ממן 13

מגישים: רועי שטרן ושקד אורן

במסמך הבא האינדקס של האיבר הראשון בערימה הוא 0

הסבר על הפרוייקט שהוגש

הפרוייקט מכיל כמה קבצים

1. Heap.py – מחלקה בשם MaxMinHeap שמכילה את כל הפונקציונליות שמתוארת במסמך הנ"ל.
2. Menu.py – אחראי על ניהול התפריט והקלט מהמשתמש
3. HeapUtils.py – מכיל כמה פונקציות עזר לטובת מימוש הפונקציות והתפריט.
4. Tests/ - תיקייה המכילה כ1000 קבצי קלט שונים שבדקנו את הקוד שלנו מולו

בבדיקת הקוד השתמשנו ב1000 קבצים שעליהם ביצענו כמות אקראית של פעולות אקראיות כפי שהמשתמש היה מבצע ובדקנו את תקינות הערימה לאחר כל שינוי.

הוראות להרצת התוכנית

על מנת להריץ את התוכנית יש להקליד בשורת terminal בתיקיית הפרוייקט, או לפתוח את הקובץ start.bat על מנת להריץ את התוכנית אוטומטית.

python main.py

התוכנית מריצה תפריט אשר מנהל אובייקט שנקרא MaxMinHeap אשר מכיל את כל הפונקציונליות המתוארות במסמך זה.

על מנת לאתחל heap מקובץ יש לקרוא לאופציה build\_heap בתפריט התוכנית ואז להכניס את מיקום הקובץ.

התוכנית מניחה שקובץ הקלט מכיל קלט תקין.

על מנת להשתמש בתוכנית עם קובץ בדיקה ניתן להקליד את הפקודה הבאה. כך התוכנית תאותחל עם הקובץ הנתון.

python main.py ‘path\_test\_file’

סעיף א:

**מציאת ערך המקסימום בערימת Max Min**

רעיון – להחזיר את הערך של האיבר הראשון

נכונות בקצרה – האיבר המקסימלי הוא האיבר במקום ה0 במערך, משום שזהו האיבר שנמצא בגובה העץ הזוגי הראשון ולכן כל שאר הצאצאים שלו(כל שאר העץ) קטנים ממנו. לכן בכדי

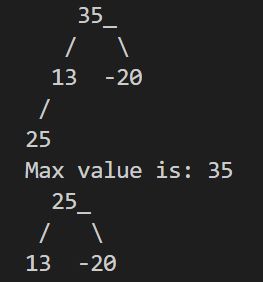
סיבוכיות – פעולה אחת של החזרת ערך האיבר הראשון – O(1)

אלגוריתם –

FIND\_MAX(Heap):

return Heap[0]

דוגמאות הרצה –



**מציאת ערך המינימום בערימת Max Min**

רעיון – להחזיר את הערך המינימלי, בדיקה בין שני האיברים בגובה העץ האי זוגי הראשון, ולהחזיר את הערך הקטן מביניהם

נכונות בקצרה – האיבר המינימלי הוא איבר שנמצא בגובה העץ האי זוגי הראשון לכן בגובה העץ 1 ולכן כל שאר הצאצאים שלו(כל שאר העץ) גדולים ממנו. לכן בכדי למצוא את האיבר המינימלי משני האיברים שנמצאים בגובה העץ 1

סיבוכיות – פעולות בדיקת מינימום בין שני איברים באינדקס ידוע ופעולה של החזרת ערך האיבר הקטן – O(1)

אלגוריתם –

FIND\_MIN(Heap):

If Heap.size() = 1:

return Heap[0]

if Heap.size() = 2:

return Heap[1]

if Heap.size() >= 3:

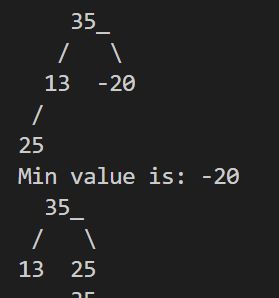
if Heap[1] >= Heap[2]:

return Heap[2]

else:

return heap[1]

דוגמאות הרצה –



**Push-up**

רעיון – לתקן את הערימה כלפי מעלה, ללא חשיבה על האם הHEAP נשבר כלפי מטה. כל עוד יש לי אבא, אבדוק אם האבא שלי בגובה MAX, אם כן אז אבדוק האם אני גדול ממנו ואחליף אותי איתו, לאחר מכן אבדוק אותי שוב מול האבא שלי שבגובה MIN ואבדוק אם אני קטן ממנו ובמידה וכן אחליף בינינו, בהתאמה, למקרה שבו האבא שלי הראשון אם קיים בגובה MIN.

נכונות – עד האינדקס I שממנו אנחנו עושים pushup הערימה תקינה כלומר A[1..i] אולי חוץ ממקרה אחד שבו A[i] מפר את הלוגיקה של parent(A[i])

עבור כל ריצה של הלולאה זה יכול להתקיים, אך לאחר ההחלפה ביניהם וכך הparent(i) נהפך לi, הערימה חוזרת למצב תקין חוץ מאולי parent(i) החדש.

בסיום הלולאה אין יותר parent לכן כל הערימה בהתחלה תקינה

סיבוכיות – המקרה הגרוע ביותר הוא שאנחנו עולים מתחילתה עד סופה כלומר פעולת השוואה והחלפה לכן מתבצעות logn החלפות, לכן הסיבוכיות היא O(logn)

אלגוריתם –

PUSH\_UP(Heap, i):

curr\_parent = i

while parent(curr\_parent) != i do

curr\_parent = parent(curr\_parent)

if curr\_parent is on max level do

if Heap[i] > Heap[curr\_parent] do

swap(Heap, i, curr\_parent)

i = curr\_parent

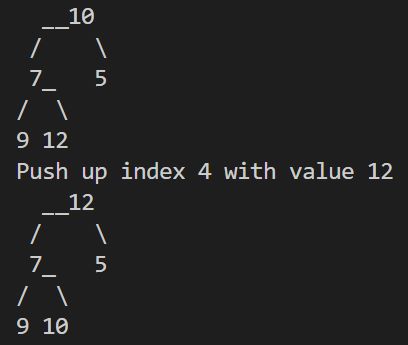
if curr\_parent is on min level do

if Heap[i] < Heap[curr\_parent] do

swap(Heap, i, curr\_parent)

i = curr\_parent

דוגמאות הרצה –



**Heapify**

רעיון – לתקן את הערימה כלפי מטה, נתונה לי ערימה ובאינדקס i שעליו אני פועל, יכול להיות שהוא מפר את הגדרות הערימה, לכן אשווה אותו מול הילדים והנכדים שלו, במידה ואשווה מול בן ישיר, אחליף ביניהם וכך הערימה תהפוך למצב תקין, אם אחליף עם נכד שלו, אצטרך לבדוק אם יש לתקן גם מול בן ישיר ואז אמשיך לקרוא לheapify עבור המיקום שהחלפתי את הערך של i

נכונות – ההנחה היא שכאשר מתבצעת קריאה לheapify העצים הבינארים המושרשים בleft(i) ובright(i) הן ערימות תקינות. לכן נראה נכונות עבור max בה"כ

1. כאשר i בגובה max
   1. אם i הוא המקסימום הרי הערימה תקינה.
   2. אם הבן הישיר של i הוא המקסימום, מנכונות תת הערימה של הבן הישיר אין לתת עץ של בנו הישיר ילדים. לכן ההחלפה ביניהם מתקנת את כל הערימה.
   3. אם אחד מנכדיו באינדקס j הוא המקסימום, נחליף ביניהם. כלומר יש לנו את איבר במיקום i=a1, הנכד שנקרא לו a3 ואביו של הנכד שנקרא לו a2. מנכונות תת הערימה של a2 כל שאר הצאצאים, כלומר מהנכד והלאה של a2 קטנים ממנו.
      1. אם a1<a2 נחליף ביניהם וכך נשמור על כך שכל שאר הצאצאים קטנים גם מa1 כי a1<a2<(rest of children), נקרא שוב לheapify עם אינדקס j כדי לתקן את ערימת הmax כי ערימת הmin תקינה.
      2. אם a1>a2 נקרא שוב לheapify עם אינדקס j כדי לתקן את ערימת הmax כי ערימת הmin תקינה.

כך עד שנגיע לתת עץ האחרון שאותו נתקן וכל הערימה תהיה תקינה

סיבוכיות – במקרה הגרוע ביותר, נבצע 4 פעולות (שתי השוואות ושתי החלפות) log4(n)

אלגוריתם –

Heapify(Heap, i):

If left(i) > Heap.size():

Return

If i is on max level do:

Max\_index = get\_index\_of\_max\_children\_and\_grandchildren(i)

If Heap[max\_index] > heap[i] do

Swap(Heap, I, max\_index)

If max\_index is grandchild of i do

If Heap[max\_index] < Heap[parent(max\_index)] do

Swap(Heap, max\_index, parent(max\_index))

Heapify(max\_index)

If i is on min level do:

min\_index = get\_index\_of\_min\_children\_and\_grandchildren(i)

If Heap[min\_index] < heap[i] do

Swap(Heap, I, min\_index)

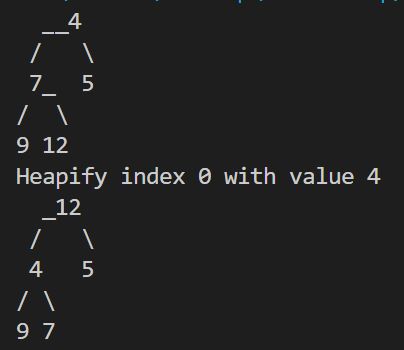
If min\_index is grandchild of i do

If Heap[min\_index] > Heap[parent(min\_index)] do

Swap(Heap, min\_index, parent(min\_index))

Heapify(min\_index)

דוגמאות הרצה –



**הכנסת איבר לערמה**

רעיון – להוסיף את האיבר למקום האחרון בערמהולהגדיל את המערך ולאחר מכן לסדר אותו למעלה בעזרת push\_up

נכונות – הוספת איבר למיקום האחרון בערמה יכול להרוס רק את נכונות תתי העצים המכילים את האיבר האחרון, השגרה push-up מסדרת איבר ביחס לכל האבות שלו, מכיוון שאין לאיבר האחרון בערמה ילדים, לאחר הקריאה לpush-up הערמה תהיה תקינה.

סיבוכיות – הוספת האיבר מתבצעת ב O(1) וסיבוכיות השגרה push-up היא ולכן סיבוכיות הכנסת האיבר לערמה היא

אלגוריתם –

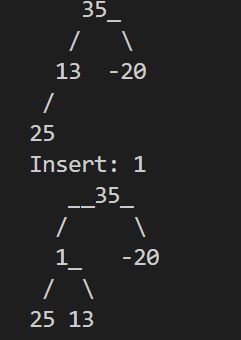
INSERT (Heap, value):

Heap.size() <- Heap.size() + 1

Heap[heap.size()] <- value

Push\_up(Heap, heap.size())

דוגמאות הרצה -

**

**הסרת איבר מהערמה**

רעיון – להחליף בין האיבר שנדרש להסיר מהרשימה לבין האיבר האחרון ברשימה, למחוק את האיבר האחרון מהרשימה ולאחר מכן לתקן את האיבר שהוחלף גם למעלה בעזרת push-up וגם למטה בעזרת heapify-node.

נכונות – הסרת האיבר האחרון בערמה אינה משפיעה על הערימה ולכן הבעיה יכולה להיות רק באיבר שהוחלף, תיקון הערימה למעלה יגרור שהאיברים שנמצאים מעל האיבר המוחלף יהיו תקינים אך הילדים שלו לא בהכרח יהיו תקינים. שימוש בheapify-node יתקן גם את הילדים של האיבר שהוחלף.

סיבוכיות – החלפת האיבר ומחיקה שלו מתבצעת ב O(1), הסיבוכיות של push-up ושל heapify-node היא ולכן הסיבוכיות של הסרת איבר מהערמה היא

אלגוריתם –

DELETE (Heap, i):

If i <0 or i > heap.size()

Return

Last\_index <- Heap.size()-1

Swap(index, last\_index)

Value = Heap[last\_index]

Heap.size() <- Heap.size() -1

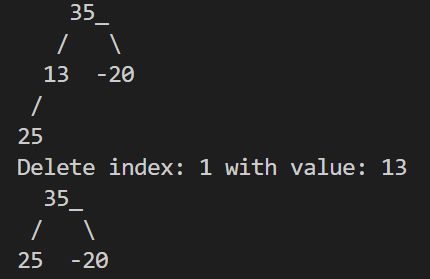
If index == last\_index

Return

Push\_up(index)

Heapify(index)

דוגמאות הרצה -



**בניית ערמה build heap**

רעיון – לעבור מהצומת שאינה עלה האחרונה ועד לאיבר הראשון בערמה ועל כל אחד מהצמתים האלה להפעיל heapify-node

נכונות – נוכיח באינדוקציה.

אתחול: בכניסה לאיטרציה הראשונה, כל אחד מהצמתים באינדקס שגדול מi הוא עלה ולכן הוא בעצמו ערימה תקינה.

תחזוקה: כמו שאמרנו לכל בן של צומת i הוא בעצמו שורש של ערימה תקינה. ולכן אלגוריתם hepaify מתאים לשימוש עבור כל צומת i, ובפרט אינה משנה את העובדה שהצמתים באינדס שגדול מi הם שורש של ערימה תקינה. לכן בהמשך ריצת הלולאה מתקיים כל התנאים שציינו.

סיום: סיום התהליך כאשר מתבצע לאחר מעבר על שורש הערימה המקורית ולפי שמורת הלולאה כל הצמתים הינם שורשים של ערימות תקינות ולכן על הערימה תקינה.

סיבוכיות – הוכחה זהה להוכחה בספר בעמוד 112 עבור build-max-heap

אלגוריתם –

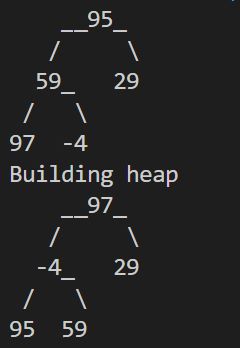
BUILD\_HEAP (Heap):

Last\_not\_leaf = |\_(Heap.size()/2)-1\_|

For i <- last\_not\_leaf to 0 do:

Heapify(Heap,i)

דוגמאות הרצה -



**הוצאת איבר מקסימלי מהערימה**

רעיון – להחליף את האיבר המקסימלי במיקום 0 עם האיבר האחרון להסיר את האיבר האחרון (המקסימלי) ולקרוא לheapify עבור האיבר במיקום 0(האיבר האחרון)

נכונות – האיבר המקסימלי הוא האיבר במיקום ה0 לכן בכדי להוציא אותו נעביר אותו לסוף הרשימה על ידי החלפה עם האיבר שמקום האחרון במערך, ונקטין את המערך באחד. בכדי לתקן את הערימה, נקרא לheapify עם האיבר במקום 0.

סיבוכיות –ההחלפה מתבצעת בO(1) ביצוע הheapify מתבצע בO(logn)

אלגוריתם –

EXTRACT\_MAX (Heap):

Max\_index = 0

Last\_index <- Heap.size() -1

Value <- Heap[max\_index]

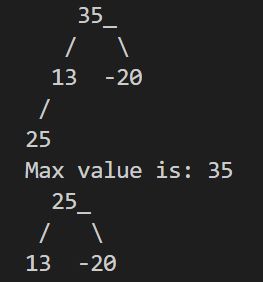
Swap(Heap, max\_index, last\_index)

Heap.size() <- Heap.size() -1

Heapify(max\_index)

Return value

דוגמאות הרצה -

****

**הוצאת איבר מינימלי מהערימה**

רעיון – דומה להוצאת איבר מקסימלי מהערימה, רק שנבצע פעולת השוואה יחידה בין שני האיברים במיקום 1 ו2, כלומר בשלב הmin נמצא את הmin כמו שהסברתי ב **מציאת ערך המינימום בערימת Max Min**  ולאחר מכן נחליף בין איבר זה לאיבר האחרון ונקטין את המערך ב1 ונקרא לheapify עם האינדקס של האיבר הקטן ביותר

נכונות – לאחר מציאת האיבר הקטן ביותר נחליף אותו עם האיבר האחרון בערימה ונקטין את המערך ב1, כלומר יש לנו איבר אחד שהוא אינו תקין בערימה, לכן נקרא לheapify עם האינדקס שלו ונתקן את הערימה.

סיבוכיות – מציאת האיבר המינימלי הוא O(1) ההחלפה מתבצעת גם בO(1) וheapify מתבצע בO(log(n))

אלגוריתם –

EXTRACT\_MIN (Heap):

Min\_index = 0

If right(0) >= Heap.size() do

Min\_index <- 1

If Heap[1] > Heap[2] do

Min\_index <- 2

Last\_index <- Heap.size() -1

Value <- Heap[min\_index]

Swap(Heap, min\_index, last\_index)

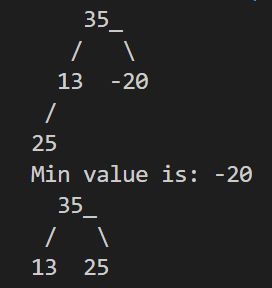
Heap.size() <- Heap.size() -1

Heapify(min\_index)

Return value

If

דוגמאות הרצה -

****

**מיון ערימה**

**מההבנה שלנו מיון הערמה מתייחס לערמה הקיימת ולא לקבלת מערך כפרמטר לשגרה, במידה והדרישה היא שונה, ההתאמה בקוד היא פשוטה וניתנת לביצוע מבלי שינויים בקוד מעבר לשינוים בשגרה עצמה.**

רעיון – נקרא לextract\_max n פעמים וכך נקבל את המערך ממוין.

נכונות – אם נוציא את האיבר המקסימלי בכל פעם מהערימה עד שלא יהיו איברים נוציא אותם בסדר מהגדול לקטן.

סיבוכיות – ביצוע הוצאת האיבר המקסימלי מתבצעת בO(logn) ופעולה זו נבצע n פעמים לכן O(nlogn)

אלגוריתם –

SORT (Heap):

While Heap.size() do

Print(extract\_max())

דוגמאות הרצה –

