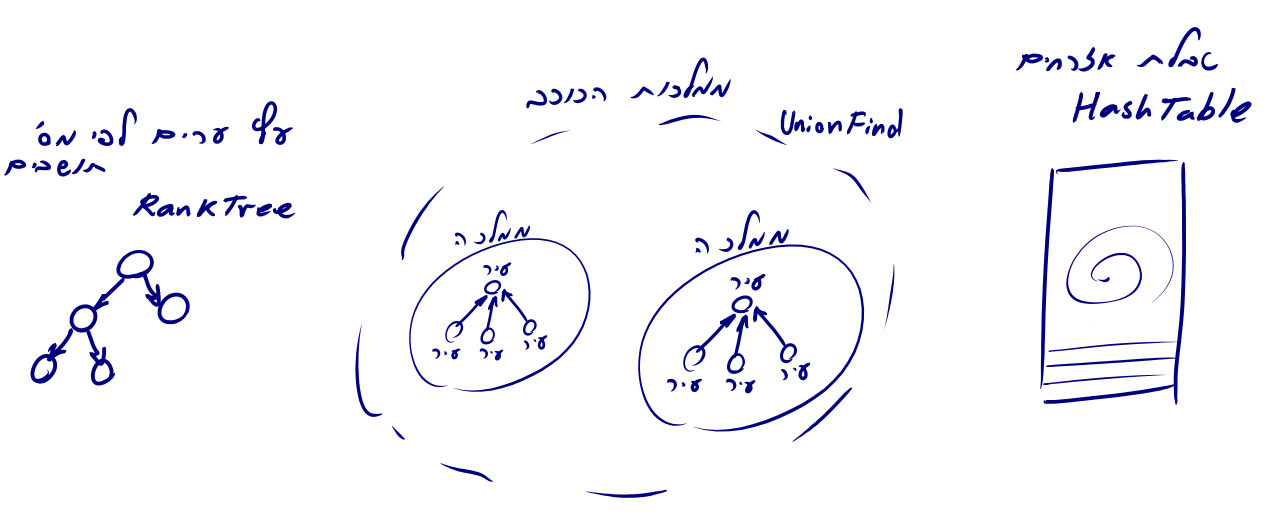
**מה מכיל:**



**CitizensTable** – HashTable – טבלה דינאמית, linear probing, אפשר רק להכניס אין מחיקה.

מאחסן את כל התושבים הרשומים על הכוכב כאשר המפתח הוא מספר תושב והמידע הוא מספר העיר לה הוא שייך או 0 (מספרי הערים ממסופרים 1 עדN )

**PlanetUnion** – UnionFind מבוסס עצים הפוכים, עם איחוד לפי גודל וכיווץ מסלולים.

כאשר הקבוצות הן הממלכות, האיברים הם הערים מטיפוס City. ערי בירה מיוצגות כעיר שהיא השורש בעץ ההפוך ושומרים על כך בפעולות השונות.

**CitiesTree** - עץ דרגות מבוסס AVL -

כל הערים על הכוכב, ממוניות לפי מספר תושבים. בעץ יופיעו רק ערים עם מספר תושבים שונה מ-0.

City – מבנה פשוט שמייצג עיר ומכיל את מספרה ואת מספר התושבים בה.

**מימוש הפונקציות והסבר סיבוכיות:**

**void\* Init(int n)**

1. צור מבנה PlanetUnion עבור N ממלכות ובכל ממלכה תהיה עיר אחת עם 0 תושבים. O(n)
2. צור CitizenTable בתור רשימת תושבים ריקה. O(1)
3. צור CitysTree מתוך רשימה של הערים: 0,1,…,n-1 יצירת עץ כמעט מלא מרשימה לפי אלגוריתם שהוסבר בתרגול, נמלא את הערך בכל צומת (גודל תת העץ) במעבר postorder - O(n)

סה"כ O(n)

**StatusType AddCitizen(void\* DS, int citizenID)**

1. חפש את המפתח עבור מספר התושב המבוקש O(1) בממוצע על הקלט
2. אם נמצא:
   1. החזר שגיאה (תושב כבר קיים במערכת)
3. אחרת:
   1. הוסף תושב חדש כאשר מספר העיר אילה הוא שייך היא -1 (לא מאותחל). O(1) בממוצע על הקלט, משוערך

הסבר סיבוכיות: טבלת הערבול דינאמית, הוכחת הסיבוכיות כמו מחסנית דינאמית שראינו בתרגול רק שההכנסה ב- O(1)ממוצע.

עבור הכנסה בלי הגדלה = O(1) ממוצע. עבור הכנסה עם הגדלה: מקצים מערך חדש עם אתחול יעיל O(1), מבצעים rehash לכל איבר בטבלה = Nפעמים O(1) ממוצע. מכיוון שהטבלה גדלה פי 2 בכל פעם, מובטח שנבצע N/2 פעולות הכנסה זולות לפני שצטרך להגדיל. לכן אם נפרוס את העלויות נקבל- 3\*O(1) בממוצע, משוערך = O(1) בממוצע, משוערך.

**StatusType MoveToCity(void\* DS, int citizenID, int city)**

1. חפש את התושב המבוקש ב CitizenTable. O(1) בממוצע על הקלט
2. אם התושב קיים או כבר משיוך לעיר
   1. החזר שגיאה.
3. עדכן את העיר אליה הוא משיוך O(1) (כבר מצאנו אותו בטבלה)
4. עדכן את מספר התושבים ב CitysTree ע"י מחיקה והכנסה O(logn)
5. עדכן את המספר התושבים בעיר שב- PlanetUnion. עדכון העץ שהעיר נמצאת בו כך שבראש העץ נקבל את העיר עם המספר המקסימלי של תושבים: מוצאים את עיר הבירה ומשווים לעומת העיר שהוספנו לה תושב. אם צריך להחליף עיר בירה, מחליפים בין המספרי עיר של שני הצמתים ומעדכנים את מערך האלמנטים כדי שיצביע בהתאם. O(log\*n)

סה"כ O(logn) בממוצע על הקלט

**StatusType JoinKingdoms(void\* DS, int city1, int city2)**

1. ערי בירה מיוצגות כעיר שהיא שורש בעץ ההפוך של ממלכה מסוימת במבנה ה UnionFind. בנוסף השורש יצביע על מבנה הGroup והGroup יחזיק מצביע לשורש העץ.
2. אם הערים לא עיר בירה (לא שורש) O(1)
   1. החזר שגיאה
3. אחרת
   1. בצע JOIN בין הממלכות לפי הממלכה עם הכי הרבה ערים O(1)
   2. אם הממלכה הקטנה יותר מכילה עיר בירה עם יותר תושבים (היא העיר בירה החדשה)
      1. עדכן שהיא תהיה שורש העץ והשורש הקודם יהפוך להיות בן של השורש החדש.

סה"כ O(1) עומד בדרישה של O(log\*n) משוערך

**StatusType GetCapital(void\* DS, int citizenID, int\* capital)**

1. חפש את התושב ב CitizenTable O(1) בממוצע על הקלט
2. אם התושב לא קיים או לא ממופה לשום עיר:
   1. החזר שגיאה
3. אחרת
   1. גש לעיר עם האינדקס המתאים לפי רשומת התושב במבנה PlanetUnion ועלה לשורש עץ הממלכה, מבצעים כיווץ למסלול, השורש הוא עיר הבירה O(log\*n) משוערך

סה"כ O(log\*n) בממוצע על הקלט, משוערך

**StatusType SelectCity(void\* DS, int k, int\* city)**

1. בצע פעולת Rank על עץ דרגות מאוזן CitysTree O(logn)

**StatusType GetCitiesBySize(void\* DS, int cities[])**

1. בצע מעבר Inorder על העץ CitysTree O(n)

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

1. שחרר טבלת ערבול O(1) (מוקצת כזיכרון רציף)
2. שחרר עץ ערים O(n)
3. שחרר UnionFind ממלכות O(n)
4. DS = NULL

סה"כ O(n)

**הסבר סיבוכיות מקום:**

1. טבלת ערבול של התושבים -

גודל מערך הטבלה לכל היותר פי 2 ממס' תושבים m . והמערך באתחול יעיל לוקח פי 3.

סה"כ 6\*O(m) = O(m)

1. UF של ממלכות –

מערך גישה לממלכות O(n), מערך גישה לערים O(n), צומת לכל עיר O(n), סה"כ O(n)

1. עץ דרגות של ערים- O(n)

סה"כ O(n+m)