<u>שלב ראשון של המימוש:</u>

1. ליצור קובץ בשם GraphNode.java ובו יש מחלקה פומבית בשם GraphNode. מחלקה זו מייצגת צומת $v \in V$ ולה המתודות הפומביות הבאות:

   1. **public int getOutDegree()** - מתודה המחזירה את דרגת היציאה של צומת v כלומר, מספר הקשתות היוצאות מצומת v.

      סיבוכיות נדרשת $O(\deg_{out}(v))$.

   2. **public int getInDegree()** - מתודה המחזירה את דרגת הכניסה של צומת v כלומר, את מספר הקשתות הנכנסות לצומת v.

      סיבוכיות נדרשת $O(\deg_{in}(v))$.

   3. **public int getKey()**- המתודה מחזירה את המזהה של הצומת.

      סיבוכיות זמן נדרשת - $O(1)$.

2. ליצור קובץ בשם GraphEdge.java ובו יש מחלקה פומבית בשם GraphEdge. מחלקה זו מייצגת קשת $e \in E$.

3. ליצור קובץ בשם RootedTree.java ובו יש מחלקה פומבית בשם RootedTree. מחלקה זו מייצגת עץ מושרש ולה המתודות הפומביות הבאות:

   1. **public RootedTree()** - בונה ברירת מחדל (default constructor) למחלקה RootedTree היוצר עץ מושרש ריק, ללא צמתים.

      סיבוכיות נדרשת - $O(1)$.

   2. **public void printByLayer(DataOutputStream out)**-

      המתודה מקבלת רפרנס לאובייקט מסוג DataOutputStream ומדפיסה ל stream את המזהים הייחודיים של הצמתים תחת הדרישות הבאות:

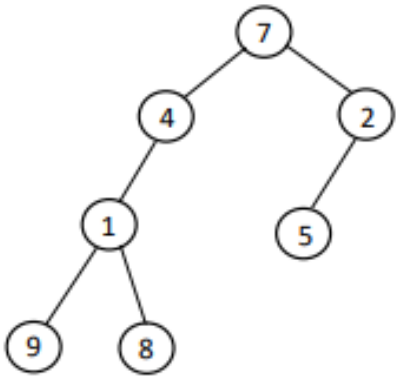
      1. מזהים של צמתים השייכים לרמה i בעץ (צמתים בעומק i) יודפסו בשורה ה $i+1$ של stream.
      2. הדפסות של מזהים באותה הרמה יופרדו רק באמצעות פסיקים.

3. הדפסה של מזהים של צמתים באותה הרמה מתבצעת משמאל לימין. פורמלית, לכל שני צמתים u ו v עם הורה משותף כך ש u הוא אח שמאלי (לאו דווקא ישיר) של v מתקיים:

המזהה של u יודפס לפני המזהה של v. אם u' ו v' הם צאצאים של u ו v, בהתאמה ונמצאים באותה רמה בעץ, אז המזהה של u' יודפס לפני המזהה של v'.

סיבוכיות זמן נדרשת - $O(k)$, כאשר k מייצג את מספר הצמתים בעץ.

לדוגמא, הפעלת המתודה הנ"ל על הייצוג של העץ המושרש



צריכה להדפיס:

7
4,2
1,5
9,8

שימו לב כי אין פסיק בסוף השורה. ניתן לראות דוגמאות נוספות בפלט החוקי שקיבלתם.

3. - **public void preorderPrint(DataOutputStream out)**
המתודה מקבלת רפרנס לאובייקט מסוג DataOutputStream ומדפיסה ל stream את המזהים של צמתי העץ בסדר preorder (כפי שנלמד בתרגול 3) תחת הדרישה שהדפסת תתי העצים של צומת מתבצעת מהילד השמאלי ביותר לימני ביותר. ההדפסה מתבצעת בשורה אחת כאשר בין מזהים מפריד רק פסיק (אין פסיק בסוף השורה).

שלב שני של המימוש:

כעת אנו יכולים להגדיר את המחלקה DynamicGraph.

עליכם ליצור קובץ בשם DynamicGraph.java ובו יש מחלקה פומבית בשם DynamicGraph.
המחלקה DynamicGraph משתמשת במחלקות GraphNode GraphEdge, ו RootedTree על מנת לממש את המתודות הפומביות הבאות:

1. **public DynamicGraph()** - בונה ברירת מחדל (default constructor) למחלקה DynamicGraph היוצר גרף דינמי חדש ריק, ללא קשתות וללא צמתים.

   סיבוכיות זמן נדרשת - $O(1)$.

2. **public GraphNode insertNode(int nodeKey)** – מתודה זו מכניסה צומת חדש לגרף עם מזהה ייחודי nodeKey ומחזירה רפרנס לאובייקט מסוגGraphNode המייצג את הצומת שהוכנס. הצומת החדש מתווסף לגרף ללא קשתות נכנסות או יוצאות.

   סיבוכיות זמן נדרשת – $O(1)$.

3. **public void deleteNode(GraphNode node)**–
   המתודה מקבלת רפרנס בשם node לצומת שנמצא בגרף. אם לצומת node יש קשתות יוצאות או נכנסות המתודה לא עושה כלום. אחרת, המתודה מוחקת את הצומת node מהגרף.

   סיבוכיות זמן נדרשת - $O(1)$.

4. **public GraphEdge insertEdge(GraphNode src, GraphNode dst)**-
   המתודה מקבלת שני רפרנסים לצמתים בגרף to ו from. המתודה מוסיפה לגרף קשת מהצומת from לצומת to ומחזירה רפרנס לאובייקט המייצג את הקשת שהוכנסה.

   סיבוכיות זמן נדרשת - $O(1)$.

5. **public void deleteEdge(GraphEdge edge)** - המתודה מקבלת רפרנס לקשת בגרף בשם edge. המתודה מוחקת את הקשת edge מהגרף.

   סיבוכיות זמן נדרשת - $O(1)$.

6. **public RootedTree scc()** - המתודה מחשבת רכיבים קשירים היטב בגרף $G=(V,E)$. המתודה מחזירה רפרנס לאובייקט מסוג RootedTree. מבנה ה RootedTree הוא כדלקמן:
   השורש הוא צומת וירטואלי (שלא קיים בגרף) עם מזהה ייחודי 0. קבוצת הצאצאים של כל ילד של השורש מהווה רכיב קשיר היטב בגרף G.

   סיבוכיות זמן נדרשת - $O(n+m)$.
   (שימו לב לדרישות הנוספות המצורפות למטה).

7. **public RootedTree bfs(GraphNode source)** - המתודה מקבלת רפרנס לצומת בגרף source, ומחזירה עץ מסלולים קצרים מהצומת source.

   סיבוכיות זמן נדרשת - $O(n+m)$.
   (שימו לב לדרישות הנוספות המצורפות למטה).