המכללה האקדמית להנדסה
אורט בראודה
ORT BRAUDE COLLEGE

המחלקה להנדסת חשמל ואלקטרוניקה

מבני נתונים ואלגוריתמים 31632

פתרון מבחן סופי, מועד א', סמסטר א' תשע"ד, 03/02/2014

<u>הוראות לנבחן:</u> משך הבחינה שלוש שעות. חומרי העזר המותרים הם השקפים של הקורס שנמסרו באתר הקורס בלבד. שימוש במחשבונים ומכשירים אלקטרוניים אסור בהחלט. רשום את תשובותיך במחברת המצורפת לבחינה. ציין בבירור את מספר השאלה במחברת והשתדל להיות <u>קצר וענייני</u> (תשובות ארוכות מדי או נסיונות לתת מספר תשובות אפשריות <u>ייפסלו</u> את תשובתך!). הקפד על כתב יד ברור ומסודר, ומחק את כל חומר הטיוטא. השימוש בשפה האנגלית מותר. השאלון מכיל 6 שאלות, ונפרש על פני 3 עמודים.
בהצלחה !

<u>שאלה 1</u> [20%]

להלן מיפרט ADT של מבנה נתונים בשם תור (Queue). זהו מבנה נתונים שבו עצמים נכנסים לסדרה בצד אחד ויוצאים ממנה מהצד השני. דוגמאות פשוטות הן תור של לקוחות לקופאי בבנק, או תור של עבודות הדפסה שנשלחות למדפסת משותפת.

q = Queue()
Create a new empty queue object q
q.enqueue(e)
Add element e to the back of the queue q.
q.dequeue()
Remove and return the first element from queue q.
An error occurs if the queue is empty.
q.first()
Return a reference to the element at the front of the queue q
without removing it; an error occurs if the queue is empty.
q.is_empty()
Return True if q does not contain any elements.
len(q)
Return the number of elements in queue q

א. כתוב מחלקה Queue בשפת Python אשר ממשת את המיפרט (ADT) הנ"ל באמצעות רשימה (list).
ב. מהי הסיבוכיות זמן של פעולות enqueue ו-dequeue שקיבלת במימוש שלך?
ג. האם ניתן לממש את מבנה הנתונים Queue באופן שונה כך שסיבוכיות הפעולות enqueue ו-dequque לא תעלה על O(1)? (אין צורך לממש אך יש להסביר את הרעיון בצורה ברורה)

<u>פתרון:</u>

א+ב. הסיבוכיות של enqueue היא O(1) (הוספה של איבר לסוף מערך דורש פעולה אחת). הסיבוכיות של dequeue לעומת זאת היא O(n) מאחר שהסרת האיבר ברשימה מחייבת אותנו להעביר את כל יתר האיברים כלפי מטה (במקום האיבר הראשון שירד) – מדובר בערך ב-n פעולות.
ג. ניתן להתגבר על בעייה זו במספר דרכים שונות ומגוונות. נציג רעיון פשוט אחד כדוגמא: בתוך המחלקה נחזיק שדה נוסף head שיצביע לראש התור. בשלה האיתחול head=-1 ולאחר הכנסת האיבר הראשון, head=0. בכל פעם שמתבצעת פעולת dequque מחזירים את L[head] ומקדמים את head ב-1. החיסרון: ייתכן ביזבוז גדול של שטח אם מספר ההכנסות הוא גדול מאוד ומספר ההוצאות קרוב למספר ההכנסות (למשל, אם הכנסנו מיליון ושלוש איברים והוצאנו מיליון איברים, אז התור יכיל שלושה איברים, אך כל שאר המיליון שהוצאנו עדיין ברשימה) ניתן להתגבר על בעייה זו על ידי איפוס הרשימה בכל שלב שבו head עובר את מחציתה למשל.

```
class Queue:
    def __init__(self):
        self.elements = []

    def enqueue(self, elem):
        self.elements.append(elem)

    def dequeue(self):
        return self.elements.pop(0)

    def first(self):
        return self.elements[-1]

    def is_empty(self, other):
        return self.elements == []

    def __len__(self):
        return len(self.elements)

    def __str__(self):
        return str(self.elements)
```

שאלה 2 [6%]

נתונה הרשימה הבאה למיון:

L = [15, 5, 4, 18, 12, 19, 14, 10, 8, 20]

רשום מה יהיה מצב הרשימה לאחר שלושה סבבים מלאים של האלגוריתם Insertion sort?

פתרון:

להלן כל הסבבים:

```
1:  [5, 15, 4, 18, 12, 19, 14, 10, 8, 20]
2:  [4, 5, 15, 18, 12, 19, 14, 10, 8, 20]
3:  [4, 5, 15, 18, 12, 19, 14, 10, 8, 20]
4:  [4, 5, 12, 15, 18, 19, 14, 10, 8, 20]
5:  [4, 5, 12, 15, 18, 19, 14, 10, 8, 20]
6:  [4, 5, 12, 14, 15, 18, 19, 10, 8, 20]
7:  [4, 5, 10, 12, 14, 15, 18, 19, 8, 20]
8:  [4, 5, 8, 10, 12, 14, 15, 18, 19, 20]
9:  [4, 5, 8, 10, 12, 14, 15, 18, 19, 20]
```

שאלה 3 [32%]

נתונה רשימה (Python list) של מספרים שלמים (שליליים וחיוביים). המטרה לחפש תת-רשימה רציפה שבה סכום המספרים הוא מינימלי.

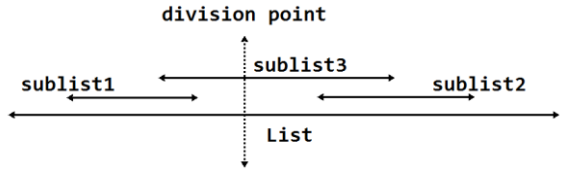
למשל אם הרשימה שלנו היא: **[12, -9, 3, 4, -17, 12, -2, 3, -5, -1]**

תת רשימה רציפה שסכומה מינימלי היא: **[-9, 3, 4, -17]** (והסכום יהיה 19-).

א. רשום פונקציית Python "נאיבית" brute_force(L) אשר בודקת את כל תתי-הרשימות הרציפות של הרשימה L ומחזירה שלשה (i, j, sum) כאשר i, j הם אינדקסים של תת-רשימה בעלת סכום מינימלי sum הסבר במדויק את הרעיון של הפונקציה שרשמת

ב. מהי סיבוכיות הזמן (worst case) של האלגוריתם שמצאת בסעיף הקודם?

ג. את אותה הבעיה ניתן לפתור בשיטת הפרד ומשול על פי הרעיון הבא:



נבחר איבר ברשימה (division point) ונחלק את הרשימה לשניים, ואז תת-רשימה מינימלית יכולה להמצא:

- כולה משמאל לנקודת החלוקה (sublist1 לא כולל נקודת החלוקה)
- כולה מימין (sublist2 לא כולל נקודת החלוקה)
- תת-הרשימה sublist3 הכוללת את נקודת החלוקה:
  חלק ראשון משמאל לנקודת החלוקה, וחלק שני מימין לנקודת החלוקה

רשום פונקציית Python בשם find_min_sublist(List) לביצוע אלגוריתם זה.

הדרכה: נסה לבנות פונקציית עזר find_min_crossing_sublist(List, i) בכדי למצוא תת-רשימה מינימלית הכוללת את האיבר ה-i.

ד. הסבר במדויק את הרעיון מאחורי שתי הפונקציות שכתבת בסעיף הקודם

ה. מהי סיבוכיות הזמן (worst case complexity) של האלגוריתם החדש? תן הסבר מילולי ברור.

הערות:

ייתכן יותר מפיתרון אחד (מספר תתי רשימות מינימליות). הפונקציה צריכה להביא את אחד מהן בלבד. בכדי לפשט את האלגוריתם מותר להניח שכאשר הרשימה ריקה, אז הפונקציה תחזיר (inf,None,None) כאשר inf=float("Infinity").

<u>פתרון:</u>

<u>א.</u>

```
def brute_force(L):
    min_sum, min_i, min_j = float("Infinity"), None, None
    n = len(L)
    for i in range(n):
        for j in range(i,n):
            s = sum(L[i:j+1])
            if s < min_sum:
                min_sum, min_i, min_j = s, i, j
    return min_sum, min_i, min_j
```

סיבוכיות הזמן של פיתרון זה היא בבירור n^3 : שתי לולאות באורך n, וסכום של בממוצע n/2 איברים.

<u>ב.</u> על פי שיטת הפרד ומשול, נחלק את הרשימה לשני חצאים באמצעות נקודת האמצע mid=n/2 (כאשר n הוא אורך הרשימה L) : L[0:mid], L[mid+1:n] ונפעיל באופן רקורסיבי את הפונקציה find_min_sublist על כל אחד מהחצאים, ומבין שתי התוצאות נקח את הטובה ביותר (סכום מינימלי יותר). הבעייה עם הרעיון הוא שאנו מפספסים את כל תתי-הרשימות שכוללות את האיבר mid בתוכן! לשם כך דרושה לנו פונקצייה נוספת למטרה זו בלבד :

```
def find_min_sublist(L):
    n = len(L)
    if n == 0:
        return float("Infinity"), None, None
    mid = n/2
    min_sum, i, j = find_min_sublist(L[0:mid])
    right_sum, right_i, right_j = find_min_sublist(L[mid+1:n])
    if right_sum < min_sum:
        min_sum = right_sum
        # need to shift indices to parent list
        i, j = mid + 1 + right_i, mid + 1 + right_j
    cross_sum, cross_i, cross_j = find_min_crossing_sublist(L, mid)
    if cross_sum < min_sum:
        min_sum, i, j = cross_sum, cross_i, cross_j

    return min_sum, i, j
```

כל תת-רשימה L[a:b] שכוללת את האיבר mid ניתן לחלק לשלושה חלקים :

L[a,b] = **L[a:mid]** + L[mid] + **L[mid+1:b]**

נסמן :

left_sum = L[a:mid], right_sum=L[mid+1:b]

בכדי שהסכום יהיה מינימלי, left_sum ו-right_sum חייבים להיות שליליים ככל האפשר.

הפונקציה מתחילה בערך 0 ומנסה למזער את החצי השמאלי והימני ככל שניתן (במקרה שכל האיברים חיוביים למשל, לא נוכל להתקדם לשום כוון, ונסתפק באיבר L[mid] בלבד).

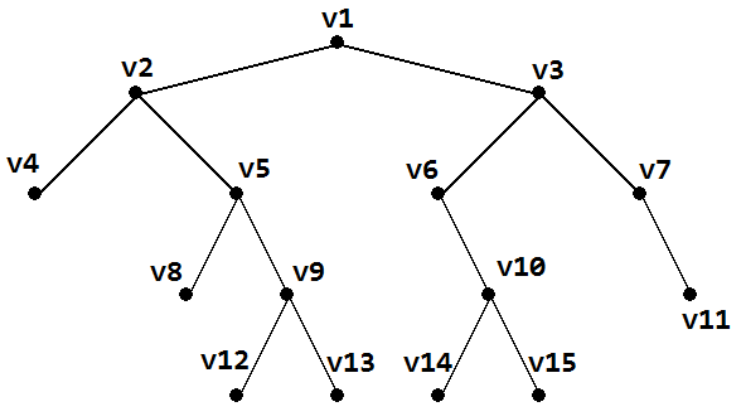
```
def find_min_crossing_sublist(L, mid):
    n = len(L)
    left_sum, right_sum = 0, 0
    cross_i, cross_j = mid, mid
    sum = 0
    for i in range(mid-1, -1, -1):
        sum += L[i]
        if sum<left_sum:
            left_sum = sum
            cross_i = i
    sum = 0
    for j in range(mid+1, n):
        sum += L[j]
        if sum<right_sum:
            right_sum = sum
            cross_j = j
    cross_sum = L[mid] + left_sum + right_sum
    return cross_sum, cross_i, cross_j
```

הסיבוכיות של שיטת החצייה היא log n (לאחר log n חלוקות נסיים). החישוב היחיד שמתבצע הוא **`find_min_crossing_sublist`** אשר בבירור בסיבוכיות O(n) (לכל היותר סכום של n איברים), והוא מתבצע log n פעמים, ולכן הסיבוכיות הכללית היא O(n log n).

שאלה 4 [12%]

לפניך דיאגרמה של עץ בינארי. לגבי כל אחד מהסעיפים הבאים, רשום את רשימת הקודקודים כפי שתתקבל באמצעות אלגוריתם הסריקה המתאים. רשום תשובה סופית בלבד (שיטת החישוב לא תיבדק!)

א. Preorder Traversal
ב. Postorder Traversal
ג. Inorder Traversal



פתרון:

א. v1, v2 , v4, v5, v8, v9, v12, v13, v3, v6, v10, v14, v15, v7, v11
ב. v4, v8, v12, v13, v9, v5, v2, v14, v15, v10, v6, v11, v7, v3, v1
ג. v4, v2, v8, v5, v12, v9, v13, v1, v6, v14, v10, v15, v3, v7, v11

שאלה 5 [15%]

להלן מיפרט ADT (חלקי) של מבנה נתונים מסוג עץ שנקרא SimpleTree בשפת Python.

```
t = SimpleTree()
    Create a new empty SimpleTree object
t.get_root()
    Return the root Position of the tree (or None if tree is empty)
t.get_parent(p)
    Return the Position of p's parent (or None if x is root)
t.get_children(p)
    Return the Position of p's children
t.num_children(p)
    Return the number of children of Position p
t.add_root(e)
    Place element e at the root of an empty tree
    Raise ValueError if tree nonempty
t.add_child(p, e)
    add a new element e at the end of children of p
    Return the Position of the new node
```

בהנחה שהמחלקה SimpleTree מומשה במלואה (בצרוף מתודות נוספות שאינן רלבנטיות כאן), וסיבוכיות הזמן של כל המתודות היא O(1), ענה על השאלות הבאות:

א. בנה אלגוריתם יעיל ככל האפשר heigth(t) בשפת Python המחשב את גובהו של עץ SimpleTree נתון.
מהי הסיבוכיות זמן של האלגוריתם שמצאת?

ב. בנה אלגוריתם יעיל ככל האפשר ancestors(t, p) בשפת Python המקבל מפתח p בעץ t (מסוג SimpleTree) שמחזיר את כל האבות הקדמונים של המפתח p.
מהי סיבוכיות הזמן של האלגוריתם שבנית? בנה דוגמא של עץ עבור המקרה הכי גרוע (worst-case).

פתרון:

```
def height(t):
    "Height of SimpleTree t"
    root = t.get_root()
    if root is None:
        return 0
    return _height(t, root)

def _height(t, p):
    "Height of position p in SimpleTree t"
    if t.num_children(p) == 0:
        return 0
    children_heights = [_height(t,c) for c in t.get_children(p)]
    return 1 + max(children_heights)

def ancestors(t,p):
    parent = t.get_parent(p)
    if parent is None:
        return []
    else:
        return ancestors(t, parent) + [parent]
```

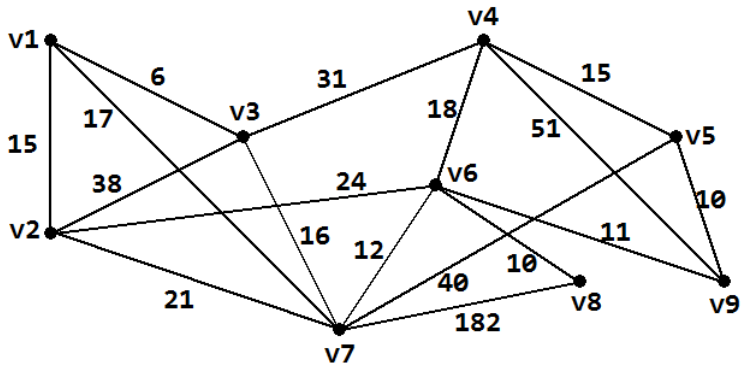
שאלה 6 [15%]

עיין בגרף g המשורטט לפניך. זהו גרף ממושקל ולא מכוון שמציג מפת עלויות (במיליארדי דולרים) לבניית גשרים בין 9 איים בים הכספי (תוכל להניח שהגרף מכוון כשכל הצלעות בעלות שני כוונים).

א. הפעל את אלגוריתם ה-DFS כפי שהוצג במהלך הקורס, ושרטט סכימה ברורה (בשטח נפרד במחברת) של העץ הפורש המתקבל.
<u>הנחיות:</u> התחל בקודקוד v1, ובכל שלב בו צריך להתקדם לקודקוד הבא, בחר תמיד את הקודקוד הכי קטן על פי הסדר המספרי.

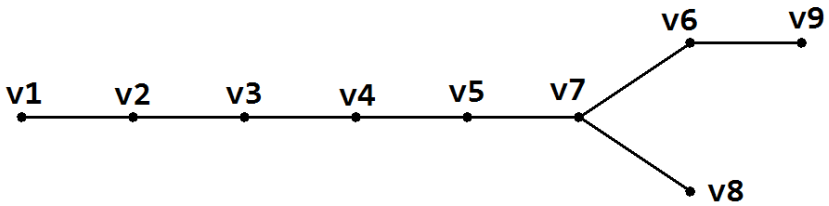
ב. הפעל את האלגוריתם של Dijkstra בכדי למצוא את העץ הפורש שמקורו באי הראשי v1 (שבו שוכנת עיר הבירה של הממלכה). שרטט סכימה ברורה של העץ בשטח נפרד במחברת (כולל מרחקים מעל הצלעות).

ג. חשב את המסלול הקצר ביותר בין האי v1 לאי v8 על ידי שימוש בעץ הפורש שמצאת בסעיף הקודם (שרטט מסלול מתאים על העץ הפורש).

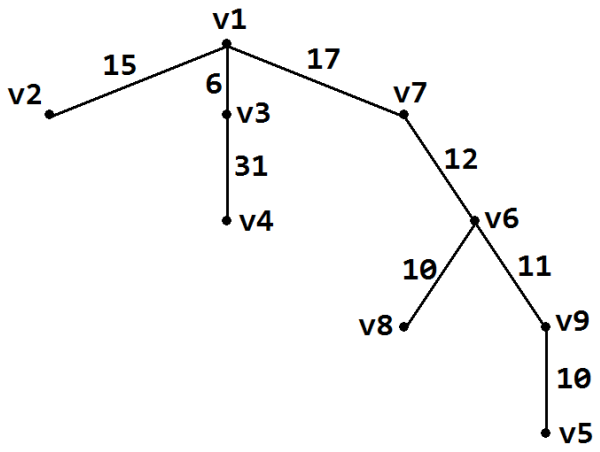


פתרון:

א. עץ DFS:



ב. העץ הפורש של דיקסטרה יראה כך לאחר הפעלת האלגוריתם של דיקסטרה:



ג. מהעץ ברור כי המסלול הכי קצר מ-v8 ל-v1 הוא: v8 -> v6 -> v7 -> v1
האורך שלו הוא: 39=10+12+17