**חלק א' – יסודות**

**בחרו שתי שאלות מבין שלשת השאלות הבאות.**

**שאלה 1 (50 נק)**

public static String raz(String s)

{

if (s.length() == 0)

return s;

char c = s.charAt(s.length()-1);

String st = raz(s.substring(0, s.length()-1));

if (c >='A' && c <= 'Z')

return c + st;

return st;

}

1. נתונה הפעולה sod.

טענת כניסה: הפעולה מקבלת מחרוזת

* 1. בצעו מעקב אחרי זימון הפעולה raz (q) עבור המחרוזת “aBCdE” **(15 נק)**

ערך חזרה ECB

|  |
| --- |
| raz(“aBCdE”( |
| if 5 == 0 🡪F  c=E  st = CB  if E>=A && E <=Z 🡪T  return ECB |
|  |
| raz(“aBCd”( |
| if 4 == 0 🡪F  c=d  st = CB  if d>=A && E <=d 🡪F  return CB |
|  |
| raz(“aBC) |
| if 3 == 0 🡪F  c=C  st = B  if C>=A && C <=Z 🡪T  return CB |
|  |
| raz(“aB”( |
| if 2 == 0 🡪F  c=B  st =“”  if B>=A && B <=d 🡪T  return B + “” |
|  |
| raz(“a”) |
| if 1 == 0 🡪F  c=a  st = “”  if a>=A && a<=Z 🡪f  return “” |
|  |
| raz(“”) |
| if 0 == 0 🡪T  return “” |

* 1. מה טענת היציאה של הפעולה raz **(3 נק)**

הפעולה raz מחירה את האותיות הגדולות במחרות s מסודרות בסדר הפוך

* 1. מה סיבוכיות הפעולה? הסבירו **(4 נק)**

**עבור מחרוזת באורך n** הפעולה raz מזומנת פעם אחת עבור כל תו במחרוזת. בזימון היר

1 – מזמנת את התפעולה בלי התו האחרון

2 – בודקת אם התו האחרון אות גדולה

3 – מחזירה ערך

סכ"ה 3 פעולות ב זימונים 🡺 3n פעולות. לכן הסיבוכיות **O(n)**

public static String paz(Queue<String> q)

{

if (q.isEmpty())

return "";

String s1 = raz(q.remove());

String s2 = paz(q);

return s2 + s1;

}

1. נתונה הפעולה paz. טענת כניסה: הפעולה מקבלת תור מחרוזות.
2. בצעו מעקב עבור זימון הפעולה” paz (q) עבור התור: **(20 נק)**

**\* אין צורך לעקוב אחרי raz**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| “abOR” | “HwT” | “affd” | “qGqRqEqB” | q=ראש התור🡪 |

|  |  |
| --- | --- |
| paz([“qGqRqEqB”, “affd”, “HwT”, “abOR”]) |  |
| if q.isEmpty🡪 F  s1= BERG  s2=ROTH  return ROTHBERG | q=[“affd”, “HwT”, “abOR”] |
|  |  |
| paz([ “affd”, “HwT”, “abOR”]) |  |
| if q.isEmpty🡪 F  s1= affd  s2=ROTH  return ROTH | q=[ “HwT”, “abOR”] |
|  |  |
| paz([ “HwT”, “abOR”]) |  |
| if q.isEmpty🡪 F  s1=TH  s2=RO  return ROTH | q=[ “abOR”] |
|  |  |
| paz([ “abOR”]) |  |
| if q.isEmpty🡪 F  s1=RO  s2=””  return RO | q=[ ] |
|  |  |
| Paz([]) |  |
| if q.isEmpty🡪 T  return “” |  |

ערך חזרה ROTHBERG

1. מה טענת היציאה של הפעולה paz? **(4 נק)**

הפעולה מקבלת תור מחרוזות ומחזירה מחרוזת המכילה את כל הרותיות הגדולות של המילים בתור כהמילים מסודרות מהסוף התור לתחילתו ובכל מילה גם סדר האותיות הפוך

1. מה סיבוכיות הפעולה paz? הסבירו **(4 נק)**

עבור תור באורך n שהחלה מקסימלית בו באורך m.

הפעולה מזומנת פעם אחת עבור כל ריבר בתור ובכל זימון

* שולפת את האיבר העליון בתור
* מפעילה עליו את הפעולה raz – O(m)
* מחזירה את תוצאת החיבור של הזימון + הזימון הרקורסיבי

כלומר יש לנו n זימונים שכל אחד מהם מזמן פעולה שהסיבוכיות שלה O(m). סה"כ הסיבוכיות הכוללת **O(n\*m)**

**שאלה 2 (50 נק)**

1. כתבו פעולה בשם **isIn** המקבלת תור שלמים ומספר שלם ומחזירה אמת אם המספר נמצא בתור ושקר אחרת. הפעולה לא תפגע בתור **(15 נק)**

|  |
| --- |
| public static boolean isIn(Queue<Integer> q, int num)  {  q.insert(null);  boolean in = false;  while(q.head() != null)  {  int n = q.remove();  if (n == num)  in = true;  q.insert(n);  }  q.remove();  return in;  } |

1. מה סיבוכיות הפעולה isIn שכתבתם? **(5 נק)**

עבור תור באורך n ומספר num. הפעולה סורקת את התור ועבור כל איבר בודקת אם הוא שווה ל num ומחזירה אותו לתור. כלומר לכל איבר בתור מתבצעות 2 פעולות ולכן הסיבוכיות **O(n)**

1. כתבו פעולה בשם **order** המקבלת 2 תורי מספרים שלמים q1 ו q2 ומחזירה תור שלישי שמסודר באופן הבא:
   1. בראש התור מספרים שמופיעים גם ב q1 וגם q2
   2. בהמשכו מספרים שמופיעים רק ב q1 ולא ב q2
   3. בסופו מספרים שמופיעים רק ב q2 ולא ב q1

**\* בתור התוצאה כל מספר מופיע פעם אחת ללא חזרות (25 נק)**

ניצור תור תוצאה ונמלא אותו ב 2 שלבים:

1. נעבור על התור הראשון ונכניס לתור התוצאה כל איבר שיש גם בתור השני. כך נקבל בתור התוצאה את כל האברים שנמצאים ב 2 התורים
2. נכניס לתור התוצאה את כל האיברים מהתור הראשון ואח"כ השני שעדיין לא נמצאים בתור התוצאה וכך נקבל את אלה שנמצאים בראשון ולא בשני ואח"כ בשי אבל לא בראשון

|  |  |
| --- | --- |
| public static Queue<Integer> order(Queue<Integer> q1,  Queue<Integer> q2)  {  Queue<Integer> res = new Queue<Integer> ();    inScnd(q1, q2, res);    notInRes(q1,res);  notInRes(q2,res);    return res;  } | private static void inScnd(Queue<Integer> q1,  Queue<Integer> q2, Queue<Integer> res)  {  q1.insert(null);  while (q1.head() != null)  {  int num = q1.remove();  if (isIn(q2, num)) {  if (!isIn(res, num))  res.insert(num);  }  else  q1.insert(num);  }  q1.remove();  } |
| private static void notInRes(Queue<Integer> q,  Queue<Integer> res)  {  while (!q.isEmpty()) {  int num = q.remove();  if (!isIn(res, num)) {  res.insert(num);  }  }  } |

1. מה סיבוכיות הפעולה order שכתבתם? **(5 נק)**

עבור 2 תורים 1 באורך m ואחד באורך n.

בפעולה זימונים ל 2 פרוצדורות. ננתח תחילה את הסיבוכיות שלהם:

1. inScnd – עבור 2 תורים אחד באורך m ואחד באורך n הפעולה עוברת על התור הראשון (באורך n) ועבור כל איבר בו מזמנת את isIn פעמיים עם התור השני. (באורך m). הסיבוכיות של isIn היא O(m) ולכן יש לנו סיבוכיות כוללת של  
    O(n) \*O(2m)= O(m\*n). ובהנחה ש n>m O(n2)
2. noInRes- עבור 2 תורים אחד באורך m ואחד באורך n הפעולה עוברת על התור הראשון (באורך n) ועבור כל איבר בו מזמנת את isIn עם התור השני. (באורך m). הסיבוכיות של isIn היא O(m) ולכן יש לנו סיבוכיות כוללת של  
    O(n) \*O(m)= O(m\*n). ובהנחה ש n>m O(n2)

הפעולה מזמנת את inScnd פעם אחת ואת notInRes פעמיים. לכן הסיבוכיות הכוללת שלה:  
3\*O(n2)

**שאלה 3 (50 נק)**

תור תווים עולה הוא תור תווים המכיל רצפים של תווים כשאורך כל רצף גדול מאורך זה שלפניו

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ‘k’ | ‘k’ | ‘k’ | ‘k’ | ‘k’ | ‘b’ | ‘b’ | ‘b’ | ’z’ | 🡪 ראש התור |

לדוגמא:

|  |  |
| --- | --- |
| public static boolean isUp(Queue<Character> q)  { boolean isUp = true;  int prevCount = 0;  q.insert(null);  while (q.head() != null)  {  char c = q.remove();  q.insert(c);  int currentCount = countSegment(q, c);    if (currentCount <= prevCount)  isUp = false;  prevCount = currentCount;  }  q.remove();  return isUp;  } | נוסיף פעולת עזר שתספור את התוים בסגמנט  private static int countSegment(Queue<Character> q, char c)  {  int count = 1;  while (q.head() != null && q.head() == c)  {  q.insert(q.remove());  count++;  }  return count;  } |

1. כתבו פעולה בשם **isUp** המקבלת תור תווים ומחזירה אמת אם הוא תור תווים עולה ושקר אחרת **(20 נק)**
2. מה סיבוכיות הפעולה isUp שכתבתם? **(5 נק)**

עבור תור באורך n הםפעולה isUP עוברת על התור ובעזרת הפעול isUp סופרת את מספר האיברים בכל מקטע. 2 הפעולות ביחד סורקות את התור ובודקות עבור כל איבר אם הוא במקטע הנוכחי או הבא מונה את אברי המקטע ומשווה לכמות אברי המקטע הקודם כלומר לכל איבר בסך הכל 3 פעולות ולכן הסיבוכיות הכוללת O(n)

1. כתבו פעולה בשם fixUp המקבלת תור תווים. אם הוא תור תוים עולה הפעולה אינה משנה אותו. אם הוא אינו תור תווים עולה הפעולה מתקנת אותו, כך שיהיה תור תווים עולה, על ידי הוספת תווים לרצפים קצרים מדי **(25 נק)**

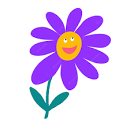
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ‘k’ | ‘k’ | ‘k’ | ‘k’ | ‘b’ | ‘b’ | ‘z’ | ‘z’ | ’z’ | 🡪 ראש התור |

לדוגמא התור:

יהפוך ל

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ‘k’ | ‘k’ | ‘k’ | ‘k’ | ‘k’ | ‘b’ | ‘b’ | ‘b’ | ‘b’ | ‘z’ | ‘z’ | ’z’ | 🡪 ראש התור |

|  |  |
| --- | --- |
| public static void fixUp(Queue<Character> q)  {  if (isUp(q))  return;  int prevCount = 0;  q.insert(null);  while (q.head() != null) {  char c = q.remove();  q.insert(c);  int currentCount = countSegment(q, c);    if (currentCount <= prevCount)  currentCount = fixSegment(q,  currentCount, prevCount, c);  prevCount = currentCount;  }  q.remove();  } | נוסיף פעולת עזרת שמתקנת סגמנט  private static int fixSegment(Queue<Character>q, int currentCount,  int prevCount, char c)  {  while (currentCount <= prevCount)  {  q.insert(c);  currentCount++;  }  return currentCount;  } |

בהצלחה