**BFS breadth-first search**

**חיפוש לרוחב**

**Graph = (V, E) – graph, s – ( first vertex )**

**אלגוריתם Breadth First Search (BFS)** הוא אחד האלגוריתמים הפשוטים ביותר לסריקה/חיפוש בגרף. הרבה אלגוריתמים אחרים מיישמים את הרעיונות הדומים לרעיון שלו.

בהינתן גרף G=(V, E) וקדקוד מסוים 𝑠∈𝑉 המשמש כ"מקור" (source).

* חיפוש לרוחב בוחן בשיטתיות את הצלעות של G כדי "לגלות" כל קדקוד שניתן להגיע אליו מ- s.
* הוא מחשב את המרחק (מספר הצלעות המינימאלי) מ- sלכל הקדקודים שניתן להגיע אליהם מ- s
* האלגוריתם פועל גם על גרפים מכוונים וגם על גרפים לא מכוונים.
* על מנת לבחון תהליך הסריקה, האלגוריתם צובע כל קודקוד ב-3 צבעים: לבן, אפור, שחור. בשלב התחלתי כל הקודקוד צובע בלבן משמעתו של טרם התגלה, במהלך האלגוריתם קודקוד הופך להיות אפור שמשמעתו התגלה ולא טופל סופי ולאחר שחור שמשמעותו טופל סופי.
* חיפוש לרוחב מכונה כך מפני שהאלגוריתם סורק ברוחב ומגלה את כל הקדקודים הנמצאים במרחק k מ- sלפני שהוא מגלה איזשהו קדקוד שנמצא במרחק k+1 מ-s.
* וכך האלגוריתם בונה **"עץ רוחב" (“breadth-first tree”)** ששורשו s והוא מכיל את כל הקדקודים הישיגים מ-s. בהתחלה הוא מתגלה קודקוד s לאחר מכן כל פעם שמתגלה קודקוד לבן ν בעקבות המעבר לשכנים של קודקוד u, קודקוד v וצלע (u,v) מוספים לעץ אפשר לומר ש-u הוא אבא (parent/predecessor) של v בעץ. היות וכל קודקוד מתגלה פעם אחת, יש לו מקסימום אבא אחד.
* עבור כל קדקוד ν שניתן להגיע אליו מ-s, המסלול שיוצא מ- sל-ν בעץ הרוחב הוא המסלול הקצר ביותר מ-s ל-ν ב- G כלומר מסלול שמכיל מספר הצלעות המינימאלי.

**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)**

1. **for** each vertex u in V[G]
2. color[u] = WHITE
3. d[u] = nil
4. p[u] = nil
5. **end for**
6. color[s] = GRAY
7. d[s] = 0
8. p[s] = nil
9. Q = empty// queue Q manages the set of gray vertices
10. Q.add(s)
11. **while** (Q != EMPTY)
12. u = Q.dequeue()
13. **for** each vertex v in Adj[u]
14. **if** (color[v] == WHITE)
15. color[v] = GRAY
16. d[v] = d[u] + 1
17. p[v] = u
18. Q.add(v)
19. **endif**
20. **endfor**
21. color[u] = BLACK
22. **end while**
23. **return** (d, p)

**end** **BFS**

**סיבוכיות האלגוריתם**

לאחר האתחול אף קודקוד לא נצבע בלבן, כך שהבדיקה בשורה 14 מבטיחה שכל קודקוד הוכנס לתור מקסימום פעם אחת ולכן גם יוצא ממנו מקסימום פעם אחת.

תור מנהל קודקודים בצבע אפור בלבד.

כל פעולת הכנסה לתור או הוצאה ממנו מתבצעת בזמן .O(1) ולכן בסה"כ זמן הכולל של כל פעולות התור הוא יהיה O(|V|). אז מקבלים |V| קודקודים ועליהם מטפלים: בכל האיתרציה עוברים על השכנים בשורה 13 וסיבוכיות כאן תהיה O(deg(V)). ובסה"כ אם נסכם על כל ה-while (שורות 11-22), עוברים על כל הצלעות בגרף ולכן סיבוכיות תהיה O(|E|).

בנוסף אתחול מתבצע בזמן O(|V|) . ולפיכך זמן הריצה הכולל של האלגוריתם הוא O(|E|+|V|).

**דוגמה להרצת האלגוריתם:**

**int p [] : [-1, -1, -1, -1, -1, -1, -1, -1]**

**int d [] : [-1, -1, -1, -1, -1, -1, -1, -1] ∞ ~ nil ~ -1**

**int color[] : [ 1, 1, 1, 1, 1, 1, 1, 1]**

**queue<Integer> Q : []**

**1 2 4 6**

**∞**

**∞**

**∞**

**∞**

**∞**

**∞**

**∞**

**∞**

**0 3 5 7**

**s = 2**

**1.**

**1 2 4 6**

**0**

**∞**

**∞**

**∞**

**∞**

**∞**

**∞**

**∞**

**0 3 5 7**

**Q: [ 2 ]**

**d: 0**

**2.**

**1 2 4 6**

**1**

**0**

**∞**

**∞**

**1**

**∞**

**∞**

**∞**

**0 3 5 7**

**Q: [ 3, 1]**

**d: 1 1**

**3.**

**1 2 4 6**

**2**

**1**

**0**

**∞**

**2**

**1**

**∞**

**∞**

**0 3 5 7**

**Q: [ 1, 4, 5 ]**

**D: 1 2 2**

**1 2 4 6**

**1**

**2**

**0**

**∞**

**2**

**2**

**1**

**∞**

**0 3 5 7**

**Q: [ 4, 5, 0 ]**

**D: 2 2 2**

**5.**

**1 2 4 6**

**3**

**2**

**1**

**0**

**2**

**2**

**1**

**∞**

**0 3 5 7**

**Q: [ 5, 0, 6 ]**

**D: 2 2 3**

**6.**

**1 2 4 6**

**3**

**2**

**1**

**0**

**3**

**2**

**2**

**1**

**0 3 5 7**

**Q: [ 0, 6, 7 ]**

**D: 2 3 3**

**7.**

**1 2 4 6**

**3**

**2**

**1**

**0**

**2**

**3**

**2**

**1**

**0 3 5 7**

**Q: [ 6, 7 ]**

**D: 3 3**

**8.**

**1 2 4 6**

**3**

**2**

**1**

**0**

**2**

**3**

**2**

**1**

**0 3 5 7**

**Q: [ 7 ]**

**D: 3**

**9.**

**1 2 4 6**

**3**

**2**

**1**

**0**

**3**

**2**

**2**

**1**

**0 3 5 7**

**Q: ᴓ**

**1 2 4 6**

**3**

**2**

**0**

**1**

**3**

**2**

**1**

**2**

**0 3 5 7**

בסיום האלגוריתם :

* כל צומת 𝑣 מחזיק את
* עבור צמתים 𝑢 בלתי ישיגים מ 𝑠 , *.*

במהלך סריקת הגרף BFS מתקבל עץ ה-BFS:

* קשת העץ היא קשת אשר סריקתה גרמה לגילוי צומת חדש בעץ סריקת ה-BFS.
* לעיתים נוח להסתכל על עץ ה BFS לפי רמותיו.
  + כל רמה 𝑖 מכילה את אוסף הצמתים 𝑣 כך ש

**An adjacency-list representation of G**

**מבנה הגרף:**

הגרף נתון כווקטור של רשימות מקושרות. כל רשימה מכילה קדקוד וכל הקדקודים הקשורים אליו. מספר קדקודי הגרף נסמן ב- n. ב-java ArrayList שמכיל בתוכו ArrayList בתור איבר המערך.  
דוגמה: הגרף שלמעלה מוצג בצורה הבאה:

|  |  |
| --- | --- |
| **vertex** | **adjacency vertexes** |
| **0** | **1** |
| **1** | **0 2** |
| **2** | **1 3** |
| **3** | **2 4 5** |
| **4** | **3 5 6** |
| **5** | **3 4 6 7** |
| **6** | **4 5 7** |
| **7** | **5 6** |

**מסקנות:**

לאחר הרצת האלגוריתם BFS(G,s) בגרף G=(V,E)

* כל רכיב הקשירות של s התגלה.
* מערך d מכיל את המסלולים הקצרים ביותר מ-s לכל קודקוד.
* דרך מערך p ניתן לשחזר מסלול קצר ביותר מ-s לכל קודקוד ומכל קודקוד ל-s.

**BFS פותר בעיות הבאות:**

1. **בדיקה אם גרף קשיר:**

* רעיון אחד: לבדוק האם לאחר הרצת האלגוריתם כל הקודקודים נצבע בשחור.
* רעיון שני: לבדוק האם יש רק nil אחד במערך p: p[u] =nil.

נראה את האלגוריתם:  
עוברים על מערך של מרחקים d ובודקים אם נשאר קדקוד u כלשהו בלתי ישיג מ 𝑠 (d[u]==nil). כאשר קדקוד כזה קיים, הגרף אינו קשיר. כאשר מרחק של כל קדקודי הגרף עד המקור s הוא מספר סופי, הגרף הוא קשיר.

* רעיון שלישי: לבדוק האם כל המרחקים במערך d הם מספרים סופיים.

1. **מציאת / הדפסת המסלול הקצר ביותר בין שני קדקודי הגרף**

**גרסה איטרטיבית:**

// The following procedure returns out the vertices on a shortest path from *s* to *v*,

// assuming that BFS has already been run to compute the shortest-path tree.

**String getPath (s, v)**

if (distance [v]==nil) return null

path = ""

if (v==s) path = s

else

path = v + path

t = parent [v]

while (t != nil)

path = t + "->" + path

t = parent [t]

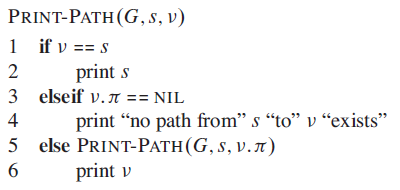
end-while

end-if

return path

**end-getPath**

**גרסה רקורסיבית:**

****

וסיבוכיות כאן תהיה O(|V|) **–** כי עוברים |V|קודקודים אחורה מ-v ל-s

1. **חישוב מספר רכיבי קשירות:**

רעיון אחד:  
1. אתחול: מגדירים קדקוד התחלתי שהוא קדקוד ראשון ברשימה : index = 0  
 מגדירים מספר רכיבי קשירות של הגרף : count = 1   
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 את הערך של indexל- i וחוזרים לשלב 2.  
 ♦ במקרה שקדקוד כזה כבר לא קיים, הפונקציה מחזירה את מספר רכיבי הקשירות המחושב (count).

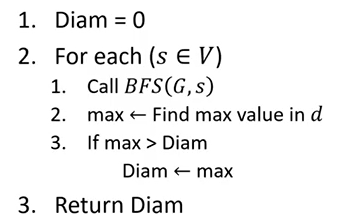
רעיון שני: אם לאחר הרצת האלגוריתם יש עוד קודוקדים לבנים, נפעיל את האלגוריתם מחדש על אחד מהלבנים ונספור שזו איטרציה שניה... וכן הלאה עד שכל הקודקודים נצבעים בשחור. שבכל הרצת BFS מגלים קודקוד מסוים, אז חושפים את רכיב הקשירות שלו, אז מספר האיטרציות שהפעלת BFS על קודקודים לבנים יהיה שווה לכמות רכיבי הקשירות בגרף.

1. **מציאת רכיבי קשירות עצמם** **(מיהם הקודקודים בכל רכיב קשירות).**

האלגוריתם מאוד דומה לאלגוריתם המתואר בסעיף הקודם:

כך שבכל האיטרציה מסמנים (שומרים) את הקודוקדים שבכל רכיב.  
ברעיון ראשון: חוזרים לסעיפים 1,2 ולסעיף 3 מוסיפים שמירת קדקודים הנמצאים באותו רכיב הקשירות.

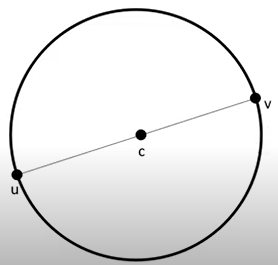
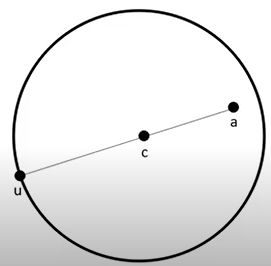
אפשר להיעזר במערך נוסף/עזר בגודל מספר הקודקודים כך שכל פעם שחושפים קודקוד מסוים נסמן במערך את מספר האיטרציה במיקום הקודקוד שנחשף, אז כל הקודקודים שנחשפו באיטרציה ראשונה יהיה להם ערך במערך 1 וכך האלה. בסיום אפשר לעבור על המערך ולומר איזה קודקוד נמצא באיזה רכיב הקשירות.

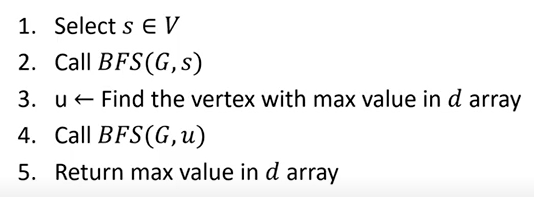
1. **חישוב קוטר הגרף – מרחק המקסימלי שקיים בין שני קודקודים בגרף:**

**גרסה ראשונה:**

וסיבוכיות כאן תהיה O(|V|\*(|V|+|E|))

**גרסה שניה – נובעת מגיאומטריה במישור:**



* 1. מפעיליםBFS על קדקוד ראשון ברשימה,
  2. מחשבים את אינדקס (ind) של קדקוד שמרחקו עד הקדקוד הראשון ברשימה גדול ביותר.
  3. שוב מפעילים BFS על קדקוד ind ומחשבים את אינדקס (ind1) של קדקוד שמרחקו עד הקדקוד ind גדול ביותר.
  4. המרחק (מספר צלעות) מקדקוד ind עד קדקוד ind1 הוא הקוטר של העץ.

וסיבוכיות כאן תהיה O(|V|+|E|)

**משימות:**

1. לממש BFS.
2. לכתוב פונקציה שבודקת האם גרף קשיר.
3. לכתוב פונקציה שמחזירה כמה רכיבי הקשירות יש בגרף.
4. לכתוב פונקציה שמדפיסה מיהם הקודקודים בכל רכיב הקשירות.
5. לכתוב פונקציה שמחזירה מרחק הקצר ביותר בין שני קודוקדים נתונים.
6. בהמשך לסעיף הקודם להדפיס את המסלול ביניהם.
7. לכתוב פונקציה מחזירה קוטר של גרף.
8. בהמשך לסעיף הקודם, להדפיס דוגמא למסלול בגודל הקוטר.