**ממ"ן 13**

1. בדומה לערימה בינראית, העץ של הערימה הD-ית מלא לחלוטין בכל רמותיו פרט אולי לאחרונה, המלאה משמאל ועד לנקודה מסויימת. לפיכך, נוכל לייצג אותו כמערך בדרך דומה לערימה בינראית, כך שהשורש מאוחסן בתא הראשון, ולאחריו האיברים של הרמה הבאה בעץ, משמאל לימן, עד הרמה האחרונה, בה כל העלים הקיימים מאוחסנים גם הם משמאל לימין.  
   על מנת להפוך את המערך חזרה לערימה D-ית נבצע פעולה הפוכה, ונשתמש בערך בתא הראשון כשורש, בערכים בתא השני עד התא D+1 כבניו, וכן הלאה, עד שנגיע לאיברים האחרונים, שהם העלים, ואותם נשבץ משמאל לימין כילדיהם של הרמה הלפני האחרונה, עד תום.  
   הכללת החישובים הנדרשים לפעולות על הערימה הd-ית על סמך ערימה בינארית (לפי פרק 6.1 בספר) תתבצע כדלהלן (הצבת d=2 תוכיח את נכונות החישוב לפי החישובים לערימה בינארית):
   1. מציאת אינדקס האב של האיבר ה-i :
   2. מציאת האינדקס של הבן הn-י (n=1,2,3…) של אינדקס i :
   3. האינדקס הראשון של הרמה ה-i-ית (כאשר השורש הוא רמה 0) :
   4. האינדקס הראשון של העלים :
   5. גובה הערימה : )ראה סעיף ב'(
2. באופן דומה לפתרון בעיה 6.1-2 עבור חישוב גובה ערימה בינארית בת n איברים, נבצע חישוב דומה עבור ערימה D-ית.  
   יהיה H גובה הערימה. מאחר והעץ המייצג את הערימה הוא עץ D-י מלא עד הרמה H-1, ולכל איבר בו יש D בנים, וברמה האחרונה יהיו k עלים. לפיכך, כמות האיברים בעץ תהיה שווה ל- .  
   מכך נובע שהגובה, כלומר, אורך המסלול הפשוט הארוך ביותר מהשורש לעלה כלשהו יהיה -

כלומר, הגובה של ערימה d-ית בת n איברים הוא .

כעת נסביר בקצרה על המימוש הפנימי של הערימה הd-ית אצלינו בקוד (המימוש נעשה בשפת python) –

הערימה הd-ית מיוצגת על ידי המחלקה DHeap, שיורשת מהמחלקה list, ומתאותחלת על ידי מערך של איברים והערך של d לערימה זו. בהתאם, מחלקת האב מאפשרת לקבל את כמות האיברים במערך (גודל הערימה) ואת היכולת לגשת ולקבוע ערכים לפי האינדקס שלהם. בנוסף, המחלקה עצמה מייצאת את התכונות heap\_size (גודל הערימה המשתנה), array\_length (גודל המערך ההתחלתי), וheight (גובה הערימה, חישובו מפורט בסעיף ב'). המחלקה מממשת פעולות פנימיות נפוצות בערימות, למשל SWAP בין ערכים, או הסרת הערך האחרון במערך.  
כמו כן, המחלקה מייצאת חישובים נפוצים, כמו שפירטנו בסעיף א'. לבסוף, מימשנו פעולות להדפסה למסך – בייצוג של רשימה (רשימה של רשימות המייצגות כל רמה בעץ), ובאילוסטרציה בתור עץ.  
בנוסף למחלקה של הערימה הd-ית, מימשנו מחלקה נוספת בשם GeneralAlgorithms שבה מימשנו אלגוריתמים נוספים הדרושים לפונקציונליות התכנית, למשל – BUILD-MAX-HEAP ו-MAX-HEAPIFY. בנספח א' למסמך צורפו כל המימושים בפסאודו-קוד לשגרות השונות שנדרשו לפתרון התרגיל.

מאחר ש MAX-HEAPIFY מרכזי לפונקציונליות של שאר הקוד, ננתח את הסיבוכיות שלו כדי להקל על הסעיפים הבאים.

1. בפתרון מימשנו את פעולה EXTRACT-MAX(A) על ערימת מקסימום d-ית (תחת השגרה *extract\_max*(heap)).
2. בפתרון מימשנו את פעולה INSERT(A, v) על ערימת מקסימום d-ית (תחת השגרה *insert*(heap,value)).
3. בפתרון מימשנו את פעולה INCREASE-KEY(A,i,k) על ערימת מקסימום d-ית (תחת השגרה *increase\_key*(heap,index\_to\_increase,new\_value)).
4. בפתרון מימשנו את פעולה EXTRACT(A,i) על ערימת מקסימום d-ית (תחת השגרה *extract*(heap,index\_to\_remove))

**נספח א' – מימושי הפעולות בפתרון**

|  |  |  |
| --- | --- | --- |
| השגרה | המימוש | הערות |
| BUILD-MAX-HEAP | BUILD-MAX-HEAP(A):  For i from A.get\_first\_leaf\_index downto 0:  MAX-HEAPIFY(A, i) | בדומה למימוש המוצג בפרק 6.3, המימוש של חישוב העזר מוגדר בסעיף א' |
| MAX-HEAPIFY | MAX-HEAPIFY(A, i):  Largest\_index = i  For child\_index from A.get\_nth\_child\_index(i, 1) to A.get\_nth\_child\_index(i, d):  If child\_index < A.HEAP\_SIZE and A[child\_index] > A[Largest\_index]:  Largest\_index = child\_index  If Largest\_index != i:  SWAP(A, i, Largest\_index)  MAX-HEAPIFY(A, Largest\_index) | בדומה למימוש המוצג בפרק 6.2 |
| EXTRACT-MAX | EXTRACT-MAX(A):  If A.HEAP\_SIZE < 1:  Error “heap underflow”  If A.HEAP\_SIZE == 1:  Return POP-LAST(A)  Max\_index = 0  SWAP(max\_index, A.HEAP\_SIZE - 1)  Popped\_node = POP-LAST(A)  MAX-HEAPIFY(A, 0)  Return Popped\_node | בדומה למימוש המוצג בפרק 6.5 |
| POP-LAST | POP-LAST(A):  Last\_index = A.HEAP\_SIZE - 1  Popped\_node = A[last\_index]  A.HEAP\_SIZE = A.HEAP\_SIZE -1  Return Popped\_node | פונקציית עזר |
| INSERT | INSERT(A, i):  A.HEAP\_SIZE += 1  If A.HEAP\_SIZE < A.length:  A[A.HEAP\_SIZE - 1] = Null  Else:  A.append(Null)  For j from A.HEAP\_SIZE - 1 downto 1:  A[j] = A[j - 1]  A[0] = i  MAX-HEAPIFY(A, 0)  For Index From 0 to A.HEAP\_SIZE:  If A[index] == i:  return index | בדומה למימוש המוצג בפרק 6.5 |
| INCREASE-KEY | INCREASE-KEY(A, i, v):  If A. HEAP\_SIZE < i:  Error "Heap overflow.”  If i < 0:  Error "Heap underflow."  If A[i] >= v:  Return  else:  A[i] = v  While i > 0:  Parent\_index = A.get\_parent\_index(i)  If A[i] > A[Parent\_index]:  A.swap(i, Parent\_index)  i = Parent\_index  Else:  Return | בדומה למימוש המוצג בפרק 6.5 |
| EXTRACT | EXTRACT (A, i):  If A.HEAP\_SIZE < i:  Error “heap overflow”  If i < 0 or A.HEAP\_SIZE < 1:  Error “heap underflow”  If A.HEAP\_SIZE == 1:  Return POP-LAST(A)  SWAP(i, A.HEAP\_SIZE - 1)  Popped\_node = POP-LAST(A)  MAX-HEAPIFY(A, i)  Return Popped\_node |  |

בכללי TODO:

* יש להסביר מהו המבנה הפנימי של הערימת-d שלכם - DONE
* להראות מספר דוגמאות הרצה- כדי להבהיר איך עובד הממשק – TODO שלי
* לתאר את השגרות השונות- להסביר את הרעיון הכללי, ולפרט את הסיבוכיות – TODO סעיפים ג-ו

הוראות מהפורום:

* כתבו את המסמך הנלווה באופן שבו הוא יכול לשמש את המנחה לבדיקה ללא צורך בהרצה (כך שכביכול ההרצה היא רק לבדיקה שמה שכתוב בקובץ המלווה הוא אכן נכון)
* d הוא לא דווקא const, ולכן יש לתאר זמן הריצה כפונקציה של n,d
* לחשב סיבוכיות של כל אחת מהפונקציות (גם של בניית ערימה)
* לגבי א' –
  + איך ניתן למצוא את האינדקס של האב.
  + איך ניתן למצוא את האינדקס של הבנים.
  + אילו תאים במערך ייצגו את גובה h כלשהו בערימה.
  + מהו גובה העירמה כתלות בn.
  + באילו אינדקסים ימצאו העלים.