מבנה נתונים:

מיונים :

מושגים:

- מיונים לא השוואתיים : מיון שאינו משתמש בהשוואה בין אברי המערך.
- מיון יציב : אם ישנם אברים שווים מבחינת השוואה הסדר שלהם נשאר אותו סדר כמו במערך המקורי. (merge sort ,bubble sort, counting sort)
- מיון לא יציב : לא בהכרח שהסדר המקורי נשמר.(selection sort , quick sort)

• חשוב שהסדר יהיה יציב, נניח בשביל הדוגמא שיש רשימת אנשים המכילה שתי שדות – שם פרטי ושם משפחה , אזי קודם נמיין את הרשימה לפי שם פרטי ואז לפי שם משפחה ובמידה ויש שם משפחה שווה אז הסדר נשאר כמו שיהיה קודם בצורה המקורית.

מיון מניה – Counting Sort :

בהינתן מערך שכל אבריו חסומים בטווח מסויים , נמיין אותו בO(n) .

דוגמא : נניח שנתון :

3	0	1	0	1	2	1	4	5	2

כלומר : מערך בטווח חסום [0,5].

נגדיר מערך חדש בגודל 1+K כלומר בגודל 6 (כל המספרים צריכים להיות עד 5,נצטרך למצוא את האיבר המינימלי והמקסימאלי בכדי לדעת את הטווח (range)) .

נקרא למערך counter ונוסיף את האיבר המינימלי במקום של המערך הנתון ונקבל מערך בו נדע כמה פעמים כל אינדקס מופיע.

0	1	2	3	4	5
2	2	2	1	1	1

נעבור על כל המערך ובכל פעם נוסיף את המקום שבו אנו נמצאים כמו הפעמים שהוא מופיע (למשל 0 פעמיים).

שלבים:

נגדיר את גודל המערך conter[range]=max-min +1

ומכאן נלך לאיבר פחות המינימום ונקבל את המערך counter (counter[i]=arr[i]-min).

ושנחזיר למערך המקורי ניקח את האינדקס ונוסיף לו את המינימום. (arr[i]=counter[i]+min).

<u>סיבוכיות :</u> O(n + range) , אבל אם range = O(n) אזי הסיבוכיות היא O(n) .

• חסרונות – מערך זה מתאים רק למספרים חיוביים ולטווח חסום בלבד.

<u>מיון מהיר – Quick Sort :</u>

במיון מהיר הרעיון הוא לקחת איבר שהוא הציר וסביב הציר הזה לבנות את האברים כך שבצד ימים כל האיברים הגדולים ממנו ובצד שמאל כל האיברים הקטנים ממנו ובכך הציר יהיה בדיוק במקומו הנכון.

בדר"כ נבחר את הציר להיות האיבר הראשון במערך ונקרא לו pivot , pivot = arr[0] .

החלוקה לשני החלקים נקראת partition – החלקים לא בהכרח חיביים להיות שווים.

נשתמש במצביעים בשמות left ו right כאשר אותם אנו בודקים אם הם גדולים או קטנים מה pivot ובהתאם לכך עושים swap .

- במיון מהיר עובדים עם הכתובות.
- מיון מהיר הינו רקורסיבי.

<u>סיבוכיות :</u> אם המערך ממויין אזי הסיבוכיות במקרה הטוב היא O(nlog n) ובמקרה הגרוע O(n^2).

<u>מיון בסיס – Radix Sort :</u>

מיון זה הינו מיון לא השוואתי שגם שייך למספרים שלמים על בסיס 10 כלומר : בעל 10 ספרות מקסימום.

תור ומחסנית :

מחסנית – Stack , ADT:

הגדרה : המבנה פתוח רק מלמעלה וסגור מלמטה ,כלומר מי שנכנס אחרון יוצא ראשון.(lifo)

יתרונות :

- חישוב ביטויים אריטמטיים.

- שמירת נקודות GPS לאורך דרך ע"מ לחזור בחזרה עליהם.

פעולות :

- Push – תכניס איבר לראש המחסנית.
- Pop – מחיקת / הוצאת האיבר הראשון שנכנס.
- Top - תחזיר את האיבר האחרון (התבוננות באיבר האחרון).
- IsEmpty – האם המחסנית ריקה.
- Empty/clear– תרוקן את המחסנית.

ייצוג/מימוש מחסנית :

1. ע"י מערך – נבנה מערך בגדול מקסימלי והTOP שלי יהיה מי שנכנס אחרון.
   נאתחל את הTOP להיות 1- ,כלומר איבר שלא קיים (באינדקסים).
   דוגמא להכנסת איבר : push(3) - נעלה את הTOP באחד (עכשיו הוא 0) ונשים במקום ה0 את 3.
   במקרה והמחסנית מלאה נחזיר IsFull למשתמש.
   כאשר נרצה לעשות Pop – נקטין ב1 את הTOP .
   במחיקת אביר – לא נמחוק פיזית אלא נדרוס אותו ע"י ה push הבא.

2. ע"י רשימה מקושרת – הTOP יהיה שווה לHead והnext שלו יהיה שווה לnull – כאשר המחסנית ריקה.
   במצב שזה מלא – נגדיר חוליה שהnext שלה הוא הTOP ונעביר את הTOP להיות החוליה עצמה.

תור - Queue:

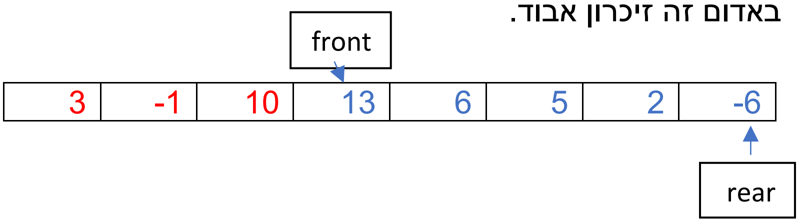
<u>הגדרה:</u> מבנה פתוח משני צדדיו כך שמי שנכנס ראשון יוצא ראשון .(fifo)

<u>פעולות :</u>

- Enqueue – נכניס לסוף התור.
- Dequeue – נוציא מראש התור.
- Front- מחזיר את האיבר הראשון בתור.(ע"י שני מצביעים)
- Rear- מחזיר את האיבר האחרון בתור.(ע"י שני מצביעים)

<u>ייצוג תור:</u>

1. ע"י מערך – נשתמש במערך מעגלי.
נתבונן במערך רגיל – בהתחלה שני המצביעים יצביעו על איבר שלא קיים (-1).
אם התור ריק נקדם את ה front והrear ונכניס את האיבר, ואם נרצה להמשיך להכניס איברים נקדם את הrear ונכניס.
כעת ישנה בעיה – גם בעיה שכאשר נגיע לסוף המערך שלי חסום (כלומר אני לא אוכל להוסיף), וגם בעיה של איבוד זיכרון – לדוגמא , כאשר במקרה בו מאחורי הfront נמצאים אברים אזי המחסנית שלי היא רק מה שמסומן בכחול ומה שמסומן באדום זה זיכרון אבוד.



לכן נגיד לreal ללכת להתחלה ולהוסיף שם איבר.

כלומר : אם הfront הוא לא האיבר הראשון תוסיף לראשון ואז נתחיל ב front ונתקדם במעגליות ע"י הנוסחא : i = i +1 % n , כאשר n = לאורך המערך.

2. ע"י רשימה מקושרת.

מבוא לתורת הגרפים :

מושגים :

- גרף לא מכוון – גרף אשר כל צלעותיו לא מכוונות (צלעות ללא כיוון).
- נקרא לשני קודקודים שכנים אם יש בינהם צלע.
- דרגה של קודקוד הוא מספר הצלעות היוצאות ממנו deg(v).
- מסלול – סדרה של קודקודים שכנים שמחברות שני קודקודים בגרף.
- אורך מסלול – מספר הצלעות שיש בו.
- מסלול פשוט – מסלול אשר לא עובר באף צומת יותר מפעם אחת.
- שורש – בגרפים מכוונים , הוא צומת שקיים מסלול ממנו לכל צומת אחרת בגרף.
- גרף שלם – הוא גרף אשר כל שני קודקודים בו מחוברים בקשת – כלומר כל הקודקודים מחוברים זה לזה.
- גרף לא מכוון נקרא קשיר אם קיים בו מסלול בין כל שני צמתים שונים (ניתן להגיע לכל קודקוד מקודקוד רחק ע"י מסלול).
- מעגל (פשוט) מסלול שמתחיל ומסתיים באותה צומת. (אם אני עובר בקודקוד מסויים פעמיים הוא יקרא מעגל).
- עץ – גרף קשיר ללא מעגלים. (כלומר ניתן להגיע מכל קודקוד לכל קודקוד ואין בהם מעגלים).
- עץ מושרש – עץ שקיים בו שורש.
- עומק / רמה – אורך המסלול בין השורש לצומת.
- אב ובן – צומת V היא האב של W ו - Wהוא הבן של V אמ"מ ישנה קשת בינהם ועומקו של V קטן באחד מעומקו של W.
- עלה – צומת שאין לה בנים.
- קודקוד פנימי – צומת שיש לו בנים.
- אב קדמון וצאצא – צומת V היא אב קדמון של W ו-W הוא צאצא של V אמ"מ W הוא הב של V או שW הוא הבן של הצאצא של V.
- גובה של צומת – מספר הקשתות במסלול הארוך ביותר בין הצומת לאחד העלים שלו.
- גובה העץ – הגובה של השורש (עומק של העץ וגובה של העץ שווים).

עץ בינארי :

הגדרה : עץ בינארי הוא עץ שבו לכל קודקוד יש לכל היותר שני בנים.

מאפיינים : - עץ בינארי מלא הוא עץ בו לכל צומת יש שני בנים בדיוק.

- עץ בינארי שלם הוא עץ בינארי מלא בו כל העלים באותה רמה.(כל שלם הוא מלא אך לא להפך).

- עץ בינארי כמעט שלם הוא עץ בינארי שלם שהוציאו ממנו עלים מצד ימין.

סיורים על עצים:

- DFS - מתחיל מצומת שרירותית (אצלנו מהשורש) ומתקדם לאורך הגרף עד שהוא נתקע ואז חוזר על עקבותיו עד שהוא יכול לבחור להתקדם לצומת שאליה טרם הגיע.

  קיימות 3 גישות לסיור על עץ בינארי :
  1. Inorder – left , root,right.
  2. Preorder – root, left,right.
  3. Postorder – lefr,right,root.

  כאשר בכל פעם שנגיע ל root נדפיס אותו.
- BFS – מעבר לפי רמות : מתחיל את הסיור (אצלנו מהשורש) ועובר על כל הצמתים במרחק של צלע אחד מ V(0) ואז כל כל הצמתים במרחק של שתי צלעות וכן הלאה.
  (ניישם את זה ע"י תור – קודם נכניס את השורש ונשאל האם יש אברים בתור).

ייצוג של עץ בינארי : כל קודקוד של עץ בינארי יכיל data,right,left . כאשר צד ימין יקבל את הערך הגדול יותר ואילו צד שמאל את הערך הקטן יותר בלי הגבלת הכלליות (ייתכן גם ההפך).

הוספת איבר לעץ בינארי : ההוספה תיהיה אקראית – נתחיל מהשורש ונבדוק אם קיים שורש בכלל אם לא – אם לא קיים נוסיף לשורש ואם קיים אזי, נגריל מספר בין 0 ל1 (random) ונבדוק אם הוא קטן מחצי או לא – אם כן נלך ימינה ואם לא שמאלה, נתקדם עם התנאי הזה עד אשר נגיע למקום פנוי ("null") ונוסיף שם.

עץ חיפש בינארי :

הגדרה : עץ חיפוש בינארי הינו עץ בינארי כך שלכל קודקוד – כל הקודקודים שנמצאים בתת עץ השמאלי שלו קטנים ממנו וכל אלה בתת עץ הימני גדולים ממנו.

מאפיינים : 1.אם אדפיס inorder בעץ זה אקבל מיון לפי הסדר.

2. נשאף שתמיד הגובה של העץ יהיה O(log n) ואז זמן הריצה יהיה O(log n).

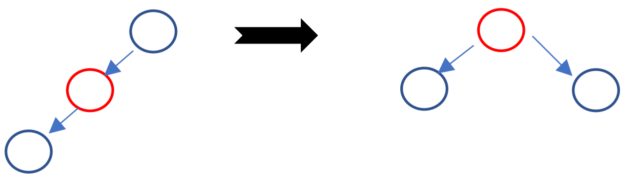
פעולות שניתן לעשות : חיפוש, הוספה, מחיקה.

מחיקת איבר : נחפש את X – אם לא מצאנו נחזיר FALSE אחרת נקרא לקודקוד שמכיל את X- V ,

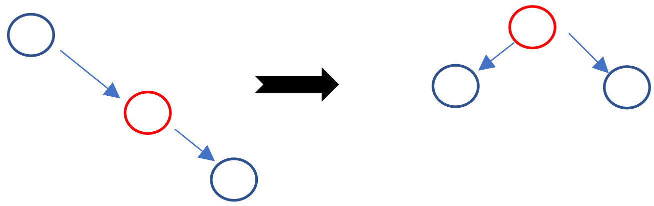
נבדוק אם יש בן אחד אזי ניתן לאבא של V להצביע על הבן היחידי של V ואם יש שני בנים – נלך הכי left ימינה ונחליף עם הקודקוד הרצוי למחיקה וניקח את הקודקוד הזה ונצביע על "null" , כלומר ניתן לאבא שלו להצביע על "null" .

רוטציות :

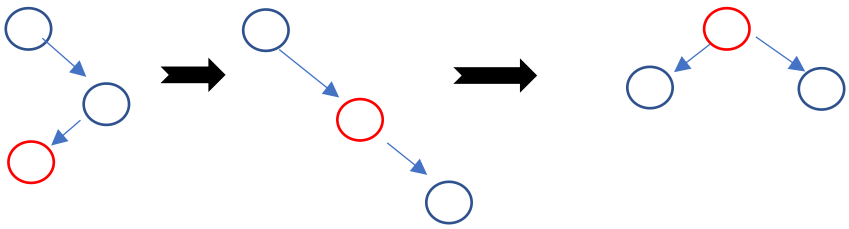
Left,left -



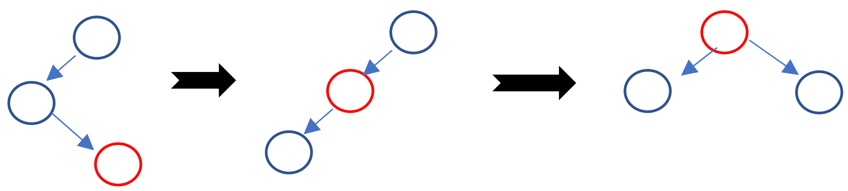
Right,right -



Right,left -



Left,right-



עץ AVL :

<u>הגדרה :</u> עץ AVL הוא עץ מאוזן כלומר לכל קודקוד ההפרש בין גובה תת העץ הימני שלו לגובה תת עץ השמאלי שלו הוא לכל היותר 1.

<u>מאפיינים :</u>

- עץ זה הינו מקרה פרטי של עץ חיפוש בינארי.
- גם בעץ זה מתקיימות רוטציות אך כיוון שהפרש יכול להיות גדול יותר ב1 הרוטציות כאן מסובכות יותר כלומר יכולה להיווצר בעיה על בעיה.
- עקרון לפתרון הבעיה – ניגש תמיד למי שטיפלנו בו אחרון ועליו נקיים את הרוטציה.

<u>סיבוכיות הוספה :</u> O(log n) כמספר הקודקודים של עץ AVL.

<u>הוכחת עץ AVL :</u>

<u>משפט :</u> יהי T עץ AVL בעל n צמתים וגובה h אזי h = O(log n) .

<u>הוכחה :</u>