גרפים 1

Ido Talbi ; ***University of Haifa***

הגדרה פורמלית לגרפים

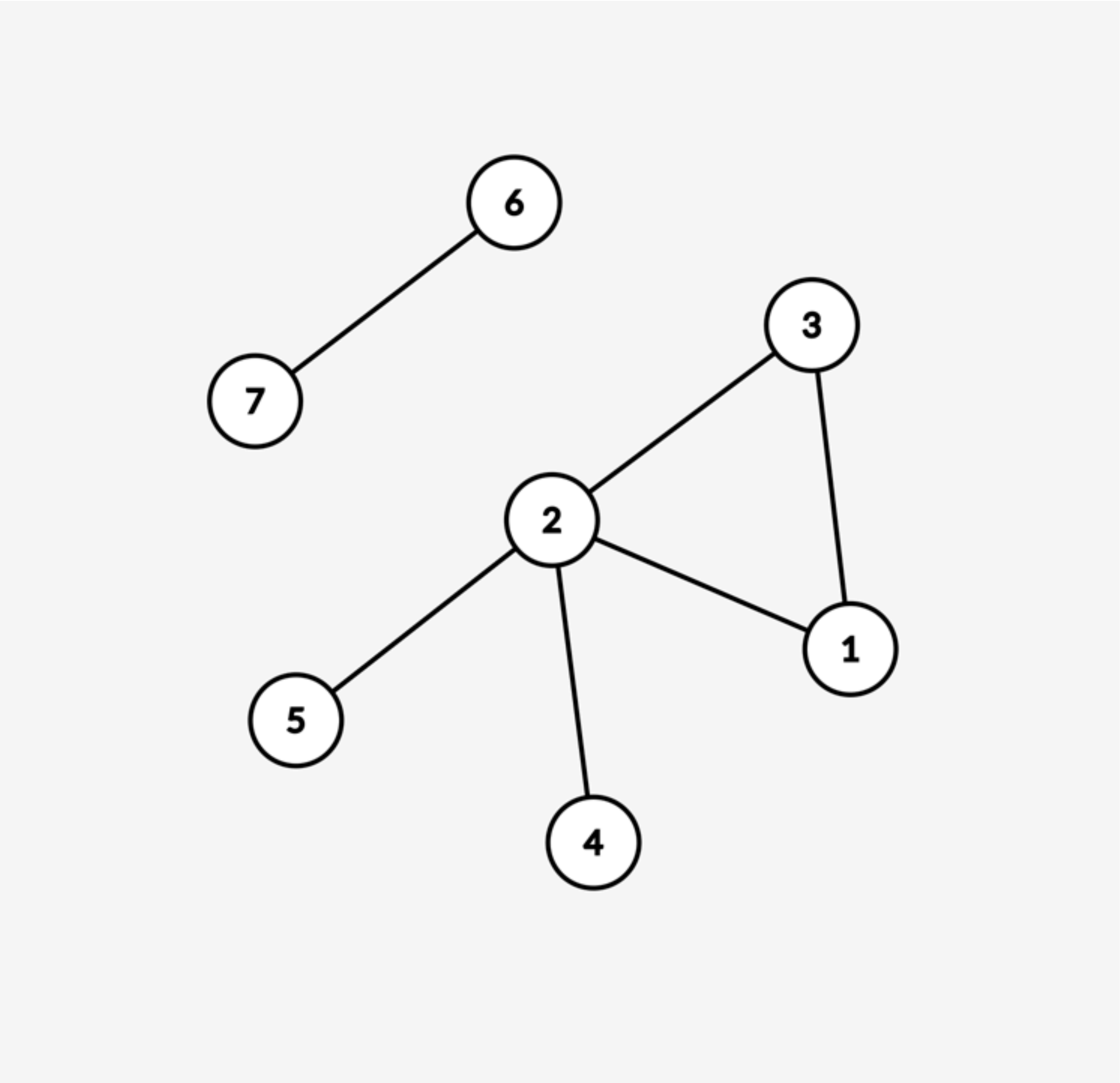
הגדרה: גרף

- יהי V קבוצה של צמתים (vertices) וE קבוצה של קשתות (edges). הזוג $G=(V,E)$ יהיה גרף.

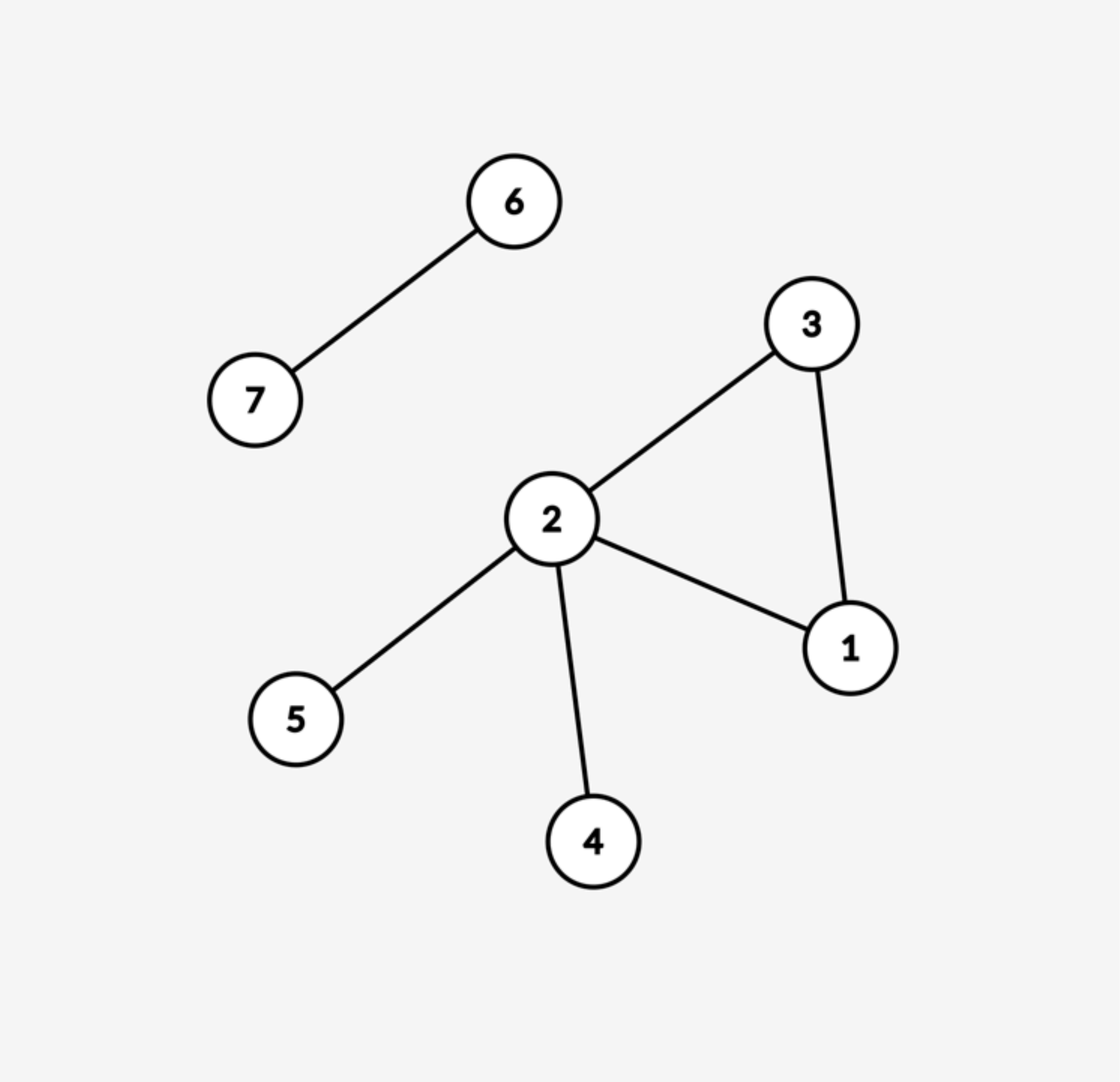
- צמתים: צומת הוא אובייקט ייחודי. כלומר, אין שני צמתים שהם אותו דבר. לדוגמה, נוכל לקרוא לצמתים שלנו $1,2,3$ **או** יניר, נועם, עומר **או** v_1, v_2, v_3. במציאות (במימוש) נשתמש במספרים מכיוון שהם יכולים לשמש גם כאינדקסים.

- קשתות: זוג (לא) מסודר של צמתים. לדוגמה, (v_1, v_2) (קשת מכוונת) **או** $\{$נועם, יניר$\}$ (קשת חסרת כיוון)

- **הערה!** בהגדרה לגרף לא מצויין האם ניתן כפל קשתות. גרף בו אסור כפל קשתות יקרא גרף פשוט, וכמעט תמיד זהו הגרף בו נתעניין.



מה ההגדרה הפורמלית לגרף? (מהם V, E?)



מה ההגדרה הפורמלית לגרף? (מהם V, E?)

$$V = \{1,2,\ldots,7\}$$

$$E = \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{2,5\},\{6,7\}\}$$

ייצוג גרפים בקוד

מבנה נתונים לייצוג גרפים

- ישנם 2 שיטות לייצוג גרפים, מטריצת שכנויות או רשימת שכנויות
- לרוב נשתמש ברשימת שכנויות, למרות שלמטריצה גם יש שימוש
- מעתה נסמן: $n = |V|$, $m = |E|$.

רשימת שכנויות

- לכל צומת v_i נשמור רשימה של הצמתים שיש קשת **שמתחילה** ב-v_i
- יש m קשתות, ולכן בסה"כ בכל הרשימות נשמור m איברים
- יש n צמתים, ואנו שומרים לכל צומת רשימה
- סיבוכיות מקום: $O(n+m)$ (ועוד קבועים של שמירת מבני נתונים)
- בעצם אנו שומרים את המינימום האפשרי
- **חסרון:** בדיקה האם קיימת הקשת (u, v) לוקחת לכל היותר $O(m)$

- ייצוג בקוד:

```
vector<vector<int>> my_graph(n);

my_graph[3].push_back(5); // add the edge (3, 5)
auto adj_4 = my_graph[4]; // all vertices adjacent to 4

for(auto &v : my_graph[3]) if(v == 5) {
    cout << "Edge (3,5) present in my graph!" << endl;
} else {
    cout << "No edge between 3 and 5." << endl;
}
```

- נוכל לייעל את המימוש ע״י שימוש ב`vector<set<int>>`, אבל זה לא הכרחי. (הבדיקה לקשת (u, v) תהיה $O(\log m)$ עכשיו)

מטריצת שכנויות

- נשמור מטריצה בגודל $n \times n$, נקרא לה A
- במיקום הi, j יהיה 1 אם יש את הקשת (v_i, v_j), 0 אחרת
- אם הגרף לא מכוון אז $A_{ij} = A_{ji}$
- סיבוכיות מקום: $\Theta(n^2)$ (!!)
- כמעט ותמיד לא נשתמש במטריצת שכנויות, מכיוון שרק להקצות n^2 מקומות לוקח יותר מדיי זמן. עם זאת יש שאלות שהעניין בהם זה ייצוג במטריצה.

• ייצוג בקוד:

```
constexpr int maxn = 1e5;
bool my_graph[maxn][maxn] = {0};

my_graph[3][5] = true; // add the directed edge (3,5)
my_graph[2][3] = my_graph[3][2] = true; // add the
undirected edge (2,3)

if(my_graph[3][5] && my_graph[5][3]) {
    cout << "Undirected edge between 3, 5!" << endl;
} else if(my_graph[3][5] || my_graph[5][3]) {
    cout << "One-way road between 3, 5." << endl;
}
```

מקצים 1e10 בתים
שזה 10GB (!!)

אלגוריתמים על גרפים (מבוא)

אבל קודם...

למה גרפים שימושיים?

שימושים

- גרפים הם אחד מהנושאים הכי שימושיים במתמטיקה (מדעי המחשב)
- ניתן למדל הרבה מאוד דברים כגרף, או כעץ (סוג מיוחד של גרף עם תכונות חמודות)
- לדוגמה:
  - גרף ממושקל המייצג מפה
  - גרף חברים (פייסבוק)
  - גרף דו-צדדי – התאמה חח"ע (ועל) בין 2 קבוצות שונות – **מאוד** שימושי
  - גרף זרימה (וייז)

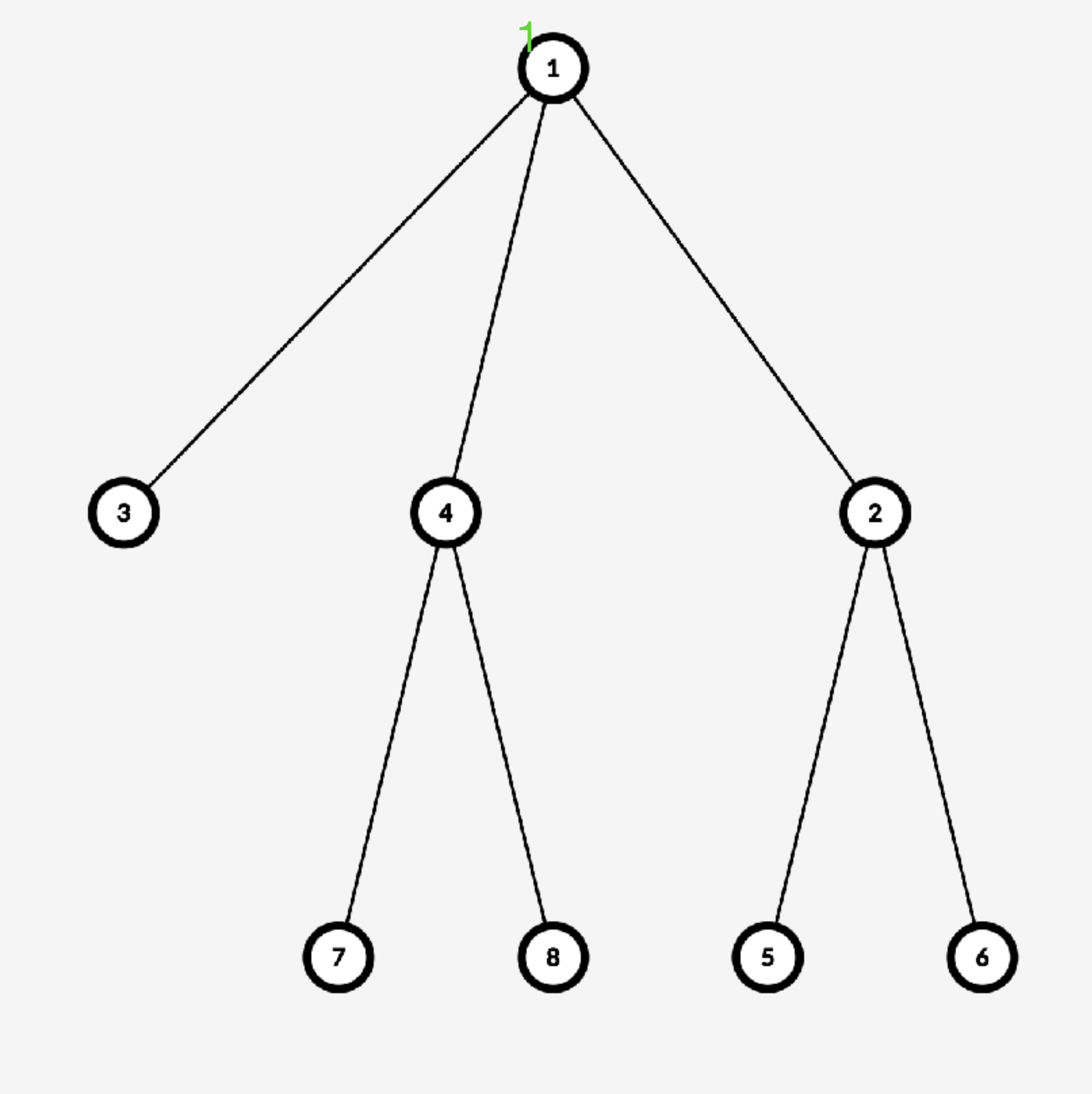
אלגוריתם - DFS

- חיפוש לעומק (depth first search) בגרף
- בביקור מסויים בצומת, נסמן שביקרנו אותו ("נכנסו אליו"), ואז נכנס לכל הבנים בסדר.
- בכל שלב נחפש לעומק (כלומר, ננסה כל הזמן להיכנס לבנים של הבנים), ואחרי זה לרוחב (נכנס לבן הבא)
- בעצם backtracking
- נשמור מחסנית ובכל פעם שנכנס לבן חדש נוסיף אותו למחסנית, וכשנסיים איתו נוציא אותו

למה זה דומה?

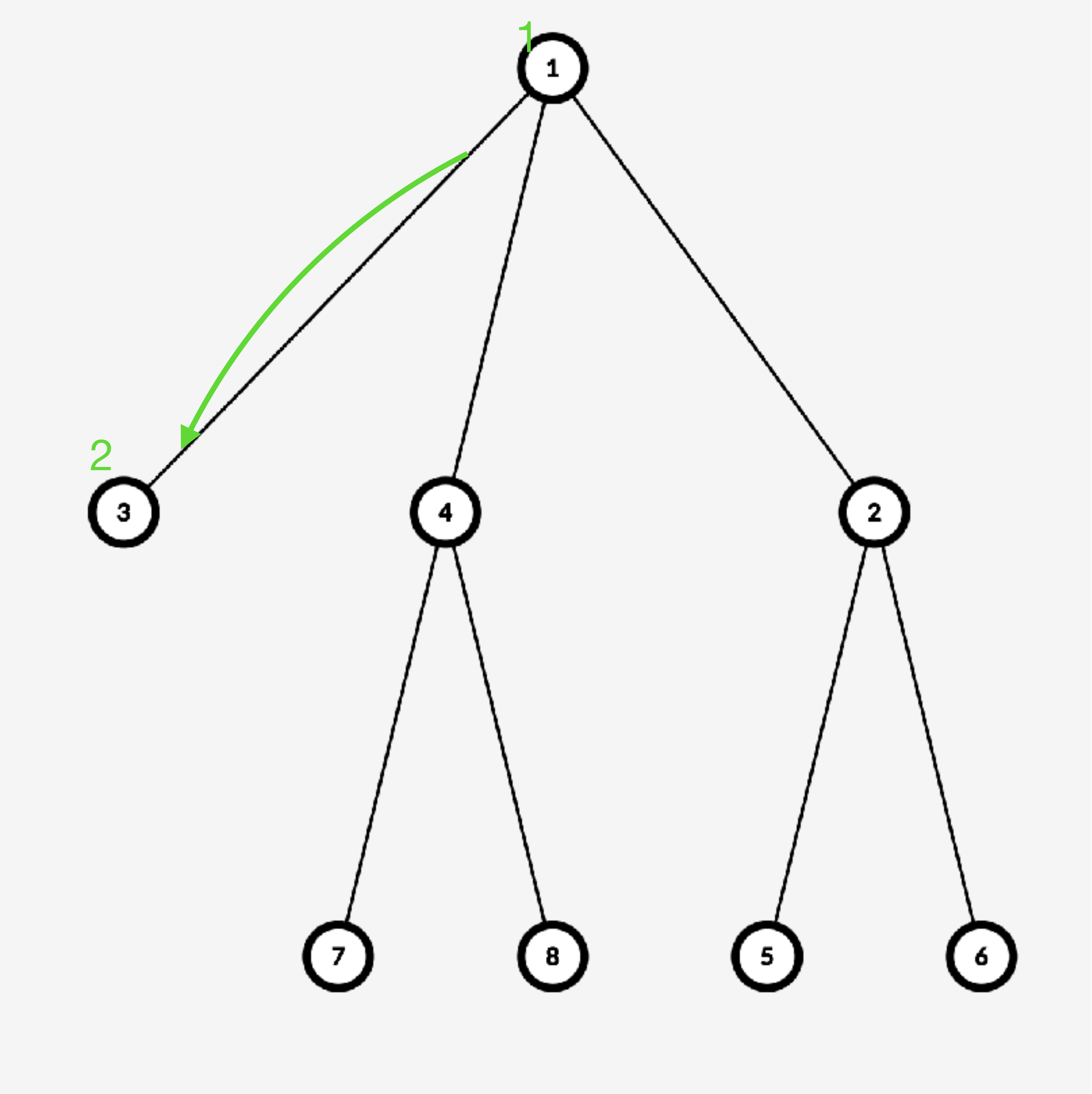
דוגמת הרצה לDFS

call: DFS(1)



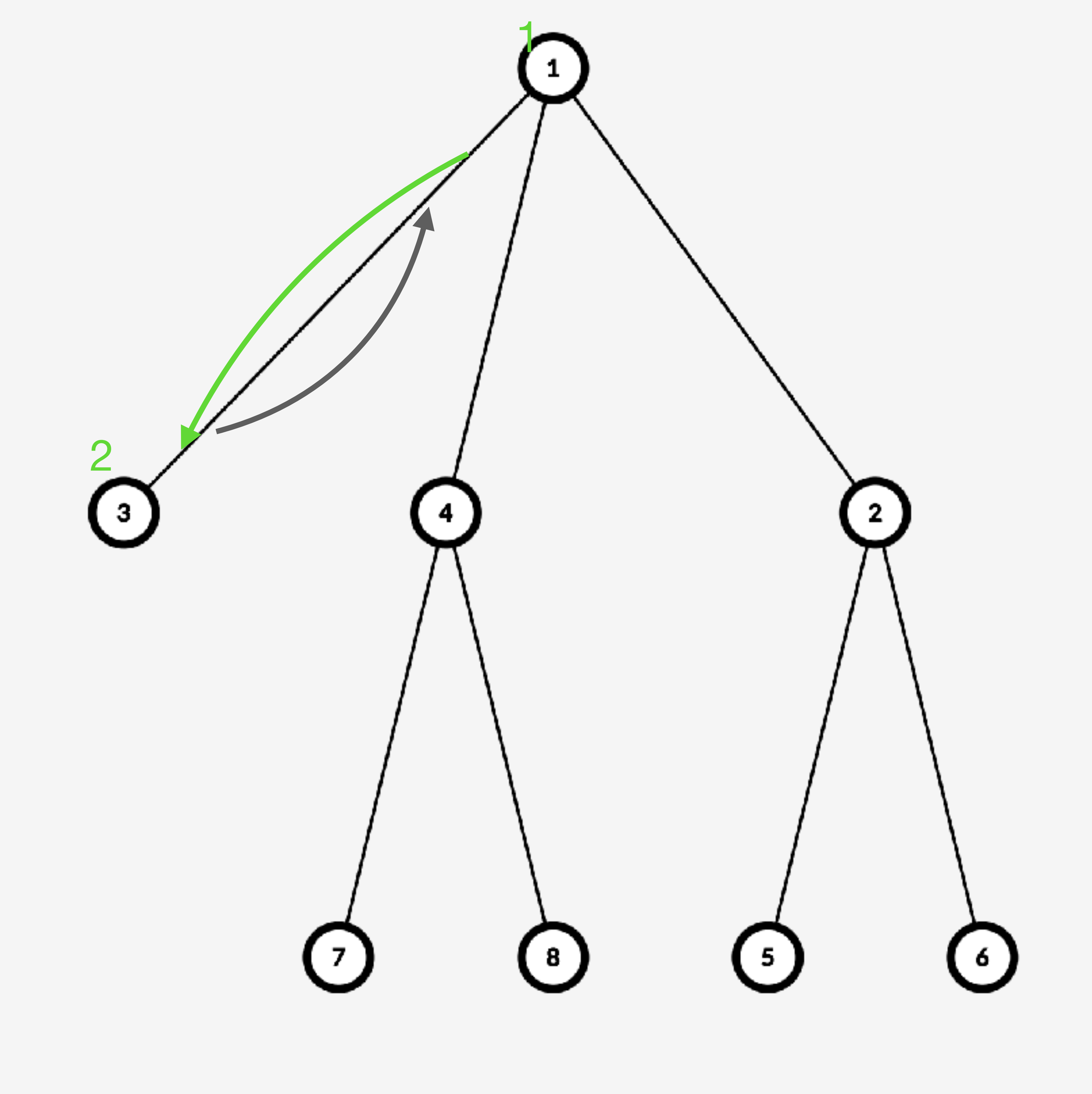
דוגמת הרצה לDFS

call: DFS(1)



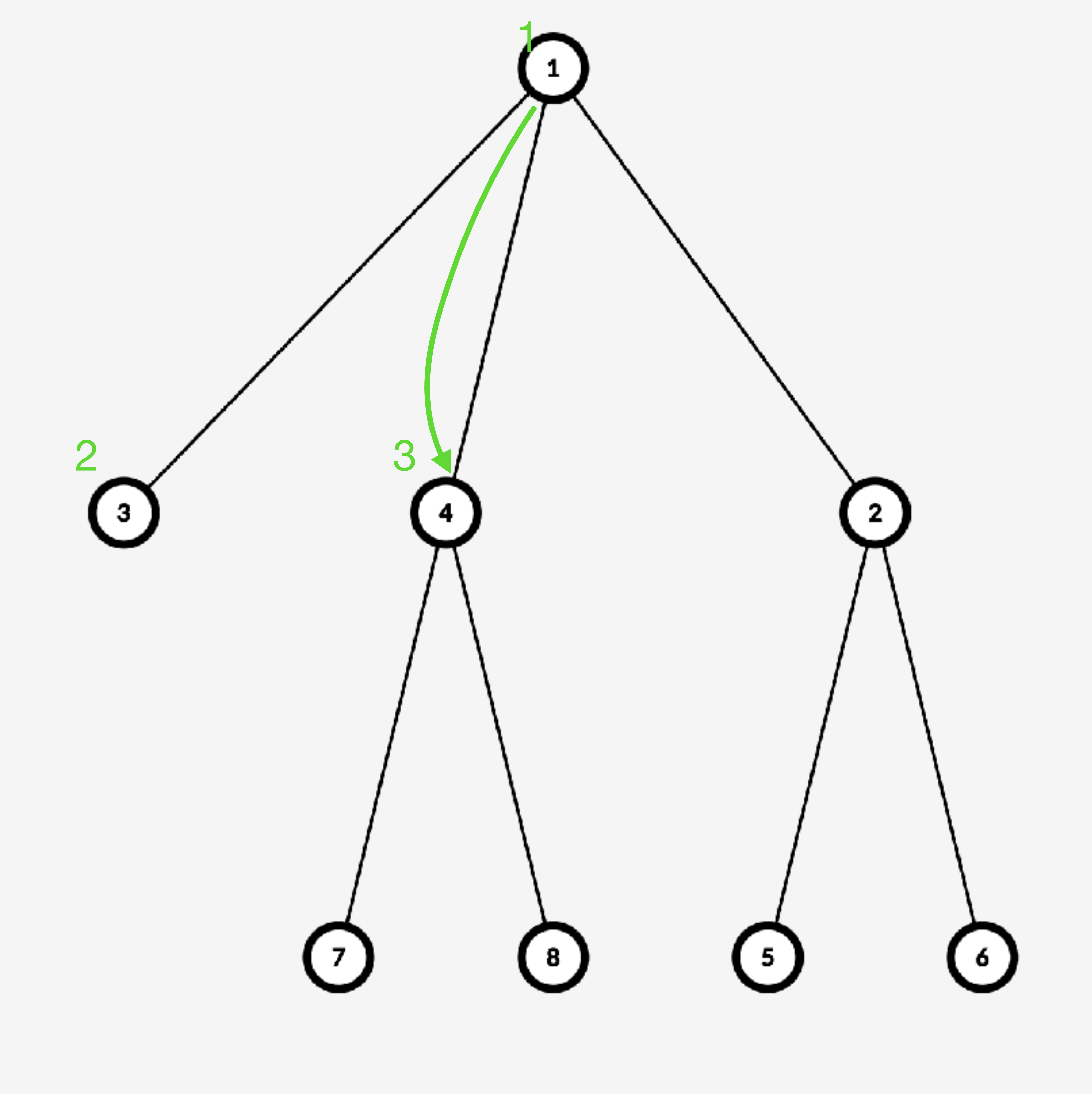
דוגמת הרצה לDFS

call: DFS(1)



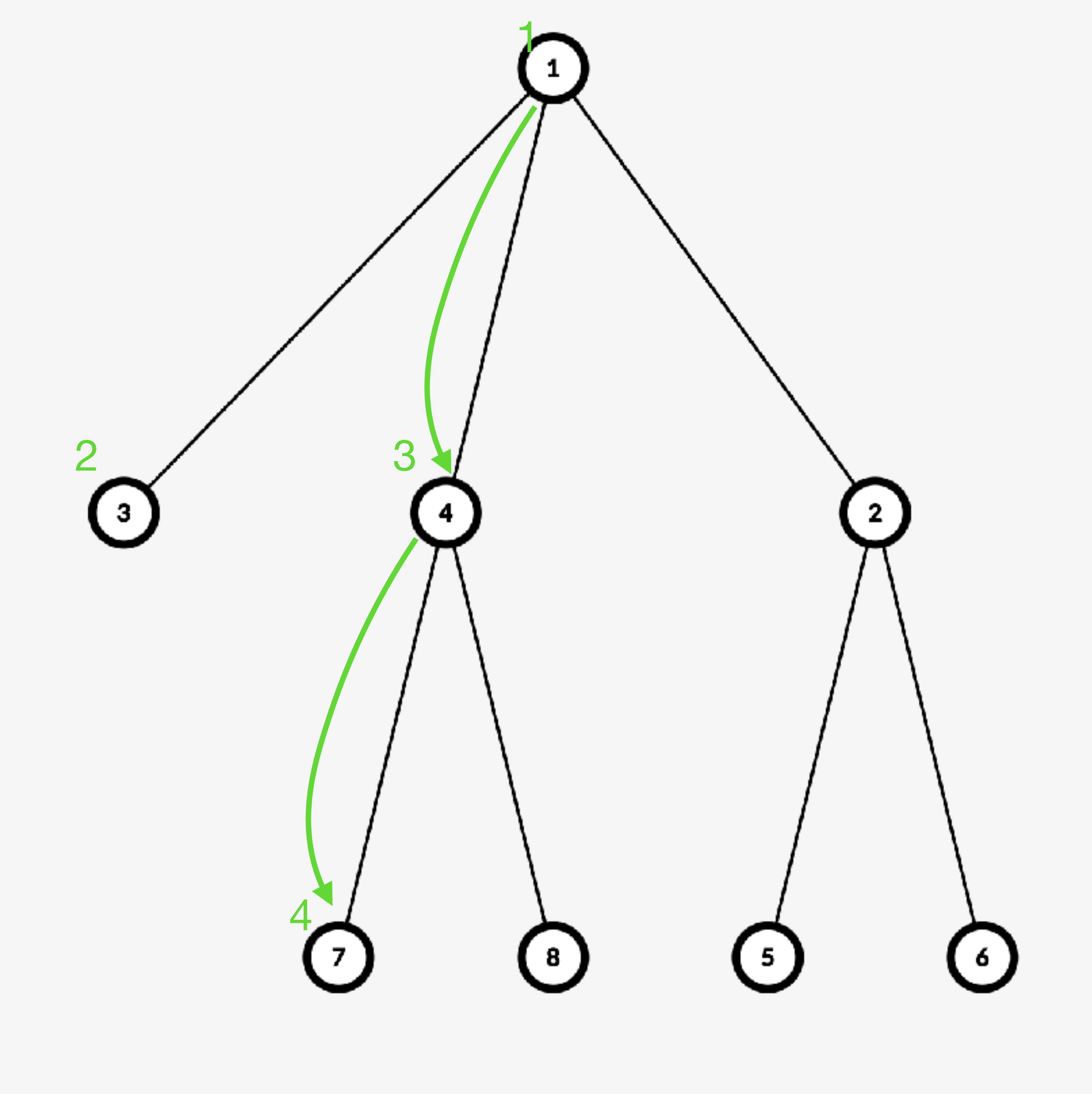
דוגמת הרצה לDFS

call: DFS(1)



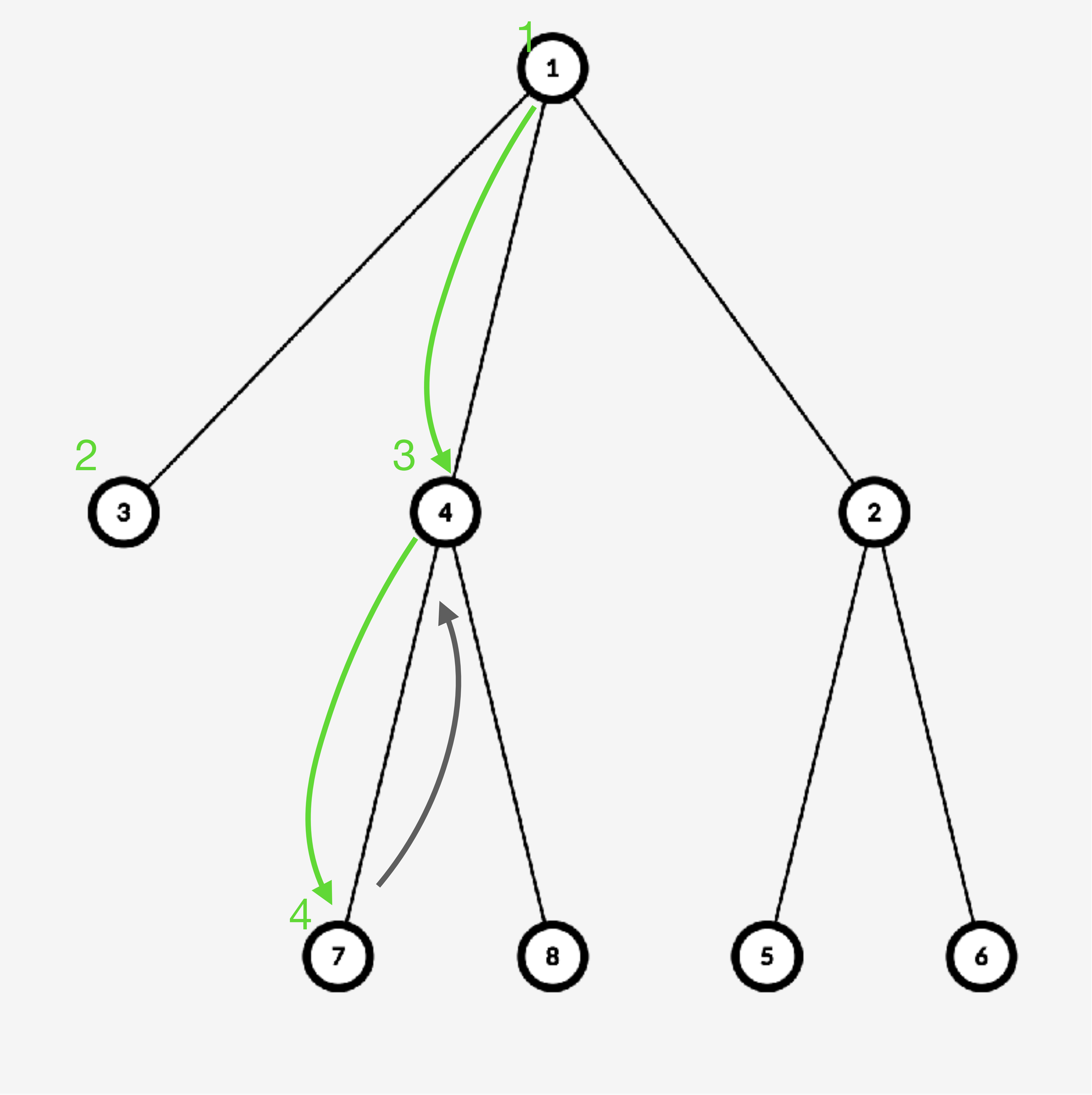
דוגמת הרצה לDFS

call: DFS(1)



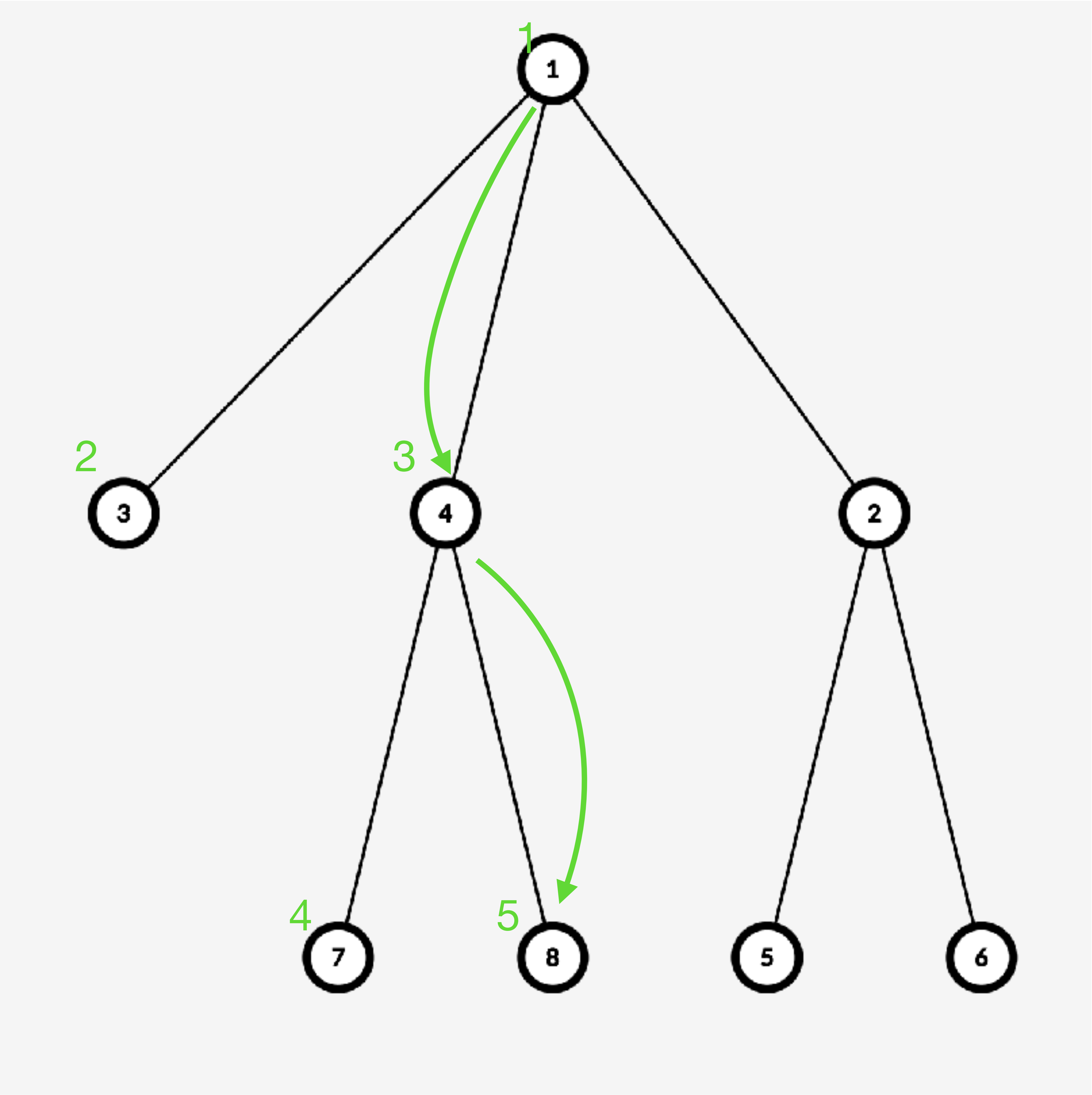
דוגמת הרצה לDFS

call: DFS(1)



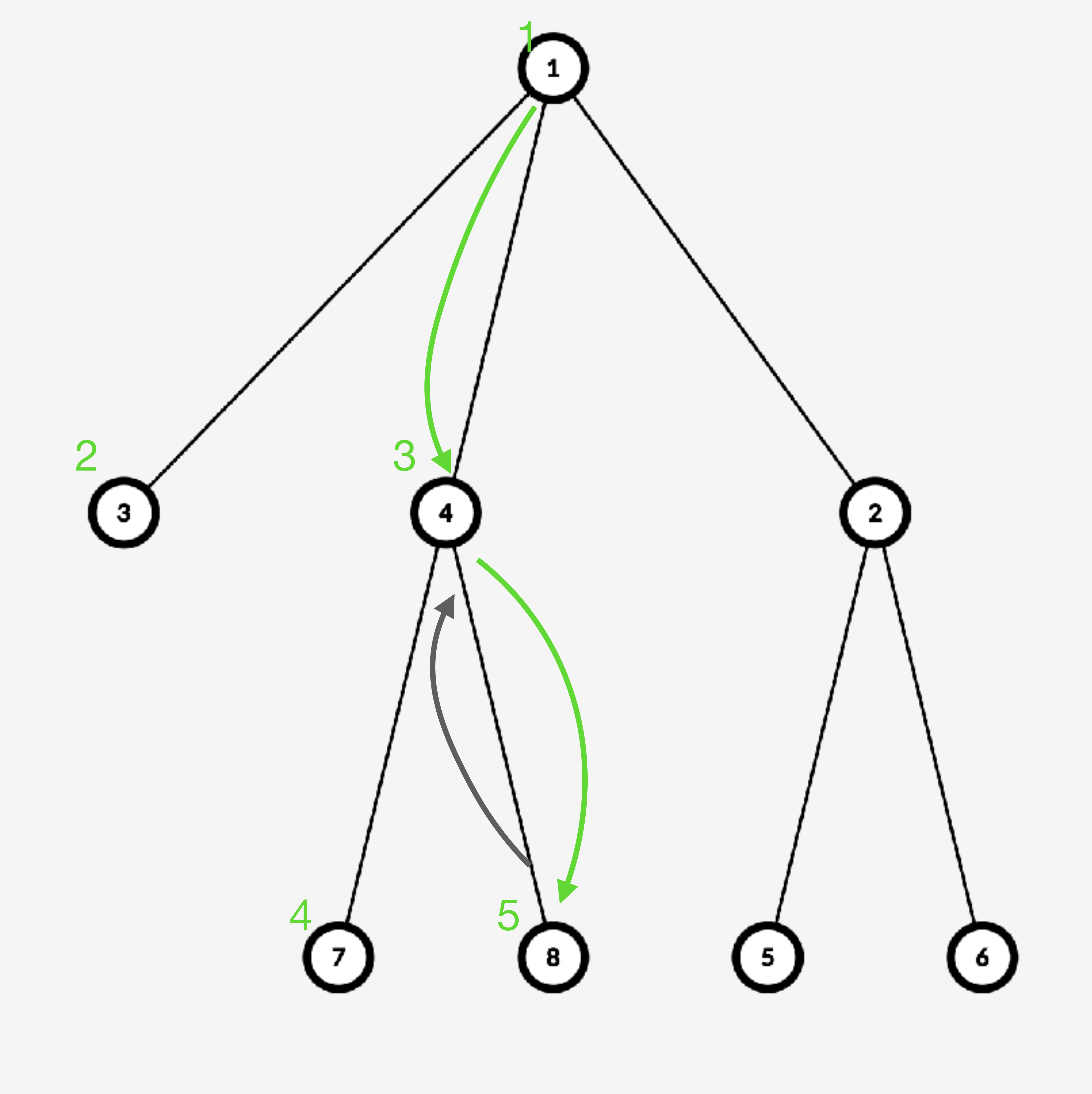
דוגמת הרצה לDFS

call: DFS(1)



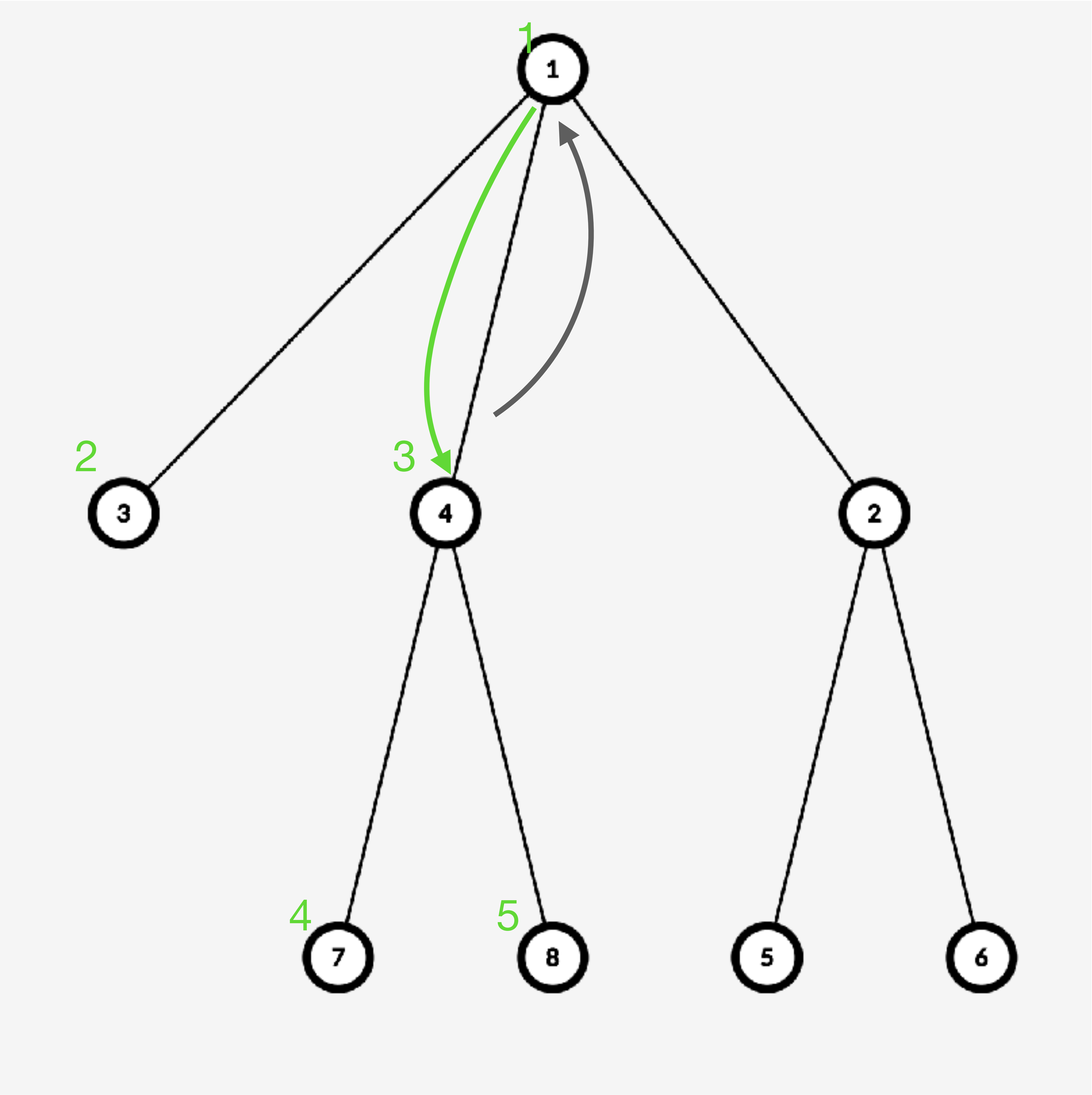
דוגמת הרצה לDFS

call: DFS(1)



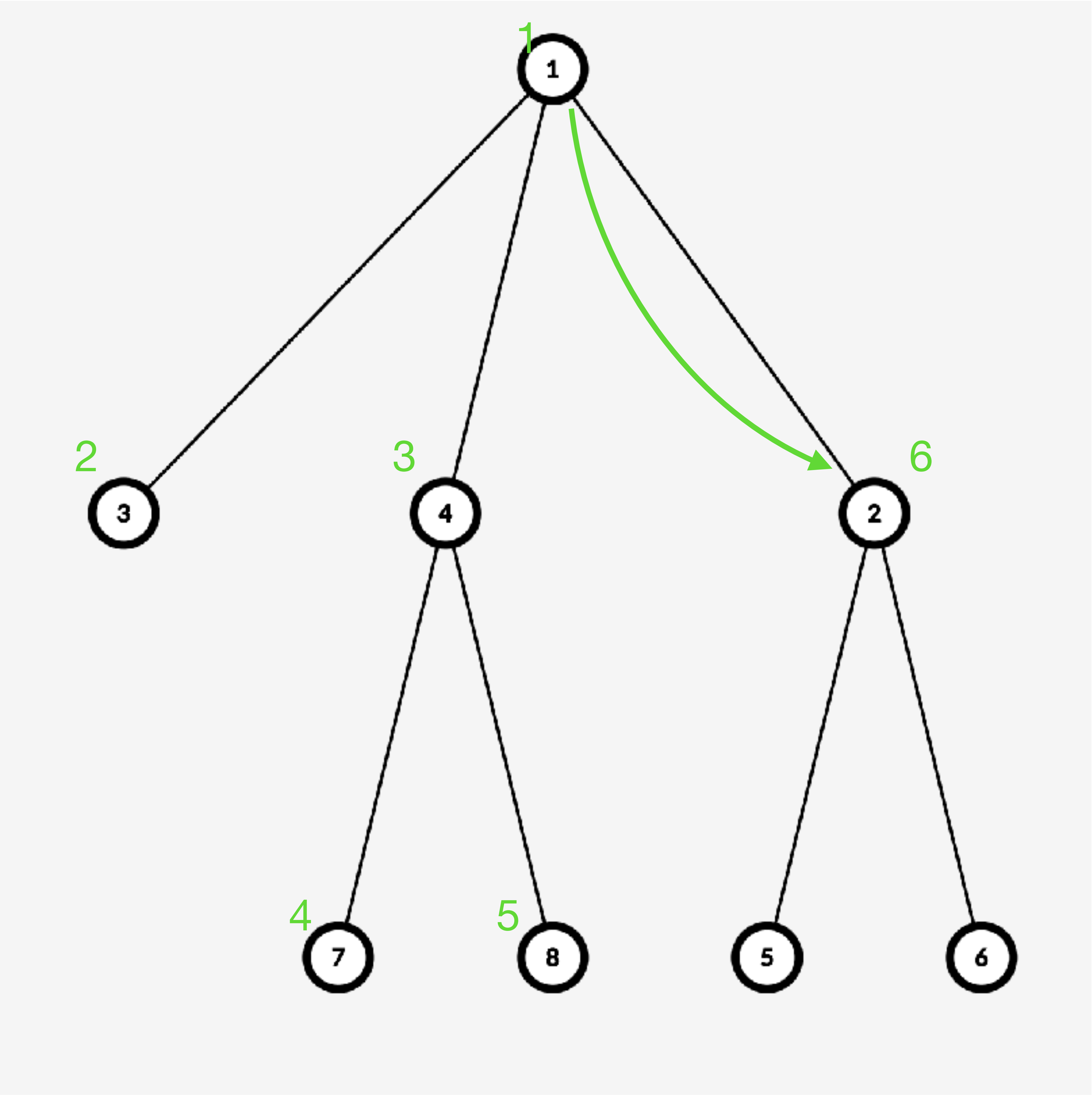
דוגמת הרצה לDFS

call: DFS(1)



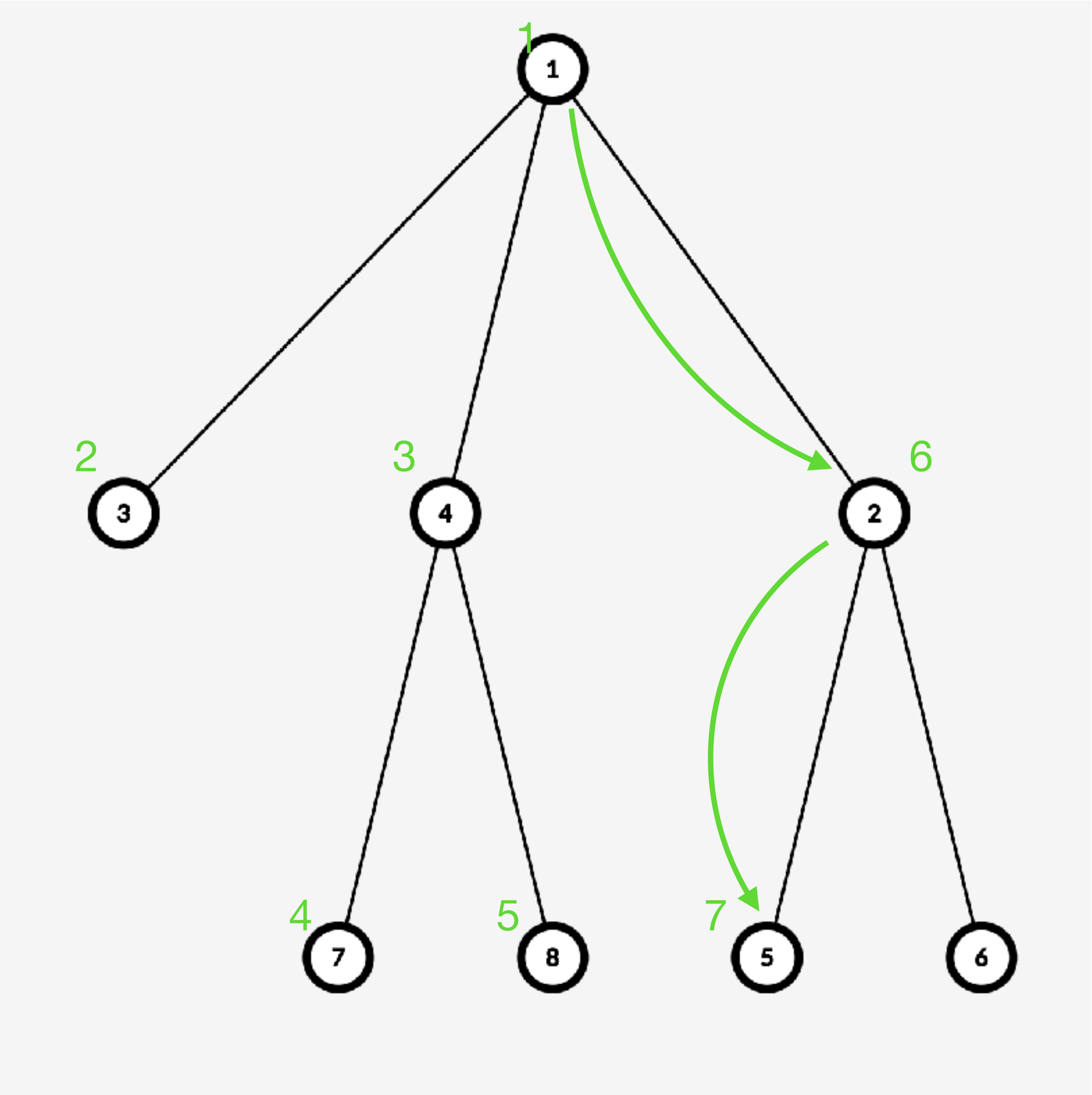
דוגמת הרצה לDFS

call: DFS(1)



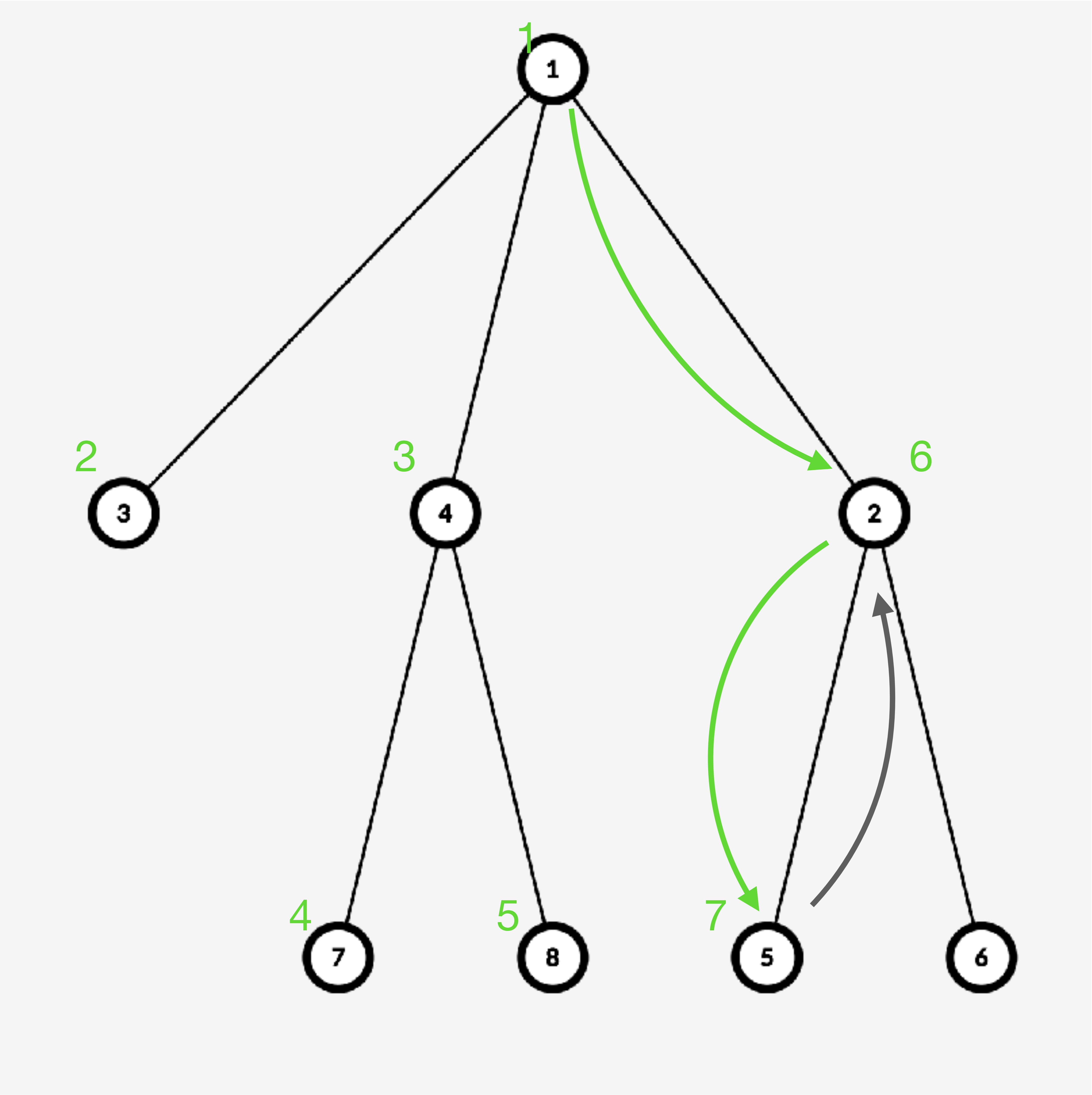
דוגמת הרצה לDFS

call: DFS(1)



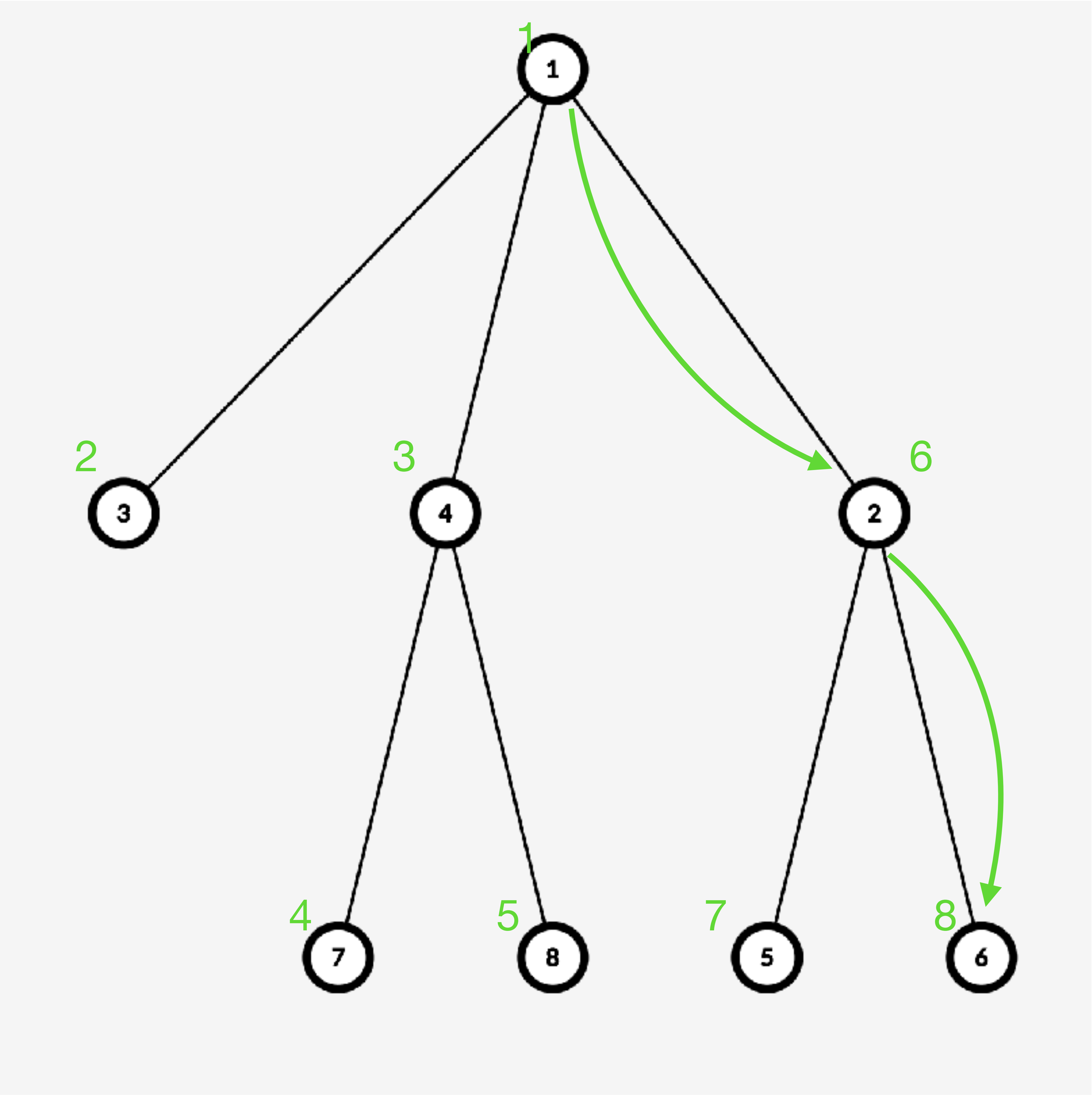
דוגמת הרצה לDFS

call: DFS(1)



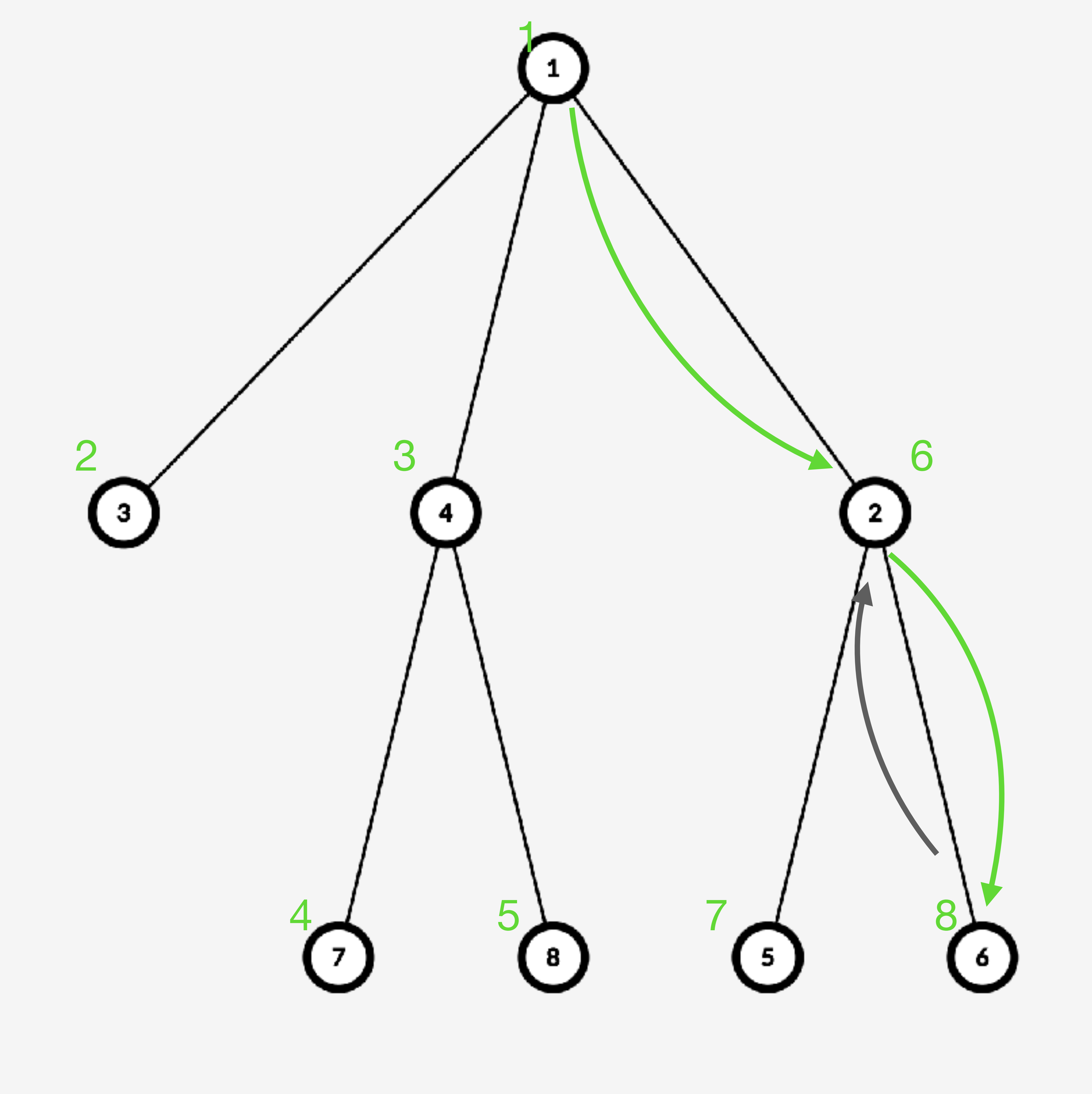
דוגמת הרצה לDFS

call: DFS(1)



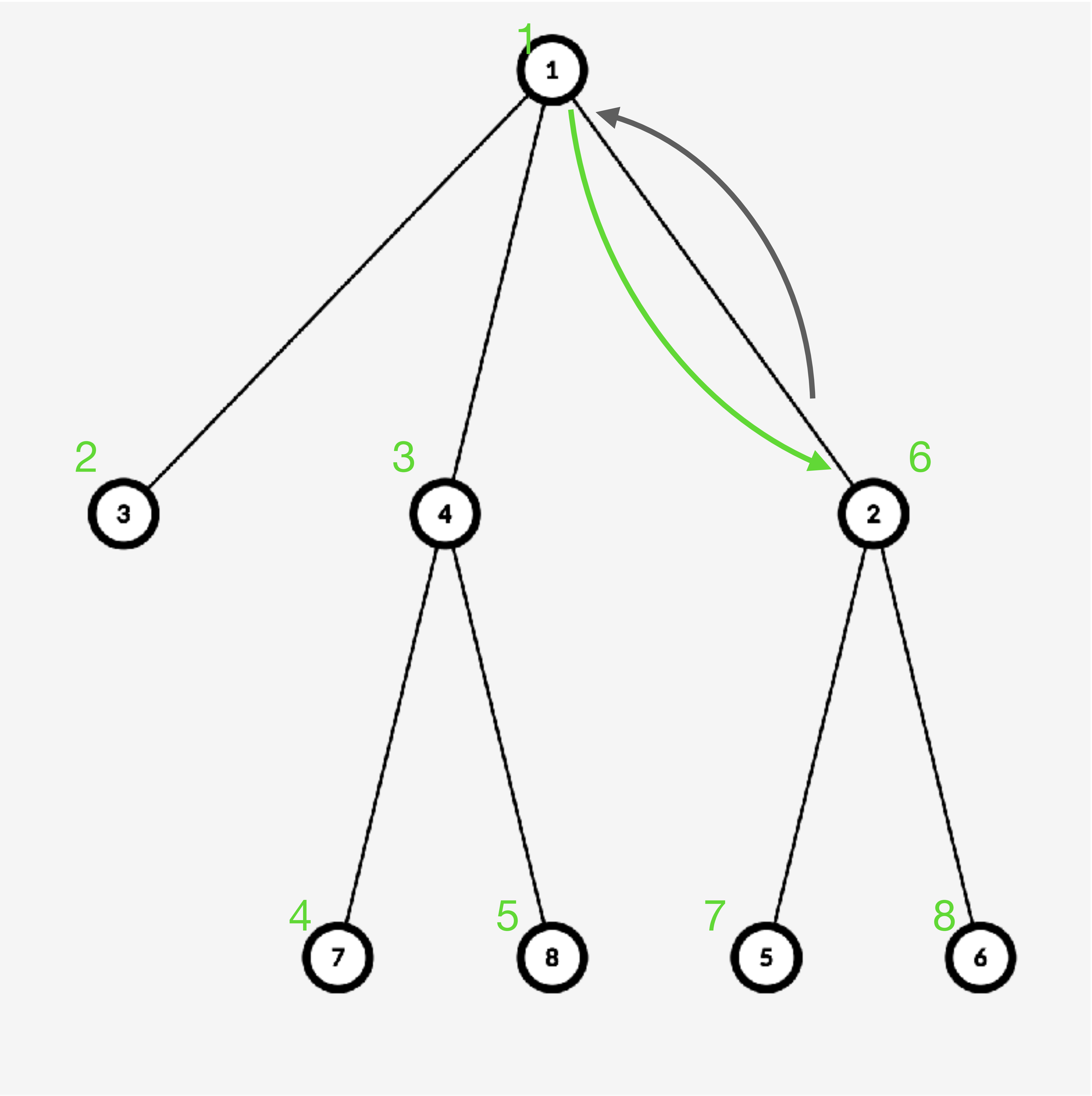
דוגמת הרצה לDFS

call: DFS(1)



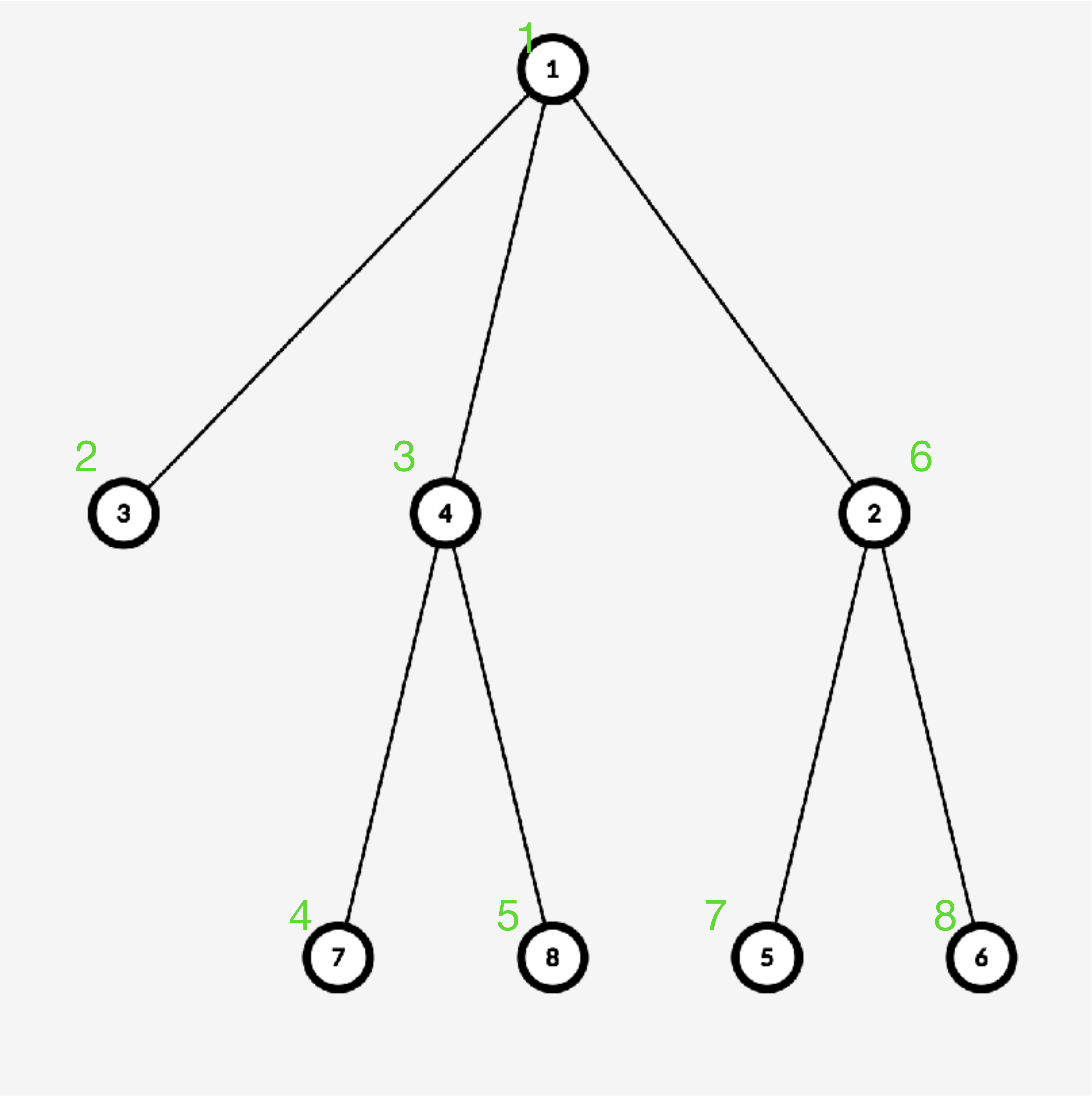
דוגמת הרצה לDFS

call: DFS(1)



דוגמת הרצה לDFS

DONE!



- לפי שיטת המספור נוכל לגלות דברים שונים על הגרף שלנו. בדוגמה שראינו המספור היה לפי סדר הופעה ו(לפחות) כרגע לא אומר לנו הרבה
- בהמשך נראה מספר שימושים לDFS, כגון מציאת רכיבי קשירות

מימוש ראשוני לDFS

```
vector<vector<int>> graph;

vector<int> DFS(int start) {
    stack<int> s;
    s.push(start);

    int time = 1;
    vector<int> visit_time(graph.size(), 0);

    while(!s.empty()) {
        int u = s.top();
        s.pop();
        if(!visit_time[u]) { // did not visit u already
            visit_time[u] = time++; // discovered u
            for(auto &v: graph[u]) s.push(v);
        }
    }

    return visit_time;
}
```

בעיות במימוש:

- מסריח בטירוף
- לא נוח

נזכר שכאשר קוראים לפונקציה רקורסיבית יש מחסנית קריאה. הבא ננצל אותה!

מימוש רשמי לDFS

```cpp
vector<vector<int>> graph;
int time = 1;
int in_time[maxn] = {0};
int out_time[maxn] = {0};

void DFS(int u) {
    if(in_time[u]) return; // already visited u

    in_time[u] = time++;
    for(auto &v : graph[u]) DFS(v);
    out_time[u] = time++;
}
```

שימושים לDFS

- מציאת רכיבי קשירות
- מיון טופולוגי
- מציאת גשרים בגרף
- בדיקת דו-צדדיות
- **ועוד!**

שימושים לDFS

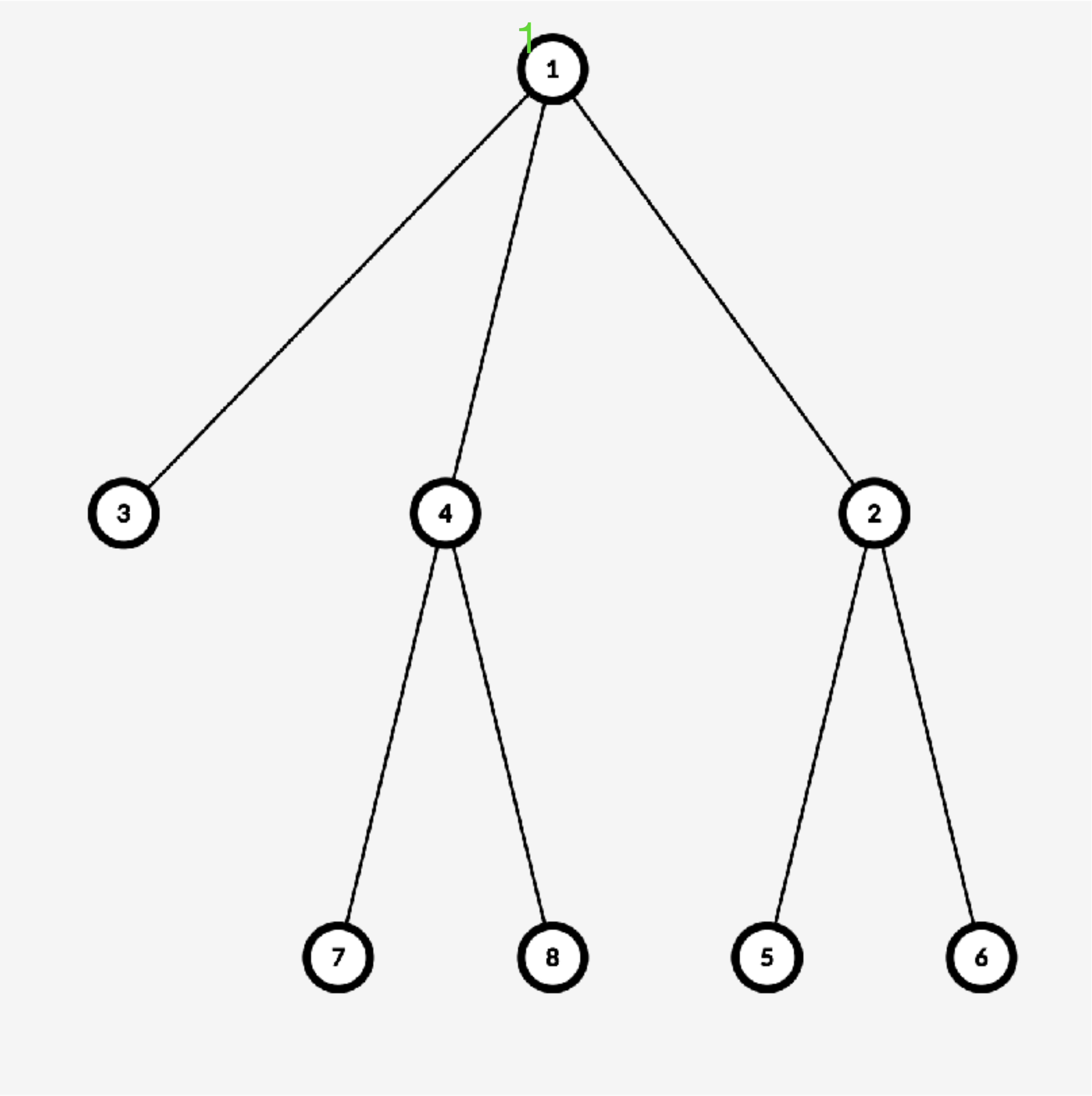
- מציאת רכיבי קשירות
- מיון טופולוגי
- מציאת גשרים בגרף
- בדיקת דו-צדדיות
- **ועוד!**
- נלמד מה מושגים אלו אומרים בהמשך

אלגוריתם - BFS

- חיפוש לרוחב (breadth first search) בגרף
- בביקור מסויים בצומת, נסמן שביקרנו אותו ("נכנסו אליו"), ואז נעבור לכל הצמתים האחרים שברמה שלו
- בכל שלב נחפש לרוחב (נכנס לכל הבנים באותה הרמה), ואחרי זה לעומק (נכנס לרמה הבא)
- נשמור מחסנית ובכל פעם שנכנס לבן חדש נוסיף אותו ל<u>תור</u>, וכשנסיים איתו נוציא אותו

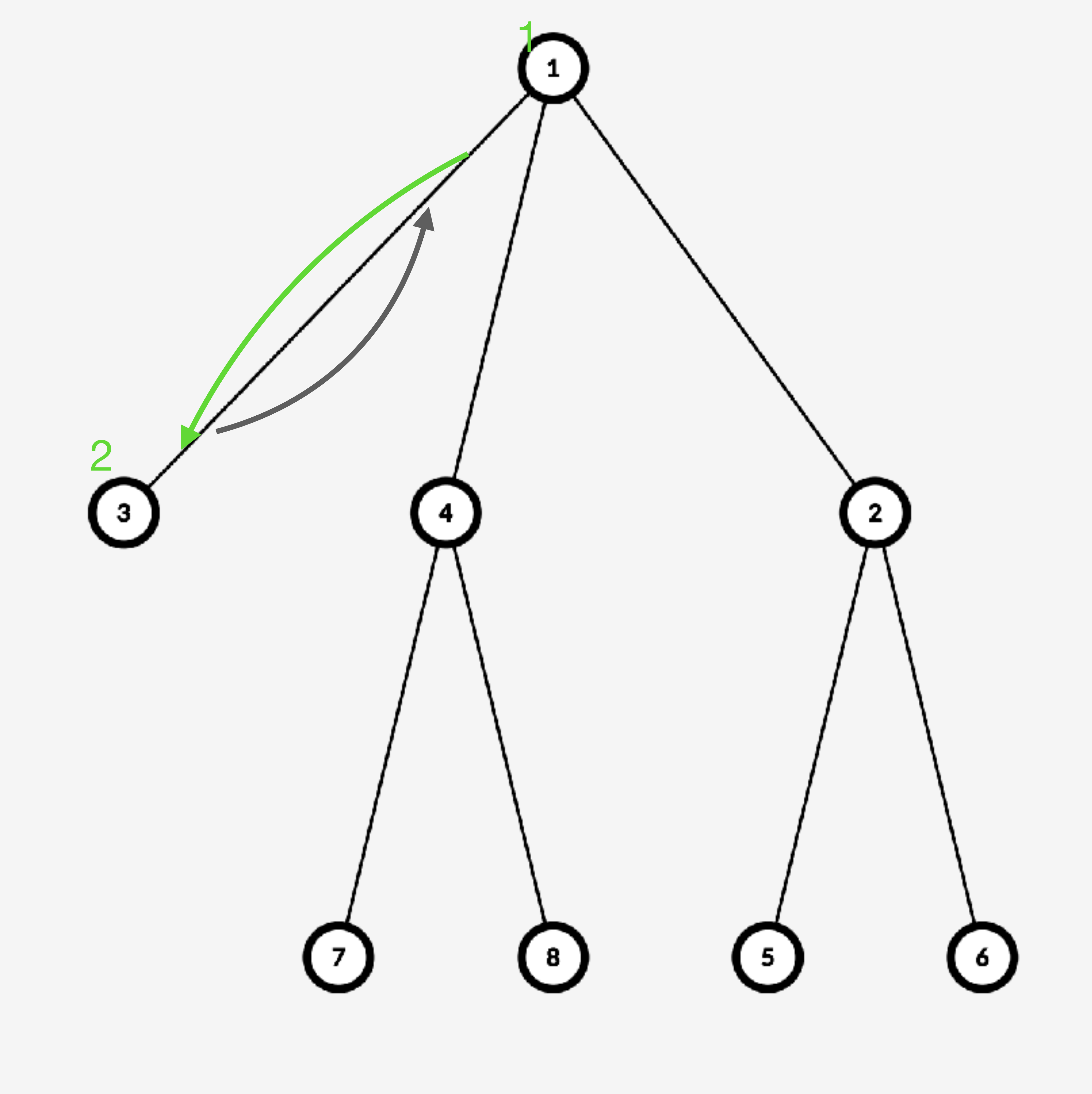
דוגמת הרצה לBFS

call: BFS(1)



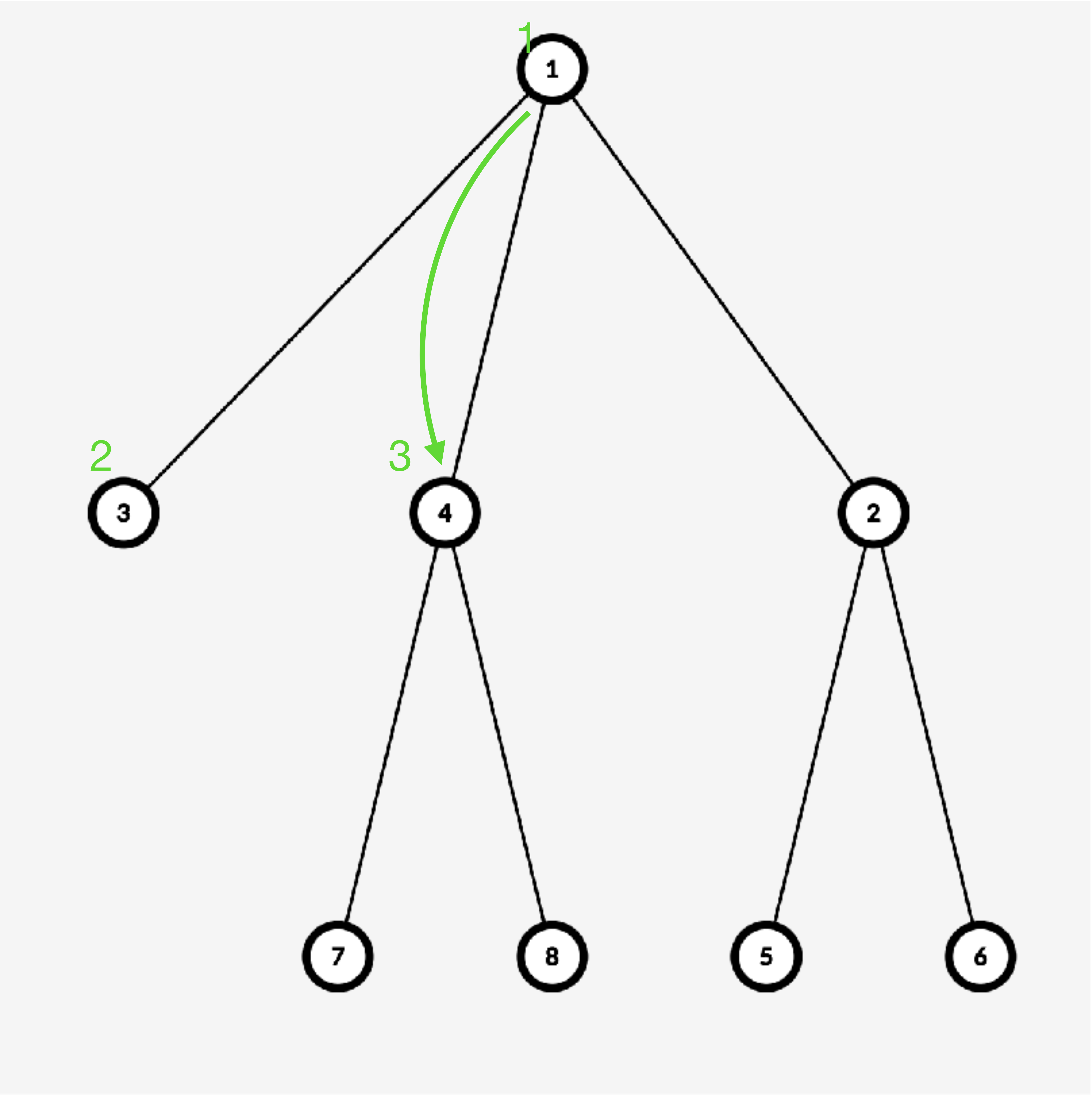
דוגמת הרצה לBFS

call: BFS(1)



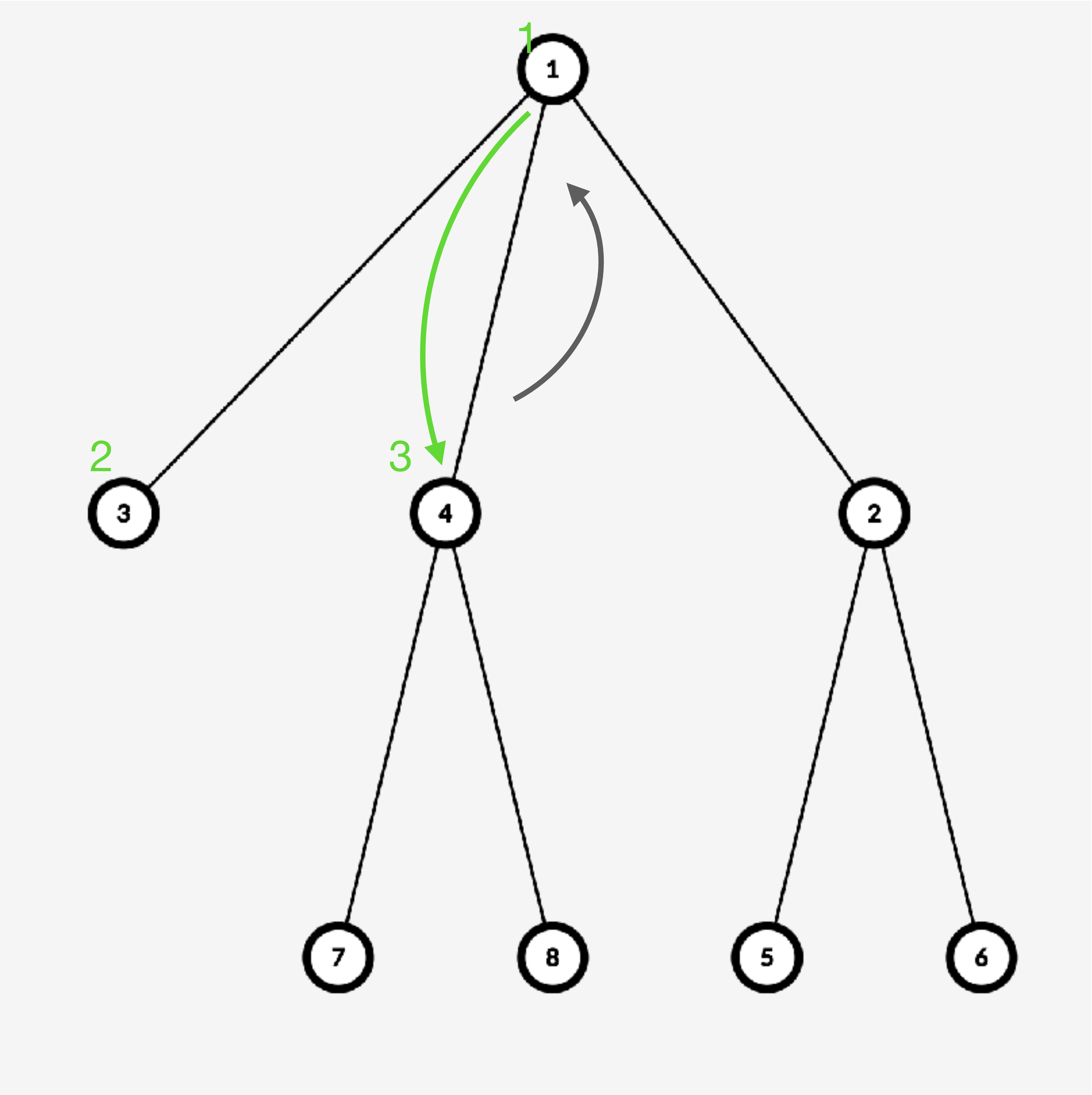
דוגמת הרצה לBFS

call: BFS(1)



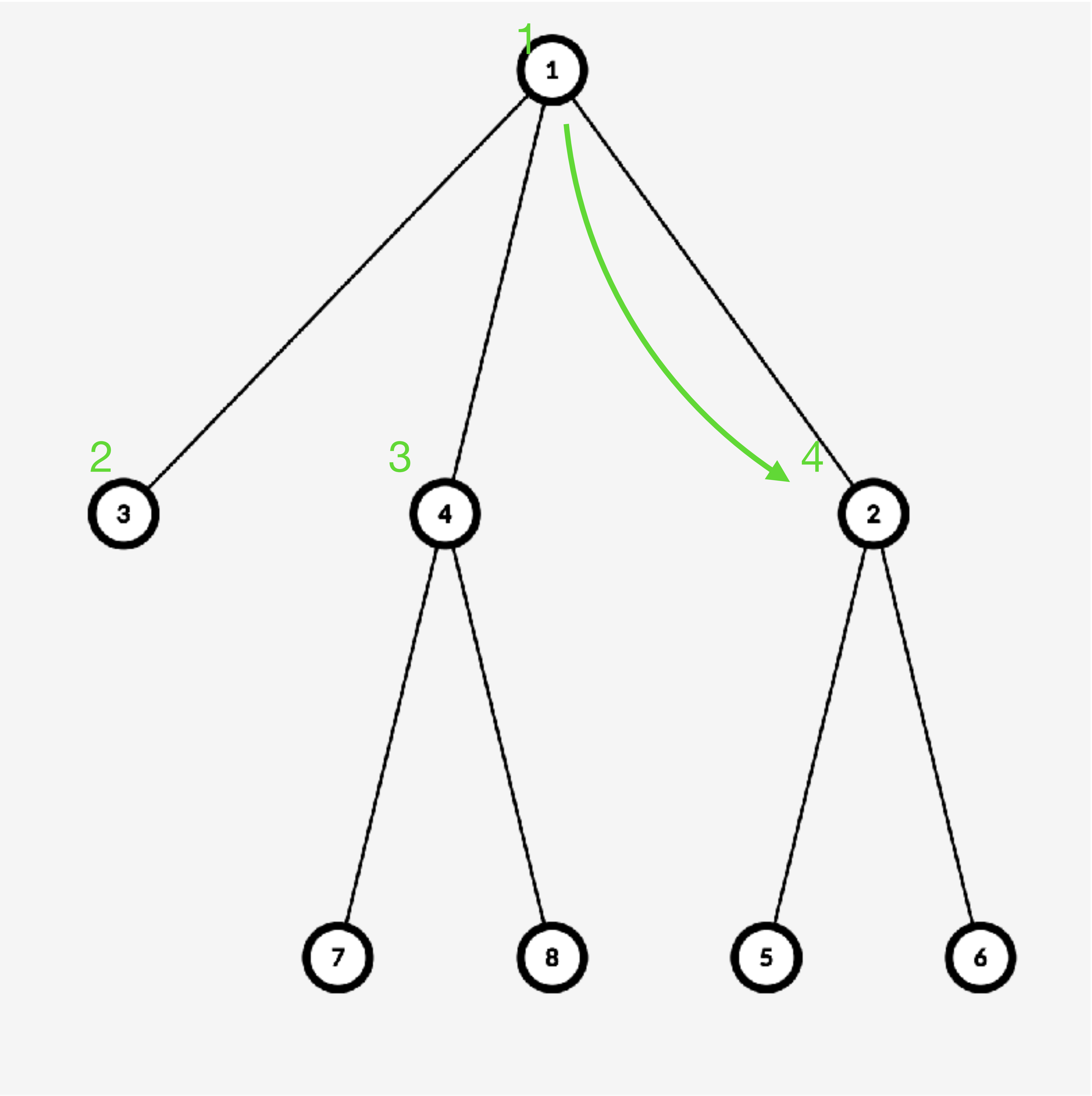
דוגמת הרצה לBFS

call: BFS(1)



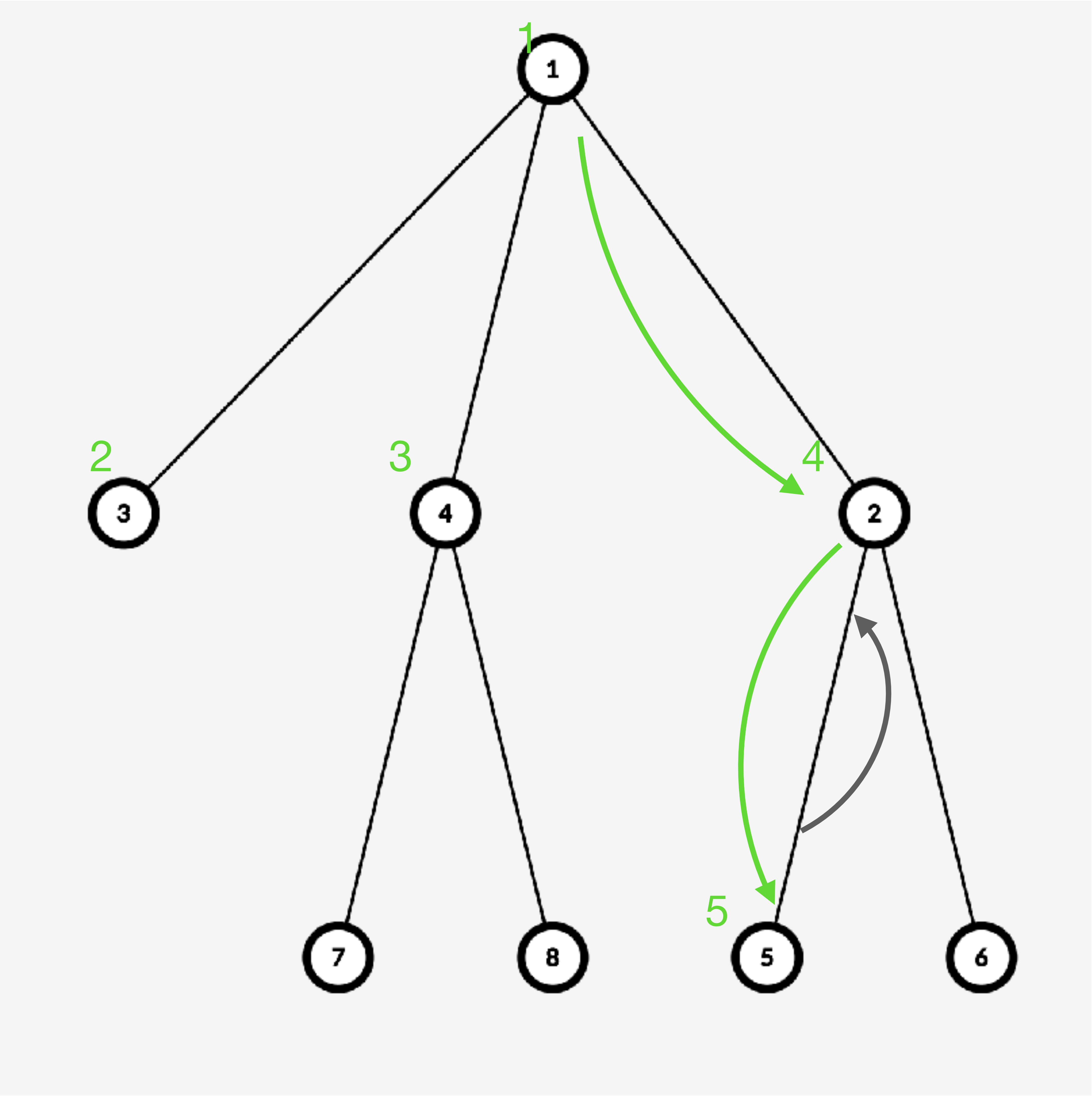
דוגמת הרצה לBFS

call: BFS(1)



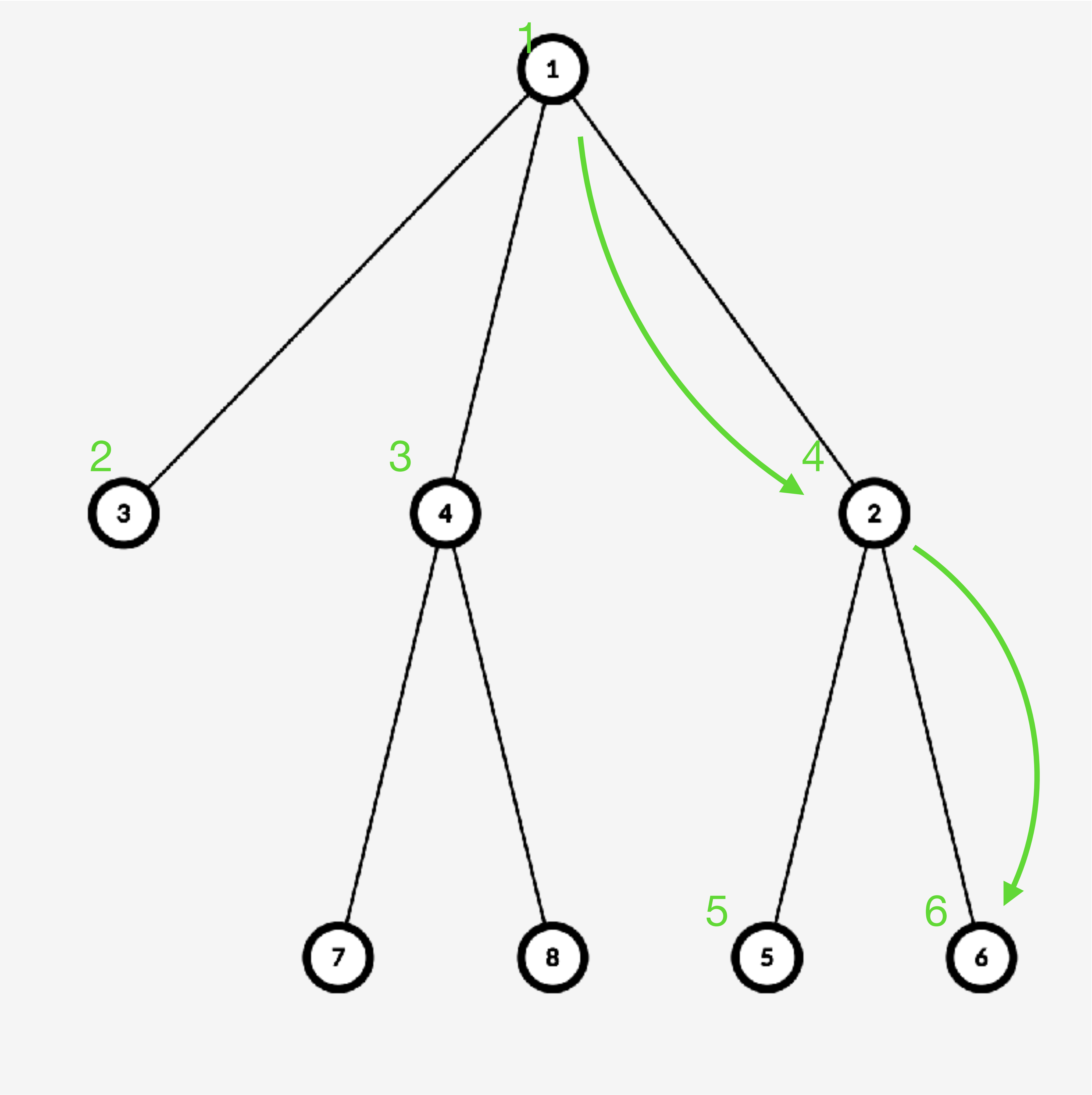
דוגמת הרצה לBFS

call: BFS(1)



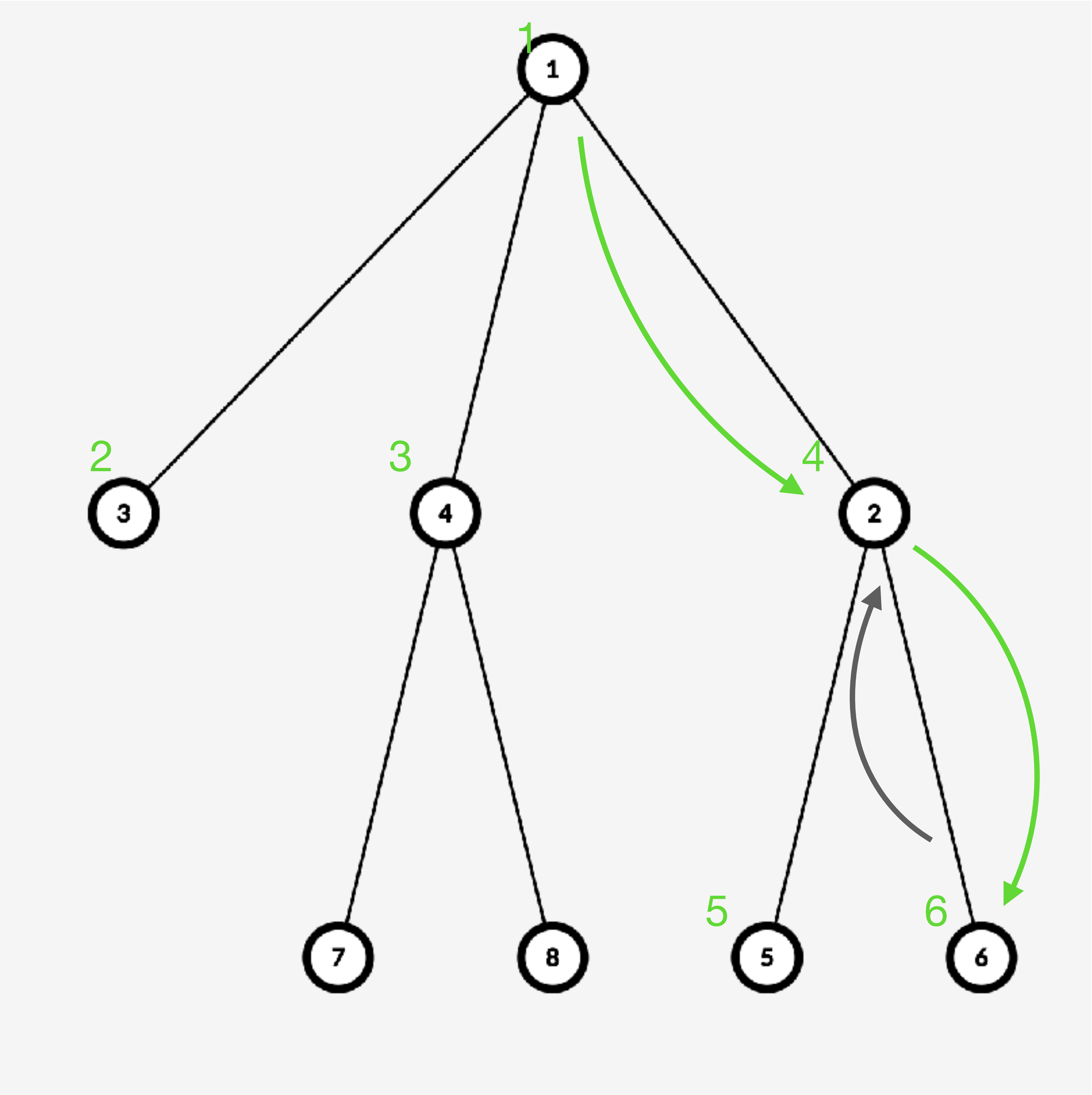
דוגמת הרצה לBFS

call: BFS(1)



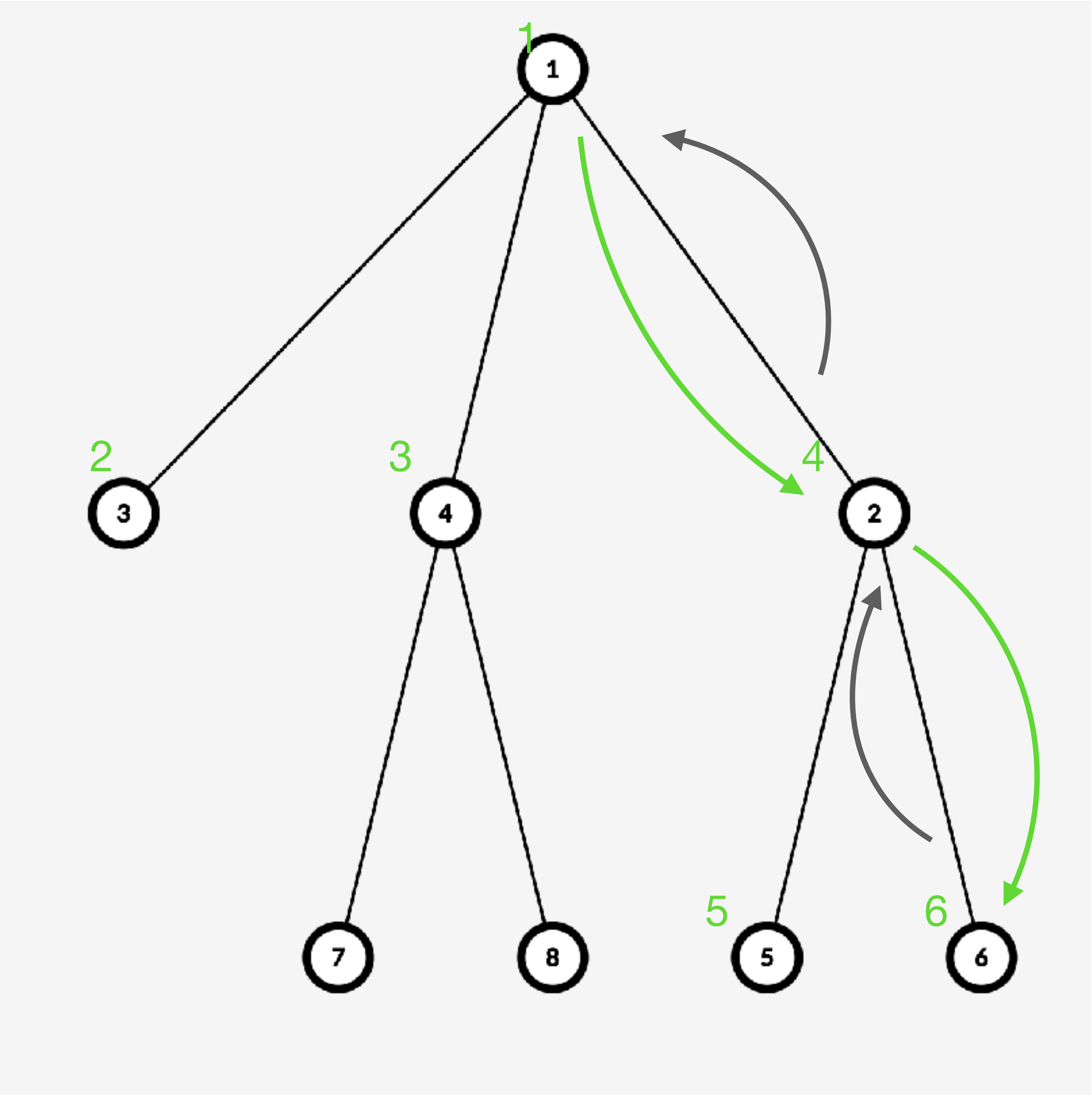
דוגמת הרצה לBFS

call: BFS(1)



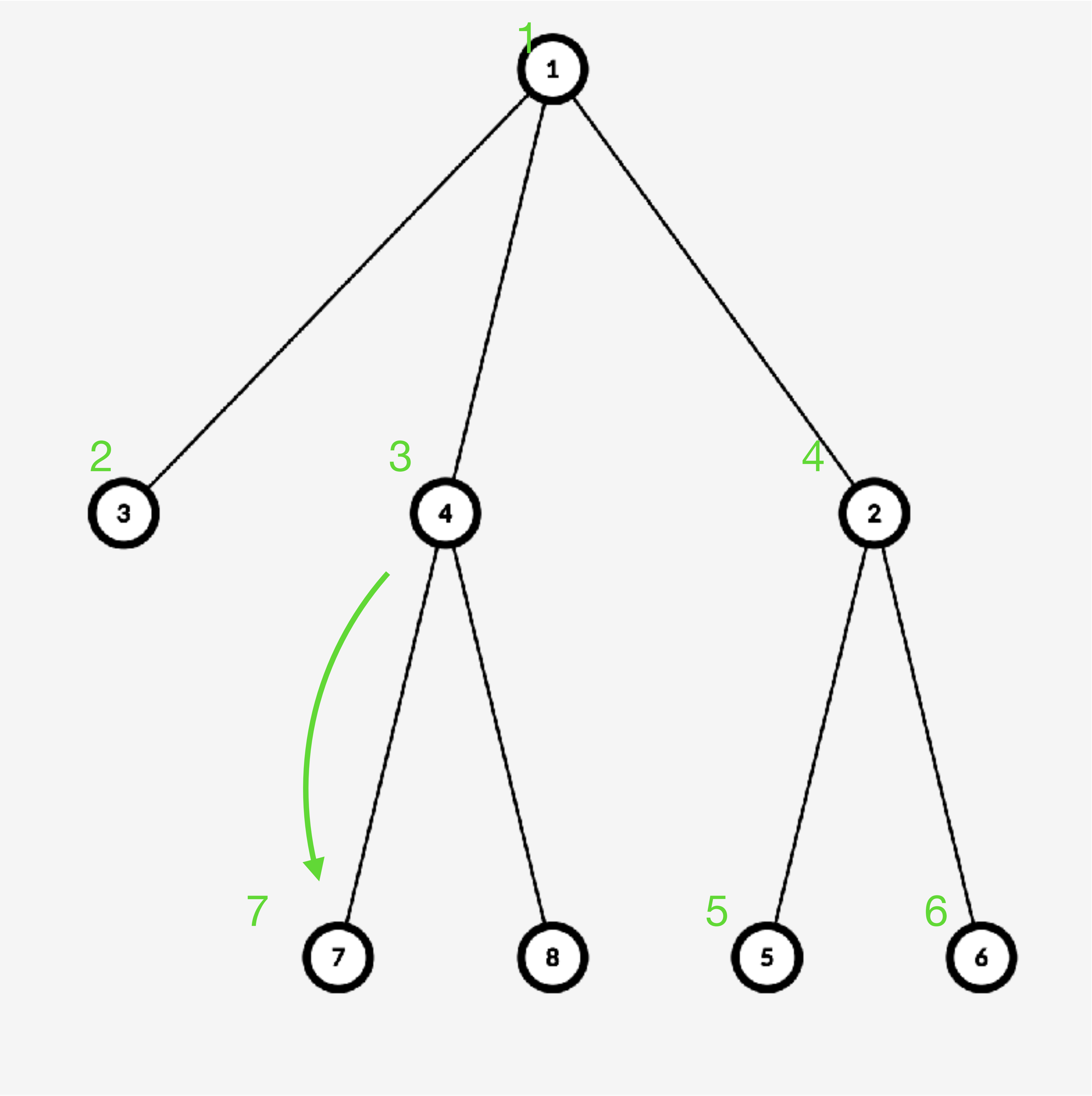
דוגמת הרצה לBFS

call: BFS(1)



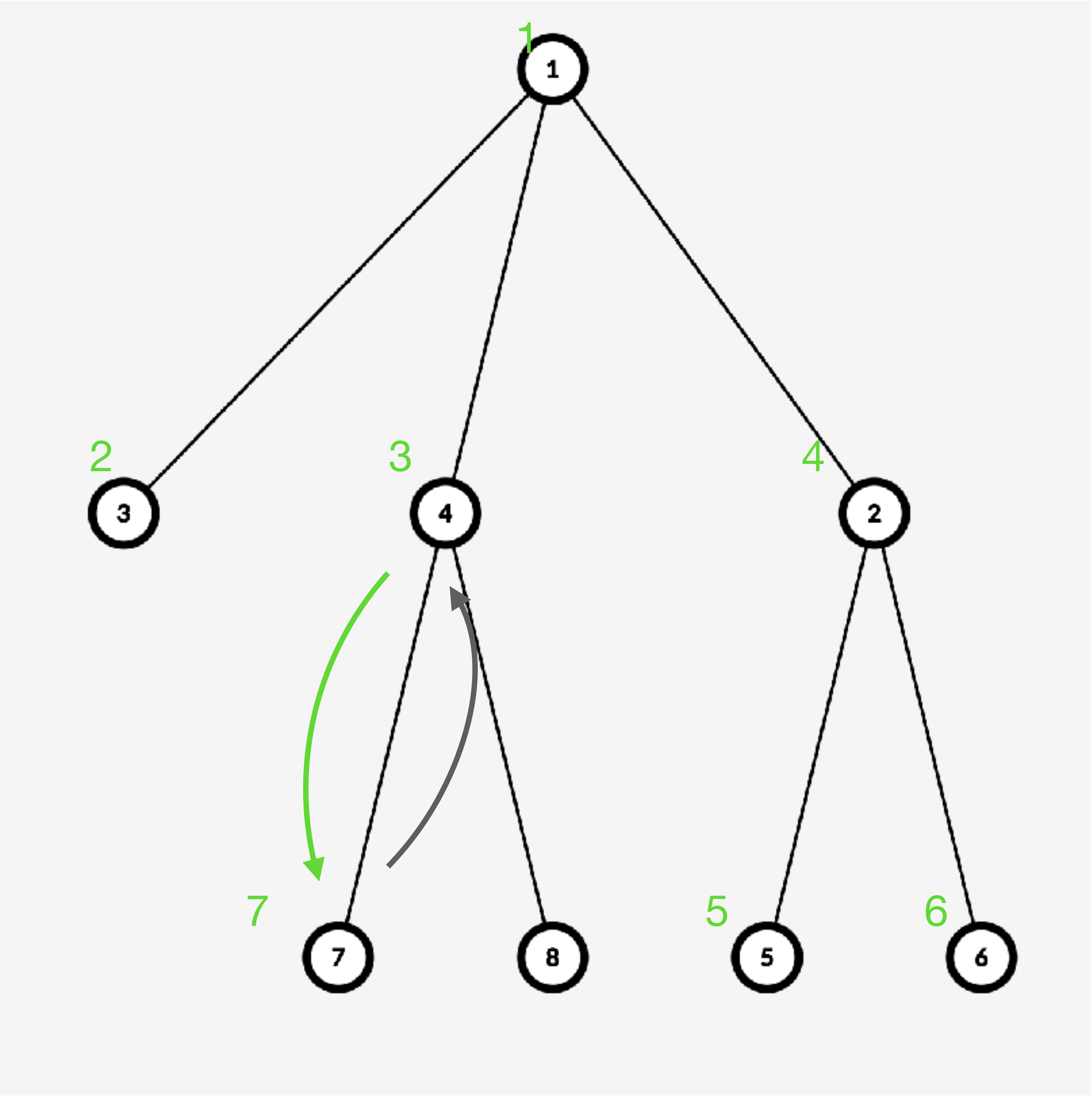
דוגמת הרצה לBFS

call: BFS(1)



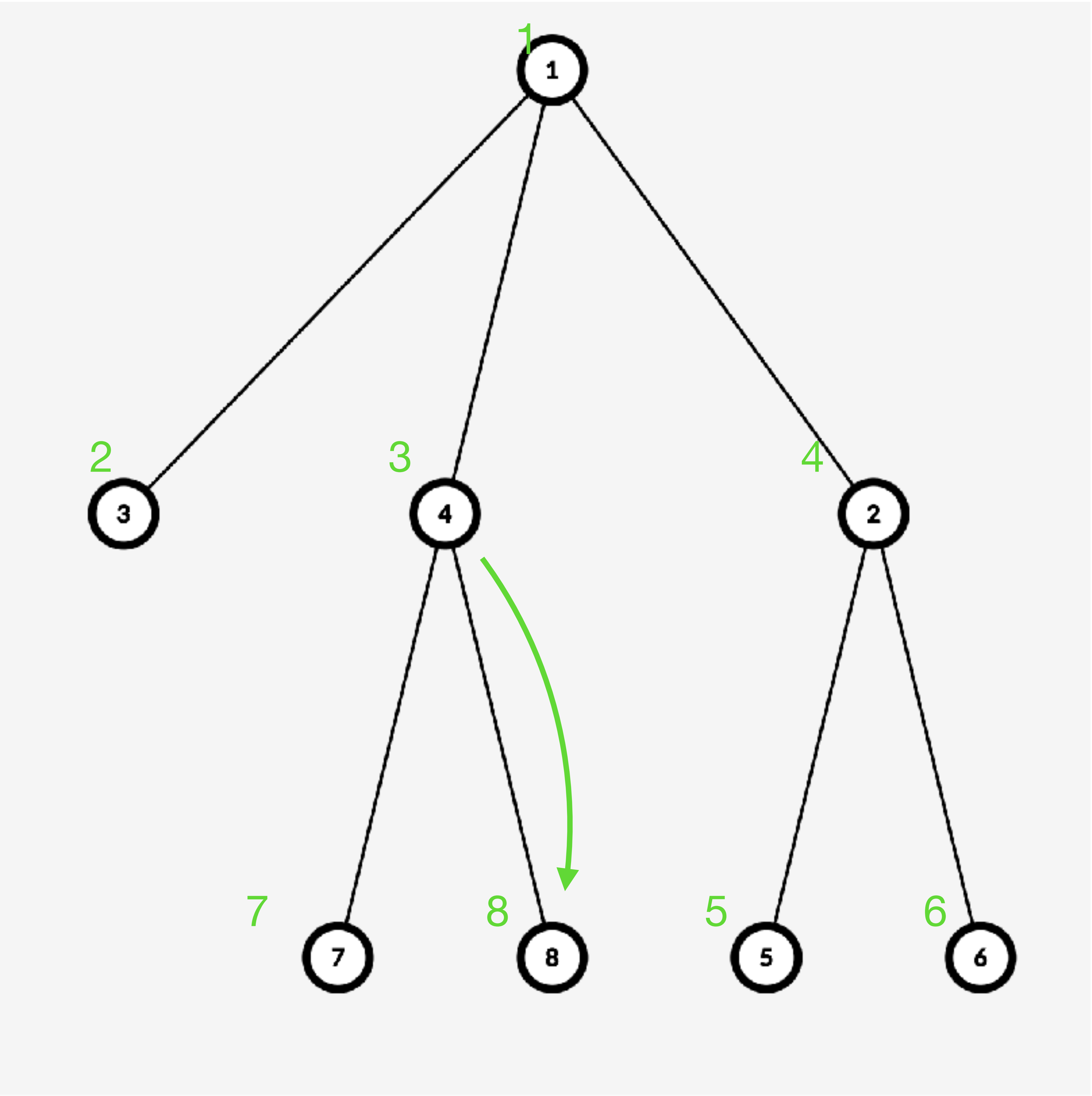
דוגמת הרצה לBFS

call: BFS(1)



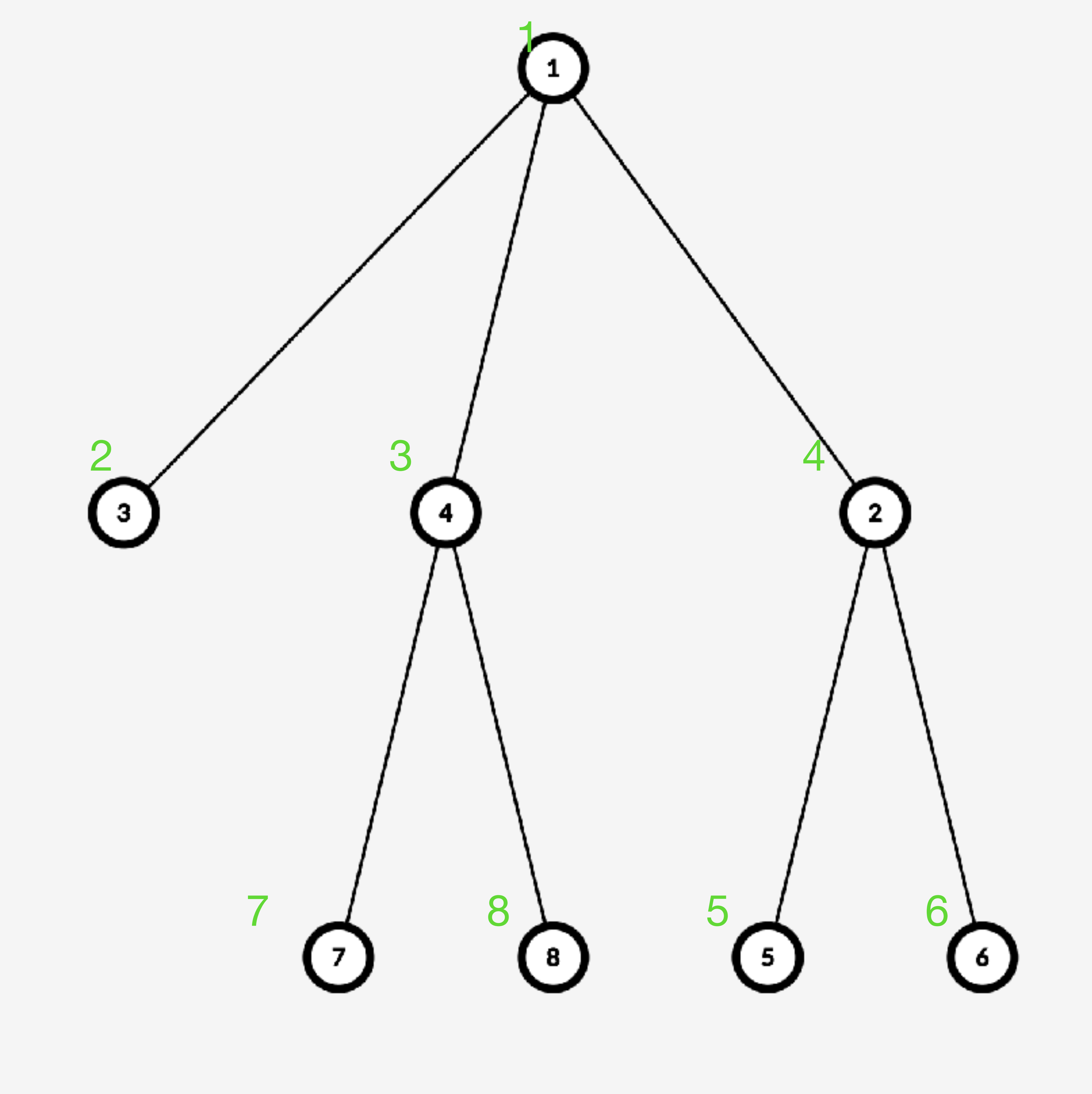
דוגמת הרצה לBFS

call: BFS(1)



דוגמת הרצה לBFS

DONE:
distances = {0,1,1,1,2,2,2,2}



- מאפשר בקלות למצוא את המרחקים (במספר הקשתות) מהצומת ההתחלתית
- בהמשך נראה הרחבה לBFS אשר מטפלת ביותר מקרים
- מימוש פחות נחמד מDFS :(

מימוש רשמי לBFS

```
vector<int> bfs(int source) {
    queue<int> q;
    vector<int> distances(n, 0); // distances[i] == distance(i, source)
    vector<bool> visited(n, false);
    q.push(source);
    visited[source] = true;

    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for(auto v : graph[u]) {
            if(!visited[v]) {
                visited[v] = true;
                distances[v] = distances[u] + 1;
                q.push(v);
            }
        }
    }

    return distances;
}
```

מימוש רשמי לBFS

```cpp
vector<int> bfs(int source) {
    queue<int> q;
    vector<int> distances(n, 0); // distances[i] == distance(i, source)
    vector<bool> visited(n, false);
    q.push(source);
    visited[source] = true;

    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for(auto v : graph[u]) {
            if(!visited[v]) {
                visited[v] = true;
                distances[v] = distances[u] + 1;
                q.push(v);
            }
        }
    }

    return distances;
}
```

שימוש:
distances[x] = # of edges between source, x

מימוש רשמי לBFS

```cpp
vector<int> bfs(int source) {
    queue<int> q;
    vector<int> distances(n, 0); // distances[i] == distance(i, source)
    vector<bool> visited(n, false);
    q.push(source);
    visited[source] = true;

    while(!q.empty()) {
        int u = q.front();
        q.pop();
        for(auto v : graph[u]) {
            if(!visited[v]) {
                visited[v] = true;
                distances[v] = distances[u] + 1;
                q.push(v);
            }
        }
    }

    return distances;
}
```

שימוש:
distances[x] = # of edges between source, x

הערה: ניתן גם לשמור את המסלול מsource אל שאר הצמתים

DFS vs BFS

- לשני האלגוריתמים (שיטות חיפוש) שראינו יש יתרונות וחסרונות.
- מתי נדע באיזה אלגוריתם להשתמש?
- לרוב וכמעט תמיד נשתמש בBFS כאשר נרצה לדעת מרחק (=מספר קשתות) מצומת קבוע. נוכל לעשות זאת בקלות ואף קל מאוד לשדרג את הקוד שראינו למרחק לפי מספרים חיוביים ($\mathbb{R}^+$).
- אם נרצה לדעת מידע על מבנה הגרף (לדוגמה: קשירות), אזי נשתמש בDFS במקום זאת.

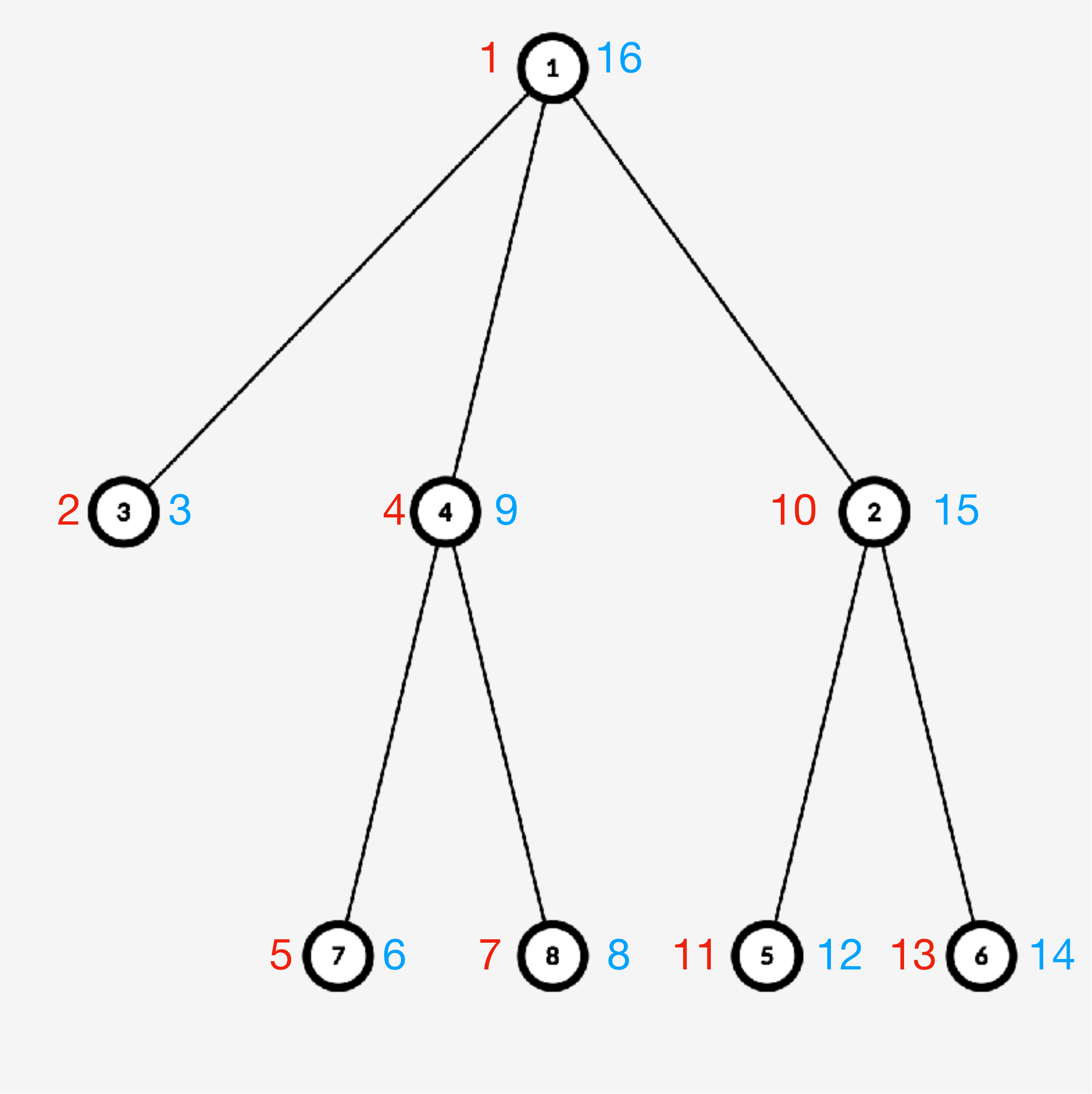
DFS על עצים

- נזכר במימוש שלנו לDFS: בשונה מBFS, במקום לשמור מרחק מהצומת ההתחלתי אנו שומרים זמני כניסה ויציאה מצומת. פעולה זו מאוד שימושית, ואפילו גוררת משפט חשוב:

- **משפט.** יהי $T = (V, E)$ עץ, ו-$v \in V$ צומת כלשהי בעץ. אם לאחר הרצת DFS עם צומת התחלתי $r \in V$ כלשהו קיבלנו זמני כניסה ויציאה מכל צומת i_v, o_v אזי הצומת $u \in V$ נמצאת בתת-עץ של v אמ"מ $i_u \in [i_v, o_v]$ (או $o_u \in [i_v, o_v]$).

- *הוכחה.* נניח כי נכנסו לצומת v בזמן t_0 כלשהו, ויצאנו בזמן t_1. אזי, בקטע (interval) הזמנים (t_0, t_1) ביקרנו רק בצמתים אשר נמצאים בתת-עץ של v. לכן, אם זמן הכניסה או יציאה של צומת כלשהו מקיים $t_0 < t < t_1$ אז בהכרח הצומת הזה נמצא בתת-עץ של v, וזאת מכיוון שמספור הזמנים הינו רציף.

DFS על עצים



- לדוגמה, אם ניקח את העץ מקודם ונרשום את זמני הכניסה, יציאה, נקבל:

- *מסקנה 1.* האינטרבל $[1,2n]$ מכיל את כל העץ, כאשר $n = |V|$.