**עצי חיפוש בינאריים – Binary Search Trees**

עץ בינארי מיוצג על ידי מערך (array) או על ידי רשומה מקושרת (linked list).

Next

Head

End

רשומה מקושרת היררכית מכילה אבא ובנים, לאבא יש שתי רגליים שמגיע אל הבנים.

השמאלי יהיה יותר קטן מהאב וימין גדול ממנו. בתחילת החיפוש נצטרך להצביע על האיבר הראשון ממנו נתחיל את החיפוש.

איבר ששני הערכים שלו בעץ שווים ל-NULL נקרא עלה.

עץ החלטה הוא רשימה מקושרת ממבנה היררכי עם אופציה בינארית של YES/NO ואיזו אופציה תמשיך את החיפוש או תקטע אותו.

עץ בנוי מצמתים (node).

**רשומה מקושרת היררכית:**

צומת (node)

**צומת בקוד:**

*struct node{*

*int value;*

*node \*left, \*right;*

*}*

* כל עץ הוא בנוי מצמתים
* צמרת העץ תקרא שורש, ממנו מתחיל העץ
* לכל צומת יהיו 0 ומעלה ילדים

תת עץ שמאל

* צומת עם 0 ילדים נקראת "עלה"
* עץ הוא ריק רק כשהשורש שלו שווה NULL

גובה עץ = n

מספר הצמתים בעץ בינארי מלא = (n^2) -1

מספר עלים בעץ בינארי מלא = 2^(n-1)

כשבוחרים צומת בעץ נזהה את זה כ-"תת עץ ימין/ שמאל".

עץ בינארי מלא הוא אחד כזה שכל צומת עלים הוא באותו העומק וכל איבר שאינו עלה יש לו בדיוק 2 ילדים.

עץ מלא

**הגדרת עץ בינארי בפייתון:**

class \_BinTreeNode:

def \_\_init\_\_(self,data):

self.data = data

self.left = None

self.right = None

**חיפוש בעץ:**

תבקר בצומת

תלך לתת עץ ימין/ שמאל

תלך לתת עץ ימין/ שמאל

(רקורסיה)

Preorder – עובר על הענפים של העץ לפי הענף השמאלי ביותר (ככל האפשר).

Inorder – עובד בשיטת “left, data, right”.

**מימוש בפייתון:**

*def preOverTrav(subtree):*

*if subtree is not None:*

*print(cuntree.data)*

*preorderTrav(subtree.left)*

*preorderTrav(subtree.right)*

**Dense Graph** – גרפים דרך מערך או רשימה מקושרת (מילון בשפת פייתון).

**Sparse Graph** - ?

**Graph Traversal** – מתחלק ל-2 קבוצות:

* Depth first – ממשיך להתקדם עד שמגיע למקום ללא מוצא או מיקום שהדרך כבר ביקרה בוא. אז חוזר ויוצא לדרך אחרת.
* Breadth first – מגלה את הגרף בצורה מעגלית דרך כל הקודקודים.