מבני נתונים 1
234218

תרגיל יבש 3

מספר

הוגש ע"י:

חגי קריטי	301781613
שם	מספר זהות
תם נלסון	204805824
שם	מספר זהות

<u>ציון</u>:

לפני בונוס הדפסה:

כולל בונוס הדפסה:

נא להחזיר לתא מס': 13

שאלה 1

נשמור את הצבעים בעץ דרגות, וכל צבע ישמור ערימת מקסימום של הכדורים שלו. הדרגות בעץ יהיו תוספת הנפח ונפח הכדור המקסימלי בכל תת עץ.

הצבעים יישמרו בעץ AVL לפי האינדקס שלהם. לכל צומת נוסיף את השדות הבאים:

vol_add_left - ישמור את כמות הנפח שיש להוסיף לכל הכדורים בצבעים בתת העץ השמאלי

vol_add_self - ישמור את כמות הנפח שיש להוסיף לכל הכדורים מהבצע הנוכחי

max_vol_ref - ישמור את תוספות הנפח מצמתי האב בזמן העדכון האחרון max_vol_left/right

max_vol_left - ישמור את נפח הכדור המקסימלי בתת העץ השמאלי.

max_vol_left_col - ישמור את מספר הצבע בעל נפח הכדור המקסימלי משמאל.

max_vol_right - ישמור את נפח הכדור המקסימלי בתת העץ הימני.

max_vol_right_col - ישמור את מספר הצבע בעל נפח הכדור המקסימלי מימין.

balls - ערימת מקסימום של כל הכדורים לפי נפחם.

מאופי המימוש של increase, הערכים max_vol_left/right יכולים להיות לא עדכניים. לכן אנחנו שומרים גם max_vol_ref שיאפשר לתקן אותם לפני רגע השימוש.

לכל כדור נשמור:

volume - נפח הכדור

color_vol_ref - תוספת הנפח שניתנה מהצבע בזמן הוספת הכדור למערכת. אנחנו שומרים ערך זה כדי לבטל תוספת נפח שניתנה מהצבע לפני שהכדור התווסף למערכת.

מימוש הפעולות:

Init(k):

ניצור עץ AVL עם k צמתים. לכל צומת נשים 0 בכל השדות המספריים. בנוסף ניצור ערימה ריקה עבור שדה balls.

סיבוכיות זמן:

יצירה של עץ AVL עם k צמתים: O(k)

יצירת ערימה ריקה: O(1) לכל ערימה, סה"כ O(k)

סה"כ: O(k)

AddBall(i, v):

נטייל בעץ הצבעים עד לצבע הדרוש, תוך כדי שאנחנו סוכמים את ערכי vol_add_left (בצומת האחרון נשתמש ב-vol_add_self), כדי לקבל את color_vol_ref של הכדור החדש. נכניס כדור חדש לערימה balls ונעדכן את שדות המקסימום של הצמתים במסלול.

ראשית נאפס מונה color_volume. לאחר מכן נתחיל ללכת בעץ הצבעים. בכל פעם שנמשיך לתת העץ השמאלי, נוסיף את הערך של vol_add_left של הצומת שממנה יצאנו למונה color_volume. כשנגיע לצומת של הצבע המבוקש, נוסיף את הערך vol_add_self למונה. לאחר מכן נוסיף את הכדור לערימה עם הנפח הנתון v בשדה volume וערך המונה color_volume בשדה color_vol_ref. ההשוואות בין הנפחים בערימה יעשו לפי הנפח האפקטיבי של כל כדור (הנפח הניתן בעת ההוספה + תוספות מהצבע) לפי:
effective_volume = volume + (color_volume - color_vol_ref).

לאחר הוספת הכדור, נבדוק אם המקסימום של הערימה השתנה. אם הוא השתנה, יש צורך לעדכן את שדות המקסימום. ראשית נעדכן את שדות max_vol_left/right לפי התוספת שהצומת הנוכחי קיבל בגלל קריאות עבר ל-increase: את שני הערכים נקדם בהפרש בין התוספת הנוכחית מצמתי האב (color_volume - vol_add_self) ל- max_vol_ref (התוספת שהייתה מצמתי אב בזמן העדכון האחרון של ערכים אלה). לאחר מכן נשווה בין נפחים אלה לנפח האפקטיבי של הכדור שהוספנו. אם המקסימום השתנה, נשנה את max_vol_left/right ו-max_vol_left/right_col בצומת האב ונעשה את אותו התהליך עבור צומת האב. נשים לב שלצומת האב יהיה ערך color_volume שונה, ונחשב אותו ע"י חיסור vol_add_self של הצומת הנוכחי ו-vol_add_left של צומת האב (במידה והגענו דרך קשת שמאל) והוספת vol_add_self של צומת האב. בנוסף, לא יהיה צורך לעדכן את max_vol_left/right ששינינו בהפרש בין color_volume ל-max_vol_ref מכיוון שערך זה כבר עדכני.

סיבוכיות זמן:

הליכה למטה בעץ הצבעים: O(log(k))

הוספת הכדור לערימה: O(log(n))

הליכה למעלה בעץ ועדכון שדות המקסימום: O(log(k))

סה"כ: O(log(k) + log(n))

PopMax():

נסתכל על שורש עץ הצבעים. נחשב את הנפח האפקטיבי של הכדור בראש הערימה שלו, נשווה אותו מול max_vol_left ו-max_vol_right ונסיק את מספר הצבע שמכיל את הכדור בעל הנפח המקסימלי (השורש, max_vol_left_col או max_vol_right_col). לאחר מכן נטייל בעץ עד שנגיע לצומת זה, נוציא את הצבע בראש הערימה של הצומת שהגענו אליו ונעדכן את ערכי המקסימום בצומת הנוכחי וצמתי האב שלו בדומה ל AddBall.

סיבוכיות זמן:

מציאת מספר הצבע עם הכדור בעל הנפח המקסימלי: O(1)

מציאת הצומת המתאים בעץ הצבעים: O(log(k))

הסרת הכדור מערימת הכדורים: O(log(n))

עדכון ערכי המקסימום: O(log(k))

סה"כ: O(log(n) + log(k))

Increase(i, j, d):

מכיוון שהתחומים רציפים, ניתן לשמור את התוספת d על תתי עצים בעץ הצבעים בעזרת השדה vol_add_left. התחום לא בהכרח יכלול את כל תת העץ השמאלי, אז נחלק את התחום [i,j] לשני תחומים: 1 עד j ו-1 עד i-1. את התחום הראשון ננפח ב-d ואת התחום השני נכווץ ב-d. בצורה זו התחום המבוקש התנפח ב-d והתחום 1 עד i-1 לא השתנה.

כדי לעדכן תחום מ1 עד m כלשהו לתוספת d: נתחיל משורש עץ הצבעים, נסמן את מספרו ב-r. אם r>m אז השורש וכל תת העץ הימני שלו לא בתחום, לכן נעבור לתת העץ השמאלי ונתחיל שוב את התהליך לאותו תחום 1 עד m. אם r<m אז השורש נמצא בתוך התחום, לכן כל תת העץ השמאלי ולפחות חלק מתת העץ הימני בתחום. נעדכן vol_add_self += d, vol_add_left += d, max_vol_left += d ונמשיך לתת העץ הימני. לא נעדכן את max_vol_right כי לא ידוע שכל תת העץ הימני בתחום. אם r=m אז השורש נמצא בגבול הימני של התחום, לכן כל תת העץ השמאלי גם בתחום. נעדכן את vol_add_left += d, vol_add_self += d ו- max_vol_left += d. לאחר מכן נחזור אחורה במסלול ונעדכן את ערכי המקסימום בכל הצמתים בדרך אם צריך.

הערות:

מובטח לנו שנמצא צומת עבורה r=m מכיון שאנחנו למעשה מבצעים חיפוש בעץ עבור הערך m.

מכיוון ששדות המקסימום בכל צומת לא כוללים תוספות מצמתי האב, נוכל לעדכן אותם לכל צומת בנפרד מבלי לפגוע בנכונות שלהם, ומבלי לעבור על כל הצמתים בתת העץ.

סיבוכיות זמן:

הליכה בעץ הצמתים ועדכון שדות תוספת הנפח: O(log(k)) לכל תחום, לשני תחומים גם O(log(k))

עדכון שדות המקסימום בצמתי האב: O(log(k)) לשני תחומים

סה"כ: O(log(k))

שאלה 2

א. מס' האיברים בערימת זנב מטיפוס k: הערימת זנב מורכבת מערימה המהווה עץ שלם בגובה k-1 ועוד עלה אחד. בעץ שלם בגובה k-1 ישנם 2^(k-1) -1 איברים ולכן בערימת הזנב ישנם 2^(k-1) איברים.

ב. <u>Union(H1,H2)</u>: נשים לב כי כל ערימת זנב הינה ערימת מינימום חוקית, לכן אם נקח את הזנב של ערימה H2 ונשתמש בו בתור שורש של ערימת מינימום חדשה כאשר הבן הימני הוא שאר H2 והבן השמאלי הוא H1 נקבל ערימת מינימום חוקית פרט לאיבר הראשון.
על מנת למצוא את האיבר זנב של H2, נלך על העץ המייצג את הערימת מינימום עד הסוף שמאלה. (הערימה מורכבת כעץ כמעט שלם אשר ברמה האחרונה יש לו עלה יחיד ולכן מובטח שעלה זה יהיה במקום הכי שמאלי). זה ייקח O(גובה H2) <= O(logn) משום שלכל היותר יש n-1 איברים בערימה H2 (ישנם סה"כ n איברים ב2 הערימות).
כעת על מנת לתקן את הערימה המאוחדת נעשה sift down של השורש החדש למקום הנכון שלו בערימה. זוהי ערימת מינימום של n איברים לכן הפעולה תיקן O(logn).
סה"כ O(logn).

ג. <u>MakeChainHeap(x1,…,xn)</u>: נעשה את הערימות באופן איטרטיבי:
נמצא את הk הכי גדול ככה שn>2^k ע"י לקיחת log של n ועיגול כלפי מטה.
נקח את 2^k האיברים הראשונים נסיר אותם מהרשימה ונבנה מהם ערימת מינימום.
O(2(2^k))=O(2^k). (פעם אחת בשביל לקחת את האיברים ולהסיר אותם ועוד אחד בשביל לבנות ערימת מינימום של 2^k איברים). (נשים לב כי תמיד נקבל ערימות זנב, כי נבנה את הערימות בתור עץ כמעט שלם וידוע כי בעץ כמעט שלם שיש בו 2^k איברים הוא בגובה k ויש בו עלה יחיד ברמה הנמוכה ביותר)
נקח את הערימת מינימום ונוסיף אותה לרשימה O(1).
נחזור על האלגוריתם הנ"ל עד אשר יסתיימו איברים, כלומר כל איטרציה לוקחת O(2^k).
נשים לב כי בכל איטרציה של האלגוריתם, אנחנו עושים מס' פעולות שהוא O של מס' האיברים שהורדנו ואנו לא חוזרים לאותו איבר פעמיים. לכן אם סך האיברים הוא n האלגוריתם כולו ייקח O(n).
נשים לב כי גם בגלל האופן שבו בוחרים את הכמויות איברים, נקבל ערימת שרשרת חוקית. משום שבכל איטרציה הk שייבחר יהיה קטן ממש מהk באיטרציה הקודמת. (נוכיח בשלילה, אם היו 2 איטרציות בו נבחר אותו k ישנם לפחות 2*(2^k)=2^(k+1) איברים ולכן באיטרציה הראשונה היה צריך להיבחר k+1 ולא k כי בוחרים את המעריך המקסימלי שניתן).

ד. <u>Merge(C1, C2)</u>:
נעזר בטענה כי: בערימת שרשרת שיש בה n איברים יש לכל היותר logn + 1 ערימות.
הערימת זנב הגדולה ביותר האפשרית בn איברים הינה לפי סעיף א כזו מסדר k המקיים כי n=2^(k-1) כלומר k = logn + 1. אם נסתכל על כל ערימת זנב בשרשרת בתור ביטים, כאשר הביט הi מייצג כי קיימת בשרשרת ערימת זנב מסדר i, סכום האיברים בערימה יהיה המספר שנקבל בייצוג בינארי. נשים לב כי עבור n איברים, ניתן לייצג את n כמספר בינארי בן O(logn) ביטים. ולכן בערימת שרשרת המכילה n איברים יש O(logn) ערימות זנב.
נרצה לבצע merge sort בין 2 הערימות, לצורך כך נצטרך לדעת את העומק הערימה (המרחק בין השורש לזנב). כלומר לכל השוואה בין 2 הערימות, נבצע הליכה בכל אחת מהערימות זנב הנוכחיות עד הסוף שמאלה (כי אנחנו כבר יודעים ששם נמצא הזנב) ונראה איזו ערימה גדולה יותר ונכניס אותה לערימת שרשרת החדשה שאנחנו בונים.
סה"כ פעולה הmerge תיקח O((logn)^2) זמן. בכל השוואה בודקים את עומק 2 הערימות הנוכחיות שזה במקרה הגרוע ערימה עם O(n) איברים ולכן היא בגובה logn. ויש במקרה הגרוע סה"כ 2logn השוואות (סכום 2 גדלי הרשימות).
נשים לב כי במסגרת פעולת הmerge הנ"ל ייתכן כי יהיו לנו 2 ערימות זנב באותו גודל שנכנסות לערימת שרשרת – דבר לא חוקי. לכן בסוף פעולת הmerge נעבור על הערימת שרשרת החדשה שבנינו ונבצע איחוד כל 2 ערימות זנב בגודל k לערימת זנב בגודל k+1 באמצעות פונקציית union

מסעיף ב, כאשר כל פעולת union תיקח במקרה הגרוע O(logn) זמן (אם מאחדים 2 ערימות זנב בעלי O(n) איברים כל אחת).
נעשה את הפעולה הזאת מהסוף להתחלה (כלומר נתחיל בערימת זנב הקטנה ביותר בערימת שרשרת). במקרה הגרוע, 2 הרשימות C1, C2 בעלות זהות מוחלטת בגדלי הערימות זנב שיש בהן, כלומר נצטרך לעשות איחודים כאורך הרשימה: O(2logn).
סה"כ נעשה O((logn)^2) איחודים.
לכן סה"כ סיבוכיות הפעולה היא O((logn)^2).

ה. <u>Insert(x,C)</u>: נסביר תחילה את אופן מימוש הפונקציה ולאחר מכן ננמק את הסיבוכיות של המימוש.
בכל הכנסה, נכנס לערימת שרשרת ערימת זנב מסדר 0 שהוא איבר x.
לאחר מכן נעבור על ערימת השרשרת C ונתקן אותה – כלומר נוודא שאין 2 ערימות זהות בגודלן, באופן דומה לאיך שעשינו זאת סעיף קודם. כלומר, נתחיל מהערימת זנב הקטנה ביותר, ונבצע איחודים עד אשר נישאר עם ערימת זנב שאין עוד אחת בגודלה.
נשים לב כי בניגוד לסעיף הקודם אין צורך לעבור עד סוף הערימת שרשרת משום שהבעיה היא רק באיבר הראשון – כלומר אם הוא בסדר, סיימנו. אם לא – נבצע איחודים עד שהוא יהיה בסדר ושאר השרשרת תהיה חוקית כי היא הייתה חוקית קודם.
נשים לב שהסיבוכיות של כל איחוד כנ"ל הינו O(logn) במקרה הגרוע, כי בערימת שרשרת עם סך איברים של n סך איברי כל ערימה יחידה שיש בה הוא גם O(n).
כעת נראה כי באופן משוערך כל אחת מההכנסות לוקחת O(logn)
עבור כל פעולה של הכנסה, נשלם מראש O(logn) פעולות אשר יהיה שייך לערימת זנב החדשה שיצרנו.
נוכיח באינדוקציה כי עבור כל הערימת זנב שיש בערימת שרשרת ישנו O(logn) פעולות ששמנו בצד "עבורה":
עבור הערימת זנב הראשונה, יוצרים ערימת זנב מסדר 0 בזמן O(1) ומכניסים אותו לשרשרת (O(1)) ומשלמים עליה O(logn). כלומר יש לה O(logn) פעולות ששמנו בצד "עבורה".
נראה כי בהנתן ערימת שרשרת קיימת, שבה יש לכל ערימת זנב שבה O(logn) פעולות בצד עבורה, נוכל לבצע את ההכנסה שתוארה מעלה ועדיין לקבל ערימת שרשרת המקיימת את התנאי.
נכניס את האיבר בתור ערימת זנב מסדר 0 בעלות של O(1) ונשים בצד עבורה O(logn) פעולות.
אם זו הערימת זנב היחידה מסדר 0 בערימת שרשרת, סיימנו וזה לקח O(logn) פעולות, ויש לנו עדיין ערימת שרשרת העומדת בתנאי.
אם זו אינה הערימת זנב היחידה מסדר 0, נצטרך לבצע איחודים. כל איחוד כנ"ל לוקח O(logn) פעולות, אותן ניקח מהO(logn) פעולות שיש בצד עבור הערימה שאיתה מאחדים את הערימה שהכנסנו – סה"כ האיחוד לא ייקח פעולות נוספות. כך נחזור עד שהערימת שרשרת חוקית. נשים לב שבתהליך זה לא נגענו בO(logn) פעולות שהיה לערימה החדשה שנכנסנו – וזה יהיה הO(logn) ששייך לערימה שיצאה מכל האיחודים שעשינו. כלומר קילבנו שוב ערימת שרשרת העומדת בתנאי.
כלומר, כל פעולת הכנסה לוקחת לנו O(logn) באופן משוערך.

שאלה 3

1. נשתמש בטבלת ערבול באורך n עם ערבול אוניברסלי כדי לאחסן כמה פעמים כל ערך נשמר במערך. אחרי שמקדמים מונה של ערך מסוים, משווים את הערך החדש שלו מול n/3. אם הוא שווה, נחזיר את הערך המתאים. אם הגענו לסוף המערך זה אומר שאף מונה לא הגיע ל n/3 ונחזיר שלא קיים איבר כזה.

סיבוכיות מקום: O(n)

סיבוכיות זמן:

מעבר על המערך: O(n)

מציאת המונה בטבלת הערבול: O(1) בממוצע הסתברותי

קידום המונה והשוואה: O(1)

סה"כ: O(n) בממוצע הסתברותי

2. נשתמש ב select() מהתרגול. נריץ את select(n/3), select(2n/3), select(n) ועבור כל מספר שחוזר מכל הרצה של select (שלושה סה"כ) נעבור על המערך ונספור כמה פעמים הוא נמצא. אם אחד מהם נמצא יותר מ n/3 פעמים נחזיר את הראשון מבינהם. אם כולם נמצאים פחות פעמים נחזיר שאין מספר כזה.

Select מוצא את האיבר במיקום המבוקש במערך ממוין. אם איבר מסוים נמצא יותר מ n/3 פעמים, אז במערך ממוין יהיה רצף של לפחות n/3 איברים רצופים שערכם איבר זה. ז"א שהמרחק המינימלי בין המופע הראשון לאחרון של האיבר הזה הוא n/3. לכן הוא חייב להופיע באחד מהמיקומים n/3, 2n/3 או n במערך הממוין.

סיבוכיות מקום: O(n)

סיבוכיות זמן:

הרצה של select אחת: O(n)

ספירת כמות מופעים של איבר: O(n)

סה"כ: O(3n+3n) = O(n)

שאלה 4

נשתמש בשני מבני Union/Find עם איחוד לפי גודל הקבוצה וכיווץ מסלולים. כל איבר במבנים האלה הוא חדר, כל קבוצה מאחסנת את מספר החדר הקרוב ביותר בצד מסוים - המבנה הראשון יהיה עבור החדר הקרוב ביותר מצד שמאל והמבנה השני לחדר הקרוב מצד ימין, כאשר מספר החדר הפנוי הוא הערך של הקבוצה.

מימוש הפעולות:

Init:

נאתחל שני מבני Union/Find באורך n. בהתחלה כל החדרים ריקים אז כל איבר בקבוצה משלו שערכה הוא מספר החדר עצמו.

סיבוכיות:

איתחול UnionFind אחד: O(n)

סה"כ: O(n)

WorkerArrival(k):

במבנה הראשון (חדר הקרוב מצד שמאל):

אם k=1 נשנה את ערך הקבוצה של k ל 1- כדי לייצג שאין חדר פנוי משמאל (כי הוא החדר השמאלי ביותר).

אחרת נבצע find על k ו- k-1 כדי למצוא את הקבוצות שלהם ונבצע union בין שתי הקבוצות, כאשר ערך הקבוצה המאוחדת יהיה הערך הקודם של k-1. זאת מכיוון שכאשר חדר k התמלא, החדר הריק הקרוב כעת תמיד יהיה החדר הקרוב ל k-1.

את המבנה השני נעדכן באופן דומה, רק שכעת הקצה יהיה ב k=n ונאחד עם k+1 במקום k-1.

ClosestRoom(k)

נבצע find(k) על שני המבנים ונקבל את החדר הקרוב ל-k מכל צד. אם שניהם 1- כל החדרים מלאים. אם אחד מהם 1- נחזיר את השני. אחרת, נקח את החדר שההפרש שלו מ-k (בערך מוחלט) מינימלי.

ניתוח סיבוכיות לWorkerArrival ו-ClosestRoom:

שתי הפונקציות מבצעות פעולות בזמן קבוע (מציאת הקצוות, מציאת מינימום בין זוג מספרים) ורצף של union ו- find על כל אחד משני מבני הUnionFind. לפי ההרצאה כל רצף של union ו-find הוא בסיבוכיות של log*(n) לכן כל רצף של שתי הפונקציות WorkerArrival ו- ClosestRoom הוא גם בסיבוכיות של log*(n).

שאלה 5

נוסיף למבנה רשימה של רשימות מקושרות כדי לשמור את הפופולריות של כל צומת. כל רשימה תהיה שייכת לדירוג מסוים ותכיל רשימה של האיברים בעלי דירוג פופולריות זה. האורך המקסימלי של רשימה זו היא n, למקרה שלכל איבר דירוג פופולריות שונה.

הראש של כל רשימה יכיל את הדירוג עצמו, ומצביע לראש של תת-הרשימה שלו שמכילה את האיברים עם דירוג זה. את תת-הרשימה נוכל לממש בעזרת הרחבה של המידע שנשמר על כל איבר: בנוסף לערכים הקיימים בunion/find נוסיף לו ערך previous ו-next שישמשו למעבר אחורה וקדימה ברשימה לפי פופולריות, וכן ערך head המצביע לראש הרשימה שלו. המקום הנוסף הוא O(n) לכן לא נפגעת סיבוכיות המקום של union/find.

כאשר נבצע find() על איבר, נקדם את מונה הפופולריות שלו ע"י העברתו לראש רשימה המתאימה לערך הפופולריות החדש שלו, המקודם ב-1. אם אין רשימה כזאת (או שהרשימה הבאה עם ערך גדול מדי או שהגענו לסוף רשימת הדירוגים) ניצור אותה ונכניס אותה במקום המתאים. אם הרשימה שהערך יצא ממנה התרוקנה, נמחק אותה ונוציא אותה מרשימת הדירוגים. הסיבוכיות של כל הפעולות האלה היא O(1) לכן לא פגענו בסיבוכיות הזמן של find().

מימוש הפעולות החדשות:

PrintPopularity():

נעבור על רשימת הדירוגים מההתחלה, ולכל דירוג נעבור על תת הרשימה שלו ונדפיס את האיברים לפי הסדר שהם מופיעים ברשימה. מכיוון שרשימת הדירוגים ממוינת, האיברים יודפסו לפי סדר דירוג פופלריות. מכיוון שסדר האיברים בעלי אותו הדירוג לא משנה, סדר ההופעה בתת-הרשימה מספיק.

סיבוכיות זמן:

נעבור על סה"כ n איברים וO(n) ראשים של רשימות דירוגים. סה"כ: O(n)

סיבוכיות מקום: O(1)

PrintByGroup():

נשתמש במנגנון דומה ל PrintPopularity כדי להשיג את כל האיברים ממוינים לפי גודל, ואז נמיין אותם לפי קבוצה בעזרת count sort. תוך כדי המעבר על האיברים נוכל גם לדעת את סדר הקבוצות הנכון, לפי האיבר הכי פופולרי, ונשמור אותו במערך נפרד. לאחר מכן נדפיס את האיברים בקבוצות לפי הסדר, כאשר נמצא כל קבוצה במערך האיברים לפי מידע על גודל הקבוצות עם מספר נמוך ממנה.

נקצה שלושה מערכים בגודל n, נקרא להם: elements, group_info, group_order. את group_info נאפס. לאחר מכן נבצע הליכה על רשימת הדירוגים, בדומה ל PrintPopularity. על כל איבר נבצע:

find() כדי לקבל את מספר הקבוצה שלו, ונכניס את מספר הערך ומספר הקבוצה למערך elements למקום הפנוי הראשון.

נסתכל על מערך group_info במיקום לפי מספר הקבוצה - אם נמצא בו הערך 1 אז ראינו כבר את הקבוצה הזו ולכן נדלג על האיבר. אם נמצא בו הערך 0 אז לא ראינו את הקבוצה הזאת קודם והאיבר הנוכחי הוא האיבר עם הדירוג המקסימלי בקבוצה זו, לכן נכניס את מספר הקבוצה למקום הפנוי הראשון ב group_order ונסמן 1 במקום המתאים ב group_info.

לאחר מעבר על כל האיברים בצורה זו נקבל ש elements מכיל זוגות של איברים ומספר קבוצה, מסודרים לפי הדירוג של האיברים ושמערך group_order מכיל את מספרי הקבוצות מסודרים לפי הדירוג של האיבר המקסימלי. כעת נבצע count sort על מערך elements לפי מספר קבוצה, ומכיוון ש-count sort הוא מיון יציב נקבל שמערך elements ממוין לפי מספר קבוצה וממוין משנית לפי דירוג האיברים בכל קבוצה.

כדי להדפיס את הקבוצות לפי מערך groups_order צריך לדעת איפה מתחילה כל קבוצה במערך elements. לשם כך נכניס לכל מקום במערך group_info את הסכום של גדלי כל הקבוצות שמספרן קטן ממספר הקבוצה שבמקום זה. נוכל לעשות זאת ע"י מעבר אחד על מערך גדלי הקבוצות של ה-union/find וחישוב סכום רץ. כעת groups_info מכיל את כמות האיברים שעלינו לדלג עליהם במערך elements כדי להגיע לתחילת רצף האיברים של כל קבוצה.

כעת נותר רק להדפיס את הקבוצות: נעבור על מערך group_order, ולכל קבוצה נמצא את האינדקס ההתחלתי שלה במערך elements לפי המקום המתאים במערך group_info. לאחר מכן נדפיס size איברים רצופים החל מאינדקס זה, כאשר size הוא גודל הקבוצה לפי מערך גדלי הקבוצות של ה-union/find. כשהגענו לסוף מערך elements נעצור.

סיבוכיות זמן:

הקצאת 3 מערכים לעבודה: O(1)

מציאת הקבוצה ל-n איברים: O(nlog*n). לפי הגדרת הסיבוכיות המשוערכת, ביצוע n פעולות find לוקח זמן O(nlog*n)

עדכון מערך group_info בפעם הראשונה: O(1) לכל איבר, סה"כ O(n)

count sort על מערך elements באורך n איברים: O(n)

מעבר על מערך גדלי הקבוצות ועדכון groups_order בפעם השניה: O(n)

הדפסת n איברים: O(n)

סה"כ O(nlog*n)

סיבוכיות מקום:

3 מערכים באורך n איברים: O(n)

count sort על מערך באורך n איברים: O(n)

סה"כ: O(n)