**УНИВЕРЗИТЕТ ,,ГОЦЕ ДЕЛЧЕВ” – ШТИП**

**ФАКУЛТЕТ ЗА ИНФОРМАТИКА**

**A picture containing calendar

Description automatically generated**

**СЕМИНАРСКА РАБОТА**

**Тема : Пребарувачки стебла**

Ментор : Проф. Цвета Мартиновска Банде

Изработил : Мартин Ристов 102620

**ЈАНУАРИ 2023**

**Содржина**

[**1.** **Пребарувачки стебла** 3](#_Toc126078726)

[**1.1.** **Врска помеѓу бројот на јазли во бинарно стебло и неговата висина** 3](#_Toc126078727)

[**1.2.** **Балансирани бинарни стебла** 3](#_Toc126078728)

[**1.3.** **Бинарни пребарувачки стебла** 5](#_Toc126078729)

[**2.** **AVL Стебла** 10](#_Toc126078730)

[**2.1.** **Што е AVL Стебло ?** 10](#_Toc126078731)

[**2.2.** **Како да се балансира AVL Стебло** 11](#_Toc126078732)

[**2.2.1.** **Десна ротација** 11](#_Toc126078733)

[**2.2.2.** **Операција на лева ротација** 12](#_Toc126078734)

[**2.2.3.** **Техники за ребалансирање** 13](#_Toc126078735)

[**2.3.** **Вметнување на јазел** 15](#_Toc126078736)

[**2.4.** **Бришење јазол** 16](#_Toc126078737)

[**2.5.** **Пребарување на јазел** 18](#_Toc126078738)

[**Користена литература** 19](#_Toc126078739)

# **Пребарувачки стебла**

# **Врска помеѓу бројот на јазли во бинарно стебло и неговата висина**

Доколку бинарното стебло не е празно и е максимално пополнето, на ниво 0 тоа има еден јазел, на ниво 1, има два јазли, на ниво 2 има четири јазли или во општ случај, на ниво к има 2к-1 јазли. Доколку сите јазли се соберат добиваме дека за максимално пополнето бинарно стебло важи дека доколку има н јазли, тој број е еднаков на самата сума

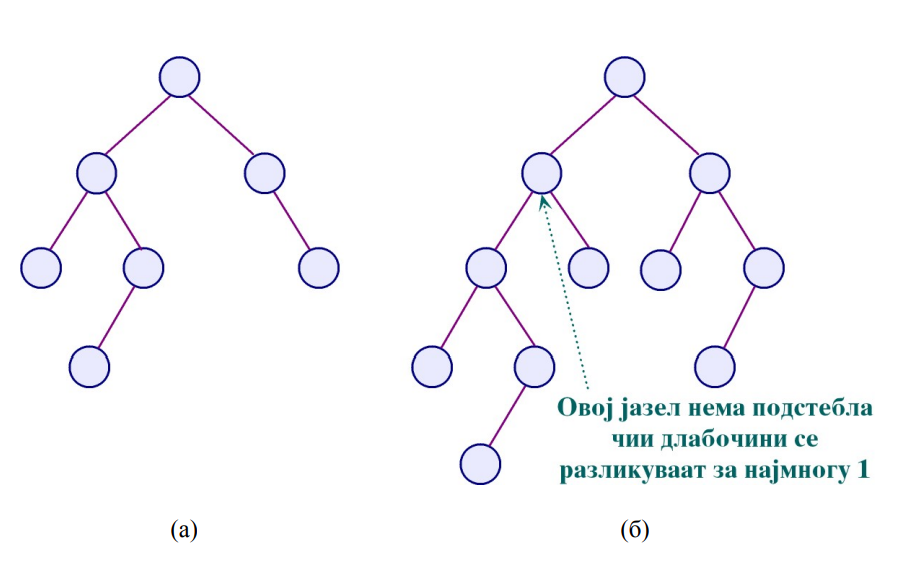
**1 + 21 + 22 + . . . + 2d-1 = 2d – 1**

Каде d е длабочината на стеблото. Од тука може да заклучиме дека за секое стебло со n јазли важи **n ≤ 2d – 1** од каде пак следи дека минмалната длабочина (висина) на бинарното стебло со n јазли (која се добива кога е тоа максимално пополнето) е **dmin = [lg (n + 1)]** каде “ А ” го означува најмалиот цел број еднаков или поголем од А. Од тука следи дека n=20, имаме

**dmin = lg 21 = 4.39 = 5** ; додека за n=1000 се добива **dmin = lg 1001 = 10.** Ова е значајно затоа што најголемата висина на стебло со 1000 јазли е 1000 ( во случај каде е секогаш активна само една врска) . Доколку претпоставиме дека стеблото како податочна структура се користи за складирање на некоја информација и под претпоставка дека бараната информација се наоѓа во листовите на стеблото ( на пример при градење на индекс), од интерес ќе биде нивото на секој јазел во стеблото да биде еднакво или приближно еднакво . Таквите бинарни стебла се нарекуваат балансирани бинарни стебла.

# **Балансирани бинарни стебла**

Балансирано бинарно стебло е она бинарно стебло каде што за секое подстебло на произволен јазел важи дека висините не се разликуваат за повеќе од еден. Балансираното бинарно стебло уште е познато и како балансирано стебло во висина. Се дефинира како бинарно стебло кога разликата помеѓу висината на левото подстебло и десното подстебло не е поголема од m, каде што m, обично е еднаква на 1. Висината на стеблото е бројот на рабовите на најдоглата патека помеѓу коренскиот јазол и јазолот на листот. На слика 1а , е претставено балансирано бинарно стебло. На слика 1б е прикажано небалансирано бинарно стебло кое е такво поради тоа што за посочениот јазел не важи дека висините на неговото лево и десно подстебло се еднакви или се разликуваат за еден. Во конкретниот случај висините на левото и десното подстебло на посочениот јазел се разликуваат за два.



**Слика 1а балансирано и слика 1б небалнасирано бинарно стебло**

**Што е балансирано бинарно стебло ?**

Балансираните бинарни стебла се пресметковно ефикасни за извршување на операции.

Балансираното бинарно стебло ќе ги следи следниве услови :

*1: Апсолутната разлика на висините на левото и десното подстебло на било кој јазел е помал од 1.*

*2: За секој јазол, неговото лево подстебло е балансирано бинарно стебло.*

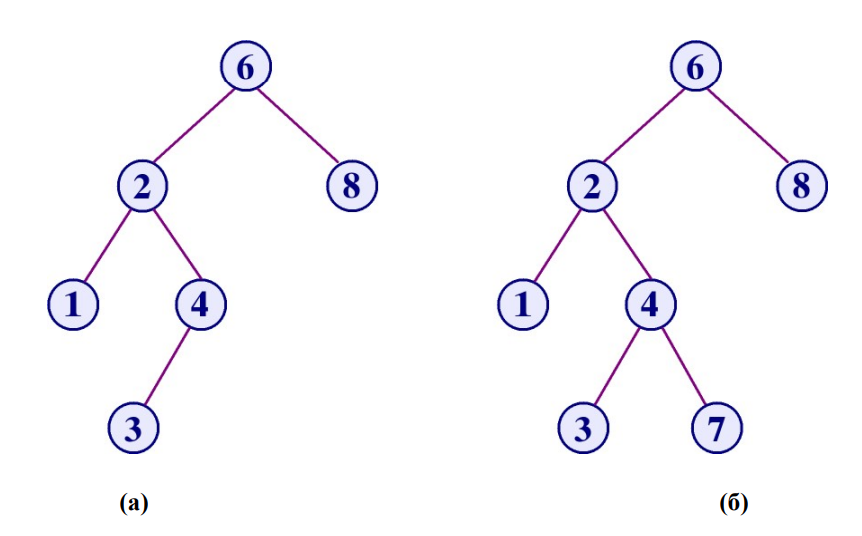
*3: За секој јазол, неговото десно подстебло е балансирано бинарно стебло.*

**Балансирани бинарни стебла во висина**

Балансираните бинарни стебла се познати и како балансирани бинарни стебла со висина. Бинарни стебла кој се избалансирани со висина може да се означат со HB(k), каде што k е разликата помеѓу висините на лево и десно подстебло . ,, k ” е познат како фактор на рамнотежа. Ако за стеблото, факторот на рамнотежа (k) е еднаков на нула, тогаш тоа стебло е позната како целосно избалансирано бинарно стебло . И може да се означи како HB(O).

# **Бинарни пребарувачки стебла**

Една од важна примена на стеблата е во областа на пребарувањето . Да претпоставиме дека на секој јазел од бинарното стебло е доделена една вредност наречена клуч. Бинарното стебло се нарекува пребарувачко доколку за секој јазел од левото подстебло на јазелот со клуч X соддржи вредности(клучеви) помали од Х и секој јазел од десното подстебло на јазелот со клучот Х , содржи јазли со клучеви поголеми од Х ( за сега се претпоставува дека не постојат дупликати на клучевите). На слика 1.2а е прикажано бинарно пребарувачко стебло и обично бинарно стебло на слика 1.2б. Второто стебло е е пребарувачко поради тоа што јазелот со клуч 7 не се наоѓа во десното подстебло на јазелот со клуч 6.



**Бинарно пребарувачко стебло 1.2а и обично бинарно стебло 1.2б**

Јазелот со најмал клуч е јазелот до кој се даоѓа доколку од коренот се оди само по левите врски се додека не се дојде до листот на стеблото. Јазелот со најголем клуч е листот до кој се доаѓа со одењето по по само десните врски почнувајќи од коренот на стеблото . Доколку стеблото се измине во инордер , се добива подредена низа од клучеви и тоа во растечки редослед.

Типична операција која се извршува над бинарното пребарувачко стебло е операцијата на пребарување . Таа операција може да се реализира со кодот на следната рекурзивна функција :

// Daden e pokazuvac kon korenot na stebloto i kluc k, TREE-SEARCH vrakja pokazuvac kon jazel so kluc k ako takov postoi; vo obraten slucaj, vrakja NIL. //

TREE-SEARCH (x, k)

1 if x = NIL or k = key[x]

2 then return x

3 if k < key[x]

4 then return TREE-SEARCH (left[x], k)

5 else return TREE-SEARCH (right[x], k)

Истата функција во нерекурзивна изведба би била

ITERATIVE-TREE-SEARCH (x,k)

1 while x ≠ NIL and k ≠ key[x]

2 do if k < key[x]

3 then x ← left[x]

4 else x ← right[x]

5 return x

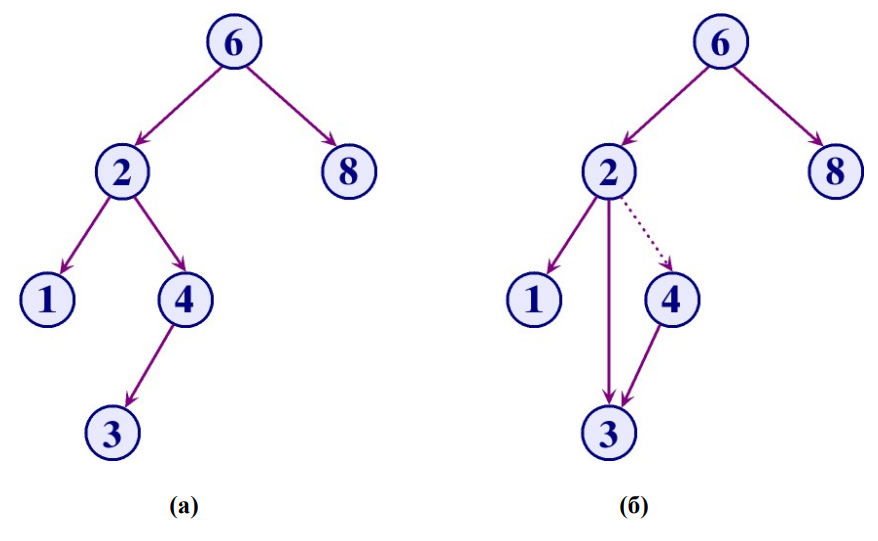
Кодот за градење (вметнување на јазел на бинарно пребарувачко стебло) е релативно едноставен затоа што се сведува на движење низ јазлите на стеблото се додека не се дојде до јазел кој е најблизок (помал или поголем) од клучот од јазелот што треба да се вметне. Јазелот што се вметнува е секогаш терминален јазел (слика 1.3).

A picture containing polygon

Description automatically generated

**Слика 1.3 Бинарно пребарувачко стебло пред (а) и по (б) вметнување на јазел со клуч 5**

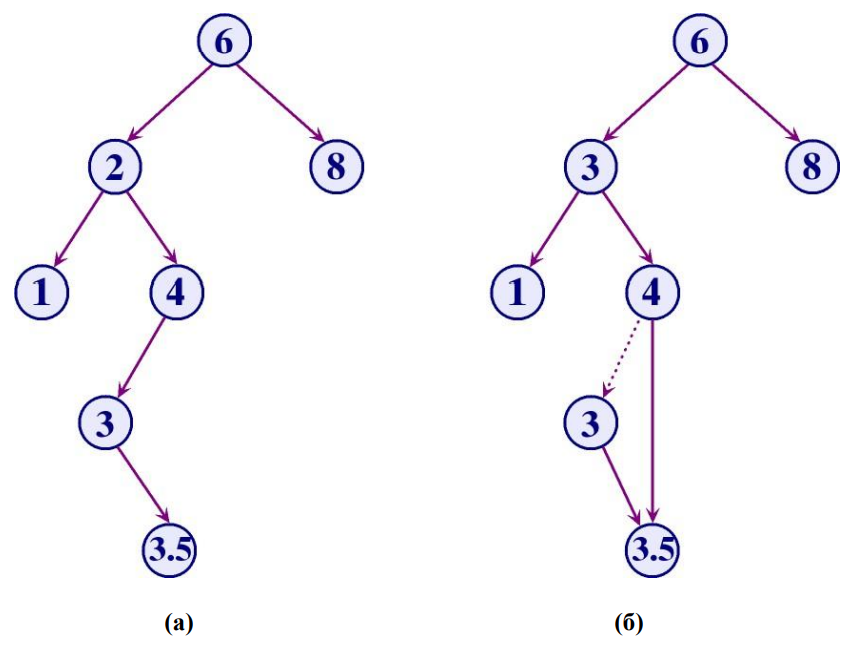
Операција на бришење на клуч (односно јазел што го содржи клучот) од бинарно пребарувачко стебло е покомлексна операција од операцијата вметнување . Тоа се должи на фактот што и по бришњето на саканиот јазел, стеблото мора да ги задоволува условите кои се барани од него за тоа да биде пребарувачко . Поради тоа, при бришење на јазел од бинарно стебло ќе бидат разгледани неколку случаио: Доколку јазелот што треба да се избрише е лист , бришењето е едноставно и се врши со едноставно премостување на покажувачот на јазелот родител (покажувачот кон јазелот што треба да се избрише станува нулев покажувач).



**Слика 1.4 Бришење на јазелот со едно дете(клуч 4) пред (а) и после операцијата (б)**

Најкомплексен е случајот кога јазелот што треба да се избрише има две деца. Алгоритмот на бришење којо треба да се реализира се состои од само два чекори. Во првиот чекор се наоѓа најмалиот клуч од десното подстебло на јазелот што треба да се избрише. Во вториот чекор тој се заменува со клучот од јазелот кој треба да се избрише. Во третиот чекор се брише јазелот од кој се искористил клучот во претходниот чекор. Овој случај на бришење е едноставен затоа што јазелот што го содржи најмалиот клуч во некое подстебло мора да има нулева лева врска , па согласно претходната анализа бришењето се сведува на премостување. Важно е да се разбере дека со оваа замена стеблото останува балансирано, поради тоа што сите јазли од десното подстебло се поголеми од јазелот што сакаме да го избришеме, па согласно тоа по неговото бришење тиа ќе бидат поголеми од најголемиот јазел во тоа подстебло .

Илустрацијата на овој случај на бришење прикажана на слика 1.4.



**Слика 1.4 Бришење на јазелот со две деца(клуч 2) пред (а) и после операцијата (б)**

Треба да се забележи дека случајот на бришење на јазол со две деца работи и кога клучот на јазелот што треба да се избрише ќе се замни со најголемиот клуч од неговото лево подстебло.

**Зошто?**

Една можна реализација на операцијата бришење е дадека со следната рекурзивна функција :

delete( element\_type x, SEARCH\_TREE T )

{

tree\_ptr tmp\_cell, child;

if( T == NULL )

error("Elementot ne e pronajden");

else

if( x < T->element ) /\* Odi levo \*/

T->left = delete( x, T->left );

else

if( x > T->element ) /\* Odi desno \*/

T->right = delete( x, T->right );

else /\* Najdeniot element da se izbrise \*/

if( T->left && T->right ) /\* Dve deca \*/

{ /\* Zameni so najmaliot od desnoto podsteblo \*/

tmp\_cell = find\_min( T->right );

T->element = tmp\_cell->element;

T->right = delete( T->element, T->right );

}

else /\* Edno dete \*/

{

tmp\_cell = T;

if( T->left == NULL )

/\* Samo desno dete \*/

child = T->right;

if( T->right == NULL )

/\* Samo levo dete \*/

child = T->left;

free( tmp\_cell );

return child;

}

return T;

}

Времетраењето на сите разгледани операции (пребарување, вметнување на јазел, бришење на јазел) кај бинарните пребарувачки стебла зависи од висината на јазелот кој треба да се обработи . Како што утврдивме на почетокот на ова подглавје нивото на јазлите во стебло со n елементи може значајно да варира во интервалот од log2n до n. Доколку сакаме подобри перформанси стеблата треба да бидат балансирани.

# **AVL Стебла**

# **Што е AVL Стебло ?**

Стеблото AVL, именувано по неговите пронаоѓачи Аделсон-Велски и Лендис, е самобалансирачко бинарно стебло за пребарување (BST). **Самобалансирачко стебло е бинарно стебло за пребарување кое ја балансира висината по вметнување и бришење според некои правила за балансирање.** Временската сложеност во најлош случај на бинарно стебло за пребарување е функција на висината на стеблото. Поточно, најдолгата патека од коренот на стеблото до јазол. За BST со N јазли, да речеме дека секој јазол има само нула или едно дете. Затоа неговата висина е еднаква на N, а времето за пребарување во најлош случај е O(N). Значи, нашата главна цел во BST е да ја задржиме максималната висина блиску до log(N).

Факторот на рамнотежа на јазолот N е *висина(десно(N)) – висина(лево(N))*. **Во AVL стеблото, факторот на рамнотежа на јазол може да биде само една од вредностите 1, 0 или -1.**

**Код за стеблото :**

**public** **class** **Node** {

**int** key;

**int** height;

Node left;

Node right;

... }

**Код за AVL Стебло :**

**public** **class** **AVLTree** {

**private** Node root;

**void** **updateHeight**(Node n) {

n.height = 1 + Math.max(height(n.left), height(n.right));

}

**int** **height**(Node n) {

**return** n == null ? -1 : n.height;

}

**int** **getBalance**(Node n) {

**return** (n == null) ? 0 : height(n.right) - height(n.left);

}

...

}

# **Како да се балансира AVL Стебло**

Стеблото AVL го проверува факторот на рамнотежа на неговите јазли по вметнување или бришење на јазол. Ако факторот на рамнотежа на јазолот е поголем од еден или помал од -1, стеблото се ребалансира.

Постојат две операции за ребалансирање на стеблото:

* десна ротација и
* лева ротација.

## **Десна ротација**

Да почнеме со вистинската ротација.

Да претпоставиме дека имаме BST наречен T1, со Y како корен јазол, X како лево дете на Y и Z како десно дете на X. Со оглед на карактеристиките на BST, знаеме дека X <Z <Y.По десна ротација на Y, имаме стеблото наречено T2 со X како корен и Y како десно дете на X и Z како лево дете на Y. T2 сèуште е BST бидејќи го задржува редот X < Z < Y .

A picture containing watch, clock

Description automatically generated

Ајде да ја разгледаме вистинската операција за ротација за нашето AVLTree:

Node **rotateRight**(Node y) {

**Node** x = y.left;

**Node** z = x.right;

x.right = y;

y.left = z;

updateHeight(y);

updateHeight(x);

**return** x;

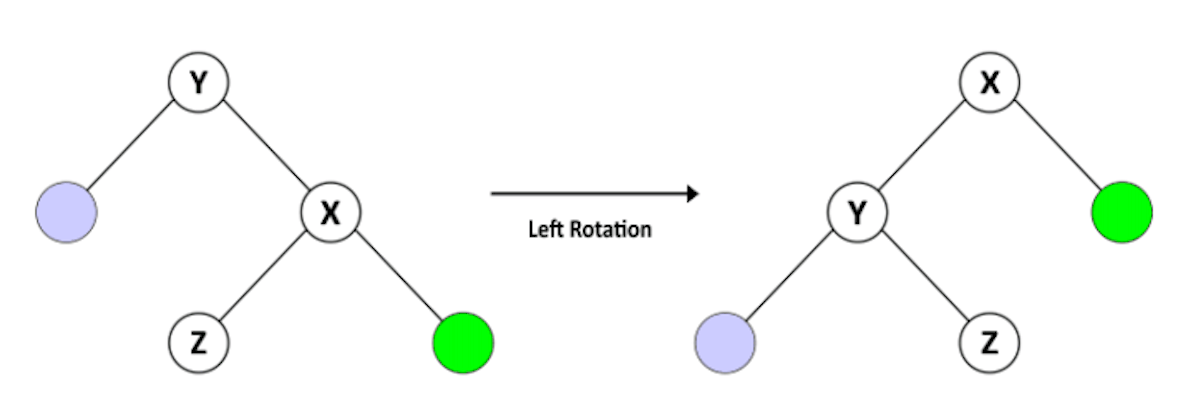
}

## **Операција на лева ротација**

Имаме и операција на лева ротација.

Претпоставете BST наречен T1, со Y како корен јазол, X како десно дете на Y и Z како лево дете на X. Со оглед на ова, знаеме дека Y <Z < X.

По левата ротација на Y, имаме стеблото наречено T2 со X како корен и Y како лево дете на X и Z како десно дете на Y. T2 сè уште е BST бидејќи го задржува редот Y < Z < X .



Ајде да ја погледнеме операцијата на лево ротирање за нашето AVLTree:

Node **rotateLeft**(Node y) {

**Node** x = y.right;

**Node** z = x.left;

x.left = y;

y.right = z;

updateHeight(y);

updateHeight(x);

**return** x;

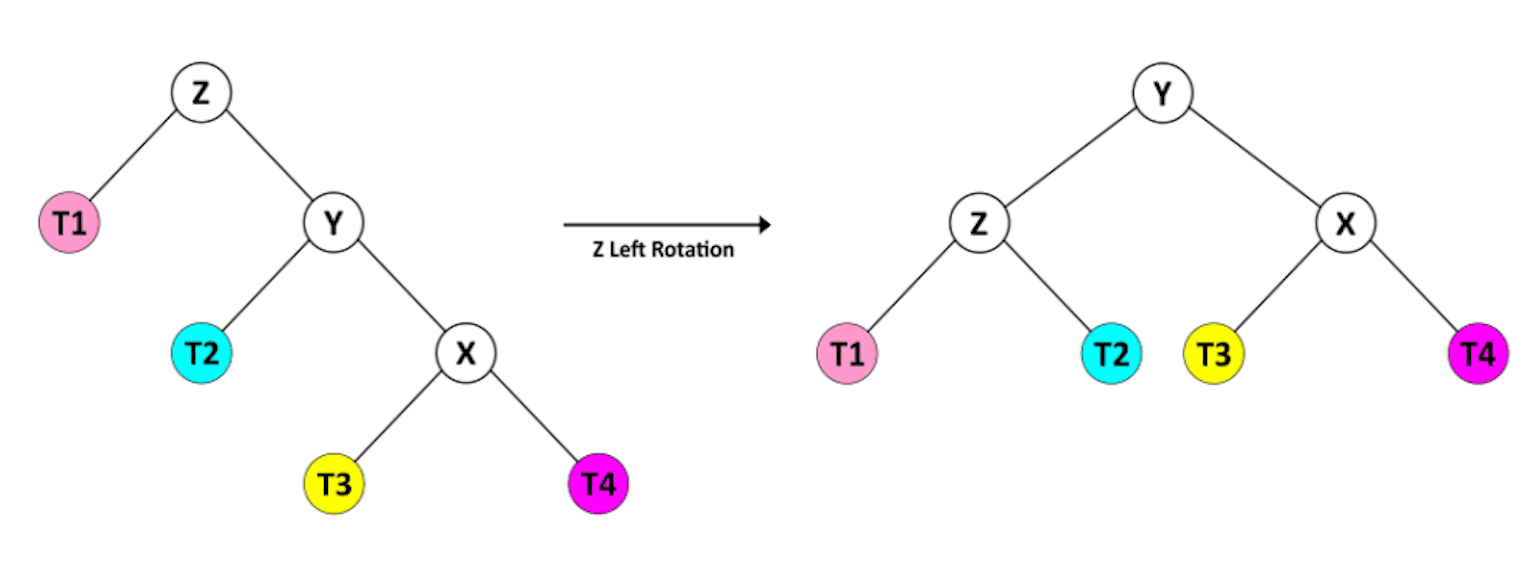
}

## **Техники за ребалансирање**

Можеме да ги користиме операциите на десната ротација и левата ротација во посложени комбинации за да **го одржиме AVL стеблото балансирано по каква било промена во неговите јазли**. Во неурамнотежена структура, барем еден јазол има фактор на рамнотежа еднаков на 2 или -2. Ајде да видиме како можеме да го балансираме стеблото во овие ситуации.

Кога факторот на рамнотежа на јазолот Z е 2, подстеблото со Z како корен е во една од овие две состојби, сметајќи го Y како десното дете на Z.

За првиот случај, висината во десното дете од Y (X) е поголема од висината на левото дете (T2). Можеме лесно да го ребалансираме стеблото со лево ротирање на Z.



За вториот случај, висината на десното дете на Y (T4) е помала од висината на левото дете (X). Оваа ситуација бара комбинација од операции на ротација.

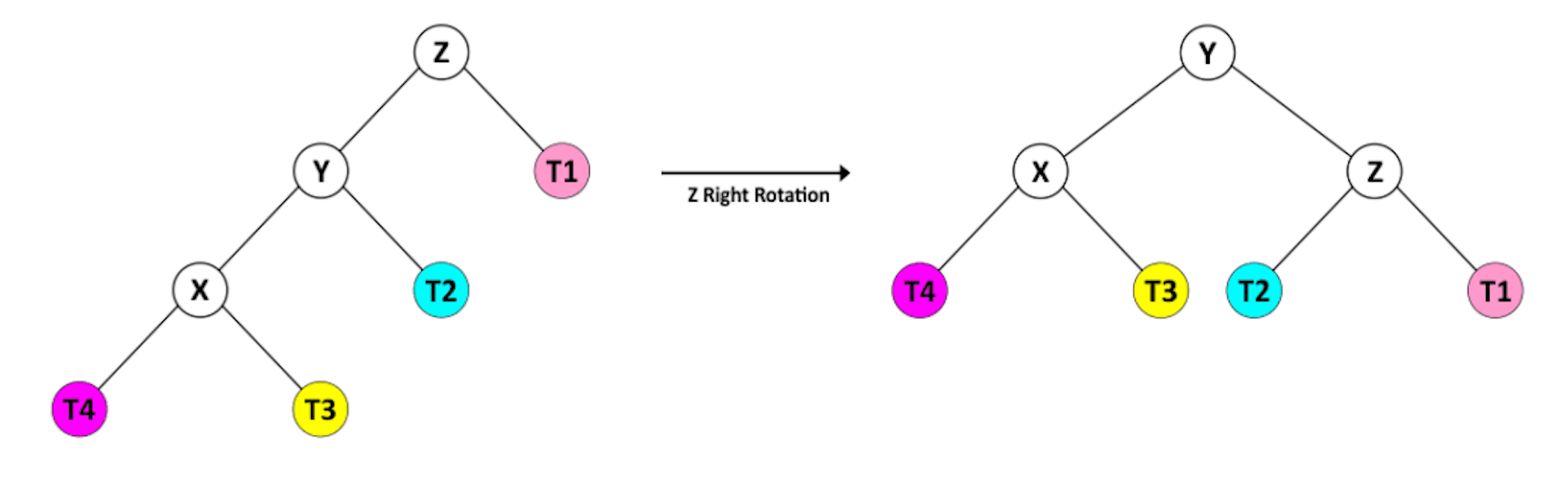
Diagram, radar chart

Description automatically generated with medium confidence

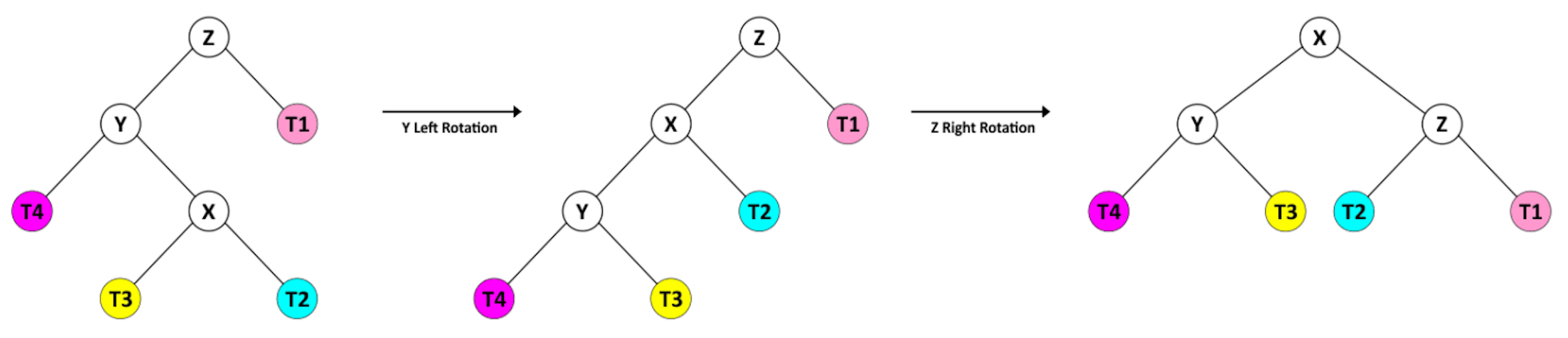
Во овој случај, прво го ротираме Y надесно, така што стеблото добива иста форма како и претходниот случај. Потоа можеме да го ребалансираме стеблото со лева ротација на Z.

Исто така, кога факторот на рамнотежа на јазолот Z е -2, неговото подстебло е во една од овие две состојби, така што го сметаме Z како корен и Y како негово лево дете.

Висината во левото дете на Y е поголема од онаа на десното дете, така што го балансираме стеблото со десната ротација на Z.



Или во вториот случај, десното дете на Y има поголема висина од неговото лево дете



Значи, прво го трансформираме во поранешната форма со лева ротација на Y, потоа го балансираме стеблото со десната ротација на Z.

Ајде да ја погледнеме операцијата за ребаланс за нашето AVLTree:

Node **rebalance**(Node z) {

updateHeight(z);

**int** balance = getBalance(z);

**if** (balance > 1) {

**if** (height(z.right.right) > height(z.right.left)) {

z = rotateLeft(z);

}

**else**

{

z.right = rotateRight(z.right);

z = rotateLeft(z);

} }

**else** **if** (balance < -1)

{ **if** (height(z.left.left) > height(z.left.right)) z = rotateRight(z); **else** { z.left = rotateLeft(z.left);

z = rotateRight(z);

} }

**return** z;

}

**Ќе користиме ребаланс откако ќе вметнеме или бришеме јазол за сите јазли на патеката од променетиот јазол до коренот.**

# **Вметнување на јазел**

Кога ќе вметнеме клуч во стеблото, треба да ја лоцираме неговата правилна позиција за да ги поминеме правилата BST. Значи, започнуваме од коренот и ја споредуваме неговата вредност со новиот клуч. Ако клучот е поголем, продолжуваме надесно - во спротивно, одиме на левото дете.

Откако ќе го најдеме соодветниот родителски јазол, тогаш го додаваме новиот клуч како јазол лево или десно, во зависност од вредноста.

По вметнувањето на јазолот, имаме BST, но можеби не е AVL Tree. Затоа, ги проверуваме факторите на рамнотежа и го ребалансираме BST за сите јазли на патеката од новиот јазол до коренот.

Ајде да ја погледнеме операцијата за вметнување:

Node **insert**(Node node, **int** key) {

**if** (node == null) {

**return** **new** **Node**(key);

}

**else** **if** (node.key > key)

{

node.left = insert(node.left, key);

}

**else** **if** (node.key < key)

{

node.right = insert(node.right, key);

} **else**

{ **throw** **new** **RuntimeException**("duplicate Key!");

}

**return** rebalance(node);

}

**Важно е да се запамети дека клучот е единствен во стеблото** **- нема два јазли кои го делат истиот клуч.** Временската сложеност на алгоритмот за вметнување е функција од висината. Бидејќи нашето стеблото е избалансирано, можеме да претпоставиме дека временската сложеност во најлош случај е O(log(N)).

# **Бришење јазол**

За да избришеме клуч од стеблото, прво треба да го најдеме во BST. Откако ќе го пронајдеме јазолот (наречен Z), треба да го претставиме новиот кандидат кој ќе биде негова замена во стеблото. Ако Z е лист, кандидатот е празен. Ако Z има само едно дете, ова дете е кандидатот, но ако Z има две деца, процесот е малку покомплициран. Да претпоставиме дека десното дете на Z се нарекува Y. Прво, го наоѓаме најлевиот јазол на Y и го нарекуваме X. Потоа, ја поставуваме новата вредност на Z еднаква на вредноста на X и продолжуваме да го бришеме X од Y.

Конечно, го нарекуваме методот на ребаланс на крајот за да го задржиме BST како AVL стебло.

Еден метод за бришење:

Node **delete**(Node node, **int** key) {

**if** (node == null)

{

**return** node;

}

**else** **if** (node.key > key)

{

node.left = delete(node.left, key);

}

**else** **if** (node.key < key)

{

node.right = delete(node.right, key);

}

**else**

{

**if** (node.left == null || node.right == null)

{

node = (node.left == null) ? node.right : node.left;

}

**Else**

{

**Node** mostLeftChild = mostLeftChild(node.right);

node.key = mostLeftChild.key;

node.right = delete(node.right, node.key);

} }

**if** (node != null) {

node = rebalance(node);

}

**return** node;

}

Временската сложеност на алгоритмот за бришење е функција од висината на стеблото. Слично на методот на вметнување, можеме да претпоставиме дека временската сложеност во најлош случај е O(log(N)).

# **Пребарување на јазел**

Пребарувањето за јазол во стеблото AVL е исто како и кај кое било BST.

Започнете од коренот на стеблото и споредете го клучот со вредноста на јазолот. Ако клучот е еднаков на вредноста, вратете го јазолот. Ако клучот е поголем, барајте од десното дете, инаку продолжете со пребарувањето од левото дете.

Временската сложеност на пребарувањето е во функција на висината. Можеме да претпоставиме дека временската сложеност во најлош случај е **O(log(N)).**

**Пример код:**

Node **find**(**int** key) {

**Node** current = root;

**while** (current != null)

{

**if** (current.key == key)

{

**break**;

}

current = current.key < key ? current.right : current.left;

}

**return** current;

}

# **Користена литература**

* [**https://www.baeldung.com/java-avl-trees**](https://www.baeldung.com/java-avl-trees)
* [**https://moodle2.ugd.edu.mk/pluginfile.php/14550/mod\_resource/content/1/6.%20Prebaruvacki%20steblav2006ok.pdf**](https://moodle2.ugd.edu.mk/pluginfile.php/14550/mod_resource/content/1/6.%20Prebaruvacki%20steblav2006ok.pdf)