מבני נתונים – תרגיל רטוב 2

המחלקה ElectionSystem המייצגת מערכת בחירות מורכבת ממבני הנתונים הבאים:

1. מערך של מועמדים בגודל n המחזיק עבור כל מועמד את מספר ההצבעות שלו.
2. עץ AVL המחזיק את המועמדים ממיונים לפי מספר הצבעות ואז לפי תעודת זהות.
3. טבלת ערבול דינמית בשיטת chain hashing המחזיקה את מספרי תעודות הזהות של אנשים שהצביעו.
4. Union-Find המספק מידע עבור כל מועמד לאיזו מפלגה הוא שייך. עבור כל מפלגה מבנה ה-Union Find מחזיק רשומה עם תעודת הזהות של המנהיג שלה. ה UnionFind עובד בתצורה של כווץ מסלולים ואיחוד קבוצות קטנות לגדולות.

הוכחת סיבוכיות ונכונות:

מערך המועמדים:

במערך פעולת הכנסה, ועדכון מתבצעות ב(O(1

סיבוכיות מקום (O(n

עץ AVL:

אתחול (O(1

חיפוש, הוצאה והכנסה מתבצעים ב ((O(log(n

סיור בעץ מתבצע ב(O(n

הריסה מתבצעת ב (O(n

סיבוכיות מקום (O(n

(הוכח בתרגיל רטוב 1)

טבלת ערבול

:HashTable

טבלת הערבול היא דינאמית ופועלת לפי עקרון chain-hashing כאשר במקום רשימות מקושרות השתמשנו בעצי AVL (שמימשנו בתרגיל 1). פונקצית הערבול שלנו היא ((floor(table size*(phi*key mod1 כאשר phi=(sqrt(5)-1)/2. הטבלה מחזיקה מערך של עצים ומונה של מספר האיברים בטבלה.

הערה: נגדיר את α להיות פקטור העומס כלומר m/n. כאשר m הוא מספר המפתחות, ו n הוא גודל הטבלה הנוכחי.

Insert:

1. הפונקציה מקבלת את המפתח להכנסה מפעילה עליו את פונקצית הערבול אשר ממנה נקבל אינדס להכנסה למערך העצים. פעולות אלו מקיימות את ההנחה שאין כפילויות מכיוון שמתבצעת בדיקה בעת ההכנסה לעץ. תזרק הערה אם המפתח קיים. ((O(1)+O(log(α
2. נגדיל את מונה האיברים בטבלה ואם הוא שווה לגודל המערך נכפיל את גודל הטבלה. על ידי קריאה לchangeSize עם הגודל החדש.(O(m

Remove:

1. הפונקציה מקבלת את המפתח להוצאה מפעילה עליו את פונקצית הערבול אשר ממנה נקבל אינדס להוצאה למערך העצים. תזרק הערה אם המפתח לא קיים.((O(1)+O(log(α
2. נקטין את מונה האיברים בטבלה ואם הוא שווה לרבע גודל המערך נחצה את גודל הטבלה. על ידי קריאה לchangeSize עם הגודל החדש.(O(m

Member:

1. הפונקציה מקבלת את המפתח לחיפוש מפעילה עליו את פונקצית הערבול אשר ממנה נקבל אינדס לחיפוש במערך העצים.
2. הפונקציה תחפש בעץ המתאים ותחזיר האם המפתח נמצא או לא.(O(1)+O(log(α)) α קבוע ולכן סה"כ O(1).

changeSize:

1. נקצה מערך חדש בגודל הרצוי O(1).
2. לכל תא בערך המקורי נבצע סיור inOrder על העץ במהלכו נכניס את המפתחות למערך זמני באורך α נעבור על המערך ונכניס אותו לפי פונקצית הערבול החדשה (עם הגודל של המערך החדש). למערך החדש, ונמחוק את המערך הזמני. סה"כ O(n+m)
3. נמחוק את המערך הישן. המחיקה תקרא להורס של כל עץ שעובד בזמן O(α) (כפי שהוכח בתרגיל בית 1). O(n+m) O(m)≈n ולכן סה"כ O(m).

~HashTable:

4. נמחוק את מערך העצים. המחיקה תקרא להורס של כל עץ שעובד בזמן O(α) (כפי שהוכח בתרגיל בית 1) .O(n+m) O(m)≈n ולכן סה"כ O(m).

סיבוכיות המקום: נשמור איבר יחיד באחד מעצי AVL עבור כל ת.ז של מצביע O(m) בנוסף- גודל

Union-Find:

על מנת לממש מבנה נתונים זה יצרנו 3 מחלקות חדשות:

UnionFindNode שמחזיק את המזהה שלו, ומצביע לUnionFindNode אחר.

UnionFindGroupObject שמחזיק מצביע לUnionFindRecord (ר' בהמשך) ומצביע לDetails. Details הינו אופציונלי ויכול להיות מכל טיפוס שהוא ולהחזיק כל מידע נוסף על הקבוצות לפי רצון המשתמש (UnionFind היא תבנית עבור Details) כאשר ברירת המחדל היא מספרים שלמים. המשתמש יכול לבחור האם לאתחל את הפרטים הנוספים עבור הUnion-Find על ידי קריאה לinitlizeDetails ועדכונם על ידי updateRecord (ר' בהמשך)

UnionFindRecord מחזיק מצביע לUnionFindNode שהוא שורש העץ של הקבוצה ב UnionFind ומונה עבור מספר איברי הקבוצה

אתחול- UnionFind(int newSize)

הפונקציה מקבלת את מספר האיברים הראשוני. נוצרים שני מערכים, אחד עבור UnionFindNode והשני עבור UnionFindGroupObject. עבור כל תא במערך UnionFindNode יאותחל צומת חדש כשהמזהה שלו הוא מספר התא. עבור כל תא במערך של UnionFindGroupObject תאותחל UnionFindRecord עם ה UnionFindNode בעל מזהה השווה לערך התא, ומונה לגודל הקבוצה יהיה 1. הDetails יהיה מאותחל ל NULL.

סה"כ סיבוכיות זמן(O(n סיבוכיות מקום O(n)

אתחול פרטים נוספים- void intializeDetails(Details* details)

הפונקציה מקבלת מערך של פרטים נוספים באורך n ושמה במערך UnionFindGroupObject בחלק הDetails את הפרטים הנוספים הרלוונטים לקבוצה שהאינדקס במערך הפרטים שווה לאינדקס במערך ה UnionFindGroupObject.

סיבוכיות זמן O(n) סיבוכיות מקום(O(n

הריסה- ~UnionFind()

הפונקציה עוברת על מערך ה UnionFindNode ומוחקת כל צומת שמוצבע על ידי המערך. לאחר מכן מוחקת את שני המערכים של UnionFindNode ו UnionFindGroupObject. ההורס של UnionFindGroupObject מוחק את ה UnionFindRecord וDetails שקיימים ב UnionFindRecordObject

סיבוכיות זמן O(n)

- איחוד בין קבוצות- unsigned int Union (unsigned int group1, unsigned int group2)

הפונקציה מקבלת שני אינדקסים של קבוצות ומחזירה את האינדקס של הקבוצה המאוחדת. הפונקציה בודקת אם שתי הקבוצות זהות, אם כן היא תחזיר את האינדקס הראשון שניתן. לאחר מכן הפונקציה תבדוק האם הקבוצות קיימות. במידה וקיימות הפונקציה תבדוק מהי הקבוצה הגדולה יותר, תעביר את הצומת UnionFindNode של הקבוצה הקטנה יותר להצביע על הצומת של הקבוצה הגדולה יותר ותעדכן את גדול הקבוצה הגדולה לסכום הגדלים. לאחר מכן הפונקציה תמחק את הרשומה של הקבוצה הקטנה יותר כי היא לא קיימת כבר ותחזיר את אינדקס הקבוצה הגדולה.

סיבוכיות זמן משוערכת O(log*(n)) כפי שהוכח בהרצאה עבור UnionFind המקצר מסלולים ומאחד קבוצות קטנות לגדולות.

- עדכון פרטים נוספים בקבוצה- void updateRecord(const Details& newDetails, unsigned int groupId)

הפונקציה מקבלת אינדקס של קבוצה ופרטים שצריך לעדכן עבור הקבוצה, אם הקבצה קיימת במערך UnionFindGroupObject הפונקציה תחליף את הפרטים הנוספים בחדשים

סיבוכיות זמן O(1)

- קבלת גודל קבוצה-unsigned int getGroupSize(unsigned int groupId)const

הפונקציה מקבלת אינדקס של קבוצה, אם הקבוצה קיימת במערך UnionFindGroupObject היא תחזיר את גודלה

סיבוכיות זמן O(1)

- חיפוש-unsigned int Find(unsigned int id)const

הפונקציה מקבלת מזהה ותחזיר את הקבוצה אליו המזהה שייך. הפונקציה תלך לצומת ה UnionFindNode שמוצבע על ידי מערך ה UnionFindNode באינדקס שערכו שווה למזהה, וכל עוד אביו של הצומת הזה אינו NULL היא תתקדם במעלה העץ. לאחר שהגיעה לצומת שאביו NULL היא תשמור ושוב תעלה באותו מסלול חיפוש רק כעת לכל צומת היא תשנה את האב להיות הצומת שאביו הוא NULL (קיצור מסלולים). בסיום הפונקציה תחזיר את המזהה של הצומת שאביו הוא NULL

סיבוכיות משוערכת O(log*(n)) כפי שהוכח בהרצאה עבור UnionFind המשתמש בקיצור מסלולים ואיחוד קבוצות קטנות לגדולות.

קבלת מידע נוסף עבור קבוצה- const Details& getGroupDetails(unsigned int groupId)

הפונקציה תחזיר את המידע הנוסף עבור הקבוצה עם האינדקס הנתון

מימוש הפונקציות הדרושות- הסברים והוכחות:

void* Init(int n):

הפונקציה תאתחל ElectionSystem על ידי קריאה לבנאי שלה. בבנאי יקרו הפעולות הבאות:

מערך מועמדים חדש בגודל n- O(n)

יצירת טבלת ערבול חדשה בגודל n- O(n)

יצירת UnionFind בגודל n- O(n)

יצירת עץ AVL ריק O(1)

לאחר מכן הבנאי יכניס למערך המועמדים את המועמדים לפי תעודות זהות בסדר עולה, ועם 0 הצבעות. כל תעודת זהות מתאימה לאינדקס במערך. O(n)

סה"כ O(n) סיבוכיות מקום וזמן.

הפונקציה תחזיר את המצביע לElectionSystem החדשה שנוצרה

statusType Vote(void* DS, int voterID, int candidate):

הפונקציה מוסיפה הצבעה למועמד ב-electionSystem. כדי לאפשר עדכון מספר הקולות- נבצע את הפעולות הבאות:

1. בדיקה שהמועמד שהוצבע הוא מועמד תקין – השוואה לשדה שקיים במערכת. במקרה הגרוע O(1).
2. בדיקה שהמצביע אינו קיים כבר במערכת – שימוש בפעולה member של hashTable O(1) בממוצע על הקלט.
3. במידה והכל תקין- הכנסת תעודת הזהות של המצביע למאגר המצביעים- שימוש בinsert של hashTable. O(1) במקרה הגרוע.
4. הוספת תמיכה למועמד- שימוש ב void ElectionSystem::support(int cadidate):
   a. אם המועמד לא הוצע לפני כן על ידי אף המצביע (מספר הקולות שצבר = 0)- O(1).
      i. הגדל את מספר ההצבעות של המועמד ב1 O(1).
      ii. צור מועמד חדש- O(1).
      iii. הכנס את המפתח המתאים למועמד לעץ AVL המחזיק את המועמדים ממוינים לפי מספר הצבעות ואז לפי ת.ז – O(log(n)).
   b. אחרת- עדכן את המועמד המתאים בעץ AVL (הוצאת הישן והכנסת חדש מעודכן) 2*O(log(n)).
5. עדכן את הפרטים בקבוצה אלי שייך המועמד שהוצבע.
   a. בדוק לאיזה קבוצה שייך המועמד- Find של UnionFind – O(log*(n)) משוערכת.
   b. עדכן את leader לאחר השוואה בין מספר ההצבעות של הנוכחי והחדש ועל פי מספר סידורי.

לסיכום: ביצוע הפעולות הנ"ל דורש-

$$5*O(1) + \underbrace{O(1) +}_{\text{ממוצע על הקלט}} 2*O(\log(n)) + \underbrace{O(\log^*(n))}_{\text{משוערך}} \approx O(\log(n))$$

בממוצע על הקלט, משוערך.

סיבוכיות מקום נוסף O(m) כמספר המצביעים.

StatusType SignAgreement(void* DS, int candidate1, int candidate2);

הפונקציה מאחדת שני מחנות שבהם חברים שני המועמדים. הפעולה מתבצעת רק אם שני המועמדים הם בעלי מספר ההצבעות הגבוה ביותר במחנה שלהם. מבצעת את הפעולות הבאות:

1. מוצאת את הקובצה אליה שייך כל מועמד על ידי שימוש ב Find של UnionFind- $2*O(\log^*(n))$ משוערך.
2. אם הקובוצות זהות תזרק הערה. $O(1)$
3. ניגש למידע הנוסף של כל אחת מהקבוצות בUnionFind $O(1)$. שם מוחזק מספר ת.ז של המועמד המוביל של הקבוצה, ומשם נחלץ את מספר ההצבעות המקסימאלי של הקבוצה. סה"כ $O(1)$.
4. נשווה את מספר ההצבעות של כל מועמד כדי לראות האם מספר ההצבעות שלו שווה למקסימום של הקבוצה שלו. אם לא תזרק הערה. $O(1)$.
5. נבצע איחוד של 2 הקבוצות. על ידי שימוש ב- Union של UnionFind עלות $O(1)$.
6. עדכון המנהיג החדש על ישי השוואת המידע הנוסף $O(1)$.
7. עדכון המידע הנוסף בקבוצה החדשה על ידי Find על המנהיג החדש משוערך$O(\log^*(n))$.
8. הקטנת מספר הקבוצות $O(1)$.

לסיכום: ביצוע כל הפעולות הנ"ל לוקחות זמן משוערך של:

$$6*O(1) + \underbrace{3*O(\log^*(n))}_{\text{משוערך}} \approx O(\log^*(n))$$

סיבוכיות מקום נוסף $O(1)$

StatusType CampLeader(void* DS, int candidate, int* leader)

הפונקציה מחזירה את תעודת הזהות של המועמד המוביל בקבוצה, שהמועמד שתעודת הזהות שהפונקציה מקבלת שייך אליה. הפונקציה מבצעת את הפעולות הבאות:

1. ביצוע find של UnionFind על תעודת הזהות וקבלת הקבוצה של המועמד בסיבוכיות משוערכת של $O(\log^*(n))$
2. החזרת המידע הנוסף מהקבוצה שהוא תעודת הזהות של מנהיג הקבוצה ב$O(1)$.

סה"כ ביצוע כל הפעולות הנ"ל לוקח סיבוכיות משוערכת של:

$$O(1) + \underbrace{O(\log^*(n))}_{\text{משוערך}} \approx O(\log^*(n))$$

סיבוכיות מקום נוסף $O(1)$

StatusType CurrentRanking(void* DS, int results[][2])

הפונקציה מחזירה מערך דו מימדי (כפי שמתואר בדרישות התרגיל)

1. יצירת מערך groupIndex בגודל כמות המועמדים ואיתחולו ל 1-, מערך זה יחזיק את דירוג הקבוצה כדירוג המועמד המוביל שלה בין יתר המועמדים המובילים. של כל מועמד. סה"כ O(n)
2. יצירת מערך resultIndex בגודל כמות הקבוצות ואיתחולו ל0. מערך זה יחזיק את האינדקס הבא במערך התוצאות אליו נכניס את פרטי המועמד ששייך לקבוצה שהאינדקס במערך הוא הדירוג שלה. סה"כ O(n)
3. מעבר inorder על העץ והפיכת תוצאות העץ למערך ממוין בסדר עולה O(n). תוצאות העץ יוכנסו לתוך תור והוצאה מהתור והכנסה למערך מהסוף להתחלה יתנו מערך ממוין בסדר עולה, בגלל תכונות העץ. סה"כ O(n) פעולות (מעבר על העץ+ n הכנסות לתור + n הוצאות מהתור והכנסות למערך)
4. העתקת המערך שקיבלנו מסעיף 3 למערך resultsCandidateArray בO(n) נשמור את גודל המערך שקיבלנו בסעיף 3 כאינדקס הבא להכנסה במידה ויש מועמדים שלא הצביעו להם.
5. בדיקה האם יש מועמדים שלא הצביעו להם, נרוץ על מערך המועמדים בסדר יורד מה שיבטיח לנו מיון לפי תעודת זהות עבור מועמדים עם הצבעות 0, במידה ויש מועמד שלא הצביעו לו נכניס אותו למערך resultsCandidateArray ונקדם את האינדקס במערך resultsCandidateArray לתא הבא. סה"כ O(n)
6. נחזיק שני מונים groupCounter שיבדוק בכמה קבוצות נתקלנו בעבר וindexCounter שיהווה צובר לכמה מועמדים יש בקבוצות שמדורגות גבוה יותר מהקבוצה שדירוגה שווה ל groupCounter ונאתחלם ל0. נעבור על המערך resultsCandidateArray ונבדוק לכל מועמד לאיזה קבוצה הוא שייך על ידי find. נקבל את האינדקס של דירוג הקבוצה אליה שייך המועמד על ידי פניה ל groupIndex במקום הקבוצה. במידה ואינדקס זה שווה ל1- לא נתקלנו בקבוצה זו בעבר, אז נשים בgroupIndex במקום הקבוצה את groupCounter ונכניס את הפרטי המועמד למקום המתאים במערך הפלט על ידי גישה לresultIndex במקום של groupCounter. אם groupCounter קטן ממספר הקבוצות פחות 1 אז, נוסיף לindexCounter את מספר המועמדים בקבוצה הנוכחית ונעדכן את התא הבא בresultIndex בערכו של indexCounter. לבסוף נקדם את groupCounter. אם אינדקס זה לא שווה ל1- נכניס את ונכניס את הפרטי המועמד למקום המתאים במערך הפלט על ידי גישה ל resultIndex במקום האינדקס . סה"כ פעולות השוואה, קידום והוספה ב O(1) ופעולת find בסיבוכיות משוערכת O(log*(n)). נבצע O(n) פעלות כאלה סה"כ O(n·log*(n)).

סה"כ סיבוכיות זמן במקרה הגרוע O(n·log*(n))
סיבוכיות מקום נוסף 3*O(n) ולכן O(n) כמספר המועמדים.

void Quit(void** DS)

הפונקציה קוראת להורס של המחלקה ElectionSystem וזה קורא לכל אחד מההורסים של האובייקטים במחלקה.

1. הורס של hashTable עובד ב-O(m) – כלומר סדר גודל של מספר המצביעים.
2. הורס של UnionFind עובד ב O(n).
3. הורס של עץ המועמדים עובד ב O(n).

סה"כ: O(m+n).