות הרלוונטיים עודכנו (מספר הגלדיאטורים + מציאת מקסימום חדש לפי הצורך) והעצים עדיין BSTs.

(5) *StatusType LevelUp(void *DS, int gladiatorID, int levelIncrease) – העלאת רמת הלחימה של הגלדיאטור בעל המזהה gladiatorID ב – levelIncrease רמות.* נחפש את הגלדיאטור הרצוי ב – squad המערכת כולה. אם הוא נמצא, נרצה לעדכן את הרמה שלו: פעם אחת ב – squad המערכת, ופעם נוספת ב – squad המאמן המתאים לו. לא נוכל להעלות ישירות את רמת הגלדיאטור שמצאנו, משום שאז מבנה עצי הרמות ב – squads המכילים אותו לא יהיו תקינים יותר. לכן, קודם כל נסיר אותו מ – squad המערכת, לאחר מכן ניגש למצביע השמור בו ע"מ לגשת למאמן שלו ולהסיר אותו גם מ – squad המאמן, ורק אז נוכל להעלות את הרמה שלו במספר הרצוי. לאחר שעדכנו את שדה זה, נוסיף את הגלדיאטור חזרה באותו האופן: נפעיל insert על squad המערכת, ופעם נוספת על squad המאמן.

- חיפוש ב – squad המערכת: O(log(n)) משוערך.
- הסרת הגלדיאטור מ – squad המערכת (כולל עדכון המקסימום): O(log(n)) משוערך.
- הסרת הגלדיאטור מ – squad המאמן (כולל עדכון המקסימום): $O(\log(n_{trainerID}))$ משוערך.
- עדכון שדה הרמה של הגלדיאטור: פעולת השמה בודדת: O(1) במקרה הגרוע.
- הוספה של הגלדיאטור ל – squad המערכת (כולל עדכון המקסימום): O(log(n)) משוערך.
- הוספה של הגלדיאטור ל – squad המאמן (כולל עדכון המקסימום): $O(\log(n_{trainerID}))$ משוערך.

סה"כ, קיבלנו ש**סיבוכיות זמן משוערכת של הפעולה היא:**

$$2\log(n) + 2O(\log(n_{trainerID})) + O(1) = \mathbf{O(\log(n))}$$

נוודא שמבנה הנתונים עדיין עומד בדרישות: השינוי היחיד בוצע לעצים במערכת, ואולי גם לגלדיאטור ברמה הגבוהה ביותר (דרך פעולת insert מחדש). כמובן, השינויים בעצי הרמות היו שינויים של הסרה אחת והכנסה אחת, ולכן אין דאגה לקיום הדרישות. מצד שני, בעצי ה – IDs בוצע עדכון של אחד משדות הגלדיאטור מבלי לשנות את מיקום הצומת שלו בעץ, כאשר העלנו את הגלדיאטור רמה. אך למרות זאת, העץ נשאר בעל מבנה תקין משום שרמת הגלדיאטור בעצים מסוג זה אינה מהווה יחס סדר – המפתחות בעצים אלו הם ה – IDs בלבד. מבנה הנתונים תקין.

(6) *StatusType GetTopGladiator(void *DS, int trainerID, int * gladiatorID) – החזרת ה – ID של הגלדיאטור ברמה הגבוהה ביותר הנמצא בבעלותו של מאמן עם ID נתון.* אם קיבלנו ID שלילי עבור המאמן אז נתבקש למצוא את הגלדיאטור ברמה הגבוהה ביותר (ואם יש כמה – אז ID הנמוך ביותר מבניהם). ב – squad של המערכת שמרנו שדה המכיל מצביע לגלדיאטור העונה על דרישה זו בכל רגע נתון. נחזיר את שדה ה – ID שלו. מדובר במספר קבוע של קריאת ערכים בלבד, ולכן **סיבוכיות worst-case O(1)**.

אחרת, אם קבלנו ID חיובי, נחפש בעץ המאמנים את המאמן המתאים ל – ID הדרוש, ובתוך ה squad של מאמן זה שמרנו, בדומה למקרה הקודם, שדה ובו מצביע לגלדיאטור ברמה הגבוהה ביותר בכל רגע נתון (ואם יש כמה, אז האחד עם ה – ID הנמוך מבניהם). נחזיר אותו. סך הכל: חיפוש find בעץ המאמנים ב –

worst-case יהיה O(k) (מקרה של שרוך), וקריאה והחזרה של השדה נעשית ב – O(1). סה"כ: **O(k)** **worst-case**.
השינוי היחיד החל על מבנה הנתונים שלנו הוא גלגול של הצומת בעץ המאמנים בו מוחזק המאמן אותו מצאנו לשורש העץ, והדבר לא נוגד את עקרונות העץ – זוהי פעילות תקינה. מלבד זאת, לא יתבצעו שינויים נוספים.

(7) *StatusType GetAllGladiatorsByLevel (void *DS, int trainerID, int **gladiators, int *numOfGladiator) – החזרת כל הגלדיאטורים של מאמן עם ID נתון, ממוינים לפי רמה וה – ID; מיון רמה בסדר יורד, ומיון משני לפי ID בסדר עולה.* בדומה לפונקציה מהסעיף הקודם (6), עבור ID שלילי נבצע את הפעולה עבור כל המערכת, ואחרת נבצעה עבור מאמן ספציפי. בשני המצבים מדובר במקרים סימטריים לחלוטין מבחינת האלגוריתם. עבור המערכת כולה: בתוך squad שמור לנו שדה המכיל את הערך n: מספר הגלדיאטורים במערכת. נקצה מקום למערך הנתון בגודל n. בנוסף, נשנה את הערך הרצוי שקבלנו כקלט להיות n. עד כה בצענו קריאה של כמות קבועה של שדות והקצאת מקום בזיכרון למערך: O(1). כעת נבצע סיור in-order בעץ הרמות שב – squad, ועבור כל רשומה בה נבקר, נשמור את ה –ID של הלגדיאטור ה - i במקום ה – n-i-1 במערך אותו הקצנו. העץ הוא עץ חיפוש בינארי מאופן הגדרתו, ולכן בתום תהליך זה נמלא את כל איברי המערך במצביעים לגלדיאטורים, ממוינים בסדר **הפוך** לסדר המיון של העץ (משום ששמירת הערכים אל תוך המערך התבצעה בסדר הפוך לסדר בו אנו עוברים על האיברים בסיור in-order): כלומר, במיון יורד לפי הרמה, ובמיון משני עולה לפי ID. זהו המערך הרצוי: נחזיר אותו. הכנסה למערך ב – O(1) במקרה הגרוע, וסיור in-order נעשה ב – **worst-case בסיבוכיות זמן O(n)**, כפי שראינו בהרצאות ובתרגולים (מעבר על n רשומות בעץ). במקרה השני, ID הוא חיובי ולכן נצטרך לבצע תהליך זהה עבור מאמן. כלומר, נלך לעץ המאמנים של המערכת ונבצע חיפוש search של המאמן הדרוש המתאים ל – ID הנתון. חיפוש בעץ splay נעשה בסיבוכיות worst-case של O(k), ואז נותר לבצע סיור in-order בעץ הרמות שב – squad של המאמן, ולחזור על התהליך הקודם. כלומר, נקבל תוספת של $O(n_{trainerID})$ ב – worst-case, כך שבסך הכל **סיבוכיות זמן worst-case במקרה זה היא $O(n_{trainerID} + k)$**.
לא קיים שינוי באף שדה במבנה הנתונים במהלך ביצוע הפונקציה, והשינוי היחיד למבני הנתונים במערך הוא לעץ המאמנים (במקרה בו ID>0) בגלל תכונת ה – splay. כמובן שהמערכת תישאר במצב תקין.

(8) *StatusType UpgradeGladiator(void *DS, int gladiatorID, int upgradedID) – עדכון מזהה ID של גלדיאטור בעל ID נתון.* מימוש הפעולה באופן כמעט זהה לחלוטין לפונקציה מסעיף (5): ההבדל היחיד במקרה זה הוא שנרצה לעדכן את שדה ה – ID של הגלדיאטור להיות המספר הנתון אותו קבלנו, במקום את שדה ה - level. הסיבה לכך שהאלגוריתם זהה לפונקציה מסעיף (5) הוא משום ששינוי ישיר של שדה בגלדיאטור הרצוי (רשומה בעצים) אשר משמש **מפתח** לסדר המיון, מבלי לשנות את מבנה העץ, תהרוס את תקינות העצים במערכת. כמובן, שינוי של שדה פרימיטיבי בודד על ידי פעולת השמה אחת תתבצע ב – O(1) במקרה הגרוע.
כתוצאה מכך, נקבל ניתוח **זהה לחלוטין** לפונקציה מסעיף (5) –
- חיפוש ב – squad המערכת: O(log(n)) משוערך.
- הסרת הגלדיאטור מ – squad המערכת (כולל עדכון המקסימום): O(log(n)) משוערך.
- הסרת הגלדיאטור מ – squad המאמן (כולל עדכון המקסימום): $O(\log(n_{trainerID}))$ משוערך.
- עדכון שדה הרמה של הגלדיאטור: פעולת השמה בודדת: O(1) במקרה הגרוע.
- הוספה של הגלדיאטור ל – squad המערכת (כולל עדכון המקסימום): O(log(n)) משוערך.
- הוספה של הגלדיאטור ל – squad המאמן (כולל עדכון המקסימום): $O(\log(n_{trainerID}))$ משוערך.
סה"כ, קיבלנו ש**סיבוכיות זמן משוערכת של הפעולה היא:**

$$2\log(n) + 2O(\log(n_{trainerID})) + O(1) = \mathbf{O(\log(n))}$$

(כמובן שאין צורך לעדכן שוב את המקסימום משום שהשינוי התבצע רק ל- ID של הגלדיאטור, אך חיפוש זה אינו גורע מן הסיבוכיות הרצויה, ומוטב להשתמש בפונקציות שכבר מימשנו). נוודא שמבנה הנתונים עדיין עומד בדרישות: השינוי בוצע לעצים במערכת, אך מדובר רק בפעולות הוצאה והכנסה, אשר שומרים על מבנה העץ כ splay tree תקין. כמובן, במצב הבא של מבנה הנתונים, השורש של כל אחד מארבעת העצים ששונו יהיה הגלדיאטור שעדכנו, מתוך עקרון splay.

(9) *StatusType UpdateLevels(void *DS, int stimulantCode, int stimulantFactor) – בהינתן מספר טבעי (אחרת מוחזרת שגיאה), עדכון רמתם של כל הגלדיאטורים במערכת שה – ID שלהם מתחלק במספר הנתון להיות מכפלת הרמה הנוכחית שלהם במספר הנתון.* ראשית, נבצע סיור in-order בעץ ה – IDs של squad ה – colosseum ובמהלכו נצבור במונה את סך כל הגלדיאטורים המקיימים את התנאי (כלומר, שה – ID שלהם מתחלק במספר הנתון). לצורך נוחות הכתיבה, נסמן מספר זה ב – m. אם m=0 אז סיימנו – אין צורך לשנות את רמתו של אף גלדיאטור במערכת. אחרת, נקצה שני מערכים של מצביעים לגלדיאטורים: מערך אחד עבור גלדיאטורים העונים על התנאי, בגודל m, ומערך משלים עבור גלדיאטורים שאינם עונים על התנאי, בגודל n-m.

כעת נבצע סיור in-order בעץ הרמות תחת אותו ה - squad, ובמהלכו נמלא את המערכים: עבור כל צומת בו נבקר, אם הגלדיאטור הנמצא בו מקיים את התנאי: נכניס את המצביע לגלדיאטור אל תוך מערך הגלדיאטורים המקיימים את התנאי, ואחרת נכניס אותו למערך הגלדיאטורים שאינם מקיימים (במקום המתאים, כמובן, כך שהמערכים יתמלאו).

בסוף תהליך זה, נקבל שני מערכים של מצביעים לגלדיאטורים אשר מובטח כי ממוינים לפי הרמה שלהם (מהקטן לגדול) כמיון ראשוני, ולפי ה – ID (מהגדול לקטן) כמיון משני. צעד הבא: נכפיל את רמותיהם של כל m הגלדיאטורים השמורים במערך הגלדיאטורים שמקיימים את התנאי במספר הנתון. נשים לב שהמערך נשאר ממוין לפי הסדר המקורי: שכן אם x < y אז גם cx < cy כאשר x,y מספרים ו – c > 0 קבוע. כעת נקצה מערך בגודל n של מצביעים לגלדיאטורים: זהו מערך שיכיל את המצביעים לכל הגלדיאטורים במערכת, ממוינים בסדר עולה לפי רמות (ומיון משני יורד לפי IDs). על מנת למלאו, נשתמש בפונקציה המוכרת (וידוע לשמצה) merge עם שני המערכים הקודמים כמקור, והמערך החדש בתור יעד. בסיום תהליך זה, נוכל לשחרר את שני המערכים הקודמים, וסיימנו עד כה עם מערך רצוי של המצב הנוכחי של המערכת.

נשים לב: הכפלת רמותיהם של m הגלדיאטורים **<u>הרסה</u> את סדר הרשומות בכל עצי הרמות במערכת!!!**
הסיבה לכך היא ששינוי רמותיהם של 0 < m < n גלדיאטורים (חלק מכולם) תעדכן את רמותיהם גם בתוך הצמתים בעצים אלו, אשר מסודרים כ – BSTs לפי יחס גודל כפי שהגדרנו קודם: לפי רמות (מיון ראשוני), ו – IDs (מיון משני). לאחר עדכון זה – לא מובטח שום סדר בעץ. מצד שני, נשים לב שעצי ה – IDs נשארים תקינים משום שלא חל עליהם סידור כלשהו לפי הרמות (כפוי שצוין קודם): המפתחות בעצים אלו הם ה – IDs בלבד. לעומת עצי ה – IDs, עבור עצי הרמות מתקיים שעדכון של רמותיהם של חלק מהגלדיאטורים לא מבטיח לנו יותר סדר הגיוני של עץ חיפוש בעץ, ולכן המערכת במצבה הנוכחי אינה תקינה ויש לשנות זאת.
לצורך כך, בשלב זה **נשחרר** את כל עצי הרמות במערכת כולה: ראשית, נשחרר את עץ הרמות ב – squad המערכת. יקרא ה - D'tor שישחרר את כל צמתי העץ. נעבור בסיור in-order על המאמנים בעץ המאמנים שבמערכת. עבור כל מאמן: נשחרר את עץ הרמות ב – squad שלו. יקרא ה – D'tor שישחרר את כל צמתי העץ. בסוף תהליך זה: לא קיימים יותר עצי רמות במערכת כולה.

בשלב זה נשתמש בפונקציית העזר **concatLinear** עליה הרחבנו בתיאור המחלקה SplayTree:
נעבור על כל n הגלדיאטורים שבמערך הממוין שבנינו, מההתחלה ועד הסוף. עבור כל גלדיאטור נבצע את התהליך הבא:

1) נגש לעץ הרמות ב – squad של ה – colosseum, ונבצע את פונקציית concatLinear עם הפרמטרים הבאים: עץ הרמות שלעיל, ומצביע לגלדיאטור הנוכחי.
2) ניגש לעץ הרמות ב – squad של המאמן של הגלדיאטור הנוכחי, דרך המצביע השמור באחד השדות שלו. כעת שוב נבצע את פונקציית concatLinear עם הפרמטרים הבאים: עץ הרמות שלעיל, ומצביע לגלדיאטור הנוכחי, כמו במקרה הקודם.

לאחר שסיימנו לעבור באופן זה על כל הגלדיאטורים במערך – נשחרר אותו. לסיכום, שחזרנו את כל עצי הרמות, וכעת הם תקינים (נזכור ששרוך יכול להיות splay tree, בתנאי שהוא עץ חיפוש בינארי תקין).

ננתח את סיבוכיות הזמן הכוללת, בניתוח worst-case:
- סיור in-order (פעמיים) על עץ ב – squad של ה - colosseum: O(n).
- n גישות למערך (סה"כ 2 פעמים במהלך ריצת הקוד): 2n*O(1) = O(n).
- merge של מערך בגודל m עם מערך בגודל n-m: O(m+(n-m)) = O(n).
- שחרור עץ הרמות ב – squad של ה – colosseum: O(n).
- סיור in-order בעץ המאמנים, ושחרור עץ הרמות השייך ל – squad של כל מאמן (ניזכר ששכום כל הגלדיאטורים השמורים עבור כל המאמנים הוא **בסה"כ** n: O(n+k).
- מעבר בלולאה על n איברים במערך, וביצוע concatLinear (פעמיים) עבור כל אחד: 2n*O(1) = O(n).
- הקצאת 3 מערכים, שחרור 3 מערכים, גישות למספר קבוע של שדות לאורך ריצת הפונקציה – O(1).
בסה"כ קיבלנו ש**סיבוכיות זמן הריצה של הפונקציה לפי ניתוח worst-case היא:**

$$O(n) + O(n) + O(n) + O(n) + O(n+k) + O(n) + O(1) = \boldsymbol{O(n+k)}$$

בנוסף, כפי שצוין, השינוי החל על מבנה הנתונים הוא מבנה עצי רמות: העצים בהם הגלדיאטורים מסודרים לפי יחס גודל ראשי של רמה, ויחס משני של IDs. כל העצים האלו בגישה הבאה למבנה הנתונים יהיו במבנה של "שרוך" ובו השורש הוא האיבר הגדול ביותר (הגלדיאטור בעל הרמה הגבוהה ביותר, ובניהם: עם ה - ID הנמוך ביותר). ניזכר שעץ splay הוא עץ המתקן את עצמו במהירות, מתוך תכונת splay אשר משנה את מבנה העץ בכל גישה, ולכן העץ יחזור להיות במצב כמעט-מאוזן במקרה הממוצע מהר מאוד עבור קריאות לפונקציות שונות על מבנה הנתונים (לדוגמה, הוספת גלדיאטור). נציין בנוסף שתוספת סיבוכיות מקום נוסף

עבור פונקציה זו היא O(2n) = O(n), שכן בכל איבר במערך נשמר מערך (טיפוס פרימיטיבי, O(1)), ובסך הכל יש לנו שלושה מערכים שמספר האיברים הכולל בהם הוא 2n. כמובן, הפונקציה לא פוגעת בעמידת הדרישות של המערכת על אודות דרישת **סיבוכיות המקום**: **O(n+k)** = O(n) + O(n+k).

(10) *void Quit(void** DS) – שחרור כל המידע בקולוסאום.* המידע השמור במערכת הוא הערכים עליהם מצביעים הרשומות בעצים (מצביעים לגלדיאטורים ומאמנים), וצמתי העצים המחזיקים רשומות אלו. על מנת לשחרר את **כל** המידע במערכת, ראשית נבצע סיור in-order על עץ המאמנים במערכת, ונשחרר את כל המאמנים בעץ זה. (נוכל לשחרר אותם מיד משום שלא מוחזקים מצביעים נוספים למאמנים במבנים נוספים במערכת). נשים לב שכל מאמן מחזיק squad: D'tor ל – squad המתאים יקרא בעת שחרור המאמן וישחרר את כל הצמתים בעצי ה - squad, ולאחר תהליך זה, נוכל לשחרר את עץ המאמנים כולו. צעד הבא: נבצע סיור in-order על עץ ה – IDs ב – squad המערכת, ובאופן דומה: נשחרר את כל הגלדיאטורים השמורים בעץ זה. שחררנו את המידע השמור בכל רשומות העצים, ולכן נוכל לשחרר את שאר הצמתים במערכת: נשחרר את squad המערכת ויקרא ה – D'tor אשר ישחרר את כל צמתי העצים.

ננתח את סיבוכיות הזמן הכוללת, בניתוח worst-case:
- סיור in-order בעץ המאמנים, ועבור כל מאמן מבצעים שחרור. עם שחרור של כל מאמן יקרא גם D'tor של ה – squad שלו המכיל $2n_{trainerID}$ צמתים ניזכר שבסה"כ, כל המאמנים מחזיקים n גלדיאטורים בכל עץ, כלומר 2n צמתים בסה"כ. כמובן, שחרור של כל עץ מתבצע ב – O(1), שכן מחזיק רק שדות פרימיטיביים (מצביעים), ולא אחראי על שחרור הרשומה. עברנו בסיור על k מאמנים, ולכן נקבל: O(2n+k) = O(n+k) במקרה הגרוע.
- שחרור עץ המאמנים: לעץ k צמתים שעלינו לשחרר, ולכן נקבל: O(k) במקרה הגרוע.
- סיור in-order בעץ ה – IDs של ה – squad הראשי במערכת, ועבור כל גלדיאטור מבצעים שחרור. שחרור של גלדיאטור מתבצע ב – O(1) ויש n גלדיאטורים בסה"כ, ולכן נקבל: O(n) במקרה הגרוע.
- שחרור squad המערכת: ל – squad יש 2n צמתים, ולכן נקבל: 2O(n) = O(n).
בסה"כ קיבלנו ש**סיבוכיות זמן הריצה של הפונקציה לפי ניתוח worst-case היא:**

$$O(n+k) + O(k) + O(n) + O(n) = \boldsymbol{O(n+k)}$$

בתום ריצת הפונקציה – המערכת שוחררה לחלוטין, ולא נותר מידע שהוקצה קודם לכן ולא שוחרר. לא ניתן יהיה להפעיל פונקציות על מבנה הנתונים יותר לאחר הפעלת פונקציה זו, שכן הוא לא קיים עוד. נצטרך לאתחל מבנה נתונים חדש על מנת שנוכל לבצע עליו את אחת הפעולות.

Our system - Colosseum:

Gladiator:

int id
int level
Trainer trainer → ...

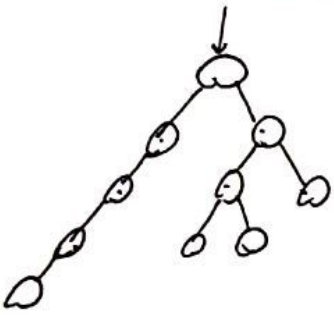
Trainer:

int id
Squad* squad

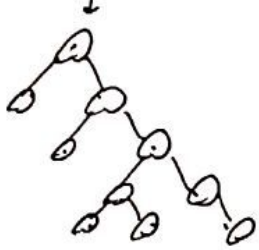
Squad:

int num_gladiators
Gladiator* best

SplayTree<Gladiators*>* id_tree

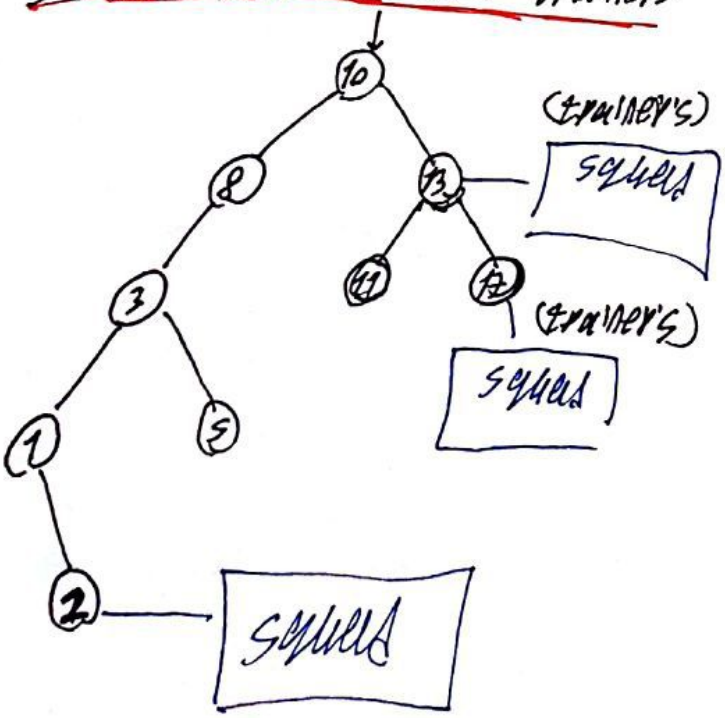


SplayTree<Gladiators*>* levels_tree



Colosseum:

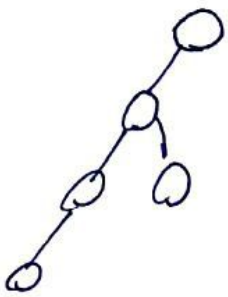
SplayTree<Trainers*>* trainers



Squad* main_squad

num: 5
Best:

id_tree:



level_tree: