**מבני נתונים תרגיל רטוב 2 – חלק יבש**

**מבנה הנתונים הפנימי**

**Min-Heap:** נממש ערימת מינימום כעץ כמעט שלם על ידי מערך דינאמי. נוסיף את השדות size (מס' האיברים בערימה) וmaxSize (גודל מקסימלי של המערך(.

**המתודות הפרטיות של הערימה:**

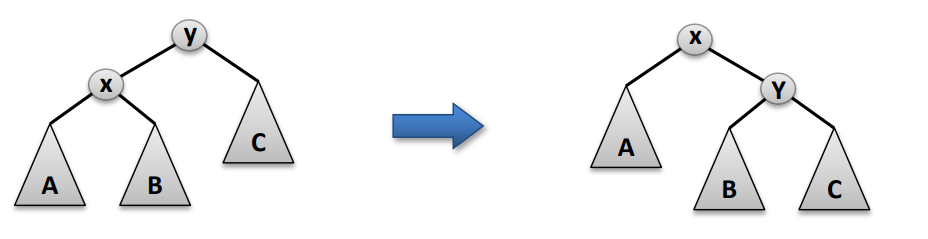
* **siftDown:** פונקציית עזר המקבלת אינדקס i המציין איבר בערימה. מבצעת את האלגוריתם אשר למדנו בתרגול: כל עוד האיבר בעל האינדקס i גדול מאחד מבניו, החלף אותו עם בנו המינימלי. ראינו כי סיבוכיות פעולה זו היא כאשר h גובה הקלט.
* **siftUp:** פונקציית עזר המקבלת אינדקס i המציין איבר בערימה. כל עוד האיבר בעל האינדקס ה-i קטן מאביו, החלף אותו עם אביו. ראינו כי סיבוכיות פעולה זו היא כאשר h גובה הקלט.
* **decKey:** פונקציה המקבלת אינדקס i המציין איבר בערימה וערך חדש val. אם הערך באיבר בעל אינדקס i קטן מ val לא מבצע דבר, אחרת משנה את הערך לval ומבצע siftUp עבור אינדקס i. סיבוכיות הפעולה במקרה הגרוע היא כאשר n מס' האיברים בערימה ובמקרה שלנו גודל המערך המנוצל בכל רגע, כלומר size.
* **resize:**  פונקציית עזר לשינוי גודל המערך. מקבל כפרמטר newSize, מקצאה מערך חדש בגודל זה ומעתיקה את העיבר מהמערך הישן (לאחר מכן משחררת אותו). בתרגול בנושא סיבוכיות משוערכת הוכחנו כי פעולת insert/remove ממערך דינאמי היא משוערך כאשר: מכפילים את גודל המערך כאשר הוא מתמלא כלומר size = maxSize, ומקווצים את המערך לחצי כאשר size <= maxSize \* 0.25.
* **DelMin:** פונקציה המוחקת את האביר המינימאלי מהערימה. מחליפה בין השורש (התא הראשון במערך) והעלה האחרון (התא האחרון במערך), משחררת את המידע בתא האחרון ולבסוף מבצעת siftDown על האינדקס של השורש החדש (כלומר 0). מאחר וsize קטן באחד, נבדוק האם אנחנו מנצלים לכל היותר רבע מגודל maxSize אם כן נקרא לresize עם maxSize\*0.5. ראינו בתרגול כי הסיבוכיות המשוערכת של הוצאה ממערך דינאמי (כולל הקטנת המערך) היא ולכן סיבוכיות פעולת DelMin היא משוערך.

**מתודות הממשק של הערימה:**

* **Heap:** בנאי המחלקה, מקבל מערך של n איברים. הבנאי מקצא מערך בגודל 2n, מעתיק את האיברים ומריץ את האלגוריתם של makeHeap מהתרגול/הרצאה (ראינו בשניהם) בסיבוכיות .
* **GetMin:** מחזיר את האיבר המינימלי בערימה. זו ערימת מינימום, לכן האביר המינימלי יופיע באינדקס 0 במערך. סיבוכיות פעולה זו היא .
* **Insert:** מוסיף איבר חדש לערימה: ראשית מציב אותו במקום הראשון הפנוי (כלומר כעלה אחרון) ולאחר מכן מבצע תיקון עם siftUp. הפונקציה בודקת האם size=maxSize כלומר האם צריך להגדיל את המערך אם כן קוראת לresize עם פרמטר maxSize\*2. סיבוכיות פעולה זו עבור הsiftUp של האיבר שהוספנו ומשוערך קריאה לresize (מערך דינאמי) ולכן בסה"כ משוערך.
* **DelKey:** פונקציה המוחקת את האיבר בעל האינדקס i מן הערימה. ראשית קוראת לdecKey ומשנה את הערך של האיבר בעל האינדקס ה-i להיות המינימלי. סיבוכיות פעולה זו . כעת, האיבר הרצוי נמצא כשורש הערימה (כלומר בתא ה0) נוכל לקרוא לDelMin בסיבוכיות של משוערך ולכן בסה"כ סיבוכיות פעולה זו היא משוערך.

**Ranked Splay Tree:** נעדכן את הSplayTree שיממשנו ברטוב 1, ונוסיף לכל Node את השדות הבאים: (score, sum , numNodes) כאשר sum הוא סכום הscore של כל הצמתים בתת העץ של Node (כולל הNode עצמו) וnumNodes ימנה את מס' המצתים בתת העץ (כולל הNode עצמו.(

**עדכון השדות הנוספים**

* **Insert:** בפעולת הכנסה של איבר חדש, ראשית בודקים אם הוא קיים בעץ. אם לא, במסלול החיפוש (המסלול אותו עוברים כדי להוסיף את האיבר), לכל איבר x נוסיף לשדה sum את הscore של האיבר החדש וכן נקדם את numNodes ב1, שכן האיבר החדש יתווסף לתת העץ של x. עדכון זה מתבצע ב על ידי פעולות אריתמטיות בלבד. נראה כעת כיצד מעדכנים את השדות בתיקונים:
* **Zig/Zag:** לאחר ההוספה, מבצעים Splay לאיבר שהוספנו כדי שיגיע לשורש של העץ. נראה כיצד עדכון מתבצע למשל עבור Zig:

תיקון לאחר הגלגול:

התתח עצים של A,B לא השתנו ולכן המידע השמור בשדות הנוספים תקין ומעודכן. כעת לכשעדכנו את Y נוכל לעדכן את השורש החדש, X:

*התיקונים הללו מתבצעים בסיבוכיות של לאחר סיום הגלגולים והם שומרים על נכונות העץ מבחינת השדות הנוספים.*

***הערה:*** *בתרגיל בית זה כלל לא השתמשנו בפעולת ה*Delete *של* SplayTree *ולכן אין צורך לעדכן את השדות בפונקציה זו או להוכיח את נכונותה. בסה"כ סיבוכיות הפעולה של* Insert *כולל הגלגולים והתיקונים לא השתנתה.*

*נוסיף את המתודות הבאות:*

* **SumFromK***: מקבלת כפרמטר מצביע ל-*Node *(שורש של תת עץ) וכן מספר* k *המציין אינדקס של איבר בתת העץ. הפונקציה תחזיר את ה*sum *של כל האיברים בעלי אינדקס גדול או שווה ל-*k. *נשתמש באלגוריתם למציאת איבר בעל דרגה* k *אשר ראינו בתרגול מס' 6 ובכל פעם שניגש לתת העץ השמאלי, נסכום את ה*sum *של תת העץ הימיני כולל הscore של השורש (שכן דרגת הצמתים האלו גדולה מ-*k*). סיבוכיות האלגוריתם למציאת איבר בעל אינדקס* k *היא , אנחנו מבצעים מספר סופי של פעולות אריתמטיות ולכן סיבוכיות פעולה זו היא .*
* **SumTopK*:*** *הפונקציה מקבלת כפרמטר* k *המציין אתכמות האיברים הגדולים עבורם אנחנו מעוניינים בסכום ה-*scores *שלהם. אם* k *גדול ממספר הצמתים בעץ או שהעץ ריק, נחזיר*-1. *אחרת, נקרא ל*SumFromK *עם פרמטר* numNodes(root)-k+1, *כלומר נרצה לסכום את הscores של כל הצמתים בעלי אינדקס גדול או שווה ל* numNodes(root)-k+1*(זה שקול לסכום של* k *הטובים ביותר).*

**List:** מימוש של רשימת Node בסיסית בעלת פונקציות הוספה וחיפוש בלבד.

* **Insert:** מקבל data חדש להוסיף. מייצר Node חדש ומוסיך אותו לראש הרשימה בסיבוכיות .
* **GetData**: מחזיר פוינטר לאיבר ברשימה, NULL אם לא קיים. סיבוכיות הפעולה במקרה הגרוע, מעבר על כל הרשימה.

**HashTable:** נממש טבלת ערבול אם פונקציית ערבול נוסיף את השדות currSize (מס' האיברים בטבלה) וmax\_size (גודל מקסימלי של המערך(. נפתור התנגשיות בשיטת Chain Hashing כך שבכל תא בטבלת הערבול יהיה מצביע לרשימה מקושרת מסוג List.

באמצעות מערכים דינאמיים נדאג לכך שמספר האיברים בטבלה יהיה בסדר גודל של max\_size, כלומר פקטור העומס יקיים . נשים לב כי פונקציית הערבול שבחרנו מפזרת באופן אחיד (ראינו בתרגול) ולכן תחת הנחת הפיזור האחיד הפשוט מספר האברים הממוצע שממופים לתא ה-i הוא (טענה מהתרגול).

לכן נסכם ונאמר כי סיבוכיות הזמן המשוערכת של פעולת הוספה לטבלה תהיה בממוצע על הקלט כפי שהראנו בשקופית 8 בתרגול על טבלאות ערבול.

**המתודות הפרטיות של טבלת הערבול:**

* **Resize:** פונקציית עזר לשינוי גודל המערך. הפונקציה מקצאה מערך חדש בגודל max\_size \* 2, ומפעילה את פונקציית הערבול החדשה על כל איבר בטבלת הערבול. הקריאה לפונקציה זו מתבצעת כאשר currSize = max\_size, היא נקראת אחת לn פעולות הוספה. בתרגול על מערכים דינאמיים ראינו כי הוספה למערך היא בסיבוכיות משוערכת. לאחר ההעתקה של הטבלה הישנה, הפונקציה משחררת את הטבלה הישנה.

**מתודות ההמשק של טבלת הערבול:**

* **בנאי:** הבנאי מקבל כפרמטר n (גודל התחלתי(. יצירת מערך של מצביעים לרשימות מקושרות ואיפוס כל התאים ל-NULL. סיבוכיות הפעולה: מעבר על כל המערך ואיפוסו.
* **Member:** פונקציה המחזירה פוינטר לאביר בתוך טבלת הערבול. NULL אם האיבר לא נמצא. הפונקציה מקבלת כפרמטר את האיבר אותו מחפשים וגם את המפתח שלו (מזהה ייחודי, למשל ID). ניגשת לתא במערך באמצעות פונקציית הערבול. אם המצביע לרשימה בתא זה הוא NULL נחזיר NULL שכן האיבר לא נמצא, אחרת נחפש ברשימה הקיימת (במקרה הכי גרוע נעבור על כל הרשימה, לפי הנחת הפיזור האחיד הפשוט בכל רשימה יש איברים) בסיבוכיות של בממוצע על הקלט.
* **Insert:** פונקציה המקבלת איבר והמפתח שלו ומוסיפה אותו לטבלת הערבול. אם התא המתאים (לפי פונקציית הערבול) ריק, כלומר אין עוד רשימה, ניצור חדשה ונוסיף את האיבר לרשימה. אחרת, נוסיף לראש הרשימה הקיימת ונגדיל את currSize ב1. אם currSize=max\_size נקרא לפונצית Resize. מציאת האינדקס המתאים לאיבר החדש היא בסיבוכיות (הפעלת פונקצית הערבול על המפתח). הוספת האיבר לרשימה היא גם כן ב(ההוספה היא תמיד לראש הרשימה). מאחר ואנו משתמשים במערכים דינאמיים, הסיבוכיות המשוערכת של פעולת ההוספה היא .

**מבנה הנתונים של המערכת**

מבנה הנתונים הכולל הוא Colosseum ומשתמש במחלקות GladGroup, Gladiator ובמבני הנתונים אשר הוצגו לעיל.

Gladiator:גלדיאטור הוא בעל השדות הבאים:

* **id:** תעודת הזהות של הגלדיאטור, מזהה ייחודי. ערך אי-שלילי.
* **score:** הציון של הגלדיאטור, ערך אי-שלילי.

שני גלדיאטורים ייחשבו שווים אם הם בעלי ID זהה (מזהה ייחודי). נשווה בין שני גלדיאטורים באופן הבא: glad1>glad2 אם הscore של glad1 גדול משל השני. אם לשניהם אותו ציון, אז הגלדיאטור הטוב מביניהם יהיה בעל הID הקטן יותר. בעזרת השוואה זו, נוכל ליצור עץ של גלדיאטורים ממוין לפי score ומיון משני לפי ID

GladGroup: קבוצת אימון של גלדיאטורים היא בעלת השדות הבאים:

* **lost:** flag בוליאני המציין האם קבוצת אימון זו הפסידה בקרב (True מציין כי הקבוצה הפסידה, ברירת מחדל False). קבוצה אשר הפסידה בקרב, אינה יכולה להתחרות יותר, אך היא עדיין קיימת – כלומר ניתן להוסיף אליה גלדיאטורים.
* **heapPointer:** מצביע לאיבר המציין קבוצה זו בערימת המינימום של הקבוצות המתחרות (נפרט עליה תחת Colosseum).
* **gladsTree:** מצביע לעץ של גלדיאטורים, כאשר המפתח הראשי הוא הציון של הגלדיאטור והמפתח המשני הוא הID. עץ זה ממוין לפי ההסבר עבור השוואת גלדיאטורים.

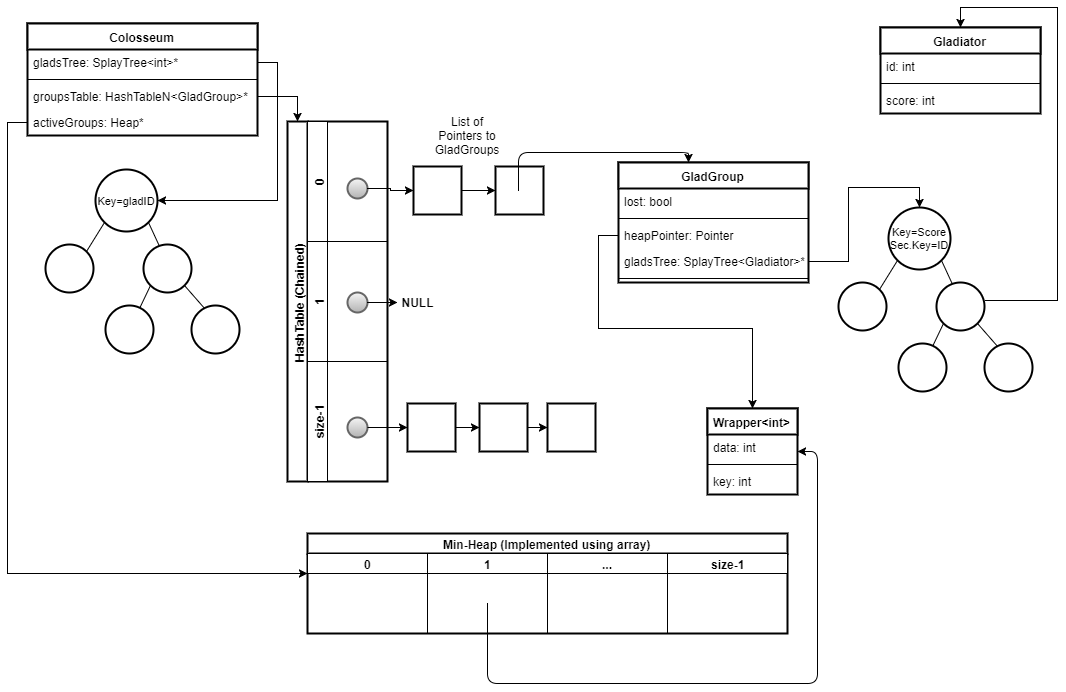
Colosseum: המערכת העיקרית שלנו היא קולוסאום, תשמור את השדות הבאים:

* **gladsTree:** מצביע לעץ מסוג int השומר את ה-ID של כל הגלדיאטורים במערכת. כאשר נרצה להוסיף גלדיאטור לקבוצה מסוימת, ראשית נבדוק האם הוא שייך למערכת, כלומר האם הוא הID שלו קיים בעץ זה. (אין שימוש בציון שלו, ולכן נשתמש בעץ מסוג int בלבד).
* **groupsTable:** מצביע לטבלת ערבול של קבוצות אימון(GladGroup). כאמור, טבלת עררבול זו היא מסוג Chain Hashing.
* **activeGroups:** מצביע לערימת מינימום השומרת את הID של הקבוצות המתחרות. המינימום בערימה זו היא הקבוצה שטרם נכבשה בעלת המזהה הקטן ביותר. כאשר קבוצה מפסידה, נדאג לעדכן את הערימה על ידי מחיקת הקבוצה שכן היא נכבשה.

מחיקת קבוצה מפסידה

נניח כי group1 (מסוג GladGroup) הפסידה וכעת צריך למחוק אותה מהערימה. בערימה, תעודות הזהות שמורות במחלקה עוטפת (Wrapper) בעלת שני שדות (Data,key). ה-ID של כל קבוצה הוא key ועליו מתבצע כלל הערימה. בשדה הData נשמור את האינדקס של הקבוצה הנ"ל במערך הערימה (כאמור מימשנו על ידי מערכים דינאמיים). בכל פעם שנשנה את מיקום הקבוצה המערך, נדאג לעדכן את האינדקס הנ"ל.

כפי שהסברנו מיקודם, לכל GladGroup יש שדה heapPointer שלמעשה מצביע על הWrapper הנ"ל. דרך המצביע נוכל לקבל את האינדקס של group1 בערימה ולמחוק אותה מהערימה על ידי קריאה לפונקציה DelKey הדורשת את האינדקס של האיבר אותו אנחנו רוצים למחוק.

**דיאגרמה לתיאור המערכת**

\*כפי שציינו מיקודם הdata בWrapper ישמור את האינדקס של הקבוצה בערימה, כך נוכל לקרוא לפונקציה DelKey.