The BFS (Breadth-First Search) Algorithm

חיפוש לרוחב תחילה

BFS הוא אחד האלגוריתמים הפשוטים ביותר לסריקת גרף ואב-טיפוס של אלגוריתמים חשובים רבים עבור גרפים. בהינתן גרף G=(V, E) וקדקוד מסוים s המשמש כמקור (source), BFS בוחן בשיטתיות את הצלעות של G כדי "לגלות" כל קדקוד שניתן להגיע אליו מ-s . BFS מחשב את המרחק (מספר צלעות מינימאלי) מ-s לכל הקדקודים שניתן להגיע אליהם מ-s וכן בונה "עץ רוחב" ("breadth-first tree") ששורשו s והוא מכיל את כל הקדקודים הללו. עבור כל קדקוד v שניתן להגיע אליו מ-s המסלול מ-s ל- v בעץ הרוחב הוא המסלול הקצר ביותר מ-s ל- v ב-G. האלגוריתם פועל על גרפים מכוונים ולא מכוונים כאחד. חיפוש לרוחב מכונה כך מפני שהאלגוריתם מגלה את כל הקדקודים הנמצאים במרחק k מ-s לפני שהוא מגלה איזשהו קדקוד שנמצא במרחק k+1 מ-s .

כדי לעקוב אחר התקדמותו, BFS צובע כל קדקוד בלבן, באפור ובשחור. בתחילה כל הקדקודים הם לבנים. קדקוד המתגלה בפעם הראשונה הופך להיות אפור, כלומר לקדקוד אפור יש שכנים לבנים. קדקוד, שכל השכנים שלו התגלו הופך להיות שחור, כלומר שכנים של קדקוד שחור הם אפורים או שחורים.

BFS בונה עץ רוחב, שהשורש שלו הוא המקור s. בכל פעם שמתגלה קדקוד לבן v כשכן של קדקוד u שכבר התגלה, הקדקוד v והצלע (u,v) נוספים לעץ. אומרים ש-u הוא קודם (predecessor) ל-v או האב (parent) של v בעץ הרוחב. מכוון שכל קדקוד מתגלה לכל היותר פעם אחת, יש לו לכל היותר אב אחד.

BFS Pseudo code:

```
G - the graph is represented using adjacency-list
Q - the queue to manage the set of gray vertices
color[] - the color of  vertex u is stored in color[u]
s - the source vertex
p[] - the predecessor (parent) of vertex u is stored
      in p[u]. If u has no predecessor then [u] = nil
d[] - the distance from the source vertex s to vertex u,
      computed by the algorithm is stored in d[u].
nil = -1 inexistent distance and inexistent vertex number
Adj[u]  - adjacency list of neighboring vertices of vertex u
```

```
BFS(G, s)
    for each vertex u in V[G]
       color[u] = WHITE
       d[u] = nil
       p[u] = nil
    end for
    color[s] = GRAY
    d[s] = 0
    p[s] = nil
    Q = empty

    ENQUEUE(Q, s)
```

```
11.   while (Q != EMPTY)
12.      u = DEQUEUE(Q)
13.      for each vertex v in Adj[u]
14.         if (color[v] == WHITE) then
15.             color[v] = GRAY
16.             d[v] = d[u] + 1
17.             p[v] = u
18.             ENQUEUE(Q, v)
19.         endif
20.      endfor
21.      color[u] = BLACK
22.   end while
23.   return (d, p)
end BFS
```

התכנית פועלת כדלקמן:

שורות 1-5 צובעות כל הקדקודים בלבן, מציבות את ערך של `nil` (מרחק אינסופי) במערך המרחקים `d[u]` עבור כל קדקוד u, ומציבות במערך של אבות `p[u]` את ערך ה-`nil` (מספר קדקוד שלא קיים) עבור כל קדקוד u.
שורה 6 צובעת את מקור `s` באפור, כי הוא נחשב לקדקוד שהתגלה.
שורה 7 מאתחלת את `d[s]` לאפס (מרחק בין s לעצמו).
שורה 8 מציבה `nil` לקדקוד המקור s (ל-s אין קודם).
שורה 9 מאתחלת את אתור Q כך שיכיל רק את קדקוד s. Q תמיד יכיל את קבוצת הקדקודים האפורים.

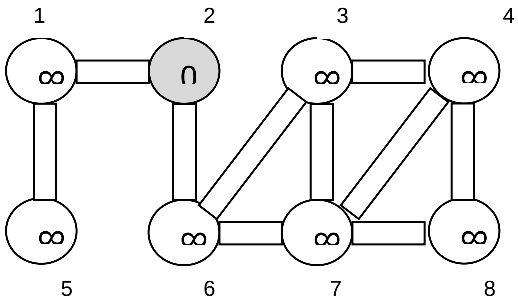
הלולאה העיקרית היא **`while`** בשורות 11-22. הלולאה חוזרת ומתבצעת כל עוד נותרו קדקודים אפורים, שהם קדקודים שהתגלו אשר ברשימות הסמוכים שלהם נותרו קדקודים לבנים.

שורה 12 מוציאה את הקדקוד האפור `u` מראש התור Q.
לולאת **`for`** בשורות 13-20 בוחנת כל קדקוד `v` ברשימת הסמוכים של `u`. אם `v` לבן, הרי ש- `v` טרם התגלה, והאלגוריתם מגלה אותו על-ידי ביצוע שורות 14-19: תחילה `v` נצבע באפור ובמשתנה המרחק שלו ממקור `d[v]` מוצב הערך `d[u] + 1`, `u` נרשם כאביו של `v` ולבסוף, `v` מוכנס לסוף התור Q.
לאחר שנבחנו כל הקדקודים ברשימת הסמוכים שלו, `u` נצבע בשחור בשורת 21 ונמחק מן התור בשורת 12.

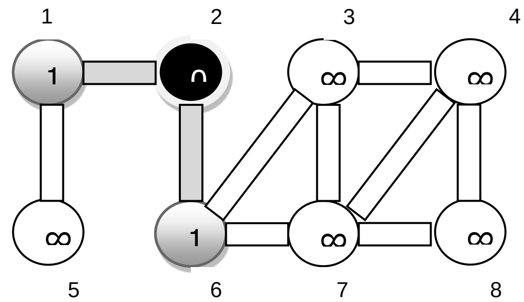
סיבוכיות האלגוריתם

כל קדקוד נכנס לתור פעם אחד לכל היותר (כאשר הוא לבן). כל פעולת הכנסת לתור או הוצאת ממנו מתבצעת בזמן O(1). בעצם לכל קדקוד האלגוריתם עובר על כל קדקודים הסמוכים לו, כלומר עוברים פעמיים על כל צלעות הגרף. לכן סריקת הגרף מתבצעת בזמן O(|E|). בנוסף אתחול מתבצע בזמן O(|V|). ולפיכך זמן הריצה הכולל של התכנית הוא O(|E|+|V|).

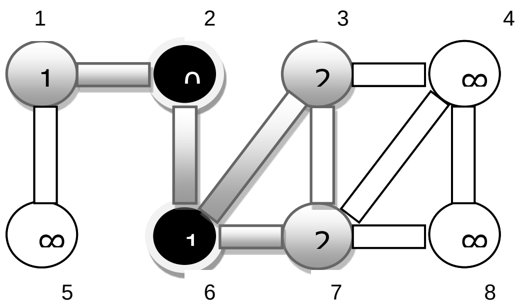
באיור הבא מתוארות פעולות של BFS על גרף בלתי מכוון.
המקור הוא קדקוד 2. הצלעות באיור הופכות להיות למוצללות עם הוספתן ע"י BFS לעץ הרוחב. בתוך כל קדקוד u רשום d[u]. התור Q מוצג בתחילת כל איטרציה של לולאת ה-**while** שבשורות 11-22.



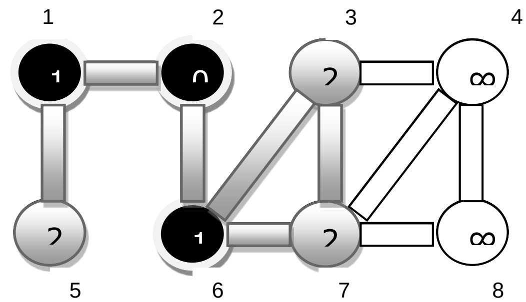
Q
2



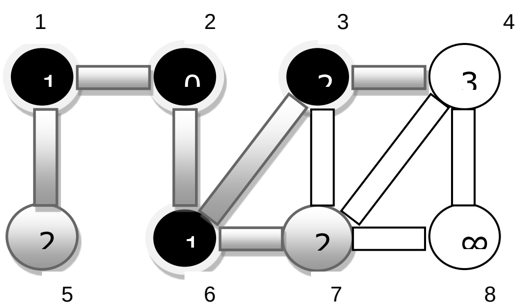
Q
6
1



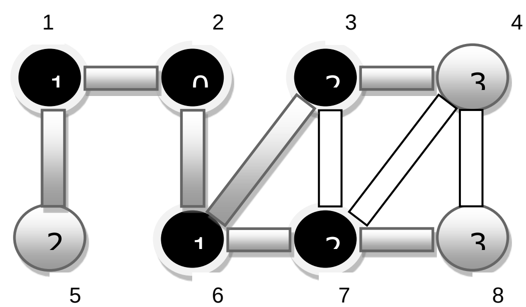
Q
1
3
7



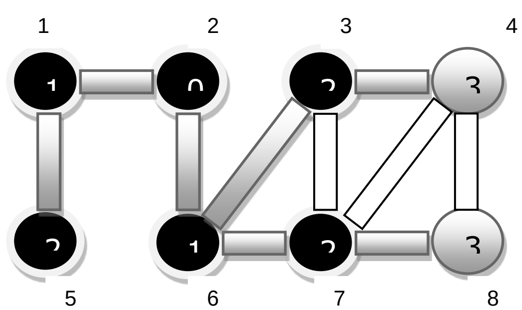
Q
3
7
5



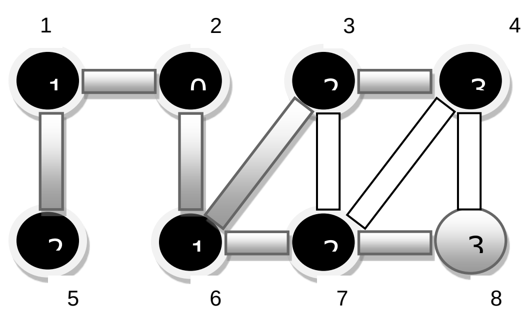
Q
7
5
4



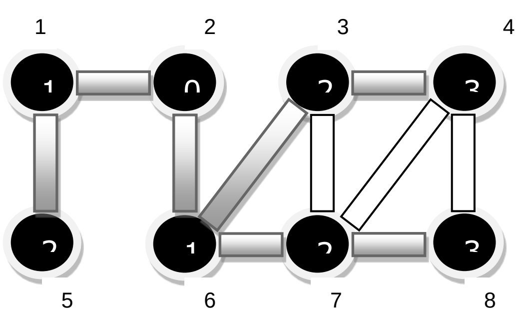
Q
5
4
8



Q
4
8



Q
8



Q
Φ

תכונות של BFS:

1. במהלך סריקת הגרף BFS בונה עץ רוחב (breadth-first tree) כתת-גרף של G בעל הקדקודים שניתן להגיע אליהם מ-s.
<u>הוכחה</u>: בשלב הראשון העץ מכיל רק קדקוד אחד – המקור s. כאשר אנו מוסיפים קדקוד חדש (לפי האלגוריתם אנו מוסיפים רק קדקודים לבנים שעדין לא התגלו) אנו מוסיפים גם את הצלע שבעזרתה הגענו לקדקוד זה. מכוון שאנו מגלים כל קדקוד לכל היותר פעם אחת יש לו לכל היותר קדקוד האב אחד. לכן המבנה הנוצר ע"י BFSהוא עץ.

2. כאשר תור מורכבת מקדקודים הבאים $Q=\{v_1, v_2, \ldots, v_r\}$, מתקיים $d[v_i] \le d[v_{1+i}]$ $(i=1,2,\ldots,r-1)$.
<u>הוכחה</u>: באינדוקציה מתמטית.
א) בסיס האינדוקציה: בשלב הראשון התור מכיל רק קדקוד אחד – המקור s, $d[s]=0$. בשלב הבא מוסיפים קדקודים סמוכים אליו $v \in Adj(s)$, $d[v]=d[s]+1=1$. נציין שאיברים חדשים נמצאים בסוף התור.
ב) הנחת אינדוקציה: בשלב מספר i מרחקם של איברי התור $Q=\{v_1, v_2, \ldots, v_r\}$, עד המקור הם $d[v_j]=k$ כאשר $j=1,2,\ldots t$ ו- $d[v_j]=k+1$ כאשר $j=t+1,\ldots r$.
ג) שלב אינדוקציה. נוכיח את הטענה עבור $i+1$. בשלב $i+1$ אנו מוציאים מהתור קדקוד שמרחקו עד המקור הוא k ומוסיפים לתור את קדקודים הסמוכים אליו שמרחקם עד המקור הוא $k+1$. לכן התכונה נשמרת. מש"ל.

3. **נכונות של BFS** . המרחקים של קדקודי הגרף עד המקור, המתקבלים בעזרת אלגוריתם BFS הם המרחקים הקצרים ביותר.
<u>הוכחה</u>: בדרך השלילה.
נגדיר $\delta(u,v)$ <u>אורך המסלול הקצר ביותר</u> מ- u ל- v (shortest path distance) כמספר המינימאלי של צלעות במסלול כלשהו מהקדקוד u לקדקוד v, או ∞ אם לא קיים מסלול בין u ל-v. מסלול באורך $\delta(u,v)$ מ- u ל- v נקרא מסלול קצר ביותר (shortest path) מ- u ל-v. כתוצאה הפעלת האלגוריתם התקבל ערך `d[u]` שהוא שווה למספר צלעות שעלינו לעבור אותן כדי להגיע מ- s ל-u.
נניח שקיימת קבוצת קדקודי הגרף לא ריקה $P=\{p_1,p_2,\ldots p_m\}$ כך שלכל $p\in P$ $d[p]> \delta(p,s)$. ניקח קדקוד $u\in P$ שמרחקו עד המקור $\delta(u,s)$ הוא מינימאלי: כלומר $\delta(u,s) = \min\{ \delta(p,s) : p\in P\}$.
יהיה w הוא קדקוד האב של קדקוד u במסלול הקצר ביותר, אז
$\delta(u,s)= \delta(w,s)+1$, כלומר $\delta(w,s)< \delta(u,s)$
זאת אומרת ש $w\notin P$ והמרחק עד אליו מ-s מחושב נכון ע"י BFS : $d[w]= \delta(w,s)$.
יהיה v הוא קדקוד האב של קדקוד u במסלול המחושב ע"י BFS, אז $d[u] = d[v]+1$.
אבל $u\in P$, לכן

$$\delta(u,s) < d[u] \quad \text{או}$$
$$\delta(w,s)+1 < d[v]+1 \quad \text{או}$$
$$d[w]+1 < d[v]+1 \quad \text{או}$$
$$d[w] < d[v]$$

לפי תכונה 2 w נכנס לתור לפני v, אבל w סמוך ל-u, כלומר v לא יכול להיות קדקוד האב של u בעץ BFS. קבלו סתירה , לכן המרחקים של קדקודי הגרף עד המקור, המתקבלים בעזרת אלגוריתם BFS הם המרחקים הקצרים ביותר. מש"ל

מבנה הגרף:

הגרף נתון כווקטור של רשימות מקושרות. כל רשימה מכילה קדקוד וכל הקדקודים הקשורים אליו. מספר קדקודי הגרף נסמן ב- n.

דוגמה: הגרף שלמעלה מוצג בצורה הבאה:

$1 \to 2 \to 5$

$2 \to 1 \to 6$

$3 \to 4 \to 6 \to 7$

$4 \to 3 \to 7 \to 8$

$5 \to 1$

$6 \to 2 \to 3 \to 7$

$7 \to 3 \to 4 \to 6 \to 8$

$8 \to 3 \to 7$

BSF פותרת בעיות הבאות:

א) בדיקה אם גרף קשיר:

עוברים על מערך של מרחקים d ובודקים אם נשאר קדקוד u כלשהו שמרחקו עד המקור s הוא אינסופי (d[u]==infinity). כאשר קדקוד כזה קיים הגרף אינו קשרי. כאשר שמרחקו של כל קדקודי הגרף עד המקור s הוא מספר סופי – הגרף קשיר.

ב) הדפסת המסלול הקצר ביותר בין שני קדקודי הגרף

```
//  The following procedure prints out the vertices on a shortest path from s to v,
//   assuming that BFS has already been run to compute the shortest-path tree.
PRINT-PATH(G, s, v)
    if v == s
        print s
    else if π[v] == NIL
        print "no path from" s "to" v "exists"
    else
        PRINT-PATH(G, s, π[v])
            print v
```

ג) חישוב מספר רכיבי קשירות:

1. אתחול: מגדירים קדקוד התחלתי שהוא קדקוד ראשון ברשימה index = 0
מגדירים count = 1 – מספר רכיבי קשירות של הגרף
2. מפעילים BFS על קדקוד index
3. בודקים האם קיים קדקוד שמרחקו עד קדקוד index שווה אינסוף.

- ♦ במקרה שקדקוד כזה קיים (קדקוד שמספרו i) מקדמים ב-1 את מספר רכיבי קשירות של הגרף: count ומעדכנים ערך של index ל- i וחוזרים לשלב 2.
- ♦ במקרה שקדקוד כזה כבר לא קיים הפונקציה מחזירה את count - מספר רכיבי הקשירות המחושב.

ד) מציאת רכיבי קשירות עצמם. האלגוריתם מאוד דומה לאלגוריתם המתואר בסעיף הקודם:
חוזרים לסעיפים 1,2 ולסעיף 3 מוסיפים שמירת קודקודים הנמצאים באותו רכיב הקשירות.

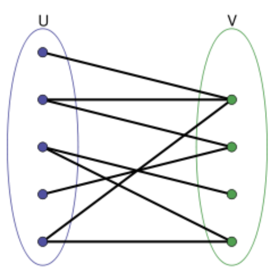
ה) חישוב קוטר העץ:

א. מפעילים BFS על קדקוד ראשון ברשימה,
ב. מחשבים את אינדקס (ind) של קדקוד שמרחקו עד הקדקוד הראשון ברשימה גדול ביותר.
ג. שוב מפעילים BFS על קדקוד ind ומחשבים את אינדקס (ind1) של קדקוד שמרחקו עד הקדקוד ind גדול ביותר.
ד. המרחק (מספר צלעות) מקדקוד ind עד קדקוד ind1 הוא הקוטר של העץ.

גרף דו-צדדי (bipartite graph)

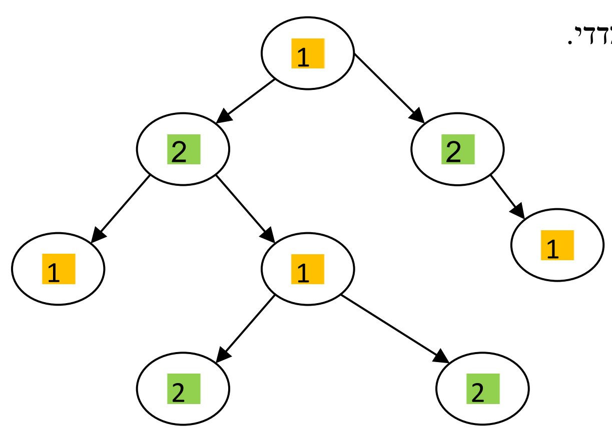
בתורת הגרפים, **גרף דו-צדדי** (נקרא גם **גרף דו-חלקי**) הוא גרף שבו ניתן לחלק את הקדקודים לשתי קבוצות זרות, כך שלא קיימת צלע בין שני קדקודים השייכים לאותה הקבוצה.

גרפים דו-צדדיים מועילים במידול בעיות התאמה. למשל, אם יש לנו קבוצה P של אנשים וקבוצה J של עבודות ואנו רוצים לבצע חלוקת עבודה, נוכל בתור מודל לתאר את האנשים והעבודות כגרף דו-צדדי שקבוצת קדקודים אחת בו היא P והשנייה J, ויש צלע בין אדם המתאים לעבודה מסוימת ועבודה זו.



דוגמה לגרף דו-צדדי

עוד דוגמה: כל עץ הוא גרף דו-צדדי.



משפט: גרף קשיר הוא דו-צדדי אם ורק אם כל המעגלים שלו הם באורך זוגי.

<u>הוכחה</u>: יהי G=(V1, V2, E) גרף דו-צדדי. נתבונן במעגל כלשהו בגרף. נצא מקדקוד $v_1 \in V_1$ הנמצא במעגל הזה ונטייל לאורך המעגל. כל צלע שעליה אנחנו מטיילים מעבירה אותנו לצדו השני של המעגל. לכן, דרוש מספר זוגי של צלעות כדי לחזור לקדקוד v_1 שבו התחלנו ולסגור את המעגל.

ולהיפך, נניח שכל המעגלים בגרף G=(V, E) הם באורך זוגי ונראה ש- G דו-צדדי. יהי $s \in V$ קדקוד כלשהו בגרף G.
נגדיר את V1, V2 באופן הבא:

$$\mathbf{V_1} = \{u \in V \mid \text{distance}\,(u,s)\ is\ even\},\ \mathbf{V_2} = \{u \in V \mid \text{distance}\,(u,s)\ is\ odd\}.$$

כמובן קבוצות אלה זרות ו- $V_1 \cap V_2 = \Phi, V_1 \cup V_2 = V$. נניח בשלילה שהגרף הוא לא דו-צדדי ויש צלע {u,v} בין קדקודים השייכים לאותה קבוצה, כלומר $v,u \in V_1$. לכן המסלול הקצר ביותר מ-s ל-u, המסלול הקצר ביותר בין מ-s ל-v, וצלע {u,v} יוצרים מעגל, שעורכו אי-זוגי – סתירה. באופן דומה מוכיחים שאין צלעות בין קדקודים של V_2.

מסקנה: כל עץ הוא גרף דו-חלקי.
בעץ אין מעגלים באורך אי-זוגי, לכן הוא דו-צדדי. מש"ל.

כדי לבדוק אם גרף הוא דו-צדדי צריך להפעיל עליו **BFS** ולסמן קדקודים בצורה הבאה:
0 – הקדקוד לא התגלה,
1 – קדקוד שייך לקבוצה 1,
2 – קדקוד שייך לקבוצה 2.
מאתחלים הקדקודים ב-0. כאשר נמצא צלע ששני הקדקודים שלה שייכים לאותה קבוצה (1 או 2) הגרף אינו דו-צדדי.

```
BFS_BIPARTITE (G, s)
      boolean  bipartite = true;
      for each vertex u in V[G]
            color[u] = WHITE
            distance[u] = infinity
            parent[u] = null
            partition[u] = 0
      end for
      color[s] = GRAY
      distance [s] = 0
      parent [s] = null
      partition[s] = 1;
      Q = empty
      ENQUEUE(Q, s)
      while (Q != EMPTY and bipartite)
            u = DEQUEUE(Q)
            for (each vertex v in Adj[u] and bipartite)
                  if (partition [u] == partition [v])
                        bipartite = false;
                  else if (color[v] == WHITE) then
                       color[v] = GRAY
                        distance [v] = d[u] + 1
                        parent [v] = u
                        partition[v] = 3 - partition[u];
                        ENQUEUE(Q, v)
                  endif
            endfor
            color[u] = BLACK
      end while
      return bipartite
end BFS_BIPARTITE
```