חזרה בנושא אוספים גנריים

ווקטור

מטרות

תרגול נושא האוספים וייצוגם.

תרגול נושא יעילות של פעולות בהתייחס לבחירת ייצוג.

רמת השאלה

שאלת סיכום העוסקת במגוון נושאים: אוסף גנרי, בחירת ייצוג, מימוש, מגבלות יעילות וחישובי יעילות.

שאלה זו יכולה לשמש לחזרה על החומר והעמקה בהבנתו החל משלב לימוד האוסף הראשון ביחידה, אך היא מורכבת מכדי להינתן בזמן מבחן. אם ברצונכם להשתמש בה לבחינה, יש לצמצם את הדרישות הכלולות בה.

השאלה

ווקטור הוא אוסף גנרי של ערכים, שהגישה אליהם נעשית באמצעות מקומם הסידורי באוסף (הערך הראשון בווקטור נמצא במקום הסידורי **1**). המחלקה **Vector** מגדירה את האוסף הגנרי ווקטור.

להלן ממשק המחלקה **Vector<T>**:

`Vector()`	הפעולה בונה ווקטור ריק
void `add(T x)`	הפעולה מוסיפה את הערך x כערך אחרון בווקטור.
void `del(`**int** `place)`	הפעולה מוחקת את הערך שמיקומו הסידורי הוא place. **הנחה**: המקום place קיים בווקטור
`T get(`**int** `place)`	הפעולה מחזירה את הערך שנמצא במקום הסידורי place. **הנחה**: המקום place קיים בווקטור
void `set(`**int** `place, T x)`	הפעולה מעדכנת את הערך שנמצא במקום הסידורי place, להיות הערך x. **הנחה**: המקום place קיים בווקטור
int `length()`	הפעולה מחזירה את מספר הערכים בווקטור
`String toString()`	הפעולה מחזירה מחרוזת המתארת את הווקטור באופן הבא: value1, value2, value3, value4 . . .

מה עליכם לעשות?

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

א) כתבו את המחלקה Vector<T> כך שהפעולות **add** ו-**length** יתבצעו בסדר גודל **קבוע** = ***O(1)***.

ב) נתחו את היעילות של שאר הפעולות במחלקה Vector: עבור כל אחת רשמו את סדר הגודל של זמן הריצה שלה, כולל הסבר.

ג) ממשו את הפעולה החיצונית הבאה:

```
public static Vector<Integer> getKSmallestValues
                                (Vector<Integer> vec, int k)
```

הפעולה מחזירה ווקטור שבו יופיעו k הערכים הקטנים ביותר שבווקטור vec. הניחו ש-k לא גדול מ-vec.length(). הווקטור vec לא ישתנה בעקבות הפעולה.

ד) נתחו את יעילות הפעולה `getKSmallestValues(…)` שכתבתם.

הנחיות מיוחדות

1. שאלה זו בודקת את היכולת של התלמיד לבנות מחלקה המגדירה אוסף כללי.
2. הפעולה del (…) מוחקת איבר ולמעשה מקומם של כל האיברים הבאים אחריו "מתעדכן" בהתאם. יש לוודא שהתלמידים מבינים שלא נוצרים "חורים" בווקטור. כלומר, אם האיבר השלישי נמחק (הגישה היא דרך האינדקס!), האיבר הרביעי הופך להיות השלישי וכן הלאה!
3. כדי לממש את הפעולות שבמחלקה, יש תחילה לבחור ייצוג מתאים לווקטור. ההגבלה על יעילות זמן הריצה של פעולות מסוימות מכַוֶנת לבחור בייצוגים מסוימים ומבטלת ייצוגים אחרים. ייצוגים סבירים אפשריים יהיו: שרשרת חוליות, רשימה ועוד. (ייצוגים שונים מצורפים לשאלה זו לשימושכם).
ברור שהייצוג הנדרש צריך לאפשר שמירה של אוסף דינמי. מערך אינו פתרון מתאים.
דיון אפשרי: כיון שהגבלת סדר הגודל של זמן הריצה מתייחסת לפעולת הוספה, הרי שייצוג בעזרת מערך אינו רלוונטי. איחסון כמות בלתי מוגבלת של איברים במערך הוא פעולה יקרה מ-*O(1)*, אם כי פעולות עדכון והחזרת ערך יעילות מאד במערך. גם פעולת המחיקה אינה יעילה במערך מכיוון שעלינו ל"סגור" את החורים שנוצרו במערך. כדאי להעלות נקודות אלו לדיון בכיתה.

- מימוש פעולות הממשק של הוקטור מסתמך על הייצוג שנבחר. רוב הקושי בו ייתקל התלמיד במימוש הפעולות הוא בהמרת המיקום של איבר בוקטור, שנתון כמספר סידורי, למיקום של האיבר בוקטור המיוצג בעזרת שרשרת חוליות או רשימה – בהם המיקום הוא הפנייה לאיבר.
- כדי לממש את הפעולות length() ו-add(...) בסדר גודל קבוע *O(1)*, יש צורך בהוספת שתי תכונות למחלקה:
  1. תכונה שתמנה את מספר האיברים.
  2. תכונה שתשמור את המיקום אחריו יוכנס איבר חדש.

4. ייצוג טוב יהיה בעזרת שרשרת חוליות, כאשר למחלקה שלוש תכונות: הפניה לראש שרשרת החוליות, הפנייה לסוף השרשרת (מאפשר לבצע את פעולת ההוספה ב-*O(1)*) ואורך הווקטור. מובן שצריך יהיה לבדוק בכל פעולה, אלו תכונות צריכות להתעדכן (length כשמוסיפים או מוחקים איבר, first בהוצאת איבר מווקטור שבו איבר יחיד וכד'). אמנם עדכונים אלו לוקחים זמן אך אין בכך פגיעה ביעילות הפעולה, שכן העדכונים אורכם קבוע ואינו תלוי באורך הקלט (מספר האיברים בווקטור) שהוא הקובע את סדר הגודל של הפעולות.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

5. טיפול במקרי קצה. בכל פעולה יש לבדוק אם קיימים מקרי קצה שיש להתייחס אליהם. למשל: ווקטור שבו איבר יחיד צריך לקבל טיפול מיוחד בפעולה del וכד'.
6. ניתן לקיים דיון בשאלה: האם הגיוני לממש פעולה בונה מעתיקה במחלקה <Vector<T>? מה המשמעות של העתקת האיברים אם הם אינם מטיפוסים פשוטים... הדיון יגיע לכך שבכלים העומדים לרשותנו כעת, אפשרית רק העתקה "שטוחה" של ההפניות שבווקטור ולא שכפול העצמים השמורים בו (deep copy).
7. לגבי סעיף ג בשאלה. כדי שהווקטור עצמו לא ישתנה, יש לבנות ווקטור חדש ולהחזירו מהפעולה. ביצוע הפעולות על גבי ההפניה שנשלחה כפרמטר, יגרום לשינוי הווקטור המקורי למרות שההפניה עצמה, vec, מתבטלת בתום ביצוע הפעולה.

פתרון

א. פתרון ראשון המייצג את הוקטור בעזרת שרשרת חוליות מצורף בתיקיה.

פתרונות נוספים המצורפים בתיקייה:

ב. המחלקה Vector מיוצגת בעזרת רשימה:

```
public class Vector<T>
{
	private List<T> list;
	private Node<T> last;
	private int numValues;

	public Vector()
	{
		this.list = new List<T>();
		this.last = null;
		this.numValues = 0;
	}

	public void add(T x)
	{
		Node<T> pos = null;

		if(!this.list.isEmpty())
			pos = this.last;

		this.last = this.list.insert(pos, x);
		this.numValues++;
	}

	public void del(int place)
	{
		if(place==1)
		{
			if(this.last==this.list.getFirst())
				this.last = null;
			this.list.remove(this.list.getFirst());
		}
		else
		{
			Node<T> prev = this.list.getFirst();

			for(int i=1; i<=place-2; i++)
				prev = prev.getNext();

			if(this.last==prev.getNext())
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

```
                        this.last = prev;
                    this.list.remove(prev.getNext());
                }

                this.numValues--;
        }

        public T get(int place)
        {
                Node<T> pos = this.list.getFirst();

                for(int i=1; i<=place-1; i++)
                        pos = pos.getNext();

                return(pos.getInfo());
        }

        public void set(int place, T x)
        {
                Node<T> pos = this.list.getFirst();

                for(int i=1; i<=place-1; i++)
                        pos = pos.getNext();

                pos.setInfo(x);
        }

        public int length()
        {
                return(this.numValues);
        }

        public String toString()
        {
                String str = new String();

                if(!this.list.isEmpty())
                {
                        Node<T> pos = this.list.getFirst();

                        while(pos!=this.last)
                        {
                                str = str + pos.getInfo().toString() + ",";
                                pos = pos.getNext();
                        }
                        str = str + last.getInfo().toString();
                }
                return(str);
        }
}
```

הרעיון בפתרון זה הוא להכניס את האיבר החדש לסוף הרשימה ולכן יש צורך בשמירת ההפניה לאיבר האחרון. אפשרות אחרת לכתיבת המחלקה וקטור מתבססת על הרעיון שהכנסת האיברים תתבצע בצורה הפוכה – כלומר לתחילת הרשימה וכך אין צורך בתכונה שתשמור את ההפניה לאיבר האחרון בוקטור. במקרה זה נצטרך לטפל במיקום האיברים בצורה הפוכה.

ג. הנה המחלקה Vector על פי רעיון זה:

```
public class Vector<T>
{
        private List<T> list;
        private int numOfItems;

        public Vector()
        {
                this.list = new List<T>();
                this.numOfItems = 0;
        }

        public void add(T x)
        {
```


© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

```
		this.list.insert(null, x);
		this.numOfItems++;
	}

	public void del(int place)
	{
		this.list.remove(this.getItemPos(place));
		this.numOfItems--;
	}

	public T get(int place)
	{
		return(this.getItemPos(place).getInfo());
	}

	public void set(int place, T x)
	{
		this.getItemPos(place).setInfo(x);
	}

	public int length()
	{
		return(this.numOfItems);
	}

	public String toString()
	{
		Stack<T> stk = new Stack<T>();
		Node<T> pos = this.list.getFirst();

		while(pos != null)
		{
			stk.push(pos.getInfo());
			pos = pos.getNext();
		}

		String str = "";

		if(!stk.isEmpty())
			str = str + stk.pop();
		while(!stk.isEmpty())
			str = str + "," + stk.pop();

		return(str);
	}

	// הנחה: קיים איבר בוקטור שמיקומו מתקבל
	private Node<T> getItemPos(int place)
	{
		Node<T> pos = this.list.getFirst();

		for(int i=1; i<=this.numOfItems-place; i++)
			pos = pos.getNext();

		return(pos);
	}
}
```

שימו לב:

1. לשימוש בפעולת העזר הפרטית getItemPos(...) שתפקידה להמיר מיקום כמספר למיקום כהפניה.
2. למימוש הפעולה toString(), המשתמשת במחסנית לצורך היפוך האיברים.

ב. ניתוח יעילות פעולות הממשק:

- כאמור , יעילות הפעולות length(...), add(...) היא מסדר גודל קבוע *O(1)*.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים

- יעילות שאר הפעולות היא מסדר גודל לינארי $O(n)$ כאשר n הוא מספר האיברים בוקטור.

ג. מימוש הפעולה:

```
public static Vector<Integer> sort(Vector<Integer> vec)
{
      Vector<Integer> sortedVec = new Vector<Integer>();

      for(int i=1; i<=vec.length(); i++)
            sortedVec.add(vec.get(i));

      for(int i=1; i<sortedVec.length(); i++)
            for(int j=1; j<=sortedVec.length()-i; j++)
                  if(sortedVec.get(j)>sortedVec.get(j+1))
                  {
                        int tmp = sortedVec.get(j);
                        sortedVec.set(j, sortedVec.get(j+1));
                        sortedVec.set(j+1, tmp);
                  }

      return sortedVec;
}
```

ד. פעולה בונה מעתיקה:

```
public Vector(Vector<T> vec)
{
      this(); // זימון הפעולה בונה ריקה
      for (int i = 1; i <= vec.numValues; i++)
            this.add(vec.get(i));
}
```

מקור השאלה

עיבוד של השאלה מפרק רשימה החדש, עיצוב תוכנה מבוסס עצמים, 2007.

© כל הזכויות שמורות למשרד החינוך ולאוניברסיטה העברית בירושלים