Algorithms II

Ariel-University

Matan-Ben Nagar 1/1/21 Computer Sciense

<u>תדירות הופעת שאלות במבחנים</u>

Fire	**6**
BFS	**2**
Water Bottle	**3**
Floyd Warshall	**3**
Euler	**5**
DFS	**5**
Bipartite	**2**
is Cycle	**2**
print cycle	**1**
Dijkstra	2
Print shortest path	1
Best	**4**
Circular Best	**3**
Super Best	3
Kruskal	**5**
Huffman	**5**
print Huffman	**3**
Build Tree from degrees list	**1**
Isomorphism	
return connected components	
Boruvka's	
Prim	
Reverse Kruskal	1

<u>שאלות יצירתיות</u>

נתונים N מספרים המצויים על קטע ומספר שלם $1 \le k \le N$. יש לפתח אלגוריתם שמחזיר את הסכום המקסימלי של מספרים הנמצאים על איזשהו תת-קטע רציף בעל אורך k
נתונים עץ וקודקוד שלו. לבנות אלגוריתם שמקבל עץ וקודקוד (עלה או פנימי) ומחזיר האם העלה שייך לאיזשהו קוטר של העץ.
האם גרף הוא עץ? (נבדוק תנאי קשירות ומס' צלעות $= \|V\| - 1$
תת קטע **קצר ביותר** עם סכום איברים מקסימלי
האם ניתן להפוך גרף לאוילריאני על ידי הוספת צלעות? **כן.**
להחזיר את רכיבי הקשירות של כל רכיב
כמה מעגלים יש בגרף?
כמות צבעים מינימלית לצביעת עץ? לצבוע עץ. (גרף דו צדדי הוא גרף ללא מעגלים באורך אי זוגי)
לבנות אלגוריתם שמקבל עץ ומחזיר את כל קודקודי העץ שלא שייכים לאף קוטר העץ
נתונים 2 קודקודים $s, t \in V$. יש להחזיר את מספר המסלולים הקצרים ביותר בין s לבין קודקוד t.
להחזיר האם גרף הוא דו-צדדי
אלגוריתם למציאת מעגל בגרף והחזרתו.
רוורס קרוסקל
לבדוק האם צלע שייכת למעגל: **רעיון:** להסיר את הצלע ולבצע BFS כדי לראות אם עדיין יש דרך להגיע מ-u ל-v

עצים: הקדמה

- <u>מהו עץ?</u>
  1. גרף קשיר ללא מעגלים
  2. גרף קשיר עם n-1 צלעות
  3. גרף עם n-1 צלעות ללא מעגלים

  **ברגע שלפחות 2 מההגדרות מתקיימות, אז הגרף הוא בהכרח עץ*

- <u>מושגים:</u>
  1. מרכז: הנקודה מימנה יוצא הרדיוס
  2. רדיוס: קטע המחבר מרכז מעגל לשפת מעגל
  3. קוטר: הקטע הכי ארוך בין 2 קצוות מעגל ובהכרח עובר דרך מרכז המעגל.
  4. עלה : כל מי שיש לו רק שכן אחד ברשימת שכנויות

- <u>איך מושגים אלו קשורים לעצים?</u>
  1. קוטר: מרחק מקסימלי בין 2 קודקודים בעץ
  2. רדיוס שווה למחצית מהקוטר
  3. מרכז הוא אמצע הקוטר

- <u>מסקנה:</u>
  1. קוטר זוגי: אז הרדיוס הוא $\frac{1}{2}$ מהקוטר ויש מרכז אחד.
  2. קוטר אי-זוגי: הרדיוס הוא $\lfloor\frac{1}{2}\rfloor$ מהקוטר בעיגול כלפי מטה ויש לו 2 מרכזים!

- <u>עצים פורסים:</u>
  1. עץ פורש: של גרף קשיר G הוא תת-גרף קשיר G, המכיל את כל צומתי G, ואין לו מעגלים.
  2. עץ פורש מינימאלי: עץ פורש בעל סך משקלים מינימלי

ייצוג גרפים במחשב

1. **<u>רשימת סמיכויות:</u>**

סיבוכיות מעבר: O(|E|+|V|)

מערך שהתא ה-i מייצג את קודקוד i. והערך בתא הוא רשימת מקושרת של כל השנים של קודקוד i (בגרף מכוון זה אומר שהחץ הוא מ-i לאותו שכן)

<u>משקלים על הצלעות:</u> אם יש משקלים על הצלעות אז כל איבר ברשימה הוא למעשה Node שיש בתוכו את האינדקס של הקודקוד השכן ואת משקל הצלע מ-i לאותו שכן.

2. **<u>מטריצת שכנויות:</u>**

סיבוכיות: O(|V|²)

מטריצה שהתא (i,j) מייצג האם יש צלע בין i ל-j.

<u>אין משקלים על הצלעות:</u> מטריצה בוליאנית (0 או 1)

<u>יש משקלים על הצלעות:</u> באלכסון תמיד יש 0 (כי קודקוד לא מחובר לעצמו). אם אין צלע המשקל הוא אינסוף, אם יש צלע המשקל מיוצגת בתא ה-(i,j).

<u>מציאת מסלולים קצרים ביותר בגרף</u>

מסלול קצר = הדרך להגיע מקודקוד אחד לאחר במספר צלעות מינימאלי (אין משקל על הצלעות)

מסלול קל = הדרך להגיע מקודקוד אחד לאחר בעלות כוללת מינימאלית (יש משקל על הצלעות)

- מסלול קל ביותר בין קודקוד לכל האחרים: Dijsktra
- מסלול קצר ביותר בין קודקוד לכל האחרים: BFS
- מסלול קצר\קל ביותר בין כולם לכולם: Floyd Warshall

אלגוריתם Floyd Warshall

קלט: מטריצת שכנויות

רעיון: כל קודקוד k משמש כמתווך בין כל 2 קודקודים i,j.

סיבוכיות: $O(|V|^3)$

פאסודו קוד:

```
FW(G)
        D = new Matrix()
        P = new Matrix()

        for i=1 to |V|
          for j=1 to |V|
              D[i][j] = G[i][j]
              if (D[i][j] != inf && D[i][j] != 0)
                P[i][j] = "->"+j
        for k = 1 to |V|
          for i = 1 to |V|
            for j = 1 to |V|
              D[i][j] = min(D[i][j], D[k][j]+D[i][k])
              if (D[i][j] > D[i][k] + D[k][j])
                  P[i][j] = P[i][k] + P[k][j]
```

מה האלגוריתם עושה: האלגוריתם למעשה מחזיר לנו מטריצה שכנויות בה, בכל תא (i,j) יופיע אורך המסלול הקצר ביותר או האם קיים מסלול.

הסבר:

נוכיח באינדוקציה על k שבשלב ה-k אנו מוצאים את המסלול הקל ביותר בין כל 2 קודקודים שלא עובר בקודקודים שהאינדקס שלהם גדול ממש מ-k.

בסיס: אתחול = אסור לעבור באמצע דרך אף קודקוד ולכן המסלול הקל ביותר הוא משקל הצלע (או אינסוף אם אין צלע).

צעד: נניח שהטענה נכונה עבור k ונוכיח עבור k+1.

לפי הנחת האינדוקציה, כל המסלולים בין i ל-j חושבו נכון (כלומר הם הכי קלים), כך שהם לא עוברים באמצע דרך קודקודים עם אינדקס גבוהה ממש מ-k.

בשלב ה-k+1: המסלול הקל ביותר בין i ל-j יכול לא לעבור דרך k+1 ואז הוא נשאר כמו מקודם, או שדרך k+1 נקבל מסלול קל יותר ואז זה שווה להגיע מ-i ל-k+1 (ללא מעבר דרך קודקודים שהאינדקס שלהם גבוהה מ-k) ואז מk+1 ל-j (ללא מעבר דרך קודקודים שהאינדקס שלהם גבוהה מ-k) ואלו חושבו כבר לפי הנחת האינדוקציה

הערות:

אלגוריתם פלויד-וורשל עובד נכון רק אם אין מעגלים שליליים בגרף.

• משקלים על הקודקודים

כתבו אלגוריתם למציאת המסלול הקל ביותר בין כל 2 קודקודים כאשר המשקלים הם רק על הקודקודים

<u>פתרון:</u>

קלט: גרף G עם מטריצת שכנויות בוליאנית (ללא משקלים על הצלעות) + מערך W של משקלי קודקודים

1. נמיר את המשקלים על הקודקודים למשקלים על צלעות

לכל צלע בין i ל-j נכניס למיקום ה-(i,j) במטריצה את הערך $w[i] + w[j]$

2. לאחר מכן נריץ את אלגוריתם פלויד-וורשל על הגרף שהתקבל.
3. נקבל מפלויד וורשל מטריצה שגויה, כל קודקוד שהיה באמצע נספר פעמיים חוץ מקודקודי הקצה.

לכן נעבור על מטריצת התוצאה ונתקן את תא (i,j) באופן הבא:

$$G[i][j] = \frac{D[i][j] + w[i] + w[j]}{2}$$

```
FW(G,W) //G weightless graph, W is array of weights
      D = new Matrix()

      for i = 1 to |V|
         for j = 1 to |V|
            D[i][j] = W[i] + W[j]

      for k = 1 to |V|
          for i = 1 to |V|
             for j = 1 to |V|
                D[i][j] = min (D[i][j], D[i][k] + D[k][j])

      for i = 1 to |V|
         for j = 1 to |V|
            D[i][j] = (D[i][j] + w[i]+ w[j]) / 2
```

• משקלים על הצלעות ועל הקודקודים

כתבו אלגוריתם למציאת מסלול קל ביותר בין כל 2 קודקודים, כאשר המשקלים הם גם על הקודקודים וגם על הצלעות

<u>פתרון</u>

קלט: נקבל מטריצת שכנויות עם משקלים וגם מערך משקלים על קודקודים.

1. נמיר את כל המשקלים למשקלים על הצלעות, לכל צלע (i,j) נכניס למיקום המתאים במטריצה את הערך:

$$D[i][j] = 2 * G[i][j] + W[i] + W[j]$$

2. כעת המטריצה שנקבל חזרה היא בעלת ערכת שגויים ולכן נרוץ על כל המטריצה שוב ונתקן את החריגה:

$$D[i][j] = \frac{D[i][j]+W[i]+W[j]}{2}$$

```
FW(G,W) //G weightless graph, W is array of weights
    D = new Matrix()

    for i = 1 to |V|
        for j = 1 to |V|
            D[i][j] = 2*G[i][j] + W[i] + W[j]

    for k = 1 to |V|
        for i = 1 to |V|
            for j = 1 to |V|
                D[i][j] = min (D[i][j], D[i][k] + D[k][j])

    for i = 1 to |V|
        for j = 1 to |V|
            D[i][j] = (D[i][j] + W[i]+ w[j]) / 2
```

- ## מטריצת מסלולים

כתבו אלגוריתם שמחזיר מטריצת מסלולים (כלומר בכל תא, יופיע המסלול הקצר ביותר)

```
FW(G)
      D = new Matrix()
      P = new Matrix()
      for i=1 to |V|
        for j=1 to |V|
          D[i][j] = G[i][j]
          if(D[i][j]!=inf && D[i][j]!=0) P[i][j] = "->" + j


      for k = 1 to |V|
        for i = 1 to |V|
          for j = 1 to |V|
            D[i][j] = min(D[i][j], D[i][k] + D[k][j])
            if(D[i][j]> D[i][k] + D[k][j]) P[i][j] = P[i][k] + P[k][j]
```

• האם הגרף קשיר (לא מכוון)

כתבו אלגוריתם שמחזיר האם המטריצה היא קשירה או לא קשירה (גם עבור מטריצה בוליאנית וגם עבור מטריצת משקלים)

<u>פתרון:</u>

נפעיל את פלויד-וורשל על המטריצה שלנו ובסוף נבצע איטרציה על השורה הראשונה בלבד כדי לראות שאין 0 או אין ∞ .

• כמה רכיבי קשירות יש? (לא מכוון)

כתבו אלגוריתם שמחזיר כמה רכיבי קשירות יש בגרף

<u>פתרון:</u>

נייצר מערך עזר עבור כל קודקוד.

נעבור על המערך ובכל פעם שנראה 0, נדע שעוד לא הגענו לקודקוד הזה.

נבדוק במטריצת השכנויות מי השכנים שלו ונסמן במערך.

נוסיף counter שבודק לנו מה מס' רכיבי הקשירות ונעלה אותו כל פעם.

כך נעבור על כך המערך, ורק על השורות הרלוונטיות במטריצה.

```
Count_Components(G)
    D = new Matrix()
    D = FW(G)
    N = new Array (|V|)
init N to 0
    counter

    for i = 1 to |V|
      if (N[i] == 0)
        N[i] ++
        for j = 0 to |V|
          if (G[i][j] != 0 && G[i][j] != inf)
            N[j]++
        counter++

return counter
```

• משקלים שליליים בגרף

<u>מה זה מעגל שלילי?</u>

מעגל שסכום משקלי צלעותיו הוא שלילי.

<u>נחלק את הבעיה עבור 2 מקרים:</u>

1. **<u>גרף מכוון:</u>**

בגרף מכוון יכול להיווצר מעגל שלילי ולא נוכל להפעיל את FW כדי לגלות את זה.
האלגוריתם לא ייתן לנו את התוצאה הנכונה, אבל הוא יודע להצביע על זה שיש מעגל שלילי בגרף.
איך? נסתכל על האלכסון במטריצה שמראה לנו את המרחק של קודקוד לעצמו.
אם קיים באלכסון מרחק שלילי, נוכל לדעת בוודאות שיש מעגל שלילי בגרף.

<u>סיבוכיות:</u>

הפעלת אלגוריתם FW עד הסוף במקרה הרע.

```
bool FW_Directed_NegativeCycle(G)
  M = new Matrix()
  for k = 1 to N
    for i = 1 to N
      for j = 1 to N
        M[i][j] = min(M[i][j], M[i][k]+M[k][j])
  for i = 1 to N
    if (M[i][j] < 0)
      return true;
  return false;
```

2. **<u>גרף לא מכוון:</u>**

מספיק שתהיה לנו צלע אחת שלילית, כדי שהמעגל יהיה שלילי.
לכן המרחק הקצר ביותר בין הקודקודים כבר לא רלוונטי כי ניתן להגיע למינוס אינסוף.

<u>סיבוכיות:</u>

הסיבוכיות למציאת משקל שלילי בגרף היא $O(|E|)$.

```
bool FW_NegativeCycle(G)
  M = new Matrix()
  M = FW(G)
  for i = 1 to |N|
    for j = 1 to |M|
      if (M[i][j] < 0)
        return true;
      end - if
  return false
```

אלגוריתם BFS

קלט: רשימת סמיכויות (רשימה שמחזיקה רשימה של קודקודים שכנים)

סיבוכיות: $O(|V| + |E|)$

הסבר: עוברים על כל קודקוד לכל היותר פעם אחת ועל כל צלע לכל היותר פעמיים (אם הגרף לא מכוון).

אנחנו צובעים כל קודקוד רק פעם אחת בלבן ולכן מובטח לנו שכל קודקוד יכנס פעם אחת לתור ויצא רק פעם אחת מהתור – הוצאה והכנסה שזה בעלות של O(1) ולכן – O(|V|).

עבור כל איטרציה אנחנו עוברים על השכנים של אותו קודקוד ולכן נעבור על O(deg(v)), או במילים אחרות O(|E|).

פאסודו קוד:

```
BFS(G,s)
    dist = new Array[|V|]
    parent = new Array[|V|]
    color = new Array[|V|]

    parent[s] = null
    color[s] = grey
    dist[s] = 0

    for i = 1 to |V|
      color[i] = white

    Queue q = new Queue()
    q.enqueue(s)

    while (q is not empty)
      v = q.dequeue()
      for each n in G[v]
        if ( color[n] == white)
          color[n] = grey
          dist[n] = dist[v] + 1
          parent[n] = v
          q.enqueue(n)
      color[v] = black
```

מה האלגוריתם עושה: סריקה לרוחב של גרף. מעבר על כל קודקודי הגרף החל מקודקוד מקור s.

כיצד עובדת שיטת המעבר? קודם עוברים על הקודקודים הקרובים ביותר ל-s ולאט לאט מתרחקים.

לכן אלגוריתם זה מוצא מסלול קצר ביותר בין s לכל שאר הגרף.

משתמשים בתור, כדי שמי שייכנס ראשון (השכנים הקרובים) גם יצאו ראשונים.

לאחר סיום האלגוריתם dist יכיל את המרחק הקצר ביותר בצלעות של כל קודקוד מ-s.

Parent יכיל בתא ה-i את הקודקוד הקודם במסלול הקצר ביותר בין s ל- i .

המערך color יהיה שחור לכל מי שיש מסלול מ-s אליו.

הערות:

- משתמשים בו עבור מסלול קצר ביותר (ללא משקלים על הצלעות)
- כדי לבדוק אם גרף קשיר:
  - עוברים על מערך הצבעים ומוודאים שכולם בשחור
  - עוברים על parent ומוודאים שקיים רק Null אחד
  - עוברים על distance ומוודאים שקיים רק 0 אחד

• שחזור המסלול הקצר ביותר בין s ל-v.

כתבו אלגוריתם למציאת תיעוד המסלול הקצר ביותר בין קודקוד S לקודקוד V.

<u>פתרון</u>

ניעזר באלגוריתם רקורסיבי שנעזר במערך parent שייצרנו באלגוריתם של BFS.

```
Print_Path(G,s,v):
    parent = new Array[|V|]
    parent = BFS(G,s)
    if v=s
       print s
    else if parent[v] = null:
       print "no path from" s "to" v "exists"
    else
       Print_Path(G,s,parent[v])
    print v
```

• מספר רכיבי קשירות בגרף

כתבו אלגוריתם למציאת מספר רכיבי קשירות בגרף.

<u>פתרון</u>

נריץ את BFS על הקודקוד, נעלה את count ולאחר מכן נבדוק האם קיימים קודקודים שנותרו לבנים.
נפעיל שוב את BFS על אותו קודקוד, נעלה את הcount וחוזר חלילה...

```
Count_Components(G)
    color = new Array[|V|]
    parent = BFS(G,s)
    counter = 1
    for i = 1 to |V|
       if (color[i] == white)
          BFS(G,i)
          counter++
    return counter
```

• אלגוריתם למציאת קוטר בגרף

כתבו אלגוריתם למציאת קוטר של גרף

פתרון

קוטר של גרף היא המרחק המקסימלי שקיים בין שני קודקודים בגרף.

איך נעשה זאת? נשלח קודקוד לBFS ונחזיר מימנו את מס' הקודקוד עם המרחק הגדול ביותר.

כעת נריץ BFS על אותו קודקוד ונחזיר מימנו את המרחק הגדול ביותר.

מה קורה בעצם? בפעם הראשונה ייתכן שהקודקוד שבחרנו הוא קודקוד אמצע כלשהו ולכן נפעיל BFS ונגיע לקודקוד הכי מרוחק מימנו.

כעת מאותו קודקוד שהיה הכי מרוחק נפעיל BFS שוב ונגלה מי הקודקוד הכי מרוחק מימנו.

```
find_diameter(G)
	v = rand()
	D = new Graph()
	D = BFS(G,v)
	v = max value in D.distance
	D = BFS(G,v)
	return max value in D.distance
```

• גרף דו-צדדי

<u>גרף דו-צדדי שלם:</u>

1. קשיר
2. כל הקודקודים מחולקים ל-2 קבוצות: V_1 ו- V_2
3. כל הקודקודים מקבוצה 1 מחוברים לקודקודים מקבוצה 2
4. הדרגה של כל קודקוד

כתבו אלגוריתם המקבל גרף קשיר ומחזיר האם הגרף הוא דו-צדדי

<u>פתרון</u>

1. נבצע BFS כאשר את הראשון נצבע בצבע X
2. אם השכנים שלו בצבע לבן && אני עצמי בצבע X, אז נצבע אותם ב-Y.
3. אחרת נצבע אותם באדום (כי אני כחול).
4. נכניס את השכן לתור.
5. נבדוק אם הצבע של השכן והצבע שלי אינם זהים ונמשיך

```
bool Bipartite_graph(G)
      color = new Array[|V|]
      for i = 1 to |V|
         color[i] = white
      color[1] = red
      Queue q = new Queue()
      q.enqueue(n)

      while ( q is not empty )
         v = q.dequeue()
         foreach n in G[v]
            if (color[n] == white)
               if (color[v] == red) color[n] = blue;
               else color[n] = red
               q.enqueue(n)
            else if (color[n] == color[v]) return false;
    return true;
```

- ## האם יש מעגל בגרף

<u>פתרון</u>

נריץ BFS כרגיל ואם במהלך הסריקה נגלה שכן שהוא לא לבן אבל גם לא ה-father שלי: זה אומר שיש מעגל ונחזיר true.
אם נסיים את כל הלולאה, נחזיר false.

```
bool isCycle(G)
    color = new Array[|Array|]
    parent = new Array[|Array|]

    for i = 1 to |V|
      color[i] = white

    color[1] = grey, parent[1] = null
    Queue q = new Queue()
    q.enqueue(1)

    while (q is not empty)
      v = q.dequeue
      for each n in G[v]
        if (color[n]==white)
          color[n] = grey
          q.equeue(n)
          parent[n] = v
        else if (parent[v] != n) return true;

    return false;
```

- ## האם יש מעגל בגרף ואם קיים, החזר אותו

נאתר את 2 הקודקודים ביניהם יש מעגל וניצור 2 מחרוזות שיתארו את המסלול חזרה של כל אחת מהן: S2v, u2S
עבור כל אחד מהקודקודים נחזור במסלול שלו אחורה עד שנגיע ל1-. נחבר את המסלולים
כל עוד האות במקום 2 שווה למקום האחרון במחרוזת, נמחק את הקצוות של המעגל. לבסוף נחזיר את המעגל

```
isCycle(G)
  color = new Array[|V|]
  pred = new Array[|V|]
  for i = 1 to |V|
    color[i] = white
    pred[i] = -1
  color[1] = grey
  Queue q = new Queue
  q.enqueue(1)
  hasCycle=false
  u' = v' = -1
  while(q is not empty && !hasCycle)
    v = q.dequeue()
    for each u in G[v]
      if(color[u] == white)
        color[u] = grey
        pred[u]=v
        q.enqueue(u)
      else if(color[u] == GRAY)
        u'=u , v'= v
        hasCycle= true
  color[v] = black
  if ( u' == -1 ) return false

  s2v = u2s = ""
  while (u' != -1)
    u2s=u2s+u'
    u'=pred[u]
  while (v' != -1)
    s2v= v'+s2v
    v'=pred[v]
  cycle= s2v+u2s
  while (cycle[2] == cycle[cycle.length-1]
    Remove first and last character from cycle

  return cycle
```

DFS

Depth First Search: אלגוריתם חיפוש לעומק בגרף.

קלט: גרף מכוון / לא מכוון

שימושים:

1. סריקת כל הקודקודים
2. בדיקת קשירות של גרף נתון
3. מציאת מסלול עמוק ביותר

דוגמת הרצה: לחץ כאן

סיבוכיות: $O(|V| + |E|)$

הסבר:

האלגוריתם בוחר באופן שרירותי קודקוד ומתקדם באותו אופן, כלומר בוחר את אחד השכנים שלו ועובר אליו עד שאנחנו נתקלים בקודקוד שכבר ביקרנו בו – נצבע קודקודים אלו בצהוב. כעת נבצע "חזרה אחורה" וכל קודקוד שביקרנו בו שוב, נסמן באפור. נעשה זאת עד שנמצא קודקוד חדש לגשת אליו.

בדומה ל-BFS, נצבע את הקודקודים במהלך החיפוש כדי לציין את המצב שלהם.

הם כולם יתחילו בלבן, לאחר מכן, עבור כל קודקוד שהתגלה נצבע אותו באפור ולאחר שנסיים איתו לחלוטין נצבע בשחור.

מתי נרצה להשתמש ב-DFS:

1. כשנרצה לבדוק מספר רכיבי קשירות בגרף
2. כשיבקשו מאיתנו לבדוק מה מספר הרכיבים שמקיימים תנאי כלשהו
3. כשיבקשו מאיתנו להחזיר מי הם רכיבי הקשירות
4. כשיבקשנו מאיתנו משהו שקשור לאיזומורפיזם

```
DFS(G)
  for each v in V
    color[v] = white
    parent[v] = null
    distance[v] = inf

  for each v in V
    if (color[v] == white)
      DFS_REC(G,v)

DFS_REC(G,v)
  color[v] = Grey
  for each u in G[v]
    if (color[u] == white)
      parent[u] = v
      DFS_REC(G,u)
  color[v] = black
```

החזר מספר רכיבי קשירות בגרף

```
DFS_num_of_connected_components(G)
  count = 0
  for i = 1 to |V|
    color[i] = white

  for each v in V\
    if color[v] == white
      REC_DFS(G,v)
      counter++

  REC_DFS(G,s)
    color[s] = gray;
    for each v in G[s]
      if (color[v] == white)
        REC_DFS(G,v)
    color[s] = black
```

- **החזר כמות רכיבים בעלי דרגות זוגיות בלבד**

```
NumberOfEvenComponents(G)
  counter = 0
  for each v in V
    color[v] = White
  for each v in V
    arr[] = 0;
    isEven = true
    if(color[v] = WHITE)
      counter++
      arr[] = DFSVisitReturnVertices(G,v,Arr)
      for i = 1 to |Arr|
        if (deg(Arr[i])%2==1)
          isEven = FALSE;
      if (isEven)
        counter++
  return counter;
```

אלגוריתם Dijkstra

קלט: רשימת סמיכויות

סיבוכיות: $O(|V| + |E| \cdot \log\ |V|)$

הסבר: דייקסטרא משמש אותנו למציאת המסלול הקל ביותר בין קודקוד אחד לכל השאר כאשר יש משקלים על הצלעות.

אנחנו עוברים על כל קודקוד וצלע לכל היותר פעם אחת ועבור כל שכן מעדכנים את התור (במקרה הגרוע) ועדכון של תור עדיפויות עולה לנו $\log(|V|)$.

פאסודו קוד:

```
Dijkstra(G,s)
      visited = new Array[|V|]
      dist = new Array[|V|]
      father = new Array[|V|]

      for i = 1 to |V|
         visited[i] = false
         father[i] = null
         dist[i] = inf

      dist[s] = 0, visited[s] = true
      PriorityQueue q = new PriorityQueue() // compare by distance
      q.enqueue(s)

      while (q is not empty)
         v = q.extractMin()
         for each n in G[v]
            if(!visited[n])
               if(dist[n] > dist[v] + G[v][n].weight)
                  father[n] = v
               dist[n] = min(dist[n], dist[v] + G[v][n].weight)
               if(q.contains(n))
                  q.decreaseKey(n,dist[n])
               else q.enqueue(n)
         visited[n] = true;
```

לאחר ריצת האלגוריתם, מערך dist מכיל בתא ה-i את העלות הקלה ביותר בין קודקוד s ל-i.

המערך father מכיל את הקודקוד הקודם ל-i, במסלול הטוב ביותר בין s ל-i.

המערך visited הוא true עבור כל מי שניתן להגיע אליו מ-s.

הערות: האלגוריתם לא עובד טוב אם יש משקלים שליליים.

• שחזור מסלול טוב ביותר

כתבו אלגוריתם המשחזר את המסלול הטוב ביותר בין קודקוד s לקודקוד v.

<u>פתרון</u>

נריץ את אלגוריתם דייקסטרא בצורה רגילה ולאחר מכן נפעיל על מערך parent אלגוריתם רקורסיבי שידפיס לנו את המסלול מימנו הוא הגיע.

```
Print_Dijkstra(G,s,v)
        if (s==v)
           print s;
        else if (parent[v]==null)
           print "There is no path between the nodes"
        else
           Print_Dijkstra(G,s,parent[v])
   Print v
```

נתונים 2 קודקודים $s, t \in V$.

יש להחזיר את מספר המסלולים הקצרים ביותר בין s לבין קודקוד t.

באלגוריתם דייקסטרא, אם יש 2 קודקודים u,v שיכולים להגיע לקודקוד w עם אותה עלות נמוכה,

אז כמות המסלולים להגיע לקודקוד w היא כמות המסלולים להגיע לקודקוד u + כמות המסלולים להגיע לקודקוד v.

```
Dijkstra(G,s)
    PriorityQueue q = new PriorityQueue()
    for v = 1 to |V|
    distance[v] = inf
    paths[v] = 0
    q.enqueue(v)

  distance[s] = 0
  paths[s] = 1
  q.enqueue(s)

  while (q is not empty)
    v = q.dequeue()
    for each u in G[v]
      if (distance[v] + G[v,u].distance < distance[u])
        distance[u] = distance[v] + G[v,u].distance
        paths[u]=paths[v]
      else
        if (dist[v]+G[v,u].weight == dist[u])
          paths[u] = paths[u] + paths[v]
  return paths[]
```

שליפה\הכנסה לתור עדיפות: $O(logV)$

אלגוריתם Fire

<u>הסבר על האלגוריתם:</u>

אלגוריתם שריפת עלים הוא אלגוריתם שבהינתן עץ (גרף קשיר ללא מעגלים) יחזיר לנו את המרכז שלו.

הרעיון הוא לקחת את העלים ולמחוק אותם מהעץ בצורה איטרטיבית עד שנישאר עם קודקוד 1 או 2 ואלו יהיו המרכזים שלנו.

1. באמצעות מערך עזר, נרוץ על כל קודקוד ונשאל אותו מה גודל השכנים שלו.
2. כל קודקוד שיש לו רק שכן אחד בלבד הוא עלה.

קלט: מערך שכנויות

סיבוכיות: $O(|E|) = O(|V|)$

בכל עץ יש לפחות 2 עלים ולכן בכל איטרציה יורדים לפחות 2 קודקודים.

כל קודקוד נכנס לרשימת העלים פעם אחת בדיוק. בעץ: $|E| = |V| - 1$

```
Fire(G)
      L = new List()
      for each v in G
        if (G[v].size == 1)
          L.add(v)
      G` = G
      n = |V|
      while (n>2)
        temp = new List()
        for each v in L
          n--
          u = G`[v].get(1)
          G`[u].remove(v)
          if(G`[u].size == 1)
            temp.add(u)
        L = temp
      return L
```

- ## מציאת קוטר ורדיוס

כתבו אלגוריתם למציאת קוטר ורדיוס של עץ

```
find_tree_center(G)
  L = new List()
  for each v in G
    if (G[v].size == 1)
      L.add(v)
  G` = G
  n = |V|
  iterations = 0
  while (n>2)
    iterations++
    temp = new List()
    for each v in L
      n--
      u = G`[v].get(1)
      G`[u].remove(v)
      if(G`[u].size == 1)
        temp.add(u)
    L = temp
  radius = iterations
  num_centers = L.size()
  diameter = radius*2

  if(L.size()==2)
    radius++
    diameter++

  return (L,diameter,radius,num_centers)
```

אלגוריתם Best

<u>מטרה:</u> בהינתן מערך בעל n איברים, מהו תת המערך שסכום איבריו הוא הגדול ביותר?

נתון לנו מערך שיש בו גם מספרים שליליים ואנחנו רוצים למצוא רצף במערך עם הסכום הגדול ביותר.

לדוגמא: [5,-2,6,-3,7,1,-4,4,2,-3,8,5,20].

קלט: מערך של מספרים

סיבוכיות: O(n)

האלגוריתם:

עוברים פעם אחת על המערך וסוכמים:

בכל שלב שומרים את המקסימום עד כה.

אם הסכום יורד מתחת ל-0, מאפסים אותו ומתחילים את הרצף מחדש.

```
Best(A)
    start = 1, end = 1, sum = 0, max = -inf, t_s = 1
    for i = 1 to A.length
        sum = sum + A[i]
        if (sum>max)
            max = sum
            start = t_s
            end = i
        if ( sum < 0)
            sum = 0
            t_s = i + 1
    return (max, start, end)
```

החזר תת מערך <u>מינימלי</u> עם סכום מקסימלי

```
Best(A)
  sum = 0 , max = 0
  start = 1 , end = 1 , ptr = 1, length = -inf

  for i = 1 to |A|
    sum += A[i]
    if (max == sum)
      if (i-ptr + 1 < length)
        length = i - ptr + 1
        start = ptr
        end = i
    if(max<sum)
      max = sum
      start = ptr
      e = i
    if (sum<0)
      ptr = i+1
      sum = 0

  return (max, start, end)
```

Circular Best

כתבו Best עבור מערך מעגלי

פתרון

הרעיון הוא להשתמש ב-Best רגיל.

ניקח את הסכום של כל המערך ונוריד את הרצף הכי מינימאלי שאינו מעגלי ובכך לקבל את הרצף המקסימלי המעגלי.

אבל זה יהיה הגדול ביותר בצורה מעגלית.

יכול להיות שבצורה הסטנדרטית יהיה לנו סכום גדול יותר.

לכן, נצטרך לבצע מקסימום בין התוצאה המעגלית לתוצאה הסטנדרטית.

איך נמצא את התוצאה המעגלית?

- ניתן לשנות את Best שימצא לנו סכום מינימלי
- נשנה את הנתונים, נכפיל אותם במינוס 1 וכך נקבל את המערך ההופכי.

כשנפעיל על מערך זה את Best שוב, נמצא את הסכום המקסימלי במערך ההופכי, שמייצג את הסכום המינימלי במערך המקורי.

Max$(S-(-Best(-A)), Best(A))$

```
cycleBest(A)
    sum = 0
    B = new Array[A.length]
    for i = 1 to A.length
        sum = sum + A[i]
        B[i] = -A[i]
    (m1,s1,e1) = Best(B)
    (m2,s2,e2) = Best(A)
    if (sum - (-m1) < m2)
        return (m2,s2,e2)
    else
        return (sum+m1, e1+1, s1-1)
```

החזר תת מערך מינימלי עם סכום מקסימלי

```
CircularMinBest(A)
  B = new Array[|A|]
  sum = 0
  for i = 0 to |A|
    sum += A[i]
    B[i] = -A[i]
  ( max1,s1,e1 ) = Best(A)
  ( max2,s2,e2 ) = Best(B)

  if (sum - (-max2) < max1)
    return (max1,s1,e1)
```

```
  else if (sum - (-max2) == max1 )
        d1 = e1 - s1 +1;
        d2 = e2 + 1 - s2 + 1 + 1
        if ( d1 < d2)
           return (max1,s1,e1)
  else
     return (sum+max2, e2 + 1, s2 -1 )

Best(A)
  sum = 0 , max = 0
  start = 1 , end = 1 , ptr = 1, length = -inf

  for i = 1 to |A|
     sum += A[i]
     if (max == sum)
        if (i-ptr + 1 < length)
           length = i - ptr + 1
           start = ptr
           end = i
     if(max<sum)
        max = sum
        start = ptr
        e = i
     if (sum<0)
        ptr = i+1
        sum = 0

  return (max, start, end)
```

אלגוריתם תחנות הדלק

<u>תיאור הבעיה:</u> נתון מעגל שסביבו יש תחנות דלק.

בכל תחנה i ניתן למלא A[i] ליטר דלק.

בין התחנה ה-i לתחנה הבאה צורכים B[i] דלק.

<u>מטרה:</u> למצוא האם ניתן להתחיל נסיעה מאחת התחנות עם 0 ליטר דלק בהתחלה ולסיים סיבוב שלם על המעגל.

במידה וכן – מאיזו תחנה.

קלט: מערך של תחנות דלק ומערך של עלויות תואמות.
A[i] – כמה דלק ניתן למלא בתחנה i
B[i] – כמה דלק יש לבזבז מתחנה i לתחנה הבאה

הרעיון:

- קודם כל נוודא שכמות הדלק הכוללת שאפשר למלא, יותר גדולה מהכמות שאנו צורכים (אחרת לא יהיה פתרון לבעיה)
  כלומר $\sum a \leq \sum b$
- ניצור מערך חדש: $C[i] = A[i] - B[i]$
- נפעיל Best Cycle על מערך C ונקבל נקודת התחלה אופטימלית (הרצף שייתן את כמות הדלק הגבוהה ביותר בנסיעה באותו הרצף)
- נתחיל מהנקודה שחזרה (נקודת התחלה) ונבדוק האם ניתן לסיים את הסיבוב.
  אם כן – זאת הנקודה.
  אם לא – לא ניתן לסיים סיבוב שלם.

סיבוכיות: O(n) כאשר n הוא גודל 2 המערכים (מספר תחנות הדלק).

תת המטריצה הגדולה ביותר (Super Best)

מציאת תת מטריצה בעלת סכום מקסימלי.

בהינתן מטריצה בגודל **m*n**, מה היא תת המטריצה שסכום איבריה הוא הגדול ביותר?

הערה: האם כדאי לרוץ על השורות ולסכום עמודות או לרוץ על העמודות ולסכום את השורות?

תשובה: זה תלוי בקלט. אם המטריצה לאורך, נרוץ על השורות ואם היא לרוחב נרוץ על עמודות.

אלגוריתם:

- עוברים על כל רצף השורות (או עמודות – המינימאלי מבניהם) וסוכמים הכל למערך אחד.
  (אם עוברים על השורות, אז כל תא במערך החדש הוא סכום אותה עמודה בכל השורות ולהיפך)
- מפעילים Best רגיל על המערך ומקבלים את הערכים – ושומרים את המקסימום.
- לבסוף מחזירים את הערך הגדול ביותר.
- כדי לחשב את הסכומים עבור מערך העזר באופן יעיל, ניצור לפני תחילת האלגוריתם מטריצת עזר **h** שבתא **[i,j]** יהיה שמור הסכום של עמודה **j** משורה 1 ועד שורה **i**.

סיבוכיות:

- יצירת מטריצת H של הסכומים המצטברים:$O(nm)$
- עוברים על כל הקומבינציות משורה x עד שורה y : $O(n^2 \cdot (m+m)) \to O(n^2)$
- ממלאים את מערך העזר בגודל **m** באמצעות מטריצת הסכומים $O(m)$
- לבסוף מפעילים Best רגיל על אותו מערך:$O(m)$

```
SuperBest(M)
  n - M-rows
  m - M-cols
  max = -inf
  row_s, row_e, col_s, col_e
  H = new Matrix[n][m]

  for i = 0 to m-1
    H[0][i] = M[0][i]

  for i = 1 to n-1
    for j = 0 to m-1
      H[i][j] = H[i-1][j] + M[i][j]

  for i = 0 to n-1
    for j = i to n-1
      A = new Array[m]
      for k = 0 to m-1
        if(i == 0) A[k] = H[j][k]
```

```
        else A[k] = H[j][k] - H[i-1][k]
      (sum, s, e) = Best(A)
      if(sum > max)
        max = sum
        row_s = i , row_e = j, col_s = s, col_e = e
return (max, row_s, row_e, col_s, col_e)
```

עצים פורסים מינימליים : Prim, Boruvka, Kruskal, RKruskal

נתון לנו גרף שיש לו משקלים על הצלעות.

המטרה: למצוא תת גרף שהוא עץ (קשיר ללא מעגלים) שסכום משקלי הצלעות שעליו הוא מינימאלי.

Kruskal Algorithm

- מיין את הצלעות מהצלע עם המשקל הנמוך ביותר לגבוהה ביותר.
- עבור על הצלעות מהנמוך לגבוה וכל צלע שלא סוגרת מעגל עם מה שכבר לקחנו לעץ לפני כן – הוסף לעץ.
- סיים כאשר לקחת בדיוק n-1 צלעות (n הוא כמות הקודקודים)

סיבוכיות:

- $O(|E|\log|E|)$ עבור המיון.
- מעבר על הצלעות מהקטנה לגדולה $O(|E|)$
- בכל מעבר צריך לבדוק שלא סוגרים מעגל.
- שומרים את הקודקודים שהם באותו רכיב קשירות בתוך מבנה נתונים union-find (איחוד קבוצות זרות)

סיבוכיות פעולות שימוש במבנה נתונים זה הן $O(\log|V|)$

סה"כ סיבוכיות: $O(|E|\log|E|)$

דוגמת הרצה: לחץ כאן

הערה : פונקציית union-find מאחדת ומחזירה האם הצלחנו לאחד (כלומר, לבדוק שהצלעות לא באותה קבוצה)

```
Kruskal(G)
    DisjointSets d = new DisjointSets(|V|)
    Tree T = {}
    sort(|E|) //by weight
    for i = 1 to |E|
        e = E[i]
        if (d.union(e.v1,e.v2))
            T.add(e)
        if (|T| == n-1 ) return T
    return T
```

- ### החזר עץ פורס מקסימלי

```
MaxKruskal(G=(V,E))
  sort(E) //sort edges from smallest to biggest
  reverse(E) //from biggest to smallest
  DisjointSets d = new DisjointSets(|V|)
  Tree t ={}

  for i = 1 to |E|
```

```
    e = E[i]
    if (d.union(e.v1,e.v2))
      t.add(e)
    if (|T| == n-1 )
      return T
  return T

MaxKruskal(G=(E,V))
  for each e in E
    e.weight = -e.weight
  T = Kruskal(G) //regular kruskal
  for each e in T[E]
    e.weight = -e.weight
  return T
```

Reverse Kruskal

הרעיון של אלגוריתם רוורס קרוסקל הוא למחוק צלעות כל עוד הגרף לא נשאר קשיר.

1. נמיין את הצלעות מראש לפי סדר יורד של משקלים
2. נבחר בכל פעם את הצלע עם המשקל הגדול ביותר
3. אם צלע זאת היא "גשר" (מנתקת את הגרף ליותר רכיבי קשירות) אז נמחק אותה
4. התהליך הזה הוא סופי וייעצר אחרי שיישארו V-1 צלעות בגרף.

```
ReverseKruskal(G)
  SortE(G)
  Queue q = new Queue()
  for each e in E(G)
    q.enqueue(e)
  size = |E|

  while (size > |V|-1 )
    e = q.dequeue()
    if (isBridge(G,e)=false)
      remove(G,e)
      size--;
  return G
 isBridge(G,e)
    G` = G-{e}
    if (numOfConnectedComponents(G`)==2)
      return true
    else
      return false
DFS(G)
  counter = 0
  for each v in V
    color[v] = WHITE
  for each v in V
    if color[v]==WHITE
      REC_DFS(G,v)
      counter++
  return counter
REC_DFS(G,v)
  color[v] = gray
  for each u in G[v]
    if color[u] == WHITE
      REC_DFS(G,u)
  color[v] = BLACK
```

Prim's Algorithm

- האלגוריתם של פרים הוא אלגוריתם חמדני המשמש למציאת עץ פורש מינימאלי בגרף ממושקל לא-מכוון.
- האלגוריתם מתחיל את בניית העץ מקודקוד פתיחה שנבחר באקראי
- בכל צעד האלגוריתם, נוסיף לעץ את הצלע בעלת המשקל המינימאלי מבין אלה היוצאות מקודקודי העץ ולא סוגרות מעגל.

איך נוודא שלא נסגור מעגל?

כאשר נרצה להוסיף צלע לעץ, נוודא שרק אחד מהקודקודים קיים בעץ.

איך נשמור את העץ?

בעזרת מערך אבות

איך נדע את המשקל המינימלי של העץ הפורש?

נשתמש במערך משקלים שבכל תא נשמור את המשקל המינימלי שיצא מאותו קודקוד בעץ שנבחר

דרך פעולה:

1. מגדירים תור עדיפויות ומערך visited[|V|] עבור הקודקודים.
2. מתחילים מקודקוד שרירותי s על הגרף (s נכנס לתור עם עדיפות 0)
3. מכניסים לתור עדיפויות את כל השכנים שלו עם עדיפות משקל ביניהם.
4. כל עוד התור לא ריק, שולפים את הקודקוד עם העדיפות המינימאלית, מוסיפים את הצלע לעץ (בין הקודקוד הנשלף לאבא שלו)
5. מכניסים לתור את כל השכנים שלא סיימנו איתם עם עדיפות המשקל שמגיע אליהם מאותו קודקוד (אם הם כבר בתוך התור, אז רק מעדכנים את העדיפות למינימאלית)

דוגמת הרצה: לחץ כאן

```
Prim(G=(V,E))
    PriorityQueue q = new PriorityQueue()
    visited = new Array[|V|]
    parent = new Array[|V|]
    distance = new Array[|V|]

    for i = 1 to |V|
      visited[i] = false
      parent[i] = null
      distance[i] = inf
    distance[1] = 0
    q.enqueue((1,0))

    while(q is not empty)
      v = q.extractMin()
      if (father[v]!=null) T.add((v,father[v]))
      for each u in G[v]
        if (!visited[u] && distance[u] > E[v][u].weight)
          distance[u] = E[v][u].weight
          father[u] = v
          if (q.contains(u)) q.decreaseKey((u,distance[u]))
          else q.enqueue(u,distance[u])
      visited[v] = true
    return T
```

סיבוכיות:

- אתחול מערכים $O(|V|)$
- לולאת ראשית עוברת על כל קודקוד פעם אחת ואז על השכנים שלו $O(|E|)$
- בכל איטרציה, מכניסים ומעדכנים בתור העדיפות (במקרה הגרוע): $O(\log|V|)$

סיבוכיות כוללת: $O(|V| + |E| \log |V|)$.

סה"כ: $O(|E| \log|V|)$

Boruvka's Algorithm

אלגוריתם Boruvka הוא אלגוריתם חמדני המשמש למציאת עץ פורש מינימלי בגרף G קשיר, משוקלל לא מכוון.
האלגוריתם מתחיל את בניית העץ מבניית יער, שכל עץ בו מורכב מקודקוד אחד, כלומר היער מורכב מ- $n = |V|$ עצים.
צעד האלגוריתם מוסיף לכל עץ את הצלע בעלת המשקל המינימלי (צלע בטוחה) מבין אלה היוצאות מקודקוד.

דרך פעולה:

1. בונים יער, המורכב מ-*n* עצים. כל עץ מורכב מקודקוד אחד ועדיין לא מכיל אף צלע.
כל קודקוד כזה הוא למעשה שורש והנציג של העץ.
2. איטרציה ראשונה: לכל עץ מוסיפים צלע בטוחה הסמוכה לשורש.
ייתכן שאת אותה הצלע נוסיף לשורשים שונים, לכן האלגוריתם מונע את המצב וכל צלע נכנסת לעץ פעם אחת בלבד.
3. איטרציה שנייה: מחברים עצים (*union*) על ידי צלע בעזרת משקל מינימלי.
4. כל עוד מספר העצים גדול מ-1, חוזרים לשלב מספר 3.

דוגמת הרצה: לחץ כאן

```
Boruvka's(G=(V,E))
        DisjointSets d = new DisjointSets(|V|)
        Tree t = {}
        isDone = false

        while (!isDone)
        {
           cheapest = new Array[|V|] //init to null
           for i = 1 to |E|
              e = E[i]
              g1 = d.find(e.v1), g2 = d.find(e.v2)
              if (g1 != g2)
                 if (e.weight < cheapest[g1].weight) cheapest[g1] = e
                 if (e.weight < cheapest[g2].weight) cheapest[g2] = e
           isDone = true
           for i = 1 to |V|
              if (cheapest[i] != null)
                 T.add(cheapest[i])
                 d.union(cheapest[i].v1, cheapest[i].v2)
                 isDone = false;
        }
```

סיבוכיות:

- בכל שלב עוברים על כל הצלעות
- לאחר כל שלב, כמות הרכיבים מצטמצמת בפי 2 לפחות

סיבוכיות סה"כ: $O(|E|\log|V|)$

אלגוריתם קידוד Hoffman

נתון מערך של תווים שלכל תו יש תדירות מופעים בטקסט.

המטרה היא לקודד כל תו לרצף ביטים כך שהרצף הכולל יהיה קצר ביותר (ככה נחסוך מקום)

לדוגמא:

טקסט: aabcbbdcaabaa

אז: התדירויות הן: 6=a#, 4=b#, 2=c#, 1=d#.

אם נבחר את הקידוד: a=0, b=101, c=100, d=11.

אז קידוד הטקסט כולו יהיה: 0010110010110111000010100.

- השימוש הוא בעיקר לכיווץ קבצים.
- הרעיון הכללי הוא לתת קידוד <u>כמה שיותר קצר</u> לתו שמופיע הכי הרבה פעמים.

<u>כיצד האלגוריתם עובד?</u>

1. הכנס את כל התדירויות לערימת מינימום.
2. בכל שלב, שלוף את 2 הערכים הקטנים ביותר.
3. בנה מהם עץ בינארי כך ש-2 הערכים הם בנים (ימני ושמאלי) ולהם אב אחד משותף שערכו הוא סכום 2 הערכים.
4. הכנס את האבא לערימה במקומם.
5. חזור על התהליך עד שנשאר רק איבר אחד בערימה והוא שורש העץ.

סיבוכיות עבור מערך לא ממוין: $O(nlog(n))$

סיבוכיות עבור מערך ממוין: $O(n)$

ניתן לייעל את הסיבוכיות אם נתון לנו מערך ממוין, לפי התדירויות.

במקום להשתמש בערימה, נשתמש ב-2 תורים רגילים:

1. נכניס את כולם לתור הראשון
2. בכל שלב נשלוף את ה-2 הנמוכים ביותר (שהם בראשי התור)
3. נמזג אותם לאבא משותף אחד
4. נכניס את האבא לתור השני

* לאורך כל האלגוריתם, 2 התורים ממויינים ולכן 2 הקטנים ביותר יהיו 2 הראשונים בתור הראשון או 2 הראשונים בתור השני, או אחד מהראשון ואחד מהשני.

```
Huffman(A) // A.char || A.freq
  Sort(A) // by A.frequency
  Queue q1 = new Queue()
  Queue q2 = new Queue()

  for i = 1 to |A|
    q1.enqueue(New Node(A[i]))
  while(q1.size() + q2.size() > 1)
    x = getMin(q1,q2)
    y = getMin(q1,q2)
    z = new Node(x.data.f + y.data.f) //creating new father node
```

```
    q2.enqueue(z)
  if(q1 is empty) return q2.dequeue()
  else return q1.dequeue()

  getMin(Queue q1, Queue q2)
    if (q1 is empty) return q2.dequeue()
    else if (q2 is empty) return q1.dequeue()
    else if (q1.head().data.f < q2.head.data.f) return q1.dequeue()
    else return q2.dequeue()
```

החזרת קידוד

כתוב אלגוריתם שמחזיר את הקידוד שהתקבל עבור תו כלשהו

```
getHuffmanCode(A)
  root = Huffman(A)
  HuffmanCode(root)

HuffmanCode(root)
  if(root.left is NULL && root.right is NULL)
    print(root.char)
  else
    print("0" + HuffmanCode(root.left))
    print("1" + HuffmanCode(root.right))
```

מסלול ומעגל אוילר

<u>מושגים</u>

נתון: G(E,V) גרף לא מכוון.

- **מסלול אוילר:**

  מסלול P(x,y) נקרא מסלול אוילר ב-G אם:

  1. הוא עובר בכל הצלעות של G
  2. כל צלע מופיעה בו <u>פעם אחת בלבד</u>
  3. $x \neq y$

- **מעגל אוילר:**

  מעגל אוילר ב-G הוא מסלול אוילר סגור.

  1. המסלול עובר בכל צלעות הגרף פעם אחת בלבד
  2. הקודקוד ההתחלתי הוא גם קודקוד הסיום

- **גרף אוילריאני:**

  גרף המכיל מעגל אוילר.

<u>משפטי זיהוי אוילר בגרפים:</u>

1. יש בגרף G מעגל אוילר (גרף אוילריאני) אם ורק אם G קשיר וכל דרגות הגרף זוגיות
2. יש בגרף G מסלול אוילר אם ורק אם G קשיר ובדיוק 2 קודקודים בעל דרגות אי זוגיות.

המטרה: בהינתן גרף, להחליט האם יש מסלול או מעגל ואז להחזיר אותו.

איך האלגוריתם עובד?

1. בדיקה האם יש מעגל

   עוברים על כל קודקוד ובודקים האם דרגתו זוגית, סופרים את אלו שהם עם דרגה אי-זוגית.

   אם יש לנו 0 אי-זוגיים: קיים מעגל.

   אם יש 2 אי-זוגיים: קיים מסלול.

   אחרת אין מעגל ואין מסלול.
2. מגדירים מחסנית ורשימה המייצגת את המסלול עצמו.

   <u>אם זה מסלול:</u> מתחילים מקודקוד מדרגה אי-זוגית.

   <u>אם זה מעגל:</u> בוחרים קודקוד בצורה שרירותית.

   מכניסים את הקודקוד הנבחר למחסנית.
3. אם יש לקודקוד שכן, מכניסים אותו למחסנית.
4. מוחקים את הצלע וממשיכים עם השכן.
5. נעשה זאת עד שיגמרו השכנים.
6. ברגע שנתקעים, מוציאים את ראש המחסנית ומוסיפים למסלול.
7. ממשיכים עם הקודקוד הבא עד שהמחסנית מתרוקנת.

סיבוכיות:

בכל איטרציה יורדת צלע ולכן יהיה לכל היותר $O(|E|)$ איטרציות כאשר בכל איטרציה זה $O(1)$.

*הערה: זה כולל מימוש נכון של מחיקת הצלע בין u ל-v ב-2 הכיוונים.

```
EulerPathCycle(G=(V,E))
  Stack s = new Stack()
```

```
List path = new List()
count = 0;
start = 1

for each v in V
   if (G[v].size()%2==1)
      count++
      start = v
if (count > 2) return "No Path or Cycle"
s.push(start)
while ( s is not empty )
   v = s.top()
   if (G[v].size()>0)
      u = G[v].getFirst()
      s.push(u)
      G.remove(v,u) //remove edge
   else
      path.add(s.pop())
return path
```

• האם ניתן להפוך גרף קשיר לגרף אוילריאני ע"י הוספת צלעות?

האם ניתן להפוך גרף $G=(V,E)$ לגרף אוילריאני על ידי הוספת צלעות?

כדי שגרף יהיה אוילריאני הוא צריך להיות קשיר[1] ו[2]כל דרגות הקודקודים זוגיות.
נתון שהגרף כבר קשיר. אם הגרף כבר אוילריאני, אז אין צורך להוסיף צלעות.
אחרת, ישנם דרגות אי-זוגיות.

משפט: כמות הקודקודים בעלי דרגה אי-זוגית הוא זוגי.
הוכחה: לא ייתכן שבגרף יש מספר אי-זוגי של קודקודים בדרגה אי-זוגית, כי <u>סכום כל הדרגות שווה לפעמיים מספר הצלעות</u>, כלומר סכום הדרגות הוא מספר זוגי.

אם כך, ניתן לקחת כל זוג קודקודים בעלי דרגה אי-זוגית ולחבר ביניהם בצלע ולהפוך לאוילריאני.

• האם ניתן להפוך גרף לגרף אוילריאני ע"י הוספת צלעות? (לא בהכרח קשיר)

אם גרף קשיר ניתן לקחת כל זוג קודקודים בעלי דרגה אי זוגית ולחבר ביניהם בצלע ולהפוך לאוילריאני.
ואז נחזיר תשובה לפי $\frac{amount\ of\ odd\ vertices}{2}$

<u>אחרת</u>
נחלק את הגרף ל-k רכיבי קשירות ונוסיף k צלעות שיצרו מעגל בין רכיבי הקשירות.
לכל רכיב יהיה צלע נכנסת וצלע יוצאת (כדי לחבר בין שאר רכיבי הקשירות)
נחלק:

1. לרכיבי קשירות שדרגותיהן זוגיות: הצלע הנכנסת והצלע היוצאת יהיו לאותו קודקוד (ככה נשמור על דרגה זוגית)
2. לרכיבי קשירות שיש 2 קודקודים בעלי דרגה אי זוגית: הצלע הנכנסת תגיע לקודקוד בעל דרגה אי-זוגית, והצלע היוצאת תצא מקודקוד אחר בעל דרגה אי זוגית.
3. עכשיו הגענו לרכיב קשירות אחד והגרף קשיר, יהיה ניתן להשתמש בנוסחה הקודמת

לכן כמות הצלעות שנצטרך להוסיף תהיה כמות הרכיבים הזוגיים (חיבור בין הרכיבים הזוגיים לרכיבים האחרים) + כמות הקודקודים בעלי דרגה אי זוגית :

$$\#even\ components + \frac{amount\ of\ odd\ vertices}{2}$$

כדי למצוא את כמות הרכיבים הזוגיים, נפעיל DFS המוצא את כל רכיבי הקשירות ונבדוק בכל רכיב האם הוא זוגי או לא.

סכום המטריצה הגדול ביותר

נתונה מטריצה שמלאה ב-1 וב-(1-) בצורה רנדומלית.

איך נהפוך את המטריצה לבעלת הסכום הגדול ביותר?

ניקח כל שורה וכל עמודה ונכפיל ב-(1-)

תהליך זה הוא סופי. מדוע?

יש לנו חסם עליון, כלומר לא יכול להיות שהסכום יהיה גדול יותר ממספר התאים במטריצה.

שאלת מחשבה: האם סדר ההכפלות של השורות והעמודות משנה?

האם זה יוריד את מספר ההכפלות?

בניית עץ מרשימת דרגות

<u>הגדרות</u>

1. בכל עץ: $|E| = |V| - 1$
2. כלומר, סכום הדרגות של כל הקודקודים בעץ שווה לפעמיים הצלעות
3. נובע מכך שסכום הדרגות של כל הקודקודים בעץ $= 2(|V| - 1)$

<u>מתי רשימת דרגות יכולה להיות עץ?</u>

1. כאשר סכום דרגות הקודקודים $= 2(|E|)$
2. כאשר יש לנו מספיק נתונים לכמות הקודקודים

<u>רעיון האלגוריתם</u>

1. משפט: בכל עץ יש לפחות 2 עלים.
2. ננסה לחבר עלה לקודקוד שהדרגה שלו גדולה מ-1
3. ניקח קודקוד ראשון שאינו עלה ונחבר אותו עם הקודקוד הראשון במערך שהוא לא עלה.
4. לאחר החיבור, נוריד את דרגות קודקודים אלו.
5. נמשיך ככה עד שכל הערכים במערך יהיו 0.

```
BuildTreeFromDegreesArray(D)
  N = |D|
  sum = 0
  tree[N] = null

  for i = 1 to N
    sum + = D[i]
  if ((sum/2)+1 != N)
    print "Not a tree degrees array"
    return
  sort(D)
  j = first index so that D[j] > 1
  for i = 1 to N-2
    tree[i] = j
    D[j]--
    if (D[j]==1)
      j++
  tree[N-1] = N
  return tree
```

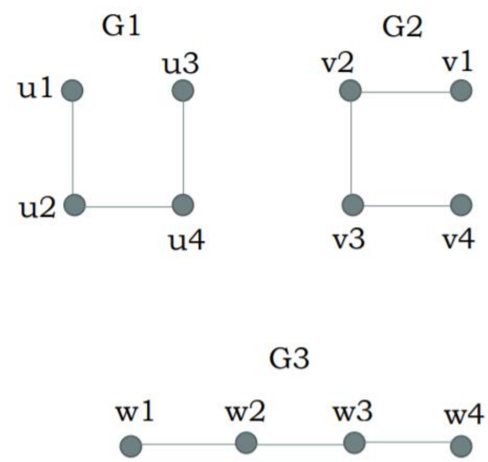
עצים איזומורפיים – חסר קוד

<u>עץ איזומורפי:</u>

גרפים H ו-G הם איזומורפיים אם קיימת פונקציה $f: V(G) \to V(H)$ חח"ע ועל כך שעבור כל $u, v\epsilon V(G)$ מספר הקשתות המקשרות בין u ל-v זהה למספר הקשתות בין $f(u)$ ו-$f(v)$.

במילים אחרות, הם מייצגים את אותו גרף אבל נראים בצורה שונה.

<u>לדוגמא:</u>



- <u>עצים איזומורפיים עם שורש</u>

הרעיון שעומד מאחורי האלגוריתם הוא שצריך להחליט על סדר בין הבנים של כל קודקוד וכך נוכל להשוות ביניהם.

כלומר, ננסה שהבנים יהיו ממויינים בסדר כלשהו וכך נוכל להכריע בבעיית האיזומורפיים.

אלגוריתם ליצירת מחרוזת המתארת עץ מושרש:

1. נסרוק את הגרף לעומק
2. כשנגיע לעלה, נגדיר את המחרוזת שלו כ-01
3. כאשר נגדיר את המחרוזת של האבא (נגדיר אותה רק אחרי שנסיים עם כל המחרוזות של הבנים שלו) נתחיל אותה ב-0, נסיים אותה ב-1 והאמצע יהיה שרשור ממוין של מחרוזות הבנים.

בשביל להשוות בין עצים ולראות האם הם איזומורפיים, נפעיל את האלגוריתם על כל אחד מהעצים ונראה האם נקבל מחרוזות שוות.

- <u>עצים איזומורפיים בלי שורש</u>

1. נפעיל על העץ את אלגוריתם שריפה שלנו
2. נחלץ את המרכז ונפעיל עליו את האלגוריתם הרגיל של איזומורפיזם

<u>שאלת מחשבה:</u>

מה קורה אם יש יותר ממרכז אחד?

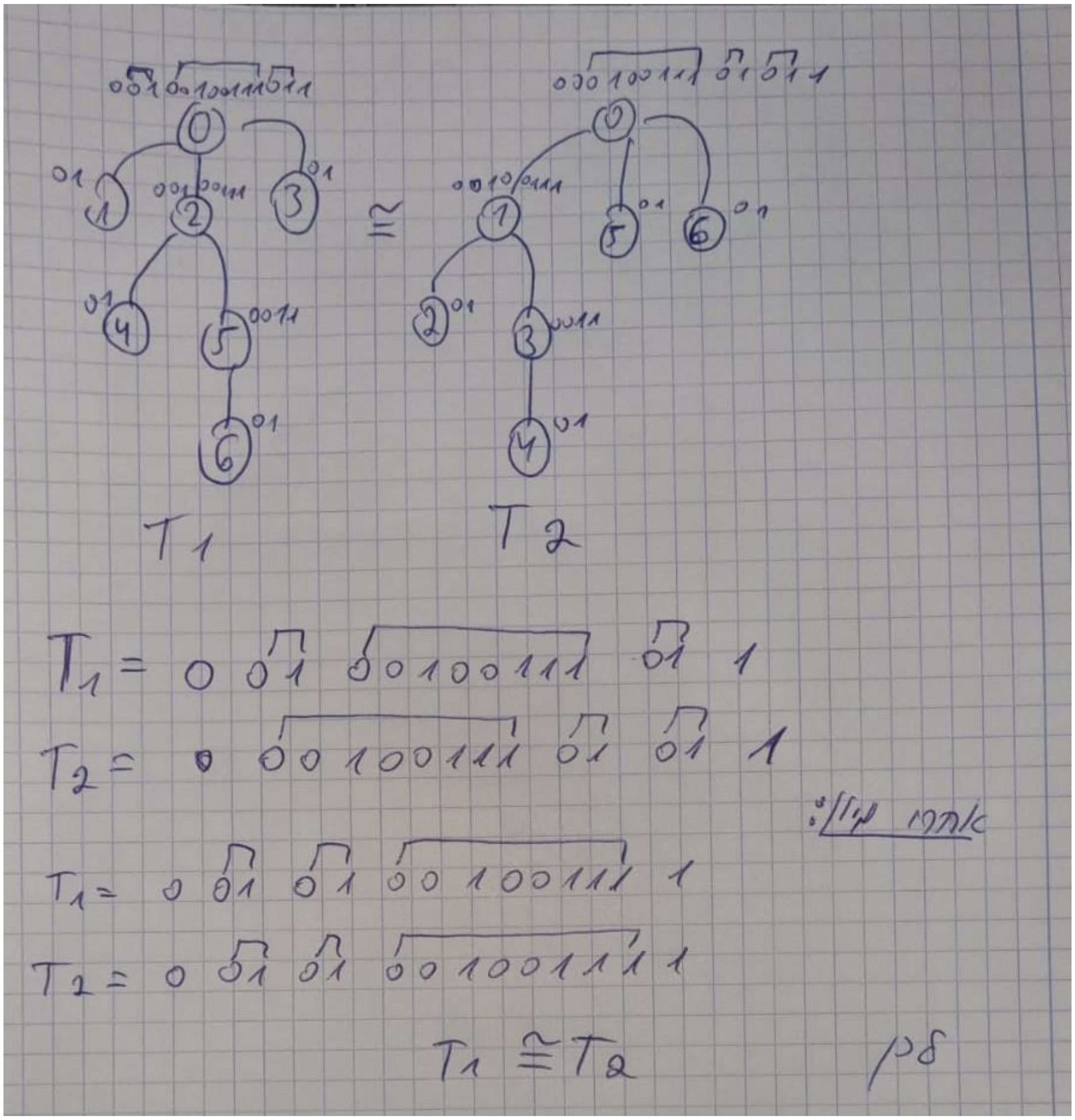
האם עץ עם מרכז אחד יכול להיות איזומורפי לעץ עם שני מרכזים? לא.

האם 2 עצים עם 2 מרכזים יכולים להיות איזומורפיים? כן.

במקרה הזה, נבחר את אחד מהמרכזים של עץ X, נפעיל עליו את האלגוריתם שלנו.

בעץ Y נריץ את האלגוריתם על שני המרכזים.

אם התוצאות יהיו זהות עבור כל המקרים, רק אז נוכל להגיד שהעץ איזומורפי.


T1
T2
$T_1 = 0\ 01\ 00100111\ 01\ 1$
$T_2 = 0\ 00100111\ 01\ 01\ 1$
אחרי מיון:
$T_1 = 0\ 01\ 01\ 00100111\ 1$
$T_2 = 0\ 01\ 01\ 00100111\ 1$
$T_1 \cong T_2$

בעיית השוקולד – להשלים

חפיסת שוקולד מוגדרת כקטע $[1, n]$ של המספרים הטבעיים.

בכל שלב חותכים את הקטע על ידי האינדקס $i\epsilon[1, n-1]$ לשני תתי קטעים: $[1, i]$ $[i+1, n]$.

העלות של חיתוך היא $i \cdot (n-i)$.

השלב האחרון הוא המצב בו כל הקטעים הם בגודל 1.

פונקציית המטרה – סכום כל החיתוכים $n \cdot \frac{(n-1)}{2}$

המטרה היא למצוא רצף של חתכים שמקטין את פונקציית המטרה.

בעיית הבקבוקים

נתונים לנו 2 בקבוקים, הראשון מכיל a ליטרים והשני מכיל b ליטרים.

פעולות מותרות על הבקבוקים:

1. מילוי מיכל A - (m, b)
2. מילוי מיכל B – (a, n)
3. ריקון מיכל A – $(0, b)$
4. ריקון מיכל B – $(a, 0)$
5. מזיגה מ-A ל-B: $(a + b - \min(n, a + b)\, , \min(n, a + b)$
6. מזיגה מ-B ל-A: $(\min(a + b, m)\, , a + b - \min(a + b, m))$

הבעיה:

האם ניתן להגיע למצב <u>בו בבקבוק הראשון יש בדיוק x</u> ליטרים של מים <u>ובבקבוק השני בדיוק y</u> ליטרים של מים?

$x \le a, y \le b$.

ניתן להמיר את הבעיה לגרף שבו כל קודקוד מתאר מצב (x,y) של 2 הבקבוקים.

כלומר, יהיו בגרף $(a + 1)(b + 1)$ קודקודים.

תהיה צלע מכוונת בין 2 קודקודים אם ניתן להגיע מהראשון לשני ע"י אחת הפעולות המותרות.

המטרה היא למצוא מסלול בין המצב ההתחלתי (0,0) למצב המבוקש (x,y).

אם יש מסלול – אז ניתן לבצע את הפעולות שמביאות אותנו למצב הדרוש וגם נוכל למצוא את המסלול הקצר ביותר שעושה זאת.

<u>כיצד נממש?</u>

1. נבנה מטריצה בוליאנית ריבועית בגודל $(n + 1)(m + 1)$ המכילה false,
   כאשר כל עמודה וכל שורה מייצגת את הקשר שבין 2 קודקודים. אם יש קשר, המיקום במטריצה יסומן כ-true.
   נשאיר את האלכסון להיות false (קשר בין קודקוד לבין עצמו).
2. כל קודקוד מיוצג על ידי 2 ערכים (ערך לכל בקבוק), לכן היה צורך להשתמש כאן במטריצה תלת ממדית.
   אך דבר כזה היה מאוד קשה ומבלבל ולכן נשתמש בפונקציה שתדע לקשר אותנו בין בקבוק לבין המיקום שלו במטריצה.
   הנוסחא היא: $(m + 1) \cdot i + j$.
   - i מייצג את גובה הנוזל במיכל A
   - j מייצג את גובה הנוזל במיכל B
   - m מייצג את הגודל המירבי של אחד המיכלים

```
BottleProblem(b1,b2)
  n = (b1+1)*(b2+1) //num of V
  M = new matrix[n][n]
  max = max(b1,b2)

  for i = 1 to b1
    for j = 1 to b2
      index = getIndex(max,i,j)

      mat[index,getIndex(max,i,0)] = 1;
```

```
        mat[index,getIndex(max,0,j)] = 1

        mat[index,getIndex(max,b1,j)] = 1
        mat[index,getIndex(max,i,b2)] = 1

        mat[index, getIndex(max, i+j -min(i+j,b2),min(i+j,b2))] = 1;
        mat[index, getIndex(max, min(i+j,b1),i+j -min(i+j,b1))] = 1;

for i = 1 to n
    mat[i,i] = 0
end-bottle-problem

getIndex(max,i,j)
    return (max+1)*i+j;
```

החזר רשימת דרגות ממויינת של גרף הבקבוקים

1. **כיצד מחשבים כמה קודקודים יש בגרף בעיית הבקבוקים?**

$$(b1+1)\cdot(b2+1)$$

```
degreesOfVerticesInBottleProblem(b1,b2)
  vertices = (b1+1)(b2+1)
  degrees = new Array [vertices]
  max = max(b1,b2)

  for k = 1 to vertices
    i = getI(k,max)
    j = getJ(k,max)
    //empty first bottle
    index = getIndex(max,0,j)
    if k != index
      degrees[k] ++
      degrees[index]++
    index = getIndex(max,i,0)
    if k != index
      degrees[k]++
      degrees[index]++
    index = getIndex(max,b1,j)
    if k != index
      degrees[k]++
      degrees[index]++
    index = getIndex(max,i,b2)
    if k != index
      degrees[k]++
      degrees[index]++
    index = getIndex(max,i+j - min(b2,i+j),min(b2,i+j))
    if k != index
      degrees[k]++
      degrees[index]++
    index = getIndex(max, min(b1,i+j), i+j - min(b1,i+j))
    if k != index
      degrees[k]++
      degrees[index]++
  sort(degrees)
  return degrees
```

<u>פתרון נוסף באמצעות מטריצה</u>

```
degrees_WaterProblem(b1,b2)
  vertices = (b1+1)*(b+2)
  Matrix = new Matrix[vertices][vertices]
  max = max(b1,b2)

  for i = 1 to b1
    for j = 1 to b2
      index = getIndex(max,i,j)

      matrix[index][getIndex(max,i,0) = 1
      matrix[index][getIndex(max,0,j) = 1
      matrix[index][getIndex(max,i,b2) = 1
      matrix[index][getIndex(max,b1,j) = 1
      matrix[index][getIndex(max,i+j - min(i+j,b2),min(i+j,b2)) = 1
      matrix[index][getIndex(max,min(i+j,b1),i+j - min(i+j,b1)) = 1

  for i = 1 to max
    matrix[i][i] = 0

  degrees = new Array[vertices]
  for i = 1 to M
    for j = 1 to M
      if (matrix[i][j] == 1)
        degrees[i]++
        degrees[j]++
  sort(degrees)
  return degrees
```