Балансирани и идеално балансирани дървета

Изготвил:
гл.ас. д-р Нора Ангелова

Балансирано и идеално балансирано дърво

Двоични наредени дървета

- Средна сложност на операциите добавяне и търсене

O(log N), N – брой на върховете в дървото

- Добавяне на елементите подредени по големина

O(N), N – брой на върховете в дърво
то

Идеално балансирано двоично наредено дърво

- Броят на възлите в лявото и дясното поддърво се различава най-много с 1.
- Лявото и дясното поддървета са идеално балансирани двоично наредени дървета.

`O(log N), N – брой на върховете в дървото`

Балансирано двоично наредено дърво

- Височините на лявото и дясното поддърво се различават най-много с 1.
- Лявото и дясното поддървета са балансирани двоично наредени дървета.

`O(log N), N – брой на върховете в дървото`

- Всяко идеално балансирано дърво е балансирано дърво, но обратното не е вярно.
- Алгоритмите за добавяне и премахване на връх от двоично наредено дърво не запазват балансираността и добри сложности.
- Съществуват алгоритми, които запазват балансираността.
- Съществува прост алгоритъм за създаване на идеално балансирано двоично наредено дърво при определени ограничения.

Алгоритъм за
създаване на идеално балансирано дърво

- Елементите, които ще се включват към празното двоично наредено дърво се подават в нарастващ ред.
- Предварително е известен броят на върховете на дървото.

```
n - брой на върховете в дървото
n = nLeft + nRight + 1 && |nLeft-nRight| <= 1
```

Алгоритъм за
създаване на идеално балансирано дърво

`n - брой на върховете в дървото`

`n = nLeft + nRight + 1 && |nLeft-nRight| <= 1`

- Създава връх на дървото
- Конструира ляво поддърво с `nLeft` върха
- Конструира дясно поддърво с `nRight` върха

Алгоритъм за създаване на идеално балансирано дърво

```
template <class T>
BinOrdTree<T>::BinOrdTree(int n)
{
  if (n == 0)
    root = NULL;
  else
  {
    int nLeft = (n-1)/2;
    int nRight = n - nLeft - 1;
    BinOrdTree<T> t1(nLeft);
    T x; cin >> x;
    BinOrdTree<T> t2(nRight);
    Create3(x, t1, t2);
  }
}
```

Балансирано дърво

- **Ротации** - операции, които пренареждат част от елементите на дървото при добавяне или при премахване на елемент от него, за да се избегне дисбаланса му.
- Ротациите зависят от реализацията на конкретната структура от данни. Примери за такива структури:
  - **червено-черно** дърво
  - **AVL**-дърво
  - **AA**-дърво и др.

AVL дърво

- **Търсене –** осъществява се по същия начин както при обикновено небалансирано двоично наредено дърво

Най-лош сценарий:
Трябва да се обходи от корена до най-далечното листо.
Време: пропорционално на височината на дървото – `O(long n)`

AVL дърво

- **Обхождане –** осъществява се по същия начин както при обикновено небалансирано двоично наредено дърво

Обхождат се всички върхове – O(n)

AVL дърво

Добавяне на връх

- Върхът се добавя като листо
- Повторно обхождане **`(retracing)`** - проверка за съответветствие с инвариантите на AVL дърво. Започва се от родителя на добавения връх и се стига до корена.

Реализира се чрез пресмятане на баланс фактор `(balance factor)` за всеки връх, който се дефинира като разликата във височините на лявото и дясното поддървета.

`balanceFactor – цяло число в интервала [-1 1]`

AVL дърво

Добавяне на връх

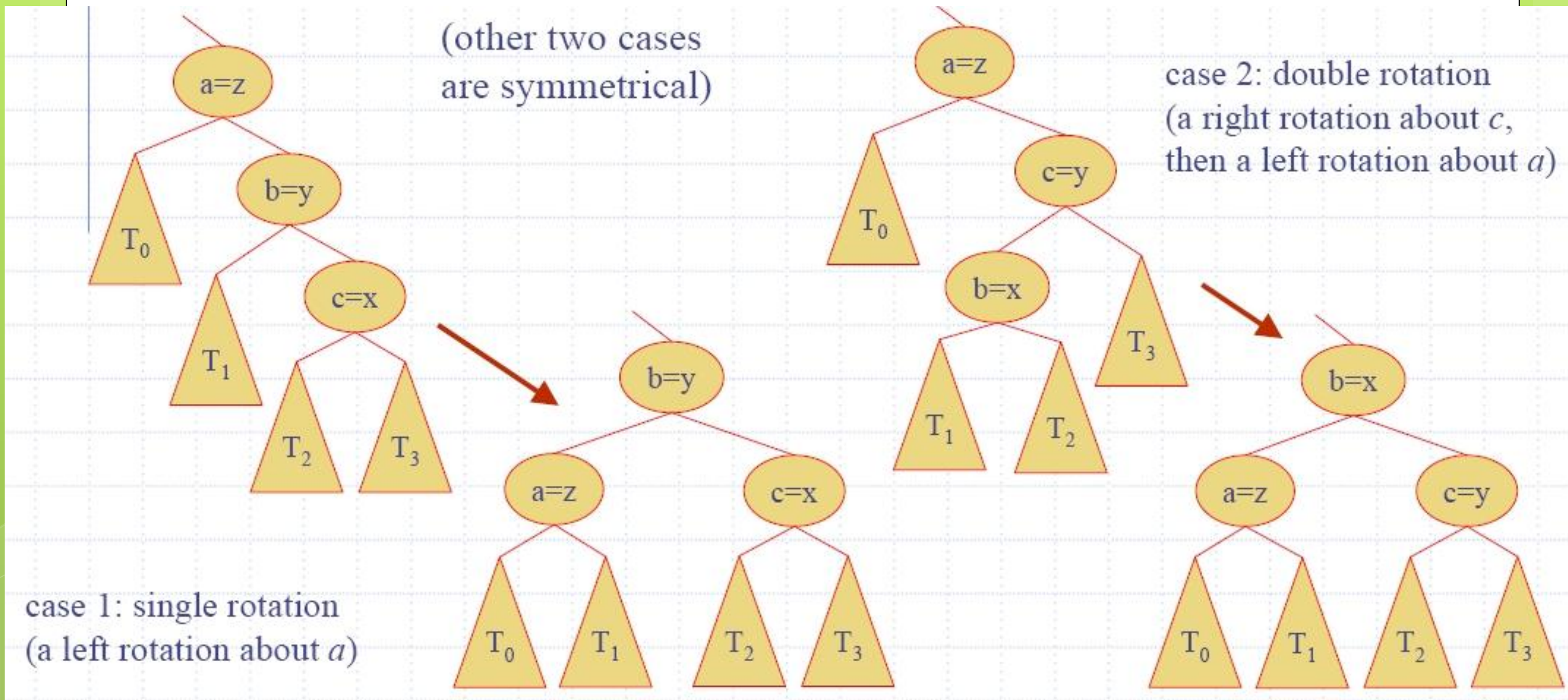
- Връх с `balanceFactor ∊ [-1 1]` - поддървото, чийто корен е съответният връх, е балансирано и не е необходима ротация.

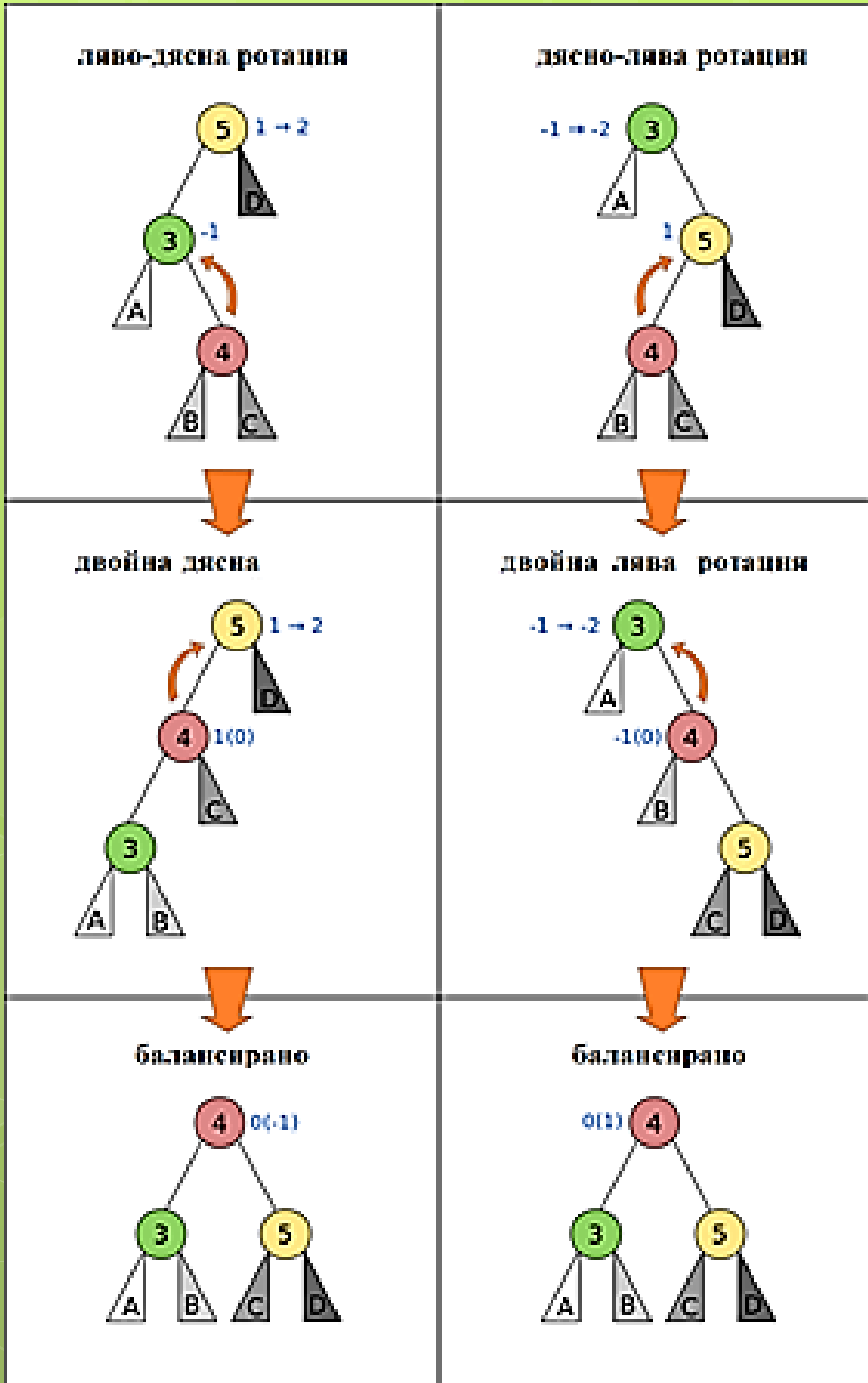
- Връх с `balanceFactor ∊ [-2 2] -` поддървото, чийто корен е съответният връх, е небалансирано и е необходима ротация.

AVL дърво

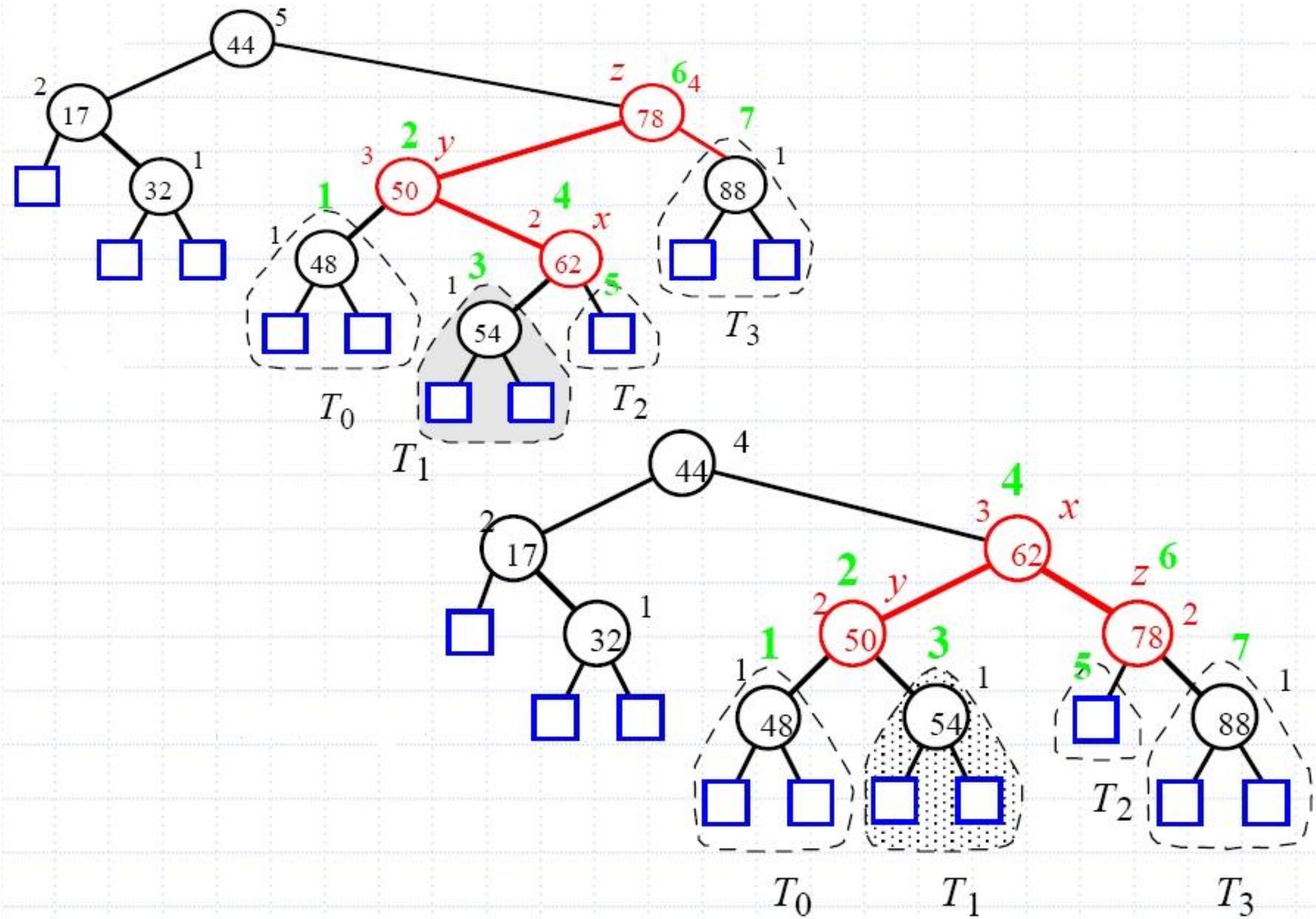
Ротационни функции

- Двойна дясна ротация (**Right Right Case**)
- Двойна лява ротация (**Left Left Case**)
- Двойна ляво-дясна ротация (състои се от лява ротация, последвана от дясна) (**Left Right Case**)
- Двойна дясно-лява ротация (състои се от дясна ротация, последвана от лява) (**Right Left Case**)


(other two cases are symmetrical)
case 1: single rotation (a left rotation about a)
case 2: double rotation (a right rotation about c, then a left rotation about a)
a=z
b=y
c=x
b=x
c=y
T0
T1
T2
T3


ляво-дясна ротация
дясно-лява ротация
двойна дясна
двойна лява ротация
балансирано
балансирано

ляво-дясна ротация



AVL дърво

Изтриване на връх

- Изтриване на дърво
- Повторно обхождане **`(retracing)`** - проверка за съответветствие с инвариантите на AVL дърво. Започва се от родителя на изтрития връх и се стига до корена.

```
cout << “КРАЙ”;
```