<u>מבני נתונים שמומשו בתרגיל</u>

Trie – מחלקה גנרית של עץ מחרוזות, אשר מחזיק מחרוזות מספרים ובו נשתמש כדי לאחסן את המספרים המזהים של השחקנים. כל צומת מכיל מערך שהאינדקסים שלו מייצגים ספרות של המספר (האיבר הרצוי), והערך באינדקס נתון השונה מ-NULL מצביע לספרה הבאה במחרוזת, והוא מחווה על כך שהספרה הנוכחית במחרוזת היא האינדקס הנתון. **סוף המחרוזת** מיוצג ע״י ערך אמת בשדה הבוליאני של הצומת, ובצומת זה <u>יאוחסן מצביע למבנה הנתונים המיוצג ע״י המחרוזת</u> (במקרה זה המבנה הוא Player).

במחלקה מוגדר מבנה פנימי בשם node עם השדות הפנימיים הבאים:

- **children (מערך צמתי בנים המייצגים את האות הבאה במחרוזת)**
- **is_end_of_word (משתנה בוליאני שמסמן האם הצומת מהווה סוף מחרוזת)**
- **object (מצביע לטיפוס המיוצג ע״י מחרוזת המפתח, קיים בפועל רק בסוף המחרוזת)**

למחלקה השדות הבאים:

- שורש המחזיק מצביע ל-node
- מספר הצמתים בעץ

למחלקה המתודות הבאות:

1. **find** – מציאת מחרוזת בעץ.
2. **insert** – הכנסת איבר ומחרוזת חדשה לעץ.
3. **remove** – הסרת איבר ומחרוזת מהעץ.
4. **getSize** – החזרת מספר המחרוזות (איברים) בעץ.

UnionFind – מחלקה גנרית של המבנה הנלמד בכיתה Union-find, אשר מחזיק מערך קבוצות, ובו נשתמש כדי לאחסן את קבוצות השחקנים והמידע שלהן. כל קבוצה מכילה מידע פנימי ומצביע לקבוצת האב שאליה אוחדה קבוצה זו (כל עוד לא אוחדה, ערך המצביע הוא NULL). נציין כי איחוד הקבוצות במבנה מתבצע לפי גודל הקבוצות וכולל כיווץ מסלולים, ולפי משפט שראינו בהרצאה: <u>**סיבוכיות זמן ביצוע m פעולות $union/find$ חסומה ע״י O(mlog*(k)). *לפיכך הזמן המשוערך לכל פעולה חסום ע״י* O(log*(k)).**</u>

במחלקה מוגדר מבנה פנימי בשם group עם השדות הפנימיים הבאים:

- **מספר מזהה id**
- **מספר הקבוצות שאוחדו לתוך הקבוצה (כולל הקבוצה עצמה) size_of_up_tree**
- **מידע גנרי group_data**
- **מצביע לקבוצת האב parent**

למחלקה השדות הבאים:

- מערך קבוצות
- מספר הקבוצות במבנה

למחלקה המתודות הבאות:

1. **findId** – מציאת קבוצת האב שאליה משויכת הקבוצה.
2. **findData** – מציאת המידע של הקבוצה.
3. **setData** – השמת מידע חדש לקבוצה.
4. **makeUnion** – איחוד שתי קבוצות.
5. **getGroupSize**– החזרת מספר הקבוצות שאוחדו לתוך הקבוצה (כולל הקבוצה עצמה)

LevelsTree – מחלקה של עץ דרגות, שבו נשתמש בהמשך לצורך שמירת נתוני רמות השחקנים בסדר ממוין לפי רמה. המפתח של כל צומת בעץ הוא רמה כלשהי והמידע שלה הוא מספר השחקנים ברמה זו. כפי שלמדנו בכיתה, בכל צומת יוחזק מידע נוסף (במקרה זה - מספר השחקנים שברמות של תת העץ של הצומת הנוכחי וסכום הרמות של כל השחקנים בתת העץ, ערך זה יחושב כסכום המכפלות כל רמה של תת העץ במספר השחקנים שנמצאים בה). את המחלקה נממש כמחלקה יורשת של עץ ה-AVL הגנרי שבנינו בתרגיל הבית הקודם, ונדרוס מתודות מסוימות בעץ ה-AVL כדי להבטיח שמירה עדכנית של המידע הנוסף של כל צומת.

במחלקה מוגדר מבנה פנימי בשם node עם השדות הפנימיים הבאים:

- **מפתח המייצג רמה Level**
- **מידע מטיפוס PlayersCounters, שהוא מחלקה עם 3 שדות פנימיים:**
  1. **מספר השחקנים שנמצאים בדרגה זו self_counter**
  2. **מספר השחקנים שנמצאים בכל הרמות של תת העץ sub_tree_counter**
  3. **סכום הרמות של כל השחקנים בתת העץ sub_tree_levels_sum**

למחלקה השדות הבאים:

- שורש המחזיק מצביע ל-node
- מספר הצמתים בעץ
- מספר השחקנים ברמה 0 (הרמה ההתחלתית)

למחלקה המתודות הבאות:

1. **find** – מציאת מחרוזת בעץ.
2. **insert** – הכנסת רמה חדשה לעץ.
3. **remove** – הסרת רמה קיימת מהעץ.
4. **update** – עדכון רמה של שחקן, ע"י עדכון נתוני הרמה הקודמת והחדשה בעץ.
5. **merge** – מיזוג עץ רמות נתון לעץ הנוכחי.
6. **getNumOfPlayersInRange** – מציאת מספר השחקנים בתחום הרמות הנתון.
7. **getAverageLevel** – מציאת הרמה הממוצעת של m השחקנים בעלי הרמה הגבוהה ביותר.
8. **getNumberOfPlayers** – החזרת מספר השחקנים הכוללת של כל רמות העץ.

<u>טיפוסים</u>

PlayersManager – מחלקת האב של כלל מבני הנתונים, עם השדות הפנימיים הבאים:

1. **Groups** – מבנה Union-Find לניהול מידע הקבוצות, שצומת בו מורכב ממספר הקבוצה (מפתח) ומטיפוס GroupData (מצביע shared_ptr לטיפוס קבוצה – זהו הערך).
2. **PlayersIDs** – עץ מחרוזות Trie שמכיל את המספרים המזהים של השחקנים <u>במשחק</u>.
3. **LevelsTree** – עץ רמות LevelsTree שמכיל את כל רמות השחקנים <u>במשחק</u>, ומאגד בכל רמה את המידע הנוסף אודות מספר השחקנים ברמה זו ואשר קטנים מרמה זו.
4. **PlayersLevelsByScore** – מערך באורך Score של עצי דרגות דומים ל-PlayersLevels, כך שבכל תא יוחזק עץ עם נתוני רמות המתייחס רק לשחקני <u>המערכת</u> בעלי ניקוד השווה לאינדקס התא.
5. **num_of_groups** – מספר הקבוצות במערכת, המתקבל בעת אתחול המערכת.
6. **scale** – טווח התוצאות של ניקוד השחקן, המתקבל בעת אתחול המערכת.

GroupData - מחלקה עם השדות הפנימיים הבאים:

1. **id** – מספר הקבוצה.
2. **LevelsTree** – עץ רמות LevelsTree שמכיל את כל רמות השחקנים <u>בקבוצה</u>, ומאגד בכל רמה את המידע הנוסף אודות מספר השחקנים ברמה זו ואשר קטנים מרמה זו.
3. **PlayersLevelsByScore** – מערך באורך Score של עצי דרגות דומים ל-PlayersLevels, כך שבכל תא יוחזק עץ עם נתוני רמות המתייחס רק לשחקני <u>הקבוצה</u>, בעלי ניקוד השווה לאינדקס התא.

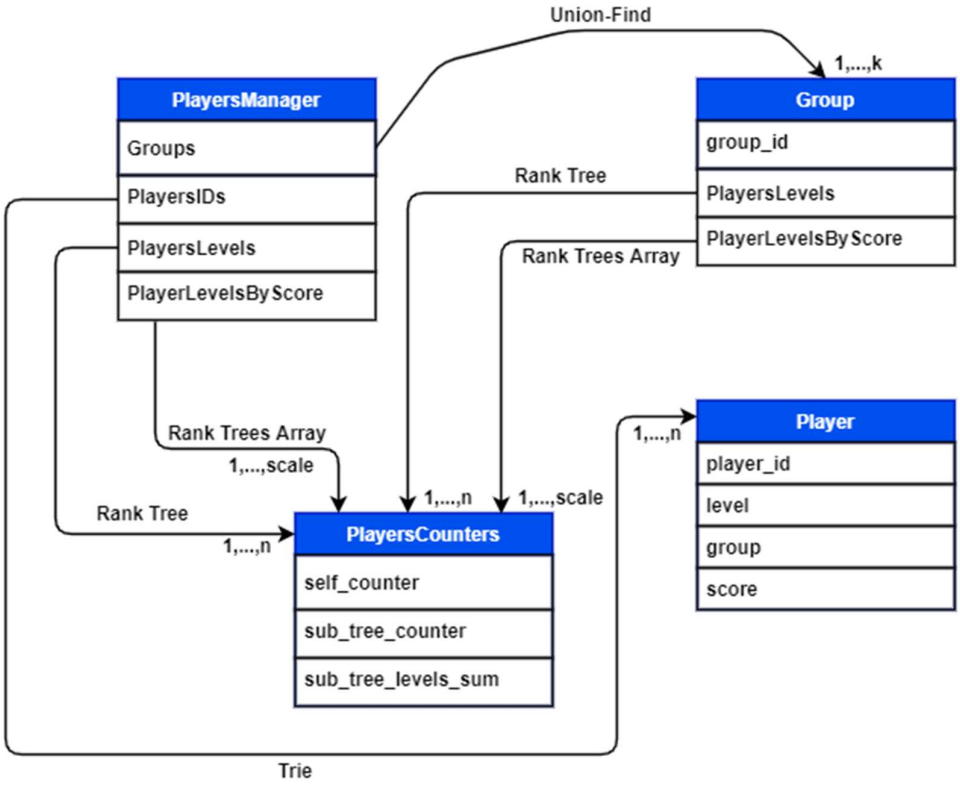
Player – מחלקה עם השדות הפנימיים הבאים:

1. **id** – מספר השחקן
2. **level** – שלב השחקן
3. **group_id** – מספר קבוצת השחקן
4. **Score** – ניקוד השחקן

למחלקה המתודות הבאות:

1. **getId** – החזרת את מספר השחקן.
2. **getLevel** – החזרת את רמת השחקן.
3. **setLevel** – השמת רמה חדשה לשחקן.
4. **getGroupId** – החזרת מספר הקבוצה של השחקן.
5. **setGroupId** – השמת מספר קבוצה חדש לשחקן.
6. **getScore** – החזרת ניקוד השחקן.
7. **setScore** – השמת ניקוד חדש לשחקן.

תרשים להמחשת המבנה



מימוש הפונקציות כולל ניתוח סיבוכיות

void **init**(int k, int scale)*

במקרה הגרוע ***O(k)***

- יצירת המבנה PlayerManager, ואתחול השדות על ידי:

  - יצירת מבנה UnionFind
  - עץ מחרוזות ריק PlayersIDs
  - עץ דרגות ריק PlayersLevels למערכת ולכל קבוצה
  - אתחול השדות scale ו-num_of_groups

כל העצים מאותחלים ללא איברים ולכן מדובר במספר קבוע של פעולות לכל המבנה O(1),
UnionFind מאותחל עם k קבוצות ולכן מדובר בסך הכל ב-O(k).

****

StatusType ***mergeGroups****(void *DS,intGroupID1, intGroupID2)*

O(log*(k)+n) משוערך, בממוצע על הקלט ,כאשר k הוא מספר הקבוצות ו-n מספר השחקנים בשתי הקבוצות המאוחדות.

- בדיקות קלט O(1)
- מיזוג שני העצי הרמות של הקבוצות (שגודלם חסום ע"י n) באמצעות פונקציית המיזוג של עץ ה-AVL שמימשנו בתרגיל הקודם, ובנוסף עדכון נתוני הרמות באמצעות סיור postOrder O(n)
- מיזוג שני מערכי עצי הרמות לפי ניקוד (גודל המערכים חסום ע"י קבוע וגודל העצים חסום ע"י n) של הקבוצות, באמצעות מעבר איטרטיבי על כל תא במערך וקריאה לפונקציית המיזוג הנ"ל O(n)
- איחוד שתי הקבוצות במבנה UnionFind O(log*(k)) משוערך

****

StatusType ***addPlayer****(void *DS, int PlayerID, int GroupID, int score)*

O(log*(k)) משוערך, בממוצע על הקלט ,כאשר k הוא מספר הקבוצות

- בדיקות קלט O(1)
- יצירת שחקן חדש O(1)
- הכנסת השחקן החדש לעץ המחרוזות PlayersIDs O(1)
- קידום מונה השחקנים ברמה 0 של עץ הרמות ומערך עצי הרמות לפי ניקוד <u>במערכת</u> O(1)
- מציאת הקבוצה בעץ הקבוצות O(log*(k)) משוערך
- קידום מונה השחקנים ברמה 0 של עץ הרמות ומערך עצי הרמות לפי ניקוד <u>בקבוצה</u> O(1)

****

StatusType ***RemovePlayer*** *(void *DS, int PlayerID)*

O(log*(k)+log(n)) משוערך, בממוצע על הקלט ,כאשר k הוא מספר הקבוצות ו- n הוא מספר השחקנים הכולל במשחק כרגע

- בדיקות קלט O(1)
- מציאת השחקן בעץ המחרוזות PlayersIDs O(1)
- הוצאת השחקן מעץ המחרוזות O(1)
- מחיקת השחקן מעץ הרמות וממערך עצי הרמות לפי ניקוד <u>במערכת</u> O(log n)
- מציאת הקבוצה של השחקן בעץ הקבוצות O(log*(k)) משוערך
- מחיקת השחקן מעץ הרמות וממערך עצי הרמות לפי ניקוד <u>בקבוצה</u> O(log n)

****

StatusType ***increasePlayerIDLevel****(void *DS,int PlayerID, int LevelIncrease)*

O(log*(k)+log(n)) משוערך, בממוצע על הקלט ,כאשר k הוא מספר הקבוצות ו- n הוא מספר השחקנים הכולל במשחק כרגע

- בדיקות קלט O(1)
- מציאת השחקן בעץ המחרוזות PlayersIDs O(1)
- עדכון נתוני הרמות בעץ הרמות ובמערך עצי הרמות לפי ניקוד <u>במערכת</u> O(log n)
- מציאת הקבוצה של השחקן בעץ הקבוצות O(log*(k)) משוערך
- עדכון נתוני הרמות בעץ הרמות וממערך עצי הרמות לפי ניקוד <u>בקבוצה</u> O(log n)

* **<u>סעיף בונוס</u>:** במימוש שהצגנו, הפעולה הנ"ל דורשת סיבוכיות של O(log*(k)+log(n)) משוערך ללא תלות בקלט, היות שפעולות העדכון בעצים עוברות על מסלול מסוים בעץ מהשורש לעלה – סיבוכיות O(log n). בנוסף, סיבוכיות מציאת השחקן בעץ המחרוזות היא O(1) בשל העובדה שאורך המחרוזת חסום ע"י קבוע, וסיבוכיות פעולת ה-find במבנה UnionFind היא O(log*(k)) משוערך. לכן אם נסכום את כלל הפעולות, נקבל בסך הכל סיבוכיות זמן של O(log*(k)+log(n))

****

StatusType ***changePlayerIDScore****(void *DS, int PlayerID, int NewScore)*

O(log*(k)+log(n)) משוערך, בממוצע על הקלט ,כאשר k הוא מספר הקבוצות ו- n הוא מספר השחקנים הכולל במשחק כרגע

- בדיקות קלט O(1)
- מציאת השחקן בעץ המחרוזות PlayersIDs O(1)
- עדכון נתוני הרמות בתאים המייצגים של הניקוד הישן והחדש בעצי הרמות לפי ניקוד <u>במערכת</u> O(log n)
- מציאת הקבוצה של השחקן בעץ הקבוצות O(log*(k)) משוערך
- עדכון נתוני הרמות בתאים המייצגים של הניקוד הישן והחדש בעצי הרמות לפי ניקוד <u>בקבוצה</u> O(log n)
- עדכון ניקוד השחקן O(1)

****

StatusType ***getPercentOfPlayersWithScoreInBounds****(void *DS, int GroupID, int score, int lowerLevel, int higherLevel, double* players)*

O(log*(k)+log(n)) משוערך ,כאשר k הוא מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.

- בדיקות קלט O(1)
- מציאת הקבוצה הרלוונטית בעץ הקבוצות O(log*(k)) משוערך
- מציאת מספר השחקנים שרמתם נמוכה מ-lowerLevel בעץ הרמות (רקורסיבית), מספר השחקנים שרמתם קטנה מ-higherLevel בעץ הרמות (רקורסיבית), חישוב ההפרש והחזרת האחוז בטווח המבוקש מתוך כלל השחקנים בעץ O(log n)

****

*StatusType **averageHighestPlayerLevelByGroup**(void *DS, int GroupID, int m, double *avgLevel)*

O(log*(k)+log(n)) משוערך, כאשר *k* הוא מספר הקבוצות ו-*n* הוא מספר השחקנים הכולל במשחק כרגע.

- בדיקות קלט O(1)
- מציאת הקבוצה הרלוונטית בעץ הקבוצות O(log*(k)) משוערך
- מציאת הרמה בעץ בה נמצא השחקן ברמה הנמוכה ביותר מבין m השחקנים הגבוהים. נאתחל משנתה סכום. כעת במסלול החיפוש מהשורש מטה, בכל פעם שתתבצע פנייה שמאלה – נוסיף את סכום הרמות של השחקנים בתת העץ של הבן הימני ברמה הנוכחית, ובנוסף את סכום רמות השחקנים ברמה הנוכחית. כאשר ימצא השחקן המבוקש, נוסיף לסכום את רמותיהם של מספר השחקנים שנדרש להשלים ל-m. עם קבלת הסכום, תחושב הרמה הממוצעת ותוחזר O(log n)

****

*void **Quit** (void **DS)*

סיבוכיות מקום – ***O(n+k)*** במקרה הגרוע, כאשר *n* הוא מספר השחקנים ו-*k* הוא מספר הקבוצות

- נהרוס את עץ המחרוזות, כלל עצי הרמות, מערכי עצי הרמות לפי ניקוד וה-*UnionFind*.
- הרס של עץ המחרוזות דורש מעבר באורך קבוע (לפי אורך המחרוזת) כדי למחוק שחקן ומעבר זה יהיה על *n* השחקנים במערכת, לכן הסיבוכיות תהיה *O(n)*.
- הרס עץ הרמות של <u>המערכת</u> דורש מעבר על כל הצמתים (רמות כל השחקנים במערכת), שמספרן הוא לכל היותר כמספר השחקנים *n* (לא יתכנו יותר רמות מאשר שחקנים). לכן סיבוכיות הרס העץ תהיה *O(n)*.
- מערך עצי הרמות לפי ניקוד של <u>המערכת</u> הוא באורך החסום ע"י גודל קבוע, וסכום הצמתים (רמות השחקנים) של כל העצים יחד במערך הוא לכל היותר כגודל מספר השחקנים *n* (שכן לא יתכנו יותר רמות מאשר שחקנים), לכן הרס העצים דורש מעבר על *n* צמתים לכל היותר, ומכאן שסיבוכיות הרס המערך תהיה *O(n)*.
- הרס של המבנה *UnionFind* כולל מעבר על *k* הקבוצות שאותחלו, עבורו ידרשו *O(k)* פעולות. בנוסף, לכל קבוצה נדרש הרס של עץ הרמות הכללי שלה, והרס של מערך עצי הרמות לפי ניקוד:

- ישנם k עצי רמות כלליים קבוצתיים (עץ לכל אחת מ-k הקבוצות), וסכום הצמתים (רמות השחקנים) של כל העצים הללו יחד הוא לכל היותר כגודל מספר השחקנים n (לא יתכנו יותר רמות מאשר שחקנים), לכן הרס העצים דורש מעבר על n צמתים לכל היותר, ומכאן שסיבוכיות הרס כל העצים יחד תהיה $O(n)$.
- בנוסף ישנם k מערכים של עצי רמות לפי ניקוד קבוצתיים (מערך לכל אחת מ-k הקבוצות) ואורכם חסום ע"י גודל קבוע. נשים לב שסכום הצמתים (רמות השחקנים) של כל העצים בכל המערכים יחד הוא לכל היותר כגודל מספר השחקנים n (שכן לא יתכנו יותר רמות מאשר שחקנים), לכן הרס העצים דורש מעבר על n צמתים לכל היותר, ומכאן שסיבוכיות הרס כלל המערכים תהיה $O(n)$.

לכן ההרס של מבנה ה-*UnionFind* יהיה בסיבוכיות כוללת $O(n+k)$.

****

ניתוח סיבוכיות מקום של המבנה: משיקולי הטענות שהוצגו בניתוח הסיבוכיות של פונקציית ה-*Quit*, נסיק כי סיבוכיות המקום של עץ הרמות של המערכת היא $O(n)$, וגם זו של מערך עצי הרמות לפי ניקוד של המערכת היא גם $O(n)$. סיבוכיות המקום של עץ הרמות של כל הקבוצות יחד היא $O(n)$, וזו של מערכי עצי הרמות לפי ניקוד של כל הקבוצות יחד היא גם $O(n)$. בנוסף לטענות אלה, במבנה ה-*UnionFind* קיימות k קבוצות, הדורשות סיבוכיות מקום $O(k)$, לכן סיבוכיות המקום של מבנה זה היא בסך הכל $O(n+k)$, וסיבוכיות המקום של המבנה הכולל היא $O(n+k)$.