בחינת מועד א - פתרון

שאלה 1 [30 נק']. עיצוב דף פירסום

בדף-אינטרנט מסויים יש מקום לפירסומות. אפשר לשים בו פירסומת אחת ארוכה **או** שתי פירסומות קצרות *[של מפרסמים שונים]*. ישנם מספר מפרסמים המתחרים על מקום בדף. ההעדפות של כל מפרסם מיוצגות ע"י המחלקה:

```
class Advertiser {
      float  longvalue;
         // כמה המפרסם מרויח (בשקלים) אם פרסומת ארוכה שלו מופיעה בדף
      float shortvalue;
         // כמה המפרסם מרויח (בשקלים) אם פרסומת קצרה שלו מופיעה בדף
};
```

מפרסם שאינו מופיע בדף מרויח 0 שקלים. מנהלי האתר רוצים שהמפרסמים יהיו מרוצים - המטרה שלהם היא **למקסם את סכום רווחי המפרסמים**. עיזרו למנהלי האתר להחליט איזה פירסומות לשים באתר!

א [10 נק']. כיתבו אלגוריתם, בעברית או בפסאודו-קוד, המקבל כקלט את רשימת המפרסמים, ומחזיר כפלט כמה פירסומות יהיו בעמוד (אחת או שתיים), ואיזה מפרסם/מפרסמים יופיעו בעמוד.

פתרון:

(1) מוצאים את המפרסם i עם ה-longvalue הכי גדול.

(2) מוצאים את שני המפרסמים j,k עם ה-shortvalue הכי גדול.

(3) אם ה-longvalue של מפרסם i גדול מסכום ה-shortvalue של מפרסמים j,k -
אז מציגים פרסומת אחת ארוכה של מפרסם i.

(4) אחרת - מציגים שתי פרסומות קצרות של מפרסמים j,k.

הערה: *היו סטודנטים שהניחו שמותר לפרסם שתי פרסומות קצרות של אותו מפרסם. הם קיבלו את מלוא הנקודות - בתנאי שהפתרון שלהם היה נכון לפי הנחה זאת.*

ב [10 נק']. כיתבו מנגנון הממקסם את סכום רווחי המפרסמים, וגם מעודד כל מפרסם לחשוף את הערכים longvalue, shortvalue **האמיתיים** שלו. המנגנון מקבל כקלט את רשימת המפרסמים, ומחזיר כפלט: כמה פירסומות יהיו בעמוד, איזה מפרסם/מפרסמים יופיעו בעמוד, ואיזה מחיר ישלם/ישלמו המפרסם/מפרסמים למנהל האתר (בשקלים). אין להשתמש במילים "כפי שראינו בכיתה" אלא לפרט את אופן החישוב.

הערה: אם מפרסם מסויים מופיע בדף ומשלם מחיר, התועלת שלו היא הרווח מהפירסום פחות המחיר.

פתרון: *המטרה כאן היא להפוך אלגוריתם למנגנון - להפוך את האלגוריתם של סעיף א למנגנון אמיתי. המשתתפים הם רב-פרמטריים (לכל אחד יש שני פרמטרים - shortvalue,longvalue), ואנחנו רוצים למקסם את סכום הערכים שלהם. לכן, המנגנון היחיד שיכול להתאים כאן הוא VCG (מיירסון מתאים רק למשתתפים חד-פרמטריים). המנגנון פועל באופן הבא:*

(1) משתמשים באלגוריתם של סעיף א כדי לחשב כמה ואיזה פרסומות להציג.

(2) עבור כל מפרסם i, מחשבים:

(א) מה סכום הרווחים של המפרסמים האחרים בתוצאה של שלב (1).

(ב) מה היה סכום הרווחים של המפרסמים האחרים,

אילו היינו מריצים את האלגוריתם של סעיף א בלי מפרסם i.

גובים ממפרסם i את ההפרש - סכום (ב) פחות סכום (א).

ג [10 נק']. הדגימו בפירוט את פעולת המנגנון שכתבתם על הקלט הבא, ובו ארבעה מפרסמים:

```
ad[0].longvalue=10;  ad[0].shortvalue=8;

ad[1].longvalue=9;   ad[1].shortvalue=1;

ad[2].longvalue=8;   ad[2].shortvalue=4;

ad[3].longvalue=7;   ad[3].shortvalue=3;
```

***פתרון**: המטרה כאן היא להפוך אלגוריתם למנגנון - להפוך את האלגוריתם של סעיף א למנגנון אמיתי. המשתתפים הם רב-פרמטריים (לכל אחד יש שני פרמטרים - shortvalue,longvalue), ואנחנו רוצים למקסם את סכום הערכים שלהם. לכן, המנגנון היחיד שיכול להתאים כאן הוא VCG. המנגנון פועל באופן הבא:*

*(1) מציגים **שתי פרסומות קצרות - של מפרסמים 0, 2.***

*(2) עבור מפרסם 0: סכום הרווחים של האחרים הוא 4; אילו לא היה משתתף היינו מציגים פרסומת ארוכה של מפרסם 1 והסכום היה 9; לכן **מפרסם 0 משלם 5**.*

מפרסם 1 לא משלם כלום - התוצאה זהה איתו או בלעדיו.

*עבור מפרסם 2: סכום הרווחים של האחרים הוא 8; אילו לא היה משתתף היינו מציגים שתי פרסומות קצרות של מפרסמים 0,3 והסכום היה 11; לכן **מפרסם 2 משלם 3**.*

מפרסם 3 לא משלים כלום - התוצאה זהה איתו או בלעדיו.

שאלה 2 [30 נק']. חלוקה של תכשיטים ללא קנאה

הצורף הידוע הנס שטרן (מייסד חברת התכשיטים H.Stern) נפטר בשיבה טובה והוריש את אוסף אבני-החן שלו לארבעת ילדיו רוברטו, ריקרדו, רונלדו ורפאל. לכל אחד מהילדים ישנן העדפות שונות לגבי אבני-החן: חלק אוהבים יותר יהלומים, חלקם אוהבים יותר טופזים, וכו'... התבקשתם לעזור להם בחלוקת הירושה.

א [10 נק']. תארו אלגוריתם לחלוקת הירושה בין ארבעת הילדים, עם התכונות הבאות: אף ילד לא יקנא בילדים האחרים; מותר לחתוך לכל היותר שלוש אבני-חן - כל שאר האבנים חייבות להישאר שלמות. מותר להשתמש באלגוריתמים שנלמדו בכיתה, אך יש להוכיח שהם אכן פותרים את הבעיה.

פתרון:

(1) מסדרים את כל האבנים בשורה בסדר שרירותי.

(2) מתייחסים לשורת האבנים כאל עוגה חד-מימדית המחולקת לאיזורים.

(3) משתמשים באלגוריתם הסימפלקסונים של סו ומוצאים חלוקה קשירה וללא קנאה של העוגה.

החלוקה מחזירה 4 פרוסות קשירות, לכן ניתן לבצעה בעזרת שלושה חתכים, לכן נחתכות שלוש אבנים לכל היותר.

הערה: *אלגוריתם הסימפלקסונים של סו מוצא, בכל זמן סופי, חלוקה ללא-קנאה-בקירוב. ניתן להשיג קירוב טוב כרצוננו ע"י בחירת גודל הצלע של הסימפלקסונים. אם מריצים את האלגוריתם אינסוף פעמים, עם סימפלקסונים יותר ויותר קטנים, מקבלים סדרת חלוקות המתכנסת לחלוקה ללא קנאה. עם זאת, לצורך השאלה הנוכחית היה מספיק לציין שהחלוקה ללא-קנאה-בקירוב.*

פתרונות לא נכונים: *אלגוריתם "המנצח המתוקן" לא מתאים כי הוא עובד רק עם שני משתתפים. כשיש יותר משני משתתפים, לא ברור מי "המנצח" ומי "המפסיד....*

אלגוריתם "מעגלי הקנאה" מוצא חלוקה שהיא ללא-קנאה עד-כדי חפץ אחד. חלק מהסטודנטים הציעו לחתוך את החפצים הגורמים לקנאה ולתת חלק מהם לזה שמקנא, אבל החלוקה הזאת עלולה ליצור קנאה חדשה - אם ננסה לחתוך חפץ של רוברטו (למשל) ולתת חלק ממנו לריקרדו כדי שלא יקנא, אז ייתכן שעכשיו רוברטו יתחיל לקנא ברפאל... בנוסף, מספר קשתות-הקנאה בגרף עלול להיות 6, כך שנצטרך לחתוך 6 אבנים ולא 3.

אם במקום סימונס-סו משתמשים באלגוריתם אבן-פז או המפחית האחרון - התוצאה היא פרופורציונלית אבל עלולה להיות עם קנאה (כפי שלמדנו בהרצאות).

ב [10 נק']. תנו דוגמה שבה האלגוריתם מסעיף א יחתוך בדיוק שלוש אבני חן. אין צורך לכתוב את פרטי הרצת האלגוריתם אלא רק להראות את הקלט ואת הפלט.

פתרון: *נניח שיש בדיוק 3 אבני-חן. כל אחד מארבעת הילדים מייחס ערך 1 לכל אבן. האלגוריתם יחתוך את העוגה בנקודות 3/4, 6/4, 9/4, כך שכל ילד יקבל בדיוק 3/4 אבן. כל האבנים נחתכות.*

ג [10 נק']. תנו דוגמה שבה האלגוריתם מסעיף א לא יחתוך אף אבני חן. אין צורך לכתוב את פרטי הרצת האלגוריתם אלא רק להראות את הקלט ואת הפלט.

פתרון: *נניח שיש בדיוק 4 אבני-חן. כל אחד מארבעת הילדים מייחס ערך 1 לכל אבן. האלגוריתם יחתוך את העוגה בנקודות 1, 2, 3 וייתן בדיוק אבן אחת שלמה לכל ילד. אין צורך לחתוך אף אבן.*

שאלה 3 [30 נק']. סידור שותפויות להגשת מטלות

בקורס מסויים, מותר להגיש מטלות בזוגות בלבד. סטודנט בלי שותף מקבל אוטומטית 0 על המטלות. המדיניות נאכפת בקשיחות: גם כשמספר הסטודנטים בקורס הוא איזוגי, אין אישור להגיש ביחידים או בשלשות. עליכם לעזור לסטודנטים להסתדר בזוגות.

א [10 נק']. בסעיף זה, לכל סטודנט יש קבוצה של סטודנטים שהוא מוכן לעבוד איתם:

```
class Student {
     set<Student> acceptable;
};
```

עם הסטודנטים שאינם בקבוצה זו, הסטודנט בשום אופן לא מוכן לעבוד - גם אם יקבל 0 *[ואי-אפשר להכריח סטודנטים לעבוד עם מי שהם לא רוצים]*. תארו אלגוריתם, בעברית או בפסאודו-קוד, המוצא סידור **יעיל-פארטו** של סטודנטים לזוגות. מותר להשתמש באלגוריתמים שנלמדו בכיתה, אולם יש להוכיח שהם אכן פותרים את הבעיה.

פתרון:

(1) יוצרים גרף לא-מכוון שבו כל סטודנט הוא צומת, ויש קשת בין שני סטודנטים s, t אם-ורק-אם כל אחד מהם מוכן לעבוד עם השני: `s.acceptable.contains(t) && t.acceptable.contains(s)`.

(2) משתמשים באלגוריתם למציאת שידוך גדול ביותר בגרף כללי - אלגוריתם המסלולים המתחלפים ("אלגוריתם הפרחים") שלמדנו בהקשר של החלפת כליות.

הוכחה שהתוצאה אכן יעילה פארטו: *יהי שידוך א השידוך של האלגוריתם. נניח בשלילה שקיים שידוך ב שהוא שיפור-פארטו שלו. יש סטודנט אחד לפחות, נניח ס, שמצבו בשידוך ב טוב יותר. אז ס לא משודך בשידוך א, אבל כן משודך בשידוך ב. מצד שני, מצבם של כל הסטודנטים בשידוך ב טוב לפחות באותה מידה כמו בשידוך א. אז כל סטודנט שהיה משודך בשידוך א, עדיין משודך. מכאן ששידוך ב גדול יותר משידוך א - אבל זו סתירה לעובדה ששידוך א הוא גדול ביותר.*

פתרונות לא נכונים: *אלגוריתם חמדני כגון "דיקטטורה סדרתית" לא בהכרח יעיל פארטו. לדוגמה, נניח שסטודנט 1 מוכן לעבוד עם 2,3; 2 מוכן לעבוד עם 1, 4; 3 מוכן לעבוד עם 1; 4 מוכן לעבוד עם 2. אז, אלגוריתם חמדני עלול לשדך את 1,2 וישאיר את 3,4 לא משודכים. אבל קיים שיפור פארטו: לשדך את 1,3 ואת 2,4.*

ב [10 נק']. הדגימו בפירוט את פעולת האלגוריתם שכתבתם בסעיף א על הקלט הבא ובו שמונה סטודנטים:

```
students[0].acceptable={1,2,3,6};  students[1].acceptable={0,2,3,5};

students[2].acceptable={0,3};      students[3].acceptable={2,1,4,5};

students[4].acceptable={0,1,2,3};  students[5].acceptable={1,3,6,7};

students[6].acceptable={5,7};      students[7].acceptable={5,6};
```

פתרון: *הגרף המתקבל הוא כמעט זהה לגרף במצגת של שבוע 2, שקפים 19-17 - ראו שם.*

ישנו שידוך מקסימלי בגודל 4, למשל: 02, 15, 34, 67. (לכן, כל שידוך בגודל 3 ומטה אינו יעיל פארטו).

ג [10 נק']. **שימו לב - סעיף זה שונה ובלתי תלוי בשני הסעיפים הקודמים.**

בסעיף זה, כל סטודנט מוכן לעבוד עם כל הסטודנטים האחרים, אבל יש לו **דירוג** - עם מי הוא רוצה לעבוד בעדיפות ראשונה, עם מי בעדיפות שניה, וכו':

```
class Student {
      list<Student> rank;
};
```

הוכיחו ש**לא קיים** אלגוריתם המוצא שידוך **יציב** של סטודנטים לזוגות.

היעזרו בדוגמה הבאה המתארת קורס עם ארבעה סטודנטים - 0, 1, 2, 3:

```
students[0].rank = {1,2,3};
      // כלומר: סטודנט 0 הכי רוצה לעבוד עם סטודנט 1, בעדיפות שניה עם 2, ובעדיפות
שלישית עם 3. //
students[1].rank = {2,3,0};
students[2].rank = {3,0,1};
students[3].rank = {0,1,2};
```

***פתרון**: הרעיון היה להראות שאין שידוך יציב. ישנם שלושה שידוכים אפשריים: 23-01 או 13-02 או 12-03.*
צריך לבדוק כל אחד מהם ולוודא שהוא לא יציב (כלומר שיש בו זוג מערער).

- *23-01: הזוג 13 מערער (1 מעדיף את 3 על 0, 3 מעדיף את 1 על 2).*
- *12-03: הזוג 02 מערער (0 מעדיף את 2 על 3, 2 מעדיף את 0 על 1).*
- *13-02: זה שידוך יציב!!! הייתה טעות בדוגמה!!!*

הדוגמה היתה צריכה להיות:

```
students[0].rank = {1,2,3};
students[1].rank = {2,0,3};
students[2].rank = {0,1,3};
students[3].rank = {0,1,2};
```

בדוגמה זו אכן אף אחד משלושת השידוכים האפשריים אינו יציב:

- *23-01: הזוג 12 מערער (1 מעדיף את 2 על 0, 2 מעדיף כל אחד על 3).*
- *12-03: הזוג 02 מערער (2 מעדיף את 0 על 1, 0 מעדיף כל אחד על 3).*
- *13-02: הזוג 01 מערער (0 מעדיף את 1 על 2, 1 מעדיף כל אחד על 3).*

כל מי שניסה לפתור את הסעיף, והשתמש בהגדרות הנכונות של "יציבות" ו"זוג מערער", קיבל את מלוא הנקודות.
(וכל הכבוד לשני הסטודנטים שהצליחו להמציא דוגמה נכונה בעצמם!)

שאלה 4 [30 נק']. בדיקת תקינות עסקאות בשרשרת בלוקים

המטבע SimpleCoin הוא מטבע חדש (דמיוני), הדומה לביטקוין אבל הרבה יותר פשוט: בכל בלוק בשרשרת יש רק עיסקה אחת; הסכום של כל עיסקה הוא מטבע אחד בדיוק; לכל עיסקה יש רק נִמְעָן אחד (לא מפצלים מטבע לכמה נמענים שונים). ישנם שני סוגי בלוקים: בלוק שבו נוצר מטבע חדש (למשל כשכר לכוֹרֶה), ובלוק שבו מועבר מטבע שנוצר קודם. כל בלוק בשרשרת מיוצג ע"י המחלקה הבאה:

```
class Block {
      Block previous;  // **הקודם בשרשרת** קישור לבלוק.  //
      Block input;     // **הקלט לעיסקה זו** הבלוק הכולל את. //
      Key receiver;  // המפתח הציבורי שאליו מועבר המטבע בעיסקה זו.  //
};
```

עיסקה חדשה מיוצגת ע"י המחלקה הבאה:

```
class Transaction {
      Block input;  // ראו הסבר למעלה
      Key receiver; // ראו הסבר למעלה
      Signature signature;   // חתימה דיגיטלית של שולח העיסקה
};
```

חתימה דיגיטלית מיוצגת ע"י המחלקה הבאה:

```
class Signature {
      bool is_valid(Key signer); // מקבלת כקלט את המפתח הציבורי של החותם
                                 // מחזירה "אמת" אם"ם החתימה תקינה.
};
```

בנוסף, נתון המשתנה הגלובאלי `latest` המייצג את הבלוק **האחרון** בשרשרת (זה שנוצר הכי מאוחר).

א [20 נק']. כיתבו פונקציה הבודקת האם עיסקה נתונה היא חוקית או לא. ניתן להניח שזו עיסקה של העברת מטבע ולא של יצירת מטבע חדש - כלומר `tx.input!=null`. הפונקציה יכולה להסתיים באחת משלוש דרכים: להדפיס "The transaction is valid"; לזרוק חריגה על "הוצאה כפולה" - DoubleSpendException ; או לזרוק חריגה על "חתימה לא חוקית" - IllegalSignatureException.

פתרון:

```
void check(Transaction tx) {
      //  **שלב א**: נבדוק אם הקלט של העיסקה הנתונה כבר נוצל בעיסקה קודמת  //.
      for (Block b=latest;  b!=null;  b=b.previous) {
           if (b.input == tx.input)
               throw DoubleSpendException;
      }
      //  **שלב ב**: נבדוק מי הבעלים של המטבע המועבר בעיסקה הנתונה, ונוודא שהוא אכן חתם על העיסקה  //.
      Key coin_owner =  tx.input.receiver;
      if (!tx.signature.is_valid(coin_owner))
            throw IllegalSignatureException;
      System.out.println("The transaction is valid");
}
```

ב [10 נק']. ציירו שרשרת בת 4 בלוקים לפחות, והדגימו עליה את פעולת הפונקציה מהסעיף הקודם. תנו שתי דוגמאות עם תוצאות שונות.

***פתרון**: נניח שהשרשרת נראית כך (מהמאוחר אל המוקדם):*

```
latest == block4;
block4 == {
     previous: block3,
     input: block3,
     receiver: 0xDDDDDDDD
}
block3 == {
     previous: block2,
     input: block1,
     receiver: 0xCCCCCCCC
}
block2 == {
     previous: block1,
     input: null,
     receiver: 0xBBBBBBBB
}
block1 == {
     previous: null,
     input: null,
     receiver: 0xAAAAAAAA
}
```

***דוגמה (1)**. מריצים את `check` על העיסקה הבאה:*

```
tx == {
      input: block1,
      receiver: 0xEEEEEEEE,
      signature: [חתימה חוקית של CCCCCCCC]
}
```

כשהבלוק b בלולאה מגיע ל-`block3`, תיזרק חריגה `DoubleSpendException`, כי ה-`input` של `block3` שווה ל-`input` של `tx` (הקלט כבר נוצל).

***דוגמה (2)**. מריצים את `check` על העיסקה הבאה:*

```
tx == {
      input: block2,
      receiver: 0xEEEEEEEE,
      signature: [חתימה חוקית של CCCCCCCC]
}
```

הלולאה תסתיים בלי חריגה כי אין אף בלוק בשרשרת שה-`input` שלו שווה ל-`block2` (הקלט לא נוצל).

אבל, החתימה היא של CCCCCCCC, בעוד שהבעלים הנוכחי של המטבע הוא BBBBBBBB (המקבל ב-block2). לכן תיזרק חריגה IllegalSignatureException.