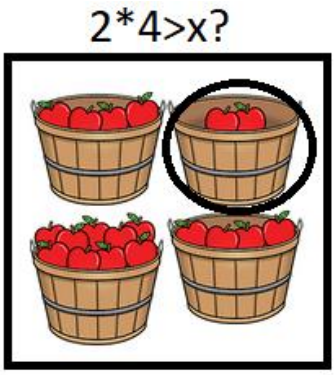
פתרון שלב ב'

1. מארזים טעימים

השאלה: לגאלה אוסף של סלים עם תפוחים, הסלים ממוספרים מ-1 עד n. בכל סל יש מספר תפוחים חיובי כלשהו. גאלה יכולה להרכיב מהסלים הללו מארזים. מארז מורכב מאוסף של סל אחד או יותר (לא בהכרח עם מספרים רצופים). מארז נחשב טעים, אם מספר התפוחים בסל עם הכי מעט תפוחים במארז זה, כפול מספר הסלים בו, גדול מ-x (x הוא מספר שלם נתון). גאלה רוצה להרכיב כמה שיותר מארזים טעימים! (וכמובן שכל סל יכול להיות לכל היותר במארז אחד)
עזרו לגאלה, וכתבו עבורה אלגוריתם, שמקבל את x (מספר שלם כלשהו), את n (מספר הסלים) ומערך בשם apples_arr באורך n כך שבכל תא במערך ימצא מספר התפוחים בסל המתאים. על האלגוריתם שלכם לחשב את מספר המארזים הטעימים הגדול ביותר שגאלה יכולה להרכיב!



אבחנות: תמיד קיים פתרון אופטימלי שבו כל המארזים העימים מורכבים מסלים רצופים מבחינת מספר התפוחים בהם (כלומר רצופים אחרי מיון). זה נובע מכך שמה שמשנה כדי שמארז יהיה טעים הוא רק הסל עם מספר התפוחים הקטן ביותר, ומספר הסלים במארז. זה מאפשר בכל מארז טעים להחליף כל סל בסל גדול יותר בלי לפגוע בטעימות שלו, ולגם להחליף כל סל שהוא לא המינימלי בסל שלא קטן מהסל המינימלי. כך ניתן להגיע מכל פתרון אחר לפתרון ששומר על תכונת הרציפות הזו בעזרת החלפות של סלים.

פתרון #1

תיאור האלגוריתם: נמיין את הסלים מהגדול לקטן ונבנה את המארזים בצורה חמדנית. נפתח מארז עם הסל הגדול ביותר ונסגור אותו כשהמארז יהיה טעים, ואז נפתח מארז חדש עם הסל הבא וכו'.

האלגוריתם עצמו:

- מיין את המארזים מהגדול לקטן
- counter → 0 // מספר המארזים הטעימים (התשובה)
- current → 1 // מספר הסלים במארז שכעת נבנה חמדנית
- עבור i מ-1 עד n:
  - אם apples_arr[i] * current > x: // אם המארז טעים
    - counter + 1 → counter // סגור את המארז ועבור לבניית המארז הבא

- 0 → current
- current + 1 → current
- הדפס את counter

נכונות: תמיד קיים פתרון שכולל את המארזים הגדולים ביותר – כי בכל פתרון אחר אפשר להחליף כל סל במארז בסל גדול יותר שלא השתמשנו בו והמארז עדיין ישאר טעים, והמארזים תמיד יכולים להיות רצופים כמו שציינו באבחנה.

פתרון #2 (מוצג רק הרעיון ולא הפתרון המלא!):

אבחנה נוספת: נניח שאנחנו יודעים על מארז כלשהו בפתרון הטוב ביותר. אם נוציא את כל הסלים שבמארז ונשים אותם בצד, בסלים שנשארו נצטרף לפתור בדיוק את אותה הבעיה – למצוא את מספר המארזים הטעימים הגדול ביותר מהם, ולכן ניתן לפתור באופן רקורסיבי (בתכנות דינאמי), אם מתחשבים גם בתכונת הרצף: ממיינים את הסלים מהקטן לגדול, ולכל סל מתייחסים ל-2 אפשרויות בתכנות הדינאמי - האפשרות שהוא לא נמצא במארז טעים, והאפשרות שהמארז שלו מורכב מרצף הסלים עד אליו.

2. תפוחונים

השאלה: האחים רד וגולדן משחקים במשחק התפוחונים.

במשחק יש m סלים עם $n_1, n_2, \ldots, n_m$ תפוחונים בכל סל בהתאמה (בסל הראשון יש n_1 תפוחונים, בסל השני יש n_2 תפוחונים וכו'). 2 השחקנים רואים כמה תפוחים יש בכל אחד מהסלים.

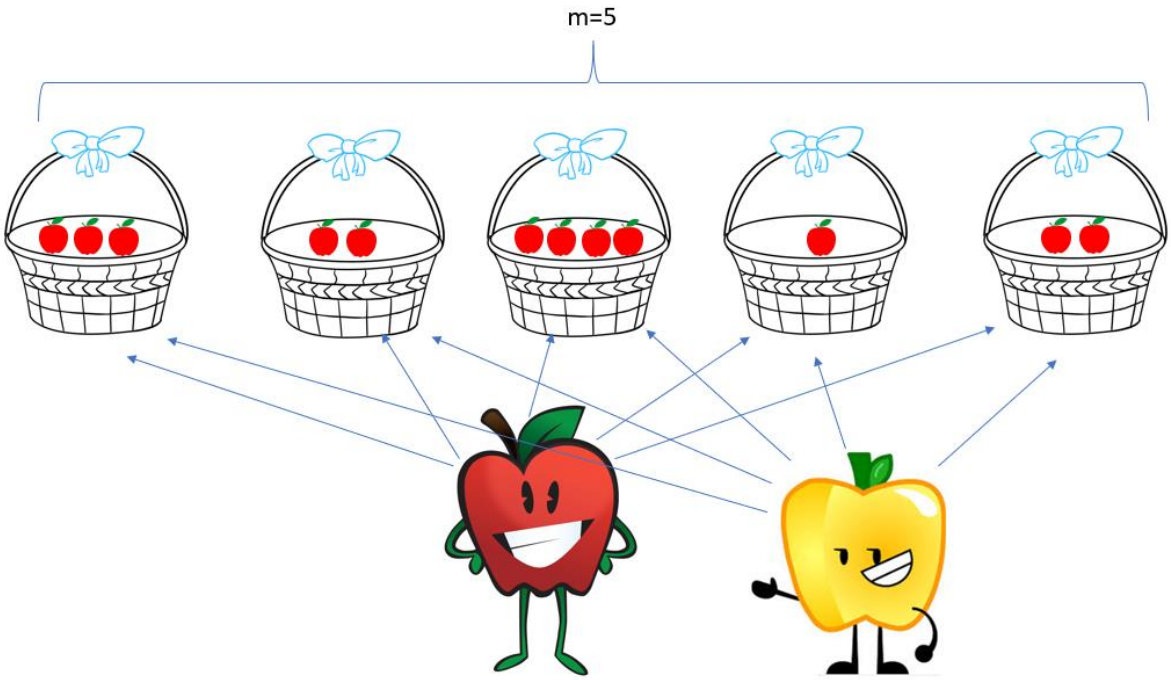
בכל תור השחקן שזהו תורו בוחר את אחד הסלים ואוכל ממנו k תפוחונים (k לא חייב להיות ערך קבוע, השחקן בוחר אותו כל תור בהתאם למגבלות בסעיפים). השחקן שבהגיע תורו אין לו מהלך אפשרי (כלומר אין באפשרותו לאכול k תפוחונים מאחד הסלים ועדיין לעמוד במגבלות על k), מפסיד במשחק. רד מתחיל במשחק.

בהנחה שרד וגולדן הם שחקנים שלא עושים טעויות ומשחקים תמיד את האסטרטגיה הטובה ביותר, עבור כל אחד מהסעיפים הבאים, כתבו אלגוריתם שמקבל את מספר התפוחונים בכל סל - $n_1, n_2, \ldots, n_m$ (מערך n_arr בגודל m), ויקבע האם רד (השחקן שהתחיל ראשון) ינצח במשחק:

א. m כללי, k חייב להיות אי זוגי (כלומר בכל תור כל שחקן יכול לקחת כל מספר אי זוגי של תפוחונים מאחד הסלים).

ב. $m=2$, ו-k חייב להיות חזקה של 2 (2^i כש-$i \geq 0$ שלם). כלומר יש 2 סלים, ובכל תור כל שחקן יכול לקחת 1, 2, 4, 8 או כל חזקה של 2 תפוחונים מאחד הסלים.

ג. m כללי, k חייב להיות חזקה של 2.



<u>פתרון</u>:

א. כשיש תפוחונים באחת הערימות תמיד לשחקן שזהו תורו יהיה תור אפשרי כי אפשר לקחת גם תפוח אחד (אחד אי זוגי). ישנה תכונה שמורה במשחק – הזוגיות של סכום התפוחונים בכל הערימות משתנה בכל תור, כי בכל תור נאכל מספק אי זוגי של תפוחונים מתוכם. לכן, אם מספר זה היה אי זוגי בתחילת המשחק, מספר התורות עד שכל התפוחונים יאכלו יהיה אי-זוגי, ורד ינצח. אחרת, מספר התורות יהיה זוגי וגולדן ינצח. לא משנה בכלל איך הם ישחקו, תמיד התכונה הזו תישמר.

אלגוריתם:

- sum ← סכום הערכים ב-n_arr
- אם sum זוגי:
  - הדפס "גולדן"
- אחרת:
  - הדפס "רד"

ב. כמו בסעיף א', ניתן לקחת תפוח אחד ולכן כל עוד יש תפוחונים באחת הערימות השחקן שזהו תורו יכול לשחק. גם כאן יש תכונה שאחד השחקנים יכול לשמור עליה, אבל כאן היא לא תמיד תישמר, וצריך לשחק חכם. התכונה היא שארית חלוקה זהה ב-3 ב-2 הערימות. השחקן שעושה את תור הניצחון בתורו לוקח את התפוחון האחרון ומשאיר 2 ערימות ריקות – אז כמובן שארית החלוקה ב-3 זהה אחרי תורו האחרון של המנצח. מכיוון שניתן לקחת בכל תור כל חזקה של 2 של תפוחונים, בפרט תפוחון אחד ו-2 תפוחונים, אם לשחקן שמגיע תורו שארית החלוקה ב-3 לא זהה ב-2 הערימות, הוא יכול להשוות אותה על ידי לקיחת תפוחון 1 או 2 (בהתאם לשוני בשארית). מצד שני, אם לשחקן שמגיע תורו שארית החלוקה ב-3 כן זהה ב-2 הערימות, הוא לא יכול להשאיר אותה זהה, כי לשם כך הוא יצטרך לא

לשנות את שארית החלוקה ב-3 בערימה שהוא לוקח ממנה תפוחונים, ואת זה ניתן לעשות רק על ידי לקיחת כפולה של 3, אבל אין חזקה של 2 שמתחלקת ב-3.
לכן, אם בהתחלה שארית החלוקה ב-3 ב-2 הערימות זהה, גולדן ינצח על ידי כך שאחרי כל תור של רד הוא יחזיר למצב הזה. אחרת, רד ינצח בצורה דומה (ישווה בתור הראשון שלו את השארית וישמור על התכונה אחרי כל תור של גולדן).

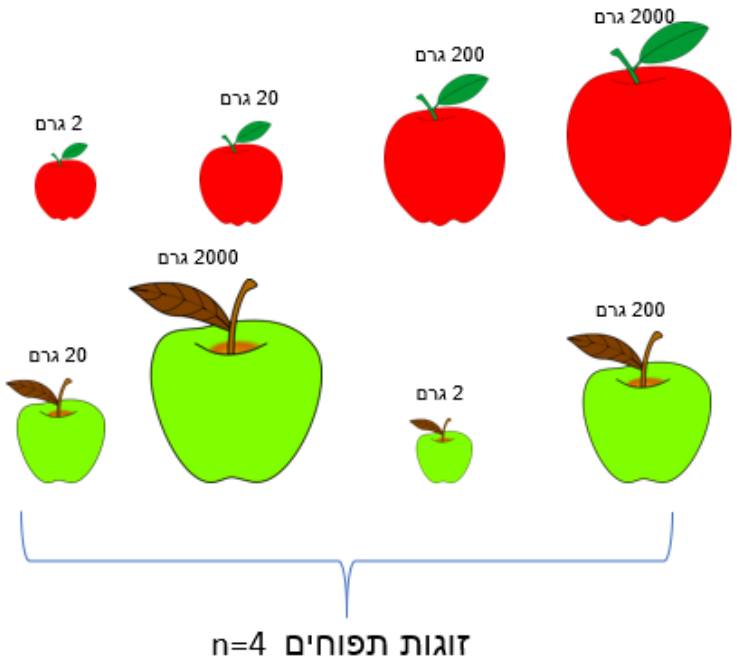
אלגוריתם:

- m1 → n_arr[1] % 3 \ מודולו \ % // שארית החלוקה ב-3, כולם יתקבלו כאן
- m2 → n_arr[2] % 3
- אם m1 שווה ל-m2:
  - הדפס "גולדן"
- אחרת:
  - הדפס "רד"

ג. בשלב זה נשאיר את הפתרון לקוראים 😊

3. החנות של גברת פינק

<u>השאלה:</u> גברת פינק מנהלת חנות תפוחים. הגיעו אליה **n** זוגות תפוחים, של תפוח אחד ירוק ותפוח אחד אדום באותו המשקל. המשקלים של הזוגות שונים זה מזה (כלומר יש **n** משקלים שונים לתפוחים בכל צבע).
גברת פינק סידרה את התפוחים האדומים בשורה, וסימנה את המקומות שלהם מ-**1** עד **n**. היא רוצה לסדר את התפוחים הירוקים בשורה מתחת לאדומים, כך שמתחת לכל תפוח אדום יהיה תפוח ירוק באותו המקום (לאו דווקא באותו המשקל). כשלקוח מגיע לחנות, עליו למסור לה את מיקומי התפוחים שהוא מעוניין לקנות, ואז גברת פינק מוכרת לו את התפוח האדום והתפוח הירוק מכל אחד מהמקומות שהוא בחר. הלקוח תמיד בוחר לפחות מקום אחד, ואף פעם לא קונה את כל התפוחים בחנות.
עזרו לגברת פינק לסדר את התפוחים הירוקים בצורה כזו, שלא משנה איזה מקומות הלקוח יבחר, המשקל הכולל (כלומר סכום המשקלים) של התפוחים הירוקים שהוא יקבל יהיה שונה מהמשקל הכולל של התפוחים האדומים שהוא יקבל.
כתבו אלגוריתם, שמקבל את המערך **red_arr** - מערך באורך **n** שמכיל את המשקל של התפוח האדם בכל מקום בשורה שגברת פינק סידרה, ומדפיס את סידור מתאים של התפוחים הירוקים לפי הסדר (או בונה את המערך **green_arr**).
לדוגמה, אם המערך **red_arr** הוא:
2 20 200 2000
סידור חוקי של התפוחים הירוקים יהיה:
20 2000 2 200



<u>אבחנות</u>: באופן ישיר מהשאלה, ברור שמתחת לתפוח אדום לא יכול להיות תפוח ירוק באותו המשקל, וגם לא יכול להיות תפוח אדום **x** שמתחתיו נמצא תפוח ירוק **y** כשמתחת ל-**y** נמצא תפוח אדום **x**. אבחנה כללית יותר היא שבכל קבוצה של מקומות שלא כוללת את כל התפוחים בחנות, התפוחים הירוקים לא יהיו באותם המשקלים בדיוק כמו התפוחים האדומים.

*שימו לב! פתרון שגוי נפוץ מאוד היה כזה שמממש את האבחנות האלו בלבד, למשל על ידי "שיפט" – לשים מתחת לתפוח האדום ה-**i** את התפוח הירוק ה-**i+1**, ומתחת לאדום האחרון את הירוק הראשון, או הפוך.*
זהו פתרון שגוי! יכול להיות מצב שבו התפוחים הירוקים במשקלים שונים משל האדומים, אבל סכומם זהה.
למשל:
15 20 10 5
5 15 20 10
מקומות 1 ל-3 מכילים אמנם תפוחים שונים (5, 20 אדומים ו-10, 15 ירוקים), אבל סכומים זהה (25).

<u>תיאור האלגוריתם הנכון</u>: מתחת לתפוח האדום הכבד ביותר נשים את התפוח הירוק הקל ביותר, מתחת לכל תפוח אדום אחר נשים את התפוח הירוק הקל ביותר שכבד ממנו (התפוח הבא בגודלו אם ממיינים את המערך).

<u>נכונות:</u> אם הלקוח לא בוחר את המקום עם התפוח האדום הכבד ביותר – בכל מקום שבחר התפוח הירוק יהיה כבד מהתפוח האדום, ולכן המשקל הכולל של התפוחים הירוקים יהיה כבד יותר מהמשקל הכולל של התפוחים האדומים. אם הלקוח כן בוחר את המקום עם התפוח האדום הכבד ביותר – נסתכל על כל התפוחים שהלקוח **לא** בחר. כמו במקרה הקודם, משקלם של הירוקים יהיה כבד יותר. מכיוון שהמשקל הכולל של כל התפוחים הירוקים והאדומים בחנות זהה, המשקל הכולל של התפוחים הירוקים שהלקוח **כן** בחר יהיה נמוך יותר מהמשקל הכולל של האדומים שבחר.

<u>אלגוריתם:</u>

- הוסף לכל ערך במערך **red_arr** את המיקום שלו
- מיין את הזוגות (ערך, מיקום) במערך **red_arr** בסדר עולה לפי משקל התפוחים
- לכל i מ-1 עד n:
  - red_arr[(red_arr[i].position + 1) % n].weight → green_arr[i]

4. מדף התפוחים

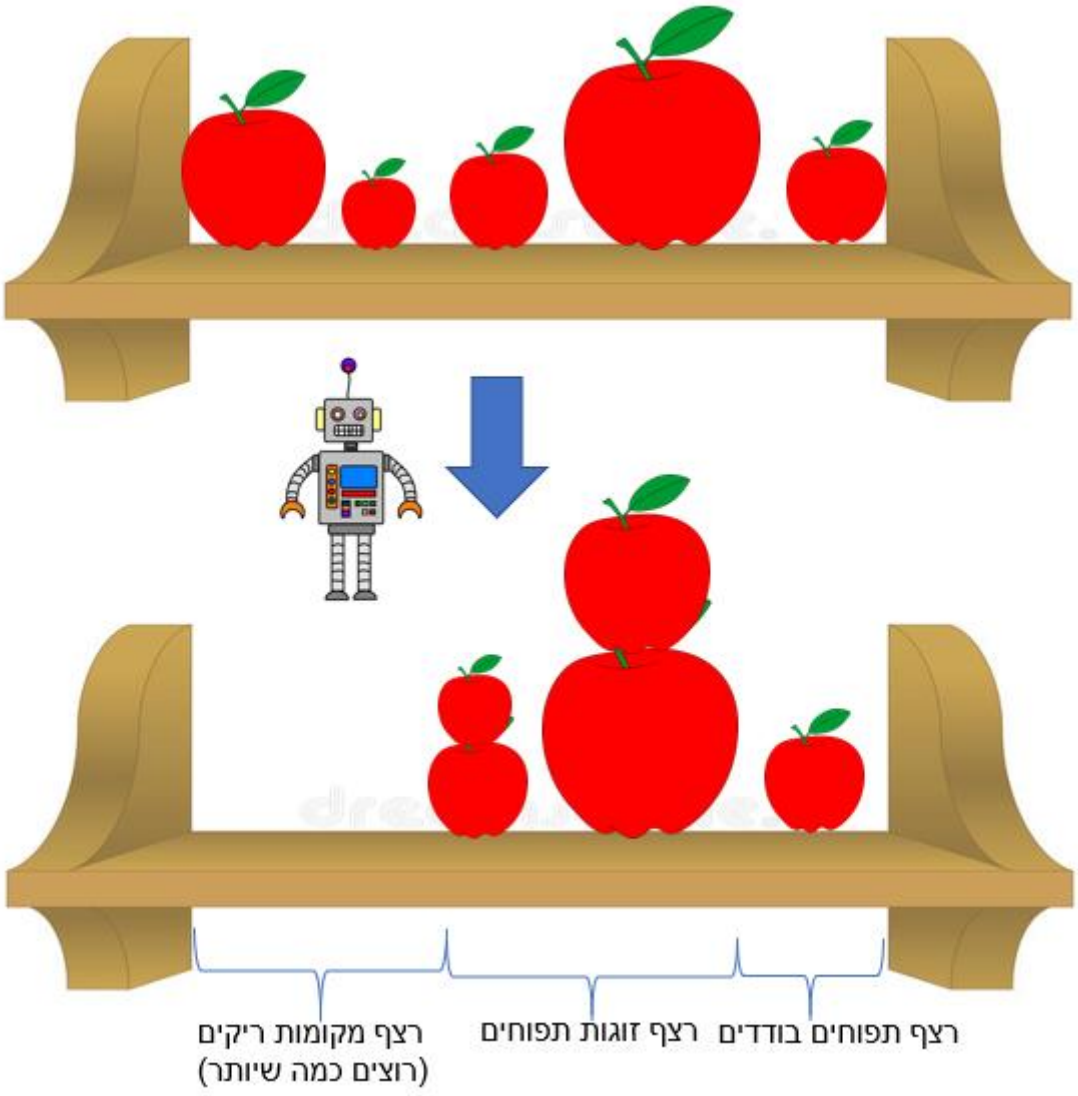
<u>השאלה:</u> לסבתא סמית מדף עם **n** תפוחים. על המדף יש **n** מקומות הממוספרים מ-1 (המקום השמאלי ביותר) עד **n** (המקום הימני ביותר). כל אחד מהתפוחים של סבתא סמית בגודל שלם כלשהו בין 1 ל-**m** (יכולים להיות מספר תפוחים באותו הגודל). בתקופת הסגר סבתא סמית פיתחה תחביב חדש של סידור תפוחים על המדף. תחילה, היא שמה בדיוק תפוח אחד בכל אחד מהמקומות. לאחר מכן, היא הזמינה באינטרנט רובוט מיוחד שמסדר תפוחים.

לרוע מזלה, הרובוט שהגיע לא מוצלח במיוחד. הוא יודע רק לקחת תפוח כלשהו שנמצא לבד במקומו, ולשים אותו מעל תפוח כלשהו שגדול ממנו ושנמצא מימינו. הוא לא יכול להזיז תפוח שמאלה, ולא יכול לשים 3 תפוחים ומעלה אחד על השני, רק עד 2. הוא גם לא יכול להניח תפוח במקום ריק, הפעולה היחידה שהוא יכול לעשות היא הפעולה המתוארת. אחלה רובוט.

סבתא סמית רוצה לסדר את התפוחים בסידור יפה. סידור יפה הוא סידור בו בצד הכי שמאלי של המדף יש כמה שיותר מקום פנוי, אחר כך יש זוגות של תפוחים זה על זה ברצף, ואז יש תפוחים בודדים ברצף. במילים אחרות, היא רוצה לשים את **K** התפוחים השמאליים ביותר על **K** התפוחים שבדיוק מימינם בסדר כלשהו, כש-**K** גדול ככל האפשר. סבתא סמית מתעקשת לא לגעת בתפוחים שלא בעזרת הרובוט. עזרו לסבתא סמית, וכתבו עבורה אלגוריתם יעיל שמקבל את n (מספר התפוחים), את m (הגודל המירבי של תפוח), ואת גדלי התפוחים על המדף לפי המיקומים שלהם (במערך **apple_arr**), ומדפיס את **K** הנ"ל, בהנחה ש (לכל סעיף יכול להתאים אלגוריתם אחר):

א. **m** גדול משמעותית מ-**n**

ב. **n** גדול משמעותית מ-**m**



בשלב זה לא נספק פתרון לשאלה זו.