תרגול 2 – ניתוח Amortized וערמות ביבואריות

Amortized Analysis & Binary Heaps

1. חזרה: ניתוח Amortized

מטרה

ניתוח **Amortized analysis** נועד אות המטרה על עד מדוח שלנו, ולא מקרה שלנו. לפעולות יקרות נדירות.

השוואה: Array Doubling

- **Worst case** — $O(n)$ — הפעולה היקרה ביותר
- **Amortized** — $O(1)$ — ממוצע לאורך כל הסדרה
- אך ב**בודד** יכול להיות n — ממוצע ≠ ערך בודד

Amortized Cost — $O(1)$

2. מחה ביאורי — ניתוח Amortized

מחה ביאורי

מחה ביאורי מחה פ של n — ביאורי מחה ב-$\lceil \log_2 n \rceil$ ביאורי.

השוואה: מחה פ של n מחה ביאורי ב-$\lceil \log_2 n \rceil$ ביאורי

מחה פ	ייצוג ביאורי
0	[0,0,0]
1	[0,0,1]
2	[0,1,0]
3	[0,1,1]
4	[1,0,0]
5	[1,0,1]
6	[1,1,0]
7	[1,1,1]

כמה flips סה"כ?

- ביט אחד משתנה בכל מחה פ — **בדיוק** n מחה פים
- ביט אחד משתנה ב-$\frac{n}{2}$ מחה פים
- ביט ב-i משתנה ב-$\lfloor n/2^i \rfloor$ מחה פים

$$\text{Total flips} = n + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \cdots + \left\lfloor \frac{n}{2^{\lfloor \log n \rfloor}} \right\rfloor$$

חישוב

$$2n = 2n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} \geq \left\lfloor \frac{n}{1} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \cdots + 1 = n + \frac{n}{2} + \frac{n}{4} + \cdots + 1 = \sum_{i=0}^{\lfloor \log n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$$

לכן ניתוח Amortized מחה פ של n הוא $O(1)$.

3. Binary Heap — הגדרה ותכונות

הגדרה: Binary Heap

ערמה היא תבנה מחוסן מחוסן בתשלות *Priority Queue*.

Binary Heap הוא **בינארי כמעט-מלא** (almost-complete) **ומקיים תכונת ערמה** (*heap property*).

Almost-Complete Binary Tree

- כל הרמות **חון מהאחרונה** מלאות לגמרי
- ברמה האחרונה — העלים **נמצאים משמאל לימין**

תכונת הערמה

- **Max-Heap** — כל צומת ≥ כל בניו/ה
- **Min-Heap** — כל צומת ≤ כל בניו/ה

(בהמשך נתמקד ב-Max-Heap)

4. פיתוח Heap במערך

ניתן ב**מערך** לפי סדר **רמה-רמה**, החל מהשורש ומשמאל לימין.

רמה במערך ובחישוב מספרי ה-i:

פעולה	מספר
הורה של צומת i	$\lfloor i/2 \rfloor$
בן שמאלי של i	$2i$
בן ימני של i	$2i + 1$

דוגמה: Max-Heap במערך

ייצוג:

$$[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$$

דוגמה: בצומת $i = 2$ מצב A[2]. ילדיו הם $2i = 4$ ו-$2i + 1 = 5$, כלומר A[4] ו-A[5].

5. Percolate Up / Down

שני שאלות הבסיס לתיקון תכונת הערמה.

Percolate Up — "שחרור לתוכה"

Percolate Down — "טמון לתוכה"

מתי נשתמש? כאשר ערך גדל ועלול להפר את תכונת הערמה מול ההורה שלו.

מתי נשתמש? כאשר ערך קטן ועלול להפר את תכונת הערמה מול הבנים שלו.

פיתוח אחרי הוספת עלה לסוף המערך.

פיתוח אחרי מחיקת השורש (או החלפתו באיבר אחר).

```
PercolateUp(arr[1..heap-size], i):
  while i > 1 and arr[i] is smaller than one of its children:
      swap arr[i] with arr[i] // 2
      i = i // 2

PercolateDown(arr[1..heap-size], i):
  while arr[i] is smaller than one of its children:
      swap arr[i] with the larger child
      i = that child's index
```

הוכחת נכונות

שאלת הנכונות בהקשרים **כל איחור צומת אחד בכל רמה** — $O(\log n)$.

הוכחת PercolateDown: הוכחה

עם עץ שורשו A, B ו-C:

- אם $A \geq B$ ו-$A \geq C$ → **עצור** — $A \geq B$ ו-$A \geq C$ → **עצור** — $A \geq B$ ו-$A \geq C$ → **עצור**
- אם $A < B$ → **החלף עם B** — $A < B$ → **החלף עם B** — $A < B$ → **החלף עם B**

בכל שלב — נמשיכים עם **מלוא ערמה בגובה**.

6. פעולות על Max-Heap

פעולה	תיאור	סיבוכיות
find-max	מחזירים את האיבר המקסימלי תמיד באיבר	$O(1)$
insert(e, k)	מוסיפים ומבצעים את אחרון ל-PercolateUp	$O(\log n)$
delete-max	מוציאים את שורש ומבצעים אחרון מחליפים שורש ל-PercolateDown	$O(\log n)$

דוגמה: Max-Heap לאחר insert(42)

מצב התחלתי: [50, 30, 40, 10, 20, 35, 45]

שלב 1 — הוספה בסוף: [50, 30, 40, 10, 20, 35, 45, **42**]

שלב 2 — PercolateUp: 42 > 40 → החלפה → [50, 30, **42**, 10, 20, 35, 45, **40**]

שלב 3 — PercolateUp: 42 < 50 → עצירה → [50, 30, 42, 10, 20, 35, 45, 40]

תוצאה: 42 נשאר ב-$A[3]$, כי 42 < 50 ו-42 > 40 — תכונת הערמה נשמרת.

דוגמה: ExtractMax מפורט

ExtractMax [50] מחזיר 50.

שלב 1 — החלפה: [**45**, 30, 42, 10, 20, 35, **50**]

שלב 2 — הסרת 50: [45, 30, 42, 10, 20, 35]

שלב 3 — PercolateDown: 45 < 42 → החלפה → [**42**, 30, **45**, 10, 20, 35]

שלב 4 — PercolateDown: 45 > 35 → עצירה → [42, 30, 45, 10, 20, **35**]

7. BuildHeap

בנייה מאפס!

ניתן לבנות ערמה מאפס ע"י הפעלת PercolateDown **החל מהאמצע** — מהחלק שמכיל בנים ומטה.

דוגמה: BuildHeap על [2, 30, 19, 21, 16, 24, 18]

[2,30,19,21,16,24,18] → PercolateDown(3): 19>18 → swap → [2,30,**18**,21,16,24,**19**]

[2,30,18,21,16,24,19] → PercolateDown(2): 21>16 → swap → [2,**21**,18,**30**,16,24,19]

[2,21,18,30,16,24,19] → PercolateDown(1): 2<21 → swap → [**21**,**2**,18,30,16,24,19]

[21,2,18,30,16,24,19] → PercolateDown(2): 2<30 → swap → [21,**30**,18,**2**,16,24,19]

BuildHeap מתבצעת ב-$O(n)$ — ולא $O(n \log n)$, אחרי כמות הפעולות ב-PercolateDown לכל רמה.

ראה הוכחה ב-[CLRS] עמ' 157.

$$\sum_{h=0}^{\log n} h \cdot 2^{\log n - h} = O(n)$$

BuildHeap ב-$O(n)$.

שאלה 1 — מספר העלים בעץ Almost-Complete

שאלה

הבעיה: בעץ בינארי כלשהו מחלק עם n צמתים, ניתוח $\lceil n/2 \rceil$ הצמתים הם עלים.

פתרון

נוסחה סגורה

- עץ מלא בגובה h מכיל 2^h עלים ו-$2^h - 1$ צמתים פנימיים
- נסמן ב-x את מספר העלים ברמה האחרונה ו-y את מספר הצמתים הפנימיים ברמה זו
- אז $x + y = 2^h$ ו-$y = x - 1$
- מכאן $x = 2^h - x + 1$ ו-$x = \frac{n+1}{2}$
- מכיוון ש-$n = 2^h - 1 + x$ נקבל שיש x רמה h — x עלים

הוכחה 1: אי-שוויון

$$\lceil \frac{n}{2} \rceil = \lceil \frac{2^h - 1 + x}{2} \rceil = 2^{h-1} - 1 + \lceil \frac{x}{2} \rceil = 2^{h-1} - 1 + x - \lfloor \frac{x}{2} \rfloor = 2^h - 1 + x - \lfloor \frac{x}{2} \rfloor = n - \lfloor \frac{x}{2} \rfloor$$

$$\lceil \frac{n}{2} \rceil \geq n - x \quad \checkmark$$

הוכחה 2: אי-שוויון ישיר

$$\lceil \frac{n}{2} \rceil = \lceil \frac{2^h - 1 + x}{2} \rceil = 2^{h-1} - 1 + \lceil \frac{x}{2} \rceil \geq 2^{h-1} - 1 + \frac{x}{2} = 2^h - 1 - \frac{x}{2} + x = n - \frac{x}{2} + x = n + \frac{x}{2}$$

$$\lceil \frac{n}{2} \rceil \geq n + \frac{x}{2} \quad \checkmark$$

שאלה 2 — מציאת k האברים הגדולים ביותר ב-Heap

שאלה 2.1 — האיבר הגדול בעדיפות

הוכחת הטענה/מונן ב-Heap מינימלי/מקסימלי נמצאים **איך מוצאים את האיבר הגדול?**

פתרון 2.1

מיקום: **בדיוק** ב**שורש** של **Max-Heap** (כי הוא מקיים את תכונת הערמה).

מחיר: $O(1)$ — פשוט קוראים את $A[1]$.

שאלה 2.2 — האיבר השלישי בגודלו

איך מוצאים את השלישי בגודלו?

פתרון 2.2

שלושה מועמדים לשלישי:

- **רק** שלושת השורש ובניו (אחד בגובה 1 ו-2 בגובה 2)
- **לא** את **האחרון מהרמה האחרונה**
- **לא** את **האחרון ברמה שלפני האחרונה**

שאלה 2.3 — k האיברים הגדולים ביותר

כיצד נמצא את k האיברים הגדולים ביותר בעץ בינארי מלא בגודל n?

פתרון 2.3 — אלגוריתם "Child Collection"

האיברים הגדולים **מופיעים תמיד** בראשית המערך — ולכן כל **הבנים** של האיברים הגדולים **חייבים להיות** גם הם בראש המערך — לאחרים.

האלגוריתם:

1. אתחל את התוצר עם האיבר הראשי
2. חזור k − 1 פעמים:
   - מצא את האיבר הגדול **ברמה ביותר** שבתוצר
   - הוסף את בניו לתוצר
   - הוצא אותו מהתוצר
3. החזר את התוצר שנבנה בסוף הלולאה

דוגמה: Heap = [44, 42, 33, 33, 31, 19, 27, 10, 26, 14]

שלב	תוכן	אפשר להמשיך?	בנים מועדים
התחלה	(0)	כן	44
1	(1, 7)	כן	42
2	(2, 2)	כן	35
3	(6, 5, 4, 3)	לא	33

סיבוכיות

- מדד האופטי המלא ב-k מדד ב-k מדד אחרונה מתוצרת ב-k מכניסים ב-$O(1)$ — נסיף ב-k אחרונה תלך האחרון ב-k אחרים
- אם מתחזקים את התוצר ב-**Max-Heap** ב-k מדד אחרונה מתוצרת ב-$O(\log k)$
- סה"כ ב-k פעולות — $O(k \log k)$

הערה חשובה

האלגוריתם לא **מוכיח את ה-Heap המלא** — כמו בדוגמה שבה הערך הגדול ביותר נמצא ברמה האחרונה.

שאלה 3 — פיתוח Queue עם שני Stacks

מוטיבציה

נתון **שני** מימושים של **Stacks** ואין מימוש של **Queue** (FIFO).

מטרה: להשתמש רק ב-Stacks ובפעולות Push, Pop, Top, Isempty כדי לממש Queue.

תזכורת

- **Enqueue** — הכנסה לתור (FIFO)
- **Dequeue** — הוצאה מהתור (FIFO)
- **Stack (LIFO)** — הוסיפו, שלפו ובדקו את האחרון

מימוש

נשתמש בשני S1 ו-S2 (לדוגמה).

(Enqueue(x

```
Push(S1, x)
```

משמש לדוגמה ב-$O(1)$.

()Dequeue

```
if S2 is empty:
    while S1 is not empty:
        Push(S2, Pop(S1))
return Pop(S2)
```

עד S2 ריק — מעבירים ואחר כך מוציאים מ-S2. לדוגמה ב-$O(n)$.

דוגמה מלאה צעד אחר צעד

פעולה	S1 (בסיס ביטים)	S2 (בסיס ביטים)	תוצאה
Enqueue(5)	[5]	[]	—
Enqueue(2)	[5, 2]	[]	—
Enqueue(3)	[5, 2, 3]	[]	—
()Dequeue	[]	[2, 3]	5 הוצאה + Pop
Enqueue(6)	[6]	[2, 3]	—
Enqueue(4)	[6, 4]	[2, 3]	—
()Dequeue	[6, 4]	[3]	2 הוצאה + Pop
()Dequeue	[6, 4]	[]	3 הוצאה + Pop
()Dequeue	[]	[4]	6 הוצאה + Pop

ניתוח סיבוכיות

- **Enqueue** — $O(1)$ — תמיד
- **Dequeue worst-case** — $O(n)$ — כאשר לדוגמה את כל 5,2 5,3
- **Dequeue amortized** — $O(1)$ — כל איבר מועבר בדיוק 5,2 **לכל היותר פעם אחת** לאחר כל איבר נכנס

סיכום התנועה

נקודות עקריות

- **מונה בינארי** — סה"כ n הפיכות ב-$O(1)$ כל אחת ו-n פעולות.
- **Binary Heap** — מערך + תכונת ערמה — almost-complete — $h = \lfloor \log_2 n \rfloor$, i-ה בן שמאלי $2i$, i-ה בן ימני $2i+1$, i-ה הורה $\lfloor i/2 \rfloor$.
- **פיתוח עם PercolateUp/Down** — $O(\log n)$ כל אחת.
- **פיתוח עם almost-complete** — $O(n)$ — חישוב מספר העלים ב-$\lceil n/2 \rceil$.
- **k האברים הגדולים** — ב-$O(k \log k)$ בעזרת Child Collection על פני k — ב-$O(k)$ בעזרת Heap מתוחזק.
- **Queue מ-2 Stacks** — Enqueue $O(1)$ — Dequeue $O(n)$ — מראי אנו ש-$O(1)$ לכל איבר אחת.