מבני נתונים – תרגיל 4 (מעשי)

נושא העבודה: המשך הכרת Java, טבלאות גיבוב ועצי AVL.

תאריך פרסום: 21.05.2023

תאריך הגשה: 23:59, 11.6.2023

מתרגל אחראי: תומר מאירמן

הנחיות:

- **שני דגשים חשובים השונים במטלה זו:**
  - **את המטלה יהיה ניתן להגיש החל מהשבוע הבא (ה28.5) ולא לפני כן, ייפתח מקום ייעודי במודל.**
  - **עבור המטלה יהיו 5 הגשות בלבד ללא הורדת ציון. הקבצים של ההרצה שקיבלתם הינם פשוטים יחסית ונועדו לוודא כי הקבצי קוד יתקמפלו ולא מעבר. לכן, יש ליצור טסטים מתאימים לוודא את נכונות מבני הנתונים אותם יצרתם.**
- יש לקרוא הנחיות לגבי הגשת העבודה באתר הקורס.
- כל סטודנט נדרש להגיש עבודות לבד. אין לעבוד בזוגות או בקבוצות גדולות יותר.
- **אין להעתיק עבודות או חלקי עבודות מתלמידים אחרים או מהאינטרנט. אסור השימוש בChatGPT לכתיבת קוד עבורכם.**
- אין לשלוח עבודות או קטעי עבודות לתלמידים אחרים. מותר ואף מומלץ לדון בבעיות מימוש וכן להציע פתרון לבעיות מקומיות. אך אין לשלוח, לשתף או להראות קטע קוד הפותר את הבעיה.
- שימו לב כי העבודה תיבדק אוטומטית, אך תיבדק גם צורת כתיבה של הקוד שלכם (כפי שנדרשתם בעבודה השנייה) באופן ידני ובצורה מדגמית.
- שאלות לגבי העבודה יש לשאול בפורום באתר הקורס או בשעות קבלה של תומר מאירמן (ימי חמישי, 10 וחצי – 11 וחצי בבוקר בזום, בתיאום מראש).
- תיאור המחלקות הנתונות והדרושות למימוש מופיע בסוף המסמך.
- יש לקרוא את כל המסמך טרם תחילת העבודה.

מבוא

המטרה של העבודה היא שיפור ההבנה בתהליכים של טבלאות גיבוב ועצי AVL.

כמו כן, מטרה חשובה לא פחות הינה יישום שילוב של מספר מבני נתונים ליצירת מבנה נתונים מאוחד ופשוט לשימוש.

שימו לב, ניתנו במטלה זו הפונקציות המינימליות שיש לממש (כמו ב interface), אך לשם המימוש המלא תצטרכו להוסיף פונקציות נוספות שלהן אין טסטים אוטומטים (רק טסטים שתכתבו לבד). אין להוסיף עוד משתנים למבני הנתונים (אחרת הטסטים יכשלו) .

<u>סיפור רקע – חלק 1</u>

מרלין הקוסם החליט שהוא רוצה ללמוד את הקסמים שיש מכל סיפורי הפנטזיה, ואיזה לחש יש להגיד בשביל לבצע כל אחד מהם. כמובן, שמרלין ידע כי ישנם הרבה קסמים והוא אולי יכול לזכור את השמות, אך הלחשים עלולים להיות מורכבים יותר ולכן רצה לאחסן אותם במבנה נתונים מתאים.

לשם כך, הוא החליט להשתמש בטבלת גיבוב עם Double hashing. המפתח יהיה שם הקסם, והערך יהיה הלחש אותו יש להגיד בכדי לבצע את הקסם. כאשר מרלין ירצה לבצע קסם כלשהו, הוא יצטרך לחפש בטבלת הגיבוב לפי השם, ולקבל את הלחש המתאים.

לא נרצה לממש מחיקה מכיוון שמרלין מעוניין אך ורק להוסיף ידע.

<u>מחלקות למימוש – טבלת גיבוב עם Double Hash</u>

בחלק הראשון עליכם לממש שתי מחלקות:

הראשונה – המחלקה `SpellSimple` – מחלקה המייצגת את ה struct עבור קסם.

השנייה – `DoubleHashTable` – מחלקה המייצגת את הטבלת הגיבוב עם double hash. הפונקציות ישתמשו בערכי ה ASCII של התווים בקסם.

בהינתן String name (שם הקסם) , וגודל הטבלה שמוגדר (גודל הטבלה הינו מספר ראשוני גדול מ2) מראש (int capacity) פונקציות הגיבוב אותן תצטרכו לממש הינן:

```
1) h1 = for char in name:
        hash = hash + char*31
   return (hash % capacity)
2) h2 = for char in name:
        hash = hash + char*13
   return (1 + hash % (capacity-2))
```

בהינתן ש h1 הוא התוצאה של הפונקציה הראשונה, h2 יהיה שווה לתוצאה של הפונקציה השנייה והצעד הוא i יהיה (עד שיימצא תא ריק):

```
index = (h1+i*h2)%capacity
```

כחלק מהמימוש תצטרכו לשמור ערך `steps` שישמור את מספר הצעדים שבוצעו בפעולת ההכנסה או החיפוש האחרונה **בלבד**. אם לא עשיתם צעד נוסף, כלומר, התוצאה הנדרשת הייתה בתא המחושב ע"י h1 בלבד, מספר הצעדים יתחיל מ-0 (מכאן - רק אם השתמשתם בתוצאה של h2 כחלק מהחישוב הערך יהיה גדול מ- 0).

DoubleHashTable class functions and properties:

This class represents the hash table, where the hash function is calculated in hash1 and hash2 (using ASCII value of characters). Each bucket in the table can hold SpellSimple struct.

- `public DoubleHashTable(int capacity)` – Constructor for DoubleHashTable class. Starts with empty table, size of the table is chosen by the input "capacity".
- `private SpellSimple[] table` - table of SpellSimple struct.
- `private int capacity` – capacity of the hash table, table capacity is assumed to be a prime number, at least 3.
- `private int size` – number of filled buckets.
- `private int steps = 0` – number of steps performed on latest `put / getCastWords` function call. Initialized with value 0.
- `public int getSize()` – Returns the number of spells currently in the hash table.
- `public int getLastSteps()` – Returns the number of steps performed in the last put or getCastWords action.
- `public boolean put(SpellSimple spell)` – inserting new spell struct to the table (based on hash functions), returns True if succeeds or False if not.
- `public String getCastWords(String name)` – returns the "words" to cast the spell.
- `private int hash1(String name)` – returns the result of hash function 1.
- `private int hash2(String name)` - returns the result of hash function 2.

SpellSimple class functions and properties:

This class represents the `SpellSimple` struct.

- `SpellSimple(String name, String words)` – class constructor
- `private String name` – stores the name of the spell
- `private String words` – stores the words required to cast the spell
- `public String getName() –` Returns the spell name
- `public String getWords() –` Returns the words to cast the spell

כדי לבדוק שאכן הפונקציות ממומשות כהלכה, ניתן להריץ באופן מקומי את הקובץ MainPart1.java.
יש לממש טסטים הבודקים את תקינות מבני הנתונים. כמו כן, יש לממש טסטים הבודקים את תקינות הפונקציות השונות.

<u>סיפור רקע – חלק 2</u>

מרלין הבין שהמשימה מסובכת ומספר הקסמים שהוא משיג הינו דינאמי. בנוסף, ישנם קסמים רבים ודרכים שונות לסדר אותם. בשלב זה הוא החליט שיש עליו להשקיע במבנה הנתונים יותר בכדי שיוכל לאחסן מספר לא מוגבל של קסמים, וכמו כן לסדר אותם בצורה חכמה יותר.

לשם כך, נעזור לו וניצור מבנה נתונים חדש – `HashAVLSpellTable`, טבלאת גיבוב עם שרשור עצי AVL. בטבלת הגיבוב בכל תא תהייה רשימה מקושרת של עצי AVL (את הרשימה המקושרת אין צורך לממש), כאשר כל עץ ייצג קטגוריה שונה של קסמים (לדוגמה, עץ נפרד עבור כל סוג של קסמים כמו אש, קרח וברק). הסדר בעץ יהיה לפי רמות ה'כוח' של כל קסם, כך שמרלין יוכל לשלוף מהר קסמים חזקים מכל קטגוריה שיחפוץ בה. נרצה גם אפשרות להחזיר את k הקסמים החזקים ביותר מכל קטגוריה.

שימו לב, תדרשו להחזיר את k הקסמים החזקים מהגדול לקטן (במערך שיוחזר מהפונקציה, הערך במקום ה- 0 יהיה הגדול ביותר לפי ה powerLevel), ללא שימוש בשיטות מיון של מערך.

גם במקרה זה **לא נרצה לממש מחיקה** מכיוון שמרלין מעוניין אך ורק להוסיף ידע.

<u>מחלקות למימוש – טבלת גיבוב עם שרשור עצי AVL</u>

בחלק זה תצטרכו לממש 3 מחלקות שונות (לא בהכרח בסדר שהן מוצגות במסמך העבודה):

הראשונה – `Spell`, הפעם, במחלקה נצטרך לאחסן בנוסף לשם והלחש, גם את המשתנים קטגוריה ורמת כוח.

השנייה – `AVLTree` – בה ניצור את עץ הAVL המייצג את הקסמים מכל קטגוריה שונה. במחלקה כל Node יכיל אובייקט מסוג Spell במקום value, וההכנסה והמיון יתבצעו לפי הרמת הכוח של הקסמים מאותה קטגוריה. **ניתן להניח שכל הpowerLevel** (רמת כוח) של אותה קטגוריה **הם שונים**.

השלישית – `HashAVLSpellTable` – במחלקה זו ניצור את טבלת הגיבוב, כאשר כל תא יהיה רשימה מקושרת (בדומה ל chaining), שברשימת המקושרת כל חוליה תהיה `AVLTree`, כך שאם לקטגוריות יש את אותו ה hash הן יהיו עצים שונים בתאים שונים ב-LinkedList (את הרשימה המקושרת לא נממש ונשתמש ב `java.util.LinkedList`). מחלקה זו תהייה המחלקה בה מרלין (משתמש הקצה) ישתמש לאחסון הקסמים, ובה נרצה להנגיש אפשרות להוסיף קסמים חדשים שמרלין מוצא, לבדוק אם קסם קיים, מה הלחש שלו, וכמובן לקבל את k הקסמים החזקים בקטגוריה מסוימת.

בפונקציה המוצאת את k הקסמים החזקים – יש עליכם להחזיר רשימה המחזיקה את הקסמים מהגדול לקטן. כחלק מפתרון חלק זה **אסור לממש או להשתמש במיון** (פתרון המשתמש במיון יאבד את הניקוד על פונקציה זו שהינה מרכזית בתרגיל).

פונקציית ה hash הנדרשת עבור המחלקה הינה (בשימוש בערך קטגוריה וערך גודל הטבלה):

```
for char in category:
      hash = hash + char
return (hash % tableSize)
```

Spell class functions and properties:

This class represents the Spell Struct. Comparing the previous part, more properties and functions are added.

- `public Spell(String name, String category, int powerLevel, String words)` – class constructor, init the name, words category and power level based on the input.
- `private String name` – stores the name of the spell
- `private String words` – stores the words required to cast the spell
- `private String category` – spell's category
- `private int powerLevel` – spell's power level
- `public String getName()` – getter for name
- `public String getCategory()` – getter for category
- `public int getPowerLevel()` – getter for power level
- `public String toString()` - overriding toString, already implemented for tests
    - `{ return name + " (" + category + ") - Power Level: " + powerLevel + ", to cast say: " + words; }`

AVLTree class functions and properties:

This class represents an AVL Tree as learned in class, where the nodes are ordered by the power level of the spell.

In this class, you need to implement an internal private class which represents the nodes of the AVL tree:

- private class **Node**
    - `private Node(Spell spell)` - constructor of the private node class
    - `private Spell spell` – replace the value of the node
    - `private Node left` – pointer to left son
    - `private Node right` – pointer to the right son
    - `private int height` – height of current node

Properties of the AVLTree class:

- `private Node root` – holds a pointer to the root of the AVL tree
- `private int size` - holds the number of spells stored in the AVL tree
- `private String category` – a category that the AVL tree represents

AVLTree Functions:

- `public AVLTree(Spell spell)` – tree constructor
- `public int getSize` – getter of tree size
- `public int getTreeHeight` – returns the tree's height
- `public String getCategory()` – return the tree category
- `public Spell search(String spellName, int powerLevel)` – search for a spell based on the name and the power level (the category is given in the `HashAVLSpellTable` class)
- `public void insert(Spell spell)` – insert new spell to the `AVLTree` (you need to assure balancing based on what you studied in class.
- `public List<Spell> getTopK(int k)` – Returns list of top-k spells in the tree (category is given in `HashAVLSpellTable` class), requires `java.util.List` and `ArrayList`. The order must be from the **largest powerLevel to the lowest**, **without creating an additional function that sorts the List**. If k > tree_size, return all spells of the category.

HashAVLSpellTable class functions and properties:

This class represents a hash table where each bucket is a linked list, and each node in the linked list is an AVL tree as implemented in previous class.

- `public HashAVLSpellTable(int size)` - constructor, initialize the table where the size is the number of buckets required (length of buckets[]).
- `private int tableSize` – the number of buckets in the hash table (length of buckets[]).
- `private int numSpells` – total number of spells inside the data structure.
- `private LinkedList<AVLTree> buckets[]` – array of LinkedLists, where each link is represented by the AVL Tree object. requires import of `java.util.LinkedList`
- `private int hash(String category)` – returns the result of the hash function (**pay attention:** it is different then on the first part of the assignment)
- `public void addSpell(Spell s)` - add a spell to the hash table
- `public Spell searchSpell(String category, String spellName, int powerLevel)` - search for a spell by category, spell name , and powerLevel

- `public int getNumberOfSpells()` – getter for the number of spells that exist in the entire data structure (without category).
- `public int getNumberSpells(String category)` – getter for the number of spells that exist for the input category.
- `public List<Spell> getTopK(String category, int k)` – returns the top-k spells (based on power level) for the input category. Requires import of `java.util.List`. If k > tree_size, return all spells of the category.

כדי לבדוק שאכן הפונקציות ממומשות כהלכה, ניתן להריץ באופן מקומי את הקובץ MainPart2.jav.
יש לממש טסטים הבודקים את תקינות מבני הנתונים. כמו כן, יש לממש טסטים הבודקים את תקינות הפונקציות השונות.