חלוקה מיטבית של חפצים בדידים

בשיעור הקודם ראינו אלגוריתמים למציאת חלוקה של חפצים בדידים שהיא "כמעט" הוגנת. אכן, במקרה הגרוע אי־אפשר להבטיח יותר מזה, למשל כשיש מספר איזוגי של חפצים זהים. אבל במקרים רבים אפשר להשיג חלוקות טובות יותר. השאיפה שלנו בפרק זה תהיה למצוא את החלוקה הטובה ביותר האפשרית, בהתחשב בנתונים.

בסעיף הבא נעסוק ב**בעיית החלטה**: בהינתן חפצים ושחקנים, האם קיימת חלוקה הוגנת לגמרי (ולא רק בקירוב)?

בסעיפים שאחריו נעסוק ב**בעיית מיטוב** (אופטימיזציה, optimization): מציאת חלוקה שבה פונקציה מסויימת מקבלת ערך גדול ביותר, מבין כל החלוקות האפשריות. נתמקד במיטוב **הערך האגליטרי** (egalitarian) – הערך הקטן ביותר שמקבל שחקן בחלוקה.

בדיקה האם קיימת חלוקה הוגנת

בעיית חלוקת־המספרים (באנגלית: Partition או Number Partitioning) היא בעייה שאפשר לראות בה מקרה פרטי של בעיית החלוקה ההוגנת: יש רק שני שחקנים, יש להם **הערכות זהות**, וסכום הערכים של כל החפצים הוא 2V, כאשר V הוא מספר שלם כלשהו. במצב זה, חלוקה היא הוגנת (פרופורציונלית וללא־קנאה) אם־ורק־אם כל שחקן מקבל ערך V בדיוק. המטרה היא לבדוק האם קיימת חלוקה הוגנת. ההגדרה המקובלת לבעיית חלוקת־המספרים היא פשוטה יותר, ואינה מתייחסת כלל לשחקנים או להוגנות אלא למספרים בלבד.

הגדרה: **בעיית חלוקת־המספרים** היא הבעיה הבאה:

- הקלט: רשימה של מספרים שלמים חיוביים שסכומם 2V.
- הפלט: "כן" אם קיימת חלוקה של המספרים לשתי קבוצות שסכומן V; אחרת "לא".
-

בעיית חלוקת־המספרים היא בעייה קשה חישובית: כבר בשנת 1972, הוכיח ריצ'ארד קארפ שהבעיה הזאת היא NP־קשה, ולכן – לפי ההנחה המקובלת על רוב מדעני המחשב בימינו – לא קיים לה פתרון בזמן פולינומיאלי. מכאן, שגם בעיית החלוקה ההוגנת – שהיא בעיה כללית יותר – היא NP־קשה.

משפט: הבעיה הבאה – מציאת חלוקה ללא־קנאה – היא NP־קשה:

- הקלט: קבוצה של n שחקנים עם הערכות חיבוריות על אוסף חפצים בדידים;
- הפלט: "כן" אם קיימת חלוקה ללא־קנאה; אחרת "לא".

גם בעיית מציאת חלוקה פרופורציונלית, המוגדרת באופן אנלוגי, היא NP־קשה.

קושי חישובי אינו סיבה לייאוש. ישנן כמה דרכים המאפשרות לפתור את הבעיה בזמן סביר, לפחות במקרים קטנים יחסית.

1. תיכנות ליניארי בשלמים

מבחינה מעשית, הדרך הפשוטה ביותר לפתרון בעיית החלוקה היא **תיכנות ליניארי בשלמים** (integer linear programming). התוכניות שנכתוב דומות לתוכניות שראינו בפרק 3 לפתרון בעיות חלוקת משאבים רציפים. ההבדל העיקרי הוא, שבחלוקת חפצים בדידים, יש להוסיף דרישה שהמשתנים הם מספרים שלמים.

הקלט הוא מטריצה v. לכל שחקן i ולכל חפץ g, הערך $v_{i,g}$ במטריצה מציין את הערך של חפץ g בעיני שחקן i.

המשתנים: לכל שחקן i ולכל חפץ g, נגדיר משתנה $x_{i,g}$ המציין את האחוז של חפץ g הניתן לשחקן i.

כדי לבדוק אם קיימת חלוקה פרופורציונלית, נפתור את התוכנית הליניארית הבאה:

$$\text{sum}_i\ x_{i,g} = 1 \qquad \text{for every item } g$$

$0 \le x_{i,g} \le 1$ for all i, g

$x_{i,g}$ is an integer for all i, g

$v_i(x_i) \ge v_i(all)/n$ for every player i

שימו לב: בתוכנית זו אין פונקציית־מטרה – יש רק אילוצים. ישנן ארבע קבוצות של אילוצים:

- הקבוצה הראשונה קובעת, שכל חפץ מחולק פעם אחת בדיוק.
- הקבוצה השניה קובעת, שאחוזי-הבעלות על חפץ כלשהו חייבים להיות בין 0 ל־1.
- הקבוצה השלישית קובעת, שאחוזי-הבעלות על חפץ כלשהו חייבים להיות מספר שלם. אלה האילוצים המייחדים את בעיית חלוקת החפצים הבדידים. יחד עם האילוצים מהקבוצה השניה, נובע שהמשתנים חייבים להיות 0 או 1, כלומר, כל חפץ נמסר בשלמותו לשחקן אחד.
- הקבוצה הרביעית קובעת, שהחלוקה היא פרופורציונלית – סכום הערכים שמקבל כל שחקן הוא לפחות n/1 מסכום הערכים הכללי שלו. בשורה זו השתמשנו בסימון $v_i(x_i)$ כקיצור לסכום: sum_g $v_{i,g}*x_{i,g}$, המייצג את הערך של שחקן i בחלוקה x. זו פונקציה ליניארית של המשתנים $x_{i,g}$, ולכן ניתן להשתמש בו בתוכנית ליניארית.

אפשר לפתור תוכנית כזאת בעזרת חבילות־תוכנה לפתרון בעיות־מיטוב ליניאריות, המאפשרות להגדיר משתנה כמספר שלם (integer). חבילות כאלו יודעות לקבל אוסף של אילוצים, ולהחליט האם קיים פתרון כלשהו העומד באילוצים. אם התשובה היא "כן", אז החבילות גם מחזירות את הפתרון. במקרה שלנו, הפתרון הוא המשתנים $x_{i,g}$. בעזרת המשתנים הללו ניתן לחשב את החלוקה הפרופורציונלית, אם היא קיימת. כיוון שהבעיה NP־קשה, במקרה הגרוע הפתרון עשוי לקחת זמן מעריכי בגודל הקלט. אבל כאשר הבעיה מספיק קטנה, הפתרון בדרך־כלל מהיר.

2. חיפוש במרחב המצבים

שיטה נוספת, המאפשרת לפתור בעיות קטנות בזמן סביר, היא **חיפוש במרחב המצבים (state-space search)**. בשיטה זו, האלגוריתם עובר על מרחב החלוקות האפשריות, אבל מסנן חלקים ממרחב זה, שבוודאות לא יובילו לפתרון מיטבי.

האלגוריתם מחזיק קבוצה של **מצבים (states)**. כל מצב הוא וקטור בגודל n+1, המייצג הקצאה חלקית – הקצאה של חלק מהחפצים. לכל i בין 1 ל־n, איבר i בווקטור מציין את הערך של הסל הנוכחי של שחקן i, בעיני שחקן i. האיבר האחרון בווקטור מציין את מספר החפצים שחולקו: אם כתוב שם t, המשמעות היא שהווקטור מייצג חלוקה של חפצים 1 עד t.

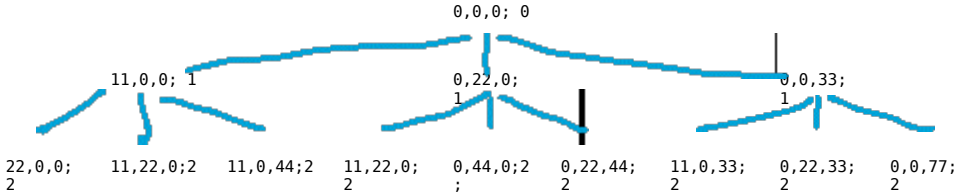
המצב ההתחלתי הוא וקטור שכולו אפסים, המייצג את החלוקה הריקה: ערך הסל של כל שחקן הוא אפס, וחולקו אפס חפצים. בהינתן מצב כלשהו המייצג חלוקה של חפצים 1,...,t, ושחקן כלשהו i, אפשר ליצור מצב חדש המשקף את נתינת החפץ ה־t+1 לשחקן i, על־ידי הוספת הערך $v_i(t+1)$ לאיבר i בווקטור, והוספת 1 לאיבר n+1 בווקטור.

ניתן לסדר את כל וקטורי־המצב האפשריים במבנה של **עץ**. **שורש** העץ הוא המצב ההתחלתי (וקטור האפסים); לשורש ישנם n ילדים המייצגים את כל n האפשרויות לתת את חפץ מספר 1; ובאופן כללי, לכל קודקוד פנימי במרחק t מהעץ ישנם n ילדים המייצגים את כל n האפשרויות לתת את חפץ מספר t+1.

דוגמה. נניח שיש שלושה שחקנים. החפץ הראשון שווה 11 לשחקן 1, 22 לשחקן 2, ו־33 לשחקן 3. אז המצב הראשון הוא [0,0,0; 0], ואפשר ליצור ממנו שלושה מצבים חדשים המייצגים את שלוש האפשרויות לנתינת החפץ הראשון:

[11,0,0; 1]; [0,22,0; 1]; [0,0,33; 1].

נניח שהחפץ השני שווה 11 לשחקן 1, 22 לשחקן 2, ו־44 לשחקן 3. אז שתי הרמות הראשונות של עץ החלוקות ייראו כך:



כל **עלֶה** בעץ מייצג חלוקה שלמה – חלוקה של כל m החפצים. איך יודעים אם עלה מסויים מייצג **חלוקה פרופורציונלית**? – בודקים אם לכל שחקן i, הערך ה־i בווקטור הוא לפחות הערך המגיע לשחקן i בחלוקה פרופורציונלית, שהוא 1 חלקי n מהערך שהשחקן מייחס לכל החפצים.

במחשבה ראשונה נראה, שכדי למצוא חלוקה פרופורציונלית, צריך לבנות את כל עץ־החלוקות.

זהו עץ מאד גדול: מספר העלים בעץ הוא n בחזקת m. אולם אפשר להקטין את מרחב־החיפוש על־ידי תהליך שנקרא **גיזום** (באנגלית pruning) – מחיקת מצבים וענפים מיותרים מעץ־החלוקות. הנה שני כללים פשוטים שאפשר להשתמש בהם כדי לגזום את עץ החלוקות.

כלל א: מצבים זהים. אם מתגלים שני וקטורי־מצב זהים ברמה כלשהי בעץ, אפשר לגזום אחד מהם. בדוגמה, ברמה השניה בעץ ישנם שני וקטורי־מצב זהים – השני משמאל והרביעי משמאל, ולכן האלגוריתם יגזום אחד מהם ולא ימשיך לבנות את תת־העץ שמתחתיו.

כלל ב: חסמים אופטימיים. לכל קודקוד פנימי בעץ, נחשב "חלוקה אופטימית" – חלוקה טובה ביותר שאפשר להגיע אליה מקודקוד זה. החלוקה לא חייבת להיות מציאותית – היא יכולה להיות אופטימית יותר מדי. לדוגמה, אפשר לחשב את החלוקה שתתקבל אם נשכפל את כל החפצים שנשארו, וניתן אותם לכל אחד ואחד מהשחקנים. ניתן לחשב את החלוקה הזאת בקלות, ע"י הוספת ערכי כל החפצים שנשארו לכל אחד מהאיברים בוקטור. אם החלוקה האופטימית הזאת אינה פרופורציונלית, אז אפשר לגזום את כל הענף, כי אף חלוקה שתתקבל בענף זה לא תהיה טובה יותר מהחלוקה האופטימית. בהמשך לדוגמה, נניח שאחרי הרמה השניה נשארו עוד שמונה חפצים, שכולם שווים 1 עבור שחקן 1. מכאן שסכום הערכים של שחקן 1 הוא 8+11+11=30, וכדי שהחלוקה תהיה פרופורציונלית, הוא צריך לקבל סל שערכו לפחות 10. לכן ניתן לגזום את הענף החמישי משמאל (0, 44, 0; 2). בענף זה, אפילו בחלוקה האופטימית, שחקן 1

מקבל רק 8, והחלוקה אינה פרופורציונלית. באותו אופן ניתן לגזום את הענף השישי, השמיני והתשיעי משמאל. על־פי ערכי החפצים הנותרים עבור שחקנים 2 ו־3, ייתכן שאפשר לגזום ענפים נוספים.[1]

במקרים מסויימים, ניתן להשתמש בכללי גיזום נוספים. לדוגמה, כאשר ההערכות של כל השחקנים **זהות**, ישנו כלל גיזום המאפשר להקטין את מרחב החיפוש פי n עצרת לפחות (נשאיר לכם כתרגיל, להגדיר כלל-גיזום זה.)

באיזה סדר נבנה את הקודקודים בעץ החלוקות? האפשרות הפשוטה ביותר היא לבנות **לרוחב**, כלומר, לבנות קודם את כל הקודקודים ברמה 1, ואז את כל הקודקודים ברמה 2, ואז את כל הקודקודים ברמה 3, וכו'. בשיטה זו, אנחנו שומרים בכל שלב t רק את קבוצת המצבים של רמה t.

אפשרות שניה היא לבנות **לעומק**, כלומר, לבנות קודם את כל המסלול מהשורש ועד לעלה השמאלי ביותר, ואז לחזור אחורה ולבנות את העלה שמימינו, וכו'. ניתן לממש שיטה זו בעזרת מחסנית, כפי שלמדתם (או הייתם אמורים ללמוד) בקורס אלגוריתמים. היתרון של בניה לעומק הוא, שאנחנו מקבלים חלוקות שלמות באופן מיידי. אם התמזל מזלנו, ואחת החלוקות הראשונות שמצאנו היא חלוקה פרופורציונלית, אנחנו יכולים להפסיק לחפש מייד, ואיננו צריכים לבנות את שאר העץ. יתרון נוסף של בנייה לעומק הוא חיסכון בצריכת הזיכרון: בבנייה לעומק, אין צורך לשמור את כל המצבים ברמה הנוכחית, אלא מספיק לשמור את המצבים שבמסלול מהשורש עד המיקום הנוכחי. [2]

כדי שהאלגוריתם יוכל להחזיר את החלוקה הפרופורציונלית במקרה שהיא קיימת, הוא צריך לשמור מידע שיאפשר לו לחזור מהסוף להתחלה: בכל פעם שנוצר מצב חדש בשלב t, יש לשמור את המצב הקודם לו (שממנו הוא נוצר) בשלב t-1. בסוף האלגוריתם, אם קיים מצב המתאים לחלוקה פרופורציונלית, אפשר לחזור אחורה למצב הקודם ולזהות למי ניתן החפץ ה־m; מכאן אפשר לחזור אחורה למצב שלפניו, ולזהות למי ניתן החפץ ה־m—1; וכו'.

כעת נחשב את **סיבוכיות זמן הריצה** של האלגוריתם. זמן הריצה של סריקת עץ הוא ליניארי במספר הקודקודים בעץ. ניתן לחשב שני חסמים עליונים על מספר הקודקודים בעץ החלוקות:

- מספר הקודקודים בכל רמה t הוא לכל היותר n^t. מספר הקודקודים הכולל הוא סכום של n^t כאשר t בין 0 ל־n. זה סכום של טור הנדסי, שהוא $O(n^m)$.

---

1 בספרות, במקום המושג "חסם אופטימי", משתמשים במושג **חסם תחתון (lower bound)** עבור בעיית מינימום, ו**חסם עליון (upper bound)** עבור בעיית מקסימום. אנחנו מעדיפים את המושג **חסם אופטימי**, המתאים לשני סוגי הבעיות.

2 **הערה לגבי מינוחים**. בספרות ישנם שני מושגים שמשתמשים בהם כדי לתאר אלגוריתמים מסויימים של חיפוש במרחב המצבים.

* אלגוריתם **סיעוף וחסימה (branch and bound)** הוא אלגוריתם המבצע חיפוש **לעומק**, וגוזם לפי חסם אופטימי (כלל ב) בלבד.

* אלגוריתם **תיכנות דינאמי (dynamic programming)** הוא אלגוריתם המבצע חיפוש **לרוחב**, גוזם רק מצבים זהים (כלל א), ומשתמש במערך בינארי כדי לזכור את המצבים שנראו עד כה. [18, 15]

בספר זה בחרנו לתאר אלגוריתם כללי – חיפוש במרחב המצבים – שבו גם כללי־הגיזום וגם סדר־החיפוש ניתנים לבחירה. הדבר מאפשר להשתמש בשני סוגי כללי־הגיזום בו־זמנית עם כל אחד מהסדרים.

- נסמן באות V את המספר הכולל של ערכים אפשריים עבור סל כלשהו לשחקן כלשהו. מספר המצבים בכל רמה t, לאחר גיזום מצבים זהים (לפי כלל א), הוא לכל היותר V^n. יש m רמות, ולכן מספר המצבים הכולל הוא לכל היותר $m*V^n$.

החסם העליון הראשון הוא מעריכי במספר החפצים. החסם העליון השני ליניארי במספר החפצים ומעריכי במספר השחקנים. בבעיות חלוקה טיפוסיות, מספר השחקנים קטן יחסית, אבל מספר החפצים עשוי להיות גדול. לדוגמה, בבעיית חלוקת ירושה, עשויים להיות שלושה או ארבעה אחים, אבל כמה עשרות חפצים. במצב זה, החסם העליון הראשון אינו מועיל. החסם העליון השני עשוי להיות מועיל - זה תלוי בגודל של V – מספר הערכים האפשריים של כל שחקן. לשם המחשה, נתבונן באתר ספלידיט[3] – אתר אינטרנט לחלוקה הוגנת. אחד היישומים באתר הוא חלוקת חפצים בדידים. האתר מבקש מכל שחקן להגיד איזה ערך הוא מייחס לכל חפץ. הערכים חייבים להיות מספרים שלמים, וסכום הערכים חייב להיות 1000. במקרה זה, $V=1000$, וכבר כשיש שלושה שחקנים, מספר המצבים יכול להגיע לעשרות מיליארדים. כדי לייעל את האלגוריתם, אפשר לתת לכל שחקן רק 100 נקודות לחלוקה. לחלופין, אפשר לתת לכל שחקן לדרג את כל החפצים ב"כוכבים" – מכוכב אחד (הכי פחות טוב) לחמישה כוכבים (הכי טוב). במקרה זה, מספר הערכים האפשריים לכל שחקן הוא $5m$, ומספר המצבים הוא $O(m^{n+1})$ – פולינומיאלי במספר החפצים, והרבה יותר טוב מ־ $O(n^m)$.

שימו לב: בחישוב סיבוכיות זמן הריצה לא התייחסנו לגיזום לפי חסם אופטימי (כלל ב). כלל זה אינו משפיע על סיבוכיות זמן הריצה, כי במקרה הגרוע ביותר ייתכן שהחסם האופטימי נותן תמיד חלוקה פרופורציונלית. עם זאת, בניסויים עם קלטים אקראיים, מתברר שדווקא כלל־הגיזום הזה מועיל ומשפר את זמן הריצה הרבה יותר מאשר גיזום מצבים זהים. מעבר לכך: גיזום מצבים זהים דורש לשמור בזיכרון את כל המצבים שראינו. כשמספר המצבים גדול, הזיכרון הנדרש לשם כך עלול להיות גדול מהזיכרון שברשותנו. במקרים אלה, גיזום מצבים זהים אינו מעשי, אבל גיזום לפי חסם אופטימי עדיין אפשרי ומועיל.

3 האתר http://www.spliddit.org/apps/goods

חישוב חלוקה אגליטרית

חלוקה אגליטרית היא חלוקה הממקסמת את הערך הנמוך ביותר של שחקן. בפרק 3 הוכחנו, שכאשר ההערכות **מנורמלות** (כל שחקן מייחס את אותו ערך לכל המשאבים), וקיימת חלוקה פרופורציונלית, אז כל חלוקה היא אגליטרית. לפי זה, החלוקה האגליטרית מבטאת שאיפה להתקרב ככל האפשר לעקרון הפרופורציונליות: אם קיימת חלוקה פרופורציונלית – חלוקה שבה כל שחקן מקבל n/1 מהשווי הכללי – אז נחזיר חלוקה פרופורציונלית; אחרת, נמצא ערך כלשהו x, שהוא קטן מ־n/1, ונחזיר חלוקה שבה כל שחקן מקבל לפחות x מהשווי הכללי; נבחר את ה־x הגדול ביותר האפשרי במסגרת אילוצי הבעיה.

בפרק 3, כשחילקנו משאבים רציפים, יכולנו לחשב חלוקה אגליטרית במהירות ע"י פתרון בעיית־מיטוב קמורה. כשהחפצים בדידים, הבעיה – כצפוי – קשה יותר: ניתן להוכיח שהבעיה היא NP-קשה (ראו במטלה). ניתן לפתור את הבעיה ע"י תיכנות ליניארי בשלמים או חיפוש במרחב המצבים, אבל זמן הריצה של האלגוריתמים הללו הוא מעריכי במקרה הגרוע.

לכן נציג דרך אחרת להתמודד עם בעיות קשות: אלגוריתמי קירוב.

אלגוריתם **קירוב גורם־קבוע** (constant-factor approximation) לבעיית מיטוב הוא אלגוריתם המחזיר, לכל קלט של הבעיה, פתרון שערכו לפחות r כפול הפתרון המיטבי, כאשר r הוא מספר ממשי קבוע (שאינו תלוי בקלט).

בסעיף זה נראה אלגוריתם קירוב גורם־קבוע עבור מקרה פרטי שבו לכל השחקנים יש אותן הערכות – הם מסכימים על ערכי החפצים. זה מצב נפוץ, למשל, כאשר הערכים הם מחירי־שוק, והמטרה היא למצוא חלוקה אגליטרית על־פי מחירי השוק.

אלגוריתמי-הקירוב הראשונים לבעיה זו פותחו עבור בעיה הנדסית, שבמבט ראשון אין כל קשר בינה לבין הוגנות: בעיית **תיזמון עבודות** (באנגלית: job scheduling. נקראת גם multiprocessor scheduling; או machine scheduling). יש לנו כמה מחשבים זהים (או מחשב עם כמה מעבדים זהים); אנחנו רוצים לבצע משימה חישובית מסויימת, המורכבת ממספר רב של עבודות-חישוב היכולות לרוץ במקביל. אנחנו יודעים מראש כמה זמן דרוש לכל מחשב לבצע כל עבודה. זמן־הסיום של המשימה כולה (באנגלית: **makespan**) הוא זמן־הסיום של העבודה האחרונה. השאלה היא, איך לחלק את העבודות בין המחשבים, כך שזמן־הסיום הכולל יהיה הקצר ביותר האפשרי?

דוגמה. יש ארבעה מחשבים ותשע עבודות. אורכי העבודות בשניות הם: [4]

4, 4, 4, 5, 5, 6, 6, 7, 7.

נבדוק שתי חלוקות אפשריות של עבודות למחשבים:

- חלוקה א: המחשב הראשון מבצע 5+6, השני מבצע 5+6, השלישי מבצע 4+7, הרביעי מבצע 4+4+7. שלושה מחשבים מסיימים תוך 11 שניות, אבל המחשב הרביעי מסיים תוך 15 שניות, ולכן זמן־הסיום של המשימה כולה (makespan) הוא 15.
- חלוקה ב: מחשב ראשון מבצע 6+6, השני מבצע 5+7, השלישי מבצע 5+7, הרביעי מבצע 4+4+4. כל המחשבים מסיימים תוך 12 שניות, ולכן זמן־הסיום הכולל הוא 12. זוהי החלוקה היעילה ביותר בדוגמה זו: כיוון שסכום אורכי העבודות הוא 48, בכל חלוקה קיים מחשב כלשהו שזמן־הסיום שלו הוא לפחות 12.

---

4 על־פי טל גרינשפון, "אלגוריתמי זימון ושיבוץ", קורס במחלקה להנדסת תעשיה וניהול באריאל.

הסיבה הראשונה לעיסוק בבעיה זו היתה, כאמור, למצוא חלוקת־עבודה יעילה ומהירה. אך למעשה, בעיה זו שקולה לחלוטין לבעייה של חלוקה אגליטרית של חפצים עם **ערך שלילי** בין שחקנים עם **הערכות זהות**. דוגמה לחפצים בעלי ערך שלילי היא **מטלות**, כגון תורנויות בבית או בבסיס צבאי. רוב האנשים אינם אוהבים לבצע מטלות, אבל הערך (השלילי) שהם מייחסים לכל מטלה עשוי להיות שונה.

דוגמה. יש ארבעה שחקנים הצריכים לבצע ביחד תשע מטלות. הם מייחסים למטלות ערכים שליליים (כולם מסכימים על הערכים של כל מטלה).

-4, -4, -4, -5, -5, -6, -6, -7, -7.

אנחנו רוצים למצוא חלוקה אגליטרית. נבדוק שתי חלוקות:

- חלוקה א: שחקן אחד מקבל שלוש מטלות עם ערכים -4, -4, -7; שחקן שני מקבל שתי מטלות עם ערכים -4, -7; ועוד שני שחקנים מקבלים כל אחד שתי מטלות עם ערכים -6, -5. ערכי השחקנים הם: -11, -11, -11, -15, והערך האגליטרי הוא -15.
- חלוקה ב: שחקן אחד מקבל שלוש מטלות עם ערכים -4, -4, -4; שחקן שני מקבל שתי מטלות עם ערכים -6, -6; ועוד שני שחקנים מקבלים כל אחד שתי מטלות עם ערכים -7, -5. ערכי השחקנים הם: -12, -12, -12, -12, והערך האגליטרי הוא -12.

הערך האגליטרי בחלוקה ב גדול יותר מבחלוקה א. למעשה, חלוקה ב היא החלוקה האגליטרית במקרה זה: כיוון שסכום ערכי המטלות הוא -48, בכל חלוקה יש לפחות שחקן אחד שערכו לכל היותר -12.

קל לראות, שהדבר נכון באופן כללי: כל חלוקת עבודות למחשבים, הממזערת את זמן־הריצה המקסימלי, שקולה לחלוקת מטלות לאנשים, הממקסמת את הערך (השלילי) המינימלי – חלוקה אגליטרית. הבעיה היא NP־קשה – ההוכחה זהה להוכחה עבור חפצים עם ערכים חיוביים.

כעת נראה אלגוריתם חמדני פשוט למציאת חלוקה אגליטרית־בקירוב. לצורך הפשטות, כדי שנוכל לעבוד עם מספרים חיוביים, נגדיר את **העלות** (cost) של כל מטלה כערך של המטלה **בערך מוחלט**, והמטרה היא למצוא חלוקה שבה העלות **המקסימלית** היא **קטנה ביותר**. בעולם תיזמון העבודות, האלגוריתם הזה נקרא "תיזמון רשימה" (List Scheduling) כי הוא מתזמן את העבודות לפי רשימה קבועה מראש. אנחנו משתמשים באלגוריתם לא לצורך תיזמון אלא לצורך חלוקת מטלות, ולכן נקרא לו "אלגוריתם הרשימה".

- 1. עבור על כל המטלות לפי הסדר שבו הן נמצאות ברשימה.
- 2. תן את המטלה הבאה ברשימה לְשחקן, שסכום העלויות הנוכחי שלו קטן ביותר.
- 3. אם יש עוד מטלות, חזור לשורה 2.

בדוגמה למעלה, המטלה הראשונה בסדרה היא ה־4 והאחרונה היא ה־7. האלגוריתם נותן את ארבעת המטלות הראשונות לארבעת השחקנים בסדר כלשהו, למשל, א-4, ב-4, ג-4, ד-5. שלוש המטלות הבאות – 6,5,5 – ניתנות לשחקנים שהעלות הנוכחי שלהם קטנה ביותר, שהם א, ב, ג; והמטלה הבאה – 6 – ניתנת לשחקן ד. העלויות לאחר הסיבוב השני הן א-9, ב-9, ג-10, ד-11. המטלה האחרונה ניתנת לשחקן א, והעלויות הסופיות הן א-16, ב-10, ג-10, ד-11. הערך האגליטרי הוא -16. אנחנו רואים שהאלגוריתם אינו מוצא את החלוקה האגליטרית (עם עלויות 12, 12, 12, 12), אבל הוא גם לא כל כך רחוק – הערך האגליטרי שלו גדול רק פי 4/3=16/12 מהערך המיטבי.

כשמנתחים אלגוריתמי קירוב, התכונה המעניינת היא **יחס הקירוב** שלהם (approximation ratio) – היחס בין ערך הפתרון שהאלגוריתם מוצא, לבין ערך הפתרון הטוב ביותר. בבעיות מינימיזציה, כמו הבעיה שלנו, יחס הקירוב גדול מ־1, כי הפתרון של האלגוריתם גדול יותר מהפתרון האופטימלי; בבעיות מקסימיזציה, יחס הקירוב קטן מ־1, כי הפתרון של האלגוריתם קטן יותר מהפתרון האופטימלי. בדוגמה למעלה, יחס הקירוב של אלגוריתם הרשימה היה 4/3.

משפט. יחס הקירוב של אלגוריתם הרשימה קטן מ-2.

הוכחה. בהינתן קלט מסויים לבעיה, נסמן ב־OPT את העלות האגליטרית (העלות הגדולה ביותר בחלוקה האגליטרית). לצורך ההוכחה, ננרמל את עלויות כל המטלות ע"י חלוקה ב־OPT. לאחר הנירמול, סכום העלויות של המטלות שמקבל שחקן כלשהו בחלוקה האגליטרית קטן או שווה 1. לכן, העלות של כל מטלה ומטלה קטנה או שווה 1. כמו כן, סכום העלויות של כל המטלות קטן או שווה n (מספר השחקנים).

בכל סיבוב באלגוריתם, סכום העלויות של כל המטלות שכבר חולקו קטן ממש מ־n. לכן, לפי כלל שובך־היונים, סכום העלויות הקטן ביותר של שחקן כלשהו קטן מ־1. לכן, האלגוריתם נותן את המטלה הבאה לשחקן שסכום העלויות שלו קטן מ־1. סכום העלויות החדש של שחקן זה קטן מ־2. מכאן, שלכל שחקן המקבל מטלה, כולל השחקן האחרון, סכום העלויות קטן מ־2. ***

ע"י שינוי קל בהוכחה, ניתן להוכיח משפט מעט חזק יותר – יחס הקירוב של אלגוריתם הרשימה בחלוקה ל-n שחקנים הוא לכל היותר 2 פחות n/1 (מטלה).

אלגוריתם הרשימה יכול לטפל במטלות בכל סדר שבו הן מגיעות, ולכן הוא שימושי גם כאשר המטלות אינן ידועות מראש. חישבו, למשל, על עובדים המקבלים בכל יום מטלות חדשות, וצריכים לחלק את המטלות ביניהם מייד, כך שהחלוקה הכוללת (בסיכום חודשי או שנתי) תהיה הוגנת.

אם כל המטלות ידועות מראש, ניתן להשתמש באלגוריתם המשיג יחס־קירוב טוב יותר. הוא מסדר את המטלות בסדר יורד של העלות, ואז מחלק אותן לפי אלגוריתם הרשימה. בעולם תיזמון העבודות, האלגוריתם הזה נקרא Longest Processing Time first, או בקיצור LPT, כי הוא מתזמן קודם את העבודות שזמן־העיבוד שלהן ארוך ביותר. בעולם חלוקת המספרים, אלגוריתם זה נקרא פשוט **האלגוריתם החמדני.**

- 1. סדר את המטלות בסדר יורד של העלות – מהגדולה לקטנה.
- 2. תן את המטלה הבאה ברשימה לְשחקן, שסכום העלויות הנוכחי שלו קטן ביותר.
- 3. אם יש עוד מטלות, חזור לשורה 2.

בדוגמה למעלה, המטלה הראשונה בסדרה היא ה־7 והאחרונה היא ה־4. האלגוריתם נותן את ארבעת המטלות הראשונות לארבעת השחקנים בסדר כלשהו, למשל, א-7, ב-7, ג-6, ד-6. שתי המטלות הבאות – 5,5 – ניתנות לשחקנים שהעלות הנוכחי שלהם קטנה ביותר, שהם ג ו-ד. שתי המטלות הבאות ניתנות לשחקנים שהעלות הנוכחית שלהם היא קטנה ביותר, שהם א ו-ב. עכשיו, סכום העלויות של כל ארבעת השחקנים הוא 11, ולכן המטלה האחרונה ניתנת לשחקן כלשהו באופן שרירותי, נניח לשחקן א. בסופו של דבר, עלויות השחקנים הן 11, 11, 11, 15. יחס־הקירוב של האלגוריתם בדוגמה זו הוא 15/12 = 5/4 – טוב יותר משל אלגוריתם הרשימה. המשפט הבא מנתח את יחס־הקירוב של האלגוריתם החמדני באופן כללי.

משפט. לכל n, יחס־הקירוב של האלגוריתם החמדני בחלוקת מטלות אגליטרית ל־n שחקנים הוא לכל היותר:

4/3 – 1/(3n).

למשל, כאשר n=4, יחס הקירוב הוא לכל היותר 4/3 – 1/12 = 15/12 = 1.25, בדיוק כמו בדוגמה. ניתן להכליל את הדוגמה ולהראות, עבור כל n, דוגמה שבה יחס־הקירוב הוא בדיוק הביטוי שבמשפט, כך שהמשפט הוא הדוק.

הוכחה. בהינתן קלט מסויים לבעיה, נסמן ב־OPT את העלות הגדולה ביותר של סל בחלוקה המיטבית לקלט זה. ננרמל את העלויות ע"י חלוקת ערכי כל המטלות ב־OPT. לאחר נירמול זה, עלויות כל הסלים בחלוקה המיטבית קטנות או שוות 1. לכן סכום עלויות כל המטלות קטן או שווה n, והעלות של כל מטלה לאחר הנירמול קטנה או שווה 1. נחלק את המטלות לשני סוגים:

- מטלות עם עלות גדולה ממש מ־1/3 ייקראו **גדולות**;
- מטלות עם עלות קטנה או שווה ל־1/3 ייקראו **קטנות**.

מהנירמול נובע, שכל סל בחלוקה המיטבית כולל לכל היותר שתי מטלות גדולות, ובסך־הכל יש לכל היותר 2n מטלות גדולות.

- בדוגמה (שבה n=4), הנירמול מתבצע ע"י חלוקה ב־12. יש שש מטלות גדולות, עם עלויות 7/12, 7/12, 6/12, 6/12, 5/12, 5/12. שאר המטלות הן קטנות, ולכל אחת מהן עלות 4/12. בחלוקה המיטבית יש שלושה סלים עם שתי מטלות גדולות, בסכום כולל 1, ועוד סל אחד עם שלוש מטלות קטנות, בסכום כולל 1.

האלגוריתם החמדני, על-פי הגדרתו, מחלק קודם את כל המטלות הגדולות, ואז את כל המטלות הקטנות. אנחנו נוכיח את המשפט בשתי טענות־עזר: טענה א מתייחסת לשלב חלוקת המטלות הגדולות, וטענה ב מתייחסת לשלב חלוקת המטלות הקטנות.

טענה א: לאחר שהאלגוריתם סיים לחלק מטלות גדולות, סכום העלויות בידי כל שחקן הוא לכל היותר 1.

הוכחה: אם מספר המטלות הגדולות הוא לכל היותר n, אז האלגוריתם החמדני נותן מטלה גדולה אחת בלבד לכל שחקן, וברור שעלויות כל השחקנים קטנות או שוות 1. לכן נניח שמספר המטלות הגדולות הוא n+t, עבור מספר שלם כלשהו t בין 1 ל־n. לכן בחלוקה המיטבית ישנם t סלים עם שתי מטלות גדולות, ועוד n–t סלים עם מטלה גדולה אחת. נקרא לשתי מטלות גדולות:

- **משודכות** – אם הן נמצאות יחד בסל אחד בחלוקה המיטבית;
- **תואמות** – אם סכום העלויות שלהן קטן או שווה 1.

כל שתי מטלות משודכות הן תואמות. העלות של כל מטלה, התואמת למטלה גדולה אחרת, קטן מ־2/3. ישנם t זוגות של מטלות גדולות משודכות (ולכן תואמות), ועוד n–t מטלות לא־משודכות. לכן:

- מתוך מטלות 1 עד n–t+1, לפחות אחת משודכת (ולכן תואמת) למטלה אחרת כלשהי. לכן, המטלה הקטנה ביותר בקבוצה זו, שהיא מטלה n–t+1, בהכרח תואמת למטלה הקטנה ביותר מבין הגדולות, שהיא מטלה n+t.
- מתוך מטלות 1 עד n–t+2, לפחות שתיים משודכות (ולכן תואמות) למטלות אחרות כלשהן. לכן, המטלה הקטנה ביותר בקבוצה זו, שהיא מטלה n–t+2, בהכרח תואמת למטלה השניה בקוטנה מבין הגדולות, שהיא מטלה n+t–1.
- משיקולים דומים, מטלה n–t+3 תואמת למטלה n+t–2, ובאופן כללי, מטלה n–t+k תואמת למטלה n+t–k+1 לכל k בין 1 ל־t. בפרט, מטלה n תואמת למטלה n+1.

עכשיו נחזור לאלגוריתם החמדני. האלגוריתם מחלק את מטלות 1, ..., n, מטלה אחת לכל שחקן. כשהוא מגיע למטלה גדולה n+1, הוא נותן אותה לשחקן שקיבל את המטלה הקטנה ביותר שחולקה עד כה, שהיא מטלה n. כאמור בנקודה השלישית למעלה, מטלה n תואמת למטלה n+1, ולכן סכום העלויות של שחקן זה הוא לכל היותר 1.

כשהאלגוריתם מגיע למטלה גדולה n+2, הוא נותן אותה לשחקן שקיבל את מטלה n–1. כאמור בנקודות למעלה, שתי המטלות הללו תואמות, כלומר סכומן לכל היותר 1, וכל אחת מהן קטנה מ־2/3. לעומת זאת, סכום העלויות של השחקן שקיבל את המטלות n+1 , n גדול מ 2/3. לכן האלגוריתם אכן נותן את מטלה n+2 לשחקן שקיבל את n–1, וערכו של שחקן זה נשאר לכל היותר 1.

באותו אופן, לכל k בין 1 ל־t, האלגוריתם נותן את מטלה n+t–k+1 לשחקן שקיבל את n–t+k. הזוגות הללו תואמים לכל k, ולכן עלויות כל השחקנים לכל היותר 1. בכך הוכחנו את טענה א.

טענה ב: כאשר האלגוריתם נותן מטלה קטנה לשחקן כלשהו, סכום העלויות החדש של השחקן (כולל המטלה שקיבל) קטן או שווה:

4/3 – 1/(3n).

הוכחה: נסמן את עלות המטלה ב־x. סכום העלויות של המטלות שחולקו עד כה הוא לכל היותר n–x. לפי כלל שובך־היונים, סכום העלויות של השחקן המקבל את המטלה הוא לכל היותר:

(n–x)/n = 1–x/n

לפני שקיבל את המטלה, ולכן לכל היותר:

1–x/n+x = 1+x*(1–1/n)

לאחר שקיבל את המטלה. כיוון שהמטלה קטנה, x≤1/3. לכן סכום העלויות החדש של השחקן הוא לכל היותר:

1+(1–1/n)/3 = 4/3–1/3n.

בכך סיימנו את הוכחת טענה ב. משתי הטענות יחד נובעת נכונות המשפט.

מש"ל.

עד עכשיו דיברנו על חלוקת **מטלות** (עם ערכים שליליים). ניתן להשתמש באלגוריתם החמדני גם לחלוקת **חפצים** (עם ערכים חיוביים). האלגוריתם עובד באותו אופן: הוא מסדר את החפצים בסדר יורד של הערך שלהם, ונותן כל חפץ לשחקן שהערך הנוכחי שלו קטן ביותר. יחס־הקירוב של האלגוריתם נתון במשפט הבא:

משפט. לכל n, יחס־הקירוב של האלגוריתם החמדני בחלוקת חפצים אגליטרית ל־n שחקנים הוא לפחות:

(3n–1)/(4n–2).

המשפט הוכח בהדרגה בכמה מאמרים על־ידי חוקרים שונים. כל ההוכחות הידועות כיום הן ארוכות ומסובכות הרבה יותר מההוכחה עבור מטלות, ולא נראה אותן כאן.

סיכום

למדנו שני סוגי אלגוריתמים: מדוייק (חיפוש במרחב המצבים) ומקורב (חמדני). במציאות מקובל להשתמש בשניהם יחד: משתמשים באלגוריתם המדוייק, ומייצרים חסמים פסימיים בעזרת האלגוריתם המקורב. כשנגמר זמן החישוב, לוקחים את התוצאה הטובה ביותר שנתן לנו האלגוריתם המדוייק – ומובטח לנו שהיא תהיה טובה לפחות כמו האלגוריתם המקורב.

מקורות

- *Wikipedia pages:*

- "*Partition problem*".
- "*Longest-processing-time-first scheduling*".
- "*Greedy number partitioning*".

סיכם: אראל סגל-הלוי.