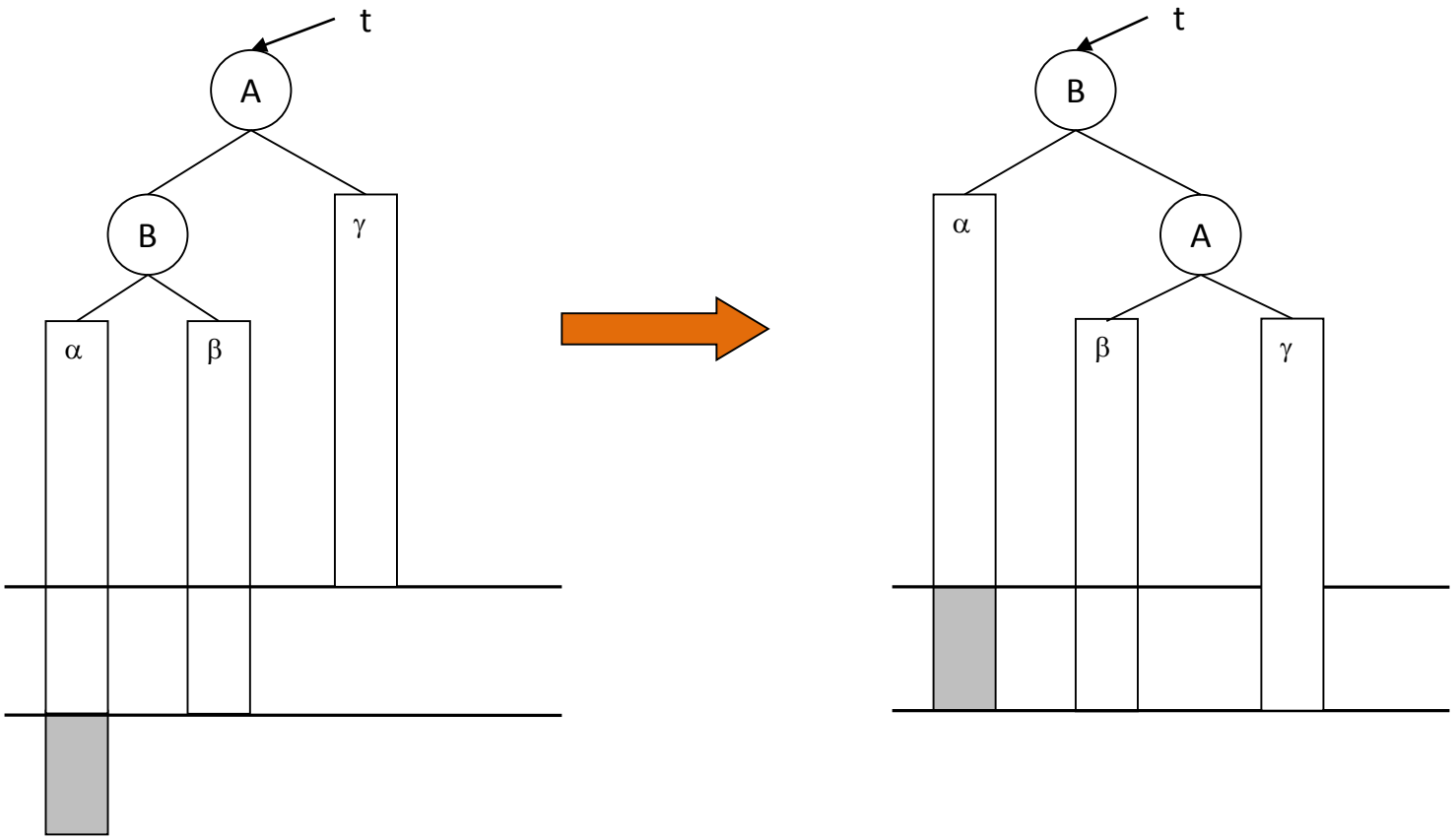
Binarna stabla

SPA2

Napisati program koji formira balansirano uređeno binarno stablo celih brojeva.

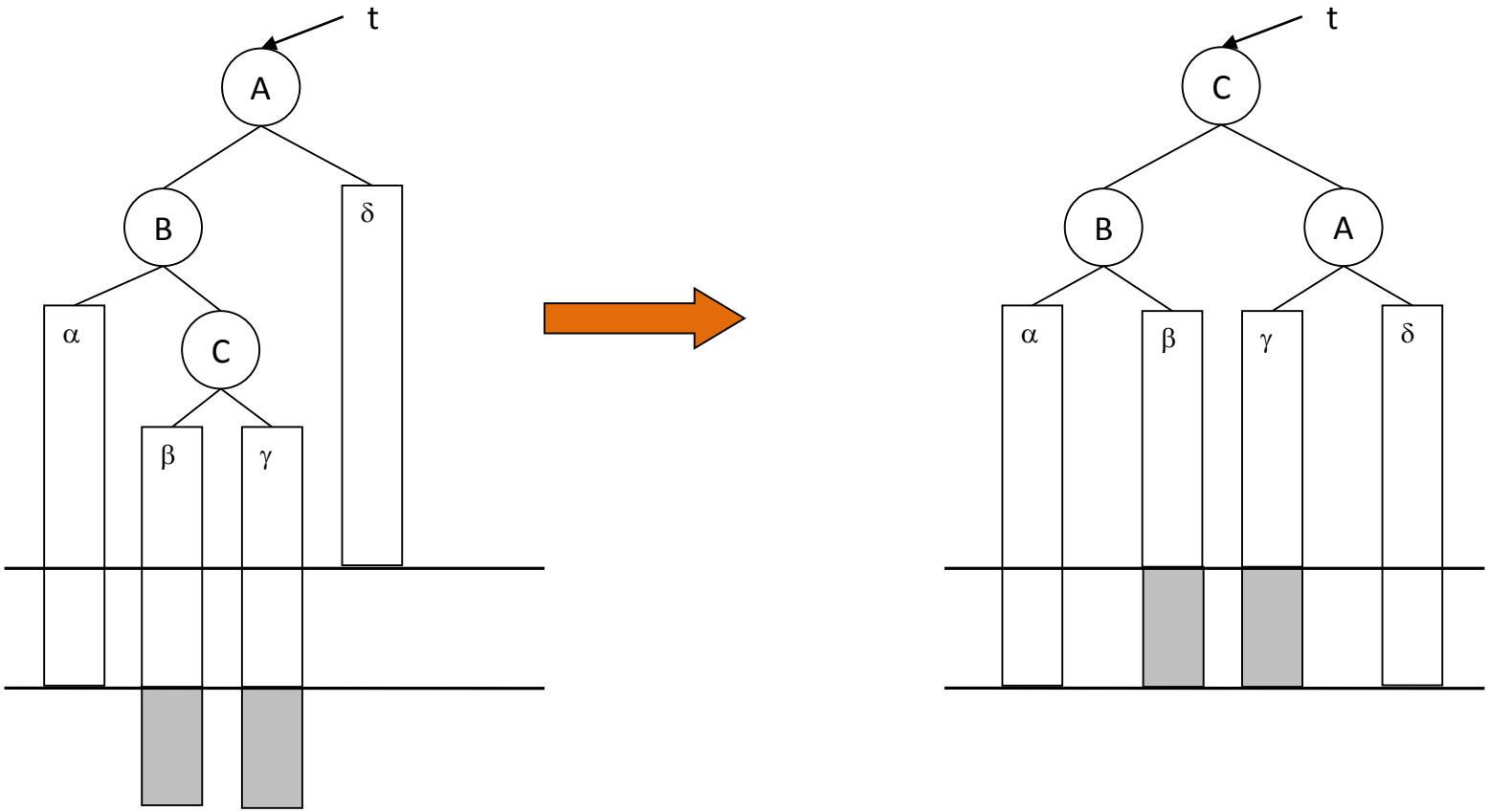
AVL-stabla (G.M. Adelson-Velsky and E.M. Landis)

(t->balans < -1) && (t->levi->balans < 0)



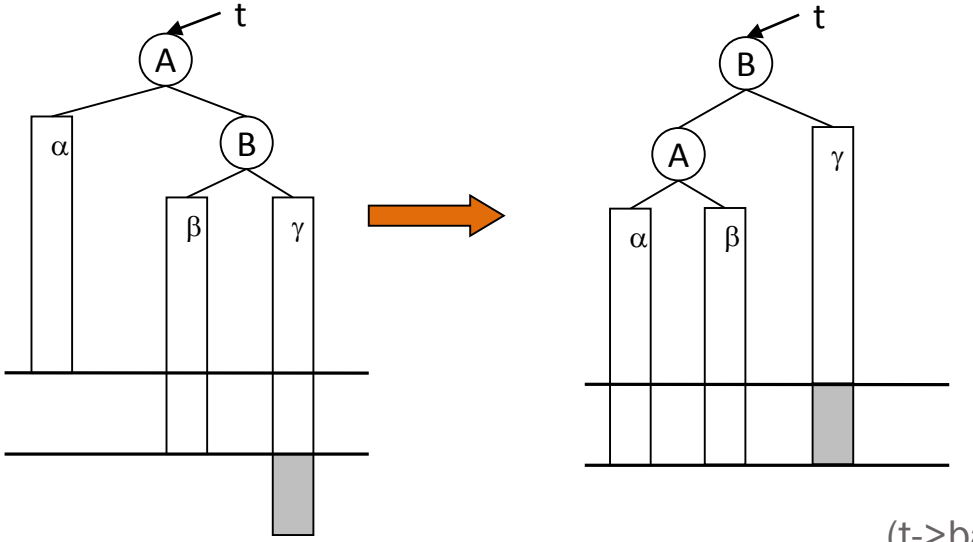
Desna rotacija

(t->balans < -1) && !(t->levi->balans < 0)



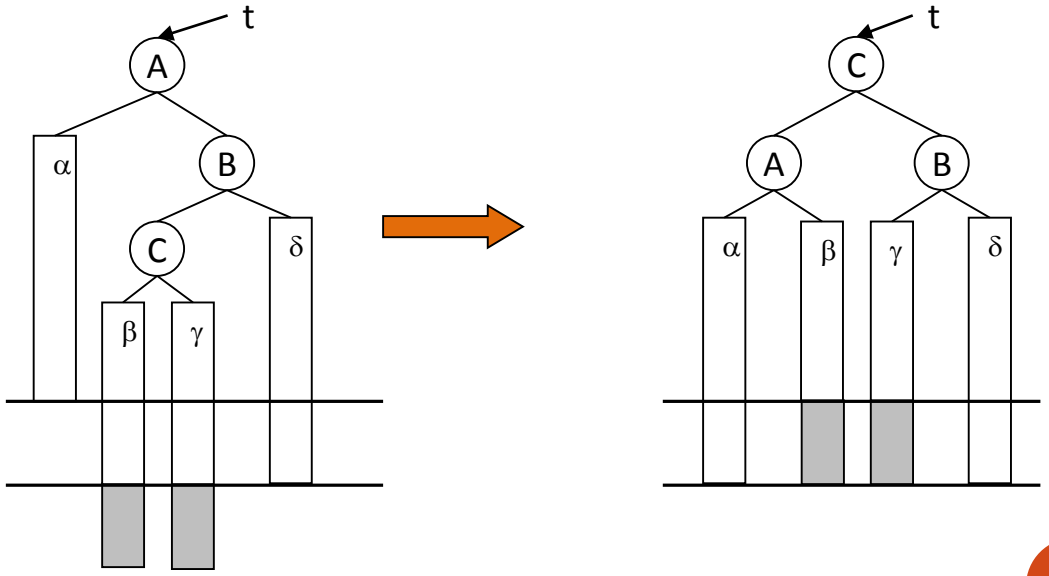
Dvostruka desna rotacija

(t->balans > 1) && (t->desni->balans > 0)



Leva rotacija

(t->balans > 1) && !(t->desni->balans > 0)



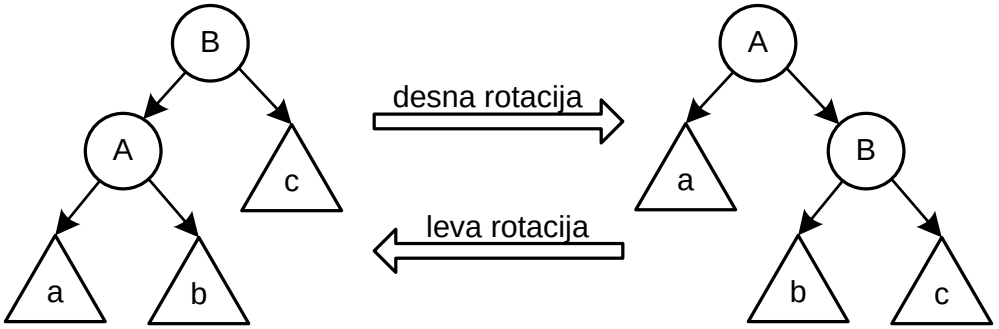
Dvostruka leva rotacija

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct drvo{
   int broj;
   int balans;
   struct drvo *levi,*desni;
};
#define novi(x) x=(struct drvo *) malloc(sizeof(struct drvo))

int dodaj(struct drvo **,int);
struct drvo* form();
void ispis(struct drvo*);
void lrotacija(struct drvo**);
void drotacija(struct drvo**);
int dubina(struct drvo*);
int balansirano(struct drvo*);
```

```
main(){
    struct drvo *p,*q;
    p=form();
    ispis(p); printf("\n");
    printf("Dubina je %d\n",dubina(p));
    if (balansirano(p)) printf("jeste balansirano\n");
    else printf("nije balansirano\n");
}
struct drvo* form(){
  struct drvo *koren;
  int k;
  koren=NULL;
  scanf("%d",&k);
  while(k) {
       dodaj(&koren,k);
       scanf("%d",&k);
  }
  return koren;
}
```

```
void lrotacija(struct drvo **t){
  struct drvo *poml,*pomd;
  int stari_balans;
  poml = *t;
  pomd = poml->desni;
  poml->desni = pomd->levi;
  pomd->levi = poml;
  *t=pomd;
  poml->balans=dubina(poml->desni)-dubina(poml->levi);
  pomd->balans=dubina(pomd->desni)-dubina(pomd->levi);}
```



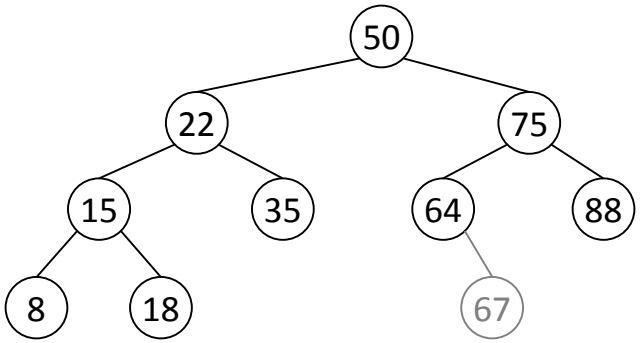

```
void drotacija(struct drvo **t){
  struct drvo *poml,*pomd;
  int stari_balans;
  pomd = *t;
  poml = pomd->levi;
  pomd->levi = poml->desni;
  poml->desni = pomd;
  *t=poml;
  poml->balans=dubina(poml->desni)-dubina(poml->levi);
  pomd->balans=dubina(pomd->desni)-dubina(pomd->levi);}
```

```
int dodaj(struct drvo **p, int vrednost){
  int inkrement, rezultat;
  struct drvo *t = *p;
  rezultat = 0;
  if(t==NULL){
       novi(t);
       if(t==NULL){
              printf("Greska pri alociranju memorije\n");
              exit(0);
       }
       t->broj = vrednost;
       t->levi = t->desni = NULL;
       t->balans = 0;
       rezultat = 1;
  }
...
```

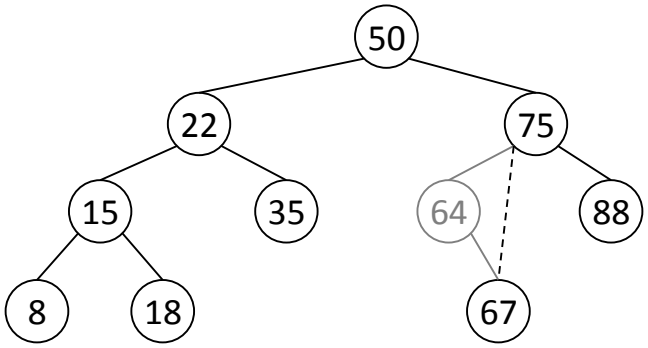
```
...
  else{
       if(vrednost > t->broj) inkrement=dodaj(&(t->desni),vrednost);
       else inkrement=-dodaj(&(t->levi),vrednost);
       t->balans += inkrement;
       if(inkrement!=0 && t->balans!=0){
              if(t->balans<-1 ){
                     if(t->levi->balans<0) drotacija(&t);
                     else{  lrotacija(&(t->levi)); drotacija(&t); }
              }
              else if(t->balans>1){
                     if(t->desni->balans>0 ) lrotacija(&t);
                     else{  drotacija(&(t->desni)); lrotacija(&t); }
              }
              else rezultat = 1;
       }
  }
  *p = t;
  return rezultat;
}
```

Brisanje čvora iz stabla

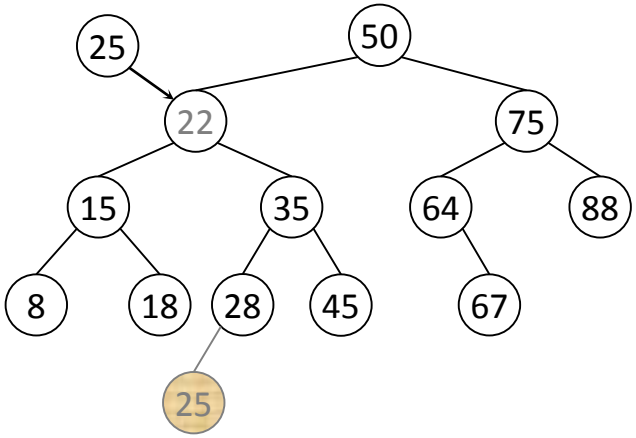
Čvor bez naslednika:

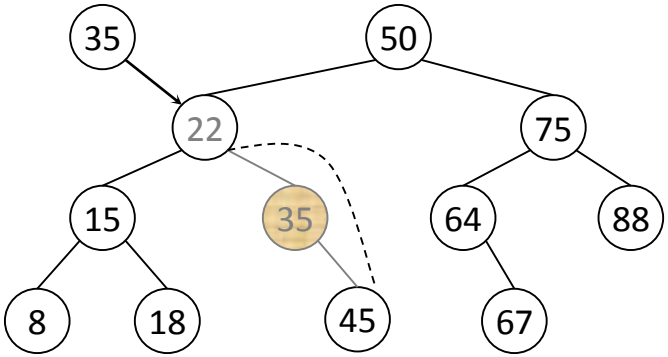


Čvor sa jednim naslednikom:

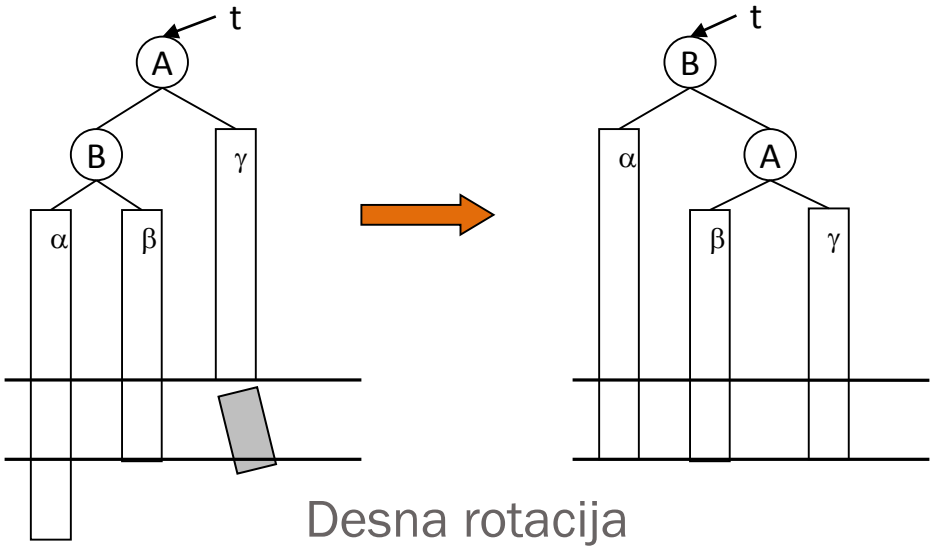


Čvor sa dva naslednika:



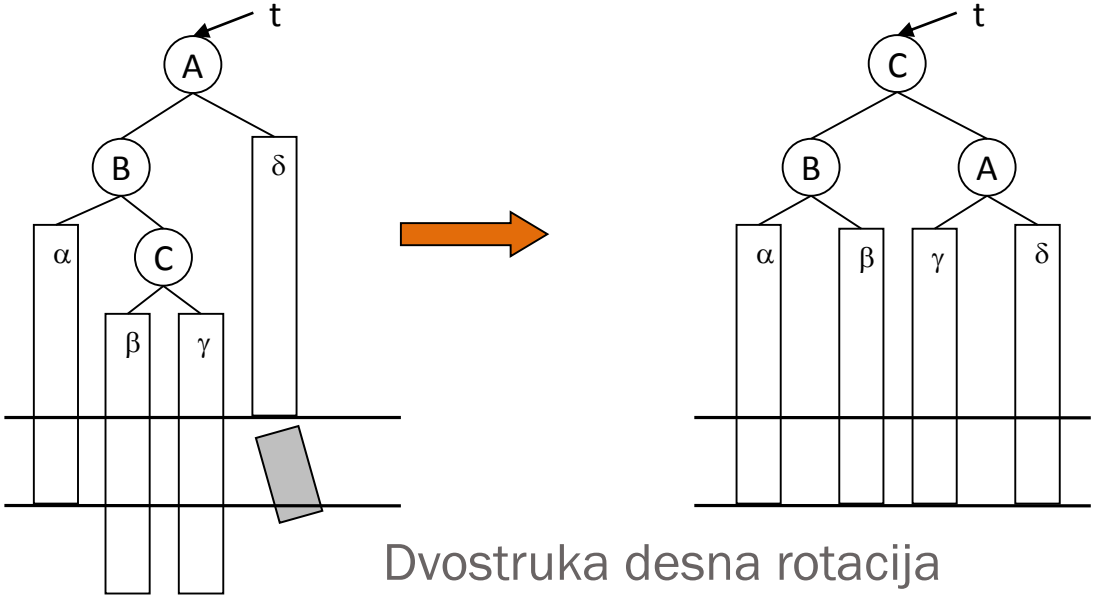


(t->balans < -1) && (t->levi->balans < 0)



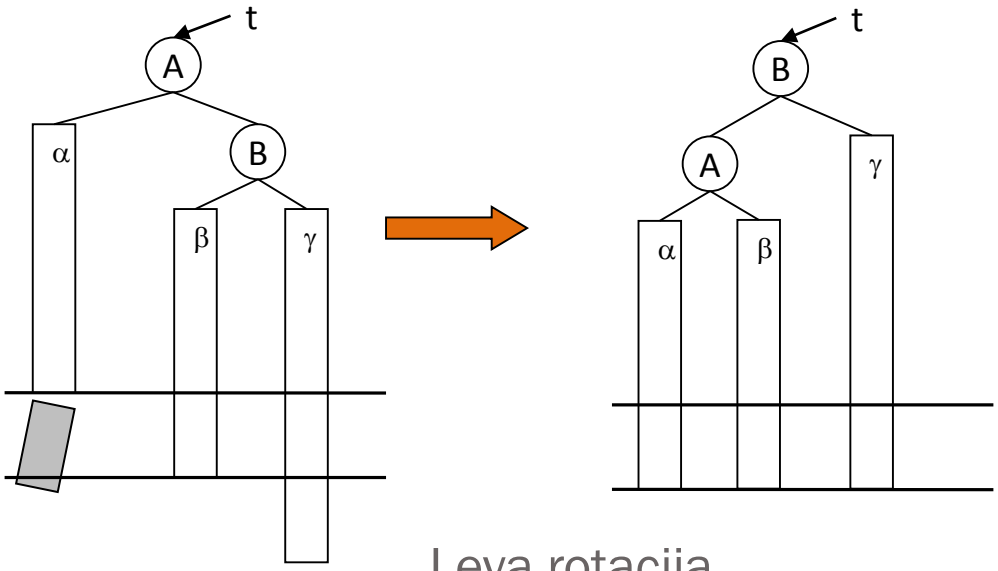
Desna rotacija

(t->balans < -1) && !(t->levi->balans < 0)



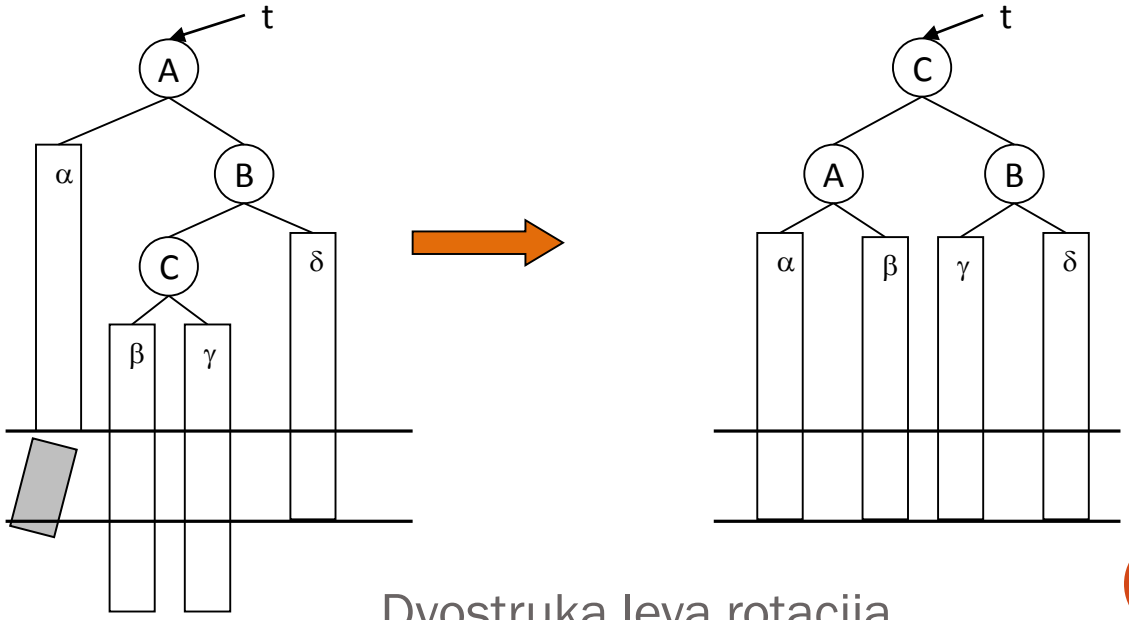
Dvostruka desna rotacija

(t->balans > 1) && (t->desni->balans > 0)



Leva rotacija

(t->balans > 1) && !(t->desni->balans > 0)



Dvostruka leva rotacija

Napisati program koji iz formiranog uređenog balansiranog binarnog stabla briše čvor čija je vrednost jednaka unetom broju **K**.

```
int nadji(struct drvo **p){
   struct drvo *pom,*pom1;
   int rez;
   pom=*p;
   if (!pom->levi) {
        rez=pom->broj;
        pom=pom->desni;
   }
   else rez=nadji(&(pom->levi));
   if (pom) {
        pom->balans=dubina(pom->desni)-dubina(pom->levi);
        sredi(&pom);
   }
   *p=pom;
   return rez;
}
```

```
void sredi(struct drvo **p){
   struct drvo *t;
   if(*p){
        t=*p;
        if( t->balans<-1 ){
               if( t->levi->balans<0 ) drotacija(&t);
               else{
                      lrotacija(&(t->levi)); drotacija(&t);
               }
        }
        else if( t->balans>1 ){
               if( t->desni->balans>0 ) lrotacija(&t);
               else{
                      drotacija(&(t->desni)); lrotacija(&t);
               }
        }
   *p=t;
   }
}
```

```
struct drvo* obrisi_b(struct drvo *p,int k){
  struct drvo *pom;
  if(!p) return NULL;
  if (p->broj==k) {
       if((!p->levi) && (!p->desni)) {
              free(p);
              return NULL;
       }
       if((!p->levi) && (p->desni)) {
              pom=p->desni;
              free(p);
              return pom;
       }
       if((p->levi) && (!p->desni)){
              pom=p->levi;
              free(p);
              return pom;
       }
...
```

```
...
        if((p->levi) && (p->desni)){
                p->broj=nadji(&(p->desni));
                p->balans=dubina(p->desni)-dubina(p->levi);
                sredi(&p);
                return p;
        }
  }
  if(k<p->broj){
        p->levi=obrisi_b(p->levi,k);
        p->balans=dubina(p->desni)-dubina(p->levi);
        sredi(&p);
        return p;
  }
  else{
        p->desni=obrisi_b(p->desni,k);
        p->balans=dubina(p->desni)-dubina(p->levi);
        sredi(&p);
        return p;
  }
}
```