תרגיל רטוב 1

מבנה נתונים

מגישות:
נתלי אטינגין 209051275
טל פלג 316276955

נפרט על מבני הנתונים שיהיו לנו:

PlayersManager	AVL_groups <group>AVLTree AVL_all_players_id <player>AVLTree AVL_all_levels <level>AVLTree int num_of_players
group	int group_id int num_of_players players_level <level>AVLTree
player	int player_level int player_id group_info_ptr <groupInfo>std::shared_ptr
level	int player_id group_info_ptr <groupInfo>std::shared_ptr int level
groupInfo	group* group_nt

הסבר כללי על מבנה הנתונים:
playersManager:
בו ינוהל המשחק. יכיל את הפרמטרים הבאים: מספר השחקנים בכל המערכת, עץ של כל השחקנים שבו כל צומת הוא מסוג player ממויינים לפי player_id, עץ של כל השחקנים שבו כל צומת הוא מסוג level ממויינים לפי הlevel, ועץ של קבוצות, בו כל צומת מסוג group.

בכל צומת של group, יהיה לנו מידע על מספר השחקנים בקבוצה הזו, ועץ של כל השחקנים בקבוצה ממויינים לפי level – בו כל צומת הוא מסוג level.

על מנת לעמוד בדרישות הסיבוכיות, שמרנו גם מצביע חכם מסוג groupinfo בעץ השחקנים הכללי ובעץ הlevelים שבכל קבוצה על מנת לא לפגוע במבנה במהלך הסרת צמתים. דוגמא בציור למבנה הנתונים שלנו יבוא בתמונה שבדץ הבא.

<u>הסבר על העץ שיצרנו:</u>

הפרמטרים שיש לעץ שלנו הם:

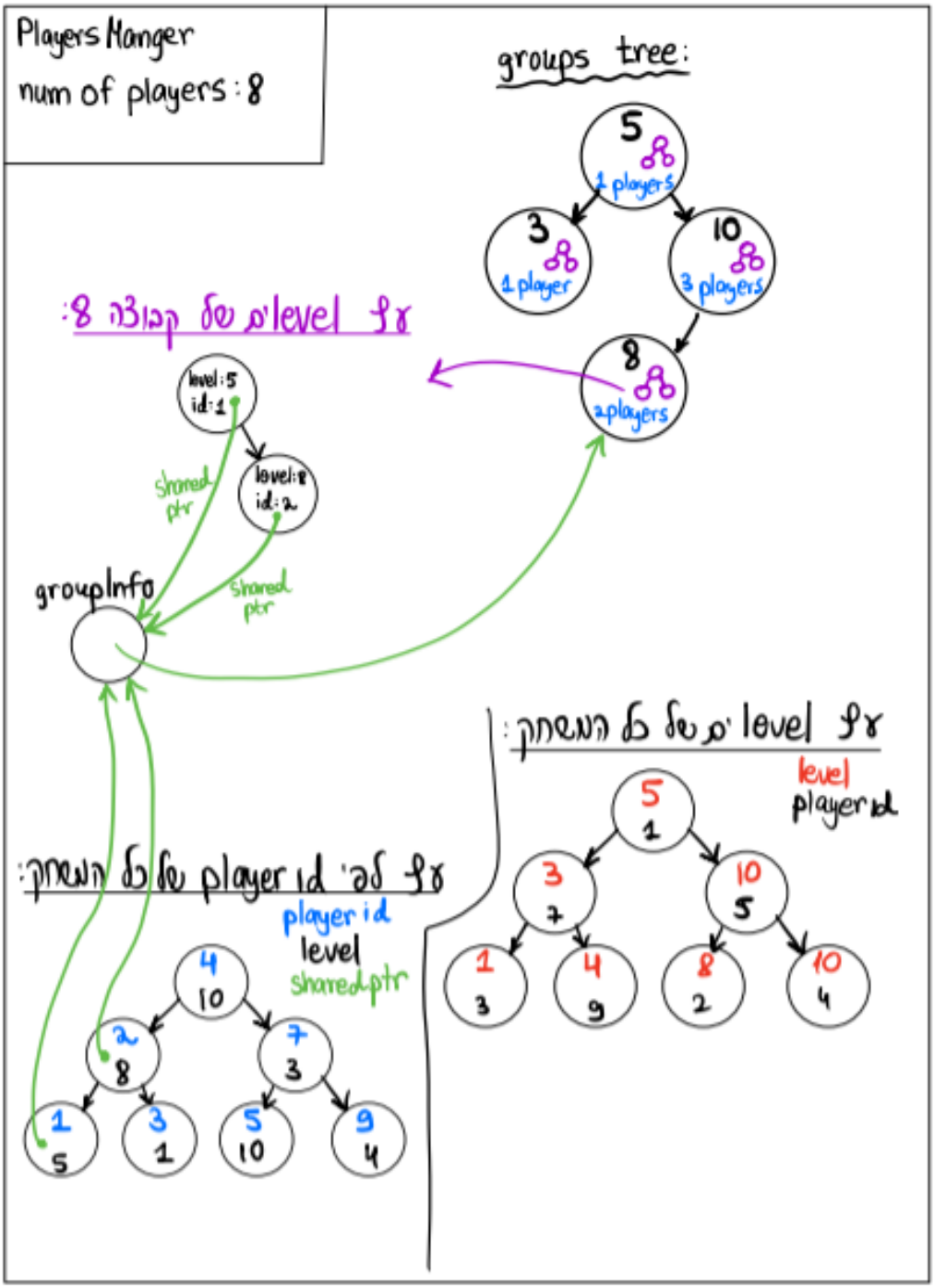
Int tree size	Node* max	Node* root

כל node יכיל את המשתנים הבאים:

Int height	Node* right	Node* left	T data

כאשר T הוא template כדי שנוכל ליצור עץ עם data מסוגים שונים. בעץ נשמור את השורש, מצביע לצומת המקסימלי ואת מספר הצמתים בעץ.

נבצע את החיפוש והמיון שלנו בעץ על פי הvalue, לכל אובייקט מסוג T יהיה אופרטור > שעל פיו נבצע את המיון.


Players Manger
num of players: 8
groups tree:
5
1 players
3
1 player
10
3 players
8
2 players
עץ level-ים של קבוצה 8:
level: 5
id: 1
level: 8
id: 2
shared ptr
shared ptr
groupInfo
עץ level-ים של כל המשתמש:
level
playerId
5
1
3
7
10
5
1
3
4
9
8
2
10
4
עץ לפי playerId של כל המשתמש:
playerId
level
sharedptr
4
10
2
8
7
3
1
5
3
1
5
10
9
4

:Void* Init()

ניצור class מסוג PlayersManager, נאתחל את כל העצים לעצים ריקים, שזה נעשה בסיבוכיות של O(1) ואת מספר השחקנים של השחקנים ל-0.
אתחול העצים – node* root, Node* max יאותחלו לnullptr וtree size יאותחל ל0.
מספר פעולות לאתחול עץ הוא קבוע, וכן אתחול מספר השחקנים ולכן בסה"כ נקבל O(1) בסיבוכיות.

:StatusType AddGroup(void *DS, int GroupID)

ראשית ניצור אובייקט מסוג group– כפי שפירטנו למעלה נאתחל את העץ שלו במספר פעולות קבוע, את num of players ל-0 ואת מספר הקבוצה למספר שקיבלנו. נוסיף אותו לעץ הgroups בסיבוכיות של O(logk) כפי שנלמד בהרצאה, בפעולת הinsert של עץ, כאשר k זה מספר הקבוצות.
בסה"כ נקבל שסיבוכיות הוספת קבוצה למשחק היא O(logk).

:StatusType AddPlayer(void *DS, int PlayerID, int GroupID, int Level)

תחילה ניצור פוינטר מסוג group שיצביע על הקבוצה עם הgruopID שקיבלנו. אם הgroup לא קיים נזרוק שגיאה. לאחר מכן ניצור shared_ptr מסוג player_info ונאתחל את השדות שלו עם המצביע לgroup שיצרנו. ניצור משתנה חדש מסוג Player,נאתחל את הplayerID שלו ואת הlevel שלו עם השדות שקיבלנו ואת הplayer_info נעדכן עם הshared_ptr שיצרנו. כעת נכניס אותו לעץ השחקנים הכללי בסיבוכיות של O(logn) כפי שלמדנו בהרצאה, בפעולת הinsert של העץ כאשר n הוא מספר השחקנים הכללי בכל המשחק. כעת ניצור 2 משתנים מסוג level. את המשתנה הראשון נאתחל עם הPlayerID והlevel שקיבלנו. את הshared_ptr שלו נאתחל לnullptr ונכניס אותו לעץ הlevel-ים הכללי, בסיבוכיות של O(log n) כפי שלמדנו בהרצאה עם פעולת insert של העץ כאשר n הוא מספר השחקנים בכללי.
נעדכן את המצביע של השחקן המקסימלי (פעולה פנימית של העץ) בסיבוכיות של O(logn). את המשתנה השני מסוג level נאחל עם הPlayerID והlevel שקיבלנו ועם הshared_ptr שיצרנו. נכניס אותו לעץ הlevel-ים של הקבוצה הרלוונטית בעזרת הפוינטר של הקבוצה שיצרנו, בסיבבויות של O(logn) כפי שלמדנו בהרצאה, עם פעולת הinsert של העץ כאשר n הוא מספר השחקנים בקבוצה. נעדכן את השחקן המקסימלי בעץ (פעולה פנימית של העץ) ואת מספר השחקנים בקבוצה. לבסוף נעדכן את מספר השחקנים הכללי בעץ. בסה"כ נקבל שהסיבוכיות הכוללת של פונקציה זו היא O(logn).

StatusType RemovePlayer(void *DS, int PlayerID)

נחפש את השחקן הרלוונטי בעץ השחקנים הכללי לפי ה-player_id של השחקן. העץ הוא עץ AVL ממויין ולכן הסיבוכיות היא O(logn) כפי שנלמד בהרצאה.
ניצור משתנה מסוג player שהערך שלו יהיה הvalue של השחקן שאותו אנחנו רוצים למחוק - נוציא אותו על ידי הפונקציה getData שמחזירה ערך של צומת בעץ, במקרה זה היא תחפש לפי הplayer_id (הגדרנו את האופרטור > במחלקה player כן שישווה עבור struct מסוג player לפי הplayer_id). הפונקציה getData עושה חיפוש בסיבוכיות של O(logn) והחזרת הערך היא פעולה אחת. באופן דומה נבצע פעולה זו גם על עץ הlevelים הכללי רק עם משתנה מסוג level. לכן בסה"כ עדיין נישאר בסיבוכיות של O(logn).
לפני המחיקה מהעץ של השחקנים הממויין לפי player_id נשמור את המצביע ל-group שבו הוא נמצא.
נמחק את השחקן משני העצים הללו בסיבוכיות של O(logn) ונבצע גלגולים בהתאם כדי לשמור על העץ מאוזן.
לאחר מכן בעזרת המצביע לקבוצה ששמרנו - נחפש את השחקן בעץ השחקנים של הקבוצה - הסיבוכיות של החיפוש בו היא לכל היותר O(logn) כאשר n זה כלל השחקנים ולכן בסה"כ נישאר עדיין בסיבוכיות כוללת של O(logn). לאחר מכן נעדכן את num_of_players במשחק ובקבוצה הספציפית - שתי פעולות.
בנוסף כאשר נעשית פעולה על עץ - נעדכן את המצביע לשחקן המקסימלי (פעולה בתוך הקובץ h של העץ), גם עדכון זה הוא נשאר בסיבוכיות של O(logn) – (עומק של עץ AVL) ולכן בסה"כ נקבל שהסיבוכיות הכוללת של פונקציה זו היא O(logn).

:StatusType ReplaceGroup(void *DS, int GroupID, int ReplacementID)

ראשית ניצור 2 משתנים מסוג group בעזרת ה GroupID והReplacementID שקיבלנו, נבדוק אם הקבוצות קיימות בעץ AVL_groups בעזרת הפונקציה isExist של העץ שמבצעת חיפוש בעץ בסיבוכיות של O(logk) כשאר k הוא מספר הקבוצות בעץ. במידה ואחת מהן לא קיימת נזרוק שגיאה. כעת ניצור 2 פוינטרים מסוג group עם הdata של הGroupID והReplacementID שקיבלנו, בעזרת הפונקציה של העץ getDataptr שמבצעת חיפוש בעץ בסיבוכיות של O(logk) כאשר k הוא מספר הקבוצות בעץ, ומחזירה פוינטר לdata של הצומת הרלוונטי. לאחר מכן נמזג את 2 עצי הlevelים של הקבוצות בעזרת הפונקציה של העץ mergeTrees. הפעולה תבוצע בסיבוכיות של O(n_replacement+n_group) כפי שלמדנו בתרגול, כאשר n_replacement הוא מספר השחקנים בעץ אליו מועברים השחקנים מהקבוצה הנמחקת וn_group הוא מספר השחקנים בקבוצה הנמחקת. נציב את העץ החדש בקבוצה שהid שלה הוא הReplacementID בעזרת האופרטור = של העץ. במימוש של האופרטור אנו מוחקים את הצמתים של העץ הקיים ומציבים בו את הצמתים של העץ החדש ולכן הפעולה תבוצע בסיבוכיות של O(n_replacement+n_group). לאחר שאיחדנו את 2 העצים, נעדכן את מספר השחקנים בקבוצה החדשה, ונוסיף לו את מספר השחקנים שהיו בקבוצה הנמחקת.נסיר את הקבוצה הנמחקת מהעץ AVL_groups בעזרת פונקציית remove של העץ. הדבר יעשה בסיבוכיות של O(logk) כאשר k הוא מספר הקבוצות בעץ.

כעת, ניצור מערך מעץ הlevelים המעודכן (שמכיל את השחקנים שהיו בקבוצה הנמחקת ואת השחקנים שהיו בקבוצה אליה העברנו את השחקנים מהקבוצה הנמחקת).הדבר יבוצע בסיבוכיות של O(n_replacement+n_group) כפי שלמדנו בתרגול, כאשר n_replacement+n_group הוא מספר השחקנים בעץ הlevelים המעודכן. כעת נעבור על כל אברי המערך, ועבור כל איבר במערך נעדכן את הplayer_info שלו על מנת שיצביע על הקבוצה המעודכנת המכילה את השחקנים מ2 הקבוצות. הדבר יבוצע בסיבוכיות של O(n_replacement+n_group) כגודל המערך.

בסה"כ כל הפעולות שביצענו יבוצעו בסיבוכיות של O(logk+n_replacement+n_group).

:StatusType IncreaseLevel(void *DS, int PlayerID, int LevelIncrease)

נחפש את השחקן בעץ השחקנים הכללי בסיבוכיות של O(logn) כפי שפירטנו בפונקציות קודמות – במקרה של כישלון נחזיר failure. בעזרת הפוינטר שלו לקבוצה נגיע ישר לעץ ה-level-ים של הקבוצה בה הוא נמצא - שם נחפש את השחקן בעץ – בסיבוכיות של O(logn), ונסיר את הצומת.

ניצור משתנה מסוג חדש שיכיל את הlevel החדש והplayer_id והמצביע group_info (לפני המחיקה נשמור את המידע בצד – מספר סופי של פעולות), ונוסיף אותו מחדש לעץ הlevelים של הקבוצה בסיבוכיות של O(logn) – על מנת שיוכנס לעץ ממויין.

ניצור משתנה זמני שהוא מסוג level (שהlevel שלו יהיה הlevel הישן – מידע ששמרנו בצד, פעולה אחת) ונבצע חיפוש בעזרת פונקצית find בעץ הlevelים הכללי של כל השחקנים בעזרת המשתנה הזמני שיצרנו, נסיר את הצומת ונוסיף אותה מחדש – פעולות שעשות בסיבוכיות של O(logn) כל אחת.

כפי שהוסבר בפוקציות קודמות – יתעדכן גם המצביע לשחקן המקסימלי בעץ – בסיבוכיות של O(logn).

בסה"כ כל הפעולות שביצענו יהיו בסיבוכיות של O(logn).

במהלך ההוספה של צומת לעץ יכולה להיות שגיאה בהקצאת הזיכרון – כי אנחנו יוצרים node חדש, ולכן במקרה זה נזרוק שגיאת bad alloc וכשהפונקציה הראשית תתפוס אותה אז נחזיר שגיאת זיכרון – אחרת נחזיר הצלחה.

:StatusType GetHighestLevel(void *DS, int GroupID, int *PlayerID)

תחילה נבדוק אם הGroupID שקיבלנו קטן מ-0. אם הוא אכן קטן מאפס נבדוק אם מספר השחקנים הכולל במשחק קטן מאפס. במידה וכן, נציב מינוס 1 בPlayerID. זה יעשה בסיבוכיות של O(1). אם מספר השחקנים גדול מ-0, בעץ הlevel-ים הכללי נשתמש בפונקציה של getMaxData של העץ שמחזירה את הצומת המקסימלי. הדבר יבוצע בסיבוכיות של O(1) מכיוון שאנו שומרים פוינטר בעץ לצומת המקסימלי. לאחר מכן נציב את PlayerID של השחקן שמצאנו את הצומת שלו בפוינטר של PlayerID.

אם הGroupID שקיבלנו גדול מ-0 נחפש את הgroup בעץ בעזרת הפונקציה getData שמחזירה ערך של צומת בעץ, במקרה זה היא תחפש לפי הGroupID (הגדרנו את האופרטור > במחלקה group כן שישווה עבור struct מסוג group לפי הGroupID). הפונקציה getData עושה חיפוש בערך לפי סיבוכיות O(logk), כאשר k הוא מספר הקבוצות בעץ, והחזרת הערך היא פעולה אחת. נבדוק אם קיימים שחקנים בקבוצה, אם לא, נציב מינוס 1 בPlayerID, ואם קיימים שחקנים בקבוצה נלך לעץ הlevelים של הקבוצה ונשתמש

בפונקציה getMaxData כפי שהשתמשנו בה בעץ הlevelים הכללי של כל השחקנים במשחק ולכן הדבר יעשה גם בסיבוכיות של O(1) .
סה"כ הפעולות שביצענו יהיו בסיבוכיות של O(1) במידה והGroupID קטן מ-0 או בסיבוכיות של O(logk) במידה והGroupID גדול מ-0.

StatusType GetAllPlayersByLevel(void *DS, int GroupID, int **Players, int *numOfPlayers):
נבדוק האם ה-groupID שקיבלנו קטן מאפס. אם כן, נקצה מערך באורך של כל השחקנים במשחק כך שכל תא הוא מסוג int - ערך שנקבל על ידי getNumOfPlayers - משתנה ששמרנו את ערכו ב-PlayersManager. ניצור מערך שהוא גם בגודל של מספר השחקנים במשחק - פה ערך של כל תא הוא יהיה מסוג level - בכל אחת מההקצאות נעשה malloc בהתאם לגודל של ה-value שיהיה במערך. עשינו מספר פעולות קבוע ולכן עד פה - אנחנו בסיבוכיות של O(1). נעביר את עץ ה-levelים של כל השחקנים למערך של הlevel-ים שיצרנו בעזרת פונקציית עזר - treeToArrayInorder שיצרנו. נשתמש באופן דומה למה שלמדנו בביתה במעבר על העץ בסדר שהוא inorder בצורה רקורסיבית- על מנת להדפיס את הערכים בסדר עולה. נעבור על כל צומת ולכן בסה"כ הסיבוכיות תהיה O(n). פונקציית עזר זו תקבל מצביע למערך levelים ומצביע לint שבו נשמור את האינדקס תא שבעת אנחנו מכניסים אליו את המידע. לאחר שניצור את המערך הזה, נוציא את המידע שאנחנו צריכים מכל תא שכרגע הוא מסוג value - מספר הפעולות יהיה כמספר הצמתים בעץ – n, וכך נעביר את הplayer_id למערך שקיבלנו. בסה"כ מספר הפעולות שעשינו הוא בסיבוכיות של O(n). נמחק את מערך העזר שיצרנו מסוג level - ונחזיר success.
במקרה של שגיאה בהקצאה של המערך נתפוס אותה ב-catch ונחזיר שגיאת allocation error.
במקרה שה-groupID גדול מ-0, נבדוק קודם אם הוא קיים בעץ הקבוצות - החיפוש בסיבוכיות של O(logk). אם הוא לא קיים נחזיר שגיאה. אחרת, נבצע באופן דומה את ההעתקה של הערכים של עץ הlevelים למערך וקבלת הplayer_id - ונטפל בשגיאות הקצאת הזיכרון באופן דומה. בגלל שפה מספר השחקנים שבעץ הlevelים של הקבוצה הוא n_groupID נקבל שהסיבוכיות של מעבר המידע למערך הוא O(n_gorupID).
בסה"כ הסיבוכיות של כל הפעולות אם groupID קטן מ-0 הינה O(logk+n_groupID) כאשר k הוא מספר הקבוצות ו-n הוא מספר כל השחקנים בקבוצה הזו, ואם groupID קטן מ-0 אז הסיבוכיות היא O(n) כאשר n הוא מספר השחקנים בכל המשחק.

StatusType GetGroupsHighestLevel(void *DS, int numOfGroups, int **Players):
ניצור מערך בגודל numOfGroups ונעבור על העץ AVL_group ע"י מעבר inorder בעץ. עבור כל קבוצה נבדוק אם קיימים בה שחקנים בעזרת num_of_players. במידה וקיימים בה שחקנים, נוסיף אותה למערך. נגדיר counter שסופר מה התא האחרון שבו הכנסו ערך, במידה והוא קטן מ numOfGroups נשחרר את המערך שהקצנו ונזרוק שגיאה. אם במהלך הריצה תהיה הקצאת זיכרון שנכשלה, נזרוק את השגיאה ALLOCATION_ERROR. במהלך הריצה, כאשר ערך ה- counter שווה ל numOfGroup, נעצור את המעבר inorder על העץ ובכך נשמור על סיבוכיות של O(numOfGroups). כעת נעבור על המערך, ועבור כל קבוצה, נוציא את השחקן המקסימלי ע"י הפונקציה getMaxData, כפי שציינו בפונקציות קודמות והסיבוכיות של הפעולה עבור כל קבוצה תהיה O(1).נוסיף את ה player_id של כל שחקן מקסימלי למערך Players, בסיום הפונקציה נדאג לשחרר את מערך הקבוצות שהקצנו.
בסה"כ נקבל שהסיבוכיות של הפונקציה היא O(numOfGroups).

void Quit(void **DS):
נשתמש בפונקציית ()deleteTree שמוחקת את העץ – נממש אותה בעזרת פונקציה פנימית שעוברת ומוחקת בצורה רקורסיבית את כל הnodeים.
נמחק את העץ של כל השחקנים הממויינים לפי level וכל השחקנים שממויינים לפי player_id בסיבוכיות של O(n)- כי אנחנו עוברים על כל צומת בעצים האלו – וסך כל הצמתים הוא n. באופן דומה נעשה את זה גם על העץ של הקבוצות - ששם מספר הצמתים שם הוא k.
כאשר אנחנו מוחקים כל node, אז גם ה-data של כל node נמחק בעזרת ה-destructor שנקרא באופן אוטומטי על ידי המערכת ובכך גם הvalue של כל צומת נמחק.
סה"כ הסיבוכיות תהיה O(k+n).

<u>סיבוכיות מקום של כל התוכנית:</u>

AVL_groups:

עץ בגודל k כגודל מספר הקבוצות במשחק. סך הכל סיבוכיות מקום של O(k).

AVL_all_players_id:

עץ בגודל n כגודל מספר השחקנים במשחק. סך הכל סיבוכיות מקום של O(n).

AVL_all_levels :

עץ בגודל n כגודל מספר השחקנים במשחק. סך הכל סיבוכיות מקום של O(n).

players_level:

עץ בגודל num_of_players_in_group. סך הכל עבור כל קבוצה סיבוכיות המקום תהיה O(num_of_players_in_group). הסיבוכיות מקום תהיה קטנה מ-O(n) כי מספר שחקנים עבור כל קבוצה קטן מ-n שהוא מספר השחקנים הכללי.

סה"כ עבור כל התוכנית נקבל סיבוכיות מקום של O(k+n)