נוסחאות כלליות

כללי:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=k<\infty \Rightarrow f(n)=\theta(g(n))$$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty \Rightarrow f(n)=\Omega(g(n)) \;\&\&\; f(n)\neq O(g(n))$$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0 \Rightarrow f(n)=O(g(n)) \;\&\&\; f(n)\neq \theta(g(n))$$

$$\lceil n/a\rceil/b=\lceil n/(a\cdot b)\rceil$$

$$\lceil n/2\rceil+\lfloor n/2\rfloor=n$$

$$a^{\log_b n}=n^{\log_b a}$$

$$\log_b a=\frac{\log_c a}{\log_c b}$$

$$\log_c ab=\log_c a+\log_c b$$

$$\log\left(\prod_{k=1}^{n}a_k\right)=\sum_{k=1}^{n}\log a_k$$

$$\binom{n}{k}=\frac{n!}{k!(n-k)!}$$

$$\binom{n}{k}\le\frac{n^k}{k!}\le\left(\frac{en}{k}\right)^k$$

$$\sum_{k=1}^{n}k=\frac{1}{2}n(n+1)=\theta(n^2)$$

$$\sum_{k=0}^{n}x^k=\frac{x^{n+1}-1}{x-1}\xrightarrow{|x|<1}\frac{1}{1-x}$$

$$\sum_{k=0}^{\infty}kx^k=\frac{x}{(1-x)^2}$$

$$(x+y)^n=\sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k}$$

$$(a+b)\bmod m=(a\bmod m+b\bmod m)\bmod m$$

$$(a\cdot b)\bmod m=\big((a\bmod m)\cdot(b\bmod m)\big)\bmod m$$

$\log^* n$ הוא מספר הפעמים שצריך לבצע $\log(\log\cdots(\log(n))))$ ע"מ שנקבל 1.

פונקציה מונוטונית עולה:

$$\int_{m-1}^{n}f(x)dx\le\sum_{k=m}^{n}f(k)\le\int_{m}^{n+1}f(x)dx$$

פונקציה מונוטונית יורדת:

$$\int_{m}^{n+1}f(x)dx\le\sum_{k=m}^{n}f(k)\le\int_{m-1}^{n}f(x)dx$$

נוסחת סטרלינג:

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\le n!\le\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n+(1/12n)}$$

פולינומים:

רגיל:

$$P=a_0+a_1x^1+a_2x^2+\ldots+a_nx^n$$

מספר פעולות כפל בפולינום רגיל:
$(n^2+n))/2$

פולינום בצורת הורנר:

$$P=(\ldots(((a_nx+a_{n-1})x+a_{n-2})x+a_{n-3})x\ldots+a_1)x+a_0$$

מספר פעולות כפל בפולינום הורנר: n.
מספר פעולות חיבור בפולינום הורנר: n.

סכום של סדרה הנדסית:

$$a_n=a_1\cdot q^{n-1}\Rightarrow S_n=\frac{a_1(q^n-1)}{q-1}$$

הסדרה תמיד מתכנסת כאשר q<1.

סכום של סדרה חשבונית:

$$a_n=a_0+n\cdot d$$

$$S_n=(a_0+a_n)\cdot\frac{n+1}{2}$$

סיבוכיות

סיבוכיות אסימפטוטית:

$$f(n)=\theta(g(n))\equiv\left\{f(n)\;\middle|\;\begin{matrix}for\ every\ n\ge n_0:C_i>0\\C_1\cdot g(n)\le f(n)\le C_2\cdot g(n)\end{matrix}\right\}$$

$$f(n)=O(g(n))\equiv\left\{f(n)\;\middle|\;\begin{matrix}for\ every\ n\ge n_0:C_i>0\\0\le f(n)\le C\cdot g(n)\end{matrix}\right\}$$

$$f(n)=\Omega(g(n))\equiv\left\{f(n)\;\middle|\;\begin{matrix}for\ every\ n\ge n_0:C_i>0\\0\le C\cdot g(n)\le f(n)\end{matrix}\right\}$$

הערה:
מספיק שהנוסחאות מתקיימות מ n מסויים.

מציאת סיבוכיות לרקורסיות:

הערה:
בד"כ ברקורסיה הסיבוכיות תלויה בסכום הסיבוכיות של כל עלה בעץ.
ובמיונים הסיבוכיות תלויה במסלול אחד ולכן תלויה בגובה העץ.

1. **שיטת ההצבה החוזרת:**
הסבר ע"פ דוגמא – ע"פ האלגוריתם רושמים נוסחה רקורסיבית של מס' הפעולות שמבצע האלגוריתם:

$$T(n)=c_1+c_2\cdot n+2\cdot T(\frac{n}{2})$$

$$T(n)=c_1+c_2\cdot n+2\cdot\left[c_1+c_2\cdot\frac{n}{2}+2\cdot T(\frac{n}{4})\right]$$

$$T(n)=(1+2)c_1+(1+1)c_2\cdot n+4\cdot\left[c_1+c_2\cdot\frac{n}{4}+2\cdot T(\frac{n}{8})\right]$$

:

$$T(m)=(2^0+2^1+\ldots+2^m)c_1+m\cdot n\cdot k\cdot c_2+2^m T(\frac{n}{2^m})$$

כל פעם מציבים את הנוסחה המקורית.

נבחר $2^m=n$ כדי שנקבל **T(1) = קבוע** (שזהו השלב האחרון):

$$T(n)=(n-1)\cdot c_1+c_2\cdot n\cdot\log_2 n+n\cdot T(1)=\theta(n\cdot\log n)$$

2. **שיטת עץ הרקורסיה:**
בונים עץ רקורסיה (ע"פ חלוקת הרקורסיה).
בכל קודקוד רושמים את מספר הפעולות (סיבוכיות) שמבצעת הרקורסיה ללא הסיבוכיות של תתי הרקורסיות.
לבסוף סוכמים את הסיבוכיות של כל הקודקודים בעץ, וזו הסיבוכיות של הרקורסיה עצמה.

3. **משפט המאסטר:**

$$a>1;b>1;\left\lceil\frac{n}{b}\right\rceil;\left\lfloor\frac{n}{b}\right\rfloor\Rightarrow p=\log_b a$$

$$T(n)=a\cdot T(\frac{n}{b})+f(n)$$

הערה: a, b אינם משתנים! אלא מספרים קבועים.

מקרה א':
אם קיים $\varepsilon>0$ קבוע כך ש: $f(n)=O(n^{p-\varepsilon})$ אז:
$T(n)=\theta(n^p)$

הערה: חייבים לתת דוגמא לאפסילון כזה.

מקרה ב':
אם $f(n)=\theta(n^p)$ אז:

$$T(n)=\theta(n^p\cdot\log n)$$

מקרה ג':
אם קיים $\varepsilon>0$ כך ש: $f(n)=\Omega(n^{p+\varepsilon})$
ואם קיים $c<1$ כך שלכל n מספיק גדול מתקיים:
$a\cdot f(\frac{n}{b})\le c\cdot(n^{p+\varepsilon})$:

אז:

$$T(n)=\theta(f(n))$$

הערה: חייבים לתת דוגמא לאפסילון כזה.

הערות:
- כל בסיס קבוע של log לא משנה את הסיבוכיות.
- אם הבסיס לא קבוע (n) אז נציב בנוסחת החלפת הבסיסים ונוכל להשתמש במשפט הקודם.

ניתוח אלגוריתם:

1. **נכונות:**
- תנאי עצירה.
- פלט נכון.

2. **יעילות:**
בדיקת זמן ריצה = **סיבוכיות**.

זמן ריצה: מספר פעולות בסיסיות שאלגוריתם מבצע כפונקציה של גודל הקלט וכאשר הוא רץ על **הקלט הגרוע ביותר.**

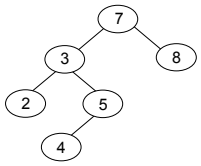
אלגוריתמי חיפוש

Naive Search:
עובר על כל הרשימה ומחפש את הערך.
$\theta(n)$

Binary Search (שיטת החציה):
$\theta(\log n)$

הנחה: משתמש בעובדה שהמספרים ממויינים.
הסבר: כל פעם הוא חותך את המערך ל- 2 ובודק היכן x נמצא (מימין או משמאל).

Breadth First Search - BFS:
הסבר: חיפוש לרוחב - עוברים על כל האיברים בעץ כלשהו משמאל לימין (או מימין לשמאל) לפי רמות (רמה = גובה מסוים בעץ).
דוגמא:



האלגוריתם ידפיס: 7,3,8,2,5,4

הוכחת נכונות:
- מוכיחים נכונות עבור הדפסת כל הקודקודים באינדוקציה לפי מרחק מהשורש.
אח"כ מוכיחים נכונות עבור הדפסת הקודקודים בסדר הנכון - באינדוקציה על i.
כאשר u_i , ... u_1 הם הקודקודים לפי סדר ההדפסה.
יש להוכיח את 2 הטענות הבאות באינדוקציה:
1.

$$1\le j<i\Rightarrow dist(v,u_i)\ge dist(v,u_j)$$

2. אם:

$$dist(v,u_i)>dist(v,u_{i-1})$$

אז בהכרח:

$$dist(v,u_i)=dist(v,u_{i-1})+1$$

וכן כל x שהמרחק שלו קטן מהמרחק של u_1 כבר ביקרנו (הודפס).

מימוש בעזרת תור:
- נתחיל בשורש ונכניס את ילדיו לתור (כאשר הילד ההשמאלי ראשון).
- כל איטרציה נוציא את הקודקוד הבא מהתור, נדפיס אותו ונכניס לתור את ילדיו (אלה שעוד לא הודפסו).
- עבור כל קודקוד שנדפיס, נסמן במערך VISITED שהוא הודפס, בכדי שלא נדפיס אותו פעמיים.
- האלג' יעצר ברגע שהתור יהיה ריק.

סיבוכיות:
x - הקודקוד שכרגע בודקים.
n - מספר הקודקודים.
m - מספר הצלעות.

$$T_{BFS}(n,m)\le c_1n+c_2+\underbrace{\sum_{x=1}^{n}(c_3+c_4\cdot\deg(x))}_{while}$$

$$T_{BFS}(n,m)=O(n+m)\Leftarrow\sum_{x=1}^{n}c_4\cdot\deg(x)\le 2m$$

הערה: כאשר משתמשים ב BFS למצוא מסלול באלגוריתם גרף שיורי, תמיד נקבל את המסלול הקצר ביותר.

Depth First Search - DFS:
הסבר:
- חיפוש לעומק - תמיד מבקרים אצל הילד השמאלי ראשון ומדפיסים אותו.
- אח"כ עובר לילד השמאלי של הילד השמאלי וכך הלאה.
- כאשר מגיעים לעלה, עוברים לתת העץ הימני של קודקוד העלה, ומשם ממשיכים באותו האופן.

דוגמא לפי העץ הנ"ל: 7,3,2,5,4,8

מימוש בעזרת מחסנית:
- נתחיל בהדפסת השורש ונכניס את ילדיו למחסנית כך שהשמאלי יכנס אחרון.
- בכל איטרציה נוציא קודקוד מהמחסנית, נדפיס אותו ונכניס את ילדיו למחסנית.
- לאחר שהדפסנו קודקוד נעדכן במערך VISITED שהוא הודפס, בכדי שלא נדפיס אותו פעמיים.
- נמשיך כך עד אשר המחסנית תהיה ריקה.

אלגוריתמי מיון

אלגוריתם מיון יציב:
אלג' נקרא יציב אם מספרים זהים בקלט, יסודרו בפלט, בדיוק באותו הסדר שהופיעו בקלט.

מציאת סיבוכיות מינ' של אלג' מיון מבוסס השוואות:
הסבר:
- בעזרת עץ בינארי ניתן להציג מיונים מבוססי השוואות.
- כל קודקוד הוא השוואה בין 2 איברים.
- כל מסלול מהשורש עד לעלה מתאר מיון שונה.
- לכן, כל עלה מייצג מיון שונה.
- ל- n איברים קיימים !n מיונים שונים.

הוכחה:
מספר העלים בעץ כזה הוא **בדיוק** !n.
מספר העלים **המקס'** (מקס' 2 ילדים לכל אב) הוא 2^h .
ע"פ נוסחאת סטרלינג ידוע ש:

$$2^h\ge n!\ge(n/e)^n$$

לכן מתקיים:

$$h\ge n\cdot\log(n/e)=\Omega(n\cdot\log n)$$

גובה העץ (h) הוא הסיבוכיות של הבעיה.

מיונים שונים:

Insertion Sort:
האלגוריתם עובר משמאל לימין על כל האברים במערך ובודק אם האיבר הבא קטן מזה שלפניו.
אם האיבר הבא קטן מזה שלפניו האלג' משכפל את האיבר הקודם ומכניס אותו במקום האיבר הקטן, ואז הוא ממשיך להעביר את האיבר הקטן שמאלה, עד שהוא מגיע למקום שהוא צריך להכנס, ולשם הוא מכניס אותו (Insert).

דוגמא לריצה:

5,	12,	7,	1
12,	12,	7,	1
12,	7,	7,	1
12,	7,	5,	1

סיבוכיות:
$\theta(n^2)$

הסבר לסיבוכיות: אם נסתכל על הקלט הגרוע ביותר (מסודר בסדר הפוך) אז זמן הריצה יתקבל ע"י סכום של סדרה חשבונית.

Merge Sort:
האלגוריתם פועל בשיטת "הפרד ומשול", הוא מחלק את הבעיה לתת בעיות ופותר כל תת בעיה בנפרד.
אחרי פתרון תתי הבעיות האלג' משתמש בפונקציה בשם Merge שמאחדת 2 מערכים ממויינים למערך אחד ממויין.
סיבוכיות:
$\theta(n\cdot\log n)$

סיבוכיות Merge:
$\theta(r-p)$
הערה: r-p+1 = מספר האיברים ב- 2 המערכים יחד.

Counting Sort (יציב):
הנחה: נתון מספר k וכל מספר בקלט הוא שלם וקטן או שווה ל- k.
הסבר: מערך A הוא הקלט. מערך B הוא הפלט הממוין. במערך C מאחסנים את השכיחות של כל איבר.
אח"כ עוברים על C מהתא מס' 1 עד n (משמאל לימין) ובכל תא רושמים את השכיחות המצטברת של כל איבר.
השכיחות המצטברת של איבר מסוים היא המיקום שלו במערך B.
מכניסים את המספרים למערך B ע"י מעבר על מערך A מהסוף להתחלה (מימין לשמאל).

סיבוכיות:
$\theta(n+k)$

Radix Sort (יציב):
הנחה: כל ההנחות של Counting Sort תקפות + לכל המספרים במערך A אותו מספר ספרות (d).
הסבר: משתמש באלגוריתם Counting Sort וממיין מספרים בעלי אותו מספר ספרות.
האלג' מתחיל למיין את המספרים לפי ספרת האחדות שלהם, וממיין באותו אופן את כל הספרות. לבסוף מקבלים מערך ממויין.
הערה: אלגוריתם זה מתבסס על Counting Sort ולכן גם הוא יציב.
הוכחת נכונות באינדוקציה על מספר הסיבות.
סיבוכיות:
$\theta(d\cdot(n+k))$

Bucket Sort:
הנחות:
- כל קבוצת מספרים, שניתן לדעת מראש שניתן לחלקם לקטעים, כך שלכל קטע הסתברות שווה, ניתנת למיון ע"י Bucket Sort.
- פיזור המספרים הוא אחיד במרחב.

הסבר: האלגוריתם בונה מערך של רשימות מקושרות ומכניס כל איבר לרשימה המקושרת כלשהי באופן ממויין.
בסוף נקבל מערך של רשימות מקושרות שמכילות את האיברים הממויינים .
רשימה מקושרת השייכת לתא מסוים במערך למעשה מכילה מספרים בתחום מסוים.
אם נחבר את כל הרשימות המקושרות לפי הסדר שהן מופיעות במערך נקבל רשימה מקושרת אחת ארוכה וממויינת.

שלבי האלגוריתם:
- ניצור מערך בן n מקומות של רשימות מקושרות.
- נעבור על כל האיברים ונבדוק עבור כל אחד איזה j יגרום לתחום הבא להכיל את המספר:

$$\left(\frac{j-1}{n}, \frac{j}{n}\right]$$

- נכניס אותו באופן ממויין לרשימה המקושרת שמבקום ה- j במערך.

סיבוכיות ממוצעת:

$\theta(n)$

Quick Sort:

כללי:
- המטרה - למיין מערך של מספרים.
- זוהי פונקציה רקורסיבית המתבססת על שיטת Divide & Conquer.

אופן פעולה:
- קיימת פונקציה בתוך Quick Sort שנקראת Partition.
- פונקציית Partition:
- בוחרת את המספר הראשון מתוך המערך = q.
- יש 2 מצביעים הרצים על המערך, אחד מצד משמאל ואחד מצד ימין.
- המצביע הימני מחפש מס' הקטן או שווה ל- q.
- השמאלי מחפש מס' הגדול או שווה ל- q.
- כאשר שניהם מצאו מספרים מתאימים, הם יחליפו ביניהם.
- כך ימשיכו עד אשר הם יפגשו.
- בסוף נקבל את כל המספרים הגדולים מ- q מצדו הימני, ואת כל המספרים הקטנים מ- q מצדו השמאלי.
- לאחר ש- Quick Sort הפעילה את Partition, היא תקרא לעצמה באופן רקורסיבי - פעם עם הצד השמאלי של q ופעם עם הצד הימני של q.
- לבסוף נקבל מערך ממויין.

הערה:
הגרסה הרנדומלית בוחרת את q באופן רנדומלי.

חלוקה מאוזנת - "טובה":
- חלוקה שבה בין ¼ מהאיברים לבין 0 נמצאים בצד אחד ובין ¾ לכל האיברים נמצאים בצד שני.
- חיבור של 2 הצדדים נותן את כל האיברים.
- **הגרסה הרנדומלית תניב חלוקה מאוזנת (טובה).**

<u>סיבוכיות פתרון בעזרת רקורסיה + Meomize וגם עבור פתרון Buttom Up</u>

$T(n)=T(q)+T(n-q)+c_1\cdot n$

הסיבוכיות של Partition:

$T(n)=\theta(r-p)$

$r=n\ ;\ p=1 \Rightarrow T(n)=c_1\cdot n=\theta(n)$

המקרה הגרוע - הלא מאוזן (כאשר q תמיד = 1):

$T(n)=T(1)+T(n-1)+c_1\cdot n$

$T(n)=O(n^2)$

המקרה הממוצע - המאוזן:

$0<\varepsilon\le 1/2$

$T(n)=T(\varepsilon\cdot n)+T((1-\varepsilon)\cdot n)+c_1\cdot n$

$T(n)=O(n\cdot\log n)$

הגרסה הרנדומלית:

$T(n)=O(n\cdot\log n)$

<u>גרפים</u>

הגדרות:
V - Vertices - קודקודים.
E - Edge - קבוצה המייצגת את כל הצלעות בגרף.
G=(V, E) - גרף - אוסף כל הצלעות E והקודקודים V.
Degree - דרגה של קודקוד = מספר השכנים.
גרף קשיר - אם קיים מסלול בין כל 2 קודקודים בגרף.

<u>מבני נתונים</u>

הגדרה: טיפוס נתונים אבסטרקטי: (Abstract Data Type - ADT)
מבנה מתמטי הכולל פעולות קבועות שניתן לבצע עליו.
1. **INSERT (X,S):** מוסיף את X.
2. **DELETE (X,S):** מוציא את X.
3. **MEMBER (X,S):** מחזיר אמת אם X ב- S ושקר אחרת.
4. **MIN(S):** מחזיר את האיבר המינימלי.
5. **MAX(S):** מחזיר את האיבר המקסימלי.
6. **SUCCESSOR(X,S):** מחזיר את האיבר הבא בגודלו אחרי X.
7. **PREDECESSOR(X,S):** מחזיר את האיבר הקודם לפני X.

ADT - קבוצה דינמית:

(יחס סדר מלא על האיברים)
1. **FIRST(L)** - מחזיר את האיבר הראשון.
2. **LAST(L)** - מחזיר את האיבר האחרון.
3. **RETRIEVE(P,L)** - מחזיר את האיבר במקום ה-P.
4. **LOCATE(X,L)** - מחזיר את מיקום X ברשימה.
5. **EMPTY(L)** - מחזיר אמת אם ריק ושקר אחרת.
6. **INSERT(X,P,L)** - מכניס את X למקום P ברשימה.
7. **DELETE(P,L)** - מוחק את האיבר הP ברשימה.

<u>רשימות (List)</u>

אוסף של איברים ביחד עם הפעולות הבאות:
1. **FIRST(L)** - מחזיר את האיבר הראשון.
2. **LAST(L)** - מחזיר את האיבר האחרון.
3. **RETRIEVE(P,L)** - מחזיר את האיבר במקום ה- P.
4. **LOCATE(X,L)** - מחזיר את מיקום X ברשימה.
5. **EMPTY(L)** - מחזיר אמת אם ריק ושקר אחרת.
6. **INSERT(X,P,L)** - מכניס את X למקום P ברשימה.
7. **DELETE(P,L)** - מוחק את האיבר הP ברשימה.

מחסנית (Stack) - FILO:

במחסנית ההכנסה וההוצאה מתבצעות מאותו קצה.
פעולות על המחסנית:
1. **TOP(S)** - מחזיר את האיבר הראשון במחסנית.
2. **EMPTY(S)** - מחזיר אמת אם S ריקה אחרת שקר.
3. **PUSH(X,S)** - מכניס איבר חדש לקצה המחסנית.
4. **POP(S)** - מוחק איבר ראשון במחסנית ומחזיר אותו.

הערה: כאשר ממשים את הפעולות הנ"ל ע"י מערך הסיבוכיות של כל פעולה היא קבוע.

סיבוכיות – Top, Empty, Push, Pop:

$\theta(1)$

תור (Queue) - FIFO:

בתור, ההכנסה וההוצאה מתבצעות בקצוות שונים.
פעולות על התור:
1. **FRONT(Q)** - מחזיר את האיבר הראשון בתור.
2. **EMPTY(Q)** - מחזיר אמת אם ריק אחרת שקר.
3. **ENQUE(X,Q)** - מכניס את X לסוף התור.
4. **DEQUE(Q)** - מוחק איבר ראשון ומחזיר אותו.

סיבוכיות–Front, Empty, Enqueue, Dequeue:

$\theta(1)$

מימוש תור ע"י מערך:
רשומה המכילה את השדות הבאים:
- מערך בשם Elements בגודל MAXSIZE.
- משתנה Front.
- משתנה Rear.

חסרון : התור מכיל רק MAXSIZE אברים.

מימוש תור ע"י מערך מעגלי:
ניתן לממש תור ע"י מערך שהתא האחרון שלו מצביע על התא הראשון (מערך מעגלי).
צורה זו יעילה לשמוש בתור שאפשר להוציא ולהכניס אברים מ 2 קצוות שלוץ

מימוש תור בעזרת 2 מחסניות:
סיבוכיות:
Reverse-stack

$\theta(n)$

ENQUEUE

$\theta(1)$

DeQUEUE (ריצה אחת של אלג')

$\theta(n)$

שיפור של מימוש תור ע"י 2 מחסניות:
סיבוכיות:
DeQUEUE (n ריצות של אלגוריתם זה)

$\theta(n)$

ENQUEUE

$\theta(1)$

תור קדימויות מינימלי - Priority Queue:
P - אוסף של אלמנטים, לכל אלמנט יש ערך (מס' שלם).
X - הקדימות עצמה.

הערה:
ניתן לממש תור קדימויות ע"י עץ בינארי סדור חלקית.

פעולות על תור הקדימויות:
1. **INSERT (X,P)** - מכניס את האלמנט x לתור.
2. **EMPTY(P)** - מחזיר אמת אם ריק אחרת שקר.
3. **DELETEMIN(X,P)** - מוחק את האלמנט עם הקדימות המינימלית, ומחזיר אותו.

הערות:
- אין שתי קדימויות זהות.
- כנ"ל לגבי תור קדימויות מקסימלי.

<u>ערימה - Heap</u>

רשומה המדמה עץ, בעלת 2 חלקים:
- מערך P.T[ ] - שומר את ערכי הקודקודים.
- P.Size - משתנה השומר את מספר אברי המערך.

<u>עצים</u>

קודקוד - רשומה עם השדות הבאים (במקרה המינ'):
Parent - מבציע להורה.
Value - ערך הקודקוד.
Right Child - מצביע לילד הימני.
Left Child - מצביע לילד השמאלי.
שורש - הקודקוד הראשי.
עלה - קודקוד ללא ילדים כלל.
דרגה - Degree - מספר הילדים שיש לקודקוד.
גובה - h - אורך המסלול הארוך ביותר מהשורש לעלה.

עץ בינארי:

- עץ שבו לכל אב יש מקסימום 2 ילדים.
- גובה העץ הבינארי הוא מקסימום $\log_2 n$ (n - מספר העלים המקסימלי).

עצי חיפוש בינארים:

- עץ בינארי.
- עבור כל קודקוד - כל ערך בתת העץ השמאלי קטן מערך הקודקוד.
- עבור כל קודקוד - כל ערך בתת העץ הימני גדול מערך הקודקוד.
- האיבר הימני ביותר (לא בהכרח ברמה התחתונה) יהיה תמיד הכי גדול.
- האיבר השמאלי ביותר (לא בהכרח ברמה התחתונה) יהיה תמיד הכי קטן.
- עבור קבוצת מספרים מסוימים, יש מספר אפשוריות לבניית עץ חיפוש בינארי.
- עץ חיפוש בינארי **אינו** בהכרח מאוזן.

הערה: BFS ו- DFS - אלגוריתם חיפוש לרוחב עץ ולעומק העץ, נמצא תחת הכותרת אלגוריתמי חיפוש.

פעולת Member בעץ חיפוש בינארי:
- משווים את X לשורש. אם הוא שווה לשורש מחזירים אמת.
- אם הוא קטן עוברים לילד השמאלי.
- אם הוא גדול עוברים לילד הימני.
- כך ממשיכים עד שמוצאים את X או עד שמגיעים לעלה (סוף העץ).

פעולת Min בעץ חיפוש בינארי:
- יורדת בעץ שמאלה עד אשר אין ילד שמאלי, ומחזירה את הערך של אותו קודקוד.
- כנ"ל לגבי **Max** רק ימינה.

פעולת Insert בעץ חיפוש בינארי:
- יורדים במורד העץ כמו ב- Member.
- כאשר מגיעים למי שאמור להיות ההורה של X, מוסיפים את X בתור הילד שלו.

פעולת Delete בעץ חיפוש בינארי:
- נמצא את הקודקוד בעל הערך X כמו ב- Member.
- אם הוא עלה, נמחוק אותו.
- אם הוא לא עלה וישׁ לו ילד יחיד – נמחוק אותו ונשים את הילד במקומו.
- אם יש לו 2 ילדים, נביט בתת העץ הימני שלו, ונמצא את הערך המינימלי שם. נמחוק את X ובמקומו נכניס את הערך המינימלי שמצאנו.

סיבוכיות פעולות בעץ חיפוש בינארי:
(Insert, Delete, Min, Max, Member)

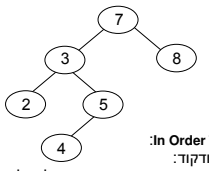
$O(n)$

אם אחרי כל פעולת Delete או Insert נאזן את העץ נקבל:

$\theta(\log n)$

איזון = סידור העץ מחדש כך שנקבל גובה מינימלי.

פרוצדורות להדפסת ערכי הצמתים בעץ חיפוש בינארי:



In Order Tree Walk:
עבור כל קודקוד:
- מדפיס קודם את תת-העץ השמאלי של קודקוד.
- אח"כ את הקודקוד עצמו.
- ולבסוף את תת-העץ הימני.

שימוש:
עבור עץ חיפוש בינארי - הדפסת ערכי העץ מהקטן לגדול.

דוגמא לסדר הדפסה: 2,3,4,5,7,8.

סיבוכיות:

$O(n)$

PreOrder Tree Walk:
עבור כל קודקוד:
- מדפיס קודם את הקודקוד.
- אח"כ את תת-העץ השמאלי.
- ולבסוף את תת-העץ הימני.

שימוש:
בעץ מסוג Trie ידפיס את כל הקודקוד בסדר מילוני (א"ב).

הגדרת Trie:
- בכל צומת שמורה מילה.
- המסלול אל מילה יוצר את המילה עצמה.
- צמתים המכילים מילים משמעותיות מודגשים.

דוגמא לסדר הדפסה: 7,3,2,5,4,8.

סיבוכיות:

$O(n)$

PostOrder Tree Walk:
עבור כל קודקוד:
- מדפיס את תת-העץ השמאלי.
- אח"כ את תת-העץ הימני.
- ולבסוף את הקודקוד עצמו.

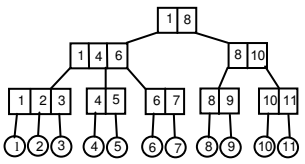
שימוש:
חישוב ערך של ביטוי מתמטי.
העץ מחשב סדרה של פעולות מתמטיות על מספרים:
- כל קודקוד מהווה פעולה מתמטית (+, -, X, /).
- העלים הם המספרים עליהם מבצעים את הפעולות.

דוגמא לסדר הדפסה:
2,4,5,3,8,7

סיבוכיות:

$O(n)$

עצי חיפוש 3-2:



הגדרות:
- לכל קודקוד פנימי יש 2 או 3 ילדים.
- כל מסלול מהשורש לעלה הוא באותו אורך (כל העלים ברמה התחתונה).
- העלים מסודרים מהקטן לגדול, כאשר הקטן משמאל.
- בכל קודקוד (לא עלה) יש 3 שדות. כל שדה מכיל את הערך המינימלי של תת העץ הקשור אליו.

פעולת Member בעץ 3-2:
- משווים את X לערכי השדות בשורש. אם הוא שווה לאחד הערכים אז מחזירים אמת.
- אם הוא קטן מהערך השמאלי (הקטן) ביותר נחזיר שקר.
- אם הוא בין הערך השמאלי לאמצעי, נעבור לילד השמאלי.
- אם הוא בין הערך האמצעי לימני (הגדול) נעבור לילד האמצעי.
- אם הוא גדול מהערך הימני, נעבור לילד הימני.
- כך ממשיכים עד שמוצאים את X או עד שמגיעים לעלה.
- אם הגענו לעלה מחזירים שקר.

פעולת Max / Min בעצי 3-2:
כמו הפעולות בעץ חיפוש בינארי.

פעולת Insert בעצי 3-2:
- נרד בעץ עד אשר נמצא מי שאמור היה להיות ההורה של X.
- **אם להורה יש 2 ילדים** - נוסיף את X כילד 3.
- **אם להורה יש 3 ילדים** - נפצל את ההורה ל- 2 הורים נפרדים. לשמאלי ניתן את 2 הערכים הקטנים, ולימני את ה- 2 הגדולים.
אם להורה של ההורה היו 2 ילדים, כעת יהיו לו 3.
אם להורה של ההורה היו 3 ילדים, נפצל גם אותו ל- 2, ונמשיך כך עד נגיע להורה עם 2 ילדים, או עד אשר נגיע לשורש (וגם אותו נפצל ונבנה שורש חדש).
- לאחר כל פעולת Insert נעדכן את שדות המינימום בכל הקודקודים הרלוונטים.

פעולת Delete בעצי 3-2:
- נרד במורד העץ עד אשר נמצא את ההורה של X.
- אם ל- X 2 אחים אז נמחוק את X.
- אם ל- X אח אחד, נמחוק את X ונתקן את העץ באופן הבא:
נסתכל על האחים של ההורה. אם לאחד מהם יש 3 ילדים, נצרף את הילד ה"קרוב" ביותר לילד הבודד.
אם לשנים יש 2 ילדים, אז נצרף את X כילד 3 לאחד מהם, וננמחוק את ההורה של X.
אם מחקנו את ההורה – יש לבדוק אם להורה של ההורה יש כעת ילד 1 או 2 ובהתאם לסד"פ הנ"ל להמשיך.
תנ"כ פעולת Delete נעדכן את שדות המינימום בכל הקודקודים הרלוונטים.

סיבוכיות פעולות בעצי 3-2 (מאוזן):
(Insert, Delete, Min, Max, Member)
נובע עקב כך שהעץ **מאוזן תמיד**:

$\theta(\log n)$

עץ בינארי סדור חלקית (ערימה מינימום):

הגדרה:
עץ בינארי, כאשר לכל קודקוד יש ערך (מספר שלם):
- הערך בכל קודקוד קטן מהערכים שבשני ילדיו.
- בכל רמה יש לכל קודקוד 2 ילדים, מלבד ברמה האחרונה.
- העלים התחתונים תמיד יהיו מרוכזים לשמאל.
- העלה העליון ביותר הוא בעל הקדימות המינימלית.
- עבור קלטים זהים תמיד נקבל עץ יחיד.

הערה:
ניתן לממש תור קדימויות ע"י עץ בינארי סדור חלקית.

פעולת Insert בעץ בינארי סדור חלקית:
- מוסיפים עלה חדש מימין לעלה הימני ביותר ברמה התחתונה ביותר.
- ע"מ לשמור על חוקיות העץ – נחלחל את העלה למעלה - נשווה אותו עם ההורה שלו.
- אם הוא קטן מההורה, נחליף אותו עם ההורה.
- נמשיך ככה עד אשר הוא יותר גדול מההורה.

פעולת Deletemin בעץ בינארי סדור חלקית:
- נחזיר את הערך בשורש.
- נמחוק את השורש.
- נעביר את ערכו של העלה הימני ביותר ברמה התחתונה לשורש.
- נמחוק את העלה הימני ביותר ברמה התחתונה.
- נחלחל את השורש למטה – נשווה אותו לבניו.
- נחליף אותו עם הקטן מביניהם.
- נמשיך ככה עד אשר הוא יותר קטן משני בניו.

סיבוכיות פעולות Insert ו- Deletemin בעץ בינארי סדור חלקית:
גובה העץ - h:

$T(n)=\theta(\log n)=\theta(h)$

פעולת Decrease בעץ בינארי סדור חלקית:
- מקטינה את ערכו של הקודקוד הנתון בערך הנתון.

מימוש ע"י בינארי סדור חלקית בעזרת Heap:
- בכל רמה בעץ האינדקסים של T מסודרים סדר עולה משמאל לימין.
- **הילדים של P.T[i] הם: P.T[2i], P.T[2i+1].**

Heapify:
הסבר: פרוצדורה המתקנת סידור של עץ בינארי סדור חלקית, הממומש ע"י ערימה (Heap).
הנחה: 2 תתי העצים תחת הקודקוד שעליו הפעלנו את Heapify כבר עונים על חוקיות עץ בינארי סדור חלקית.
$T(n)=\theta(h)$
גובה העץ - h .

Array to Heap (פועל על עץ בינארי סדור חלקית):
הסבר: הופך מערך לערימה.
$T(n)=\theta(n)$

מיון מערך באמצעות Array to Heap:
הסבר: מבצעים על המערך Array to Heap, מוציאים את האיברים ע"י DeleteMin ומכניסים אותם למערך לפי הסדר.
$T(n)=O(n\cdot\log n)$

טבלאות Hashing

מבני נתונים למילון:
מערך שירות - הסבר ע"פ דוגמא:
נתון מאגר מידע בצורת מערך של כל תעודות הזהות האפשריות מ- 1 עד 999,999,999.
האינדקס של המערך הוא מספר תעודת הזהות.
ערך התא במערך הוא 0 אם הבנאדם מת, ו- 1 אם הוא חי.
במקרה זה כל הפעולות: Member, Insert, Delete הם בעלות סיבוכיות של:
$\theta(1)$

הערה:
החסרון הוא שגודל המערך עלול להיות עצום ביחס למס' האלמנטים הקיימים (בעלי ערך 1) במאגר המידע (מילון).

Hashing Table:
- נעריך את m (גודל המערך שאנו צריכים).
- n = מספר האיברים האמיתי שנכניס.
- **פונקציית Hashing:** ע"פ הערכה זו, נפתח פונקציה שמקשרת בין מפתח (מס' תעודת זהות למשל) לבין תא במערך, באופן חד-חד ערכי.
- אם לא נוכל לפתח פונקציה כזו, נקבל התנגשויות – 2 מפתחות ימופו לאותו תא במערך.

פתרון התנגשות:
1. **טבלה פתוחה:** כל תא במערך מצביע לרשימה מקושרת.
דוגמאות לפונקציית Hashing עבור טבלה פתוחה:
$h(x)=X \bmod m$
$h(x)=\lfloor m\cdot(x\cdot A-\lfloor x\cdot A\rfloor)\rfloor \Leftarrow 0<A<1$

הערות:
- מומלץ להשתמש (ע"פ Knitt) ב-
$A=2\left(\sqrt{5}-1\right)=0.618033$
- מומלץ להשתמש ב- m ראשוני.

דוגמא לפונקציית Hashing פתוחה:
$A(x)=a_0+x(a_1+x(a_2+x(a_3+x\cdot a_4)))$
$P_4=a_4 \bmod M$
$P_3=(a_3+x\cdot P_4) \bmod M$
$P_2=(a_2+x\cdot P_3) \bmod M$
$P_1=(a_1+x\cdot P_2) \bmod M$
$P=(a_0+x\cdot P_1) \bmod M$

$\left.\begin{matrix}x>a_4\\M>x\end{matrix}\right\}a_4 \bmod M=a_4$

2. **טבלה סגורה (שימוש ב- Hashing חוזר):** במקום בפונקציית Hashing בודדה, יש סדרה של פונקציות כאלו. מריצים על הערך פונקציה ראשונה, ואם המקום תפוס, מריצים פונקציה שניה, וכן הלאה עד אשר מגיעים למקום ריק.

הערה: כדי להבטיח הצלחה חשוב ש- m יהיה גדול או שווה ל- n.

דוגמאות לפונקציית Hashing עבור טבלה סגורה:
- Linear Probing:
$h_i(x)=h(x,i)=\left(h'(x)+i\right) \bmod m$
- Quadric Probing:
$h_i(x)=h(x,i)=\left(h'(x)+c_1i+c_2i^2\right) \bmod m$
- Double Hashing:
$h_i(x)=h(x,i)=\left(h'(x)+i\cdot h''(x)\right) \bmod m$

הערה: פונקציה `h ו- `h הן פונקציות Hashing כלשהן.

מציאת X מתוך טבלה סגורה:
מזינים את X לנוסחה ומקבלים את מס' התא המתאים. משווים את הערך בתא ל- X.
אם אין ערך בתא אז X לא קיים במערך.
אם X לא זהה לערך בתא אז מזינים את X שוב לנוסחה (ניסיון 2) ושוב משווים את X לערך שבתא שקיבלנו, וכך הלאה...

סיבוכיות:
- טבלה אידיאלית
הסיבוכיות של: Member, Insert, Delete, וחישוב פונקציה Hashing:
$\theta(1)$
- טבלה פתוחה או סגורה
הסיבוכיות במקרה הגרוע של: Member, Insert, Delete, וחישוב פונקציה Hashing:
$\theta(n)$
אם נשתמש בעצי 2-3 במקום רשימות מקושרות נקבל:
$\theta(\log n)$
במקרה הממוצע הסיבוכיות תהיה:
$\theta(1)$

קוד רישא בינארי

הגדרה:
קוד בינארי: לכל אות בא"ב מתאימה סדרת ספרות בינאריות ייחודית.

רישא (ראשית): סדרת ספרות אחת היא רישא של השניה, אם הסדרה הראשונה מוכלת בסדרה השניה מהראשית (מה- MSB) ימינה.

קוד רישא בינארי: קוד בינארי אשר לכל אות יש סדרת ספרות **שאינה** רישא של סדרה אחרת (אות אחרת).

מטרת קוד רישא בינארי:
להביא את מספר הספרות הנשלחות למינימום ע"י מתן קוד בעל מספר מינימלי של ספרות לאות בעלת השכיחות הגבוהה ביותר.

קוד הופמן:
אלגוריתם לבניית קוד רישא אופטימלי.
- נתחיל מ- n עצים כשבכל עץ יש רק קודקוד אחד.
- בכל שלב מאחדים את 2 העצים עם המשקלים המינימליים = שכיחות מינימלית.
- הקודקוד של העץ החדש הוא הסכום של 2 העצים שאוחדו.
- ממשיכים לחבר כך את כל העצים.

הערה: העץ האופטימלי שנקבל **אינו** יחיד.
מימוש ע"י תור קדימויות:
נכניס את כל האותיות והקדימות שלהן לתור קדימויות ע"י Insert.
נוציא בעזרת DELETEMIN נבנה עץ חדש מסכום הקדימויות שלהם.
נמשיך כך עד אשר נשאר עם עץ אחד.

סיבוכיות קוד הופמן:
הסבר: אלגוריתם הבונה קוד רישא אופטימלי.
$O(n\cdot\log n)$

מציאת עץ פורש מינימלי
(MST - Minimum Spanning Treatment)

הגדרות:
- **עץ לא משורש:** גרף קשיר חסר מעגלים.
- בגרף G לכל צלע יש **משקל W.**
- **עץ פורש מינימלי (T) של גרף קשיר (G):** קומבינציה של צלעות המחברת בין כל הקודקודים ונותנת משקל מינימלי עבור סכום משקלי הצלעות.
- **חתך בגרף:** חלוקה של קודקודי הגרף ל- 2 קבוצות (גרפים) זרות (לפעמים לא נוכל לבצע זאת).
- **צלע חוצה:** צלע חוצה את החתך אם כל קודקוד שלה נמצא בקבוצות זרות.
- **חתך מכבד קבוצה A (קבוצת צלעות כלשהי)** אם אף צלע מקבוצה A אינה חוצה את החתך.
- **צלע קלה:** מבין כל הצלעות שחוצות את החתך, זו הצלע עם המשקל המינימלי.
- **צלע בטוחה:** צלע שאם נוסיף אותה לקבוצת צלעות כלשהי, הקבוצה תהפוך לתת-קבוצה של צלעות בעץ פורש מינימלי כשלהו.

משפט הצלעות הבטוחות - תנאי מספיק לבטיחות:
- קבוצה A היא קבוצה של צלעות, שמוכלת בעץ פורש מינימלי כלשהו.
- קיים חתך המכבד את A (אף צלע מ- A לא חוצה את החתך).
- אם נוסיף צלע קלה (ז"א צלע החוצה את החתך אך לא שייכת ל- A) אז היא גם בטוחה עבור A.

אלגוריתמים למציאת עץ פורש מינ' (MST):
אלגוריתם ג'נרי:
- **אתחול:**
A - קבוצת צלעות ריקה.
W - קבוצת קודקודים בעלת קודקוד אחד בלבד (מאילו שבגרף).
- **כל עוד יש צלע בטוחה עבור A נוסיף אותה ל- A, ואז A יהיה העץ הפורש המינימלי.**

הערה: הוכחת נכונות - באינדוקציה על צעדי האלגוריתם.

האלגוריתם של Prim:
- **אתחול:**
A - קבוצת צלעות ריקה.
W - קבוצת קודקודים בעלת קודקוד אחד בלבד (מאילו שבגרף).
- **נמצא צלע (x,y) בעלת משקל מינימלי** (קלה), כך ש- x נמצא ב- W ו- y לא נמצא ב- W.
- נוסיף את הצלע (x,y) ל- A, ונוסיף את y ל- W.
- נמשיך כך עד שכל הקודקודים יהיו מחוברים במסלול כלשהו (לא יישארו צלעות קלות).
- Prim הוא מקרה פרטי של האלגוריתם הג'נרי ולכן הוא מוצא עץ פורש מינימלי.

סיבוכיות:
- איתחולים
$O(n)$
- DeleteMin
יש 1-n פעולות כאלו, שכל אחת בעלת סיבוכיות:
$O(\log n)$
- Decrease
יש m פעולות כאלו, שכל אחת בעלת סיבוכיות:
$O(\log n)$
- סה"כ
$O(m\cdot\log n)$

האלגוריתם של Kruskal:
- **אתחול:**
A - קבוצת צלעות ריקה.
F - קבוצה של n עצים, בכל עץ קודקוד אחד.
- נבחר צלע (x,y) בעלת משקל מינימלי (קלה), המחברת בין 2 עצים שונים ב- F.
הערה: יש לשים לב שלא מחברים 2 קודקודים אשר כבר מחוברים במסלול כלשהו (באותו עץ).
- נמשיך כך כל עוד F מכיל יותר מעץ אחד.
- Kruskal הוא מקרה פרטי של האלגוריתם הג'נרי ולכן הוא מוצא עץ פורש מינימלי.

סיבוכיות Kruskal למציאת עץ פרוש מינימלי:
- n פעולות Make_Set (n - מספר הקודקודים).
- מיון m צלעות (מבוסס השוואות):
$O(m\cdot\log m)$
- 2m פעולות של Find_Set.
- n-1 פעולות Union.
- סה"כ:
$n-1\le m\le n^2\ ;\ n=|V|\ ;\ m=|E|$
$O(m+n\cdot\log n)+\underbrace{O(m\cdot\log m)}_{SORT\ EDGES}$
$=O(m\cdot\log m)$

שיפור זמן הריצה :
נתון כי משקלי הקשתות שלמים חיוביים בתחום 1...V
נמיין קשתות בעזרת Counting Sort .
נקבל סיבוכיות:
$T(n)=O(n)+\underbrace{O(m\log^* n)}_{SORT\ EDGES}+O(m)=O(m\log^* n)$

ADT - מבנה נתונים - קבוצות זרות:

הגדרה:
- מבנה מתמטי הכולל קבוצת בסיס המכילה את כל קודקודי הגרף.
- קיימות קבוצות קודקודים - כל קבוצת כזו מכילה מספר קודקודים מהגרף ואחד מהם הוא נציג הקבוצה.
- **הקודקודים ממוספרים מ- 1 עד n.**

הערה: יתכן מצב שקודקוד לא שייך לאף קבוצה.

מערך של רשימות:
- כל הנתונים לגבי קבוצות הקודקודים מאוחסנים במערך Sets[].
- אינדקס תאי המערך מייצג קודקודים בגרף.
- לכל תא יש 2 שדות:
- שדה המכיל את נציג קבוצת הקודקודים.
- שדה המכיל את הקודקוד הבא אחריו בקבוצה.

דוגמא:
קבוצות קודקודים: {**4**, 8}, {**6**, 2}, {**3**, 1, 7}
מערך Sets:

Index	1	2	3	4	5	6	7	8
Rep	3	6	3	4	-	6	3	4
Next	7	-	1	8	-	2	-	-

פעולות על מערך של רשימות:
1. **Make_Set(x)** - מאתחל קבוצה חדשה ובה האיבר x (הוא היחיד ולכן הוא הנציג).
הנחה: x לא נמצא באף קבוצה אחרת.
סיבוכיות:
$\theta(1)$
2. **Find_Set(x)** - מחזיר את הנציג של הקבוצה אליה שייך x. מחזיר שקר אם x לא שייך לאף קבוצה.
סיבוכיות:
$\theta(1)$
3. **Uniun(x,y)** - מאחד בין 2 קבוצות ובסיום יהיה נציג יחיד לקבוצה המאוחדת:
- נבדוק שהנציגים x,y שונים.
- נעבור על כל מי שבקבוצה של y (נתחיל בנציג), ונשנה את הנציג להיות הנציג של x.
- כאשר נגיע לאיבר האחרון ב- y נשנה אותו כך שיצביע לאיבר השני ב- x.
- את האיבר הראשון ב- x נשנה כך שיצביע לאיבר הראשון ב- y.
סיבוכיות:
$O(n)$
הערה:
ע"מ שתמיד נוכל להגדיר את y כקבוצה הקטנה מבין השתיים, נוסיף **לכל ערך** שדה נוסף = Size גודל הקבוצה. **לאחר איחוד, נעדכן רק את שדה ה- Size רק אצל הנציג.**

יער של עצים:
- כל קבוצת קודקודים מתאימה לעץ.
- נציג הקבוצה הוא שורש העץ.
- כל הנתונים לגבי קבוצות הקודקודים מאוחסנים במערך של מצביעים.
- אינדקס תאי המערך מייצג קודקודים בגרף.
- **לכל תא יש 3 שדות:**
- Parent - שדה המכיל את ההורה של הקודקוד.
- Size - שדה גודל העץ (מספר קודקודים).
- Element - ערך הקודקוד עצמו.

פעולות על יער של עצים:
1. **Make_Set(x)** - מייצר עץ חדש בעל קודקוד אחד (שורש) ושם מצביע ממיקום x במערך לקודקוד הזה.
סיבוכיות:
$\theta(1)$
2. **Find_Set(x)** - הולך לקודקוד x, עולה במעלה העץ עד שמגיע לשורש, ומחזיר אותו.
סיבוכיות:
$\theta(h)$
3. **Uniun(x,y)** - מאחד בין 2 עצים x,y.
- מוצא את השורשים של x,y.
- מוסיף מצביע מהשורש של העץ **הנמוך לשורש של העץ הגבוה.** מחברים את העץ הנמוך לעץ הגבוה ע"מ שהגובה של העץ החדש לא יגדל.

סיבוכיות:
$O(\log n)$

הערות:
- מכיוון שהסיבוכיות תלויה בגובה העץ, נרצה לראות מה הגובה המקסימלי שיכול להתקבל מ- n קודקודים.
- גובה מקסימלי זה יתקבל כאשר נחבר לעץ אחד, עץ אחר, בעל אותו גובה ובעל מספר מינימלי של איברים.
- במצב זה נקבל עץ חדש בעל פחות פי 2 קודקודים.
- לכן, כאשר יהיו לנו n קודקודים, נקבל עץ בגובה מקסימלי של log(n).

שיפור - Path Compression:
- "אם כבר" עולים במעלה העץ, אז נדאג שכל קודקוד שעברנו דרכו יצביע ישירות לשורש. ניתן לבצע זאת ע"י רקורסיה - פעם אחת עולים עד שמוצאים את השורש, ואז יש לנו מצביע אליו (תנאי עצירה), ובחזרה הרקורסיה מחברת ישירות בין כל קודקוד לשורש.
- הסיבוכיות של Path Compression היא:
$O(m\cdot\log^* n)$

$\log^* n$ הוא מספר הפעמים שצריך לבצע $\log(\log\cdots(\log(n))))$ ע"מ שנקבל 1.
$\log^*(n)<\log(n)$

תכנות דינמי - Dynamic Programming

הגדרות:
בעיית אופטימיזציה:
- נתונה בעיה שיש לה הרבה פתרונות אפשריים.
- לכל פתרון יש מחיר, ומחפשים פתרון עם מחיר "הכי טוב".
- לפתרון זה נקרא "פתרון אופטימלי".

תכנון דינמי:
גישה יעילה לפתרון בעיית אופטימיזציה.

פתרונות לבעיית אופטימיזציה:
1. **פתרון נאיבי.**
2. **רקורסיה רגילה.**
3. **רקורסיה + Memoize** - רקורסיה שלא חוזרת על אותו חישוב פעמיים. השינויים שמתבצעים במעבר מרקורסיה רגילה לרקורסיה + Memoize:
- הוספת מערך ששומר את ערכי הרקורסיה ומאותחל בד"כ ל- ∞.
- הוספת תנאי עצירה אשר בודק אם כבר חישבנו את הרקורסיה הנוכחית.
- אם כן – מחזיר את הערך השמור במערך.
- אחרת – מבצע את הרקורסיה הרגילה ושומר את הערך המתקבל במערך.
4. **Buttom Up** - פתרון בצורת לולאות אשר משתמש ב- Memoize ולכן לא חוזר על אותו חישוב פעמיים.

הערה:
סיבוכיות רקורסיה + Memoize וסיבוכיות Buttom Up תמיד יהיו אותו הדבר באלגוריתמים שאנו מדברים עליהם.

דוגמאות ספציפיות:
1. **מציאת תת-סדרה רציפה שסכום איבריה גדול ביותר:**
P(i) - תת סדרה רציפה בעלת סכום מקסימלי המסתיימת באיבר i.
A[ ] - מערך הקלט המכיל את כל איברי הסדרה.
$P(1)=A(1)$
$P(i)=\max\{A(i),A(i)+P(i-1)\}$

סיבוכיות:
סיבוכיות פתרון בעזרת רקורסיה רגילה

Recursive Part: $T_P[i]=T_P[i-1]+c_1\Rightarrow T_P[i]=\theta(i)$
Algorithem : $T[n]=\theta(T_P[1]+T_P[2]+\ldots+T_P[n])=$
$=\theta\left(\sum_{i=1}^{n}i\right)=\theta(n^2)$

סיבוכיות רקורסיה + Memoize

Recursive Part: c_1
Whole Array : $\sum_{i=1}^{n}c_1=\theta(n)$
Algorithem : $T[n]=\underbrace{\theta(n)}_{INITIALIZE}+\underbrace{\theta(n)}_{LOOP}+\underbrace{\theta(n)}_{WHOLE\ ARRAY}=\theta(n)$

סיבוכיות Buttom Up

Single Loop: c_1
All Loops : $\sum_{i=1}^{n}c_1=\theta(n)$
Algorithem : $T[n]=\underbrace{\theta(n)}_{INITIALIZE}+\underbrace{\theta(n)}_{ALL\ LOOPS}=\theta(n)$

הערה:
ע"מ למצוא את האיבר הראשון בסדרה המקסימלית P(i), נתחיל ללכת אחורה, מהאינדקס i במערך שבו נשמרו תתי הסדרות המקסימליות, עד אשר נגיע למצב שבו הערך בתא של המערך שווה לערך של איבר בעל אותו אינדקס במערך המקורי.

2. הכפלת מטריצות
הבעיה:
רוצים לחשב את תוצאת מכפלת n מטריצות, עם מספר מינימלי של פעולות.

הפתרון:
m(i,j) - מספר הכפלים הסקלרים המינימלי שנדרש לחישוב מכפלה של Ai…Aj מטריצות.

$$m(i,j)=$$

$$P(i-1)\left\{\underbrace{\left(m(i,k)\right)}_{P(k)}\cdot\underbrace{\left(m(k+1,j)\right)}_{P(j)}\right\}P(k)$$

$$m(i,j)=\begin{cases}0 & ;\ i=j\\ \min\limits_{i\le k<j}\{m(i,k)+m(k+1,j)+P_{i-1}\cdot P_k\cdot P_j\} & ;\ i<j\end{cases}$$

- מחלקים את המטריצות ל- 2 חלקים.
- כל חלק מחולק שוב ל- 2 חלקים, וכן הלאה.
- **k** קובע היכן תהיה השבירה ל- 2 חלקים.

הסבר מעמיק:
יש מטריצה m(i,j) בעלת n^2 תאים (אברים).
כל פעם שנרצה לחשב את m(i,j) המינימלי נצטרך לחשב אותו עבור K נקודות שבירה שונות ומתוך ערכים אלה לבחור את המינימאלי.
מספר נקודות השבירה הוא **מקסימום** (n-1).
לכן הסיבוכיות היא:
$T(n)=n^2(n-1)=O(n^3)$

סיבוכיות:
סיבוכיות פתרון בעזרת רקורסיה רגילה
$T(n)\ge 1+\sum_{k=1}^{n-1}(T(k)+T(n-k)+1)$
$T(n)\ge 2^{n-1}=\Omega(2^n)$
(הוכחה באינדוקציה על n)

סיבוכיות דינמית (רקורסיה+Meomize או Btm up)
$T(n)=O(n^3)$

הערות:
- הלולאה בנויה על ההפרשים בין i,j. קודם תחשב את כל הזוגות של i-j=1, אח"כ את כל הזוגות שההפרש הוא 2 וכן הלאה.
- אם רוצים **לבצע** את ההכפלה באופן היעיל ביותר שמצאנו, נבנה מערך S דו-מימדי שאוגר את כל ה"שבירות" האופטימליות (ערכים של k).

3. בעיית All Pairs Shortest Path:
הבעיה:
- נתונה קבוצת קודקודים וקבוצת צלעות עם משקלים.
- מוצאים את המסלול בעל המשקל המינימלי בין כל 2 קודקודים.
- נתונה מטריצה W דו-מימדית שהמאחסת את כל משקלי הצלעות:

$$W_{i,j}=\begin{cases}0 & ;\ i=j\\ weight & ;\ i\ne j\ \&\ (i\to j)\in E\\ \infty & ;\ i\ne j\ \&\ (i\to j)\notin E\end{cases}$$

- **הנחה:** מחיר כל מעגל בגרף תמיד גדול או שווה ל- 0.
- $d_{ij}^{(k)}$ - אורך מסלול אופטימלי מבין אילו שמתחילים ב- i, ומסתיימים ב- j. מותר להם להשתמש **רק** בקודקודי ביניים הבאים: $1,2,\ldots,k$
- הנוסחה הרקורסיבית:

$$d_{ij}^{(k)}=\begin{cases}W_{ij} & ;\ k=0\\ \min(\underbrace{d_{ij}^{(k-1)}}_{(1)},\underbrace{d_{ik}^{(k-1)}+d_{kj}^{(k-1)}}_{(2)}) & ;\ k>0\end{cases}$$

בוחרים את המסלול מקודקוד i ל- j, בעל המשקל המינימלי מבין 2 מסלולים מינימליים אפשריים:
(1) הגעה ל- j ללא מעבר דרך קודקוד k.
(2) הגעה ל- j כאשר חייבים לעבור דרך קודקוד k.

- הדפסת המסלול האופטימלי:
כל פעם שנבחר מינימום מבין 2 מסלולים מ i ל j (אחד עם k ואחד בלעדיו) נכניס את הקודקוד **שלפני** j למערך דו מימדי (מערך חדש).
אם נרצה להדפיס את המסלול האופטימלי, נוכל לבנות שגרה רקורסיבית שתבצע זאת.

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)
$T(n)=\theta(n^3)$

4. K - Partition:
הבעיה:
- נתון מערך A, ומספר שלם k.
- מחלקים את המערך ל- k קבוצות.
- סוכמים את האיברים בכל קבוצה ומוצאים את **הסכום המקסימלי** מבין סכומי כל הקבוצות.
- מחלקים את המערך שוב ל- k קבוצות בצורה שונה, ומוצאים שוב את הסכום המקסימלי.
- ממשיכים כך על כל האפשרויות.
- המטרה - למצוא את **הסכום המינימלי** מבין כל הסכומים המקסימליים שמצאנו (מחלוקות k השונות).

הפתרון:
M(j,P) = הסכום המינימלי המתקבל מחלוקה של A[1] עד A[j] ל- P קבוצות.

$$M(j,p)=\min_{1\le i\le j-1}\left(\max\left(M(i,p-1),\sum_{r=i+1}^{j}A[r]\right)\right)$$

הסבר:
- המערך מחולק כל איטרציה ל- 2 קבוצות (ע"פ i), כאשר בקבוצה האחת תהיה חלוקה אחת (תת-קבוצה אחת) בלבד, ואת הקבוצה השניה נחלק p-1 תתי-קבוצות (באותו אופן שבו חילקנו את המערך).
- בכל איטרציה נמצא את המקסימום מבין כל סכומי תתי-הקבוצות.
- מכל האיטרציות נבחר את הסכום המינימלי.

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)
$T(n)=\theta(k\cdot n^2)$

5. בעיית Knapsack 1-0:
הבעיה:
- גנב פורץ לחנות שבה n פריטים.
- פריט i שוקל Vi שקלים ושוקל wi קילוגרמם.
- הגנב יכול לסחוב עד W קילוגרמים.
- אילו פריטים על הגנב לבחור כך שיקבל שווי מקסימלי?

הפתרון:
C(i,w) = בוחרים מבין הפריטים מ- 1 עד i, מספר פריטים, כך שנקבל משקל קטן או שווה ל- w, וערך פריטים מקסימלי.

$$C(i,w)=$$

$$\begin{cases}0 & ;\ i=0\ or\ w=0\\ C(i-1,w) & ;\ w_i>w\\ \max\left(v_i+C(i-1,w-w_i),C(i-1,w)\right) & ;\ i>0\ \&\ w\ge w_i\end{cases}$$

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)
$\theta(nW)$

6. בעיית השילוש האופטימאלי:
הבעיה:
- נתון פוליגון בעל N+1 קודקודים P=(v0,v1,vn)
- נתונה פונקציית המשקל של המשולשים.
- נרצה למצוא את השילוש האופטימאלי.

הפתרון:
- נוסחת הרקורסיה נתונה בצורה הבאה:

$$t(i,j)=$$

$$\begin{cases}0 & ;\ i=j\\ \min\limits_{i\le k<j}(t(i,k)+t(k+1,j)+w(\Delta V_{i-1}V_kV_j)) & ;\ i<j\end{cases}$$

בסיום ההרצה של האלג' נקבל את הערכים בתוך מערך t(i,j) ואת ה K שנתן את הערך המינימאלי בתוך מערך S[ ].

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)
$T(n)=O(n^3)\leftarrow Time$
$O(n^2)\leftarrow Memory$

7. בעיית הסוכן הנוסע TSP:
הבעיה:
- נתון גרף לא מכוון שלכל צלע בו יש מחיר חיובי C(u,v) .
- צריך למצוא מסלול סגור המבקר בכל קודקוד בדיוק פעם אחת וסכום מחירי הצלעות המשותפות מינימאלי.
- אילוץ סביר:
$C(x,y)\le C(x,z)+C(z,y)$

פתרון:
- נבחר קודקוד r, ונגדיר אותו לשורש העץ.
- נבנה עץ פורש מינימלי - T, בעזרת אלג' Prim .
- נמצא מהלך מלא w (מתחיל מהשורש ועובר על כל הקודקודים) על העץ הפורש מינימלי- T.
- נהפוך את w למסלול סגור - H (שאינו בהכרח מינימלי), שעובר דרך כל קודקוד בדיוק פעם אחת. נבצע זאת ע"ב עקרון PostOrder.
- ניתן את H כפלט לאלגוריתם.

טענה:
מחיר מסלול H לכל היותר פי 2 ממחירו של מסלול סגור אופטימאלי - H*.
הוכחה:
H* - מסלול סגור עם מחיר מינימלי (יכול לכלול צלעות אחרות מ- T).
C(T) - סכום מחירי על הצלעות בעץ T הפרוש מינימלית.
C(w) - מחיר המהלך המלא (בעץ פורש מינימלי) שעובר על כל צלע בדיוק פעמיים.
C(H) - מחיר מסלול סגור בעץ פורש .
C(H*) - מחיר מסלול סגור בעץ פורש מינימלי.

$$\left.\begin{array}{l}C(w)=2\cdot C(T)\\ C(H^*)>C(T)\\ C(H)\le C(w)\end{array}\right\}C(H)<2\cdot C(H^*)$$

הערה:
C(T) הוא עץ פורש מינימלי (חסר מעגלים).
מכיוון ש C(H*) הוא מסלול סגור (מעגלי) יש לו לפחות צלע אחת יותר מאשר העץ הפורש המינימלי וכתוצאה מכך מחירו גדול יותר מ C(T) כלומר:
$C(H^*)>C(T)$
מהלך מלא: מעבר על כל הקודקודים בעץ הפורש המינימלי התחלה וסיום בשורש ולכן נעבור על כל צלע בדיוק פעמיים.

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)
$T(n)=\sum_{SIZE=0}^{n-2}\binom{n-1}{SIZE}\cdot(n-1)\cdot SIZE$
$\approx(n-1)\cdot 2^{n-1}=\theta(n^2\cdot 2^n)$

8. הזמנה לא-עוקבת בהיררכיה:
הבעיה:
- נתונה חברה שבה לכל עובדת יש מנהלת, למעט המנהלת הראשית.
- רוצים להזמין מספר גדול ככל האפשר של עובדות למסיבה, כך **שלא יוזמנו עובדת ומנהלתה הישירה.**

פתרון:
- מציגים את ההיררכיה בחברה, כעץ.
- עבור כל צומת j מגדירים 3 ערכים:
- Y(j): המספר המקסימלי של עובדות שאפשר להזמין מתת-העץ של j, כאשר **כן** מזמינים את j.
- N(j): המספר המקסימלי של עובדות שאפשר להזמין מתת-העץ של j, כאשר **לא** מזמינים את j.
- P(j): המספר המקסימלי של עובדות שאפשר להזמין מתת העץ של j, כאשר מותר להזמין את j, וגם מותר לא להזמין את j (מקסימום בין Y ו- N).
- עבור כל **עלה** j מתקיים:
$P(j)=1\ ;\ N(j)=0\ ;\ Y(j)=1$
- עבור כל צומת k (כאשר ילדיו: k1, k2, k3) מתקיים:
$Y(k)=1+N(k_1)+N(k_2)+\ldots+N(k_m)$
$N(k)=P(k_1)+P(k_2)+\ldots+P(k_m)$
$P(k)=\max\left(\ Y(k)\ ,\ N(k)\ \right)$
- הפתרון הוא רקורסיבי ומתחיל מלמטה למעלה.
- ע"מ להדפיס את ההרכב האופטימלי עצמו נבצע Backtracing:
- מתחילים בשורש.
- אם P(j) = Y(j) אז j הוזמן, אחרת j לא הוזמן.
- אם j עלה אז מדפיסים אותו.
- אם j הוזמן אז מדפיסים את j **ומדלגים** על הצומת הבאה (אם הצומת הבאה היא עלה אז סיימנו).
- אם j לא הוזמן אז עוברים לצומת הבאה ובודקים שוב האם j עלה (ואז מדפיסים אותו ומסיימים), ואם j לא עלה אז בודקים את התנאי P(j) = Y(j) וכן הלאה.

סיבוכיות דינמית:
(רקורסיה + Meomize או Btm up)

סיבוכיות מציאת ההרכב האופטימלי
$T(n)=\sum_{j=1}^{n}(c_1\cdot m(j)+c_2)=\underbrace{\sum_{j=1}^{n}c_1\cdot m(j)}_{\text{מספר הילדים}}+\sum_{j=1}^{n}c_2=\theta(n)$
n - מספר הצמתים.
m(j) - מספר הילדים עבור קודקוד j.

סיבוכיות הדפסת ההרכב האופטימלי
$T(n)=\theta(n)$

9. בעיית מציאת תת סדרה עולה באורך מקסימלי:
הבעיה:
נתונה סדרת מספרים טבעיים שונים זה מזה באורך n .
צריך למצוא את תת הסדרה העולה הארוכה ביותר ולמצוא את האורך שלה (הסדרה לא חייבת להיות רצופה).
הגדרה:
f(i) הוא אורך תת הסדרה העולה המקסימלית שמתיימת באיבר A[I].

$$f(i)=\begin{cases}1 & ;\ \forall_{1\le j\le i}A[j]\ge A[i+1]\\ \max(f(j)\,|\,1\le j\le i, A[j]<A[i+1])+1 & ;\ otherwise\end{cases}$$

שומר את האינדקס של האיבר שלפני.

$$prev(i+1)=\begin{cases}0 & ;\ \forall_{1\le j\le i}A[j]\ge A[i+1]\\ \arg\max(f(j)\,|\,1\le j\le i, A[j]<A[i+1]) & ;\ otherwise\end{cases}$$

סיבוכיות:
$T(n)\le\sum_{i=1}^{n}C_2 i=C_2\frac{n(n-1)}{2}$
$T(n)\ge\sum_{i=1}^{n}C_1 i=C_1\frac{n(n-1)}{2}$
$T(n)=\theta(n^2)$

זרימה ברשתות

הגדרות:
רשת זרימה:
- נתון **גרף מכוון** G=(V,E) עם משקלים.
- נתון קודקוד מוצא s.
- נתון קודקוד יעד t.
- לכל צלע (u,v) יש כיוון והוא מ- u ל- v.
- לכל צלע (u,v) יש משקל (קיבול) c(u,v)≥0.
- אם אין צלע בין u ל- v אז הקיבול הוא 0.

הערה: קיבול הוא מספר שלם תמיד.

פונקציית זרימה:
- מה שמוזרם בין 2 קודקודים u,v ברשת. $f(u,v)$

פונקציית הזרימה מקיימת את התנאים הבאים:
- אילוץ הקיבול
על אף צלע לא מוזרם יותר מהקיבול שלה.
$f(u,v)\le c(u,v)$
- אנטי-סימטריה
$f(u,v)=-f(v,u)$
- שימור הזרימה
לכל קודקוד השונה מ- s ו- t כמות הזרימה היוצאת שווה לכמות הזרימה הנכנסת.

ערך הזרימה:
הזרימה שיוצאת מ- s היא הזרימה שנכנסת ל- t והיא שווה ל- $|f|$.

בעיית הזרימה המקסימלית:

הבעייה:
מהו הקצב המקסימלי שבו ניתן להזרים חומר ברשת מ- s ל- t בלי לחרוג מקיבול הצלעות.אם אין צלע בין u ל- v אז הקיבול הוא 0.

הגדרות:
- צלע רוויה
צלע (u,v) שבה עוברת זרימה מירבית.
$f(u,v)=c(u,v)$
- צלע משפרת
צלע (u,v) שאפשר להזרים לאורכה עוד זרימה.
$f(u,v)<c(u,v)$
- מסלול משפר
מסלול מ- s ל- t שמורכב כולו מצלעות משפרות.
- קיבול שיורי c_f
כמות הזרימה שעדיין אפשר להזרים לאורך צלע.
$c_f(u,v)=c(u,v)-f(u,v)$
- קיבול שיורי של מסלול p (צוואר הבקבוק)
מבין הצלעות במסלול p, הצלע עם הקיבול השיורי הנמוך ביותר מהווה את הקיבול השיורי של מסלול p.
$c_f(p)=\min c_f(u,v)\Leftarrow(u,v)\in p$
- הגרף השיורי G_f
הגרף G_f - מכיל את כל הצלעות המשפרות של G.
משקלי הצלעות הם הקיבולים השיוריים.
כל מסלול בגרף השיורי הוא מסלול משפר.
- חתך ברשת
חלוקה של קודקודי הרשת ל- 2 קבוצות זרות S, T, כאשר:
$s\in S\ ;\ t\in T$
- קיבול חתך
סכום הקיבולים (משקל צלעות) החוצים את החתך.
יסומן ב:
$c(S,T)$
- זרימה בחתך
סכום הזרימות החוצות את החתך.
יסומן ב:
$f(S,T)$
הערה: הגרף המקורי הוא מכוון, אך במסלול משפר **מותר** ללכת "נגד הכיוון".

למת החתך:
ערך הזרימה מוגדר כמה שנכנס לרשת ומה שיוצא מהרשת. לכן הוא גם מה שעובר בין חתך S לחתך T.
$f(S,T)=|f|$

משפט Min Cut , Max Flow:
התנאים הבאים שקולים:
- ב- G זורמת זרימה מירבית.
- בגרף השיורי G_f אין מסלול מ- s ל- t.
- קיים חתך (S,T) שעבורו:
$c(S,T)=|f|$

החתך המינימלי בגרף מגביל את הזרימה המקסימלית בגרף.

אלגוריתם פורד-פלקרסון - פתרון בעזרת הזרימה המקסימלית:
- אתחול:
בכל צלע בגרף G תסומן זרימה 0.
הגרף השיורי G_f = לגרף המקורי G אך ליד כל צלע רשומה רק הקיבולת (ללא הזרימה, כלומר, ללא 0).
- כל עוד יש מסלול מ- s ל- t בגרף G_f:
- נמצא מסלול p בגרף ונחשב את קיבולו השיורי (צוואר הבקבוק).
- נבצע את הזרימה ב- G לאורך מסלול p למקסימום האפשרי ע"פ הקיבול השיורי שמצאנו.
- לכל צלע (u,v) במסלול p נעדכן את הגרף השיורי.

מסקנה:
- אל אלגוריתם פורד-פלקרסון עוצר - אז הזרימה |f| ברשת היא מירבית.
- האלגוריתם עוצר כאשר אין יותר מסלולים משפרים ב- G_f.
- לפי משפט Min Cut, Max Flow הזרימה ברשת מירבית.

סיבוכיות הפתרון:
F - ערך הזרימה המירבית.
m - מספר הצלעות.
יש מקסימום F איטרציות, מכיוון שבמקרה הגרוע נוכל להוסיף 1 בכל איטרציה.

ע"פ משפט Edmonds Karp: אם בוחרים מסלול משפר בעזרת BFS אז מספר האיטרציות הוא:
$O(n\cdot m)$
לכן הסיבוכיות הכוללת היא:
$O(\min(n\cdot m^2, m\cdot F))$

זיווג מירבי בגרף דו-צדדי:

גרף דו-צדדי:
- גרף שקודקודיו מחולקים ל- 2 קבוצות זרות L, R.
- צלעותיו מחברות רק בין קודקודים השייכים לקבוצות שונות.

זיווג (Marching):
- לכל קודקוד יש מקסימום צלע אחת.
- **זיווג מירבי** - כאשר בגרף יש מקסימום זיווגים.
- **זיווג מושלם** - לכל קודקוד, בכל צד יש זיווד: $|M|=|L|=|R|$

M - מספר הצלעות.

אלגוריתם למציאת זיווג מירבי:

פתרון ע"ב רדוקציה (לבעיית הזרימה המירבית):
- נוסיף 2 קודקודים חדשים לגרף - s,t: אחד משמאל ל- L ואחד מימין ל- R.
- נחבר את s לכל הקודקודים ב- L.
- נחבר את t לכל הקודקודים ב- R.
- נגדר את כל כיווני הצלעות מ- s ל- t.
- הקיבולים של כל הצלעות הן 1.
- נמצא זרימה מירבית ע"י אלגוריתם פורד-פלקרסון.

פסאודו קוד – אלגוריתמי חיפוש

Naive Search:

עובר על כל האיברים אחד אחד.

```
NaiveSearch(int n, sorted A[1..n], int x)
{ found ← FALSE; i ← 1;
  while( found == FALSE  & i ≤ n)
  {if (A[i] == x) found ← TRUE
    else i ← i+1
  }
if ( found == TRUE) output i
else output ("not found")
}
```

Binary Search:

נתון מערך ממוין ולכן הוא חוצה את המערך כך פעם ל 2 ובודק באיזה חלק המספר נמצא.

```
BinarySearch(int n, sorted A[1..n], int x)
{min ← 1; max ← n; found ← FALSE;
 while( found == FALSE & min ≤ max)
  {mid ← ⌊(min+max)/2⌋
  if (A[mid] == x) found ← TRUE
  else {if (x < A[mid]) max ← mid − 1;
        else min ← mid + 1;
       }
  }
if ( found == TRUE) output(mid);
else output("not found")
}
```

פסאודו קוד – אלגוריתמי מיון

Insertion Sort:

עובר על האיברים משמאל לימין ומשווה כל איבר עם קודמו אם הוא מוצא איבר שקטן מקודמו הוא מעביר אותו אחורה ומשווה אותו עם איבר אחד לפניו וכך הלאה עד אשר הוא מוצא איבר שהוא גדול ממנו ושם הוא מכניס אותו.

```
InsertionSort(int n, A[1..n])
{ for j = 2 to n
  {new_num ← A[j];
  i ← j − 1;
  while(i > 0 & new_num < A[i])
    {A[i + 1] ← A[i];
     i ← i − 1;
    }
  A[i + 1] ← new_num;
  }
}
```

Merge Sort:

ממיין את מערך A מאיבר p עד איבר r בצורה רקורסיבית (משתמש בעקרון הפרד ומשול). כל פעם חוצה את המערך לחצי עד אשר נשאר עם איבר אחד.

ואז Merge ממזגת את האיברים בצורה ממויינת.

```
MergeSort(int p, r, A[])
{if (r ≤ p) return;
 else q ← ⌊(p+r)/2⌋;
 MergeSort(A, p, q);
 MergeSort(A, q+1, r);
 Merge(A, p, q, r);
}
```

Merge:

האלגוריתם יודע לאחד 2 מערכים ממויינים הראשון מ-p ל-q והשני מ-q ל-r למערך אחד ממויין.

```
Merge(A, p, q, r)
{i ← p
 j ← q+1
 k ← 1
 while ((i ≤ q)&(j ≤ r))
  {if A[i] ≤ A[j] then
    {B[k] ← A[i]
     i ← i+1
     k ← k+1
    }
   else
    {B[k] ← A[j]
     j ← j+1
     k ← k+1
    }
  }
 while (i ≤ q)
  {B[k] ← A[i]
   i ← i+1
   k ← k+1
  }
 while (j ≤ r)
  {B[k] ← A[j]
   j ← j+1
   k ← k+1
  }
 for i = p to r
  A[i] ← B[i − p+1]
}
```

Counting Sort:

A הוא מערך הקלט ו B הוא מערך הפלט שניהם בעלי n איברים וכל איבר קטן או שווה ל k.

האלג' בונה מערך שכיחויות ומסב אוו למערך שכיחות מצטברת.

אז מכניס כל איבר מ A ל B למקום שהוא השכיחות המצטברת שלו.

```
CountingSort(A[], B[], int n, k) // A[i] ≤ k //
{ for val = 1 to k
    C[val] ← 0;
  for p = 1 to n
    C[A[p]] ← C[A[p]] + 1;
  for i = 2 to k
    C[i] ← C[i] + C[i − 1];
  for j = n downto 1
  {B[C[A[j]]] ← A[j];
   C[A[j]] ← C[A[j]] − 1;
  }
}
```

Radix Sort:

יש לו את אותן ההנחות כמו ל counting sort ובנוסף ידוע שלכל המספרים יש את אותו מספר הספרות (d).

האלג' ממיין ספרה אחר ספרה מה LSB עד ל MSB.

```
RadixSort(A[ ], int n, d)
{ for i = 1 to d
  {use a stable sorting a lg orithm
   to sort A[ ] on digit i
  }
}
```

BFS - Breadth First Search (ע"י תור):
- האלג' מקבל גרף G.
- האלג' יודע להדפיס את הקודקודים לפי המרחק שלהם מהנעץ.
- v: הקודקוד ההתחלתי של החיפוש.

```
BFS(G, v)  // G = graph, v = a   ll vertexes
{ for every x ∈ v  visited[x] ← FALSE;
  pr int(v); visited[v] ← TRUE;
  ENQUEUE(v, Q);
  while(not EMPTY(Q))
   {x ← DEQUEUE(Q);
    for every vertex adjacent to x
     {if (not visited[y])
       {pr int(y);
        visited[y] ← TRUE;
        ENQUEUE(y, Q);
       }
     }
   }
}
```

DFS (מימוש רקורסיבי ולא ע"י מחסנית) :

האלג' מקבל גרף G וקבוצת קודקודים V.

```
DFS(G,v)
{for i ← to n  visited[i] ← FALSE
DFS-Recursive(G,v)
}
```

DFS-Recursive:

```
DFS-Recursive(G,v)
{print(x)
 visited[x] ← TRUE
 for every vertex y such that (x,y) ∈ E
  {if (not visited[y])
      DFS-Recursive(G,v)
  }
}
```

פסאודו קוד - מימוש מחסנית ע"י מערך

מחסנית S היא רשומה עם 2 שדות:
- Elements[ ]: מערך של אלמנטים בגודל MAXSIZE.
- Top: האינדקס של האיבר העליון במחסנית.

Empty:

```
Empty(S)
{if (S.top = 0) return(TRUE);
 else return(FALSE);
}
```

Top:

```
Top(S)
{if (!EMPTY(S)) return(S.elements[S.top]);
 else return("stack is empty");
}
```

Pop:

```
Pop(S)
{if (!EMPTY)
  S.top ← S.(top − 1);
  return(S.elements[S.(top + 1)]);
else return("stack is empty")
}
```

Push:

```
Push(S)
{if (S.top < MAXSIZE)
  {S.top ← S.top + 1;
   S.elements[S.top] ← x;
  }
 else return("stack is full")
}
```

פסאודו קוד - מימוש תור ע"י מערך

תור Q הוא רשומה עם 2 שדות:
- Elements[ ]: מערך של אלמנטים בגודל MAXSIZE.
- Front: מצביע אינדקס.
- Rear: מצביע אינדקס.

Empty:

```
Empty(S)
{if (Q.front == Q.Rear) return(TRUE);
 else return(FALSE);
}
```

Front:

```
FRONT(Q)
{
if (!EMPTY(Q)) return (Q.elements[Q.front])
   else return (STACK_IS_EMPTY)
}
```

Dequeue(Q):

```
DEQUEUE(Q)
{
   if (!EMPTY(Q))
      {
         x ← Q.elements[Q.front];
         Q.front++;
         return(x);
      }
   else
      return(STACK_IS_EMPTY);
}
```

Enqueue(Q):

```
ENQUEUE(x, Q)
{
   if (Q.rear == MAXSIZE − 1)
         return(END_OF_STACK);
   else
      {
         Q.elements[Q.rear] ← x;
         Q.rear++;
      }
}
```

מימוש תור ע"י 2 מחסניות:

DeQUEUE:

```
DeQUEUE(Q)
{While(!EMPTY(S1))
   {x ← POP(S1)
    PUSH(S2,x)
   }
dequeue_value=POP(S2)
While(!EMPTY(S2))
   {x ← POP(S2)
    PUSH(S1,x)
   }
return(dequeue_value)
}
```

ENQUEUE:

```
ENQUEUE(Q,x)
{PUSH(S1,x)
}
```

היפוך מחסנית:

```
Reverse_Stack(S1)
{While(!EMPTY(S1))
   {x ← POP(S1)
    PUSH(S2,x)
   }
While(!EMPTY(S2))
   {x ← POP(S2)
    print(x)
   }
}
```

שיפור תור ע"י 2 מחסניות:

```
QUEUE_EMPTY(Q)
{
return(EMPTY(S1)&&EMPTY(S2))
}

DeQUEUE_EMPTY(Q)
{IF(EMPTY(S2))
   {While(!EMPTY(S1))
      {x ← POP(S1)
       PUSH(S2,x)
      }
   }
return(POP(S2))
}
```

פסאודו קוד מימוש תור ע"י מערך מעגלי

Dequeue:

```
DeQUEUE(Q)
{IF(!EMPTY(Q))
   {x ← Q.elements[Q.front]
    Q.front ← (Q.front + 1) mod MAXSIZE
    Q.num_of_elements --
    return(x)
   }
 ELSE
   {return("stack is enpty")
   }
}
```

Enqueue:

```
ENQUEUE(Q,x)
{IF(Q.num_of_elements=MAXSIZE)
   return ("stack is full")
 ELSE
   {Q.elements[Q.rear] ← x
    Q.rear ← (Q.rear + 1) mod MAXSIZE
    Q.num_of_elements ++
   }
}
```

Empty:

```
EMPTY(Q)
{
 IF(Q.num_of_elements=0)  return (TRUE)
 ELSE return(FALSE)
}
```

פסאודו קוד – פעולות על תור קדימויות מינימלי ממומש בעזרת ערימה

Insert:

```
Insert(P, x)   // heap
{if (P.size = MAXSIZE)
  return("overflow");
 P.size++;
 P.T[size] ← x;
 curr ← P.size;
 while(curr > 1) &
   (P.T[curr] < P.T[⌊curr/2⌋])
   {P.T[curr] ↔ P.T[⌊curr/2⌋];
    curr ← ⌊curr/2⌋;
}
```

Delete Min:

```
Delete _ Min(P)
{if (EMPTY(P))
  return(underflow);
 min ← P.T[1];
 if (P.size = 1)
  {P.size ← (P.size) − 1;
   return(min);
  }
 P.T[1] ← P.T[P.size];
 P.size ← (P.size) − 1;
 HEAPIFY(P,1);
}
```

Heapify:

```
HEAPIFY(P,i)
{if (2i > P.size)  return();

 if (2i+1 > P.size)
   {if (P.T[i] > P.T[2i])
     {P.T[i] ↔ P.T[2i];
      return();
     }
   }
 else
   {if (P.T[2i] < P.T[2i+1]) then  j ← 2i;
    else j ← 2i+1;
    if (P.T[i] > P.T[j])
      {P.T[i] ↔ P.T[j];
       HEAPIFY(P, j);
      }
    else return()
   }
}
```

Array to Heap:

```
ArrayToHeap(A[ ],n)
{for i ← 1 to n
   P.T[i] ← A[i];
 P.size ← n;
 for i ← ⌊P.size/2⌋ downto 1
   Heapify(P,i)
}
```

פסאודו קוד - קבוצות זרות

Find Set:

```
FindSet(x)
{return(Sets(x).rep);
}
```

Make Set:

```
MakeSet(x)
{if (FindSet(x) = −1)
   Sets[x].rep ← x;
 else return(element in set);
}
```

פסאודו קוד – עץ פורש מינימלי

Kruskal - מימוש ע"י ADT - קבוצות זרות:

```
MST − Kruskal(G, w) // w = weight
                    // G = graph
{A ← ∅;
 for each vertex v ∈ V
 do MakeSet(v);
 sort E from min to max;
 for each edge (x, y) ∈ E in sorted order
 if FindSet(x) ≠ FindSet(y)then
  {A → A ∪ {x, y};
   union(x, y)
  }
}
```

Prim - מימוש ע"י ADT - קבוצות זרות:

מבני הנתונים שבהם נשתמש לבניית האלגוריתם:

- מערך Parent [y]
בכל תא נשמר המשקל של הצלע שאם נחבר אותה עם העץ נקבל עץ פורש מינימלי.
בסוף האלגוריתם מערך זה מהווה את העץ הפורש המינימלי.
- תור קדימויות מינימום
הממומש ע"י **ערימה** בו יוחזקו בכל שלב כל הקודקודים שעדיין אינם שייכים לעץ.
קדימות = משקל הצלע המינימלית היכולה להתחבר לעץ.
- מערך Place[ ]:
הערך של Place [y] יהיה האינדקס של קודקוד y בערימה.

```
MST-Prim(G,w,r)
{Q ← V[G]
 for each u ∈ Q
    do key[u] ← ∞
 key[r] ← 0
 π[r] ← NIL
 while(Q ≠ ∅)
  {u ← ExtractMin(Q)
   for each v ∈ Adj[u]
     {if v ∈ Q and w(u,v)<key[v]
       then π[v] ← u and key[r] ← 0
     }
  }
}
```

פסאודו קוד - פרוצדורות להדפסת ערכי הצמתים בעץ חיפוש בינארי

In Order Tree Walk:

```
InOrderTreeWalk(x)
{if (x ≠ NULL)
  {InOrderTreeWalk(x → lchild);
   print(x → val);
   InOrderTreeWalk(x → rchild);
  }
}
```

Post Order Tree Walk:

```
PostOrderTreeWalk(x)
{if (x ≠ NULL)
  {PostOrderTreeWalk(x → lchild);
   PostOrderTreeWalk(x → rchild);
   print(x → val);
  }
}
```

Pre Order Tree Walk:

```
PreOrderTreeWalk(x)
{if (x ≠ NULL)
  {print(x → val);
   PreOrderTreeWalk(x → lchild);
   PreOrderTreeWalk(x → rchild);
  }
}
```

פסאודו קוד - פעולות בעץ חיפוש בינארי

Insert:

```
Insert(x, pnode)
{if (pnode → val = x)
  return(vall − in − tree);
 if (pnode → val > x)
  {if (pnode → rchild != null)
     Insert(x, pnode → rchild);
   else
     {create new node;
      newpnode is point er to it;
      newpnode → val = x;
      (newpnode → parent) ← pnode;
      (pnode → rchild) ← newpnode;
     }
  }
 else
  {if (pnode → lchild != null)
     Insert(x, pnode → lchild);
   else
     {create new node;
      newpnode is point er to it;
      newpnode → val = x;
      (newpnode → parent) ← pnode;
      (pnode → lchild) ← newpnode;
     }
  }
}
```

פסאודו קוד – תכנות דינמי

Matrix Multiply:

מכפיל מטריצה A במטריצה B.

```
MatrixMultiply(A, B)
{if column[A] ≠ row[B]
   return(error);
 else
  {for i ← 1 to rows[A]
     {for j ← 1 to columns[B]
        {C[i, j] ← 0;
         for k ← 1 to columns[A]
           C[i, j] ← C[i, j] + A[i, k] · B[k, j];

        }
     }
  }
return(C)
}
```

Matrix Chain Multiply:

מכפיל מספר רב של מטריצות.

```
MatrixChainMultiply(A, s, i, j)
{if (j > i)
  {x ← MatrixChainMultiply(A, s, i, s[i, j]);
   y ← MatrixChainMultiply(A, s, s[i, j] + 1, j);
   return(MatrixMultiply(x, y));
  }
 else  return(Ai);
}
```

Recursive Matrix Chain:

מחשב את אופן הכפלת המטריצות האופטימלי בצורה רקורסיבית רגילה (ללא Meomize).

```
RecursiveMatrixChain(p, i, j)
{if i = j  return(0);
 m[i, j] ← ∞;
 for k ← i to  j − 1
  {q ← RecursiveMatrixChain(p, i, k)
      + RecursiveMatrixChain(p, k + 1, j)
      + p_{i−1} · p_k · p_j;
   if q < m[i, j]
      m[i, j] ← q;
  }
return(m[i, j])
}
```

Matrix Chain Order:

מחשב את אופן הכפלת המטריצות האופטימלי ב- Buttom Up.

```
MatrixChainOrder(p)
{n ← length[p] − 1;
 for i ← 1 to n
    m[i, i] ← 0;
 for l ← 2 to n
  {for i ← 1 to n − l + 1
    {j ← i + l − 1;
     m[i, j] ← ∞;
     for k ← i to j − 1
      {q ← m[i, k] + m[k + 1, j] + p_{i−1} · p_k · p_j;
       if q < m[i, j]
         {m[i, j] ← q;
          s[i, j] ← k;
         }
      }
    }
  }
return m and s;
}
```

Memoized Matrix Chain:

מחשב את אופן הכפלת המטריצות האופטימלי ע"י רקורסיה דינימית (עם Meomize).
עד לקריאה ל- LookUpChain זהו רק אתחול.

```
MemoizedMatrixChain(p)
{n ← length[p] − 1;
 for i ← 1 to n
  {for j ← i to n
     m[i, j] ← ∞;
  }
return(LookUpChain(p,1,n))
}
```

Delete:

```
Delete(x, T)
{pnode ← Find(x, T);
 if (pnode = null)
   return(not − found);
 if (pnode = (pnode → parent) → lchild)
   side ← left;
 else side ← right;
 if (pnode → lchild = null &
    pnode → rchild = null)
   {if (side = left)
      [(pnode → parent) → lchild] ← null;
    else
      [(pnode → parent) → rchild] ← null;
    free(pnode);
   }
 else
   {if (pnode → lchild = null)
     {if (side = right)
        [(pnode → parent) → rchild] ← (pnode → rchild);
      else
        [(pnode → parent) → lchild] ← (pnode → rchild);
      ([pnode → rchild] → parent) ← (pnode → parent);
      free(pnode);
     }
    else
     {if (pnode → rchild = null)
       {if (side = right)
          ([pnode → parent] → rchild) ← (pnode → lchild);
        else
          ([pnode → parent] → lchild) ← (pnode → lchild);
        ([pnode → lchild] → parent) ← (pnode → parent);
        free(pnode);
       }
      else
       {min_val ← Min(pnode → rchild);
        Delete(min_val, pnode → rchild);
        (pnode → val) ← min_val;
       }
     }
   }
}
```

Member:

מימוש ע"י עץ מקושר, כאשר כל קודקוד הוא רשומה עם השדות הבאים:

- Val - הערך שהוא מחזיק.
- Par - מצביע להורה.
- Rchild - מצביע לילד הימני.
- Lchild - מצביע לילד השמאלי.

```
Member(x, T) // T − point er to root //
{pnode ← T; found ← FALSE;
 while((! found) & (pnode != NULL))
  {if (pnode → val = x)
     fount ← TRUE;
   else if (x < pnode → val)
          pnode ← (pnode → lchild);
        else pnode ← (pnode → rchild);
  }
 return(found)
}
```

Recursive Member:

```
RecMember(x, T) // T − point er to root //
{if (pnode = NULL)
   return(FALSE);
 else if (pnode → val = x)
         return(TRUE);
      else if (x < pnode → val)
             return(Member(x, pnode → lchild));
           else
             return(Member(x, pnode → rchild));
}
```

מציאת המספר הקודם לקודקוד:

מוצא את המספר הקודם לקודקוד מבין כל הערכים בעץ.

```
predecessor(node_ptr)
{if(node_ptr → lchild ≠ NULL)
     return(MAX(node_ptr → lchild))
 else
   {temp_ptr ← node_ptr
    found ← FALSE
    while((temp_ptr → paretn ≠ NULL)
                  & (found=FALSE))
      {if (temp_ptr=(temp_ptr → parent) → lchild)
         temp_ptr ← (temp_ptr → parent)
       else  found ← TRUE
      }
   }
}
```

סיבוכיות:

$\theta(n)$

LookUp Chain:

הפרוצדורה שמוצאת את אופן הכפלת המטריצות האופטימלי ע"י רקורסיה דינמית.

```
LookUpChain(p,i,j)
{if m[i,j] < ∞ return(m[i,j]);
 if i = j then m[i,j] ← 0;
 else
  {for k ← i to j −1
    {q ← LookUpChain(p,i,k) + LookUpChain(p,k+1,j) +
          + p_{i-1} · p_k · p_j;
     if q < m[i,j]
      then m[i,j] ← q;
    }
  }
return(m[i,j]);
}
```

Quick sort:

P - מצביע לאיבר הראשון בתת-מערך.

R - מצביע לאיבר האחרון בתת-מערך.

```
QS(A,P,R)
{if P ≥ R then return "can't sort"
 else
  {q ← partition(A,P,R)
   QS(A,P,q)
   QS(A,q+1,R)
  }
}
```

Partition:

```
Partition(A,P,R)
{x ← A[P]
 i ← P −1
j ← R +1
 while(true)
  {repeat j ← j −1 until A[j] ≤ x
   repeat i ← i +1 until A[i] ≥ x
  if i < j then A[i] ↔ A[j]
  else return (j)
  }
}
```

Dynamic 0-1 Knapsack:

```
Dynamic 0-1 Knapsack(v(·),w(·),n,W)
{for w=0 to W
 {c(0,w) ← 0
 }
for i=1 to n
 {c(i,0) ← 0
  for w=1 to W
   {if w_i > w then c(i,w) ← c(i−1,w)
    else
     c(i,w) ← max(v_i + c(i−1,w−w_i), c(i−1,w))
   }
 }
}
```

Print-Optimal-Knapsack:

```
Print-Optimal-Knapsack(c(·),v(·),n, w)
{i=n
 w=W
 while(i>0)
  {if (c(i,w)=c(i-1,w-w_i)+w_i)
    {print i
     w=w-w_i
    }
  }
i=i-1
}
```

All Pairs Shortest Path(Floyd Warsall):

מוצאים את המסלול האופטימלי בין **כל** 2 קודקודים בגרף מסוים בשיטת Buttom Up.

```
Floyd − Warshall(W)
{n ← rows[W]
 D^(0) ← W
 for k ← 1 to n
 {for i ← 1 to n
   {for j ← 1 to n
     {
      d_ij^(k) ← min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
     {
   }
 }
 return(D^(n))
}
```

Print All Pairs Shorteset Path:

```
Print-All-Pairs-Shorteset-Path(π,i,j)
{if i = j print i
 else
  {if π_ij = NIL then print "no path found"
   else
     {
     Print-All-Pairs-Shorteset-Path(π,i,π_ij)
     print j
     }
  }
}
```

$$\Pi_{ij}^{(0)} = \begin{cases} NIL & ; \quad i = j \ or \ w_{ij} = \infty \\ i & ; \quad otherwise \end{cases}$$

$$\Pi_{ij}^{(k)} = \begin{cases} \Pi_{ij}^{(k-1)} & ; \quad d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \Pi_{kj}^{(k-1)} & ; \quad otherwise \end{cases}$$

בעיית שילוש אופטימלי:

```
t(1,n)
{
 for i=1 to n
  {for i=1 to n-l+1
   j=i+l-1
   t(i,j) ← ∞
   for k ← i to j-1
    {q ← t(i,k)+t(k+1,j)+w(ΔV_{i-1}V_kV_j)
     if q<t(i,j)
      {t(i,j) ← q
       S(i,j) ← k
      }
    }
  }
return t and S
}
```