

עצי חיפוש בינאריים - binary search tree

בURITY חיפוש היא בעיה נפוצה ונתן רצים שהייתה לנו לבנייה נתונים שיאפשר לנו לעשות פעולה זו בזמן יעיל, וכן שפעולות הכנסה ומחיקה תהינה ייעילות.

כמו"כ ישנו חיפושים מוחכמים יותר הכוללים חיפוש איברים בטוח בין X ל- Y, וכן חיפוש אחר איבר הקרוב ביותר למפתחו, וגם זאת אנו רצים שיישנה בזמן ריצה יעיל.

זמן הריצה לכל הפעולות הנ"ל במבנה הנתונים שכבר למדנו:

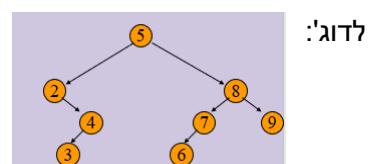
הfonction	הפעולה	(n)	מתקנות דו כיוונית	מערך ורשימה	רשימה ממויין	טבלאות גיבוב
Find	חיפוש	O(1) - בממוצע	O(n)	O(log n)	O(n)	O(n)
Insert	הוספה	O(1)	O(n)	O(1)	O(n)	O(n)
Delete	מחיקה	O(1) - בממוצע	O(n)	O(n) - כשיין מצביע	O(n)	O(n)
rangeSearch	חיפוש טווח	O(n+m)	O(n)	O(n)	O(log n)	O(n)
nearestNeighbors	חיפוש איבר סמוך	O(n+m)	O(n)	O(n)	O(n)	O(n)

כדי שכל הפעולות תבוצענה בזמן ריצה יעיל נשתמש במבנה נתונים:

עץ חיפוש בינארי - עץ שעבור כל צומת מתקיים:

כל מפתחות הצמתים מתחת לעץ השמאלי קטנים מהמפתח שלו

ולכל מפתחות הצמתים מתחת לעץ הימני גדולים מהמפתח שלו



הפעולות הבסיסיות:

1. **חיפוש – find(k,R)** : חיפוש צומת בעל מפתח k בעץ R.

החל מהשורש עבור כל צומת בודק :

* אם k שווה למפתח שלו - החזר את הצומת.

* אם K קטן מהמפתח שלו - המשך לחפש מצד שמאל (אם קיימ)

* אם K גדול מהמפתח שלו - המשך לחפש מצד ימין (אם קיימ)

אם הערכ לא נמצא יוחזר הצומת שאחריו ניתן להוסיף את המפתח.

הימוש:

```

Find(k,R)
if R.Key = k:
    return R
else if R.Key > k :
    if R.Left ≠ null:
        return Find(k, R.Left)
    else
        return R
else if R.Key < k :
    if R.Right ≠ null:
        return Find(k,R.Right)
    else
        return R

```

2. איבר עוקב- (N)next- חיפוש צומת בעל מפתח מינימלי הגדל מהمفחה של N

החל מהצומת N :

אם אין לו בן ימני - עליו להחזיר את האב הראשון שהוא בן שמאל שלו (אם קיים)	אם יש לו בן ימני - עליו לחפש את המפתח המינימלי בתת עץ הימני של N כלומר: הבן הכי שמאל של הבן הימני

המימוש:

If N.Parent=null Return null	RightAncestor(N) <code>if N.Key < N.Parent.Key return N.Parent else: return RightAncestor(N.Parent)</code>	LeftDescendant(N) <code>if N.Left = null return N else: return LeftDescendant(N.Left)</code>	Next(N) <code>if N.Right!= null: return LeftDescendant(N.Right) else: return RightAncestor(N)</code>
---------------------------------	---	--	--

3. חיפוש טווח- (y,x)-rangeSearch(y,x)- חיפוש איברים בעלי מפתח בטוויה בין X ל-Y

למציאת X עושים חיפוש - זמן הריצה הוא גובה העץ, אח"כ מפעלים את פונקציית next עד שmaguiim ל-Y – פועלה זו מתבצעת כמו סריקה תווית, لكن זמן הריצה במקרה הגרוע הוא (n)O

המימוש:

RangeSearch(x , y , R)
$L \leftarrow \emptyset$
$N \leftarrow \text{Find}(x,R) \text{ while } N.\text{Key} \leq y$
$\text{if } N.\text{Key} \geq x:$
$L \leftarrow L.\text{Append}(N)$
$N \leftarrow \text{Next}(N) \text{ return } L$
Return L

4. הכנסה- **Insert(k,R)** - הכנסת ערך K לעץ חיפוש.
מבצע את פונקציית החיפוש ומוצא את הצומת ש-K אמור להיות בנו, יוצר צומת חדש בעלי ערך K ומשרשר אותו להיות בן ימני או שמאלית בהתאם למפתח.

המימוש:

Insert(k , R)
$P \leftarrow \text{Find}(k,R)$
Add new node with key k as child of P

5. מחיקה- **Delete(N)** – מחיקת צומת N:

שיטת 1:

אם N הוא עלה: ניתן למחוק אותו ולשים במקומו NULL.

אם לא-N יש בן ייחד: נמחק אותו ונעשה מעקף מבבא של N לבן.

אם לא-N 2 בנים: יהיו X העוקב של N (הבן הכי שמאל של הבן הימני שלו), נשים במקום N את התוכן של X, נמחק את הצומת X ונעשה מעקף מבבא של X לבן הימני שלו (אם קיים).

-
-
-

שיטת 2:

- אם ל-N אין בן ימני: נמחק אותו ו암 יש לו בן שמאלי נעשה מעקב מאבא של N אליו.
- אם ל-N יש בן ימני: יהי X העוקב של N (הבן הכי שמאלי של הבן הימני שלו), נשים במקום N את התוכן של X, נמחק את הצומת X ונעשה מעקב מאבא של X לבן הימני שלו.

המימוש: (עפ"י שיטה 2)

Delete(N)
if N.Right = null: Remove N, promote N.Left else: X \leftarrow Next(N) \ \ X.Left = null Replace N by X, promote X.Right

סיכון זמני הריצעה של הפעולות:

זמן הריצעה	הפעולה
(h)-גובה העץ במקרה הגרוע (n)	find(k,R)
O(n)	next(N)
(h)-גובה העץ במקרה הגרוע (n)	rangeSearch(x,y)
O(n)	Insert(k,R)
	Delete(N)

בעץ חיפוש בינרי הסריקה התואכית תביא לנו מערך ממויין