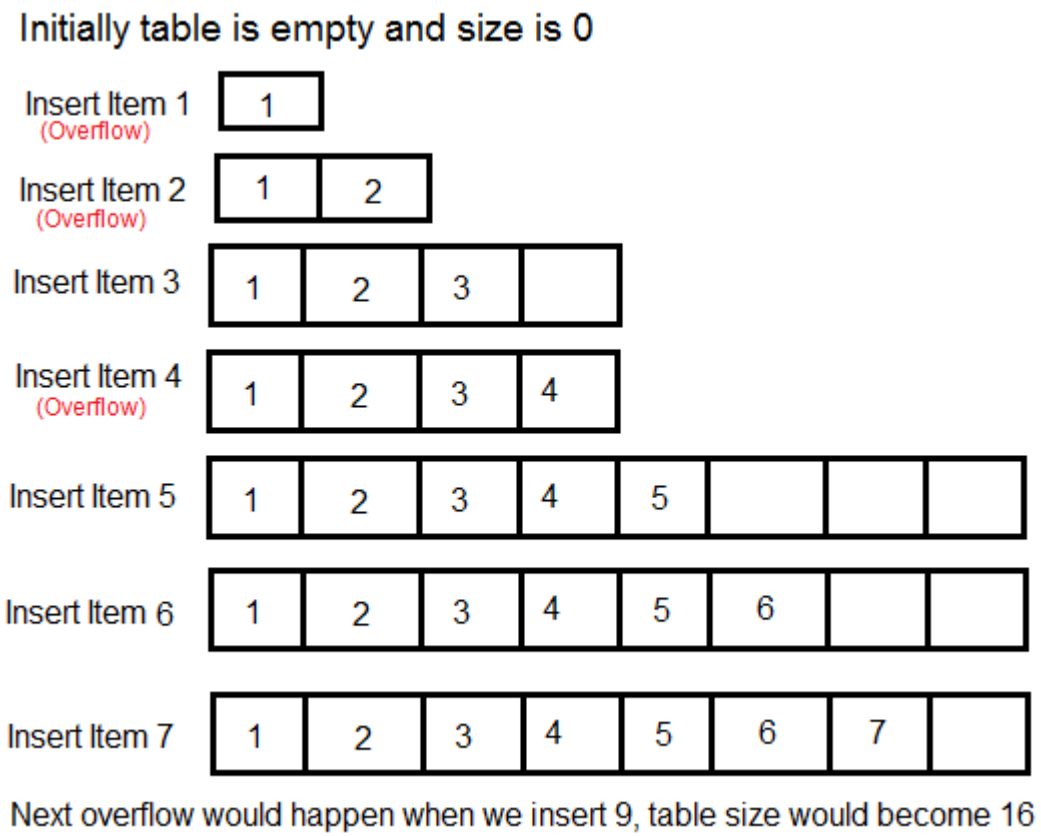
Dynamic Array:

בתרגיל מימשנו מבנה נתונים מסוג מערך דינאמי. המערך הדינאמי יהווה בהמשך בסיס למימוש ערימת המינימום. גודלו ההתחלתי המקסימלי של המערך הוא 2 ונרצה שבכל הכנסה והוצאה גודלו של המערך ישתנה לפי הצורך. זאת יאפשר לנו לממש ערימה מינימום כך שאין לנו חסם על גודלה בהמשך. אם מכניסים איבר למערך והגענו למספר המקסימלי נגדיל את המערך פי 2. אם אנחנו מבצעים מחיקה מהמערך ומספר האיברים קטן מרבע גודל המערך נקטין אותו פי 1/2.

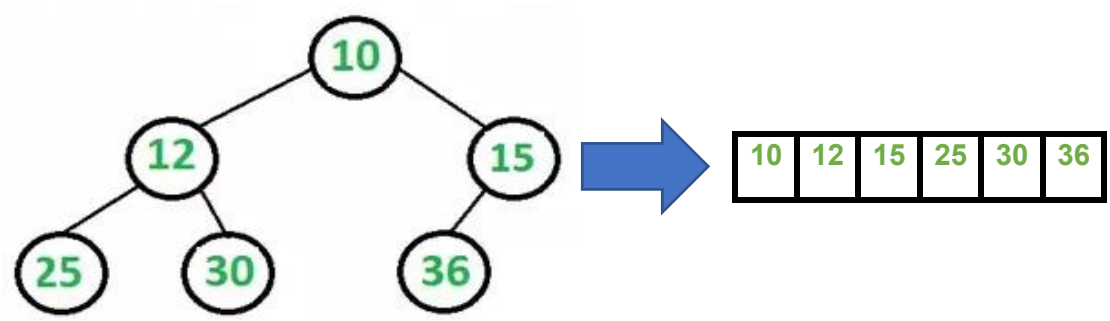
תיאור גרפי:(עבור הכנסה)



Minimum Heaps:

בתרגיל מימשנו ערימת מינימום על בסיס מערך דינאמי(שפירטנו אודותיו מעלה). איננו יודעים חסם על מספר השבטים לכן מערך דינאמי עוזר לנו להתגבר על כך. הצורך לערימת מינימום נבע מהדרישה להחזיר את השבט עם המזהה הקטן ביותר שאינו נכבש בסיבוכיות O(1). נחשוב על הערימה כעץ בינארי כמעט שלם, כך שצמתיו הם השבטים במערכת. נממש זאת בעזרת מערך דינאמי כך שהתא ה-i במערך הוא השבט בעל האינדקס ה-i בעץ הבינארי הכמעט שלם. בעזרת שמירת הסדר כך שכל ילד גדול מאביו יבטיח לנו שהאיבר הקטן ביותר יהיה בשורש. גישה לשורש נעשית ב-O(1) ולכן כשנידרש להחזיר מינימום מהערימה נוכל לעשות זאת ביעילות.

תיאור גרפי:



HashTable:

בתרגיל זה מימשנו מבנה נתונים מסוג טבלת ערבול. הצורך נבע מהדרישה בתרגיל להוסיף שחקן לשבט מסוים בממוצע על הקלט של O(m) כאשר m זה מספר השחקנים במערכת. כלומר נצטרך למצוא את השבט שנוסיף אליו ב-O(1). על פי הנלמד בכיתה בעזרת טבלת ערבול נוכל לעמוד בסיבוכיות זאת. פונקציית הערבול שבחרנו בה היא f(x)=X(modn). כאשר n נקבע על פי פקטור העומס שאנו צריכים לעמוד בו בכדי שישמר הפיזור האחיד. כל עוד נשמר הפיזור האחיד נוכל לגשת ולמצוא שבט מסוים ב-O(1). הגודל ההתחלתי שהשתמשנו בו הוא 10 והוא משתנה תוך כדי ריצת התוכנית כפי שפירטנו קודם.

AVLRankedTree:

עץ דרגות מרחיב מבנה הנתונים עץ AVL, בנוסף תכונות עץ הAVL, בכל צומת בעץ נשמור גם "דרגה" – מספר הבנים בתת העץ שהצומת הוא השורש שלו, כולל הצומת עצמו. ובנוסף לדרגה נשמור גם את סכום הערכים של הבנים בתת העץ שהצומת הוא השורש שלו, כולל הצומת עצמו, שתי התכונות האלה יעזרו לנו במימוש הפונקציה clanFight. פעולות ההכנסה,הוצאה,מחיקה וחיפוש נשארות עם אותה סיבוכיות כמו עץ AVL רגיל.

AVL Tree:

בדומה לתרגיל הקודם לצורך שמירת מזהי השחקנים של המערכת.

תיאור המערכת:

המערכת שלנו מכילה עץ AVL השומר את מזהי השחקנים שנכנסו למערכת, ערימה השומרת את מזהי השבטים שלא נכבוש ממויינת לפי סדר יורד של מזהי השבטים. טבלת ערבול לצורך שמירת מצביעים לשבטים לפי מזהה שבט.

כל שבט מכיל עץ דרגות השומר את תוצאות השחקנים אשר שייכים לשבט.

<u>הסבר המימוש ועמידה בדרישות הסיבוכויות של פונקציות המערכת:</u>

אתחול המערכת – void* init(int n, int* clanIDs)

סיבוכיות נדרשת: O(n) כאשר n הוא מספר השבטים.

נאתחל את המבנים שהצגנו בהתחלה: אתחול טבלת הערבול O(1), והכנסת השבטים לתוכה O(n).

אתחול עץ הAVL עבור אחזקת הId של השחקנים O(1), הכנסת השבטים לתוך טבלת הערבול O(n) והכנסת השבטים לערימה O(n).

סה"כ: O(n)

הכנסת שבט למערכת - addClan (void* DS, int clanID)

סיבכויות נדשרת: O(logn) בממוצע על הקלט משוערך כאשר n הוא מספר השבטים במערכת.

נבדוק אם קיים השבט במערכת - O(1) בממוצע על הקלט בעזרת טבלת הערבול, אם לא קיים נכניס אותו לטבלת הערבול O(1) בממוצע על הקלט ונכניס אותו לערימה השומרת על השבט המינימאלי שלא נכבש ב O(logn)

סה"כ: O(logn) בממוצע על הקלט.

הכנסת שחקן - addPlayer(void* DS, int playerID, int score, int clan) למערכת

סיבכויות נדשרת: O(logm) בממוצע על הקלט כאשר m הוא מספר השחקנים במערכת.

נבדוק אם קיים השחקן במערכת על ידי חיפוש הId של השחקן בעץ AVL, O(logm), אם לא קיים, נכניס את ה Id שלו לעץ AVL – O(logm), נמצא אם השבט אליו נרצה להכניס אותו קיים – O(1) בממוצע על הקלט, אם לא קיים נמצא את השבט הרלוונטי בטבלת הערבול - O(1) בממוצע על הקלט, נכניס את הScore שלו לעץ הדרגות של התוצאות של השבט – O(logm).

סה"כ: O(logm) בממוצע על הקלט.

מלחמת שבטים – clanFight (void* DS, int clan1, int clan2, int k1, int k2)

סיבוכיות מבוקשת: O(logm + logn) בממוצע על הקלט כאשר m מספר השחקנים במערכת וn מספר השבטים במערכת.

נבדוק אם קיימים השבטים במערכת- O(1) בממוצע על הקלט , אם קיימים השבטים לנקרא לפונקציה המחשבת את סכום התוצאות של k השחקנים עם התוצאות הכי גבוה בצורה הבאה: כזכור אנו שומרים את התוצאות של השחקנים לכל שבט בעץ דרגות ולכל צומת מחזיקים את סכום התוצאות של כל תת העץ של הצומת כולל הצומת עצמו. עבור k כלשהו ועבור h=logm (גובה העץ לכל היותר) אנחנו ניגש לכל היותר לh-k צמתים בעץ על מנת לדעת את הסכום ולכן סיבוכיות הפונקציה הזאת היא O(logm). נבצע שתי קריאות לפונקציה ולכן הסיבוכיות עדיין O(logm). לאחר שנבדוק מי השבט שניצח, נוציא אותו מהערימה, O(logn) בממוצע.

סה"כ: O(logm + logn) בממוצע על הקלט.

מציאת השבט שלא נכבש בעל המזהה הנמוך ביותר – getMinClan (void* DS, int* clan)

סיבוכיות מבוקשת:O(1)

כפי שתיארנו את המערכת, אנחנו מחזיקים ערימה שבראשה השבט על המזהה הקטן ביותר שלא נכבש, הוצאתו היא O(1).

סה"כ: O(1)

מחיקת המערכת ושחרור הזכרון – quit(void** DS)

סיבוכיות מבוקשת: O(m +n) כאשר n הוא מספר השבטים וm מספר השחקנים במערכת.

נעבור על כל התאים בטבלת הערבול ונמחק את השבטים מהמערכת – O(n), עבור כל שבט נמחק את עץ תוצאות השחקנים O(m), נמחק את עץ מזהי השחקנים מהמערכת – O(m) , נעבור על ערימה ונשחרר את התאים O(n).

סה"כ: O(n +m)

סיבוכיות מקום המערכת:

השתמשנו בעץ AVL לכל השחקנים – O(m) וגם עצי דרגות לתוצאות השחקנים O(m), מערך דינאמי בממוצע O(n), וטבלת הערבול בממוצע O(n).

סה"כ: O(n+m)