מטלה 10 - חלוקת חפצים - פתרון

שאלה 1: אלגוריתם "המנצח המתוקן" ומכפלת הערכים

100. באלגוריתם "המנצח המתוקן", כל משתתף מייחס ערך לכל חפץ, כך שסכום הערכים של כל משתתף הוא האם תוצאת האלגוריתם תמיד ממקסמת את מכפלת הערכים? הוכיחו או הראו דוגמה נגדית.

- **תשובה**: לא. ראו דוגמה נגדית בפתרון של אריה וניסן.
- **שימו לב**: כשמדברים על "מיקסום מכפלת הערכים" מתכוונים לערכי המשתתפים ולא לערכי החפצים. למשל, אם דוגאלד קיבל שני חפצים בשווי 30 ואיוואנה קיבלה שני חפצים בשווי 40, אז הערך הכולל של דוגאלד הוא 60 ושל איוואנה 80, ולכן מכפלת הערכים היא **4800** (ולא 30*30*40*40).

שאלה 2: אלגוריתם "המנצח המתוקן" והסכמי קמפ-דייויד

המאמר בקישור זה: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1025.1941&rep=rep1&type=pdf

מנתח מה היה קורה אילו הסכמי קמפ-דייויד בין ישראל למצרים היו נעשים באלגוריתם המנצח המתוקן.

א. מיצאו במאמר את הנתונים שבהם השתמשו הכותבים - איזה נושאים עמדו למשא-ומתן? כמה אחוזים ייחס כל אחד מהצדדים לכל אחד מנושאים? מה התוצאות התיאורטיות של האלגוריתם? והאם הן דומות להסכם שהושג בפועל?

ב. מה דעתכם על המאמר - האם אכן היה אפשר במציאות להשתמש באלגוריתם "המנצח המתוקן" במקום משא-ומתן? ומה היו התוצאות?

- **פתרון**: ראו בפתרון של יובל, יצחק, משה ואביב.

שאלה 3: אלגוריתם "מעגלי הקנאה" ויעילות פארטו

הראו דוגמה שבה אלגוריתם מעגלי הקנאה מחזיר חלוקה שאינה יעילה פארטו, **בכל** סדר שבו מסדרים את החפצים, ולפי **כל** כלל בחירה בין שחקנים (כשיש שני שחקנים או יותר שאף אחד לא מקנא בהם).

- **פתרון**: [המשך יבוא]

שאלה 4: שילוב חפצים בדידים ורציפים

נניח שאנחנו רוצים לחלק אוסף של חפצים, שחלקם בדידים וחלקם רציפים.

אנחנו רוצים למצוא חלוקה יעילה-פארטו המקיימת את תנאי-ההגינות הבא (הרחבה של EF1): "לכל שני שחקנים א ו-ב, אם נסיר מהסל של שחקן ב חפץ בדיד אחד, **או** אחוז אחד מחפץ רציף, אז א לא יקנא".

הסבירו איך אפשר להשתמש באלגוריתם "מיקסום מכפלת הערכים" כדי למצוא חלוקה שהיא גם יעילה-פארטו וגם מקיימת את תנאי-ההגינות המורחב.

- **פתרון**: נחלק כל אחד מהחפצים הרציפים ל-100 חפצים בדידים, כל אחד מהם בגודל 1% מהחפץ המקורי. ואז נפעיל את אלגוריתם "מיקסום מכפלת הערכים" על אוסף החפצים שהתקבל.

שאלה 5: רמת הדיוק הדרושה להגינות

כשמבצעים במחשב אלגוריתמים עם פונקציות ממשיות, כמו לוגריתמים, החישוב לא מושלם, ושגיאות הנובעות מעיגול-מספרים עלולות לשבש את האלגוריתם ולתת תוצאה לא אופטימלית.

א. נניח שיש n שחקנים, וכל שחקן מבטא את ההעדפות שלו ע"י חלוקת 1000 נקודות בין החפצים, כאשר כל חפץ מקבל מספר שלם של נקודות (כמו כאן: http://www.spliddit.org/apps/goods). מהו ההפרש החיובי הקטן ביותר בין לוגריתמים של שתי חלוקות-חפצים שונות?

- **פתרון**: מכפלת הערכים הגדולה ביותר האפשרית היא 1000^n - זו המכפלה המתקבלת כאשר כל שחקן מקבל את מספר הנקודות המירבי שלו - 1000. כיוון שכל המספרים שלמים, המכפלה הבאה בתור היא לכל היותר 1000^n-1. לכן ההפרש הקטן ביותר האפשרי בין לוג של חלוקה אופטימלית לבין לוג של חלוקה לא-אופטימלית הוא:

- $\log(1000^n) - \log(1000^n-1) = -\log(1 - 1000^{-n}) > 1000^{-n}$.

ב. לפי תשובתכם ל-א, כמה סיביות צריכות להיות בייצוג של מספרים במחשב שלנו, כך שהאלגוריתם למיקסום מכפלת הערכים אכן ימצא את החלוקה האופטימלית ולא יושפע משגיאות של עיגול?

- מספר הסיביות צריך להיות מספיק גדול כך שהמחשב יוכל להבחין בהפרשים בגודל של 1000 בחזקת מינוס n. כלומר בערך $10n$ סיביות.

שאלה 6: תיכנות אלגוריתם

נתונה המחלקה הבאה:

```
class Agent {
     double item_value(int item_index);
}
```

המחלקה מייצגת שחקן המשתתף במשחק חלוקה הוגנת. יש בה פונקציה אחת המתארת את הערך שהשחקן מייחס לחפץ שהאינדקס שלו הוא item_index.

כיתבו פונקציה הבודקת האם חלוקת-חפצים נתונה כלשהי היא EF1. כותרת הפונקציה:

```
bool is_EF1(Agent[] agents, set<int>[] bundles);
```

הפרמטר agents הוא מערך בגודל n המייצג את השחקנים.

הפרמטר bundles הוא מערך באותו גודל - n - המייצג את החלוקה: bundle[i] הוא אוסף אינדקסי החפצים שמקבל שחקן i.

פתרון: באופן כללי, כשמנסים לפתור בעיה תיכנותית מורכבת מסוג זה, כדאי להתחיל מבעיות פשוטות יותר. כדי לדעת אם חלוקה היא *EF1*, צריך לעבור כל כל הזוגות של שחקנים, ולבדוק לגבי כל זוג, אם אחד מהם

מקנא בשני עד-כדי חפץ אחד. נתחיל מבעיה פשוטה יותר: לבדוק אם שחקן אחד מקנא בשחקן אחר. לשם כך נוסיף ל-*Agent* כמה שיטות:

```
class Agent {
     // חשב ערך של סל מסויים לשחקן הנוכחי
      double bundle_value(set<int> bundle) {
           double sum = 0;
           for (int item: bundle)   sum += this->item_value(item);
           return sum;
     }

     // החזר "אמת" אם השחקן הנוכחי לא מקנא בסל האחר:
     bool is_EF(set<int> this_bundle, set<int> other_bundle) {
           return this->bundle_value(this_bundle) >=
                  this->bundle_value(other_bundle);
     }

     // החזר "אמת" אם השחקן הנוכחי לא מקנא בסל האחר עד-כדי חפץ יחיד:
     bool is_EF1(set<int> this_bundle, set<int> other_bundle) {
           if (is_EF(this_bundle, other_bundle))
                return true;
           for (int other_item: other_bundle) {
               set<int> smaller_other_bundle = other_bundle.without(other_item);
                      // "without" is a pseudo-method that returns a set without a given item.
                      // It does not change the original set.
               if (is_EF(this_bundle, smaller_other_bundle))
                  return true;
           }
           return false;
     }
}
```

עכשיו הפונקציה הראשית כבר פשוטה למדי:

```
bool is_EF1(Agent[] agents, set<int>[] bundles) {
      for (int i=0; i<agents.size(); ++i)
         for (int j=0; j<agents.size(); ++j)
             if (! agents[i].is_EF1(bundles[i], bundles[j]))
                 return false;     // i envies j
     return true;
```

}