מבני נתונים - אוניברסיטת רייכמן

הרצאה 2 - Priority Queue, עצים וערמות

Priority Queues, Trees & Heaps

תוכן עניינים

1. חזרה: רשימה מקושרת כפולה (DLL)
2. השוואת סיבוכיות: מערך מעגלי מול DLL
3. מערכים דינמיים עם Resizing
4. אסטרטגיית הכפלה (Doubling)
5. ניתוח Amortized
6. Amortized מול Average Case
7. Priority Queue ADT
8. עצים — הגדרות
9. טרמינולוגיית עצים
10. עצים בינאריים
11. סיכום

1. חזרה: רשימה מקושרת כפולה (Doubly Linked List)

ברשימה מקושרת כפולה, לכל צומת יש שדה **prev** (מצביע לקודם) ושדה **next** (מצביע לבא). פעולות Insert/Delete-Last וכן Concatenation מתבצעות ב-$O(1)$.

מחיקת צומת — Delete(A)

פסאודו-קוד: Delete(A)

```
A.prev.next ← A.next
A.next.prev ← A.prev
```

שימו לב

- הצומת A עצמו **לא משתנה** — רק הפוינטרים של השכנים מתעדכנים.
- צריך לטפל בנפרד ב-`L.length` ובמקרי קצה (ראשון/אחרון).

הכנסת צומת — Insert-After(A, B)

פסאודו-קוד: Insert-After(A, B)

```
B.prev ← A
B.next ← A.next
A.next ← B
B.next.prev ← B
```

שתי הפעולות ב-$O(1)$ — לא תלויות בגודל הרשימה.

2. השוואת סיבוכיות: מערך מעגלי מול רשימה מקושרת כפולה

פעולה	Circular Array	Doubly Linked List
Insert/Delete-First	$O(1)$	$O(1)$
Insert/Delete-Last	$O(1)$	$O(1)$
Insert/Delete(i)	$(\{1+i-n\ ,1+O(\min\{i$	$(\{1+i-n\ ,1+O(\min\{i$
Retrieve(i)	$O(1)$	$(\{1+i-n\ ,1+O(\min\{i$
Concatenation	$(1+O(\min\{n_1,n_2\}$	$O(1)$

מסקנה

- **יתרון DLL:** שרשור (Concatenation) ב-$O(1)$.
- **יתרון מערך מעגלי:** גישה אקראית (Retrieve) ב-$O(1)$.

3. מערכים דינמיים עם Resizing

כשהמערך מלא (`length = maxlen`), לא ניתן להרחיב אותו — יש להקצות מערך חדש גדול יותר ולהעתיק.

הבעיה: הגדלה ב-1

אם מגדילים את המערך ב-1 בכל פעם שהוא מתמלא:

$$\text{עלות } n \text{ פעולות Insert-Last} = 1+2+\ldots+n = \frac{n(n+1)}{2} = O(n^2)$$

בעיה

$O(n^2)$ עבור n הכנסות — זה **מאוד יקר**. צריך אסטרטגיית הגדלה חכמה יותר.

4. אסטרטגיית הכפלה (Doubling)

הרעיון

כשהמערך מלא — **מכפילים את גודלו** ומעתיקים את כל האיברים.

ניתוח עלות — המקרה $^{k}2 = n$

$$\underbrace{(1+1+\ldots+1)}_{n\text{ insertions}} + \underbrace{(1+2+4+\ldots+\tfrac{n}{2})}_{\text{copying}} = n + (n-1) = O(n)$$

המקרה הכללי — $^{k}2 > r \geq 0$,$r + {}^{k}2 = n$

$$\underbrace{(1+1+\ldots+1)}_{n} + \underbrace{(1+2+4+\ldots+2^k)}_{\text{copying}} = n + (2^{k+1}-1) \leq n + 2n -$$
$$1 = 3n - 1 = O(n)$$

תוצאה

- העלות **הכוללת** של n פעולות Insert-Last עם doubling: $O(n)$.
- העלות ה-**amortized** (ממוצעת לפעולה): $O(1) = O(n)/n$.
- העלות ה-**worst-case** של פעולה בודדת: $O(n)$ (כשיש העתקה).

5. ניתוח Amortized — הגדרה פורמלית

הגדרה: Worst-case bound

$\text{worst}(op)$ = הזמן **המקסימלי** לביצוע פעולה op.

חסם על סדרה של n פעולות:

$$\text{Time}(n \times op) \leq n \times \text{worst}(op)$$

לפעמים החסם הזה **מאוד רופף**. למשל: $O(n) = \text{worst(Insert-Last)}$, אבל $O(n) = (\text{Insert-Last} \times \text{Time}(n$, לא $O(n^2)$.

הגדרה: Amortized bound

$\text{amort}(op)$ הוא חסם amortized על עלות פעולה op אם **לכל** n ולכל סדרה של n פעולות op:

$$\text{amort}(op) \geq \frac{\text{Time}(n \times op)}{n}$$

דוגמה: Insert-Last עם Doubling

- $O(n) = \text{worst(Insert-Last)}$
- $O(1) = \text{amort(Insert-Last)}$
- $O(n) = \text{amort(Insert-Last)} \cdot n \geq (\text{Insert-Last} \times \text{Time}(n$

6. Amortized Analysis ≠ Average Case Analysis

Amortized Analysis

חוסם את הזמן **הממוצע לפעולה** על פני **סדרת הפעולות הגרועה ביותר** (worst-case sequence).

אין הסתברות.

Average Case Analysis

חוסם את הזמן הממוצע של **פעולה בודדת** על פני **כל הקלטים האפשריים** (בהינתן התפלגות).

מבוסס הסתברות.

מתי להשתמש במה?

- **Amortized:** כשהעלות **הכוללת** של סדרת פעולות חשובה יותר מעלות כל פעולה בודדת.
- **Worst-case:** למערכות **real-time** שצריכות כל פעולה בודדת להיות מהירה.

7. Priority Queue ADT

בתור רגיל (Queue) — סדר השירות נקבע לפי זמן הגעה (FIFO). אבל מה אם צריך סדר לפי **עדיפות**?

מוטיבציה

- שידור השיר הבא בפופולריות.
- שירות הלקוח הבכיר/הצעיר ביותר.
- ניהול תהליכים במערכת הפעלה לפי עדיפות.

הגדרה: Priority Queue

ADT לניהול קבוצת איברים כאשר **לכל איבר יש עדיפות (מפתח, key)**.

פעולות:

- **Insert(x, k)** — הכנסת איבר x עם מפתח k.
- **FindMin()** — החזרת handle לאיבר עם מפתח **מינימלי**.
- **DeleteMin()** — מחיקת האיבר עם מפתח מינימלי והחזרת handle אליו.
- **DecreaseKey(h, k)** — הקטנת המפתח של איבר קיים (לפי handle h) ל-k.

פתרון נאיבי

שמירת סדרה **ממוינת** לפי עדיפות — אבל Insert יעלה $O(n)$.

יש פתרון יעיל יותר — מבוסס על עצים (Heap)!

8. עצים — הגדרות

רשימות, מחסניות ותורים הם מבנים **ליניאריים**. הרבה מידע מאורגן בצורה **היררכית**: תיקיות, מהלכים במשחק, היררכיות ארגוניות, ביטויים אריתמטיים.

הגדרה רקורסיבית

הגדרה: עץ (רקורסיבית)

עץ הוא קבוצת צמתים (nodes) שהיא:

- **ריקה**, או
- מכילה צומת אחד הנקרא **שורש** (root) שמצביע לשורשים של אפס או יותר **עצים זרים** (disjoint) הנקראים **תתי-עצים** (subtrees).

disjoint = אין צמתים משותפים בין תתי-העצים.

הגדרה לא-רקורסיבית

הגדרה: עץ (לא-רקורסיבית)

מבנה בעל רמות (levels):

- איבר **יחיד** ברמה העליונה (שורש).
- כל איבר ברמה i מצביע ל-**אפס או יותר** איברים ברמה $i+1$.
- כל איבר ברמה $i+1$ מוצבע ע"י **איבר יחיד** ברמה i.

9. טרמינולוגיית עצים

מונחים בסיסיים

מושג	אנגלית	הגדרה
שורש	Root	הצומת היחיד ללא הורה
ילד	Child	B הוא ילד של A אם A מצביע ל-B
הורה	Parent	A הוא הורה של B אם A מצביע ל-B
עלה	Leaf	צומת ללא ילדים
צומת פנימי	Internal node	צומת שאינו עלה
אח	Sibling	A ו-B אחים אם יש להם אותו הורה

מסלולים, עומק וגובה

מושג	אנגלית	הגדרה
מסלול מכוון	Path (directed)	סדרה של צמתים שכל צומת הוא הורה של הבא
אורך מסלול	Length of path	מספר ה-**קשתות** (edges) במסלול
אב קדמון	Ancestor	A הוא ancestor של B אם קיים מסלול מ-A ל-B
עומק צומת	Depth	אורך המסלול מהשורש לצומת
עומק עץ	Depth of tree	העומק של הצומת העמוק ביותר
גובה צומת	Height	אורך המסלול הארוך ביותר מהצומת לעלה
גובה עץ	Height of tree	גובה השורש (= עומק העץ)

מושגים נוספים

מושג	אנגלית	הגדרה
תת-עץ	Subtree	תת-העץ שבשורשו צומת B
דרגה	Degree	מספר הילדים של צומת
עץ סדור	Ordered tree	מוגדר סדר על ילדי כל צומת
עץ לא-סדור	Unordered tree	אין סדר על הילדים

strict — Ancestor ולא-strict

- **Strict ancestor:** המסלול באורך **לפחות 1** (צומת אינו strict ancestor של עצמו).
- **Non-strict ancestor:** המסלול יכול להיות באורך 0 (צומת **כן** נחשב ancestor של עצמו).

10. עצים בינאריים (Binary Trees)

הגדרה: עץ בינארי

עץ בינארי הוא **עץ סדור** שבו לכל צומת יש **לכל היותר שני ילדים** — ילד שמאלי וילד ימני.

העץ הנפוץ ביותר במדעי המחשב.

סוגים מיוחדים

סוג	אנגלית	תנאי
עץ מלא	Full binary tree	כל צומת הוא בדרגה 2 או 0 (כל צומת פנימי עם בדיוק 2 ילדים)
עץ שלם	Complete binary tree	עץ מלא (full) שבו **כל העלים באותו עומק**
עץ כמעט-שלם	Almost-complete BT	ייתכנו צמתים חסרים רק ב**סוף הרמה העמוקה ביותר** (מימין)

> **חשוב**
>
> עץ בינארי **כמעט-שלם** (almost-complete) הוא הבסיס למבנה **Heap** — שילמד בהרצאה הבאה.

11. סיכום ההרצאה

DLL Review → Dynamic Array → Amortized Analysis → Priority Queue → Trees & Binary Trees

נקודות עיקריות

- ברשימה מקושרת כפולה — Delete ו-Insert-After ב-$O(1)$. שרשור ב-$O(1)$ (יתרון על מערך).
- מערך דינמי עם הגדלה ב-1: $O(n^2)$ עבור n הכנסות. עם **הכפלה (doubling)**: $O(n)$.
- **Amortized analysis** חוסם את העלות הממוצעת לפעולה על פני סדרת ה-worst-case. שונה מ-average case!
- $\text{amort}(\text{Insert-Last}) = O(1)$ אף ש-$\text{worst}(\text{Insert-Last}) = O(n)$.
- **Priority Queue ADT:** Insert, FindMin, DeleteMin, DecreaseKey. מימוש יעיל ע"י Heap.
- **עץ:** מבנה היררכי. מונחי מפתח: שורש, עלה, הורה, ילד, עומק, גובה, דרגה.
- **עץ בינארי:** כל צומת עם עד 2 ילדים. סוגים: full, complete, almost-complete.

> **בהרצאה הבאה**
>
> מימוש Priority Queue באמצעות **Heap** (ערמה) — עץ בינארי כמעט-שלם עם תכונת הערמה.