**הסבר אלגוריתם**

**הסבר כללי על השיטה הראשית – Main**

\*נציין כי במטלה זו **לאורך כל התוכנית** יש סינון **לכפילויות** ברשומות.

ראשית המשתמש מקבל פלט מהממשק שקורא לו לבחור האם הקלט שברשותו ינתן דרך הקונסול או דרך קובץ,

**אם הוא בחר את אופציית הקובץ:** הוא יקבל פלט נוסף מהממשק שיקרא לו לבחור שם \*תקין\* של קובץ , בדיקת התקינות(קיום הקובץ בתיקיית הפרויקט) תתבצע ע"י השיטה file\_open , כאשר נקלט מהמשתמש שם קובץ תקין , הקובץ יפתח, יעביר את תוכנו אל תוך המשתנה הלוקאלי file ע"י שימוש בשיטה readlines של השפה ולאחר מכן יוחזר ערך זה מהשיטה(בסיום עבודתה) אל תוך המשתנה הלוקאלי של main בשם file .

לאחר מכן המשתנה listTypeInput בשימוש השיטה rstrip יקבל את סוג הרשימה שהמשתמש בחר מהאפשרויות : ממוינת , לא ממוינת, ולא ממוינת זרים,כאשר :

ממוינת := Sorted

לא ממוינת := UnSorted

לא ממוינת זרים := Foreign

**אם הוא בחר את אופצית הקונסול:** המשתמש יקבל תפריט לבחירת סוג הרשימה מהאפשרויות שהוגדרו לעיל, כאשר לכל אחד מהאפשרויות משויך מספר.

בשני האפשרויות אופציית סוג הרשימה תהיה מספרית ותסמל איזו סוג רשימה תיבנה מפקודת MakeHeap למשך המשך התוכנית .

אותו מופע מחלקה של הרשימה הרלוונטי יתווסף לסוף רשימת listOfLists ועליו יופעלו הפקודות הבאות עד שיווצר מופע מחלקה חדש ע"י שימוש בפונקציה MakeHeap בשנית.

בכל איטרציה של לולאת הwhile תתבצע פקודה נוספת מתוכן הקובץ ע"י גישה למשתנה file באינדקס fileInd הנוכחי(אשר גדל בכל איטרציה) / ע"י הקונסול.

איטרציות אלו יימשכו עד הגעה לסוף הקובץ / עד שתנתן הפקודה exit שתוביל לשבירת לולאת ה while האינסופית, ולסיום התוכנית.

**הסבר על שיטות הרשימות השונות וניתוח סיבוכיותם:**

\*פקודת MakeHeap בונה רשימה באותו אופן עבור כולם,ומבצעת השמה שלהם אל תוך המשתנה Lst אשר לאחר מכן מתווסף לסוף הרשימה ListOfLists .

**סיבוכיות:** הגודל הדיפולטיבי של הרשימה יספיק לטובת פרויקט זה ולכן לא יגדיל את עצמו , ולכן הסיבוכיות של ההוספה אל סוף הרשימה תהיה (1)O וכך גם שאר הפעולות, ולכן הסיבוכיות הכללית תהיה **(1)O.**

**רשימה ממוינת**

1. **פקודת Insert :**

\*במידה והרשימה הייתה ריקה, הוספת האיבר החדש מתבצעת ע"י השמת תוכן לראש הרשימה והגדלה ב1 של משתנה הספירה של הכמות צמתים ברשימה ולכן הסיבוכיות של הפקודה תהיה ב (1)O .

\*אחרת, אל ראש הרשימה הנוכחית יצביע המשתנה Curr ו המשתנה Prev גם הוא יצביע לאותו מקום.

נכנס ללולאה אשר תפסיק את פעילותה במידה ותיעצר בצורה יזומה ע"י פקודה Return אשר תסיים גם את פעולת השיטה כולה, או ע"י כך שהמצביע Curr יתקדם מספיק צעדים ויצביע לבסוף על אזור בזכרון שאיננו שייך לרשימה, כלומר במילים אחרות שיסיים את המעבר על הרשימה.

בתוך הלולאה מתבצעת ראשית השוואה אם התוכן של הצומת הנוכחית ברשימה המקושרת הינה בעלת ערך זהה לערך שניתן כארגומנט לשיטה (אשר התבקשנו להכניס לתוך הרשימה) , אם זה המצב זה אומר שהצומת כבר קיימת ולכן לא נוסיף כפילות שלה כפי שציינו לעיל.

אחרת אם ערך הצומת הינו קטן מהערך value שהתבקשנו להכניס, כיוון שרשימה זו אנחנו דואגים שתהיה ממוינת, נשווה את המשתנה prev אל המיקום הנוכחי של Curr ולאחר מכן נקדם את Curr אל הצומת הבאה בתור.

אחרת,(ערך הצומת גדול מהערך value) ניצור צומת חדשה אשר תצביע בשדה ה next שלה אל הצומת של curr (זאת עם הערך הגדול מ value ) ונעדכן את שדה ה value שלה לארגומנט value שניתן לשיטה.

לאחר מכן,

אם המצביע curr עצר את עבודתו כאשר הצביע על צומת הראש ברשימה, זה אומר ש value קטן מכל הצמתים ברשימה הממוינת שכן קטן מהצומת הראשונה שהינה הקטנה ביותר, ולכן נגדיר את מצביע ראש הרשימה אל temp שכן הוא הפך להיות הצומת בעלת הערך הקטן ביותר ברשימה.

אחרת, זה אומר שהצומת הינה צומת פנימית בתוך הרשימה (אינה אחרונה ואינה ראשונה) ולכן נגדיר ששדה הnext של הצומת prev יהיה temp ובכך ניצור את החיבור של הצומת temp במקום המתאים לה בסדר עולה של ערכים בתוך הרשימה.

ולבסוף יגדל ערכו של המונה size אשר סופר את כמות הצמתים ברשימה, ותסתיים השיטה ע"י הפקודה return .

במצב בו נשברה הלולאה ע"י כך ש המצביע curr עבר כבר על כל איברי הרשימה ולא מצא מיקום מתאים, זה אומר שכל האיברים היו עם שדה value קטן מה value

שניתן כארגומנט לשיטה ולכן נוצרת צומת חדשה בעלת הערך value אשר prev שהינו המצביע על האיבר האחרון ברשימה, מצביע עם שדה ה next שלו על אותה צומת, ולבסוף כמובן מוגדל המונה size ב1.

**סיבוכיות השיטה:**

מלבד המעבר על כל הצמתים ברשימה הנוכחית שהוא בסיבוכיות (O(n,

וזה במקרה הגרוע ביותר בו value גדול מערך כל הצמתים ברשימה הנוכחית,

כל שאר הפקודות התבצעו בסיבוכיות קבועה כלומר O(1) ולכן סיבוכיות השיטה הכללית היא O(n) .

1. **פקודת ExtractMin :**

כיוון שבמהלך השיטה דאגנו להכניס לרשימה את האיברים בסדר ממוין מקטן לגדול, האיבר הקטן ביותר יהיה האיבר הראשון ולכן שמרנו את ערכו ע"י פונקציית ה getter ,

get\_val אל תוך המשתנה val, ראש הרשימה התקדם לצומת הבאה ברשימה המקושרת, חוסר האיבר מתוך ספרן האיברים (self.size) ולבסוף הוחזר הערך val .

**סיבוכיות השיטה:**

כל הפעולות שבוצעו היו בסיבוכיות קבועה ולכן כך תהיה גם סיבוכיות השיטה, (1)O .

1. **פקודת Minimum :**

אם ספרן הצמתים ((size גדול מ 0 מחזיר את תוכן צומת הראש,

אחרת , מחזיר סטרינג למשתמש על כך ש אין מינימום כי הרשימה ריקה.

**סיבוכיות השיטה:**

הקריאה אל ראש השיטה מתבצעת בסיבוכיות קבועה ע"י גישה ישירה , וכך גם יצירת הסטרינג שיש להדפיס ולכן סיבוכיות השיטה כולה קבועה כלומר (1)O

1. **פקודת Union :**

\*נקרא לרשימה שעליה הופעלה השיטה A ולזו שהוכנסה כארגומנט B

אם מונה האיברים( (sizeשל הרשימה שעליה מופעלת השיטה שווה ל 0, זה אומר שהרשימה הינה ריקה ולכן מוחזרת הרשימה השניה , אם גם אם היא גם ריקה מהתנאים של ה else נקבל רשימה ריקה כנדרש.

**אחרת**((else,רצה לולאת whileאשר פועלת כל עוד לפחות אחת מהרשימות לא ריקה.

\*נקרא למצביע לצומת של רשימה A currA ומצביע לצומת של רשימה B \*currB

\*נגדיר בכל איטרציה את המשתנה temp להצביע על Node(צומת) ריק חדש \*

**אם** currB לא מצביע על מקום בזכרון שהוקצא ל B (כלומר אם הרשימה ריקה / המצביע התקדם עד שיצא מתחום הזכרון) **או** currAמצביע על איזור בזכרון של הרשימה של A וגם התוכן של currA הוא קטן מהתוכן של currB אזי נכנס לתנאי.

**בתנאי** נגדיר Aval בתור הערך של currA , נעדכן את ערכו של temp לערכו של currA ונקדם את currA לצומת המקושרת הבאה אחריו.

**אחרת אם** מנימוקים אנלוגיים לחלוטין **לאם הקודם** פרט לשינוי לכך שהבדיקה הפעם על כך שערך הצומת קטן מהשני היא על ידי קטן שווה ולא רק קטן.

ממשיכים להמשך הקוד ונתקל ב-

**אם** הרשימה החדשה שיצרנו dummylinkedLst עדיין ריקה כלומר עודלא הכנסנו אליה את האיבר הראשון, אזי הראש יוגדר לאיבר הראשון ע"י הצומת temp שקיבלנו קודם ובנוסף ספרן הצמתים יגדל ב1.

**אחרת אם** הערך של הצומת הנוכחית(האחרונה עדכה) ברשימה החדשה שונה מהערכים מערכים שהיו לרשימה A ו B באיטרציה הנוכחית נוכל לדעת שאנחנו מוסיפים איבר ללא כפילות ולכן נגדיר אותו בתור הצומת הבאה ברשימה החדשה ונגדיל את ספרן הצמתים ב1.

לבסוף, ללא תלות בתנאי , המשתנים Aval וBval יאופסו לערכי ברירת מחדל,

ובסיום הלולאה,כלומר כאשר סיימנו לעבור על כל צמתי רשימה A ו B נחזיר את הרשימה החדשה שיצרנו.

**סיבוכיות השיטה:**

מתקיים מעבר ליניארי יחיד כלומר O(n) על שני הרשימות A ו B והוספת צמתיהם לרשימה חדשה,

כל הפעולות מלבד האיטרציה עצמה על הצמתים הם פעולות בסיבוכיות קבועה כלומר (1)O , לכן נקבל שסיבוכיות השיטה כולה היא (O(n .

**רשימה לא ממוינת**

1. **פקודת Insert :**

השיטה נפתחת בתנאי **אם הרשימה ריקה**, אז כיוון שצומת הראש כבר הוגדרה בפקודת MakeHeap אז ערך הצומת יוגדר להיות הערך שהתקבל בשיטה וזנב הרשימה יכוון את הצבעתו לראש הרשימה.

**אחרת אם,** מתבצעת בדיקה שהערך לא נמצא ברשימה עדכה, אם אכן זה הדבר מוגדרת צומת חדשה עם ערך זה אשר מצביעה להיכן שמצביע ראש הרשימה, ואז משתנה ההצבעה של ראש הרשימה אל הצומת החדשה וספרן הצמתים גדל ב1.

לבסוף מתבצעת בדיקה אם האיבר שהתקבל הוא קטן יותר מהמינימום שנמצא עדכה, ומשתנה המינימום מתעדכן בהתאם, מתבצעת הוספה של הערך ל קבוצת הערכים elements (כמובן שאם ערך זה קיים זה לא ישנה דבר כי זו קבוצה) וספרן הצמתים גדל ב1.

אם נקבל כי הגודל של הרשימה גדול מ1 ( כלומר ישלה 2 צמתים לפחות) וגם הערך שהתקבל כארגומנט איננו המינימום של הרשימה , נבצע בדיקה אם הערך הזה קטן מהערך של משתנה המינימום המשני,ונעדכנו בהתאם.

**סיבוכיות השיטה:**

אין לולאות וכל השיטות והפעולות מתבצעות בזמן קבוע,

במקרה של בדיקת שורת ה "אחרת אם" כיוון שSet הינו hashtable אנחנו נסיק כי

במקרה הממוצע הפעולה תתבצע ב O(1) ובמקרה הגרוע תתבצע ב O(n).

**(2פקודת ExtractMin :**

השיטה כולה תתבצע רק במידה וקיים איבר אחד לפחות ברשימה.

במידה וזה המצב, נגדיר את curr כמצביע לראש הרשימה, נגדיר את minValLocPrev להיות הצומת לפני הצומת בעלת הערך הקטן ביותר(במקרה בו הצומת הקטנה ביותר היא הראש נטפל פרטנית), אך אתחול משתנה זה יהיה הראש.

ואת minToRemove בערך של מינימום הרשימה שכבר ידוע לנו שאותו נרצה להסיר.

נכנס ללולאה אשר תרוץ כל עוד יש צומת הבאה בתור אחרי curr ,

בתוך הלולאה נבדוק **האם** תוכן הצומת הבאה בתור(אחרי curr כמובן) היא הצומת הרצויה, אם זה הוא המצב נעדכן את המשתנה minValLocPrev להצביע על curr .

**-אחרת** נבצע בדיקה האם תוכן צומת זו קטנה מתוכן המינימלי המשני,נעדכנו בהתאם

בסיום ריצת הלולאה נמשיך לבדיקת תנאים נוספים.

**-אם** minValLocPrev מצביע לאותה כתובת כמו הראש, זה אומר שתוכן צומת הראש היא עם הערך מינימלי **או** הצומת שסמוכה לה, נתייחס לזה כעת נקודתית.

**-אם** הצומת שסמוכה לצומת הראש בעלת הערך המינימלי, נניח ונסמן את הצומת הסמוכה לצומת הראש ב 1 ואת הצומת שסמוכה לה ב 2 , אז נגדיר שהצומת הבא בתור לצומת הראש היא 2 במקום 1 ובכך נוציא את צומת 1 משרשרת הצמתים של הרשימה כלומר נסלקה מהרשימה.

**-אחרת** כלומר במידה וצומת הראש היא הצומת בעלת הערך המינימלי, נקדם את הראש לצומת הבא בתור ובכך נסלקה משרשרת הצמתים של הרשימה כלומר נסלקה מהרשימה.

כעת לאחר שסיימנו את המקרים הנקודתיים נעבור **לאחרת אם** הבא בתור ,

**-אחרת אם** הצומת בעלת הערך הקטן ביותר היא צומת הזנב כלומר האחרונה, נסיק כי על מנת להסירה נגדיר את הצומת שמצביעה אל זנב הרשימה שתצביע אל None כלומר נקבל שתוסר הצומת האחרונה ברשימה שכן כעת אף צומת לא מצביעה עליה.

בנוסף המשתנה tail יצביע אל הזנב החדש.

**-אחרת** הצומת בעלת הערך הקטן ביותר היא בין צמתים,לכן מאותם נימוקים כמו לעיל אם הצומת הבאה היא 1 ואחריה 2,נבצע הצבעה ע"י minValLocPrev אל צומת 2,

דבר אשר ינתק את צומת 1 מן השרשור של הרשימה ולכן ימחק אותה. לבסוף, נמחוק את הערך מ elements ,נעדכן את המינימום ונסדר את המצביעים.

**סיבוכיות השיטה:**

כל הפעולות בשיטה מבוצעות בסיבוכיות קבועה כלומר O(1) .

לעומת זאת, קיימת לולאה אשר רצה על כל צמתי הרשימה ולכן O(n) הינה הסיבוכיות

**3) פקודת Minimum :**

יורשת מ linkedLst, **אם** ספרן הצמתים אומר כי הרשימה לא ריקה ע"י בדיקה שאיננה 0, הוא מחזיר לנו את הערך של האיבר המינימלי ביותר ברשימה הנוכחית, אחרת מחזיר לנו שאין כזה משום שהרשימה ריקה.

**סיבוכיות השיטה:**

המשתנה של המינימום כבר עודכן מן השיטות הקודמות ומנוהל על ידם ולכן הגישה אליו הינה ישירה ב O(1) כלומר קבועה .

**(4פקודת : Union**

\*נסמן את הרשימה שעליה הופעל השיטה בA ואת הרשימה שהוכנסה כארגומנט בB

**אם** ספרן הצמתים הוא 0 , כלומר הרשימה ריקה , הקבוצה של איברי הרשימה תהפוך לקבוצת איברי B והרשימה של B מוחזרת.

**אחרת ,** נכנס ללולאה כאשר כל עוד הרשימה B בעלת איבריםתמשיך לרוץ.

המשתנה item יקבל את ערך איבר הראש של רשימה B , ע"י פונקציית העזר pop אשר מוציאה את ראש הרשימה , מעבירה את ההצעה של ראש הרשימה לצומת הבאה,מורידה 1 מספרן הצמתים של הרשימה ומחזירה את ערך הצומת שהוסרה.

**אם**  item הוא בעצם איבר אשר קטן מאיבר המינימום של A אז יעודכן ערכו של איבר המינימום של A.

**אחרת** יבדק אם ערכו קטן מאיבר המינימום המשני ואם זה המצב,ערכו יתעדכן בהתאם.

לאחר מכן הערך item יצא מקבוצת הערכים של רשימה B ויוכנס אל הרשימה של A

לבסוף תוחזר הרשימה A .

**סיבוכיות השיטה:**

במקרה הטוב ביותר, הרשימה A ריקה ותוחזר רשימה B כלומר סיבוכיות O(1).

במקרה הממוצע יתקיים מעבר ליניארי על כל איברי B וinsert יעבוד ב O(1) ונקבל כי הסיבוכיות כולה תהיה (m)O שכן אר הפעולות מתקיימות בסיבכויות קבועה .(O(1))

במקרה הגרוע ביותר, יתקיים מעבר ליניארי על איברי B ב O(m) ויתבצע insert בסיבוכיות של O(n) ,שאר הפעולות כאמו בסיבוכיות קבועה ולכן נקבל O(n\*m).

**רשימות לא ממוינות וזרות**

\*ראשית נציין כי רשימה זו יורשת מ רשימה לא ממוינת,לכן מלבד פקודת Union וכיוון שהגדרנו שלאורך כל התוכנית איננו שומרים כפילויות בתוך הרשימות, נקבל כי משתמשת בשיטות של הרשימה הלא ממוינת.

**פקודת : Union**

**\***נסמן את הרשימה שעליה הופעלה השיטה ב A ואת זאת שהוכנסה כארגומנט בB

**אם** ספרן הצמתים הוא 0 זה אומר שהרשימה ריקה ולכן מחזיר את רשימה B.

**אחרת**, המינימום של 2 הרשימות מוגדר להיות המינימום החדש של A,

המינימום המשנה באותו אופן גם כן,לאחר מכן זנב הרשימה של A מצביע על ראש הרשימה של B,מונה הצמתים מתעדכן לכמות הצמתים של 2 הרשימות יחד,

אנחנו יודעים כי בוודאות אין לנו כפילות מההנחת קלט ולכן יכולים לעשות זאת,ולבסוף מוחזרת רשימה A.

**סיבוכיות השיטה:**

השיטה מתבצעת ב (1)O שכן כל הפעולות בה הם קבועות וישירות ואין פעולות איטרטיביות