The DFS (Depth First Search) Algorithm

חיפוש לעומק-תחילה

במדעי המחשב, **אלגוריתם חיפוש לעומק** (אנגלית: **Depth First Search**, ראשי תיבות: **DFS**) הוא אלגוריתם המשמש למעבר על גרף או לחיפוש בו.

אינטואיטיבית, האלגוריתם מתחיל את החיפוש מצומת שרירותי בגרף ומתקדם לאורך הגרף עד אשר הוא נתקע, לאחר מכן הוא חוזר על עקבותיו עד שהוא יכול לבחור להתקדם לצומת אליו טרם הגיע. דרך פעולת האלגוריתם דומה במידת מה לסריקה שיטתית של מבוך.

כמו בחיפוש לרוחב, גם בחיפוש לעומק נצבעים קדקודים במהלך החיפוש כדי לציין את מצבם. כל קדקוד צבוע בתחילה לבן, הוא נצבע באפור כאשר הוא מתגלה במהלך החיפוש, והוא נצבע בשחור כאשר הטיפול בו מסתיים, כלומר כאשר רשימת הסמיכות שלו נבדקה במלואה.

כמו בחיפוש לרוחב, בכל פעם שמתגלה קדקוד v במהלך הסריקה של רשימת הסמיכות של קדקוד u שכבר התגלה, החיפוש לעומק מציין אירוע זה על-ידי הצבת u, הקודם של v, בשדה $\text{pred}[v]$. שלא כמו חיפוש לרוחב, אשר תת-גרף הקודמים שלו יוצר עץ, תת-גרף הקודמים הנוצר על-ידי חיפוש לעומק עשוי להיות מורכב מכמה עצים, שכן החיפוש עשוי להתנהל מכמה מקורות. **תת-גרף הקודמים** של חיפוש לעומק יוצר יער עומק (depth-first forest) המורכב מכמה **עצי עומק** (depth-first trees).

בנוסף, חיפוש לעומק שומר בכל קדקוד **חותמות זמן** (timestamps) . לכל קדקוד v יש שתי חותמות זמן: הראשונה, firstTime[v] מייצגת מתי התגלה V לראשונה (ונצבע באפור), השנייה lastTime[v] מייצגת מתי סיים החיפוש לבחון את רשימת הסמיכות של v (וצבע אותו בשחור). חותמות זמן אלה משמשות באלגוריתמי גרפים רבים, ובאופן כללי מסייעות בניתוח התנהגותו של חיפוש לעומק. פונקציה dfs שלהלן שומרת את מועד הגילוי של קדקוד u במשתנה firstTime [u] ואת מועד סיום הטיפול בקדקוד u במשתנה lastTime [u]. חותמות אלה הם ערכים שלמים בין 1 ל-2|V|, שכן כל אחד מ-|V| הקדקודים מתגלה פעם אחת והטיפול בו מסתיים פעם אחת. לכל קדקוד מתקיים: firstTime [u]< lastTime [u]. קדקוד u הוא לבן לפני זמן firstTime [u], אפור בין הזמן firstTime [u] לזמן lastTime [u] ושחור אחרי הזמן lastTime [u]. הפסאודו-קוד שלהלן הוא האלגוריתם הבסיסי לחיפוש לעומק. גרף G יכול להיות מכוון או בלתי-מכוון. המשתנה time הוא משתנה גלובאלי המשמש לניהול חותמות-הזמן.

```
dfs(G)
1 for each vertex u ∈ V[G]
2     color[u] = WHITE
3     pred [u]= NIL
4 time = 0
5 for each vertex u ∈ V[G]
6      if color[u] == WHITE
7           then dfs_visit(G,u)
```

```
dfs_visit(G,u)
1 color[u] = GRAY // White vertex u has just been discovered.
2 time = time+1
3 d[u] = time
4 for each v ∈ Adj[u]  // Explore edge (u, v).
5     if color[v] == WHITE
6           pred [v] = u
7           dfs_visit (v)
8 color[u] = BLACK // Blacken u; it is finished.
9 f [u] =++time
```

התכנית פועלת כדלקמן:

<u>שורות 1-3</u> צובעות את כל הקדקודים בלבן ומאתחלות את השדות π שלהם לערך NIL.

<u>שורה 4</u> מאפסת את מונה הזמן הגלובאלי.

<u>שורות 5-7</u> בודקות כל קדקוד ב-v בתורו, וכאשר נמצא קדקוד לבן, נערך בו "ביקור" באמצעות **dfs_visit**.

בכל פעם שמתבצעת בשורה 7 קריאה ל-(u) **dfs_visit** , u הופך לשורש של עץ חדש ביער העומק.

כאשר **dfs** חוזרת, בכל קדקוד u כבר הוצב מועד גילוי firstTime [u] ומועד סיום lastTime [u].

בכל קריאה ל-(u) **dfs_visit** , הקדקוד u הוא בתחילה לבן.

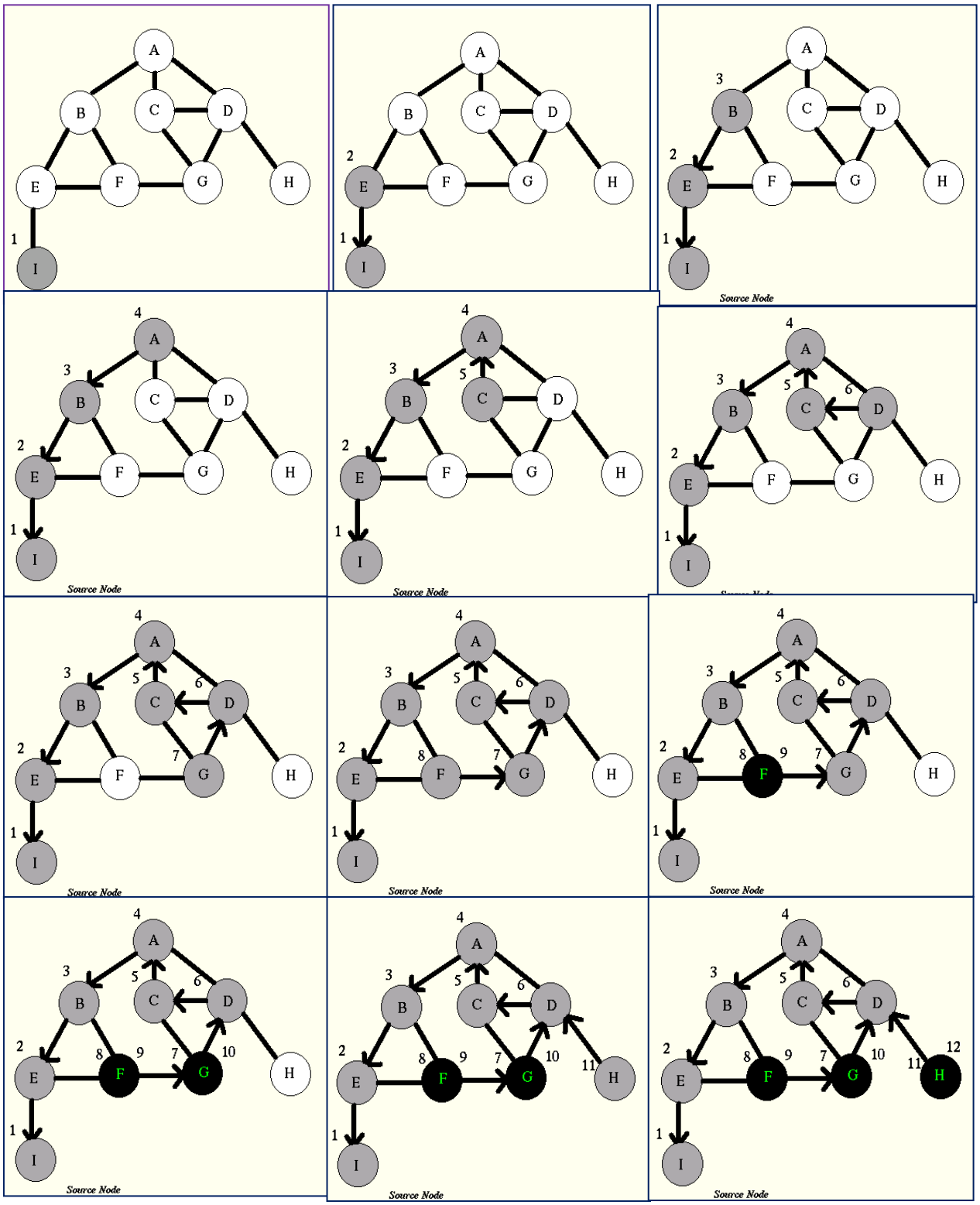
<u>שורה 1</u> של **dfs_visit** צובעת את u באפור

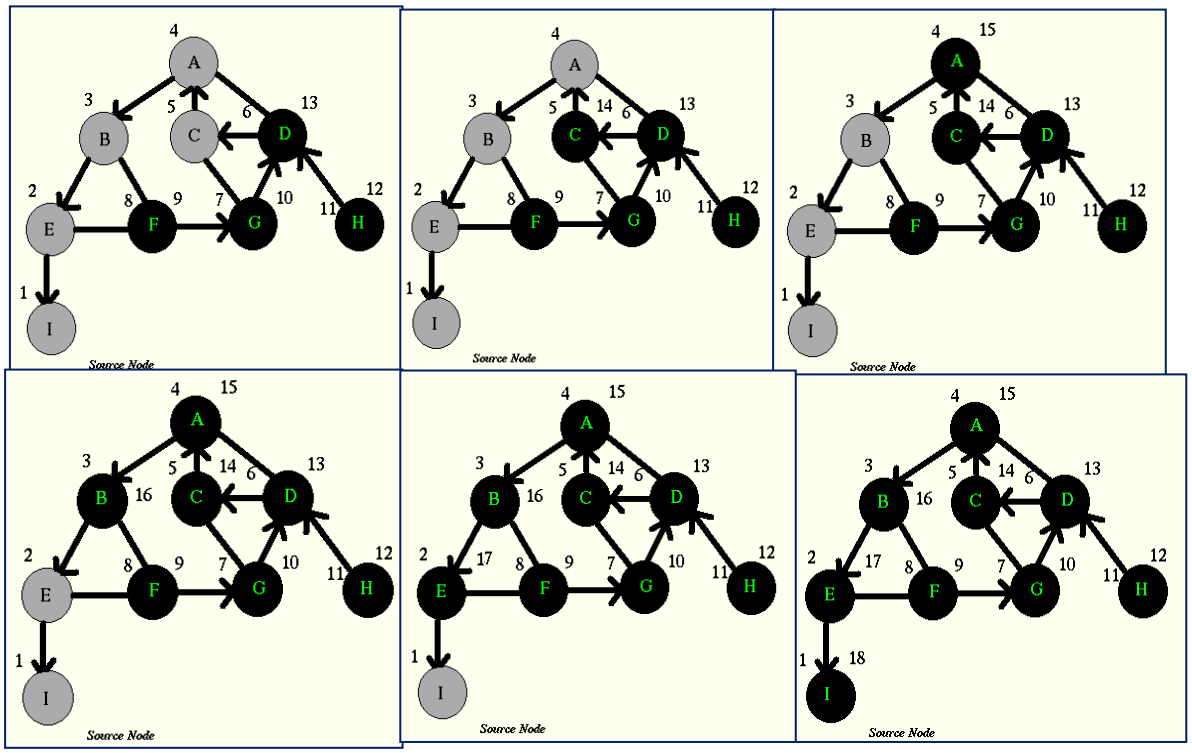
<u>שורה 2</u> מחשבת את מועד הגילוי של u על-ידי הוספת 1 למשתנה גלובאלי *time*, ומציבה אותו ב- firstTime [u].

<u>שורות 3-6</u> בוחנות כל קדקוד v הסמוך ל-u ואם v לבן, עורכות בו ביקור באפן רקורסיבי. בכל פעם שקדקוד $v \in Adj[u]$ נבחן בשורה 5, אנו אומרים שצלע (u,v) נבדקת על-ידי החיפוש לעומק.

<u>שורות 7-8</u> . לאחר שנבדקו כל הצלעות היוצאות מ-u שורות אלו צובעות את u בשחור, ומציבות ב- lastTime [u] את מועד הסיום.

The DSF (Depth-first search) Algorithm Example





שימושים

לאלגוריתם החיפוש לעומק יתרונות על אלגוריתם חיפוש לרוחב אם קיימת ידע קודם או אינטואיציה שמסוגלת לעזור לחיפוש. למשל, בחיפוש היציאה ממבוך, אם יופעל חיפוש לרוחב מאמצע המבוך תימצא היציאה רק בשלב האחרון של האלגוריתם. לעומת זאת, בחיפוש לעומק ניתן יהיה למצוא את היציאה כבר בתחילת ריצת האלגוריתם, עוד לפני שיהיה עליו לשוב על עקבותיו, ובפרט אם יש לו דרך טובה להעריך מהו הכיוון הנכון של היציאה.
לעומת זאת, אלגוריתם חיפוש לעומק סובל מחסרונות כאשר הגרפים עליהם הוא פועל הם גדולים מאוד. בפרט, בגרפים בעלי מסלולים אינסופיים, האלגוריתם עלול לא לעצור גם אם האיבר שהוא מחפש נמצא בגרף (כי המסלול שבו הוא יבחר עלול להיות מסלול אינסופי שבו האיבר שמחפשים לא נמצא, ואז האלגוריתם יתקדם ללא הפסקה באותו מסלול מבלי לחזור על עקבותיו), וזאת להבדיל מחיפוש לרוחב, שמובטח לו שיגיע מתישהו אל האיבר שאותו מחפשים.
ניתן להשתמש באלגוריתם חיפוש לעומק בתור בסיס לאלגוריתמים רבים שפועלים על גרפים. למשל, מציאת רכיבי קשירות בגרף, מציאת מסלול אוילרי ומציאת מעגלים בגרף.

טענה 1

dfs –אלגוריתם עובר על כל קדקוד הגרף פעם אחת בלבד.

הוכחה: הפונקציה **dfs_visit** נקרא על קדקוד v רק עם הצבע שלו הוא לבן, והצבע מיד משתנה.

טענה 2 dfs –אלגוריתם עובר על כל צלע הגרף פעם אחת בלבד.

הוכחה: : הפונקציה **dfs_visit** נקרא על קדקוד פעם אחת בלבד (טענה 1) וגוף הלולאה

for each $v \in Adj[u]$

מבוצעת לכל צלע הגרף פעם אחת בלבד.

<u>טענה 3</u> **dfs**-אלגוריתם עובר על כל קדקודי הגרף (כמובן, באותו רכיב קשירות הגרף).

<u>הוכחה:</u> נוכיח את הטענה באינדוקציה לפי מרחק k של קדקוד כלשהו עד המקור.

<u>בסיס</u> האינדוקציה: k=0. האלגוריתם עובר על קדקוד המקור.

<u>הנחת אינדוקציה</u>: האלגוריתם עובר על כל קדקודי הגרף שמרחקם עד המקור שווה k-1.

<u>שלב אינדוקציה</u>: נתבונן בקדקוד v כלשהו שמרחקו עד המקור הוא k. בגלל שקיים מסלול מ-v עד המקור, v קשור בעזרת צלעה קדקוד w שמרחקו עד המקור הוא k-1. אזי האלגוריתם יעבור על v, כאשר הוא עובר על w.

<u>סיבוכיות</u>

הלולאה 1-3 ולולאה 5-7 מתבצעות בסיבוכיות O(|V|). פונקציה **dfs_visit** נקראת בדיוק פעם אחת עבור כך קדקוד $v \in V$, שכן היא נקראת רק קדקודים לבנים. במהלך ביצוע של **dfs_visit**(v), לולאה שבשורות 4-7 מתבצעת $|Adj[v]|$ פעמים. מאחר שמתקיים $\sum_{v \in V} |Adj(v)| = O(E)$ העלות הכוללת של לביצוע שורות 4-7 של **dfs_visit** היא O(|E|). סיבוכיות של DFS היא אפוא O(|E| + |V|).

מציאת מעגל בגרף (מכוון או לא מכוון) – פסאודו-קוד:

```
hasCircle(G)
      ans = false
      for each vertex u ∈ V[G]
            color[u] = WHITE
            pred[u]= NIL
      for each vertex  u in G and ans==false
            if  color[u] == WHITE
                  ans = dfsVisit(u)
      return ans;
end- hasCircle

dfsVisit(int u)
      ans = false
      color[u] = GRAY
      for each vertex  v in Adj(u) and ans==false
            if (color[v]==GRAY and pred[u]≠v)
                  ans = true
                  getCycle(u, v)
            else if  color[v]==WHITE
                  pred[v] = u;
                  ans = dfsVisit(v)
      color[u] = BLACK
      return ans;
end-dfsVisit
```

```
getCycle(int u, int v)
      Stack  cycle
      x = u
      while x ≠ v
            cycle.push(x)
            x = pred[x]
      end-while
            push(cycle , v)
            push(cycle, u)
      reverse(cycle)
end-getCycle

public String dfsPath(int u, int v)
      String ans = null;
      dfs(u);
      if (color[v] != WHITE)
            ans = new String() + v;
            while (v!=u){
                  v = pred[v];
                  ans = v + "->" + ans;
            end-while
      end-if
      return ans;
end-dfsPath
```