מבני נתונים קיץ 2019
תרגיל בית 3

תאריך הגשה: 18 בספטמבר 2019

את העבודה יש להגיש במערכת ההגשות submission system
על העבודה להיות מוגשות בכתב יד ברור סרוק או הקלדה במחשב

- הנכם נדרשים לנסח **תשובות ברורות וקצרות**
- עליכם לנסח תחילה את האלגוריתם במילים בכמה משפטים
- לאחר מכן בפסאדו קוד (לא בשפת תכנות אלא כמו שלמדנו בכיתה, ללא פרטים טכניים)
- לאחר מכן להסביר את נכונותו (אין דרישה להוכיח אך ההסבר חייב להיות **מאוד משכנע**)
- ולבסוף לנתח את זמן הריצה

שאלות לגבי העבודה ניתן לשאול בפורום של העבודה הנמצא באתר של הקורס.

שאלה 1:

נתונה רשימת זוגות $(a_1,b_1),\dots,(a_n,b_n)$ כאשר a_i הינו תאריך כניסה של בן אדם i לעיר ו b_i הינו תאריך יציאה של בן אדם i מהעיר. הציעו אלגוריתם יעיל ככל הניצן המחשב מהו מספר מקסימלי של אנשים שנוכחו בעיר בו זמנית.

שאלה 2:

סעיף א:
נתון מערך לא ממויין בגודל n. הציעו אלגוריתם להדפסת k האיברים הקטנים במערך בזמן O(n) בממוצע.
סעיף ב:
אלגוריתם מסוים צריך לבצע n פעולות על מבנה נתונים מסוג מילון (מבנה שתומך בפעולות insert, delete ו- find). $n/10$ מהפעולות הן מסוג insert, והשאר מסוג delete ו- find. לא ידוע בדיוק אופן החלוקה של הפעולות בין delete ו- find. מוצעים למילון שני מימושים. במימוש הראשון הזמן הגרוע ביותר לפעולת ה- insert הוא $\theta(n)$ והעלות לשיעורין (amortized) של כלל הפעולות היא $\theta(1)$. במימוש השני הזמן הגרוע ביותר לפעולת ה- insert הוא $\theta(log\ n)$ וגם העלות לשיעורין (amortized) של כלל הפעולות היא $\theta(log\ n)$. איזה מהמימושים תציעו לאלגוריתם?
סעיף ג:
נדרש לממש 3 מחסניות S_1,S_2,S_3 בעזרת מערך בגודל **6N**, כאשר: סך כל האיברים שניתן להכניס ל- 3 המחסניות הוא לכל היותר 5N. הזיכרון הנוסף המותר הוא לכל היותר O(1). על המימוש לעמוד בדרישה הבאה: לכל רצף של הכנסות ומחיקות מהמחסניות הזמן לשיעורין (amortize) לפעולה הוא O(1) כאשר המחסניות מתחילות ריקות והמקום עבור המערך הוקצה כבר.

סטודנט הציע את הפתרון הבא: נגדיר שמחסנית S_1 תתחיל מתחילת המערך (אינדקס 0, והאינדקס גדל בכל הכנסה), S_2 תתחיל מהאמצע (אינדקס 3N, והאינדקס גדל בכל הכנסה) , ו-S_3 תתחיל מסוף המערך (אינדקס 6N, והאינדקס קטן בכל הכנסה). כאשר רוצים להכניס איבר למחסנית מסוימת ואין מקום במערך מזיזים את S_2 (המחסנית האמצעית) לכיוון המחסנית שנקודת הסיום (אינדקס במערך) שלה רחוקה יותר. כלומר, אם הכנסת איבר תגרום להתנגשות בין S_1 ל-S_2 נזיז את S_2 לעבר S_3, ואם הכנסת איבר תגרום להתנגשות בין S_2 ל-S_3 נזיז את S_2 לעבר S_1.

עליכם להסביר את אופן הטיפול בהתנגשות בין מחסניות, כלומר להגדיר את אופן הזזת המחסנית S_2 כך שפתרון הסטודנט יעמוד בדרישות השאלה ולנתח את זמן הריצה.

שאלה 3:

נתון גרף לא מכוון $G = (V, E)$, שני צמתים $s, t \in V$ וקשת $e = (u1,u2) \in E$.

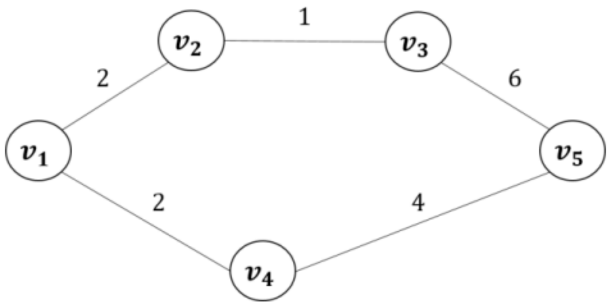
א. (6 נק') תארו אלגוריתם יעיל ככל האפשר הבודק האם הקשת e נמצאת על כל המסלולים הקצרים ביותר בין s ל t בG. נתחו את סיבוכיות זמן הריצה של האלגוריתם שלכם.

ב. (6 נק') תארו אלגוריתם יעיל ככל האפשר הבודק האם הקשת e נמצאת על מסלול קצר ביותר כלשהו בין s ל t בG. נתחו את סיבוכיות זמן הריצה של האלגוריתם שלכם.

שאלה 4:

א. הוכח או הפרך: אם $G = (V, E)$ גרף מכוון וחסר מעגלים, אזי בכל ריצת DFS על G, יווצר אותו יער.

ב. תארו אלגוריתם, אשר בהינתן גרף ממושקל ולא מכוון $G = (V, E)$, זוג קדקודים $s, t \in V$ $(s \neq t)$, ומספר m, מכריע האם קיים מסלול בין s ל-t אשר סכום צלעותיו הינו לכל היותר m.
לדוגמא:



עבור $s = v_1, t = v_5, m = 5$ ישיב האלגוריתם שלא קיים מסלול העונה לדרישות,
ואילו עבור $s = v_3, t = v_4, m = 5$ ישיב האלגוריתם בחיוב.

הרחיבו באילו מבני נתונים אתם משתמשים ונתחו את זמן ריצת האלגוריתם.

שאלה 5:

בהינתן קבוצת המספרים S המכילה n ערכים, עליכם לממש מבנה נתונים התומך בפעולות הבאות:

שם הפעולה	תיאור הפעולה	זמן הריצה הרצוי
$init(S)$	אתחול המבנה בקבוצה S המכילה n ערכים	$O(n)$
$insert(x)$	הכנסת הערך x למבנה הנתונים	$o(\log n)$
$find\_min()$	החזרת הערך המינימלי במבנה	$O(1)$
$find\_max()$	החזרת הערך המקסימלי במבנה	$O(1)$
$extract\_min()$	הוצאת האיבר המינימלי במבנה	$O(\log n)$
$extract\_max()$	הוצאת האיבר המקסימלי במבנה	$O(\log n)$

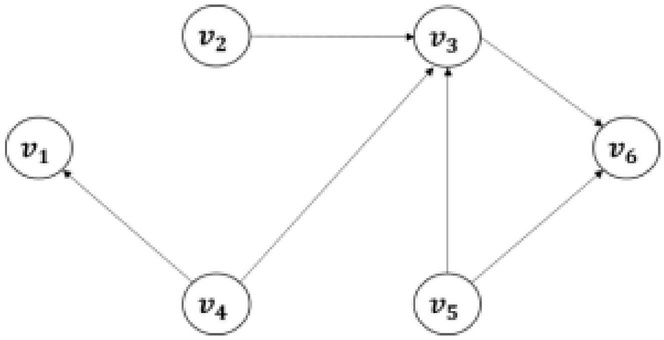
א. הציעו מימוש המשתמש בשתי ערימות.

ב. הציעו מימוש המשתמש **בערימה אחת בלבד**. ניתן להשתמש ב-$O(1)$ זיכרון נוסף עבור כל אחד מאיברי הערימה.

שאלה 6:

בהינתן גרף מכוון $G=(V,E)$, קבוצה $C \subseteq V$ תקרא **קבוצת ליבה** אם לכל $u,v \in C$, מתקיים ש-$(u,v) \notin E$, וכן לכל $v \in V \setminus C$ קיים $u \in A$ כך ש-$(u,v) \in E$.

לדוגמא:



בגרף זה הקבוצה $\{v_2,v_4,v_5\}$ הינה קבוצת ליבה, בעוד $\{v_4,v_5\}$ אינה קבוצה ליבה, וכך גם הקבוצה $\{v_1,v_2,v_4,v_5\}$.

א. הציעו אלגוריתם המוצא קבוצת ליבה בגרף מכוון חסר מעגלים בזמן $O(|V|+|E|)$. הוכיחו את נכונות האלגוריתם ונתחו את סיבוכיותו.
<u>הדרכה:</u> אתם רשאים להניח שלכל גרף חסר מעגלים קיימת קבוצת ליבה. נסמן ב-C^* את הקבוצה אשר הוחזרה ע"י האלגוריתם. עליכם להוכיח ש-C^* מקיימת את הדרישות הבאות:
  i. לכל $v \in V \setminus C^*$, קיים $u \in C^*$ כך ש-$(u,v) \in E$.
  ii. לכל $u,v \in C^*$, מתקיים ש-$(u,v) \notin E$.

ב. הוכיחו או הפריכו: לכל גרף מכוון קיימת קבוצת ליבה.

שאלה 7:

נתונה קבוצה של n זוגות , כאשר כל זוג מייצג קשר (אב, ילד), (אם, ילד), או (בעל, אשה).

הציעו מבנה נתונים כך שבהנתן זוג אנשים A ו B ניתן יהיה לקבוע באופן יעיל האם יש ביניהם קשר משפחתי.

שאלה 8:

יהא A מערך המכיל n מספרים טבעיים בתחום $[1,\dots,k]$ (k אינו קבוע).

א. עבור כל אחד מהמקרים הבאים, צינו את שיטת המיון היעילה ביותר ונתחו את זמן ריצתה:
  i. $k=O(2^n)$.
  ii. $k=O(2^{\log^3 n})$.
  iii. $k=O(n^5)$.

ב. יש לקבוע האם קיימים אינדקסים שונים i,j כך ש-$A[i]+A[j]=k$. בהנחה <u>**שלא ניתן לשנות את A**</u>, הציעו אלגוריתם לפתרון הבעיה בהתאם להגבלות בכל סעיף.
  i. זמן הריצה של האלגוריתם הוא $O(n^2)$ וניתן להשתמש ב-$O(1)$ זיכרון נוסף.
  ii. זמן הריצה של האלגוריתם הוא $O(n\log n)$ וניתן להשתמש ב-$O(n)$ זיכרון נוסף.
  iii. זמן הריצה של האלגוריתם הוא $O(n)$, ניתן להשתמש ב-$O(k)$ זיכרון נוסף, וכמו-כן $k<\sqrt{n}$.

שאלה 9:

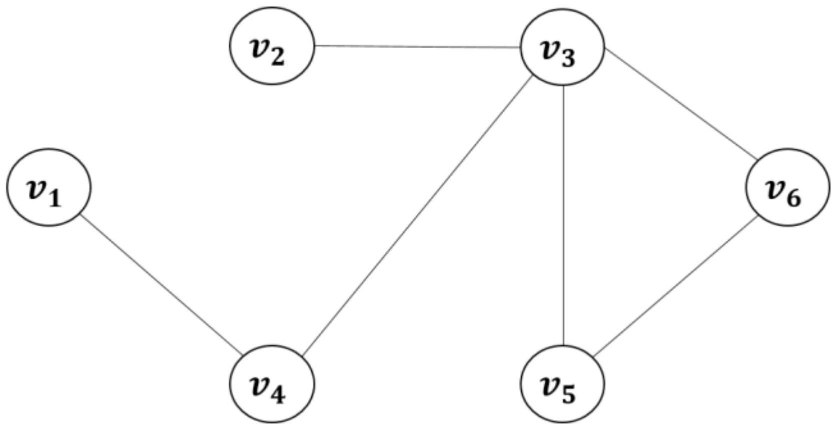
הינכם מקבלים כקלט n מספרים ממשיים השונים זה מזה, ואת החציון של ערכים אלו. תארו מבנה נתונים התומך בפעולות הבאות:

שם הפעולה	תיאור הפעולה	זמן הריצה הרצוי
$init()$	אתחול המבנה עבור n הערכים הנתונים	$O(n)$
$insert(x)$	הוספת איבר חדש בעל הערך x (ניתן להניח כי זהו אכן ערך חדש במבנה)	$O(\log n)$
$median()$	החזרת החציון הנוכחי מבין כל הערכים המופיעים במבנה	$O(1)$
$delete\_median()$	מחיקת החציון הנוכחי מבין כל הערכים המופיעים במבנה	$O(\log n)$

שאלה 10:

נגדיר גרף שכבות בצורה הבאה: גרף $G=(V,E)$ לא מכוון עבורו קיימת פונקצית שכבות $f: V \to \{1,2,\dots,k\}$ המתאימה לכל צומת קומה בין 1 ל-k.

לדוגמא:



כאשר $f(v_1)=1$, $f(v_2)=f(v_4)=2$, $f(v_3)=f(v_5)=3$, $f(v_6)=4$. כלומר G שבדוגמא הינו גרף בן 4 שכבות.

כתבו אלגוריתם שבהנתן גרף לא מכוון $G=(V,E)$, פונקצית שכבות f עם k שכבות ושני קדקודים $s,t \in V$, מוצא מסלול קצר ביותר מ-s ל-t אשר עובר בקדקוד אחד לפחות שנמצא בקומה ה-k. על האלגוריתם לרוץ בזמן $O(|V|+|E|)$. שימו לב שיתכן והמסלול הנבחר יכיל את אותו הקדקוד/הצלע יותר מפעם אחת.

בדוגמא שעיל, אם $s=v_1, t=v_6$, נקבל שהמסלול הקצר ביותר העובר דרך השכבה ה-k (כלומר שכבה מספר 4) הינו (v_1,v_4,v_3,v_6), ואילו אם $s=v_2, t=v_3$ אזי המסלול המתאים הינו (v_2,v_3,v_6,v_3).

Assignment 3 - Data Structures summer 19

Segev Nagar, Almog Lamay

שאלה 1

פירוט האלגוריתם

- נבנה מערך בגודל 2n, נעבור על רשימת הזוגות ונכניס את תאריך הכניסה ואת תאריך היציאה למערך באופן הבא:
  - כל תאריך נמיר לפורמט הבא: DDMMYY
  - נוסיף שדה בוליאני המציין אם התאריך הינו תאריך כניסה או יציאה
- כעת הבטחנו כי מס' הספרות של כל איבר במערך הוא 6, ומשאלה שעלתה בפורום התאריכים חסומים ע"י קבוע כלשהו k, לכן ניתן למיין את המערך באמצעות Radix.
- נתחזק משתנה counter ו-max, ונרוץ על המערך. כל פעם שנתקלים בתאריך כניסה: counter++, ו-counter-- כשנתקלים בתאריך יציאה. בכל העלאה של הקאונטר נבדוק האם הוא גדול מ-max ונעדכן את max בהתאם.
- max מסמל את כמות האנשים המקסימלית ברגע נתון בעיר, לכן נחזיר אותו בסיום

זמן ריצה

- עדכון המערך: O(2n)
- מיון Radix עולה O(d(k+b) = O(6(n+10) = O(n)
  (k קבוע, נתייחס אליו כאל O(n))
- מעבר נוסף על המערך: O(2n) = O(n)

Total of O(n) +O(n) + O(n) = O(n)

פסאודו

1. A = new empty array in size 2n
2. Scan pairs, reformat dates and insert to A
3. Perform Radix sort on A
4. for i = 0 to 2n-1:
   4.1. if A[i].isEntryDate = True
      4.1.1. counter++
      4.1.2. if counter > max
         4.1.2.1. max = counter
   4.2. else
      4.2.1. counter--
5. return max

שאלה 2

סעיף א'

פירוט האלגוריתם

- נמצא את האיבר ה-k בגודלו באמצעות Deterministic Select, נסמנו x
- נסרוק את המערך למציאת האינדקס של x במערך המקורי.
- נשתמש בפונקציית partition עם האינדקס של x שיהווה את ה-pivot.
- נדפיס את איברי המערך מההתחלה עד שמגיעים ל-x.

זמן ריצה

Total: O(n) + O(n) + O(n) + O(n) = O(n)

פסאודו

```
PrintKlittle(A, k)
        pivot = determinticSelect(A, k)
        for i = 0 to n
                if A[i] = pivot
                        pivotIndex = i
        partition(A, 0, A.len-1)        \\ Set pivot as A[pivotIndex]
        while A[i] != pivot
                print A[i]
                i++
```

<u>סעיף ג'</u>

- **אבחנה 1:** בעת התנגשות, יש לכל הפחות N מקומות פנויים מעברה השני של s2.
זאת משום שמס' האיברים המקסימלי במחסנית הינו 5N - לכן לכל הפחות ישנם N מקומות פנויים במחסנית, ובהכרח כל המקומות הפנויים יהיו בין S2 והמחסנית שלא התנגשה עימה.
- **נגדיר את פעולת ההזזה:** בכל הזזה S2 נעה N/2 מקומות בכיוון המקומות הפנויים.
- **אבחנה 2:** לאחר כל הזזה, יש להכניס לפחות N/2 איברים נוספים כדי שתתבצע הזזה נוספת. זאת משום שבמקרה הקצה בו יש N איברים פנויים בלבד במערך, לאחר שנזיז N/2 מקומות את S2, היא תהיה במרחק N/2 ממחסנית S1 וממחסנית S3. לכל הפחות נכניס N/2 איברים מאותו צד של S2 עד שתיווצר שוב התנגשות.

נעבוד לפי הנחה שהזזה, מחיקה והכנסה של איבר כולן עולות O(1), ונפתור בשיטת האסימונים:

- נקצה 11 אסימונים לפעולת הכנסה, ולפעולת הוצאה לא נקצה דבר.
- כל פעולת הכנסה מבזבזת אסימון אחד, לכן סה"כ נוסיף לקופה 10 אסימונים בכל הכנסה.
- מאבחנה 2, נחסוך בין התנגשות להתנגשות לכל הפחות 5N = 10 * N/2 אסימונים.
- ההזזה היקרה ביותר שנעשה לאחר התנגשות היא כאשר S2 בגודלה המקסימלי - כלומר בגודל 5N. זה הרגע הבטחנו כי תמיד יהיה לנו מספיק אסימונים לשלם הזזה כזו.
- נשים לב שפעולת ההוצאה אומנם עולה לנו אסימון אחד, אך גם מגדילה ב-1 את מס' ההכנסות עד להתנגשות. כל הכנסה כזו מכניסה לנו כאמור 10 אסימונים לקופה, לכן פעולת ההוצאה לא מזיקה.

<u>ניתוח זמן ריצה לשיעורין</u>
לכל היותר ביצענו n הכנסות במסגרת n הפעולות שלנו, לכן סך העלות הוא: 11n.
נחלק ב-n ונקבל שהעלות לשיעורין של כל פעולה הינה O(1) כנדרש.

שאלה 3

סעיף א'

פירוט האלגוריתם

- נריץ סריקת BFS מקודקוד S ונאחסן במשתנה את המרחק המינימלי השמור ב-t
- נסיר את הקשת e, נאתחל את שדות כל הצמתים, ונבצע שוב סריקת BFS מקודקוד s, ונשווה את המרחק המינימלי החדש בקודקוד t למרחק הישן השמור במשתנה:
    - ערך חדש > ערך ישן: משמע שכל המסלולים המינימליים בהכרח עברו בקשת e, לכן נחזיר True
    - ערך חדש = ערך ישן: משמע שיש מסלול מינימלי אלטרנטיבי שאינו עובר ב-e, לכן נחזיר False

זמן ריצה

- סריקות BFS עולות O(V+E) כל אחת.
- אתחול שדות הצמתים O(V)
- הסרת צלע O(1)

Total: O(V+E) + O(V) + O(1) + O(V+E) = O(V+E)

פסאודו

```
IsEinAllMin(G, e, s, t)
    run BFS on G, start from s
    int min = t.d
    remove e from G
    for i = 1 to G.n
        initialize vertex i fields
    run BFS on G, start from s
        if t.d > min
            return true
        else
            return false
```

סעיף ב'

פירוט האלגוריתם

- נריץ סריקת BFS מקודקוד S:
  - נאחסן במשתנה min את המרחק המינימלי השמור ב-t
  - נאחסן במשתנה sToU1 את המרחק השמור בקודקוד u1
- נאפס את שדות הקודקודים
- נריץ שוב סריקת BFS, אך הפעם מקודקוד u2:
  - נאחסן במשתנה u2toT את המרחק השמור ב-t
- אם sToU1 + 1 + U2toT = min נחזיר True, אחרת False. הפלוס אחד מגיע מתוספת הקשת e לסכום.

זמן ריצה

- סריקות BFS עולות O(V+E) כל אחת.
- אתחול שדות הצמתים O(V)

Total: O(V+E) + O(V) + O(V+E) = O(V+E)

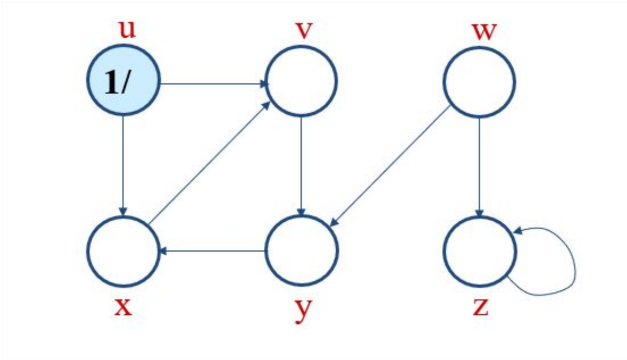
פסאודו

```
IsEinMin(G, e, s, t)
        run BFS on G, start from s
        int min = t.d
        int sToU1 = u1.d
        for i = 1 to G.n
                initialize vertex i fields
        run BFS on G, start from u2
        u2toT = t.d
        sToU1 + 1 + U2toT = min
                return true
        else
                return false
```

שאלה 4

סעיף א'

לא נכון, נפריך באמצעות דוגמה נגדית. נתבונן בגרף הבא:



- אם נתחיל את סריקת ה-DFS מקודקוד u, נקבל את העצים הבאים ביער:
T1 = {u, v, y, x}, T2 = {w, z}
- אם נתחיל את הסריקה מקודקוד w, נקבל את העצים הבאים ביער:
T1 = {w, y, x, v, z}, T2 = {u}

סעיף ב'

נשתמש באלגוריתם דייקסטרה שיתחיל מקודקוד s, ובעת סריקת הקודקודים נוסיף תנאי שבודק שאם הגענו לקוד t והמרחק קטן שווה m, נחזיר true.

פסאודו

```
isLessThanM(G, s, t, m)
    1.  initialize vertices distances = infinity, except d.s = 0
    2.  H = new minimum heap
    3.  Insert all vertices to H
    4.  While H.size != 0
        4.1.  u = extract min from H
        4.2.  for each v that is neighbour of u
            4.2.1.  if v = t AND u.d + w(u,v) <= m
                4.2.1.1.  return True
            4.2.2.  if v.d > u.d + w(u,v)
                4.2.2.1.  v.d = u.d + w(u,v)
                4.2.2.2.  H.heapify
    5.  return False
```

ניתוח זמן ריצה

- אתחול קודקודים O(v)
- הכנסת v קודקודים לערימה O(v)
- בכל לולאת while מוציאים קודקוד אחד, לכן סה"כ נכנסים ל-v לולאות. הוצאת קודקוד מהערימה מצריכה heapify, לכן סהכ נקבל בחלק זה O(VlogV)
- בשורה 4.2 - מעבר על השכנים של כל הקודקודים מתבצע כמספר הקשתות. בכל מעבר כזה ייתכן ונצטרך לעדכן מרחק של קודקוד ולבצע Heapify (קטע 4.2.2), לכן בסהכ נקבל כאן O(ElogV)

Total: O(v) + O(v) + O(VlogV) + O(ElogV) = O((V+E)logv)

שאלה 5

סעיף א'

- נשתמש בשתי ערימות, ערימת מקסימום וערימת מינימום בעלות מצביעים הדדיים
- Init: נכניס ל-2 הערימות את n הערכים בקבוצה: O(n) + O(n) = O(n)
- Insert: נכניס לכל אחת מהערימות את האיבר, ונוסיף מצביע הדדי בין 2 הצמתים
  O(logn) + O(logn) + O(1) = O(logn)
- find_min(): נחזיר את השורש בערימת המינימום - O(1)
- find_max(): נחזיר את השורש בערימת המקסימום - O(1)
- extract_min: נוציא את השורש מערימת המינימום. נגיע לצומת המקביל בערימת המקסימום דרך המצביע ונוציא גם אותו.

  O(logn) + O(logn) = O(logn)
- extract_max: נוציא את השורש מערימת המקסימום. נגיע לצומת המקביל בערימת המינימום דרך המצביע ונוציא גם אותו.

  O(logn) + O(logn) = O(logn)

סעיף ב'

<u>פירוט האלגוריתם</u>

- נתחזק ערימת מינימום H
- לכל צומת x בערימה יהיו 2 שדות נוספים:
  - maxSub: ישמור את ערך הצומת המקסימלי בכל תת העץ המושרש ב-x
  - maxNode: ישמור מצביע אל עבר אותו צומת מקסימלי
- Init: נאתחל את ערימת המינימום כרגיל עם n, ואז נעבור על כל הצמתים בעץ ונעדכן את 2 השדות הנ"ל בכל אחד מהצמתים. **איך נעשה זאת?**
  - ❏ נשתמש בסריקת Post Order, כך שתת העץ השמאלי והימני ייסרקו לפני שנעלה למעלה אל השורש שלהם.
  - ❏ נרד כרגיל עד לתחתית העץ, ונעצור כאשר נגיע לצומת נטול בנים. נעדכן את השדה maxSub שלו להיות הוא עצמו, ואת maxNode להצביע אליו. הסריקה תמשיך כעת לאח הימני של הצומת הנ"ל, שמהגדרת ערימה גם הוא נטול בנים, ובאופן דומה תעדכן את maxSub ואת maxNode שלו.
  - ❏ כעת נגיע לאבא של שני אחים אלו, ונעדכן את maxSub להיות הגדול מבין right.maxSub ו-left.maxSub, ואת maxNode למצביע ששמור אצל הבן הגדול.
  - ❏ רעיון זה ימשיך באופן רקורסיבי בסריקת ה-PostOrder, כאשר למעשה הערכים המקסימלים שמגיעים מן העלים "ילחמו ביניהם" כל פעם כשנעדכן את maxSub ברמה הגבוהה יותר - עד אשר השורש שלנו יכיל את המנצח הגדול - ערך הצומת המקסימלי בעץ, ופוינטר אליו.
- find_min: החזרת הערך שבשורש הערימה
- find_max: החזרת השדה maxSub של השורש

- extract_min: בהתאם לאלגוריתם של ערימה, נוציא את השורש ונכניס במקומו את העלה האחרון בערימה x. נתקן את maxSub ו-maxNode במסלולים הבאים:
  - מהאבא המקורי של העלה x ועד לשורש
  - מהמיקום החדש של x לאחר heapify ועד לשורש
- extract_max: ניגש לצומת המקסימלי דרך השדה maxNode של השורש, נוציא אותו ונחליף אותו בעלה הימני ביותר בערימה x. נתקן את maxSub ו-maxNode במסלולים הבאים:
  - מהאבא המקורי של העלה x ועד לשורש
  - ממיקומו החדש של x לאחר heapify ועד לשורש

<u>ניתוח זמן ריצה</u>

- Init: אתחול ערימה כרגיל O(n). סריקת PostOrder וביצוע עדכוני השדות בזמן קבוע עולה גם O(n), סה"כ O(n) כנדרש
- find_min, find_max: זמן קבוע O(1)
- extract_max, extract_min: הוצאה בהתאם לערימה עולה O(logn). תיקון השדות תלוי בגובה הערימה לכן גם עולה O(logn), סה"כ O(logn)

<u>פסאודו</u>

```
Init()
	Insert n values to new heap H
	Perform Post Order on H
		if X.right = null AND X.left = null
			X.maxSub = X.Value
			X.maxNode = X
		Y.maxSub = max(Y.right.maxSub, Y.left.maxSub) \\ Y is parent
		Y.maxNode =  max(Y.right.maxSub, Y.left.maxSub).maxNode
find_min
	return root.value
find_max
	return root.maxNode
extract_min
	H.extractMin()
	Let X be the last leaf before the extraction
	Use similar logic of Init() to update the fields from X original parent to the root
	Use similar logic of Init() to update the fields from X new place to the root
extract_max
	Let X be the last leaf
	Delete maxNode and move X to its place
	Use similar logic of Init() to update the fields from X original parent to the root
	Use similar logic of Init() to update the fields from X new place to the root
```

<u>נ.ב</u> מעולם לא עשיתי סעיף ב' כל כך אכזרי בהשוואה לסעיף א'

שאלה 6

סעיף א'

תיאור האלגוריתם

- לכל צומת נוסיף שדה flag בוליאני מאותחל ל-False
- נמיין את הגרף טופולוגית
- נסרוק את הצמתים לפי סדר המיון הטופולוגי. בכל קודקוד בו נעבור כאשר flag=false:
  - נוסיף אותו לקבוצה C
  - נעדכן flag=true
  - נעדכן flag=true לכל הקודקודים המחוברים אליו בקשת (כשהקשת יוצאת ממנו).

פסאודו

```
getC(G)
    1.  Sort G topologically
    2.  Scan the topical linked list in order, let v be the current node:
        2.1.  if v.flag = false
              2.1.1.  C = C U V
              2.1.2.  v.flag = True
              2.1.3.  for all e={v, u} in G:
                      2.1.3.1.  u.flag = True
```

הוכחת נכונות

א. לכל v ב-C לא קיים u ב-C כך שקיימת e = {u,v}

נניח בשלילה שקיים u ב-C כך שקיימת e = {u,v}

מהגדרת מיון טופולגי, u ממוקם לפני v ברשימה המקושרת.

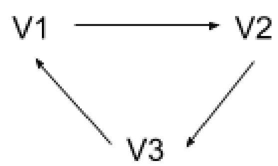
לכן כאשר נסרוק את u, נעדכן את השדה flag של שכנו v להיות True, ומכאן בהכרח לא נכניס את v ל-C בסתירה לנתון.

ב. לכל v ב- V\C קיים u ב-C כך שקיימת e = {u,v}:

בתום ריצת האלגוריתם על כל קודקוד לקבל ערך True. שינוי הערך מתבצע בשני מצבים:

1. הקודקוד התגלה ממעבר סדרתי על הרשימה המקושרת:
   אך אז ע"פ האלגוריתם הקודקוד שייך ל-C, לכן לא יכול להיות שמדובר ב-v.
   הקודקוד התגלה מסריקת קודקוד אחר המחובר אליו: אך אז ע"פ האלגוריתם הקודקוד המחובר אליו בהכרח שייך ל-C. לכן הטענה נכונה.

סעיף ב'



הטענה אינה נכונה, דוגמה נגדית:

בין כל שני קודקודים קיימת קשת,

לכן מהתנאי השני הקבוצה C ריקה.

<u>שאלה 7</u>

נממש את המבנה באמצעות גרף:
נעבור על כל הזוגות ונבדוק עבור כל זוג אם הוא נמצא במבנה. אם לא נכניסו לגרף.
כשנתבקש לבדוק האם שניים קרובי משפחה נריץ BFS מאחד מהם ונבדוק אם הוא מגיע לשני. אם כן מדובר בקרובי משפחה.
זמן ריצה כמו BFS:

$O(V+E)=O(n)$

<u>שאלה 8</u>

<u>סעיף א'</u>

i. $k = O(2^n)$.

- Counting Sort לא בא בחשבון
- יש לנו n במעריך, לכן לא ניתן לייעל באמצעות המרת בסיסים ו-Radix Sort.

לכן נשתמש ב-Merge Sort - המיון היעיל ביותר ללא הנחות על הקלט, ונקבל O(nlogn)

ii. $k = O(2^{\log^3 n})$.

גם כאן יש לנו n במעריך (או פונקציה שתלויה ב-n), לכן מאותן סיבות לעיל נמיין באמצעות Merge Sort ונקב O(nlogn)

iii. $k = O(n^5)$.

- נמיר את כל המספרים מבסיס עשרוני לבסיס n. ההמרה של כל מספר מתבצעת בזמן קבוע, לכן סך ההמרות יעלו O(n).
- נמיין אותם באמצעות Radix Sort שיעשה שימוש ב-Counting Sort.

  כדי לחשב חישוב זמן זמן הריצה של Radix נמצא את קודם את d.

  נחשב על פי הנוסחה את מספר הספרות בבסיס n של המספר המקסימלי בטווח - n^5 :

  $$d = log_n n^5 + 1 = 5 + 1 = 6$$

  לכן זמן הריצה של המיון:

  O(d(n+b)) = O(6(n + n) = O(n)

ובסה"כ נקבל O(n) + O(n) = O(n)

<u>סעיף ב'</u>

1. <u>אלגוריתם:</u> נסרוק איבר-איבר ב-A, ובמסגרת סריקה של כל איבר נסרוק את כל האיברים במערך ונבדוק אם סכומם שווה ל-k.

   <u>פסאודו:</u>

```
for i from 0 to n:
        for j from 0 to n:
                if A[i] + A[j] = k
                        return true
```

   <u>זמן ריצה:</u> 2 לולאות מקוננות בעלות n איטרציות, נקבל O(n*n) = O(n^2)

   <u>זיכרון:</u> כל הפעולות התבצעו על גבי A, לכן לא השתמשנו בזיכרון נוסף

2. <u>אלגוריתם:</u>
- נעתיק את A למערך זהה B
- נבצע מיון Heap Sort על B
- נסרוק את המערך באמצעות שני פוינטרים מתחילתו ומסופו - אם הסכום קטן או גדול מ-k נקדם את הפוינטר הרלוונטי, עד שנגיע לסכום המבוקש

<u>פסאודו:</u>

```
Copy A to new array B
Perform Heap Sort on B
 i = 0 , j = n-1
while i < j
        if B[i] + B[j] = k
                return true
        if B[i] + B[j] < k
                i++
        else
                j--
return false
```

<u>זמן ריצה:</u>
- העתקה למערך עזר B עולה O(n)
- Heap Sort מתבצע ב-O(nlogn)
- לכל היותר נעבור על n איברים בשלב הסכימות, לכן O(n)

Total: O(n) + O(nlogn) + O(n) = O(nlogn)

<u>זיכרון:</u> מערך B משתמש ב-O(n) זיכרון נוסף, Heap Sort לא משתמש בזיכרון נוסף, ושלב ההשוואות מתבצע על גבי B לכן גם לא צורך זיכרון נוסף. סה"כ O(n) כנדרש.

3. <u>אלגוריתם:</u>
- נבנה מערך עזר B בגודל K.
- נעבור על איברי המערך A ונעלה את ערך האינדקס המתאים במערך B בדומה לחלקו הראשון של האלגוריתם של counting sort. במהלך סריקת A נתעלם מכל מספר שגדול מ-k, זאת כדי לא לחרוג מ-B ומשום שמספרים אלו לא מעניינים אותנו שכן ממילא לא יוכלו להיות חלק מהסכום אותו אנו מחפשים.
- נסרוק את המערך B באמצעות שני פוינטרים - מההתחלה ומהסוף.
- נבדוק האם כל תא ב-B גדול מאפס:
  - אם לא, נקדם/נחסר מהפוינטר הרלוונטי כדי לעבור למס' הבא.
  - אם כן - אזי הפוינטר הנ"ל הינו איבר ב-A.

  נסכום את ערכי <u>הפוינטרים</u> ונשווה ל-k - אם כן נחזיר True.
  אם הסכום גדול/קטן מ-k, נקדם את הפוינטרים בהתאם.

<u>פסאודו:</u>

```
Create new array B in length k+1
for i = 0 to n:
	if A[i] <= k
		B[A[i]] ++
i = 0, j = k-1
while i < j
	if B[i] = 0
		i++
	else if B[j] = 0
		j--
	else
		if i + j = k
			Return True
		if i + k < k
			i++
		else
			j--
return False
```

<u>זמן ריצה:</u>

- מעבר על איברי A לעדכון איברי B עולה O(n)
- במעבר עם הפוינטרים על B לכל היותר נעבור על כל k התאים במערך, לכן O(k)

  Total: $O(n) + O(k) = O(n)$

<u>זיכרון:</u>

- יצירת מערך B עולה O(k) זיכרון נוסף, שאר הפעולות מתבצעות על גבי שני המערכים לכן לא צורכות זיכרון נוסף. לכן סהכ O(k) כנדרש.

שאלה 9

פירוט האלגוריתם וניתוח זמן ריצה

- למימוש האלגוריתם נתחזק 2 ערימות:
  - ערימת מינימום שתכיל את כל האיברים שגדולים מהחציון
  - ערימת מקסימום שתכיל את כל האיברים שקטנים מהחציון
- נתחזק 2 שדות נוספים:
  - median: ישמור את החציון בכל רגע נתון
  - balance: ישמור את ההפרש במספר איברי הערימות בכל רגע נתון. מס' חיובי יעיד שערימת המינימום גדולה יותר, ושלילי ההפך. משתנה זה יסייע לנו להבין מתי יש לעדכן חציון, ומאיזו ערימה יש להגדיר את החציון החדש.
- Init: נסרוק את כל n האיברים, אם האיבר גדול מהחציון נכניס אותו לערימת המינימום, ואם האיבר קטן מהחציון נכניס אותו לערימת המקסימום.
  כשהאיבר גדול מהחציון: balance++, כשהוא קטן מהחציון: balance--
- Insert: נבדוק אם האיבר גדול או קטן מהחציון, נכניס אותו לערימת המינימום או המקסימום בהתאמה, ונעדכן balance++ או balance-- בהתאמה.
  אם balance = 2 או balance = -2, משמע שיש להזיח את החציון הנוכחי השמור ב-median מקום אחד. נכניס את החציון הנוכחי לערימת המקסימום או המינימום בהתאמה, נשלוף את השורש בערימה השניה ונשמור אותו כ-median החדש.
  נאפס את balance בסיום.
- median: החזרת הערך השמור ב-median
- delete_median:
  - אם balance = 0 OR 1, נגדיר את החציון החדש להיות השורש של ערימת המינימום המחזיקה בערכים גדולים מהחציון. לצורך כך נשלוף את איבר זה ונעדכן את median ואת balance בהתאם.
  - אם Balance -1, כנ"ל רק לגבי ערימת המקסימום.

ניתוח זמן ריצה

- Init: סריקת n איברים: O(n), בניית ערימה מ-O(n) איברים: O(n).
  Total: O(n) + O(n) + O(n) = O(n)
- Insert: הכנסה לערימה O(logn).
  Extract Min \ Extract Max עולות גם הן O(logn)
  במקרה הגרוע כאשר balance=2 נקבל:
  O(logn) + O(logn) + O(logn)
- median: פעולה קבועה O(1)
- delete_median:
  Extract Min \ Extract Max עולות O(logn), לכן בסה"כ נקבל O(logn)

פסאודו

```
Init()
        Initialize median as the given median, balance = 0
        LargerH = New min Heap
        SmallerH = New max Heap
        Scan elements i to n
                if element i >= median
                        Insert i to LargerH
                        balance++
                else
                        Insert i to SmallerH
                        balance--

Insert(x)
        if x >= median
                Insert x to LargerH
                balance++
        else
                Insert x to SmallerH
                balance --
        if balance = 2
                insert median to SmallerH
                median = Extract Min from LargerH
                balance = 0
        if balance = -2
                insert median to LargerH
                median = Extract Max from SmallerH
                balance = 0

delete_median()
        if balance = 0 OR balance = 1
                median = Extract Min from LargerH
                balance--
        else
                median = Extract Max from SmallerH
                balance++
```

שאלה 10

פירוט האלגוריתם

- נבנה העתק של הגרף המקורי - 'G, עם קודקודים וקשתות זהים.
- נחבר כל קודקוד ב-G השייך לשכבה ה-k לקודקוד החופף שלו ב-'G בקשת.
- קיבלנו גרף 'GUG, נריץ BFS מקודקוד s ב-G לקודקוד 't ב-'G, ונחזיר את המסלול הקצר ביותר - פחות הצלע המיותרת {'k, k} (הקשת המחברת בין G ל-'G).

ניתוח זמן ריצה

- בניית גרף נוסף 'G עולה O(V+E) משום שנצטרך ליצור V קודקודים ו-E קשתות.
- חיבור הקודקודים בשכבה ה-k עולה O(V)
- הרצת BFS על 'GUG עולה O(2V + 2E) = O(V+E)

Total: O(V+E) + O(V) + O(V+E) = O(V+E)

הוכחת נכונות

(לא פורמלית כי אין לי מושג איך).
נבחין כי במסלול הקצר ביותר שמחזיר BFS לעולם לא תופיע אותה הקשת פעמיים, משום שאז קשת זו מיותרת ומאריכה שלא כצורך את המסלול. בשאלה זו למעשה נצטרך "להנדס" את אלגוריתם ה-BFS כדי לכפות עליו לעבור בקודקוד משכבה K, ואז לחזור חזרה על קשתות.
בטריק בו השתמשנו בשאלה זו, ברגע שאלגוריתם ה-bfs מגיע לקודקוד ה-k הרלוונטי, הוא עובר לגרף 'G - וכך למעשה הוא יכול לחזור חזרה על קשתות שכבר עבר עליהן - רק שכעת הן הקשתות החופפות ב-'G.
כמו כן, סריקת ה-bfs - שמהגדרתה היא סריקת רוחב - מוצאת גם באיזה קודקוד k עלינו לעבור על מנת לדלג כמה שיותר מהר בין הגרף 'G ל-G ולהגיע ל-'t, ומכאן מינימליות המסלול.

פסאודו

```
getMinK(G)
        Scan G and create duplicate G’
                for each v in G in layer k:
                        Add new edge {v,v’}
        Perform BFS on G             // now GUG’
        output = new String
        u = t’
        while u has predecessor:
                S = S + u            // Omit case in which u = u’
                u = u.predeccesor
        return output
```