

סיכום מבני נתונים

סיבוכיות	פעולות מוגדרות	מבנה נתונים
O(1)	Create(S) creates an empty stack	מחסנית טוב כשצריכים את האיברים בצורת LIFO
O(1)	Push(S,x) insert x into the stack	
O(1)	Pop(S) extracts the last inserted item (returns void)	
O(1)	Is_Empty(S) boolean true if stack is empty	
O(1)	Top(S) returns the top item on the stack	
O(1)	Create(Q) Creates an empty Queue	תור טוב כשצריכים את האיברים בצורת FIFO
O(1)	Head(Q) Returns the first member in the Queue	
O(1)	Enqueue(Q,x) Insert x into the Queue	
O(1)	Dequeue(Q) Extract the first member in the Queue (returns void)	
O(1)	Is_empty(Q) Boolean returns true if Queue is empty	
O(1)	הכנסה והוצאה של איבר בהינתן מצביע למקום המתאים	רשימה מקושרת יש גם רשימה מעגלית ודו כיוונית
O(n)	חיפוש איבר	
O(log(n))	הכנסת איבר	עץ חיפוש AVL עץ חיפוש בינארי שבו הפרש גבהים בין בנים הוא לכל היותר 1
O(log(n))	הוצאת איבר	
O(log(n))	חיפוש איבר	
O(log(n))	הכנסת איבר	עץ 2-3 (עץ B+) עץ חיפוש שבו לכל צומת 2 או 3 בנים. זהו עץ B+ מדרגה 3.
O(log(n))	הוצאת איבר	
O(log(n))	חיפוש איבר	
O(log(n))	הכנסת איבר	עץ דרגות - Rank tree תומך במציאת אינדקס של איבר בסבוכיות לוגריתמית
O(log(n))	הוצאת איבר	
O(log(n))	חיפוש איבר	
O(log(n))	מציאת אינדקס של איבר (כמה איברים יש לפניו)	
~O(log(n))	הכנסת איבר	רשימת דילוגים חסרון: סיבוכיות ממוצעת יתרון: מימוש פשוט
~O(log(n))	הוצאת איבר	
~O(log(n))	חיפוש איבר	
~O(1)	הכנסת איבר	Hash Table יתרון: זמן ממוצע O(1). חסרון: יש לדעת מראש סדר גודל מספר מפתחות
~O(1)	הוצאת איבר	
~O(1)	חיפוש איבר	
O(1)	Makeset(i) Return new set with single element i	Union/Find מימוש בעזרת עצים הפוכים
O(log(n))	Find(i) Returns the set that i belongs to	
O(1)	Union(p,q) returns a new group containing all elements of groups p,q	
~O(m)	סיבוכיות משוערכת (לא אמיתית!) ל m פעולות Union/Find	
O(1)	Make_heap(Q) Create an empty Heap	Heap טוב כאשר צריכים לקבל את האיבר הגדול או הקטן ביותר מהה. זהו לא עץ חיפוש. מציאת איבר ב O(n).
O(n)	Make_heap(Q,A) Create a Heap from an array (עץ כמעט שלם)	
O(log(n))	Insert(x,Q) Insert x into the heap	
O(log(n))	Max(Q) Returns the maximal/minimal value in the heap	
O(log(n))	Del_max(Q) Deletes the maximal/minimal value in the heap	
O(s)	הכנסת מחרוזת באורך s	Trie טוב כאשר עובדים עם מחרוזות של תווים או ספרות
O(s)	הוצאת מחרוזת באורך s	
O(s)	חיפוש מחרוזת באורך s	
O(s)	מציאת מחרוזת מינימום לקסיקוגרפי באורך s	
O(s)	יצירת עץ סיומות מתוך מחרוזת באורך s	עץ סיומות (Suffix Tree)
O(k)	מציאת תת מחרוזת באורך k	
O(L ₁ +L ₂)	מציאת תת מחרוזת משותפת למחרוזות באורך L ₁ , L ₂	

סיכום סיבוכיות

$S = \frac{(a_1 + a_n) \cdot n}{2}$	סכום סדרה חשבונית:
$S = \frac{a_1 \cdot (q^n - 1)}{q - 1}$	סכום סדרה הנדסית:
$S = \frac{a_1}{1 - q}$	סכום טור הנדסי מתכנס:

הפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $O(g(n))$ אם קיימים קבועים c, n_0 כך שלכל $n \geq n_0$ מתקיים:
 $f(n) = O(g(n)) \Leftrightarrow f(n) \leq c \cdot g(n)$
 מהווה חסם עליון אסימפטוטי.
 אותו הדבר רק הפוך לגבי חסם תחתון (אומגה).
 $f(n) = \Omega(g(n)) \Leftrightarrow f(n) \geq c \cdot g(n)$
 $f(n) = \theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \ \&\& \ f(n) = \Omega(g(n))$

פתרון רקורסיות

שיטת ההצבה: שיטה שנועדה רק להוכיח סיבוכיות שנחשנו מראש. הוכחה בעזרת אינדוקציה.
שיטת האיטרציות: פותחים את הרקורסיה כסכום איברים התלויים בתנאי ההתחלה ובא.
שיטת המאסטר: פותרת רקורסיות מהצורה: $T(n) = aT(n/b) + f(n)$ כאשר $a \geq 1$ ו $b > 1$ קבועים ו $f(n)$ פונקציה.
 $T(n)$ מוגדרת עבור שלמים אי שליליים.
 שיטת הפתרון:
 1. החשב $x = \log_b(a)$
 2. אם $f(n) < c \cdot n^{x-\epsilon}$ אזי $T(n) = \theta(n^x)$
 3. אם $f(n) = c \cdot n^x$ אזי $T(n) = \theta(n^x \log(n))$
 4. אם $f(n) > c \cdot n^{x+\epsilon}$ וגם $a \cdot f(n/b) \leq k \cdot f(n)$ כאשר $k < 1$ קבוע אזי $T(n) = \theta(f(n))$

מעבר על עצים:

Pre-Order: קודם על השורש, אז בן שמאלי ואז בן ימני
In-order: קודם בן שמאלי, אז שורש, ואז בן ימני
Post-Order: קודם בן שמאלי, אז בן ימני, ואז שורש.

מיונים:

Counting Sort (Bucket Sort) - מיון יציב (לא משנה סדר יחסי של איברים בעלי ערכים זהים) בו ניתן להשתמש כאשר תחום הערכים k סופי וידוע מראש (קבוע). סיבוכיות - $O(n+k)$
Radix-Sort - אם נתונים n מספרים בעלי d ספרות בבסיס b אזי ניתן למיינם בסיבוכיות $O(d \cdot (n+b))$ וכאשר d, b קבועים $O(n)$. משתמש ב Bucket-Sort ולכן הינו מיון יציב.
Quick-Sort - בבחירה אקראית של Pivot נותן סיבוכיות ממוצעת של $\theta(n \cdot \log(n))$ ואם ניתן למצוא חציון ב $O(1)$ אזי הסיבוכיות היא $\theta(n \cdot \log(n))$ אמיתית.
Heap-Sort - בניית ערימת מיונים מהאיברים למיון, ואז הוצאתם אחד אחד מהקטן לגדול. סיבוכיות $O(n \cdot \log(n))$

פונקציות ערבול יעילות:

- שיטת המודולן
 $h(x) = x \text{ mod } m$ כאשר m מספר ראשוני שאינו קרוב לחזקה של 2.
- שיטת הכפל בקבוע בין 0 ל-1.
 $h(k) = \lfloor m \cdot (a \cdot k \text{ mod } 1) \rfloor$
 א. הכפל את המפתח k ב- a .
 ב. קח רק את השבר העשרוני של התוצאה
 ג. הכפל אותו ב- m (פקטור גודל טבלת ערבול) ועגל כלפי מטה.
 $a = \frac{\sqrt{5}-1}{2}$
- פונקציות ערבול למחרוזות ארוכות:
 א. הכנס פרמוטציה אקראית מעל המספרים 0-255 למערך $T[0..255]$
 ב. בצע Bitwise xor בין האות הראשונה s_1 במחרוזת לבין $T[0]$. התוצאה a_1 נמצאת בתחום 0..255.
 ג. בשלב ה- i בצע Bitwise xor בין האות S_i במחרוזת לבין $T[a_{i-1}]$ כאשר a_{i-1} הוא תוצאת ה xor בשלב הקודם.
 ד. תוצאת פונקצית הערבול היא תוצאת xor עם האות האחרונה של מחרוזת המפתח כלומר a_n xor $S_n = \text{hash(string)}$
- קבוצה אוניברסלית
 א. בחר גודל טבלת ערבול ראשוני m .
 ב. הכנס פרמוטציה אקראית מעל התחום 0.. $m-1$ למערך $T[0..r]$
 ג. שבור את המפתח ל $r+1$ חלקים ואסוף אותו במערך $\text{key}[0..r]$
 ד. תוצאת פונקצית הערבול:

$$h(x) = \left(\sum_{i=0}^r \text{key}[i] \cdot T[i] \right) \text{ mod } m$$