Lab Tutorial 7A

AVL Tree

Nama berkas kode sumber	:	SDA<npm>L7A.java
Batas waktu eksekusi program	:	3 detik / kasus uji
Batas memori program	:	256 MB / kasus uji

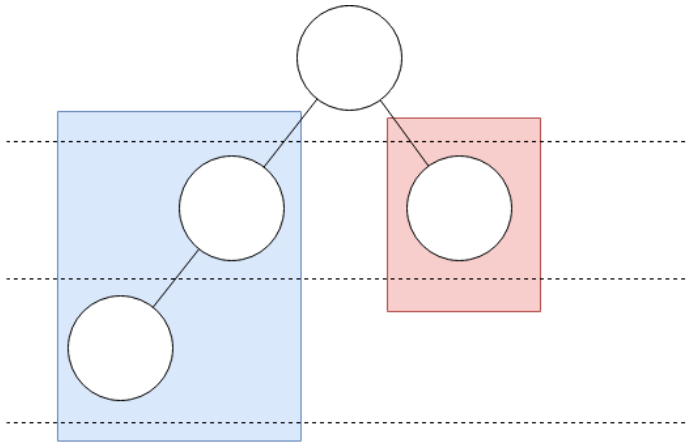
Revisi 1: Ada perbaikan deskripsi soal untuk perintah PRINT

Binary Search Tree yang tidak *balance* dapat membuat seluruh operasi memiliki kompleksitas running time O(N) pada kondisi *worst case*. Untuk menanggulangi masalah tersebut, maka dibuatlah sebuah variant dari *Binary Search Tree* yang dinamakan *AVL Tree*. *AVL Tree* adalah *Binary Search Tree* yang sudah *balance*.

A. Tutorial

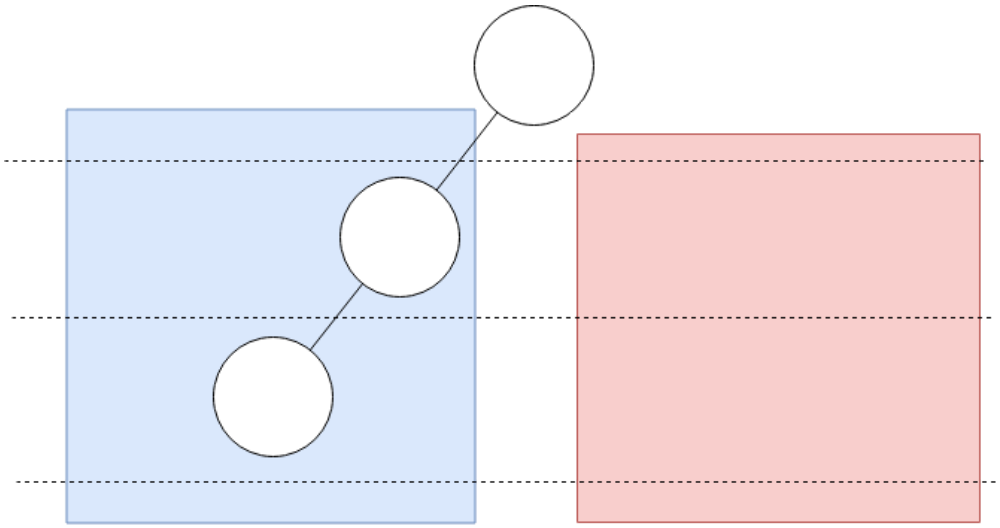
Syarat balance suatu AVL Tree adalah subtree kiri dan subtree kanan dari suatu node memiliki perbedaan tinggi sebesar **maksimal** 1.

Berikut adalah contoh AVL Tree yang balanced.



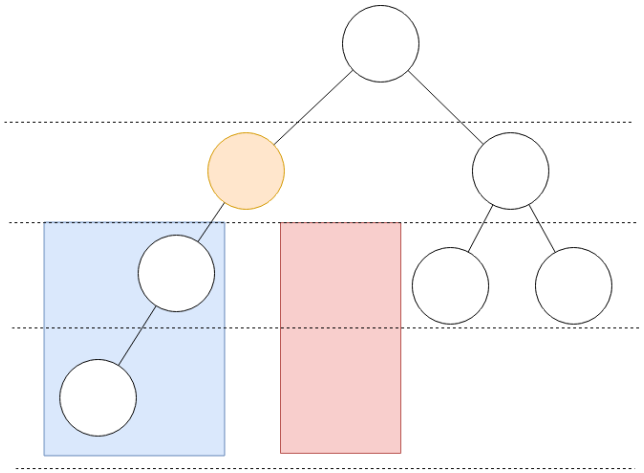
Kita dapat melihat bahwa subtree kiri dari root (berwarna biru) memiliki beda height sebesar 1 dibandingkan dengan subtree kanan (berwarna merah)

Berikut adalah contoh AVL Tree yang tidak balanced.



Kita dapat melihat bahwa subtree kiri dari root (berwarna biru) memiliki beda height sebesar 2 dibandingkan dengan subtree kanan (berwarna merah) dan hal tersebut **tidak diperbolehkan** dalam AVL Tree.

Perlu diperhatikan bahwa balance atau tidaknya suatu tree tidak hanya dilihat dari root saja, melainkan seluruh node yang berada di tree tersebut. Berikut adalah contoh lain dari AVL Tree yang tidak balanced.



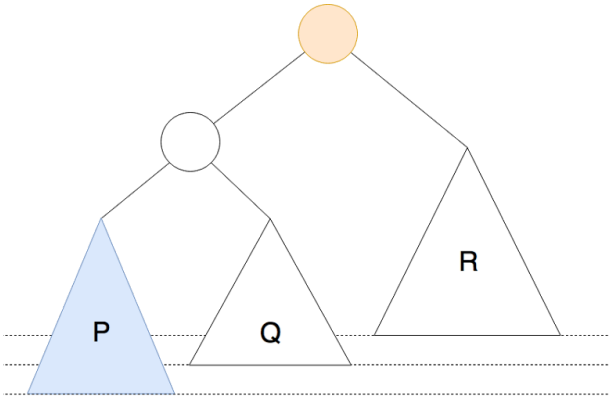
Kita dapat melihat bahwa subtree kiri dari root memiliki beda height sebesar 1 dibandingkan dengan subtree kanan dari root. Akan tetapi, subtree dari node yang berwarna orange memiliki beda height sebesar 2 dan hal tersebut **tidak diperbolehkan** di AVL Tree.

Untuk menjaga balance dari sebuah AVL Tree, dibutuhkan suatu operasi yang dinamakan rotation. Terdapat dua rotation, yaitu *single rotation* dan *double rotation*. Ketika melakukan

rotation, *rotation* yang dapat dilakukan adalah ke kiri (left) atau kanan (right). Pivot dari *rotation* adalah node yang memiliki subtree yang tidak balanced (pada ilustrasi-ilustrasi di bawah, node tersebut berwarna orange).

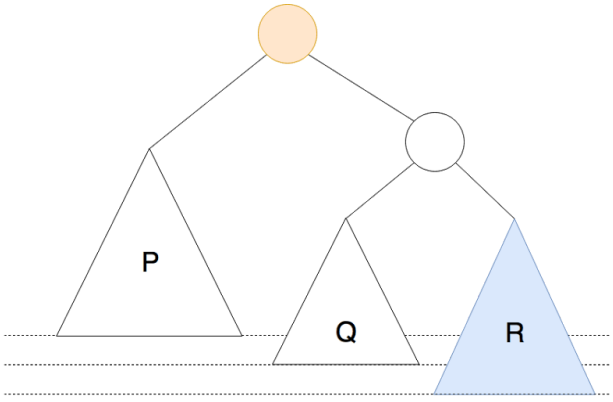
Terdapat 2 case untuk menentukan rotation manakah yang akan dipakai, yaitu *inside case* dan *outside case*.

Berikut adalah ilustrasi kasus ***outside case*** (kasus 1).



Pada ilustrasi di atas, kita dapat melihat bahwa subtree kiri dari root yang berwarna orange memiliki height lebih besar sebesar 2 dibandingkan dengan subtree kanan dari root (subtree R). Hal tersebut disebabkan oleh **subtree P** lebih besar sebesar 2 dibandingkan dengan subtree R.

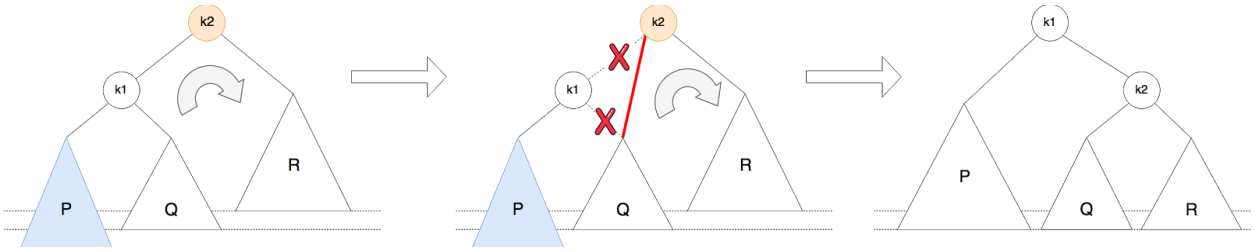
Berikut adalah ilustrasi kasus lain ***outside case*** (kasus 4).



Pada ilustrasi di atas, kita dapat melihat bahwa subtree kanan dari root yang berwarna orange memiliki height lebih besar sebesar 2 dibandingkan dengan subtree kiri dari root (subtree P).

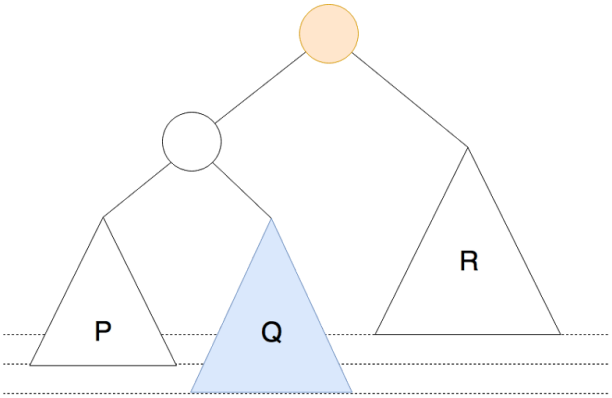
Hal tersebut disebabkan oleh **subtree R** lebih besar sebesar 2 dibandingkan dengan subtree P. Untuk ***outside case***, rotation yang digunakan adalah ***single rotation***.

Ilustrasi dari ***single rotation*** kasus 1 adalah sebagai berikut. Pada kasus 1, dilakukan ***single rotation right***.



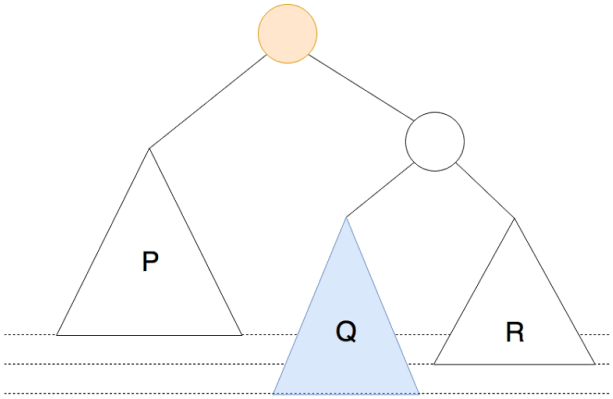
Keterangan: Subtree berwarna biru adalah penyebab tree tersebut tidak balanced.

Berikut adalah ilustrasi kasus ***inside case*** (kasus 2).



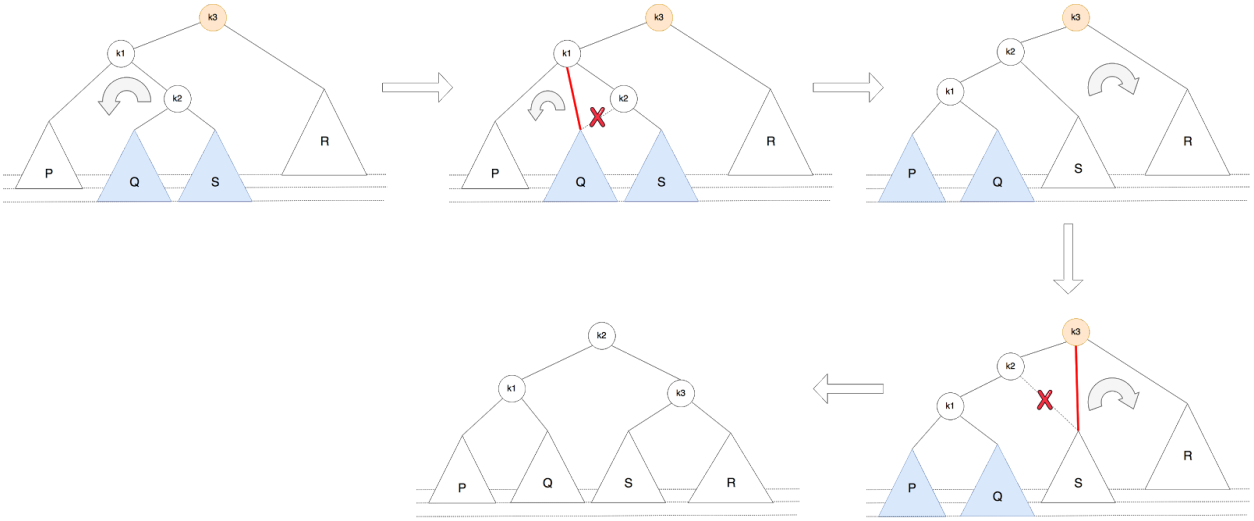
Pada ilustrasi di atas, kita dapat melihat bahwa subtree kiri dari root yang berwarna orange memiliki height lebih besar sebesar 2 dibandingkan dengan subtree kanan dari root (subtree R). Hal tersebut disebabkan oleh **subtree Q** lebih besar sebesar 2 dibandingkan dengan subtree R.

Berikut adalah ilustrasi kasus lain ***inside case*** (kasus 3).



Pada ilustrasi di atas, kita dapat melihat bahwa subtree kanan dari root yang berwarna orange memiliki height lebih besar sebesar 2 dibandingkan dengan subtree kiri dari root (subtree P). Hal tersebut disebabkan oleh **subtree Q** lebih besar sebesar 2 dibandingkan dengan subtree P. Untuk ***inside case***, rotation yang digunakan adalah ***double rotation***.

Ilustrasi dari ***double rotation*** kasus 2 adalah sebagai berikut. Pada kasus 2, dilakukan ***double rotation left right***.



Keterangan: Subtree berwarna biru adalah penyebab tree tersebut tidak balanced.

Dari keempat kasus di atas, masing-masing memiliki rotation yang berbeda. Untuk **kasus 1**, diperlukan ***single rotation right***. Bagaimana dengan ketiga kasus yang lain? Silahkan dianalisis dengan melihat contoh-contoh yang ada di slides.

1. **Operasi Add**

Operasi add adalah operasi untuk menambahkan suatu objek ke dalam tree. Objek dimasukkan ke dalam tree sedemikian hingga hingga menjaga struktur dari balanced BST.

Ide dari operasi add pada AVL Tree adalah sebagai berikut:

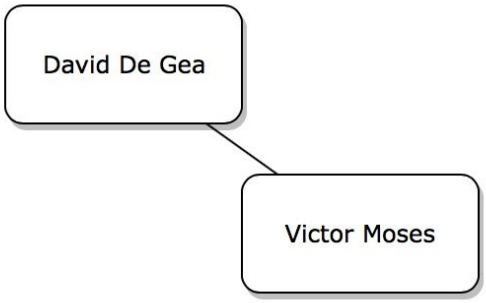
- Proses insertion adalah top-down, dimulai dari root hingga posisi yang diinginkan hingga menjadi sebuah leaf
- Proses untuk menjaga balance dari suatu AVL Tree dilakukan dengan cara bottom-up, dimulai dari leaf yang baru dimasukkan hingga root

Misalnya, terdapat sejumlah string sebagai berikut:

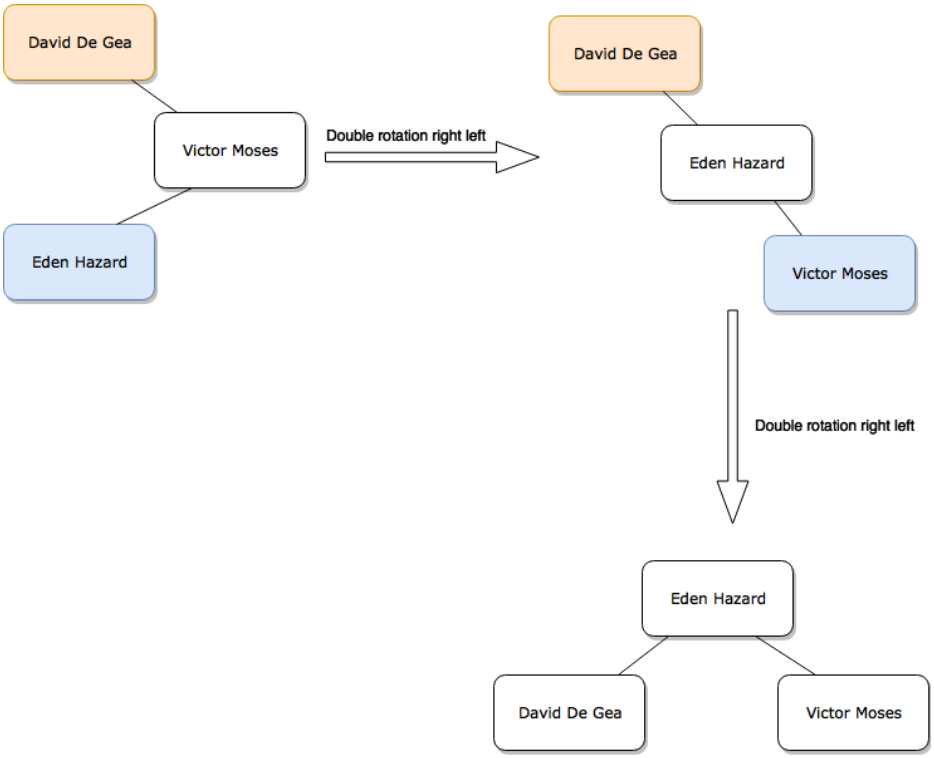
David de Gea
Victor Moses
Eden Hazard
Sergio Aguero
Fabio Borini
Nacer Chadli
Hector Bellerin
Mamadou Sakho
Nicklas Bendtner
Tony Adams

String-string tersebut akan dimasukkan ke dalam tree. Setelah memasukkan string “David de Gea”, kondisi tree akan menjadi sebagai berikut.

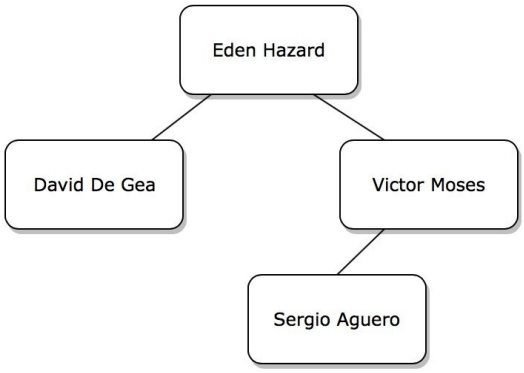
Setelah memasukkan “Victor Moses”:



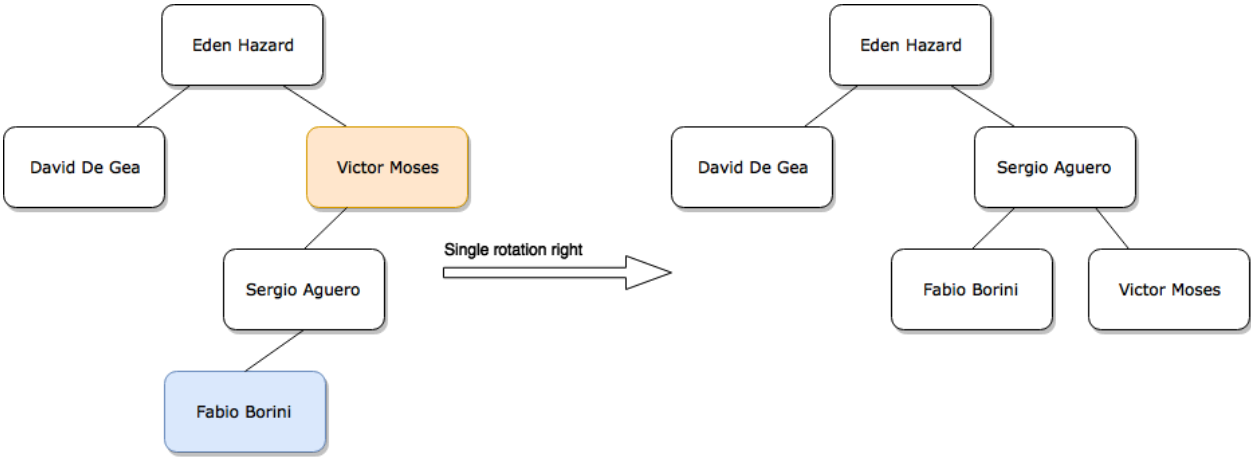
Ketika memasukkan “Eden Hazard”, diperlukan *double rotation right left*. (Mengapa *double rotation right left*? Kasus berapakah ini? Silahkan dianalisis)



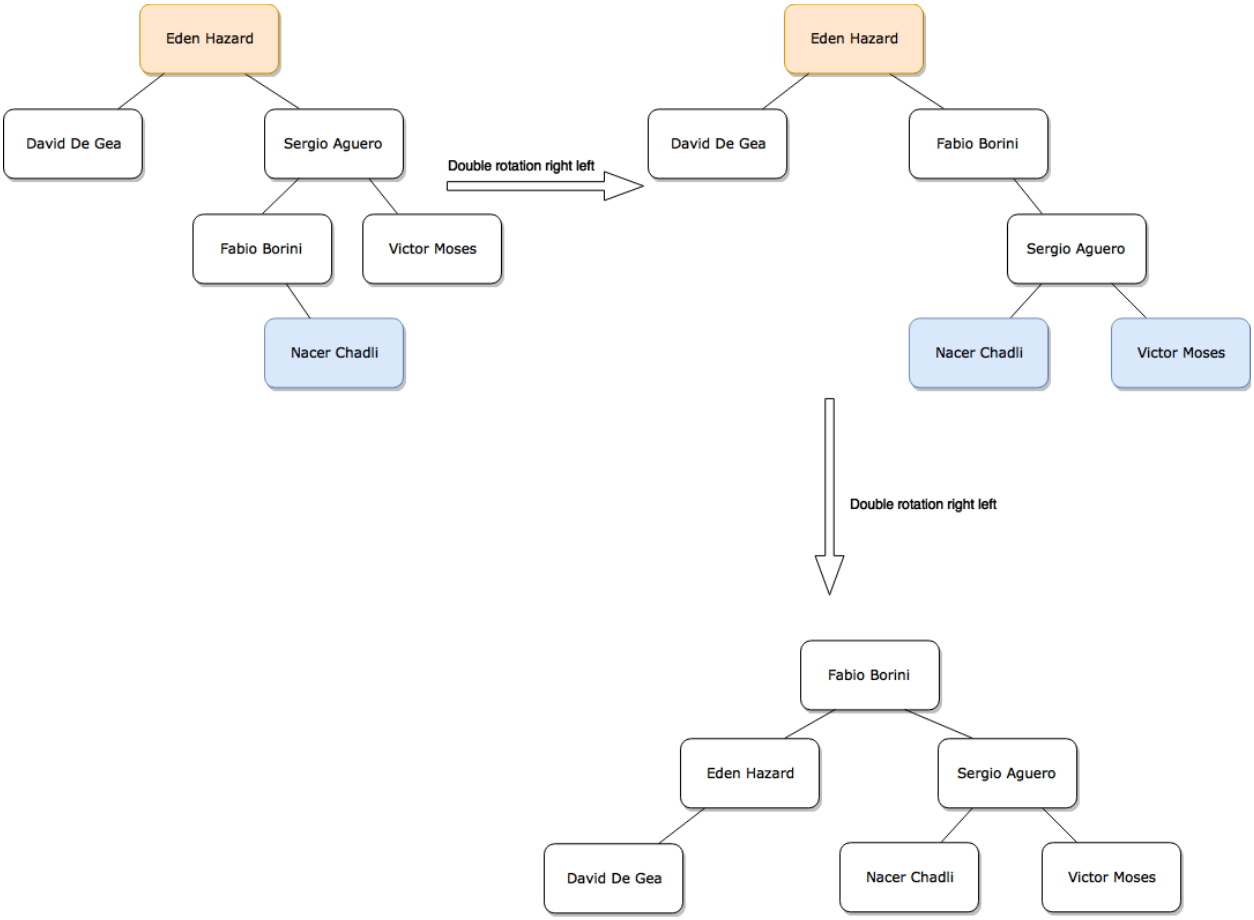
Setelah memasukkan “Sergio Aguero”:



Ketika memasukkan “Fabio Borini”, diperlukan *single rotation right*. (Mengapa *single rotation right*? Kasus berapakah ini? Silahkan dianalisis)

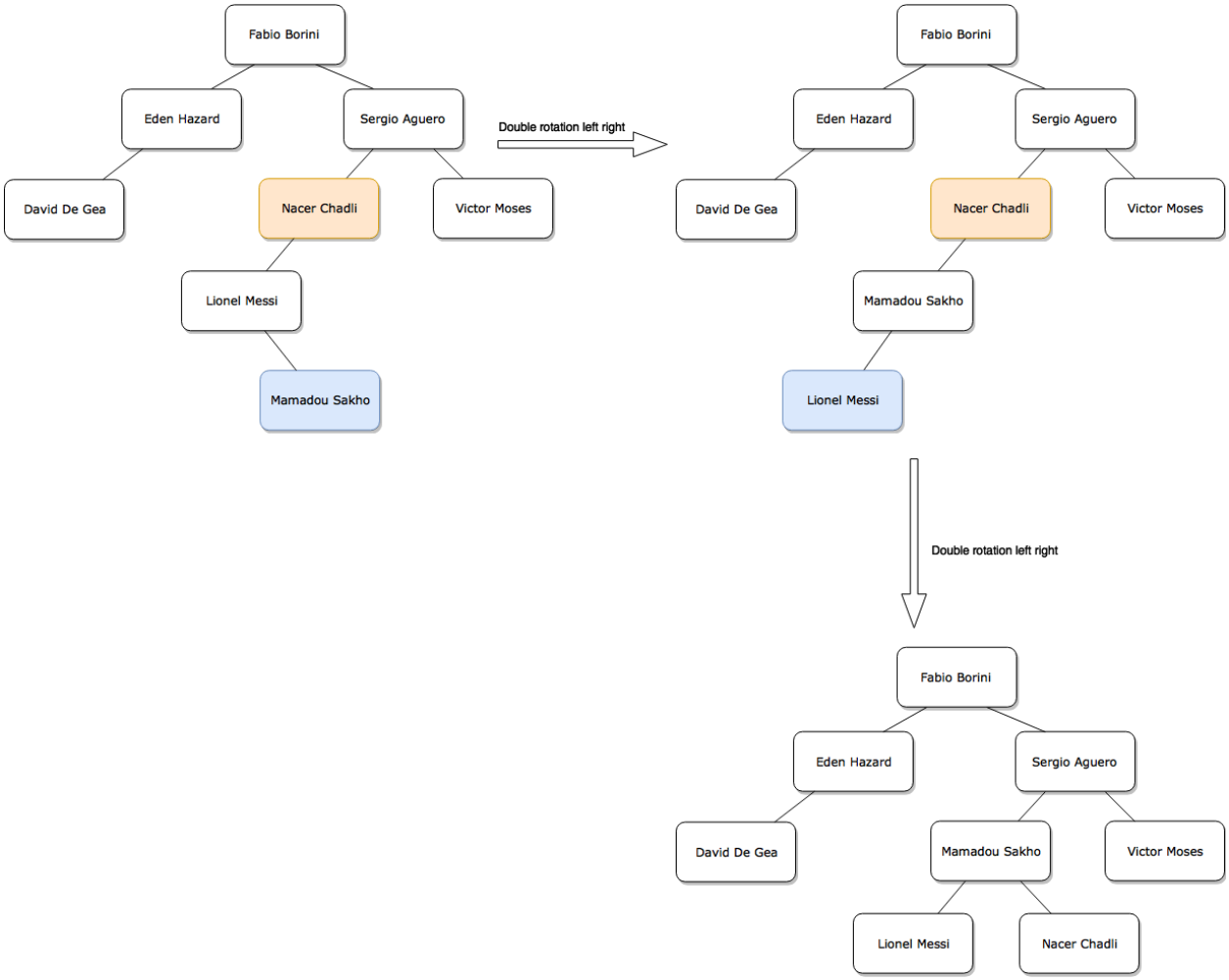


Ketika memasukkan “Nacer Chadli”, diperlukan *double rotation right left*. (Mengapa *double rotation right left*? Kasus berapakah ini? Silahkan dianalisis)

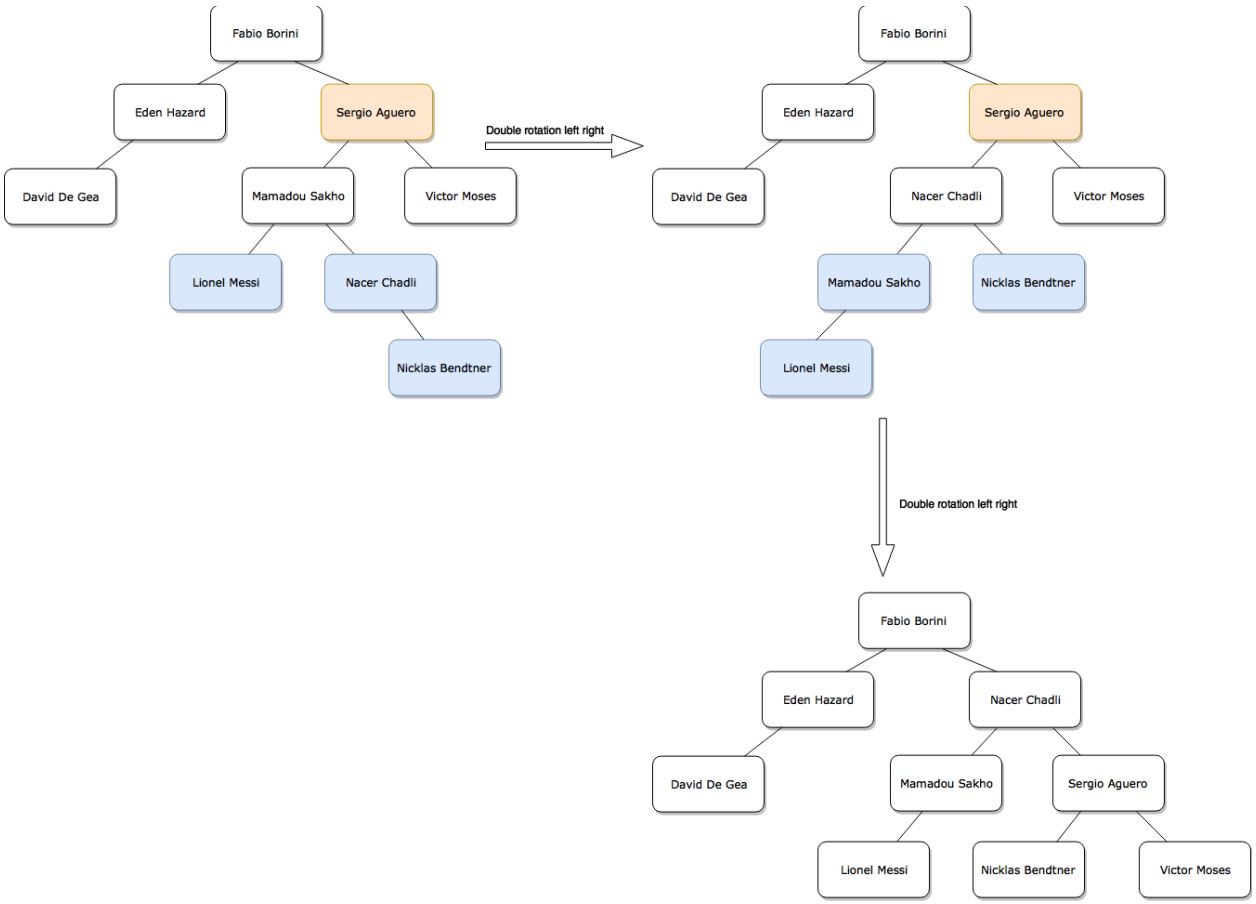


Setelah memasukkan “Lionel Messi”:

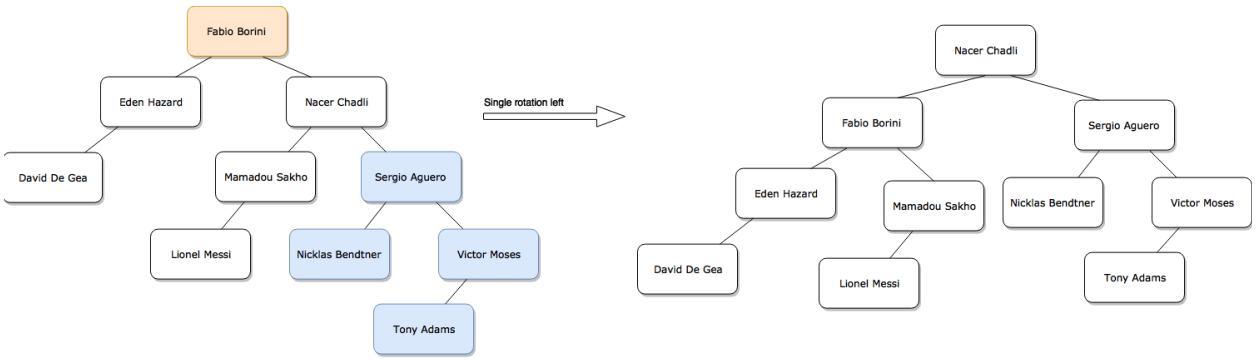
Ketika memasukkan “Mamadou Sakho”, diperlukan *double rotation left right*. (Mengapa *double rotation left right*? Kasus berapakah ini? Silahkan dianalisis)



Ketika memasukkan “Nicklas Bendtner”, diperlukan *double rotation left right*. (Mengapa *double rotation left right*? Kasus berapakah ini? Silahkan dianalisis)



Ketika memasukkan “Tony Adams”, diperlukan *single rotation left*. (Mengapa *single rotation left*? Kasus berapakah ini? Silahkan dianalisis)



2. **Operasi Delete**

Menghapus node pada AVL Tree sama dengan menghapus node pada Binary Search Tree. Perbedaannya terletak pada penanganan tidak balanced. Penanganan tidak

balanced sama dengan operasi add AVL Tree yaitu dengan melakukan bottom-up proses dimulai dari node yang menggantikan node yang akan dihapus hingga root.

Terdapat beberapa kasus ketika delete. Berikut adalah kasusnya:

Kasus 1: jika X adalah leaf, delete X.

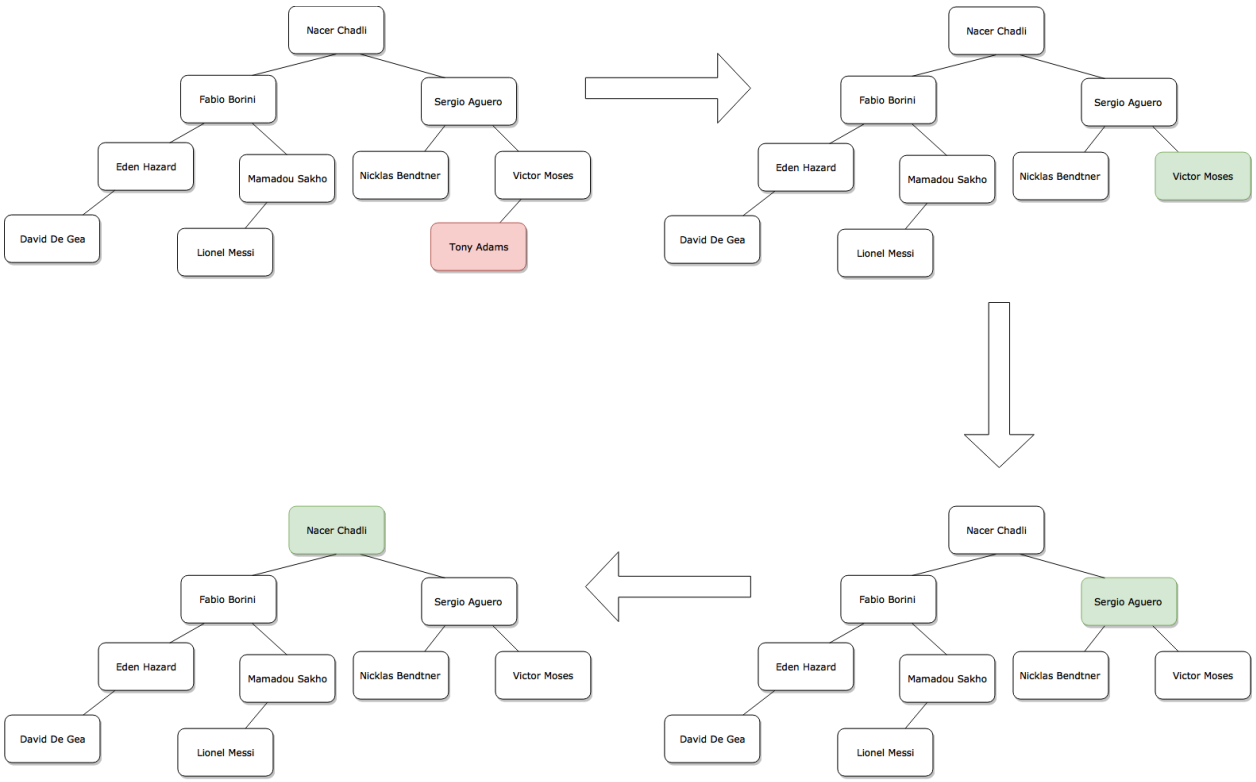
Kasus 2: jika X punya 1 child, X digantikan oleh child tersebut.

Kasus 3: jika X punya 2 child, ganti X secara rekursif dengan predecessor inorder-nya.

Keterangan: X adalah node yang akan dihapus

Setelah delete node berdasarkan 3 kasus di atas, kita harus melakukan balancing.

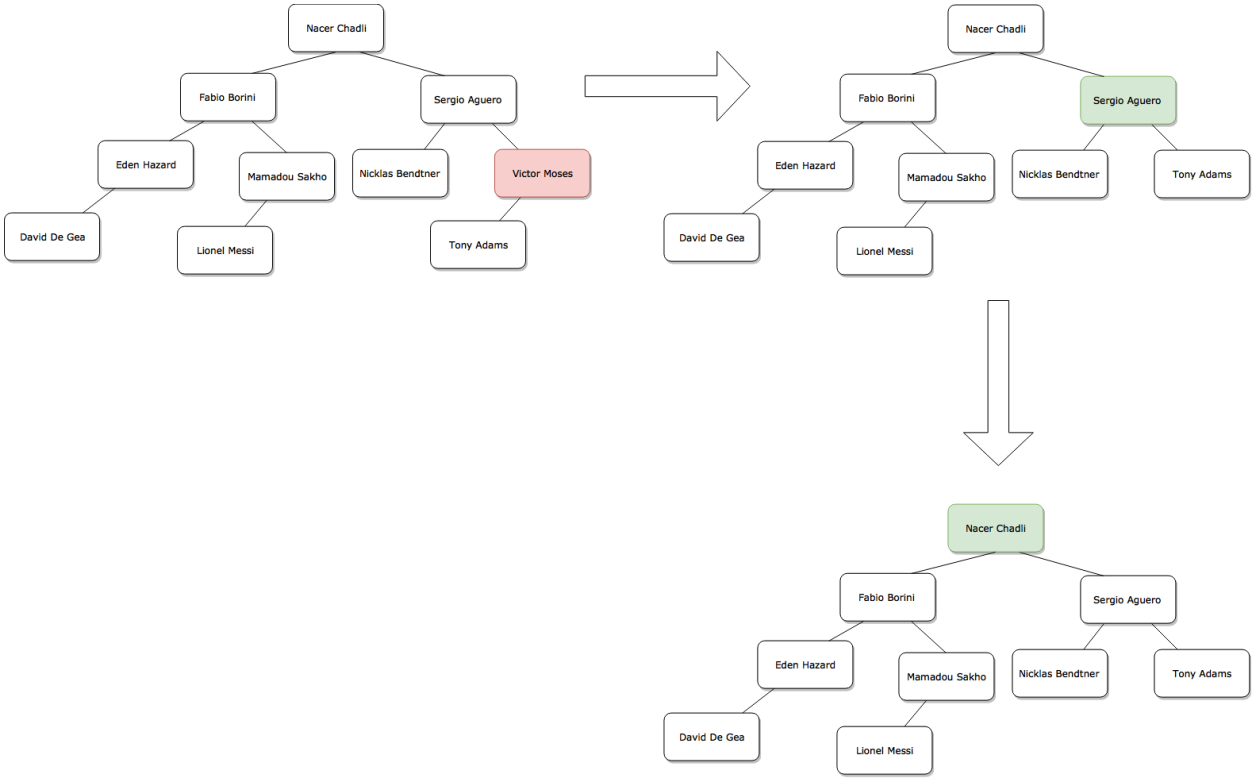
Berikut adalah proses delete dan balancing untuk kasus 1.



Pada awalnya, kita ingin menghapus node dengan nama “Tony Adams”. Dikarenakan node tersebut adalah leaf, maka tidak diperlukan penggantian node. Kemudian, dilakukan balancing dimulai dari “Victor Moses” sebagai parent dari “Tony Adams”. Dikarenakan subtree dari node “Victor Moses” balanced, maka dilanjutkan ke parentnya yaitu node “Sergio Aguero”. Dikarenakan subtree dari node “Sergio Aguero” balanced, maka dilanjutkan ke parentnya yaitu

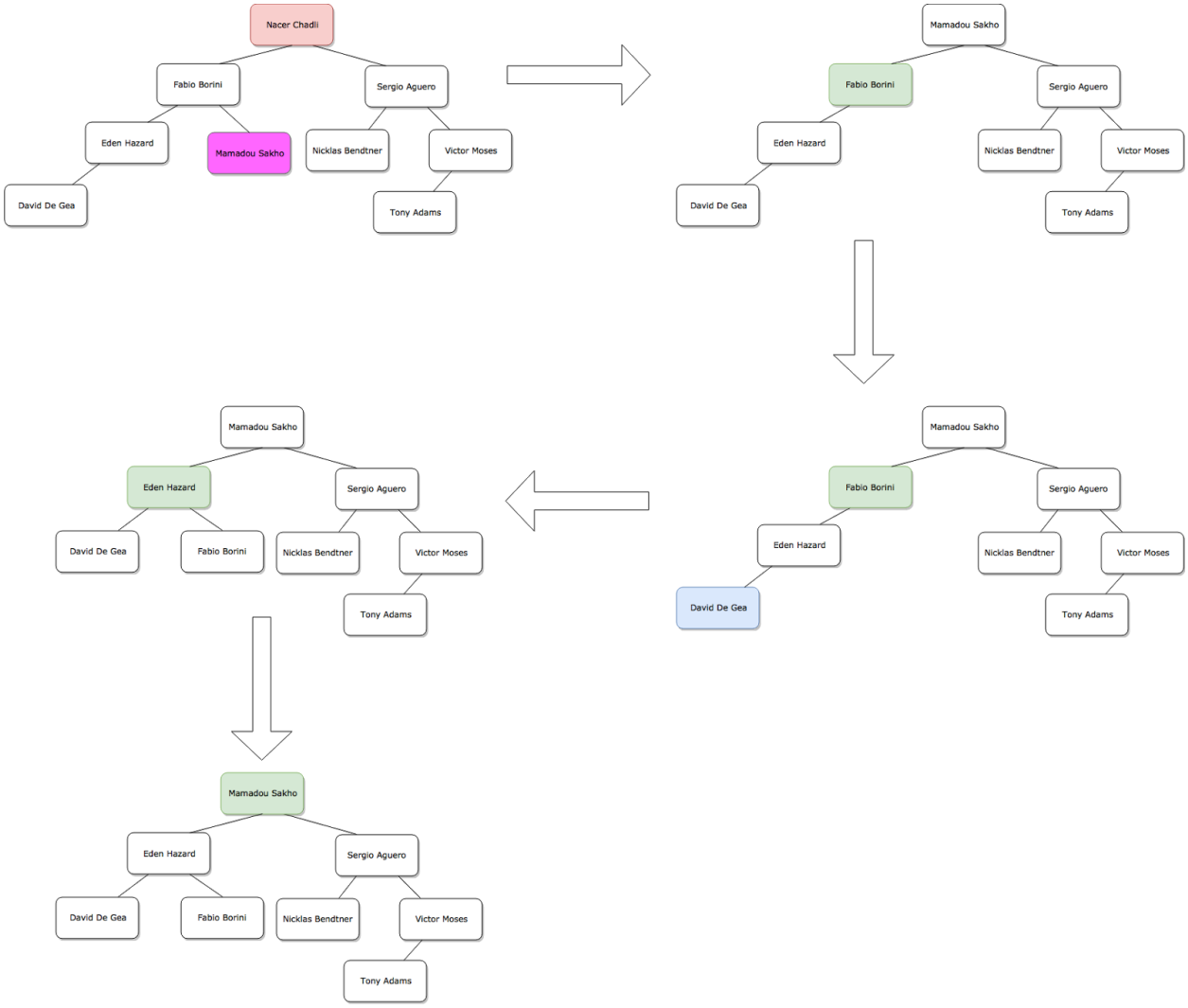
node “Nacer Chadli”. Dikarenakan subtree dari node “Nacer Chadli” balanced dan node “Nacer Chadli” adalah root dari tree, maka balancing dihentikan.

Berikut adalah proses delete dan balancing untuk kasus 2.



Pada awalnya, kita ingin menghapus node dengan nama “Victor Moses”. Dikarenakan node tersebut memiliki satu child, maka node “Victor Moses” digantikan oleh node “Tony Adams”. Kemudian, dilakukan balancing dimulai dari node “Sergio Aguero” sebagai parent dari node “Victor Moses” yang dihapus. Dikarenakan subtree dari node “Sergio Aguero” balanced, maka dilanjutkan ke parentnya yaitu node “Nacer Chadli”. Dikarenakan subtree dari node “Nacer Chadli” balanced dan node “Nacer Chadli” adalah root dari tree, maka balancing dihentikan.

Berikut adalah proses delete dan balancing untuk kasus 3.



Note: Tree di atas hanyalah sebuah contoh. Node “Lionel Messi” dihilangkan hanya untuk mempermudah penjelasan.

Pada awalnya, kita ingin menghapus node dengan nama “Nacer Chadli”. Dikarenakan node tersebut memiliki dua child, maka node “Nacer Chadli” digantikan oleh node “Mamadou Sakho” sebagai predecessor inorder dari node “Nacer Chadli”. Kemudian, dilakukan balancing dimulai dari node “Fabio Borini” sebagai parent dari node “Mamadou Sakho” ketika sebelum menggantikan node “Nacer Chadli”. Dikarenakan subtree dari node “Fabio Borini” tidak balanced, maka diperlukan adanya rotation. Setelah itu, dilanjutkan ke node “Mamadou Sakho” sebagai parent dari node “Fabio Borini” sebelum dilakukannya balancing. Dikarenakan subtree dari node “Mamadou Sakho” balanced dan node “Mamadou Sakho” adalah root dari tree, maka balancing dihentikan.

B. Soal

Agus adalah seorang manajer JaringToon yang merupakan website penyedia komik-komik daring. Saat ini JaringToon sering menghadapi komplain dari banyak pembacanya karena masalah lambatnya membuka komik-komiknya yang ada di JaringToon, sehingga Agus merasa perlu merombak struktur sistem penyimpanan setiap episode komik-komik dalam JaringToon agar lebih efisien. Agus juga menginginkan fitur dimana JaringToon dapat menampilkan komik paling sering dibaca dan juga paling jarang dibaca (kasihan..) serta bisa menampilkan urutan komik berdasarkan jumlah dilihatnya komik tersebut.

Agus baru saja mendengar (entah dari mana) bahwa struktur data AVL Tree dapat memenuhi kebutuhan dia untuk penyimpanan komik-komik di websitenya secara efisien. Sebagai programmer dari JaringToon, Agus meminta anda untuk membuat *file system* komiknya menggunakan AVL Tree.

Input dan Output

Program akan menerima input untuk *command* seperti di bawah ini hingga *End of File*:

1. **ADD <Nama komik> <Jumlah dilihat>**
   - Mengunggah komik ke dalam JaringToon
   - **<Nama komik>** berupa string yang terdiri dari **1 kata**, dan dipastikan **unik**.
   - **<Jumlah dilihat>** adalah integer dan dipastikan **unik** (kok bisa ya?).
   - Jika terjadi rotasi saat memasukkan komik dalam tree, maka cetak:

     `Lakukan rotasi <sekali/dua kali> pada <Nama node yang dirotasi>`
   - Setelah selesai memasukkan komik dalam sistem, maka cetak:

     `Komik <Nama komik> sudah disimpan dalam JaringToon`

2. **REMOVE <Nama komik> <Jumlah dilihat>**
   - Membuang komik dari JaringToon
   - **<Nama komik>** yang dicari dipastikan **sudah ada** di dalam Tree **jika tree tidak kosong**.
   - **<Jumlah dilihat>** yang dicari dipastikan **sesuai ketika input jika nama komik yang dicari ada pada tree.**

- Jika terjadi rotasi saat memasukkan komik dalam tree, maka setiap melakukan rotasi cetak:

  `Lakukan rotasi <sekali/dua kali> pada <Nama node yang dirotasi>`

- Setelah selesai menghapus komik dalam sistem, maka cetak:

  `Komik <Nama komik> sudah dihapus dari JaringToon`

- Jika tidak terdapat komik di dalam tree, maka cetak:

  `Tidak ada komik dalam JaringToon`

3. POPULARITY

- Menampilkan dua komik yang memiliki popularitas **tertinggi dan terendah** berdasarkan jumlah dibaca.
- Jika tidak ada komik di dalam tree, maka cetak:

  `Tidak ada komik dalam JaringToon`

- Jika hanya terdapat 1 komik di dalam sistem, maka cetak:

  `Hanya ada komik <Nama komik>`

- Jika terdapat setidaknya 2 komik di dalam sistem, maka cetak:

  `Tertinggi <Nama komik 1>; Terendah <Nama komik 2>`

4. PRINT

- Jika terdapat data di dalam tree, keluarkan 3 baris string:

  `In Order: <Nama1>; <Nama2>; ... ; <NamaN>`

  `Pre Order: <Nama1>; <Nama2>; ... ; <NamaN>`

  `Post Order: <Nama1>; <Nama2>; ... ; <NamaN>`

- Jika tidak ada komik di dalam tree, maka cetak:

  `Tidak ada komik dalam JaringToon`

Contoh Input

```
ADD Dadu 20
POPULARITY
ADD GodOfPacil 50
ADD 7Wonders 70
ADD Qimpulnoid 100
PRINT
REMOVE Qimpulnoid 100
POPULARITY
REMOVE Dadu 20
```

```
REMOVE 7Wonders 70
PRINT
REMOVE GodOfPacil 50
PRINT
```

Contoh Output

```
Komik Dadu sudah disimpan dalam JaringToon
Hanya ada komik Dadu
Komik GodOfPacil sudah disimpan dalam JaringToon
Lakukan rotasi sekali pada GodOfPacil
Komik 7Wonders sudah disimpan dalam JaringToon
Komik Qimpulnoid sudah disimpan dalam JaringToon
In Order: Dadu; GodOfPacil; 7Wonders; Qimpulnoid
Pre Order: GodOfPacil; Dadu; 7Wonders; Qimpulnoid
Post Order: Dadu; Qimpulnoid; 7Wonders; GodOfPacil
Komik Qimpulnoid sudah dihapus dari JaringToon
Tertinggi 7Wonders; Terendah Dadu
Komik Dadu sudah dihapus dari JaringToon
Komik 7Wonders sudah dihapus dari JaringToon
In Order: GodOfPacil
Pre Order: GodOfPacil
Post Order: GodOfPacil
Komik GodOfPacil sudah dihapus dari JaringToon
Tidak ada komik dalam JaringToon
```

Catatan

- **Anda harus mengimplementasikan AVL Tree sendiri**
- **Program yang tidak mengimplementasi AVL Tree tidak akan dinilai**
- **Penggunaan Arrays.sort dan Collections.sort pada soal ini tidak diperbolehkan**
- **Anda tidak diperbolehkan menggunakan kelas TreeSet, TreeMap, dan PriorityQueue bawaan Java**
- **Sudah disediakan template untuk soal ini**

Keterangan Test Case

- Test case 1: contoh input
- Test case 2 - 5: add dan delete yang hanya menggunakan single rotation
- Test case 6 - 8: add dan delete yang hanya menggunakan double rotation

- Test case 9 - 10: gabungan single rotation dan double rotation