חסמי סיבוכיות

יהיו $f(n), g(n)$ פונקציות חיוביות.

הסימן O (או גדול): נאמר ש-$f(n)=O(g(n))$ אם קיימים קבועים $c, n_0 > 0$ כך שלכל $n \geq n_0$:
$\boxed{f(n) \leq c \cdot g(n)}$.

הסימן Ω (אומגה גדולה): נאמר ש $f(n)=\Omega(g(n))$ אם קיימים קבועים $c, n_0 > 0$ כך שלכל $n \geq n_0$: $\boxed{f(n) \geq c \cdot g(n)}$.

הסימן Θ (תטא גדולה): נאמר ש $f(n)=\Theta(g(n))$ אם $\boxed{f(n)=O(g(n))}$ וגם
$\boxed{f(n)=\Omega(g(n))}$

הסימן o (או קטן):

1. נאמר ש $f(n)=o(g(n))$ אם **לכל** $c>0$ קיים $n_0 > 0$ כך שלכל $n \geq n_0$: $\boxed{f(n) \leq c \cdot g(n)}$

2. נאמר ש $f(n)=o(g(n))$ אם: $\boxed{\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0}$

הסימן ω (אומגה קטנה):

1. נאמר ש $f(n)=\omega(g(n))$ אם **לכל** $c>0$ קיים n_0 כך שלכל $n \geq n_0$: $\boxed{f(n) \geq c \cdot g(n)}$.

2. נאמר ש $f(n)=\omega(g(n))$ אם: $\boxed{\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0}$

טענות:

- לכל קבוע $k \geq 2$ מתקיים: $\sum_{i=0}^{n} k^i = O(k^n)$

- יהי $P(n)$ פולינום מדרגה k שמקדמיו חיוביים. כלומר: $P(n)=\sum_{i=0}^{k} a_i n^i \quad a_i \geq 0,\ a_k > 0$ אז מתקיים: $P(n)=\Theta(n^k)$.

- $\log(n!)=\Theta(n \log n)$

- לכל $\varepsilon > 0$ מתקיים: $\log n = o(n^{\varepsilon})$

משפטים על חסמי סיבוכיות:

נתונות 4 פונקציות $f_1(n), f_2(n), g_1(n), g_2(n)$ כך ש:

$$f_1(n)=O(g_1(n)) \qquad f_2(n)=O(g_2(n))$$

טענה-1: $f_1(n)+f_2(n)=O(g_1(n)+g_2(n))=O(\max\{g_1(n),g_2(n)\})$

טענה-2: $f_1(n)\cdot f_2(n)=O(g_1(n)\cdot g_2(n))$.

הוכחת טענות מהצורה: $f(n)\neq O(g(n))$ **:**

1. מניחים בשלילה שהטענה לא נכונה, כלומר ש $f(n)=O(g(n))$ מכאן קיימים c,n_0 כך ש- $f(n)\le c\cdot g(n)$ לכל $n\ge n_0$.

2. מוצאים n עבורו הטענה לא מתקיימת (שלא קיימים עבורו קבועים כאלה).

משוואות רקורסיביות

שיטות לפיתרון רקורסיות:

שיטת ההצבה:

1. מנחשים צורת פיתרון.

2. משתמשים באינדוקציה כדי למצוא קבועים מתאימים ולהוכיח שהפיתרון נכון.

שיטת האיטרציות:

1. אין צורך לנחש פיתרון.

2. פותחים את הביטוי הרקורסיבי כסכום של איברים שתלויים ב- n ובתנאי ההתחלה בלבד.

שיטת המאסטר:

$a\ge 1, b>1$ קובעים. $f(n)$ פונקציה. $T(n)$ מוגדרת על שלמים אי-שלילייפ ע"י הרקורסיה:

$T(n)=a\cdot T\left(\frac{n}{b}\right)+f(n)$ ניתן לחסום את $T(n)$ אסימפטוטית באופן הבא:

1. **אם** $f(n)=O\left(n^{\log_b a-\varepsilon}\right)$ כאשר $\varepsilon>0$ כלשהו,

   אזי $T(n)=\Theta\left(n^{\log_b a}\right)$

2. **אם** $f(n)=\Theta\left(n^{\log_b a}\right)$

אזי $T(n)=\Theta\left(n^{\log_b a}\cdot\log n\right)$

3. **אם** $f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)$ כאשר $\varepsilon>0$ כלשהו,

וגם $a\cdot f\left(\frac{n}{b}\right)\le c\cdot f(n)$ עבור קבוע $c>1$ והחל מ-n מספיק גדול,

אזי $T(n)=\Theta\left(f(n)\right)$.

<u>הערה:</u> המשמעות של $\frac{n}{b}$ יכולה להיות $\left\lceil\frac{n}{b}\right\rceil$ או $\left\lfloor\frac{n}{b}\right\rfloor$

סיבוכיות ממוצעת:

1. **<u>ממוצע הסתברותי:</u>** ממוצע על תוצאות ההגרלה של שהאלגוריתם מבצע.

למשל, הגובה הצפוי של רשימת דילוגים אקראית הוא $O(\log n)$ בממוצע הסתברותי.

2. **<u>ממוצע על הקלט:</u>** ממוצע על התפלגות הערכים בקלט.

למשל, הגובה הצפוי של חיפוש בעץ בינארי לא מאוזן הוא $O(\log n)$ בממוצע על הקלט.

וגם: זמן צפוי של גישה לטבלת ערבול הוא $O(1)$ בממוצע על הקלט.

3. **<u>סיבוכיות משוערכת:</u>** תהי F קבוצה של פעולות. נאמר שהקבוצה F רצה בסיבוכיות זמן משוערכת של $O\left(g(n)\right)$, אם **כל** רצף באורך m של פעולות מהקבוצה F, לוקח זמן כולל של $O\left(m\cdot g(n)\right)$.

עצים

הגדרות:

צומת פנימי: צומת שאינו עלה.

עומק של צומת: מספר הקשתות משורש העץ אל הצומת. (עומק שורש=0).

גובה של צומת: מספר הקשתות ממנו לצאצא הרחוק ביותר (עלה).

גובה עץ- גובה השורש. גובה עלה=אפס.

עץ בינארי מלא: עץ שבו לכל צומת פנימי 2 בנים.

עץ בינארי שלם: עץ בינארי מלא שבו כל העלים באותו עומק.

עץ בינארי כמעט שלם: בץ שינארי שלם שהוצאו ממנו עלים ("מצד ימין").

בעץ בינארי שלם בעל n צמתים, L עלים וגובה h:

1. מספר העלים בעומק i: $\boxed{n_i = 2^i}$
2. מספר העלים: $\boxed{L = n_h = 2^h}$
3. מספר הצמתים: $\boxed{n = \sum_{i=0}^{h} n_i = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1}$
4. הגובה: $\boxed{h = \log_2(n+1) - 1}$
5. מספר צמתים פנימיים: $\boxed{n - L = 2^h - 1 = L - 1}$

סיורים בעצים בינאריים: (שמות הסיורים מתארים את השלב בו מטפלים בשורש)

סיור Preorder: טיפול ב**שורש**; Preorder בתת העץ ה**שמאלי**; Preorder בתת העץ ה**ימני**.

סיור Inorder: Inorder בתת העץ ה**שמאלי**; טיפול ב**שורש**; Inorder בתת העץ ה**ימני**.

סיור Postorder: Postorder בתת העץ ה**שמאלי**; postorder בתת העץ ה**ימני**; טיפול ב**שורש**.

סיבוכיות: זמן: $O(n)$ מקום: $O(h)$ - גובה מחסנית הרקורסיה.

דוגמאות והערות:

- ביטוי אריתמטי: inorder- מספרים בעלים ופעולות בצמתים הפנימיים.

- שחזור עץ בהינתן סיור Preorder וסיור Inorder: שורש העץ הוא האיבר הראשון ב-Preorder. נקבע אותו כ-pivot בסיור Inorder, ונפעיל שוב את האלגוריתם על אלה משמאלו-תת-עץ שמאלי ואלה מימינו- תת עץ ימני.

- חישוב מספר הצמתים בעץ: Postorder- קודם מטפילם בבנים ואז סוכמים אותם ורושמים בשורש.

- מציאת המסלול הכבד ביותר בעץ: Postorder: לאחר טיפול בבנים, לוקחים את המקסימאלי מביניהם ומוסיפים לו את הערך השורש.

עץ חיפוש בינארי:

בהינתן צומת עם מפתח X:

- כל הצמתים ב**תת העץ הימני** הם בעלי מפתחות **גדולים** יותר.

- הצמתים ב**תת העץ השמאלי**, הם בעלי מפתחות **קטנים** יותר.

עצים מאוזנים:

הגדרה: עצים המקיימים: $h(T)=O(\log n)$.

עצי-AVL:

$h_L(v)$ - גובה תת-העץ השמאלי של v.

$h_R(v)$ - גובה תת-העץ הימני של v.

$BF(v)$ - ההפרש בין גבהי תתי העצים של v

בעץ AVL: $|BF(v)|=|h_L(v)-h_R(v)|\le 1$.

עץ פיבונאצי (ההפך מעץ שלם)- עץ עם מספר מינימאלי של NULL-ים.

N_h - מספר מינימאלי של NULL-ים בעץ AVL בגובה h.

טענות:

1. נגדיר: F_H - עץ פיבנאצי. לכל h, לעץ F_h גובה h.

2. $|F_h|=1+|F_{h-1}|+|F_{h-2}|\ge|F_{h-1}|$

3. $N_h = N_{h-1} + N_{h-2}$

4. יהי T עץ AVL בעל גובה h. אזי $|T| \geq |F_h|$

5. יהי T עץ AVL בן n צמתים וגובה h, אזי $\boxed{h = O(\log n)}$ ($h = 2\log(n+1)$).

וגם: $\boxed{h \approx \frac{\log N_n}{\log \Phi} = 1.44 \log n}$

6. לעץ פיבונאצי F_i יש $|F_i| = n_{i+3} - 1$ צמתים כאשר n_i הוא מספר פיבונאצי ה-i.

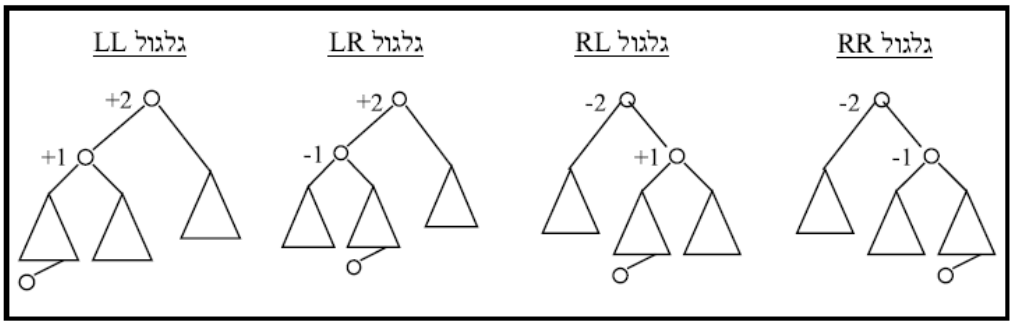
<u>גלגולים:</u>

- הצמתים היחידים שאולי הופר בהם איזון הם הצמתים לאורך מסלול הכנסה/הוצאה.

- אם עבור צומת v במסלול הנ"ל גובה העץ ששורשו v לא השתנה אזי גורמי האיזון בצמתים שמעליו במסלול לא השתנו.

- אם גורם האיזון הופך ל-2 או ל -2, אזי יש לבצע גלגול כדי שהעץ יחזור להיות עץ AVL.

- גורם האיזון לא יכול להיות גדול מ-2 בערכו המוחלט כי בכל הכנסה/הוצאה הוא משתנה ב-1 לכל היותר.



בשורש v	בבן השמאלי v_L	בבן הימני v_R	הגלגול המתאים
BF(v)=2	$BF(v_L) \geq 0$		LL
BF(v)=2	$BF(v_L) = -1$		LR
BF(v)= -2		$BF(v_R) \leq 0$	RR
BF(v)= -2		$BF(v_R) = 1$	RL

<u>לאחר הכנסה:</u> בארבעת הגלגולים, גובה תת-העץ המופר לאחר הגלגול זהה לגובה תת-העץ ההתחלתי (בטרם ההוספה). לכן, לאחר גלגול אחד, העץ יהיה תקין.

<u>לאחר הוצאה:</u> ברוב המקרים, לאחר הגלגול גובה תת-העץ המופר ישתנה. לכן, לאחר גלגול, יש להמשיך למעלה במסלול החיפוש ולבדוק האם צמתים אחרים הופרו.

<u>דוגמא:</u> כאשר מוציאים עלה ראשון בעץ פיבונאצי (העלה שהכי קרוב לשורש), גלגול בכל צומת עד לשורש.

עצי +B:

תכונות:

1. כל הערכים נמצאים בעלים וכל העלים באותה רמה.

2. לכל צומת פנימי פרט אולי, לשורש, יש c בנים כאשר $\left\lceil \frac{m}{2} \right\rceil \le c \le m$

מספר הבנים של השורש מקיים: $2 \le c \le m$.

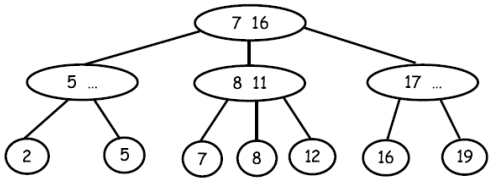
3. לצומת בעל c בנים יש $c-1$ מפתחות: $k_1, k_2, \ldots, k_{c-1}$ כך שמתקיים:

- כל הערכים ב**תת העץ השמאלי ביותר** של הצומת **קטנים** מ- k_1.

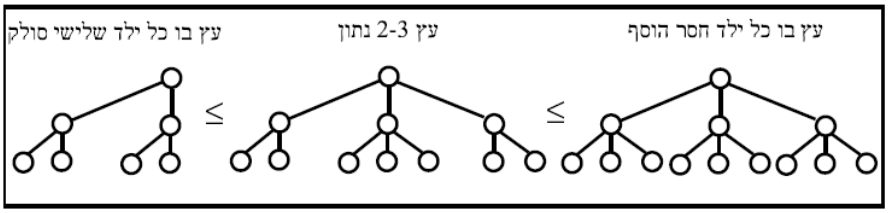
- כל הערכים הנמצאים **בתת העץ שבין המפתחות** k_{i-1}, k_i נמצאים בתחום $[k_{i-1}, k_i)$

- כל הערכים הנמצאים ב**תת העץ הימני ביותר** של הצומת **גדולים או שווים** ל- k_{c-1}.

עץ 2-3: עץ בו כל העלים נמצאים באותה רמה ולכל צומת פנימי 2 או 3 בנים.



משפט: מספר העלים L בעץ 2-3 מקיים $2^h \le L \le 3^h$ כאשר h הוא גובה העץ.



מסקנה: הגובה מקיים: $\log_3 L \le h \le \log_2 L$ ולכן: $h = \Theta(\log L)$.

תכונות תהליך הוספת צומת:

- שינויים מתבצעים רק על המסלול מהשורש לעלה שהוסף.

- בזמן פיצול של צומת, הצמתים החדשים הם בעומק שווה לצומת שפוצל.

- בפיצול השורש נוצר צומת חדש שמגדיל ב-1 את העומק של כל הצמתים.

ניתוח פעולת ההוצאה:

- העומק של שום צומת אינו משתנה בזמן ההוצאה מלבד בסוף התהליך,

- במידה ומתבטל השורש, אז העומק של כל הצמתים קטן ב-1.לפיכך כל העלים נותרים בעומק שווה.

- לכל צומת פנימי נותרים 2-3 בנים ונשמר יחס הסדר בין האינדקסים כנדרש.

עצי דרגות:

אינדקס-Rank: אינדקס של איבר בקבוצת איברים הוא המקום שלו בסדרה הממוינת.

עץ דרגות- Rank Tree: עץ חיפוש בינארי מאוזן, בו כל צומת מחזיק, בנוסף למידע הקיים, שדה נוסף w השומר את מספר הצמתים בתת-העץ השמאלי של הצומת (או בתת-העץ כולו ששורשו הצומת).

select(k): מציאת האיבר בעץ בעל דרגה-k.

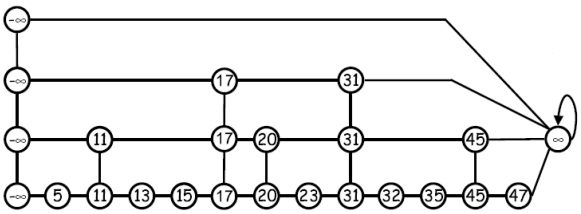
rank(x): מציאת הדרגה של איבר בעץ.

הערות:

- כל הפעולות הינן ב- $O(\log n)$.

- ניתן לממש rank tree על בסיס עצי 3-2. ערכי $w(v)$ יתייחסו רק לעלים.

- ניתן להשתמש ב-Rank Tree לצורך חישובים אחרים ולהגדיר מידע אחר שהוא תכונה של תת-העץ.



רשימת דילוגים

הגדרת המבנה:

- רשימת דילוגים מורכבת מכמה רמות. כל רמה היא רשימה מקושרת.

- כל איברי המבנה נמצאים ברמה התחתונה.

- כל רמה אחרת היא תת-קבוצה של הרמה שמתחתיה.

- ברמה עליונה יש $O(1)$ צמתים.

רשימת דילוגים רנדומאלית:

()toss- לאחר הוספת צומת, בכל רמה האלגוריתם מטיל מטבע כדי לקבוע אם יוצר העתק נוסף מעליה. מספר הרמות הממוצע למפתח שווה למספר הפעמים הממוצע שיש להטיל מטבע עד לקבלת- '1'. כל צומת יופיע בממוצע- $\frac{1}{1-p}$ פעמים (p- הסתברות להוספת צומת).

מסקנה: מספר הצמתים הממוצע הוא 2n כאשר n הוא מספר המפתחות במבנה (כל מפתח מופיע בממוצע פעמיים).

משפט: האורך הממוצע L של מסלול חיפוש ברשימת דילוגים עם n מפתחות כאשר למטבע הסתברות-p לצאת '0' (להמשיך) מקיים: $\boxed{L \le {}^{1}\!/_{p} \log_{1/p} n + {}^{1}\!/_{p}}$.

הערה: לפרמטר p אין השפעה גדולה על האורך הממוצע של המסלול.

מסקנה מהמשפט: חיפוש, הכנסה והוצאה מרשימת דילוגים נעשים בזמן ממוצע $O(\log n)$.

יתרונות:

1. מימוש פשוט יחסית (אין גלגולים, איחוד ו/או פיצול צמתים...).

2. אלגוריתם רנדומי: לא תלוי בהתפלגות הקלט או ביריב.

רשימת דילוגים דטרמיניסטית:

הרעיון: שומרים שמספר הצמתים של רמה i בין כל שני צמתים של רמה i-1 יהיה 1 או-2. אם מספר הצמתים הוא 3, מוסיפים צורמת ברמה ה-(i-1). אם ברמה העליונה נוסף עוד מפתח, אז מוסיפים רמה חדשה.

הערות ומסקנות:

- בניגוד לרשימת דילוגים רנדומאלית, המפתח שהוכנס אינו בהכרח המפתח שמשוכפל ברמה מעל.

- הגובה מקיים: $\boxed{\log_3 n \le h \le \log_2 n}$.

- בזמן החיפוש מבצעים לכל היותר 3 צעדים ימינה (כי זה גודל האינטרוואל שבו מחפשים) ולכן סיבוכיות הזמן של פעולת החיפוש היא $O(\log n)$ במקרה הגרוע.

טבלאות ערבול- Hash tables

צורך: הפעולות הבסיסיות במילון (חיפוש, הכנסה והוצאה) מתבצעות ב- $O(\log n)$ כל אחת. אבל אם ידוע הטווח של המפתחות ניתן לממש במערך שבו כל גישה היא- $O(1)$.

אבל: לעיתים גודל התחום של ערכי המפתחות גדול בהרבה ממספר המפתחות בהם משתמשים.

דוגמא: מספרי זהות. ולכן ניצור מערך קטן בהרבה מן הטווח ונחשב אינדקס- $h(k)$ מתוך המפתח k באמצעות פונקצית ערבול.

פונקצית ערבול: $h: U \to \{0,\ldots,m-1\}$ בהינתן מפתח בתחום U , הפונקציה מחשבת אינדקס בטווח המתאים, כאשר- m גדול המערך.

פתרונות להתנגשויות: (כאשר $x \neq y$ אבל $h(x)=h(y)$).

Chain-hashing: כל תא בטבלת הערבול מצביע לרשימה מקושרת. אם איבר $x \in U$ הוכנס למבנה, אזי הוא נמצא ברשימה המקושרת שמוצבעת ע"י: $T[h(x)]$. מימוש יעיל: עצים במקום רשימות.

הנחת הפיזור האחיד: h מפזרת את המפתחות באופן אחיד.

פקטור העומס: $\alpha = \frac{n}{m}$ כאשר n - מספר המפתחות, m - גודל הטבלה.

תחת הנחת הפיזור האחיד, האורך הממוצע של רשימה הוא α מכיוון שהאיברים מתחלקים בצורה שווה בין הרשימות השונות.

משפט: בשיטת השרשראות ותחת הנחת הפיזור האחיד, זמן חיפוש כושל ממוצע הוא: $\Theta(1+\alpha)$.

משפט: בשיטת השרשראות ותחת הנחת הפיזור האחיד, זמן חיפוש מוצלח ממוצע הוא $\Theta\left(1+\alpha/2\right)$.

אם נבחר $m = O(n)$ נקבל: $\alpha = \frac{n}{m} = O(1)$ ← גורם העומס הוא קבוע. כל פעולה- $O(1)$ בממוצע.

Rehashing with Open Addressing: האיברים נמצאים בתוך טבלת הערבול.

הוצאת איבר: מסמנים את מקום האיבר שהוצא בסימן- delete, כדי לא לנתק את שרשרת החיפוש.

Rehash: עוברים על הטבלה ושומרים את איבריה בצד. מנקים את הטבלה וממלאים אותה מחדש. סיבוכיות הזמן הכוללת של פעולה זו: $O(m)$ בממוצע. אם נבצע את פעולת ה-Rehash פעם ב-m פעולות delete, אז סיבוכיות הזמן המשוערכת של כל הפעולות במבנה תהיה $O(1)$ בממוצע.

סריקה ליניארית: $r(x)=1$. אורך חיפוש ממוצע: $\frac{1}{1-\alpha}=\frac{m}{m-n}$.

יתרונות וחסרונות: המימוש פשוט, אבל סריקה ליניארית נוטה למלא בלוקים מכיוון שכאשר קיים בלוק, האיברים הבאים מצטרפים אליו בסבירות גבוהה יותר מאשר הסבירות למלא תא חדש בודד.

כאשר השימוש דורש הוצאות, אורך החיפוש תלוי גם באיברים שכבר הוצאו ולא רק באיברים שכרגע במבנה. **כאשר יש צורך בהוצאות, עדיף להשתמש בשיטת הרשימות המקושרות.**

ערבול נשנה- Rehashing: נגדיר סדרה אינסופית של פונקציות:
$h_k(x)=\left(h(x)+k\cdot r(x)\right)\bmod m$ כאשר $h(x)$ - פונקצית הערבול ו-$r(x)$ - פונקצית הצעד.
הרעיון הכללי: אם התא $h_k(x)$ תפוס, נסה את התא $h_{k+1}(x)$. סריקה ליניארית היא מקרה פרטי.

ערבול כפול: שימוש בשתי פונקציות בלבד- h,d הנבחרות באופן בלתי תלוי כאשר:
$h_1(x)=h(x)+i\cdot d(x)$. גודל הטבלה ו-$d(x)$ צריכים להיות מספרים זרים כך ש-
$h_0(x),\ldots,h_{m-1}(x)$ תכסה את כל האינדקסים האפשריים. לכן **נוח לבחור את** m **להיות ראשוני.**

משפט: בהנחת הפיזור האחיד בשיטת ערבול Rehashing מתקיים:

- זמן ממוצע של חיפוש כושל קטן מ-$\frac{1}{1-\alpha}$

- זמן ממוצע של חיפוש מוצלח קטן מ-$\frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)+\frac{1}{\alpha}$

משפט: בהנחת הפיזור האחיד בשיטת ערבול בסריקה ליניארית מתקיים:

- זמן ממוצע של חיפוש כושל ושל חיפוש מוצלח קטן מ-$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$

מסקנה: אפשר להשתמש בסריקה ליניארית כל עוד הטבלה לא מלאה מידי (80%).

ערבול אוניברסאלי:

הבעיה: לכל בחירה של פונקצית ערבול קיימת סדרה גרועה של מפתחות כך שתיווצר רשימה באורך מקסימאלי.

הפיתרון: לבחור באקראי, בזמן יצירת טבלת ערבול, פונקצית ערבול מתוך קבוצה שהוגדרה מראש.

קבוצה אוניברסאלית: תהי H קבוצת פונקציות ערבול מתחום U לקבוצה $\{0,\ldots,m-1\}$. הקבוצה H נקראת אוניברסאלית אם לכל זוג מפתחות שונים $x,y\in U$ מספר הפונקציות עבורן $h(x)=h(y)$ הוא: $\frac{|H|}{m}$. בשלב האתחול של טבלת הערבול, נגריל באופן אקראי פונקציה h מתוך קבוצה H ונשתמש בה לאורך כל זמן פעולת המבנה.

<u>אבחנה:</u> לכל זוג מפתחות x,y ההסתברות p שבבחירה אקראית של פונקצית ערבול מתוך H, תהיה התנגשות בין x ו-y היא: $p=\frac{|H|/m}{|H|}=\frac{1}{m}$

<u>משפט:</u> תהי H קבוצה אוניברסאלית של פונקציות ערבול לתוך טבלה T בגודל m. אם h נבחרה באקראי מתוך H, ונשתמש בה לערבול n מפתחות כלשהם, אזי לכל מפתח, המספר הצפוי של התנגשויות בשיטת הרשימות המקושרות שווה ל-$\frac{n-1}{m}=\frac{\alpha-1}{m}$

<u>מסקנה:</u> מספר ההתנגשויות הצפוי לכל מפתח קטן מפקטור העומס.

לכן, אם $n=O(m)$ אז סיבוכיות הזמן של הפעולות הינה $O(1)$ בממוצע הסתברותי.

אם משתמשים ב-Rehash- זה הופך להיות $O(1)$ בממוצע הסתברותי משוערך.

הערה: ניתן להניח שלכל גודל קבוצה קיימת פונקצית ערבול וקבוצת פונקציות אוניברסאלית.

<u>מגבלות הערבול:</u>

צריך לדעת מראש סדר-גודל למספר האיברים. כאשר הטבלה **מתמלאת** (ב-Open Addressing) או כאשר פקטור העומס גדול מידי נגדיל את הטבלה פי-2. מקובל להקטין את הטבלה בחצי כאשר היא **25% מלאה**.

קבוצות זרות - Union-Find

הגדרת המבנה: נתונים n איברים: $1,2,\ldots,n$. איברים אלה מחולקים לאורך כל ריצת המבנה, לקבוצות זרות. בתחילת ריצת המבנה, כל איבר הוא קבוצה בפני עצמה (singletone).

Makeset(i)- מחזיר קבוצה חדשה בעלת איבר בודד i.

Find(i)- מחזיר את הקבוצה לה שייך איבר i.

Union(p,q)- מאחד את הקבוצות p ו-q, כלומר מחזיר קבוצה חדשה המכילה את איחוד האיברים.

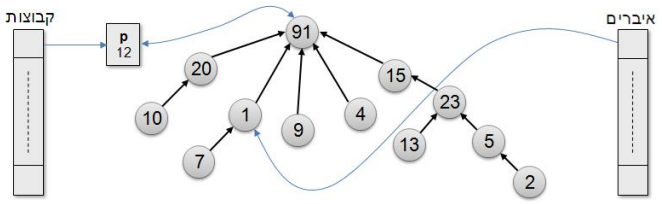
מימוש נאיבי- מערך: כל קבוצה היא מספר, וכל-פעם מגדילים את ה-counter הכללי כדי ליצור קבוצה חדשה באיחוד או באתחול. ב-Union(p,q): עוברים על המערך ובכל מקום שבו רשום p או q רושמים: counter+1.

סיבוכיות: Find ,Makeset- $O(1)$, Union- $O(n)$.

מימוש נאיבי- רשימות מעגליות: כל קבוצה תהיה רשימה מעגלית ונחזיק רשימה מקושרת של מצביעים אל הרשימות. ניתן לאחד רשימות מעגליות ב-$O(1)$, אבל החיפוש ייקח הרבה זמן.

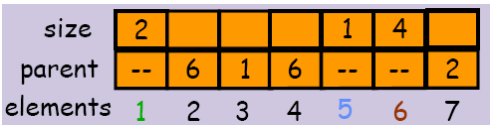
סיבוכיות: Find ,Makeset- $O(n)$, Union- $O(1)$.

מימוש בעזרת עצים הפוכים: לכל קבוצה ניצור עץ הפוך (עץ בו כל הבנים מצביעים לאבותיהם) מכל איברי הקבוצה. שורש כל עץ יצביע על רשומה שתכיל את שם הקבוצה ומספר איבריה. בנוסף נחזיק שני מערכים:



1. מערך הממפה בין איבר לצומת המתאים לו בעץ ההפוך.

2. מערך הממפה בין מספר הקבוצה לרשומה המתאימה לקבוצה. הרשומה מחזיקה מצביע לשורש העץ.



מימוש עצים הפוכים בעזרת מערכים: האיבר בשורש משמש מציין לקבוצה. וה-size מעודכן רק עבור השורשים.

טענה-1: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר $\log_2 n$ פעמים כאשר n הוא מספר האיברים במבנה.

שיפור-1: איחוד לפי גודל: אם נבצע איחוד לפי גודל, כל פעולת find תיקח $O(\log n)$ ב-w.c.

שיפור-2: כיווץ מסלולים: לאחר מציאת האיבר ה-i, נסרוק שוב את המסלול בין i לשורש ונחבר כל צומת במסלול לשורש.

* שימוש באיחוד לפי גודל ובכיווץ מסלולים ביחד, ייתן זמן משוערך לשתי הפעולות- $O(\log^* n)$.

הערה: $\log^* n$ - מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה ל-1.

תור עדיפויות/ ערימה

הגדרת מבנה:

MakeHeap(Q): בניית ערימה מתוך n איברי קלט.

Insert(x,Q): הכנסת איבר x לערימה.

FindMin(Q): מציאת המינימום בערימה.

DelMin(Q): הוצאת האיבר המינימאלי מהערימה.

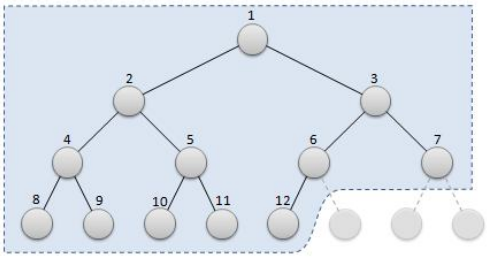
שימוש נפוץ: תור עדיפויות- המשימה בעלת העדיפות הגבוהה ביותר תמיד נמצאת בראש הערימה.

כלל הערימה (ערימת מינימום): המפתח של הורה קטן או שווה למפתחות ילדיו. הסדר בין הילדים אינו מוגבל.

נמספר את הצמתים לפי סדר השורות, משמאל לימין, ב"אינדקסים".

מימוש ע"י עץ חיפוש מאוזן: יוצרים ערימה ב- $O(n\log n)$.

מימוש ע"י עץ בינארי כמעט שלם:



תכונות עץ כמעט שלם:

1. עץ כמעט שלם בעל n צמתים וגובה h מקיים: $2^h \le n \le 2^{h+1}-1$ ומכאן $h=O(\log n)$.

2. מספר העלים הוא $\lceil n/2 \rceil$ מספר הצמתים הפנימיים הוא $\lfloor n/2 \rfloor$

3. אינדקס השורש הוא תמיד 1. העלה האחרון בעל אינדקס n. צומת חדש שיוכנס יקבל את האינדקס $n+1$. $\lfloor n/2 \rfloor$- הלא עלה האחרון. $\lfloor n/2 \rfloor+1$- העלה הראשון.

4. עבור צומת i:

- אינדקס בנו השמאלי הוא 2i, אינדקס בנו הימני הוא 2i+1.

- מכאן, צומת הוא בן שמאלי אם"ם יש לו אינדקס זוגי.

- אינדקס אביו הוא $\lfloor i/2 \rfloor$

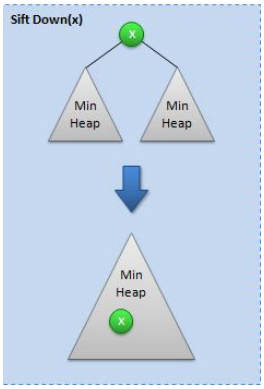
5. צומת i הוא עלה אם"ם $2i > n$.

מימוש במערך (כאשר ידוע חסם על מספר האיברים):

* האינדקסים של הצמתים בעץ ישמשו כאינדקסים של כניסות המערך.

* ניתן להגיע מאב לבן ולהפך ב- $O(1)$.

* נשמור מקום פנוי בסוף המערך לאיברים חדשים.

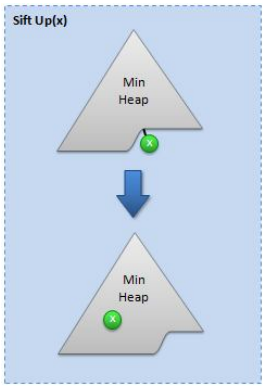


* נשמור גם את n - מספר התאים המנוצלים במערך.

<u>פרוצדורת עזר- sift down:</u> בזמן הוצאה והכנסה יש לשמור על תכונת הערימה.לשם כך הפרוצדורה הזו שהופכת עץ בינארי כמעט שלם שעבורו מופרת תכונת הערימה רק בשורש בחזרה לערימה. **<u>אלגוריתם:</u>** כל עוד x גדול מאחד מבניו ו- x אינו עלה, החלף בין x לבין בנו המינימאלי.

<u>makeHeap:</u> זמן הריצה שלו נתון ע"י (סכום הגבהים של כל הצמתים)O שכן לכל צומת מבצעים $O\left(h\left(v\right)\right)$ תיקונים כאשר $h\left(v\right)$ הוא גובה הצומת v. ניתוח פשוט נותן חסם של $O\left(n\log n\right)$, ניתוח מדויק יותר ייקח בחשבון שלרוב הצמתים גובה קטן. והחסם הוא: $O\left(n\right)$.

<u>משפט:</u> עבור עץ בינארי שלם בגובה h הכולל $n=2^{h+1}-1$ צמתים, סכום הגבהים $=O\left(n\right)$.



<u>sift up:</u> ב-Insert נוסיף את האיבר החדש x כעלה חדש (באינדקס הפנוי הראשון במערך). כדי לתקן את הערימה, נעזר ב-sift up- **<u>אלגוריתם:</u>** כל עוד x קטן מאביו, החלף בין x לאביו.

Del Min/Max: נחליף בין השורש לעלה האחרון ונמחק את העלה. נבצע sift down על השורש החדש לקבלת ערימה תקינה.

<u>מימוש ע"י עץ כמעט שלם בעץ בינארי (כשלא ידוע חסם על מספר האיברים):</u>

נחזיק מצביע לעלה האחרון בעץ.

<u>MakeHeap:</u> נעבור על הצמתים הפנימיים לפי סדר preorder.

<u>Insert:</u> באמצעות המצביע לעלה האחרון בעץ, נמצא את המקום הבא להכנסה:

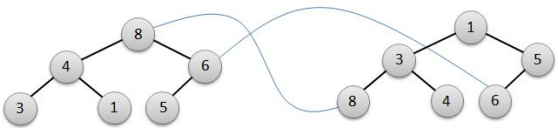
- ניגש למצביע לעלה האחרון.

- נעלה בעץ עד שנגיע לצומת שממנו ניתן לרדת ימינה.

- בהנחה וקיים צומת כזה, נרד ימינה ונמשיך משם שמאלה עד הסוף.

- אחרת (סימן שהגענו לשורש העץ), נלך שמאלה עד הסוף ונוסיף שם את הצומת החדש.

<u>DelMin:</u> נשתמש במצביע לעלה האחרון בשביל להחליף בינו ובין השורש.



<u>שתי ערימות- מינימום ומקסימום:</u>

כל איבר יחזיק מצביע לעותק שלו בערימה השנייה.

<u>מחיקת שורש ערימה אחת- עלה בערימה השניה:</u> העותק של **שורש** ערימה אחת הוא **עלה** בערימה השנייה. תהליך המחיקה: מחליפים בין העלה לעלה האחרון. מוחקים את העלה האחרון. מבצעים sift_up מהמקום המקורי של העלה שנמחק.

מיון – sorting

Quick Sort (מיון ע"י בחירת איבר ציר):

העלמת רקורסיית זנב: החלפת הקריאה הרקורסיבית השנייה באיטרציה.

הערה: בקומפיילרים מודרניים מימוש של העלמת רקורסיית זנב נעשה אוטומאטית.

העלמת רקורסיה לפי גודלה: החלפת הקריאה הרקורסיבית הגדולה באיטרציה. כלומר, קריאה רקורסיבית רק על חלק המערך הקטן. במימוש זה, עומק הרקורסיה חסום ע"י $\log_2 n$ כיוון שבכל קריאה רקורסיבית המערך קטן בלפחות חצי.

מציאת האיבר ה-i בגודלו (ללא מיון):

רנדומאלי:

מקרה פרטי: מציאת החציון- חיפוש האיבר בגודל: $\lfloor (n+1)/2 \rfloor$.

1. בחר איבר ציר.

2. חלק את המערך לשני חלקים. האיברים הקטנים מ-pivot יאוחסנו בחלק השמאלי והגדולים בימני.

3. אם מספר האיברים בחלק השמאלי (נסמנו-q) קטן מ-i, מוצאים רקורסיבית בחלק השמאלי, אחרת- מוצאים בימני את האיבר ה-i-q בגודלו.

מקרה אופטימאלי: איבר הציר הוא חציון- $O(n)$.

מקרה גרוע: בכל שלב המערך קטן באחד בלבד- $O(n^2)$.

מקרה ממוצע: ניתן "לתקן" כך שאם יבחר האיבר ה-'1', נתקן אותו לחלוקה של $1:n-1$/.

לאחר התיקון: $T(n) \le c \cdot n$ עבור c המקיים: $T(1) \le c$.

משפט: אם $T(1)=c \quad \alpha+\beta<1, \quad T(n)=T(\alpha n)+T(\beta n)+cn,$ אזי: $T(n)=O(n)$.
בדרך זו ניתן להוכיח כי אלגוריתם הוא ליניארי.

דטרמיניסטי:

אם נבטיח שבכל קריאה רקורסיבית גודל המערך קטן "בצורה משמעותית", אז זמן הריצה במקרה הגרוע יהיה $O(n)$. נרצה שבכל שלב האלגוריתם יבצע חלוקה לשתי קבוצות כך שהגדולה עדיין קטנה משמעותית מהקבוצה המקורית.

האלגוריתם **Select**(A, first, last, i):

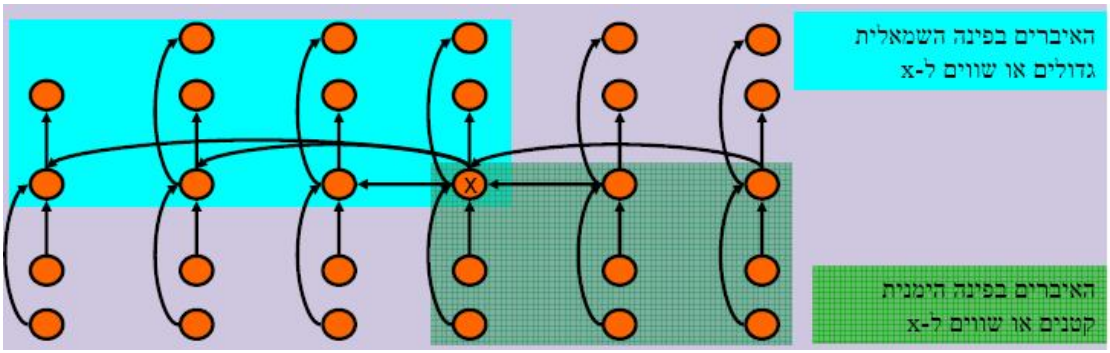
1. חלק את מערך הקלט A לחמישיות.

2. מצאת את החציון של כל חמישייה. הכנס את החציונים למערך B (שגודלו בערך $1/5$ מ-A).

3. הפעל את Select רקורסיבית על המערך B למציאת חציון של חציונים (נסמנו-x).

4. x יהיה איבר הציר הנבחר ולפיו נחלק את המערך A ונמשיך במיון רקורסיבית.

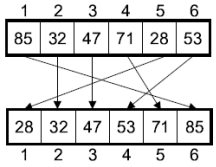
מספר האיברים במערך שקטנים מחציון החציונים וגם מספר האיברים שגדולים ממנו $\geq 3n/10$ לכן:



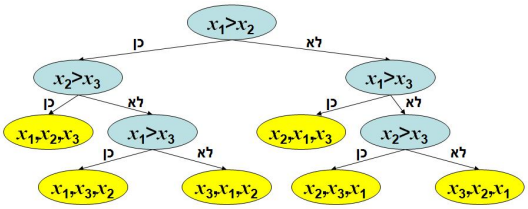
אם נשתמש בו כאיבר ציר נקבל חלוקה: $7n/10,\quad 3n/10$.

בסה"כ: $T(n)=T(7n/10)+T(n/5)+cn$ ולפי המשפט: $\frac{7}{10}+\frac{1}{5}<1\rightarrow T(n)=O(n)$

חסם תחתון למיון מבוסס השוואות:



הסבר-1: כל סדרה בת n מספרים שונים מגדירה פרמוטציה אחרת על האינדקסים של מערך הקלט. כל פרמוטציה אפשרית כפלט כי האיבר ה-$A[i]$ יכול להגיע לכל מקום במערך הפלט לפי גודלו היחסי במערך הקלט. קיימות $n!$ פרמוטציות.



הסבר-2: אלגוריתם מיון מבוסס השוואות ניתן לתיאור ע"י עץ החלטות בינארי. סיבוכיות הזמן = גובה העץ.

משפט: גובה העץ $\geq$ (מספר העלים) $\log$. בהוכחה מקבלים:

$$\boxed{h>\log_2 n!\geq\frac{1}{2}n\log_2 n}$$

. כמו-כן מספר ההשוואות k מקיים:

$\frac{n!}{2^k}=1$ כי רוצים שקבוצת כל הפרמוטציות תגיע לגודל של-1, כדי שנמצא את המתאימה למיון.

משפט-חסם תחתון למיון מבוסס השוואות: כל אלגוריתם מיון הפעול באמצעות השוואות בלבד, דורש לפחות $\Omega(n\log n)$ השוואות (כאשר לא ידוע מידע נוסף על הקלט- זה מבוסס השוואות).

חסם תחתון על אלגוריתמים: באמצעות המשפט, ניתן למצוא חסמים תחתונים על אלגוריתמים אחרים ולהוכיח שהם לא קיימים- מניחים בשלילה שהם כן, ומראים שאפשר באמצעותם למיין n איברים בפחות מ-$\Omega(n\log n)$ בסתירה למשפט.

אלגוריתם מיון יציב: שומר על הסדר היחסי בין איברים בעלי ערכים זהים.

מיון Counting /Bucket Sort:

אם n המספרים לקוחים מהטווח $[1,k]$, ניתן למיין אותם בזמן $O(n+k)$.

הערה: עבור $k=O(n)$ נקבל מיון בזמן ליניארי.

שיטה-1: שימוש במערך בוליאני-A בגודל-k. מאפסים אותו, ולכל איבר-x מגדילים את $A[x]$ ב-1.
ואז עוברים על המערך פעם נוספת ומוציאים את האיברים לפי הסדר בצורה הבאה: תחילה עוברים על המערך ומעדכנים כל תא להיות האינדקס האחרון שבו מופיע האיבר במערך הממוין. יוצרים מערך פלט-B, ואז עוברים על איברי הקלט מהסוף להתחלה, ומוסיפים כל איבר לפי האינדקס הרשום, לאחר הוספה- מקטינים את האינדקס ב-1.

דוגמא: קלט

1	2	3	4	5	6	7	8
6	1	2	1	2	5	6	2

$n=8, k=6$

אינדקסים אחרונים-

1	2	3	4	5	6
2	5	5	5	6	8

שיטה-2: מערך של k תורים. בזמן האיסוף נשרשר את התורים.

במיון זה, תוצאת "השוואה" אחת שווה ל-$\log_2 k$ השוואות בינאריות.

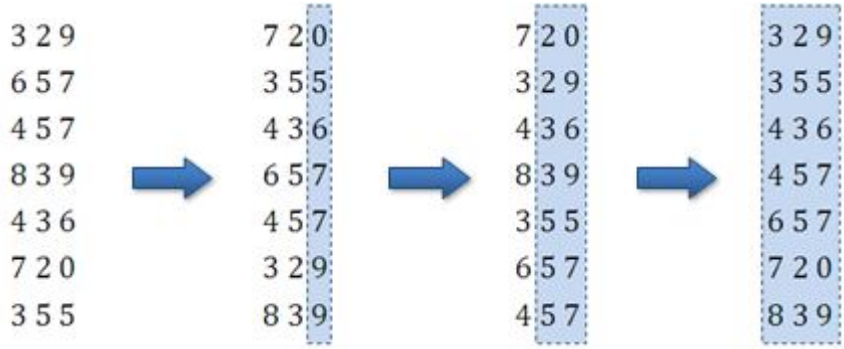
אבחנה: שיטת Bucket Sort אינה יעילה כאשר תחום המספרים רחב מידי ביחס למספרם $n \ll k$.

מיון Radix Sort:

אם נתונים n מספרים בעלי d ספרות המיוצגים בבסיס b, אז ניתן למיין אותם בסיבוכיות $O(d\cdot(n+b))$. עבור d,b קבועים זהו מיון בזמן ליניארי.

השיטה: נבצע d שלבים, כאשר בשלב ה-i נמיין את הספרה ה-i-ית מהסוף (LSD ממוינות קודם), בעזרת מיון יציב, למשל-Bucket Sort.

הערה: אם נמיין מה-MSB לא נקבל את המספרים ממוינים.



מחרוזות:

מחרוזת: רצף אותיות מעל א"ב $\sum$ כלשהו. כל מחרוזת נגמרת בתו מיוחד \$ (נניח כי $\$ \notin \sum$), המציין סוף מחרוזת. התו \$ קטן לקסיקוגרפית מכל אות בא"ב $\sum$.

דרוש מבנה נתונים למימוש הפעולות הבאות על אוסף מחרוזות מעל א"ב קבוע $\sum$:

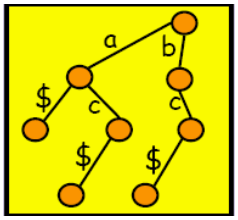
Insert(s): הכנסת מחרוזת חדשה s למבנה.

Delete(s): מחיקת מחרוזת s מהמבנה.

Find(s): האם המחרוזת s נמצאת במבנה?

FindMin(): מציאת המחרוזת המינימאלית (לפי סדר לקסיקוגרפי) במבנה.

מבנה Trie:



עץ בו לכל צומת פנימי לכל היותר $|\sum|+1$ ($\$\rightarrow$) בנים. את הקשתות לבנים ישמור כל צומת ע"י מערך בגודל זה, כל קשת מסומנת בתו המתאים לה. כל מחרוזת מיוצגת ע"י מסלול מהשורש לעלה.

דוגמא ל-Trie עבור המחרוזות: ac, a, bc ←

מימוש ע"י עץ חיפוש: עץ חיפוש מאוזן שבו כל צומת מחזיק מחרוזת. Find, Insert, Delete, יבצעו $O(\log n)$ פעולות השוואה, אך כל פעולת השוואה בין מחרוזות תיקח $O(m)$ לכן כל אחת מהפעולות הנ"ל תיקח $O(m\log n)$ זמן.

מימוש ע"י טבלת ערבול: נגדיר פונקצית ערבול על מחרוזות. במימוש זה Find, Insert, Delete, לוקחות $O(m)$ בממוצע, אבל FindMin לוקחת $O(nm)$ במקרה הגרוע.

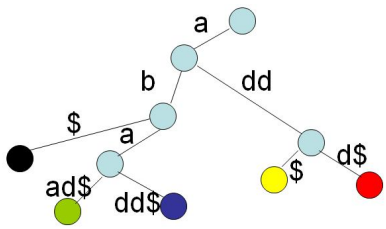
מיון מחרוזות באמצעות Trie:

טענה: סיור Preorder ב-Trie בו עוברים על קשתות כל צומת לפי סדר לקסיקוגרפי מגיע לעלה שמתאים למחרוזת s_1 לפני שהוא מגיע לעלה שמתאים למחרוזת s_2 אם"ם: $s_1 < s_2$ (לקסיקוגרפית).

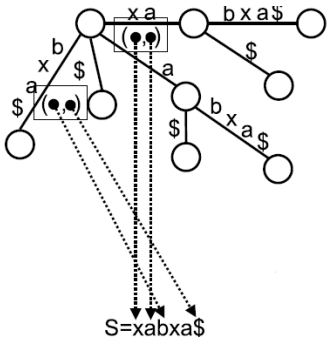
מיון מחרוזות שאורכן הכולל הוא- $m=\sum_{i=1}^{n}|s_i|$ יתבצע ע"י הכנסת כל המחרוזות ל-Trie בזמן $O(m)$ וסיור Preorder רקורסיבי, ובכל פעם שנגיע לעלה נדפיס את תוכן מחסנית הרקורסיה (בסדר הפוך).

דחיסת Trie: נסלק מהעץ צמתים בעלי בן אחד ע"י החלפת שרשרת קשתות בקשת בודדת שתסומן בתווית המקודדת את המחרוזת המתאימה. **חיסכון במקום:** מספר העלים הוא n, לכל צומת פנימי לפחות 2 בנים, לכן יש לכל היותר $n-1$ צמתי פנימיים. לכן סה"כ הצמתים קטן או שווה ל $2n-1$.

משפט: אם בעץ לכל צומת פנימי יש לפחות 2 בנים אזי מספר העלים > מספר הצמתים הפנימיים.



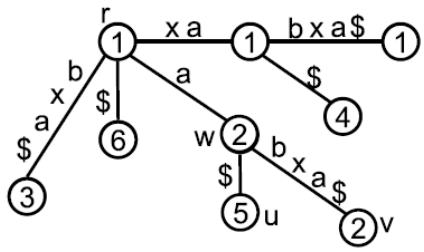
עץ סיומות – Suffix tree:



Trie שבו הוכנסו כל הסיומות של המחרוזת s עם תו סיום $.

דחיסה: נסלק מהעץ צמתים בעלי בן יחיד ונחליף שרשרת קשתות בקשת בודדת שתכיל את תת-המחרוזת המתאימה.

ייעול נוסף: אמנם מספר הצמתים והקשתות הוא $O(|s|)$ אך קיימות מחרוזות שבהן סך המקום הדרוש לשמירת המחרוזות בקשתות הינו $O(|s|^2)$. לכן נשמור העתק נפרד של המחרוזת s. כל קשת תחזיק שני מצביעים- לתחילת ולסוף תת-המחרוזת ב-s שקשת זו מייצגת.



מופע ראשון של כל מחרוזת בטקסט: ע"י סיור Preorder בעץ נוכל בזמן ליניארי $O(|s|)$ לחשב לכל צומת את המיקום הראשון של תת-המחרוזת המיוצגת ע"י המסלול מהשורש ועד אליו.

החישוב: חיסור מספר התווים (כולל $) המופיעים על המסלול לצומת מהאורך הכללי-$|s|$ ועוד 1. בצמתים הפנימיים ישמר המינימום של הבנים.

עץ סיומות מוכלל: Trie שבו הוכנסו כל הסיומות של קבוצת מחרוזות $\{s_1,\ldots,s_n\}$ עם תו סיום שונה $\$_i$ לכל מחרוזת s_i.

אלגוריתם ליניארי:נבנה עץ סיומות עבור המחרוזת (ביחיד!) $S_n\$_n\ldots S_2\$_2S_1\$_1$ ונקצץ את כל תתי העצים מתחת לקשתות שהתווית שלהן היא $\$_i$, כי מה שבא אחרי סימן $\$$ הוא כבר בעצם תחילת מחרוזת אחרת ולכן לא מהווה חלק מהסיומות המקוריות של המחרוזות. לכן ניתן להוריד אותן מהעץ.
סיבוכיות הזמן: $O(\sum_i |s_i|)$.

שימושים:

- מציאת תת-מחרוזת בתוך מחרוזת נתונה. תת-מחרוזת היא רישא של סיומת. נחפש את תת-המחרוזת בעץ הסיומות. אם לא הגענו ל-NULL, סימן שיש סיומת שמתחילה בתת המחרוזת שחיפשנו.

- דחיסת אינפורמציה. ע"י אלגוריתם זיו-למפל.

- מציאת המחרוזת המשותפת הארוכה ביותר של שתי מחרוזות נתונות, ע"י עץ סיומות מוכלל.

משפט: קיים אלגוריתם ("**קופסה שחורה**") לבניית עץ סיומות ממחרוזת באורך-$|s|$ בזמן- $O(|s|)$ אלגוריתם זה מחזיר עץ סיומות דחוס.

דחיסת אינפורמציה:

נעבור על המחרוזת הנתונה s משמאל לימין, בכל פעם שעוברים על תת-מחרוזת z שכבר ראינו, נחליף את z עם האינדקס והאורך של המופע השמאלי ביותר של z במחרוזת s.

מימוש יעיל של Ziv-Lempel: בנה עץ סיומות דחוס T עבור המחרוזת s בזמן $O(m)$. בכל איטרציה כדי לחשב את (s_i, L_i) - צעד על המסלול מהשורש של T לפי התווים במחרוזת $S[i..m]$ כל עוד $c_V < i$ (התווים שאנו מוצאים נמצאים לפני התו-i במחרוזת) כאשר החיפוש נעצר לאחר L_i תווים.

גרפים

הגדרה: גרף $G=(V,E)$ מורכב מקבוצת צמתים V וקבוצת קשתות E.

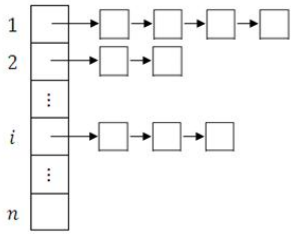
סימון: $|V|=n,\ |E|=m$

גרף ממושקל: גרף בו לכל קשת יש משקל (מספר כלשהו).

גרף קשיר: גרף לא מכוון שקיים בו מסלול בין כל שני צמתים.

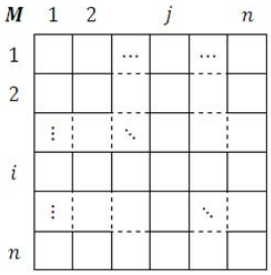
גרף קשיר היטב: גרף שבו קיים מסלול מכוון בין כל זוג צמתים.

ייצוג ע"י רשימת סמיכויות:



- כל תא i מחזיק מצביע לרשימה מקושרת שאיבריה הם שכני הצומת i בגרף.
- אם הגרף ממושקל, נשמור את המשקל של כל קשת ברשומות המתאימות לה.
- בגרף לא מכוון כל קשת מופיעה פעמיים (פרט לחוגים עצמיים).

ייצוג ע"י מטריצת סמיכויות:



- כל תא $M[i,j]$ מייצג את הקשת (i,j): $M[i,j]=\begin{cases}1 & (i,j)\in E\\ 0 & (i,j)\notin E\end{cases}$

- אם הגרף ממושקל (משקלים חיוביים ממש): $M[i,j]=\begin{cases}w(i,j) & (i,j)\in E\\ 0 & (i,j)\notin E\end{cases}$

- עבור גרף לא מכוון, המטריצה היא סימטרית.
- האלכסון מציין חוגים עצמיים.

ייצוג משולב: מכל תא "דלוק" במטריצה נשמור מצביע (או מצביעים) לאיבר המתאים ברשימת הסמיכויות. יתרונות: מציאה, כנסה והוצאה של קשת- $O(1)$, מעבר על שכני צומת- $O(d_i)$. חיסרון: סיבוכיות זיכרון- $O(n^2)$.

מיון טופולוגי:

מספור של צמתי גרף מכוון, כך שלצומת i יינתן המספר $N[i]$ - כך שיתקיים: $i \xrightarrow{e} j \in E \Rightarrow N[i] < N[j]$. ניתן למצוא מספור כזה כאשר הגרף הוא-DAG: Directed Acyclic Graph.

מקור: הוא צומת שלא נכנסות אליו קשתות. לכל DAG יש לפחות מקור אחד.

אלגוריתם: כל עוד קיימים מקורות בגרף, תן למקור כלשהו את המספר הנוכחי וקדם את המונה ב-1. סלק את הצומת (ואת הקשתות היוצאות ממנו) מהגרף. אם המונה שווה למספר הצמתים אז המספור הושלם, אחרת בגרף יש מעגל מכוון.

עץ פורש מינימום: עץ פורש של גרף ממושקל לא מכוון G, הוא עץ פורש של G שסכום משקלי קשתותיו מינימאלי.

איסוף אשפה

הבעיה: איך לשחרר צמתים שהמשתמש לא שחרר באמצעות פקודת free.

הזיכרון מחולק פונקציונאלית ל"מחסנית ריצה" ול-"ערימת אובייקטים". הערימה (Heap) משמשת לשמירת אובייקטים המיוצרים באופן דינאמי והמחסנית שומרת לכל תוכנית מסגרת (frame) אשר מכילה את משתני התוכנית.

קיימת רשימה-AVAIL של צמתים לא מנוצלים. כל פעם שעושים malloc מקבלים צומת מרשימה זו וכשעושים free מחזירים צומת לרשימה (כלומר: לא באמת זורקים את הצמתים שכבר לא משתמשים בהם).

ניתן לגשת לצמתים ב-Heap רק דרך "משתני התוכנית: הנמצאים במחסנית הריצה.

אשפה: צמתים מנותקים, כאלה שאין אליהם מסלול מאף משתנה בתוכנית.

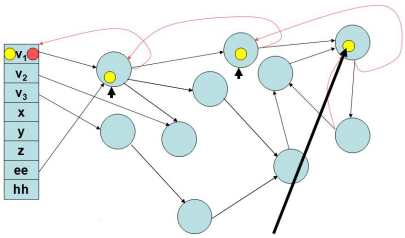
פיתרון ע"י ספירה (reference counting):

נוסיף לכל צומת מונה שסופר את מספר המצביעים אליו. כאשר מספר זה ירד לאפס, נוסיף אותו לרשימת הצמתים הלא מנוצלים ונעדכן את המונה עבור צמתים אליהם הוא הצביע.

בעיה: מעגלים תמיד מכילים צמתים עם מונה חיובי אפילו אם אין גישה למעגל כולו.

פיתרון ע"י סימון (tracing collector):

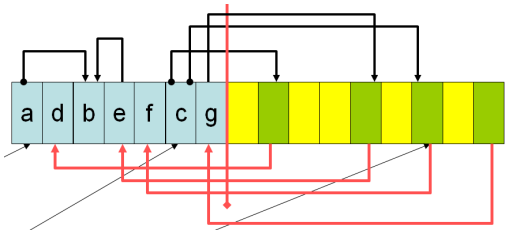
כאשר הרשימה AVAIL מתרוקנת, עוברים על הזיכרון כולו ומסמנים את כל הצמתים שיש אליהם גישה מתוך משתני התוכנית. עוברים שנית על הזיכרון ומכניסים לרשימה AVAIL את כל הצמתים שלא סומנו.



בעיה: רקורסיה לוקחת מקום: $O(n)$.

שיפור: במקום לחזור באופן רקורסיבי נשמור לצומת אליו מגיעים גם מצביע להורה שלו בשדה left/right כך שנוכל חזור אליו בסוף הביקור- *Pointer Reversal*.

פיתרון ע"י דחיסה (Compaction):



ריכוז כל הצמתים הקשורים בתחילת הזיכרון- כל שאר הצמתים הופכים להיות פנויים. נדרשת זהירות בעדכון מצביעים: העתקה של צמתים למקומם החדש אינה מספיקה. בזמן העתקה צומת v לכתובתו החדשה, נשמור במקומו הישן מצביע לכתובת החדשה. מצביע זה ישמש לעדכון המכוונים המצביעים אל v.

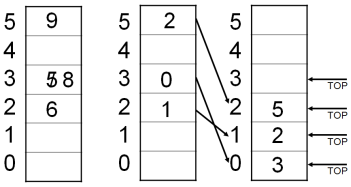
מבני נתונים נוספים:

אתחול מערך בזמן קבוע:

הרעיון: לא באמת נאתחל את המערך. נשתמש בזיכרון נוסף לציין אם כבר נכתב ערך בתא במערך.

תיאור: מלבד המערך הרגיל שבו נשים את הערכים, שני מערכים נוספים:

מערך מחסנית- ישמור את האינדקסים של המקומות התפוסים במערך האמיתי ומשתנה top- שמצביע אל התא הראשון הפנוי.



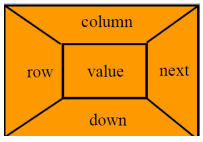
מערך מיקום האיברים במחסנית- ישמור בכל אינדקס הזהה למערך המקורי את ערך מיקום האיבר במחסנית (כך לא נצטרך לעבור על כל המחסנית כדי לבדוק אם איבר מסוים נמצא בשימוש).

גישה לאיבר שלא נעשה בו שימוש:

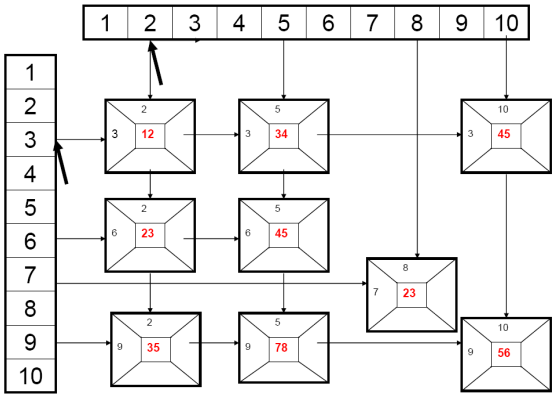
1. הערך המערך מיקום האיברים במחסנית יהיה מחוץ למחסנית (מעל top).
2. הערך יהיה בתחום של $[0,top]$ אבל באיבר הרלוונטי במחסנית לא יופיע האינדקס של המערך.

ייצוג מטריצה דלילה ע"י 2 רשימות וצמתים:

מבנה צומת:



רשימת אינדקסים של שורות ורשימת אינדקסים של עמודות.



עבור $A_{m\times n}$, בד"כ: $n^2 \gg m > n$.

גישה: $O(m)$ אתחול: $O(m)$

חיבור: $O(m+n)$ כפל: $O((q+r)n)$ כאשר q,r הוא מספר האיברים השונים מאפס בשתי המטריצות.

מחסנית:

שם הפעולה	מימוש בעזרת מערך	מימוש בעזרת רשימה מקושרת חד-כיוונית
create(S)	זמן $O(1)$, מקום $O(N)$	זמן $O(1)$
push(S,x)	זמן $O(1)$	זמן $O(1)$
top(S)	זמן $O(1)$	זמן $O(1)$
pop(S)	זמן $O(1)$	זמן $O(1)$
is-empty(S)	זמן $O(1)$	זמן $O(1)$
סה"כ זיכרון	$O(N)$	$O(n)$

תור:

שם הפעולה	מימוש בעזרת מערך	מימוש בעזרת רשימה מקושרת חד-כיוונית
Create(Q)	זמן $O(1)$, מקום $O(N)$	זמן $O(1)$
Head(Q)	זמן $O(1)$	זמן $O(1)$
Enqueue(Q,x)	זמן $O(1)$	זמן $O(1)$
dequeue(Q)	זמן $O(1)$	זמן $O(1)$
is-empty(Q)	זמן $O(1)$	זמן $O(1)$
סה"כ זיכרון	$O(N)$	$O(n)$

מטריצה תלת-אלכסונית:

מטריצה ריבועית שבה כל האיברים שווים לקבוע כלשהו פרט (אולי) לאיברי-3 האלכסונים הארוכים.

שם הפעולה	מימוש סטנדרטי: מערך דו-מימדי	מימוש מיוחד: שמירת שלושת האלכסונים הארוכים במערך
get(i)	זמן $O(1)$	זמן $O(1)$
put(i)	זמן $O(1)$	זמן $O(1)$
סה"כ זיכרון	$O(N^2)$	$O(N)$

מטריצת אלכסונים:

מטריצה ריבועים שאיברי כל אלכסון בה זהים זה לזה.

שם הפעולה	מימוש סטנדרטי: מערך דו-מימדי	מימוש מיוחד: שמירה של שורה עליונה ועמודה ראשונה בלבד.
get(i)	זמן $O(1)$	זמן $O(1)$
put(i)	זמן $O(N)$	זמן $O(1)$
סה"כ זיכרון	$O(N^2)$	$O(N)$

מטריצה דלילה:

מטריצה שבה רוב האיברים שווים לקבוע כלשהו. ידוע חסם- r על מספר האיברים השונים מהקבוע ומתקיים: $r \ll m \cdot n$.

שם הפעולה	מימוש סטנדרטי: מערך דו-מימדי	מימוש מיוחד: רשימת שלשות (i, j, a_{ij}) בסדר לקסיקוגרפי.
get(i)	זמן $O(1)$	זמן $O(\log r)$ (חיפוש בינארי)
put(i)	זמן $O(1)$	זמן $O(1)$
transpose(M)	זמן $O(nm)$	זמן $O(r \log r)$
חיבור מטריצות	זמן $O(n^2)$	זמן $o(n^2) = O(m_1 + m_2)$ מיזוג שתי הרשימות
סה"כ זיכרון	$O(mn)$	$O(r)$

<u>רשימת שלשות:</u> חיסרון עיקרי: אין גישה אקראית לפי במציין ב- $O(1)$.

יתרונות: 1. חוסך בזיכרון עבור מטריצות. 2. מאיץ פעולות חיבור וכפל של מטריצות.

רשימה מקושרת:

שם הפעולה	מימוש סטנדרטי	הערות
init(head)	זמן $O(1)$	
find(x,head)	זמן $O(n)$	
delete(t)	זמן $O(1)$	t מצביע לצומת שאחריו מוחקים
insert(t,x)	זמן $O(1)$	t מצביע לצומת שאחריו מוסיפים
שרשור רשימות מעגליות	זמן $O(1)$	
סה"כ זיכרון	$O(n)$	

<u>רשימה עם כותרת:</u> מפשט מחיקה/הכנסה של איבר ראשון.

<u>רשימה דו-כיוונית:</u> מאפשר להוציא איבר בהינתן מצביע עליו (ולא רק את האיבר שאחרי המצביע).

<u>הוצאה טריקית מרשימה חד-כיוונית:</u> העתקת מידע מהתא העוקב לתא ש-t מצביע עליו, והוצאת התא העוקב. חסרונות: ייתכן שהמידע מורכב להעתקה. ייתכן שיש מצביעים אחרים לתא שאין אפשרות לעדכן אותם.

עץ חיפוש בינארי:

שם הפעולה	מימוש סטנדרטי	הערות
create(T)	זמן $O(1)$	
find(T,x)	זמן $O(\log n)$, מקום $O(\log n)$	הזמן הינו בממוצע על הקלט
insert(T,x,info)	זמן $O(\log n)$, מקום $O(\log n)$	הזמן הינו בממוצע על הקלט
delete(T,x)	זמן $O(\log n)$, מקום $O(\log n)$	הזמן הינו בממוצע על הקלט
min(T)	זמן $O(1)$	אם קיים מצביע למינימום
okev(T,x)	זמן $O(1)$	אם קיים מצביע מכל איבר
סה"כ זיכרון	$O(n)$	

פעולות התלויות בגובה העץ: במקרה הגרוע: $O(n)$ למשל כאשר העץ הוא שרוך.

במקרה הטוב: $O(\log n)$ למשל עבור עץ שלם. במקרה הממוצע: $O(\log n)$.

עץ- AVL ועץ 2-3:

שם הפעולה	מימוש סטנדרטי	הערות
create(T)	זמן $O(1)$	
find(T,x)	זמן $O(\log n)$, מקום $O(\log n)$	
insert(T,x,info)	זמן $O(\log n)$, מקום $O(\log n)$	
delete(T,x)	זמן $O(\log n)$, מקום $O(\log n)$	
min(T)	זמן $O(1)$	אם קיים מצביע למינימום
okev(T,x)	זמן $O(1)$	אם קיים מצביע מכל איבר
סה"כ זיכרון	$O(n)$	

רשימת דילוגים רנדומאלית:

שם הפעולה	ללא מצביעים כלפי מעלה	עם מצביעים כלפי מעלה
init()	זמן $O(1)$	זמן $O(1)$
find(x,head)	זמן $O(\log n)$ בממוצע	זמן $O(\log n)$ בממוצע
insert(T,x,info)	זמן $O(\log n)$, מקום $O(\log n)$	זמן $O(\log n)$, מקום $O(1)$
delete(T,x)	זמן $O(\log n)$ בממוצע	זמן $O(\log n)$ בממוצע
סה"כ זיכרון	$O(n)$	$O(n)$

רשימת דילוגים דטרמיניסטית:

שם הפעולה	ללא מצביעים כלפי מעלה	עם מצביעים כלפי מעלה
init()	זמן $O(1)$	זמן $O(1)$
find(x,head)	זמן $O(\log n)$	זמן $O(\log n)$
insert(T,x,info)	זמן $O(\log n)$, מקום $O(\log n)$	זמן $O(\log n)$, מקום $O(1)$
delete(T,x)	זמן $O(\log n)$, מקום $O(\log n)$	זמן $O(\log n)$
סה"כ זיכרון	$O(n)$	$O(n)$

טבלת ערבול:

שם הפעולה	chain hashing	open addressing
init()	זמן $O(m)$	זמן $O(m)$
find(x,head)	זמן $O(1)$ בממוצע w.c $O(n)$	זמן $O(1)$ בממוצע w.c $O(n)$
insert(x)	זמן $O(1)$	זמן $O(1)$ בממוצע w.c $O(n)$
delete(x)	זמן $O(1)$ בממוצע w.c $O(n)$	זמן $O(1)$ בממוצע w.c $O(n)$
סה"כ זיכרון	$O(m+n)$	$O(m+n)$

ייעול chain hashing: שימוש בעץ מאוזן מקטין את הזמן הגרוע ל-$O(\log n)$.

Union-Find:

שם הפעולה	מערך בו עבור כל איבר נשמר שם הקבוצה	כל קבוצה היא רשימה מעגלית, ויש רשימה מקושרת של קבוצות
MakeSet(i)	זמן $O(1)$	זמן $O(1)$
Find(i)	זמן $O(1)$	זמן $O(n)$
Union(p,q)	זמן $O(N)$	זמן $O(1)$
סה"כ זיכרון	$O(N)$ - עבור המערך	$O(n+N)$ - לרשימות מעגליות

שם הפעולה	כל קבוצה רשימה מקושרת, מערך המכיל מצביעים אל כל האיברים	אותו הדבר עם "איחוד לפי גודל"
MakeSet(i)	זמן $O(1)$	זמן $O(1)$
Find(i)	זמן $O(1)$	זמן $O(1)$
Union(p,q)	זמן $O(n)$	זמן $O(\log n)$ משוערך
סה"כ זיכרון	$O(N+n)$	$O(N+n)$

שם הפעולה	כל קבוצה כעץ הפוך ומערך עם מצביעים אל האיברים	אותו הדבר עם "איחוד לפי גודל"
MakeSet(i)	זמן $O(1)$	זמן $O(1)$
Find(i)	זמן $O(h)$	זמן $O(\log n)$
Union(p,q)	זמן $O(1)$	זמן $O(1)$
סה"כ זיכרון	$O(N+n)$	$O(N+n)$

שם הפעולה	כל קבוצה כעץ הפוך, מערך המכיל מצביעים אל כל האיברים, "איחוד לפי גודל" ו"כיווץ מסלולים"
MakeSet(i)	זמן $O(\log^* n)$ משוערך
Find(i)	זמן $O(\log^* n)$ משוערך
Union(p,q)	זמן $O(\log^* n)$ משוערך
סה"כ זיכרון	$O(N+n)$

תור עדיפויות/ ערימה:

שם הפעולה	מימוש ע"י עץ מאוזן	מימוש ע"י עץ כמעט שלם
MakeHeap(Q)	זמן $O(n\log n)$, מקום $O(n)$	זמן $O(n)$, מקום $O(\log n)$
insert(x,Q)	זמן $O(\log n)$ מקום $O(\log n)$	זמן $O(\log n)$, מקום $O(1)$
max(Q)	זמן $O(\log n)$ מקום $O(\log n)$	זמן $O(1)$, מקום $O(1)$
DelMax(Q)	זמן $O(\log n)$ מקום $O(\log n)$	זמן $O(\log n)$ מקום $O(\log n)$
סה"כ זיכרון	$O(n)$	$O(n)$

Trie:

שם הפעולה	מימוש ע"י עץ מאוזן	מימוש ע"י עץ כמעט שלם
Find(s)	זמן $O(\|s\|)$	זמן $O(n)$, מקום $O(\log n)$
Insert(s)	זמן $O(\|s\|)$	זמן $O(\log n)$, מקום $O(1)$
Delete(s)	זמן $O(\|s\|)$	זמן $O(1)$, מקום $O(1)$
FindMin()	זמן $O(\|s_{\min}\|)$	זמן $O(\log n)$ מקום $O(\log n)$
סה"כ זיכרון	$O(n)$	$O(n)$

גרפים

שם הפעולה	מטריצת סמיכויות	רשימת סמיכויות
Find(i,j)	זמן $O(1)$	זמן $O(d_i)$
Insert/Delete(i,j)	זמן $O(1)$	זמן $O(d_i)$ לבדוק האם קיים
Neighbors(i)	זמן $O(n)$ - מעבר על שורה i	זמן $O(d_i)$
סה"כ זיכרון	$O(n^2)$	$O(n+m)$

הערה: d_i (דרגת צומת i) במקרה הגרוע יכול להיות $O(n)$.

אלגוריתמי מיון:

שם המיון	זמן	מקום נוסף	הערות
Bubble Sort	$O(n^2)$	$O(1)$	מתבסס על החלפת איברים סמוכים על בסיס השוואות. בכל איטרציה מובא האיבר הגדול ביותר לקצה המערך, והמערך שמטפל באיטרציה הבאה "קטן" ב-1.
Heap Sort	$O(n\log n)$	$O(1)$	מיון המתבסס על מבנה הנתונים ערימה. בתחילה מאוחסנים הנתונים במבנה ערימה בזמן $O(n)$. לאחר מכן בכל איטרציה מוצא האיבר המקסימאלי ומתבצעת פעולת עדכון לשמירת חוקיות הערימה- $O(\log n)$. סה"כ מוציאים n איברים, ולכן הסיבוכיות היא $O(n\log n)$.
Quick Sort רנדומאלי	$O(n\log n)$ ממוצע $O(n^2)$ במקרה הגרוע	$O(n)$ במימוש הרגיל. $O(\log n)$ אם מעלימים את רקורסיית הזנב	בכל איטרציה נבחר איבר ציר באופן אקראי והאיברים במערך מסודרים כך שמשמאל לאיבר הציר כל האיברים הקטנים ממנו, ומימין האיברים הגדולים או שווים לו. לאחר-מכן נשלחים שני המערכים שנוצרו בקריאה רקורסיבית לתהליך דומה.
Quick Sort דטרמיניסטי	$O(n\log n)$	$O(\log n)$	בכל איטרציה נבחר החציון בצורה דטרמיניסטית, ולכן מושג חסם עליון טוב יותר על זמן המיון.
Merge Sort	$O(n\log n)$ כל איטרציה לוקחת $\Theta(n)$ וקיימות $\Theta(\log n)$ איטרציות	$O(n)$	פונקצית merge- ממזגת שני מערכים ממוינים (שאורכם יחד k) בזמן $O(k)$. תהליך המיון כולל חלוקה של מערך הקלט לתת-מערכים, עד לאורך של-1 ולאחר מכן מיזוג של כל שניים באופן רקורסיבי.
Bucket Sort	$O(n+k)$	$O(1)$	יעיל עבור מיון n המספרים הראשונים בתחום $1\ldots k$. מיון זה הוא מיון יציב.
Radix Sort	$O(n)$ אם הבסיס קבוע $O(d(n+b))$	$O(1)$	מיון זה הוא מיון יציב. בסיבוכיות- d - מספר הספרות, b - הבסיס