

עצי חיפוש בינאריים - binary search tree

בעיית החיפוש היא בעיה נפוצה ואנו רוצים שיהיה לנו מבנה נתונים שיאפשר לנו לעשות פעולה זו בזמן יעיל, וכן שפעולות הכנסה ומחיקה תהיינה יעילות.

כמו"כ ישנם חיפושים מורחבים יותר הכוללים חיפוש איברים בטווח בין X ל-Y, וכן חיפוש אחר איבר הקרוב ביותר למפתח כלשהו, וגם זאת אנו רוצים שיעשה בזמן ריצה יעיל.

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

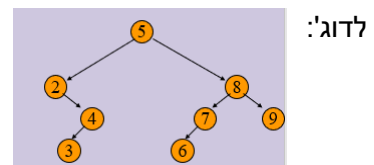
הפונקציה	הפעולה	מערך ורשימה מקושרת דו כיוונית	מערך ממוין	רשימה ממוינת	טבלאות גיבוב
Find	חיפוש	$O(n)$	$O(\log n)$	$O(n)$	$O(1)$ - במוצא
Insert	הוספה	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Delete	מחיקה	$O(n)$ - כשאין מצביע	$O(n)$	$O(n)$	$O(1)$ - במוצא
rangeSearch	חיפוש טווח	$O(n)$	$O(\log n)$	$O(n)$	$O(n+m)$
nearestNeighbors	חיפוש איבר סמוך	$O(n)$	$O(\log n)$	$O(n)$	$O(n+m)$

כדי שכל הפעולות תתבצענה בזמן ריצה יעיל נשתמש במבנה נתונים:

עץ חיפוש בינארי - עץ שעבור כל צומת מתקיים:

כל מפתחות הצמתים בתת עץ השמאלי קטנים מהמפתח שלו

וכל מפתחות הצמתים בתת עץ הימני גדולים מהמפתח שלו



הפעולות הבסיסיות:

1. **חיפוש** - $\text{find}(k, R)$: 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 בצד שמאל (אם קיים)

* אם k גדול מהמפתח שלו- המשך לחפש בצד ימין (אם קיים)

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

המימוש:

```

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- $next(N)$ - חיפוש צומת בעל מפתח מינימלי הגדול מהמפתח של N
החל מהצומת N :



המימוש:

If N.Parent=null Return null	RightAncestor(N)	LeftDescendant(N)	Next(N)
	if N.Key < N.Parent.Key return N.Parent else: return RightAncestor(N.Parent)	if N.Left = null return N else: return LeftDescendant(N.Left)	if N.Right≠ null: return LeftDescendant(N.Right) else: return RightAncestor(N)

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

למציאת X עושים חיפוש- זמן הריצה הוא גובה העץ, אח"כ מפעילים את פונקציית $next$ עד שמגיעים ל- Y – פעולה זו מתבצעת כמו סריקה תוכית, לכן זמן הריצה במקרה הגרוע הוא $O(n)$

המימוש:

```

RangeSearch(x , y , R )
L ← ∅
N ← Find(x,R) while N.Key ≤ y
  if N.Key ≥ x:
    L ← L.Append(N)
  N ← Next(N) return L
Return L

```

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

המימוש:

```

Insert(k , R )
P ← Find(k,R)
Add new node with key k as child of P

```

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

שיטה 1:

- אם N הוא עלה: ניתן למחוק אותו ולשים במקומו $NULL$.
- אם ל- N יש בן יחיד: נמחק אותו ונעשה מעקף מאבא של N לבן.
- אם ל- N 2 בנים: יהי X העוקב של N (הבן הכי שמאלי של הבן הימני שלו), נשים במקום N את התוכן של X , נמחק את הצומת X ונעשה מעקף מאבא של X לבן הימני שלו (אם קיים).

שיטה 2:

- אם ל-N אין בן ימני: נמחק אותו ואם יש לו בן שמאלי נעשה מעקף 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 מעקף מאבא של X לבן הימני שלו.

המימוש: (עפ"י שיטה 2)

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

סיכום זמני הריצה של הפעולות:

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

בעץ חיפוש בינרי הסריקה התוכית תביא לנו מערך ממזין