בס"ד

קורס תכנות אלגוריתמים מחקריים

פסאודו-קוד למאמר הקצאות בורדה פרופורציונליות:

מחברי המאמר: Andreas Darmann, Christian Klamler
מחבר סיכום המאמר ופסאודו-קוד: שלמה גליק

בעברית:

מקרה א' (n=k) (טענה 2)

תיעוד:

מקבלת פריטים והעדפות של כל שחקן כך שמספר הפריטים שווה למספר ההעדפות
בודקת אם קיימת חלוקה פרופורציונלית ואם קיימת מחזירה אותה
במידה ולא קיימת, מחזירה "לא קיימת חלוקה פרופורציונלית"

קלט:

רשימת_הפריטים,
דירוגי_פריטים, // באינדקס הi יהיה את האינדקסים של הפריטים מדורגים לפי הדירוג של השחקן הi

מימוש:

n :=: אורך רשימת תורי העדיפויות // מספר השחקנים
k :=: אורך רשימת הפריטים // מספר הפריטים
// $k == n$
סף :=: $(k-1)/2$
גרף :=: גרף של אפסים בגודל $n*n$
נרוץ בלולאה $0 \leqslant i \leqslant n$: // ריצה על כל השחקנים
- נרוץ בלולאה $0 \leqslant j \leqslant n$: //ריצה על דירוג הפריטים של השחקן הi
  - אם דירוגי_פריטים$[i][j]$ $\leqslant$ סף:
    - גרף$[i][j]$ = 1

// כרגע גרף$[i][j]$ == 1 אם"ם הדירוג שהשחקן הi נתן לפריט j גדול מהסף
// ובעצם נוצר לנו גרף דו צדדי (צד אחד שחקנים וצד שני פריטים)

שידוך = שידוך_מושלם_בגרף_דו-צדדי(גרף)
// הפונקציה הזאת מצורפת בסוף, היא לא חלק מהמאמר ולקחתי אותה מ
// https://www.geeksforgeeks.org/maximum-bipartite-matching/
// עם תוספת קטנה שתבדוק אם השידוך מושלם ובמידה וכן שתחזיר את השידוך עצמו

אם קיבלנו שידוך:
נחזיר אותו
אחרת:
נחזיר "לא קיים שידוך"

בס"ד

באנגלית:

```
proportional_division_equal_number_of_items_and_players(items, preferences):
    k = len(items)
    n = k
    threshold = (k-1)/2
    graph = [n][n]
  for i in n:
    for j in n:
      if preferences[i][j] >= threshold:
        graph[i][j] = 1

  ans = GFG(graph).getMatch()
  if ans:
    return ans.map(x → items[x])
  else:
    return "not exists match"
```

בס"ד

פונקציית הבחירה:

בעברית:

תיעוד:

מקבלת סדר בחירה, רשימת של רשימות הפריטים שכל שחקן כבר בחר ,פריטים והעדפות,
רצה על סדר הבחירה ומחלקת לכל שחקן את הכי מועדף עליו שעדין לא נבחר. כמות הפעמים שנרוץ על סדר הבחירה יהיה לפי כמות האיטראציות שנקבל בקלט

קלט:

סדר_הבחירה, // רשימה

נבחרים, // רשימה של רשימות במיקום הi יהיה רשימת הפריטים שהשחקן הi בחר כבר

רשימת_הפריטים,

רשימת_תורי_העדיפויות, // באינדקס ה i יהיה את העדיפויות של השחקן ה i

מימוש:

נרוץ בעוד לולאה על $i \ni$ סדר_הבחירה

- תור עדיפויות נוכחי := התור שנמצא במיקום הi ברשימת התורים // ההעדפה של השחקן הi
- פריט נבחר := נשלוף את הפריט הראשון בתור העדיפויות הנוכחי // הפריט ימחק מראש התור
- נשתמש בלולאת while ואם הפריט הנבחר לא נמצא בפריטים שנשארו נשלוף את הפריט המועדף הבא
- נוסיף את הנבחר לנבחרים במיקום הi
- מחיקת הנבחר מרשימת הפריטים

פלט:

הנבחרים,

רשימת_הפריטים, // רשימת הפריטים תכיל רק את הפריטים שנשארו

רשימת_תורי_העדיפויות // רשימת_תורי_העדיפויות תכיל עבור כל שחקן את העדיפויות שעוד לא שלפנו

באנגלית:

```
selection_in_order(order, selected, items, preferences):
    for j in order:
        favorite = preferences[j].poll()
        while(favorite not in items):
            favorite = preferences[j].poll()
        slected[j].append(favorite)
        items.remove(favorite)
    return selected, items, preferences
```

בס"ד

מקרה ב' ($\varphi \geqslant 2$ זוגי) (טענה 3)

בעברית:

קלט:

רשימת_הפריטים,

עדיפויות (רשימה כך שבאינדקס הi יהיה את תור העדיפויות של השחקן הi)

מימוש:

נגדיר:

n := אורך רשימת תורי העדיפויות // מספר השחקנים

k := אורך רשימת הפריטים // מספר הפריטים

$p := k/n$

s := $[1,2,3,....,n,n,n-1,n-2,.....,1]$ // רשימה של אינדקסים באורך $2n$

הנבחרים: = רשימה באורך n של רשימות ריקות

נרוץ בלולאה $\frac{p}{2}$ פעמיים:

- עדיפויות, הנבחרים, רשימת הפריטים := **פונקציית הבחירה(**סדר_הבחירה = s, הנבחרים**,** רשימת הפריטים, עדיפויות**)**

פלט: הנבחרים

באנגלית:

proportional_division_with_p_even$(items,\ preferences)$:

$n = preferences.len$

$selected = [n][]$

$k = items.len$

$p = k/n$

$s1 = [1,2,3,....,n]$

$s2 = s1.reverse()$

$order = s1 + s2$

$len = p/2$ // p even

$for\ i\ in\ len$:

$selected,\ items,\ preferences =$ **selection_in_order**$(order,\ selected,\ items,\ preferences)$

return $selected$

בס"ד

מקרה ג' (n אי-זוגי) (משפט 1)

בעברית:

קלט: רשימת_הפריטים, עדיפויות (רשימה כך שבאינדקס הi יהיה את תור העדיפויות של השחקן הi)

מימוש:

נגדיר:

n := אורך רשימת תורי העדיפויות (מספר השחקנים)

k := אורך רשימת הפריטים (מספר הפריטים)

אם $\frac{k}{n} == 1$:

- נריץ את הפונקציה ממקרה א' ונחזיר כמוה

אם k זוגי:

- נריץ את הפוניקציה של מקרה ב' על הקלט ונחזיר את מה שהיא תחזיר

אחרת:

- $\frac{k}{n} - 3 =: p$ // k מחלק את n וגם $\frac{k}{n} \geqslant 3$
- $[1, 2, 3, ..., n] =: q1$
- $[n, n-2, n-4, n-6, ..., 1] =: q2$
- $[n-1, n-3, n-5, ..., 2] =: q3$
- $q1 + q2 + q3 + q3 + q2 =: q$ // רשימה של אינדקסים באורך $3n$
- $q1 + q1.reverse() =: s$ // רשימה של אינדקסים באורך $2n$
- הנבחרים := רשימה באורך n של רשימות ריקות
- עדיפויות, הנבחרים, רשימת הפריטים := **פונקציית הבחירה(**סדר_הבחירה = q, הנבחרים**,** רשימת הפריטים, עדיפויות**)**
- נרוץ בלולאה $\frac{p}{2}$ פעמיים:
  - עדיפויות, הנבחרים, רשימת הפריטים := **פונקציית הבחירה(**סדר_הבחירה = s, הנבחרים**,** רשימת הפריטים, עדיפויות**)**

פלט: הנבחרים

בס"ד

באנגלית:

```
proportional_division_with_n_odd(items, preferences):
    k = items.len
    n = preferences.len
    if k/n == 1:
        return proportional_division_equal_number_of_items_and_players(items, preferences)
    if k mod 2 == 0:
        return proportional_division_with_p_even(items, preferences)
    p = k/n - 3
    q1 = [1, 2,..... n]
    q2 = [n, n-2, n-4, n-6,..., 1]
    q3 = [n-1, n-3, n-5,..., 2]
    order3n = q1 + q2 + q3 + q3 + q2
    order2n = q1 + q1.reverse()
    selected = [n][]
    selected, items, preferences = selection_in_order(order3n, selected, items, preferences)
    len = p/2                    // p even
    for i in len:
        selected, items, preferences = selection_in_order(order2n, selected, items, preferences)
    return selected
```

בס"ד

מקרה כללי (משפט 3)

בעברית:

קלט: רשימת_הפריטים, עדיפויות (רשימה כך שבאינדקס הi יהיה את תור העדיפויות של השחקן הi)

מימוש:

n := אורך רשימת תורי העדיפויות (מספר השחקנים)

k := אורך רשימת הפריטים (מספר הפריטים)

אם $\frac{k}{n} == 1$:

- נריץ את הפונקציה ממקרה א' ונחזיר כמוה

אם n אי זוגי:

- נריץ את הפונקציה ממקרה ג' ונחזיר כמוה

אחרת אם $\frac{k}{n}$ זוגי :

- נריץ את הפונקציה ממקרה ב' ונחזיר כמוה

אחרת: // $\frac{k}{n} \geqslant 3$ אי זוגי

- $p := \frac{k}{n} - 3$
- $q1 := [1, 2, 3, ..., n]$
- $q2 := [n, n-2, n-4, n-6, ..., 2]$
- $q3 := [n-1, n-3, n-5, ..., 1]$
- $q := q1 + q2 + q3 + q3 + q2$ // רשימה של אינדקסים באורך $3n$
- $s := q1 + q1.reverse()$ // רשימה של אינדקסים באורך $2n$
- הנבחרים := רשימה באורך n של רשימות ריקות
- עדיפויות, הנבחרים, רשימת הפריטים := **פונקציית הבחירה**(סדר_הבחירה = q, הנבחרים**,** רשימת הפריטים, עדיפויות**)**
- נרוץ בלולאה $\frac{p}{2}$ פעמיים:
  - עדיפויות, הנבחרים, רשימת הפריטים := **פונקציית הבחירה**(סדר_הבחירה = s, הנבחרים**,** רשימת הפריטים, עדיפויות**)**

פלט: הנבחרים

בס"ד

באנגלית:

```
proportional_division(items, preferences):
    n = preferences.len
    if k/n == 1:
        return proportional_division_equal_number_of_items_and_players(items, preferences)
    if n mod 2 == 1:
        return proportional_division_with_n_odd(items, preferences)
    else if k/n mod 2 == 0:
        return proportional_division_with_p_even(items, preferences)
    else:
        p = k/n - 3
        q1 = [1, 2,..... n]
        q2 = [n, n-2, n-4, n-6,..., 2]
        q3 = [n-1, n-3, n-5,..., 1]
        order3n = q1 + q2 + q3 + q3 + q2
        order2n = q1 + q1.reverse()
        selected = [n][]
        selected, items, preferences = selection_in_order(order3n, selected, items, preferences)
        len = p/2                // p even
        for i in len:
            selected, items, preferences = selection_in_order(order2n, selected, items, preferences)
        return selected
```

בס"ד

נספח מציאת שידוך מושלם בגרף דו-צדדי

ע"י מימוש מאתר https://www.geeksforgeeks.org/maximum-bipartite-matching/

(הוספתי תנאי שאם לא קיים שידוך מושלם שיחזיר false ושאם קיים יחזיר את השידוך)

```
# Python program to find
# Maximal Bipartite matching.

class GFG:
    def __init__(self,graph):
        # Residual graph
        self.graph = graph
        self.ppl = len(graph)
        self.jobs = len(graph[0])
    # A DFS based recursive function
    # That returns true if a matching
    # For vertex u is possible
    def bpm(self, u, seen):
        # Try every job one by one
        for v in range(self.jobs):
            # If applicant u is interested
            # in job v and v is not seen
            if self.graph[u][v] and seen[v] == False:
                # Mark v as visited
                seen[v] = True

                '''If job 'v' is not assigned to
                   an applicant OR previously assigned
                   applicant for job v (which is matchR[v])
                   has an alternate job available.
                   Since v is marked as visited in the
                   above line, matchR[v]  in the following
                   recursive call will not get job 'v' again'''
                if self.matchR[v] == -1 or self.bpm(self.matchR[v], seen):
                    self.matchR[v] = u
                    return True
        return False
    # Returns maximum number of matchings
    def getMatch(self):
        '''An array to keep track of the
           applicants assigned to jobs.
           The value of matchR[i] is the
           applicant number assigned to job i,
           the value -1 indicates nobody is assigned.'''
        self.matchR = [-1] * self.jobs

        # Count of jobs assigned to applicants
```

```
        count = 0
        for i in range(self.ppl):

            # Mark all jobs as not seen for next applicant.
            seen = [False] * self.jobs
            # Find if the applicant 'u' can get a job
            if self.bpm(i, seen):
                count += 1
        if count == self.jobs:
            return self.matchR
        return False
```