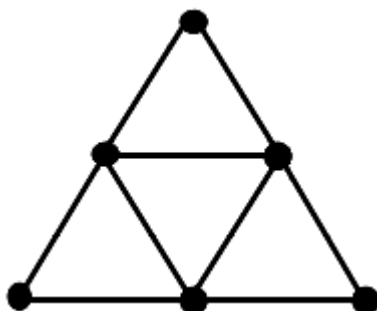
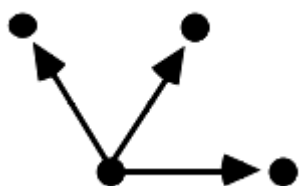


חידה לחימום

נתון פיגום משולש של מוטות המחברים קודקודים ויוצרים קומות של משולשים קטנים, כמודגם באיור הבא, בו מתואר פיגום משולש בן שתי קומות:



משימתו של פועל העובד בפיגום היא להתקדם מן הקודקוד השמאלי התחתון אל הקודקוד העליון. בכל צעד שלו יכול הפועל להתקדם לאורך מוט המחבר קודקוד בו הוא נמצא אל קודקוד סמוך. כיווני



התנועה המותרים הם: ימינה, חצי-ימינה ולמעלה, וחצי-שמאלה ולמעלה – כמתואר באיור. אסור לנוע למטה ואסור לנוע שמאלה.

כתבו תכנית שהקלט שלה הוא מספר שלם n , המבטא את מספר הקומות בפיגום ($2 \leq n \leq 20$), והפלט שלה הוא מספר המסלולים השונים האפשריים להתקדמות הפועל מן הקודקוד השמאלי התחתון אל הקודקוד העליון.

מסלול פירושו רצף של מוטות סמוכים. שני מסלולים נקראים **שונים** אם הרצפים שמהם הם מורכבים אינם זהים. נשים לב שייתכנו מסלולים באורכים שונים.

למשל: עבור הקלט 2 (שפירושו הפיגום שבדוגמא), יהיה הפלט 6.

מבני נתונים ויעילות אלגוריתמים

(21.1.2015)

1. עצי AVL

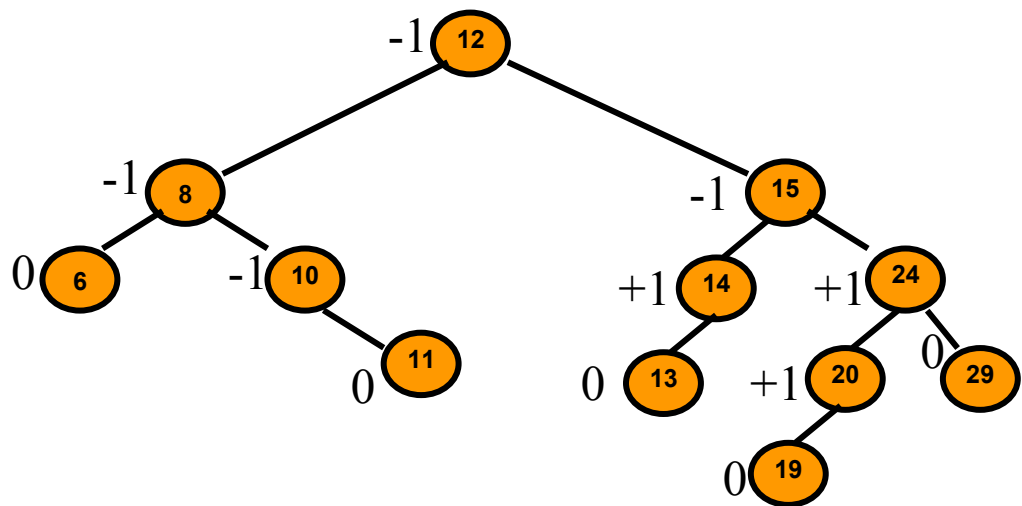
עבור עצי חיפוש בינארי, ראינו שפעולות עליהן עשויות להתבצע בזמן לינארי (אם העץ בלתי מאוזן, ונראה כמו "שרוך"), או בזמן לוגריתמי (אם העץ מאוזן).

אולם – לא הגדרנו באופן פורמלי מהו עץ 'מאוזן'. ישנן כמה דרכים להגדיר איזון של עץ. נכיר אחת מהן, שנקראת עץ AVL (על שם שני מתמטיקאים סובייטיים, Evgenii Landis ו-Georgy Adelson-Velskii, שפיתחו שיטה זו ב-1962).

עבור כל צומת v בעץ בינארי, נסמן ב- $h_L(v)$ את גובה תת-העץ השמאלי של v , וב- $h_R(v)$ את גובה תת-העץ הימני של v . כמו כן נגדיר את גורם האיזון (Balance Factor) של צומת v בתור ההפרש: $BF(v) = h_L(v) - h_R(v)$.

עץ AVL הוא עץ חיפוש בינארי המקיים את תכונת האיזון הבאה: $-1 \leq BF(v) \leq 1$. כלומר, גורם האיזון הוא אחד מבין הערכים: $-1, 0, +1$.

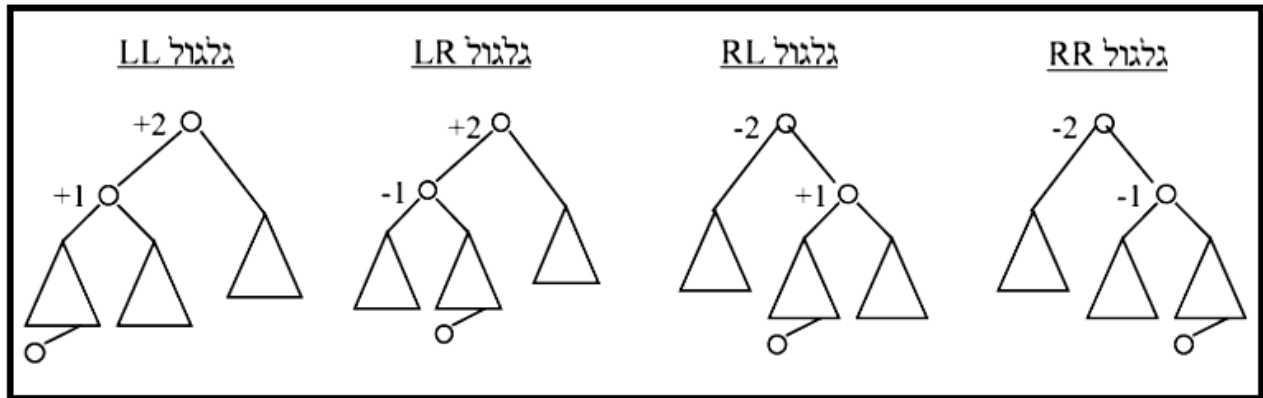
לדוגמה: נתון עץ AVL, כאשר לצד כל צומת כתוב גורם האיזון שלו.



כשמוסיפים או מוציאים צומת מעץ AVL, אז לעיתים גורם האיזון מופר. למשל, אם נוסיף את 18 לעץ (הוא מתווסף כבנו השמאלי של 19), אז גורם האיזון של 20 הופך להיות +2, וזה לא תקין. כאשר גורם האיזון הופך להיות +2 או -2, צריך לבצע **גלגול** על מנת שהעץ יחזור להיות עץ AVL. כל גלגול אורך זמן קבוע $\Theta(1)$.

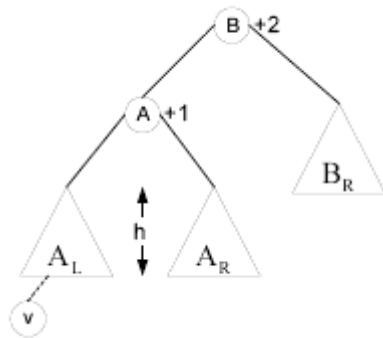
נגדיר ארבעה סוגי גלגולים: LL, RR, LR, RL. בכדי לדעת מתי יש לבצע כל גלגול, נשתמש בטבלה הבאה:

סוג הגלגול המתאים	בבן-הימני v_R	בבן-השמאלי v_L	בשורש v
LL		$BF(v_L) = +1$	$BF(v) = +2$
LR		$BF(v_L) = -1$	$BF(v) = +2$
RR	$BF(v_R) = -1$		$BF(v) = -2$
RL	$BF(v_R) = +1$		$BF(v) = -2$

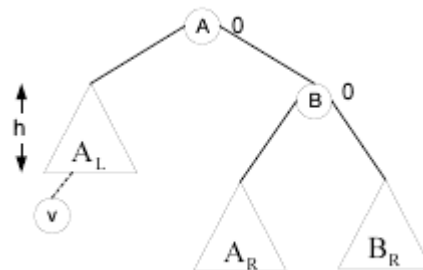


גלגול LL

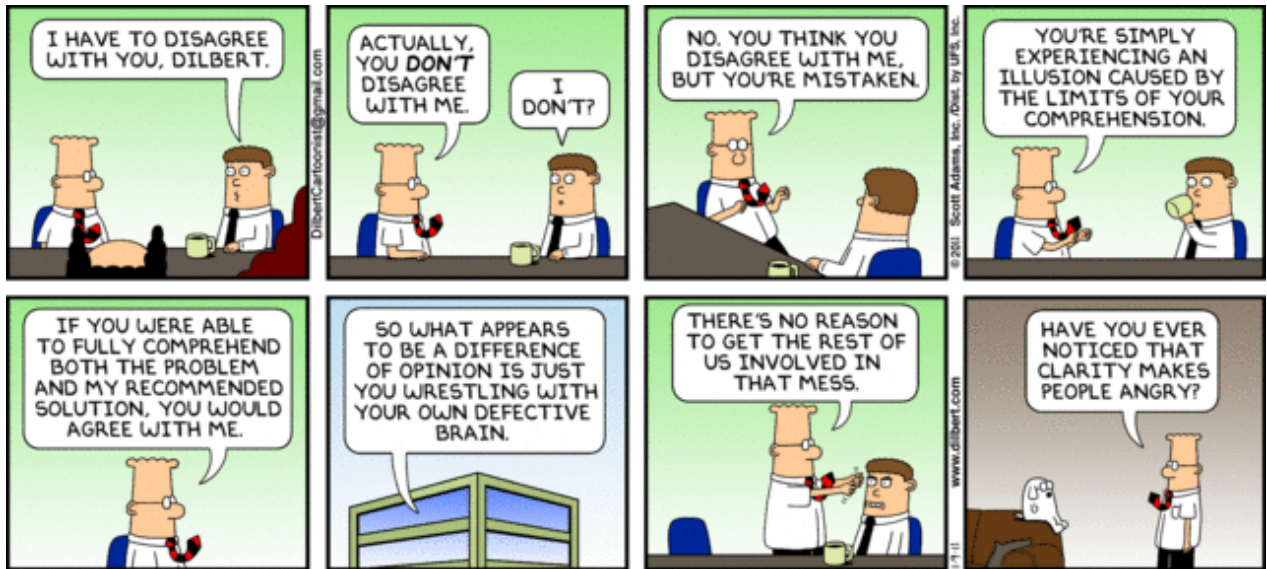
לפני שהכנסנו את v , גובה העץ היה $h+2$. הוספת הצומת v גרמה להגדלת גובהו של A_L ל- $h+1$. מהטבלה בעמ' הקודם אנו רואים שצריך לבצע גלגול LL, שיעביר את A לשורש.



לאחר ביצוע הגלגול, העץ יראה כך:

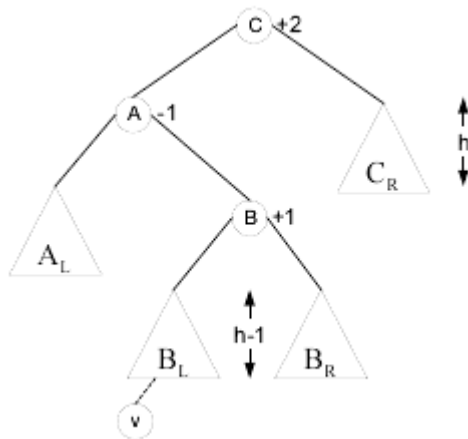


(גלגול RR סימטרי לגמרי ל-LL, ומתבצע באותו האופן).

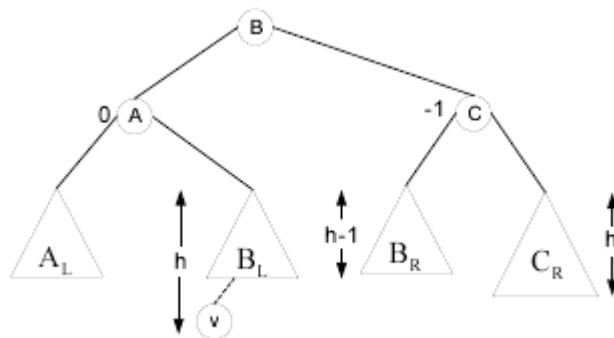


גלגול LR

הוכנס איבר ל-BL, שגרם לו להעלות את גובהו ל-h:



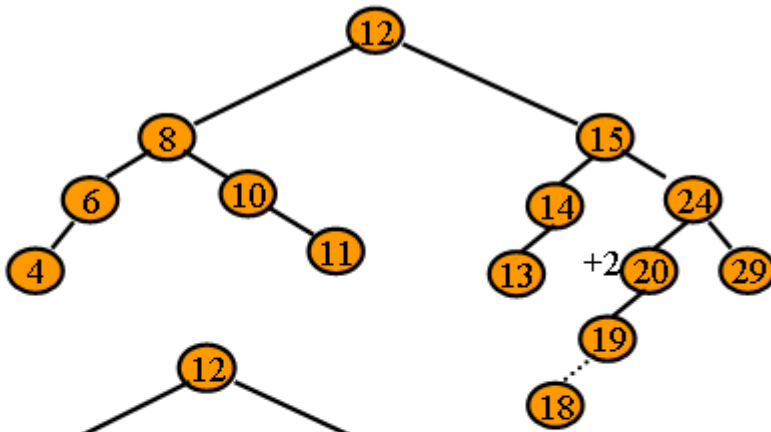
לאחר ביצוע הגלגול, העץ יראה כך:



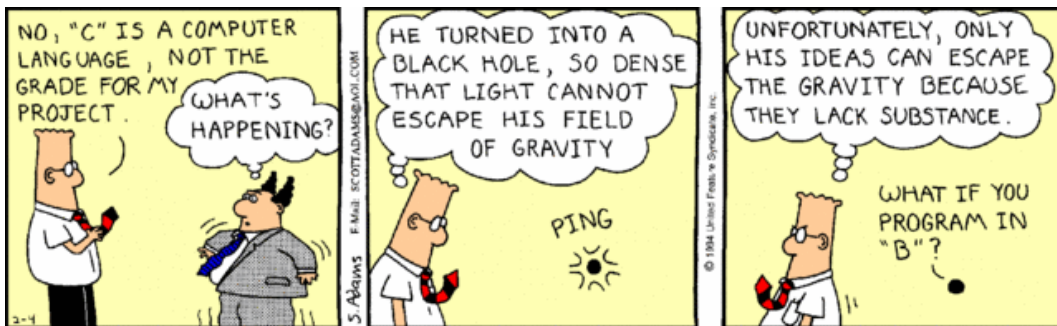
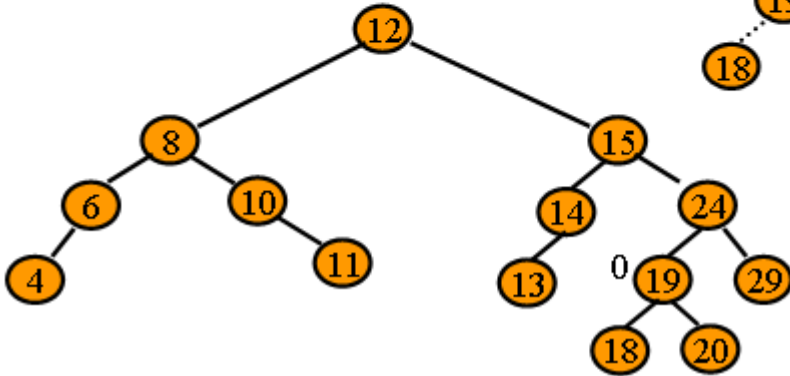
גלגול RL סימטרי לגמרי ל-LR,
ומבצע באותו האופן)

דוגמאות:

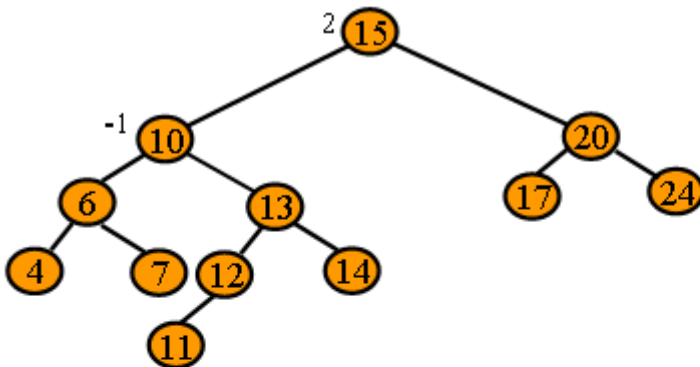
הוספת 18 :



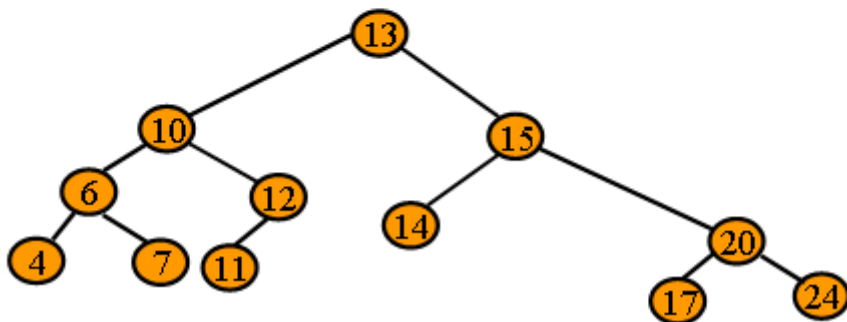
אחרי גלגול LL:



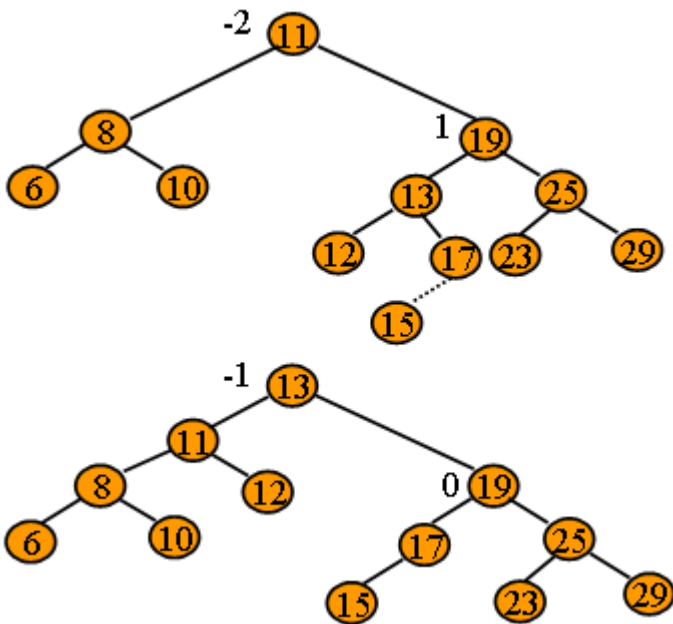
הוספת 11 :



אחרי גלגול LR:



בזכות היותם מאוזנים, עצי AVL מאפשרים ביצוע חיפוש, הוצאה והכנסה בזמן $O(\log n)$.



הוספת 15 :

אחרי גלגול RL :

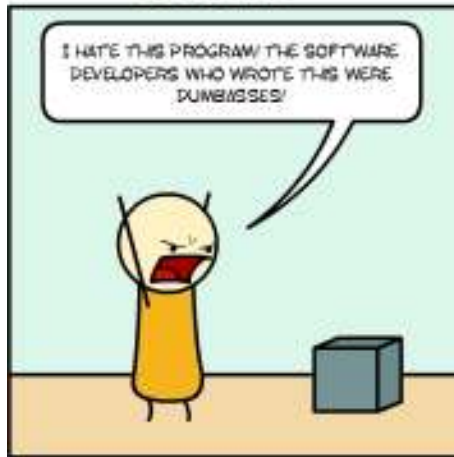
יישומוני (applet) ב-Java, וסרטוני אנימציה, שממחישים הוצאה/הכנסה/חיפוש בעץ AVL :

<http://www.cs.jhu.edu/~goodrich/dsa/trees/avltree.html>

<http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html>

http://www.youtube.com/watch?v=LS_U6g-Fsts

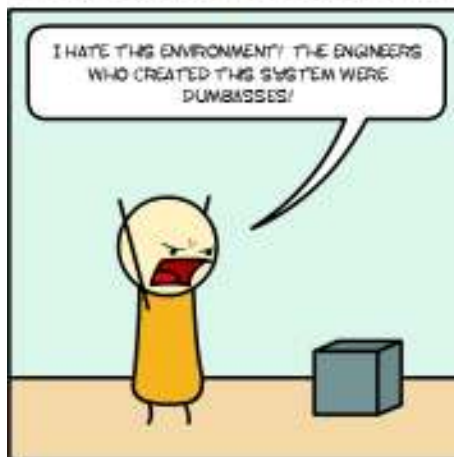
USER



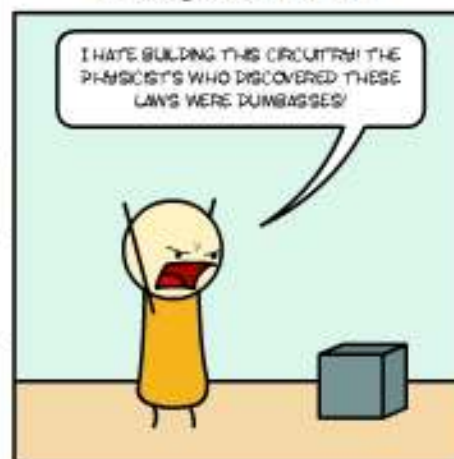
PROGRAMMER



LANGUAGE DESIGNER



ENGINEER



PHYSICIST



2. מסלולים קצרים בגרף

הגדרה: בהינתן גרף לא-ממושקל, אורך של מסלול יוגדר בתור מספר הקשתות במסלול. המרחק בין v לבין u יוגדר בתור האורך הקטן ביותר של מסלול בין v ל- u .

הגדרה: בהינתן גרף ממושקל, אורך של מסלול יוגדר בתור סכום המשקלים על הקשתות במסלול. המרחק בין v לבין u יוגדר בתור האורך הקטן ביותר של מסלול בין v ל- u .

ישנן ארבע בעיות אלגוריתמיות שניתן להציג בהקשר של מסלולים קצרים ביותר בגרף:

בעיה 1 (מקור יחיד – יעדים רבים): בהינתן גרף $G = (V, E)$ וקודקוד $s \in V$, המטרה היא למצוא את המרחקים הקצרים ביותר בין s לבין כל יתר קודקודי הגרף.

בעיה 2 (מקור יחיד – יעד יחיד): בהינתן גרף $G = (V, E)$, קודקוד $s \in V$ וקודקוד $t \in V$, המטרה היא למצוא את המרחק הקצר ביותר בין s לבין t .

בעיה 3 (מקורות רבים – יעד יחיד): בהינתן גרף $G = (V, E)$ וקודקוד $t \in V$, המטרה היא למצוא את המרחקים הקצרים ביותר בין כל יתר קודקודי הגרף לבין t .

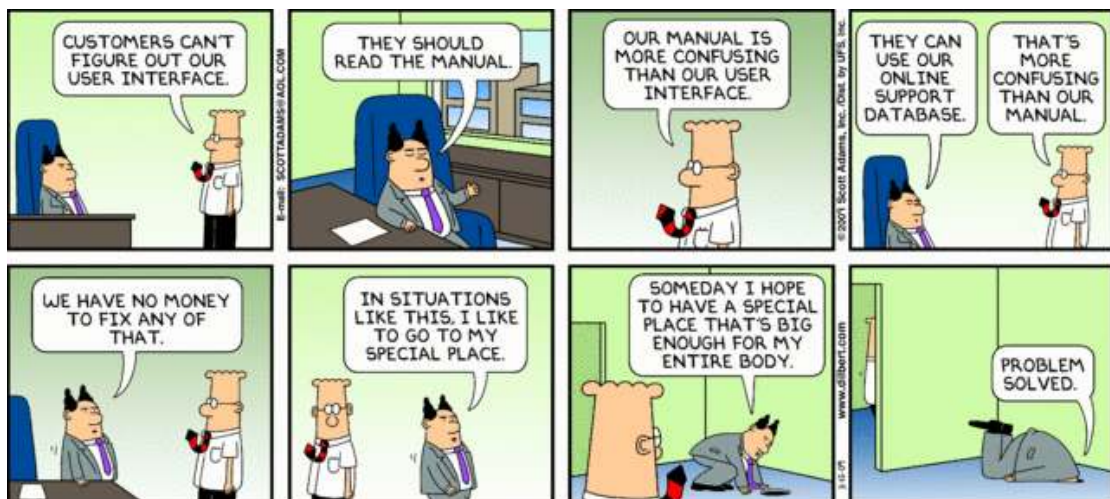
בעיה 4 (מקורות רבים – יעדים רבים): בהינתן גרף $G = (V, E)$, המטרה היא למצוא את המרחקים הקצרים ביותר בין כל קודקוד בגרף לבין כל יתר קודקודי הגרף.

אם קיים לנו אלגוריתם הפותר את בעיה 1 – הוא פותר, על הדרך, גם את בעיה 2.

אם קיים לנו אלגוריתם הפותר את בעיה 1 – ניתן להשתמש בו לפתרון בעיה 3 (אם הגרף בלתי-מכוון, פשוט ניקח את t בתור קודקוד מקור; ואם הגרף מכוון – נהפוך את כיווני כל הקשתות, ואז ניקח את t בתור קודקוד מקור).

אם קיים לנו אלגוריתם הפותר את בעיה 1 – ניתן להשתמש בו לפתרון בעיה 4 (נזמן את האלגוריתם עבור כל קודקוד וקודקוד).

לכן – בעיה 1 היא הבעיה היסודית בהקשר של מציאת מסלולים קצרים בגרף, ונפתח אלגוריתמים שונים הפותרים אותה. נוכל להשתמש באלגוריתמים האלה, תוך שינויים, כדי לפתור גם את בעיות 2, 3 ו-4.



3. חיפוש לרוחב תחילה (Breadth-First Search)

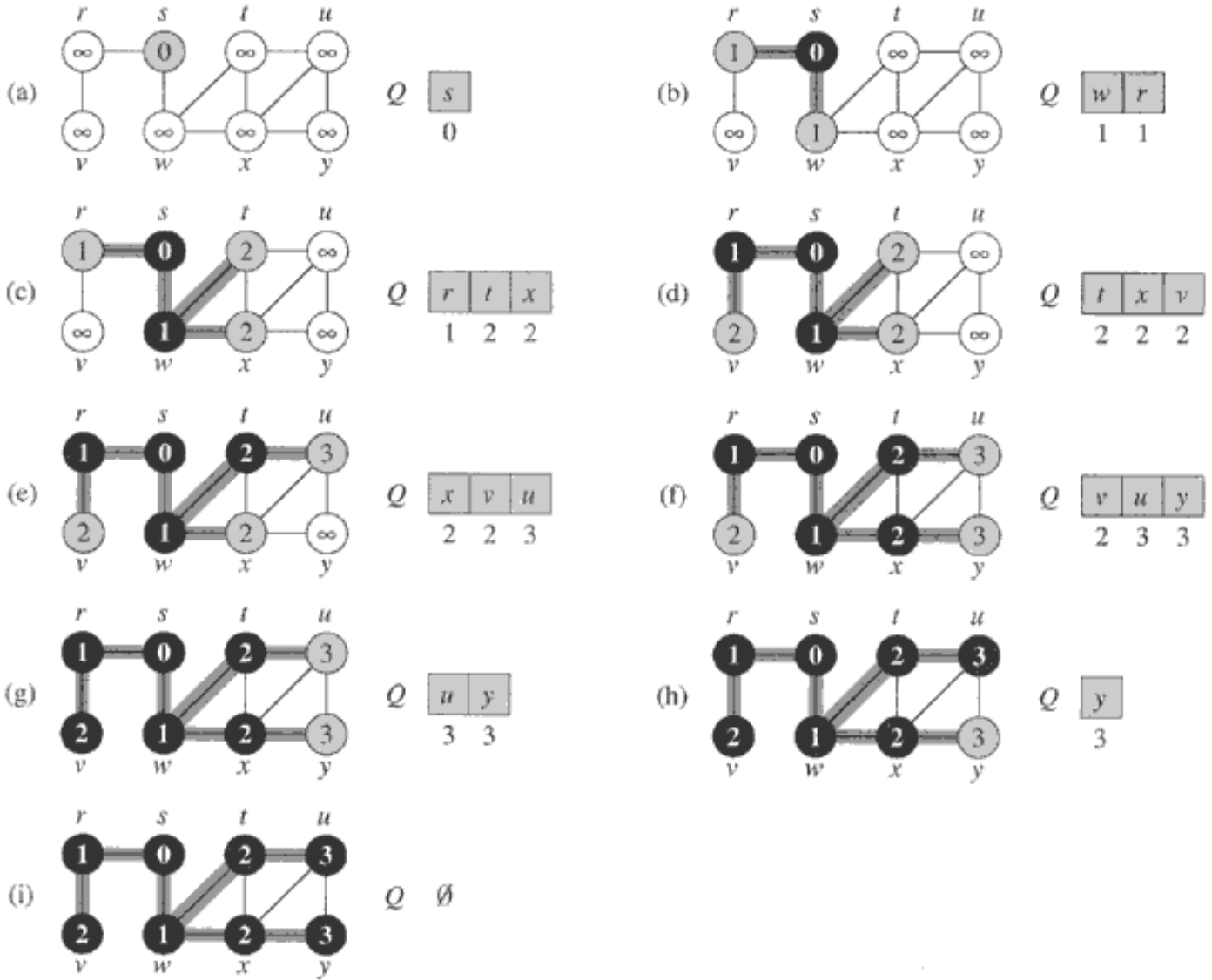
בהינתן גרף $G = (V, E)$ וקודקוד מסוים s המשמש כמקור (source), חיפוש לרוחב בוחן בשיטתיות את הקשתות ב- G כדי לגלות כל קודקוד שניתן להגיע אליו מ- s . הוא מחשב את המרחק (מס' הקשתות המינימלי) מ- s לכל הקודקודים שניתן להגיע אליהם מ- s . האלגוריתם פועל על גרפים מכוונים ובלתי-מכוונים כאחד. חיפוש לרוחב מכונה כך מפני שהוא מקדם את הגבול בין קודקודים שהתגלו לקודקודים שטרם התגלו באופן אחיד לרוחב הגבול. כלומר, האלגוריתם מגלה את כל הקודקודים הנמצאים במרחק k מ- s לפני שהוא מגלה איזשהו קודקוד הנמצא במרחק $k+1$ מ- s .

כדי לעקוב אחר התקדמותו, חיפוש לרוחב צובע את כל הקודקודים בלבן, באפור או בשחור. בתחילה, כל הקודקודים הם לבנים ועשויים להיצבע מאוחר יותר באפור ולאחר מכן בשחור. קודקוד מתגלה בפעם הראשונה שנתקלים בו במהלך החיפוש, וברגע זה הוא הופך להיות לא-לבן. קודקודים אפורים ושחורים הם קודקודים שהתגלו, אולם חיפוש לרוחב מבחין ביניהם כדי להבטיח שהחיפוש אכן יתקדם לרוחב.

אם $(u, v) \in E$ והקודקוד u הוא שחור, אזי הקודקוד v הוא אפור או שחור; כלומר, כל הקודקודים הסמוכים לקודקודים שחורים כבר התגלו. לקודקוד אפור עשויים להיות כמה קודקודים סמוכים לבנים; קודקודים אפורים מייצגים את הגבול בין קודקודים שהתגלו לקודקודים שטרם התגלו.

השגרה BFS שומרת עבור כל קודקוד $u \in V$ את צבעו ($u.color$), את המרחק מהמקור s ועד אליו ($u.distance$), ואת הקודקוד הקודם אליו ($u.previous$). אם ל- u אין קודקוד קודם (למשל, אם $u = s$ או אם u לא התגלה), אזי $u.previous = \text{NULL}$. האלגוריתם משתמש בתור Q לניהול קבוצת הקודקודים האפורים.

```
BFS( $G, s$ )
1   for each vertex  $u \in V$ 
2       do  $u.color \leftarrow \text{WHITE}$ 
3            $u.distance \leftarrow \infty$ 
4            $u.previous \leftarrow \text{NULL}$ 
5    $s.distance \leftarrow 0$ 
6    $s.color \leftarrow \text{GRAY}$ 
7    $Q \leftarrow \{s\}$ 
8   while  $Q \neq \emptyset$ 
9       do  $u \leftarrow \text{DEQUEUE}(Q)$ 
10           $u.color \leftarrow \text{BLACK}$ 
11          for each vertex  $v$  adjacent to  $u$ 
12              do if  $v.color = \text{WHITE}$ 
13                  then  $v.color \leftarrow \text{GRAY}$ 
14                       $v.distance \leftarrow u.distance + 1$ 
15                       $v.previous \leftarrow u$ 
16                       $\text{ENQUEUE}(Q, v)$ 
```



חיפוש לרוחב בונה עץ רוחב המכיל בתחילה רק את השורש, שהוא קודקוד המקור s . בכל פעם שמתגלה קודקוד לבן v במהלך הסריקה של קודקוד שכבר התגלה, הקודקוד v והקשת (u,v) נוספים לעץ. אנו אומרים ש- u הוא הקודם (predecessor) ל- v , או האב (parent) של v בעץ הרוחב.

לאחר ריצת האלגוריתם BFS, נוכל להדפיס את המסלול הקצר ביותר מ- s לקודקוד כלשהו v , על-די הפעלת האלגוריתם הרקורסיבי הבא:

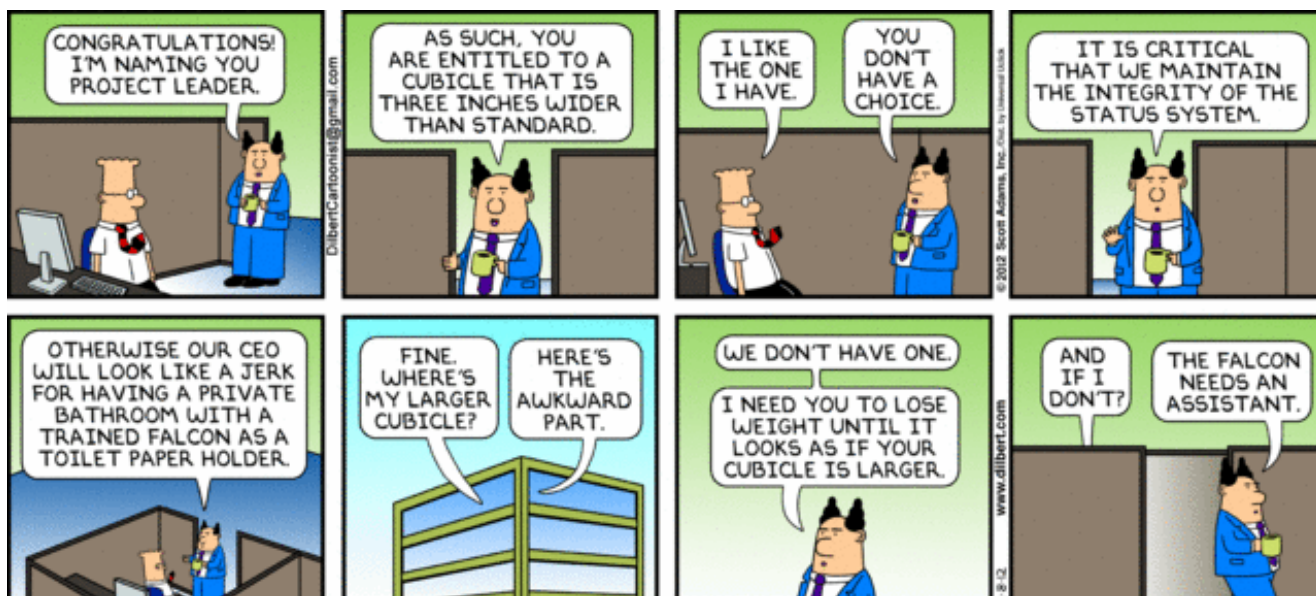
```

PRINT-PATH( $G, s, v$ )
1   if  $v = s$ 
2     then print  $s$ 
3     else if  $v.previous = \text{NULL}$ 
4         then print "no such path"
5     else PRINT-PATH( $G, s, v.previous$ )
6         print  $v$ 

```

ננתח את סיבוכיות האלגוריתם BFS: מכיוון שכל קודקוד נצבע בלבן אך ורק באתחול, הבדיקה בשורה 12 מבטיחה שכל קודקוד יוכנס לתור פעם אחת לכל היותר, ולכן גם יוצא ממנו פעם אחת לכל היותר. כל פעולת הכנסה לתור או הוצאה ממנו מתבצעת בזמן $\Theta(1)$, ולכן הזמן הכולל המוקדש לפעולות על התור הוא $\Theta(|V|)$. גם האתחול מתבצע בזמן $\Theta(|V|)$. הזמן הכולל לסריקת הקשתות (שורה 11) הוא $\Theta(|E|)$, ובסה"כ נקבל כי זמן הריצה של השגרה BFS הוא $\Theta(|V|+|E|)$.

סיבוכיות האלגוריתם הרקורסיבי PRINT-PATH היא $\Theta(|V|)$, שכן היא סורקת את המסלול מ-v ועד ל-s, ומסלול זה עובר לכל היותר פעם אחת בכל קודקוד.



4. מיון מנייה (מיון ספירה – Counting Sort)

הוכחנו שכל אלגוריתם מיון המתבסס על השוואות, יתבצע בזמן $\Omega(n \log n)$. אולם כיצד יראה אלגוריתם מיון שאינו מתבסס על השוואות?

נניח שאנחנו מקבלים כקלט מערך A בגודל n איברים, שכל אחד מהם לקוח מהתחום $0..k-1$. כלומר, אנחנו מניחים כאן שהקלט מוגבל לתחום מסוים. מטרתנו היא להחזיר מערך B , בגודל n איברים, המכיל את איברי מערך הקלט A כשהם ממוינים. לשם כך, ניעזר במערך עזר בשם C שגודלו k איברים (נשים לב! גודלו של המערך C יכול להיות שונה מגודלם של A ו- B).

נפעיל את האלגוריתם הבא, הנקרא **מיון מנייה** (או **מיון ספירה**), המנצל את העובדה שאיברי הקלט לקוחים מתחום מוגבל, כדי למנות (לספור) כמה פעמים כל ערך אפשרי מופיע בקלט.

```
COUNTING-SORT (A, B, n, k)
1   for i ← 0 to k-1
2     do C[i] ← 0
3   for j ← 0 to n-1
4     do C[A[j]] ← C[A[j]] + 1
5   for i ← 1 to k-1
6     do C[i] ← C[i] + C[i-1]
7   for j ← n-1 downto 0
8     do C[A[j]] ← C[A[j]] - 1
9     B[C[A[j]]] ← A[j]
```

הסבר האלגוריתם

בשורות 1-2 אנחנו מאתחלים את מערך המונים C .

בשורות 3-4 אנחנו עוברים על כל איברי המערך המקורי A , ומגדילים ב-1 את המונה המתאים. בסיום ביצוע לולאה זו, התא $C[x]$ מכיל את כמות ההופעות של הערך x .

בסיום ביצוע הלולאה בשורות 5-6, התא $C[x]$ יכיל את כמות הערכים הקטנים-או-שווים ל- x .

הלולאה בשורות 7-9 מציבה כל איבר במיקומו המתאים.

לדוגמא, עבור מערך הקלט A הבא, כך יראה מערך המונים C בסוף שורה 4:

A

2	5	3	0	2	3	0	3
---	---	---	---	---	---	---	---

C

2	0	2	3	0	1
---	---	---	---	---	---

אחרי ביצוע הלולאה בשורות 5-6, המערך C יראה כך:

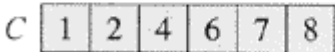
C

2	2	4	7	7	8
---	---	---	---	---	---

אחרי ביצוע האיטרציה הראשונה של הלולאה האחרונה :



אחרי ביצוע האיטרציה השנייה של הלולאה האחרונה :



אחרי ביצוע האיטרציה השלישית של הלולאה האחרונה :



ובתום ביצוע האלגוריתם :



B מכיל את איברי הקלט A, כשהם בצורה ממוינת בסדר עולה.

סיבוכיות זמן הריצה

מבחינת סיבוכיות זמן הריצה, הלולאה הראשונה מבצעת k איטרציות, הלולאה השלישית מבצעת k-1 איטרציות, והלולאות השנייה והרביעית מבצעות n איטרציות. גוף כל אחת מהלולאה מתבצע בזמן קבוע, ולכן, בסך הכל, סיבוכיות זמן הריצה היא $\Theta(n+k)$, כלומר – היא תלויה באופן לינארי הן בגודלו של הקלט והן בגודלו של טווח הערכים האפשרי עבור ערכי הקלט.

אם טווח הערכים הוא קטן, האלגוריתם יהיה יעיל מאוד. לדוגמא, אם מתקיים $k = O(n)$, אז סיבוכיות זמן הריצה של מיון מנייה תהיה $\Theta(n)$. נשים לב שזמן ריצה לינארי זו תוצאה יותר טובה מאשר החסם התחתון $\Omega(n \log n)$ במודל ההשוואות שהוכחנו קודם. אם כך, האם נפלה טעות בהוכחה שראינו קודם? לא, האלגוריתם למיון מנייה פשוט אינו מבוסס על השוואות, ולכן – החסם התחתון שהוכחנו אינו תקף לגביו.

סיבוכיות המקום בזיכרון

עוד נציין שמבחינת סיבוכיות המקום בזיכרון (space complexity), יש להקצות מערך עזר שגודלו כגודלו של טווח הערכים האפשרי. זו עוד סיבה מדוע אנחנו מעוניינים שגודלו של הטווח יהיה קטן – לא רק כדי שסיבוכיות זמן הריצה (run-time complexity) תהיה נמוכה, אלא גם כדי שלא נאלץ להקצות מערך עזר מאוד גדול בזיכרון המחשב.

יציבות המיון

נשים לב ששני איברים בעלי אותו הערך, שהופיעו במערך הקלט A, יופיעו שניהם במערך הפלט B, כשהסדר היחסי ביניהם נשמר.

לדוגמא, המספר 3 מופיע כמה פעמים במערך הלא-ממוין שאנחנו מקבלים כקלט. במהלך ביצוע האלגוריתם, המופע הימני ביותר של 3 במערך הקלט, יהיה זה שיועתק למקום הימני ביותר (מבין כל המופעים של 3) במערך הפלט; המופע השני-הכי-ימני של 3 במערך הקלט, יהיה זה שיועתק למקום השני-הכי-ימני (מבין כל המופעים של 3) במערך הפלט, וכו'.

על שיטת מיון שמקיימת תכונה זו, נאמר שהיא יציבה (stable). יציבות של אלגוריתם מיון זו תכונה רצויה, ומסתבר – שמיון מנייה זהו מיון יציב.