Sudėtingesnės dvimačių medžių struktūros

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Spalio 20 d., 2023

AVL paieškos medis

Pagrindinis dvejetainio medžio trūkumas yra tas, kad blogiausiu atveju jis gali virsti tiesiniu sąrašu. Tada duomenų įterpimo, šalinimo ir paieškos veiksmai tampa labai neefektyviais, jų sudėtingumas yra proporcingas medžio viršūnių skaičiui.

AVL paieškos medis

Pagrindinis dvejetainio medžio trūkumas yra tas, kad blogiausiu atveju jis gali virsti tiesiniu sąrašu. Tada duomenų įterpimo, šalinimo ir paieškos veiksmai tampa labai neefektyviais, jų sudėtingumas yra proporcingas medžio viršūnių skaičiui.

Nagrinėsime sudėtingesnes dvimačių paieškos medžių struktūras, kai visų pagrindinių operacijų sudėtingumas net ir blogiausiu atveju yra $O(\log n)$ veiksmų (medyje saugome n elementų).

Apibrėžimas. AVL paieškos medis, tai iš dalies subalansuotas dvejetainis paieškos medis, kurio bet kurios viršūnės kairiojo ir dešiniojo pomedžių aukščiai gali skirtis ne daugiau nei vienetu.

Apibrėžimas. AVL paieškos medis, tai iš dalies subalansuotas dvejetainis paieškos medis, kurio bet kurios viršūnės kairiojo ir dešiniojo pomedžių aukščiai gali skirtis ne daugiau nei vienetu.

AVL medžių pavyzdžiai yra pateikti paveiksle, prie kiekvienos viršūnės nurodytas jos subalansuotumo faktorius. Šis faktorius gali būti lygus $-1, 0$ arba 1.

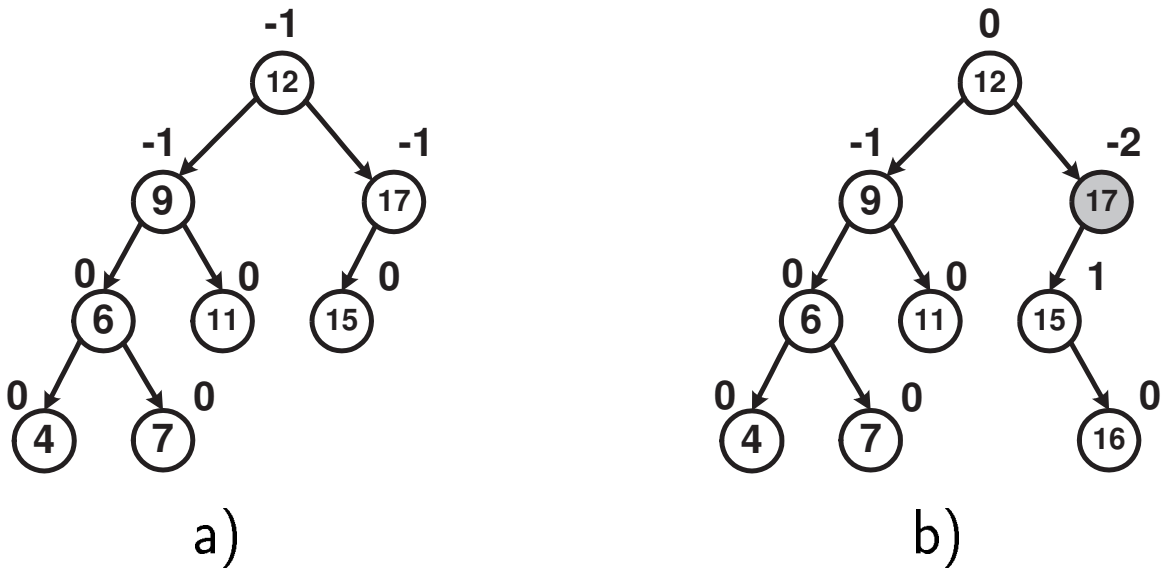
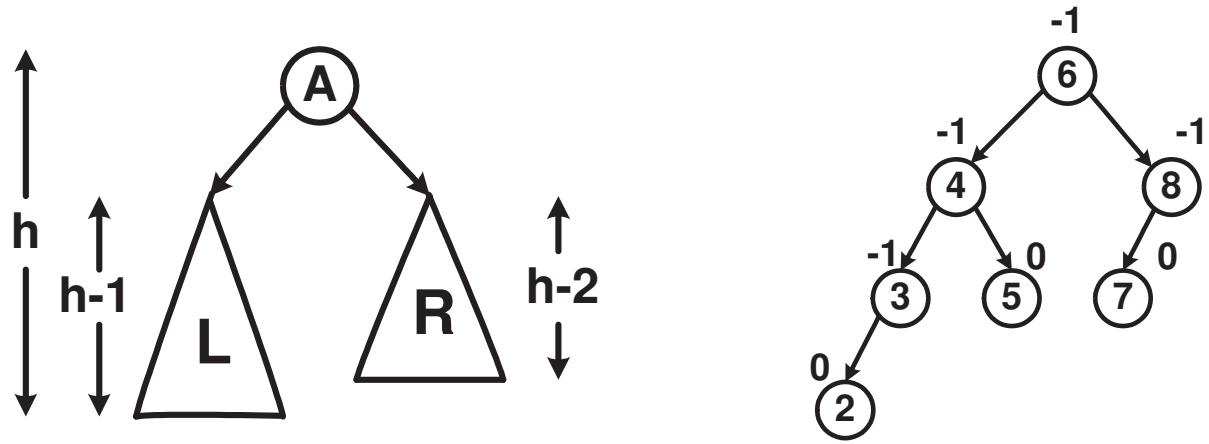


FIGURE: Dvejetainiai paieškos medžiai: a) AVL medis, b) tai nėra AVL medis, nes 17 viršūnės kairiojo pomedžio aukštis yra lygus 2, o dešiniojo pomedžio aukštis yra 0, taigi jų skirtumas yra -2.

Realizuodami AVL paieškos medį modifikuojame dvejetainio medžio *elementą*, jame papildomai saugome informaciją apie viršūnės subalansuotumo faktorių:

```
struct nodeAVL {
    T data;
    int key;
    nodeAVL * left;
    nodeAVL * right;
    nodeAVL * p;
    int balance;
}
```

Labiausiai išbalansuoto AVL medžio kiekvienos viršūnės kairiojo (arba dešiniojo) pomedžio aukštis yra vienetu didesnis už dešiniojo (atitinkamai, kairiojo) pomedžio aukštį. Tokio medžio T_h schema ir AVL medžio pavyzdys yra pateikti paveiksle.



a) b)

FIGURE: Labiausiai išbalansuotas AVL paieškos medis: a) bendroji medžio T_h schema, b) AVL medžio pavyzdys, kai kairiojo pomedžio aukštis visada didesnis už dešiniojo pomedžio aukštį.

Pažymėkime N_h labiausiai išbalansuoto AVL medžio viršūnių skaičių. Iš pateiktosios medžio schemos gauname lygtį

$$N_h = N_{h-1} + N_{h-2} + 1\,.$$

Papildomai suformuluojame dvi pradines sąlygas. Jas gauname iš AVL medžio apibrėžimo $N_0 = 0$, $N_1 = 1$.

Pažymėkime N_h labiausiai išbalansuoto AVL medžio viršūnių skaičių. Iš pateiktosios medžio schemos gauname lygtį

$$N_h = N_{h-1} + N_{h-2} + 1\,.$$

Papildomai suformuluojame dvi pradines sąlygas. Jas gauname iš AVL medžio apibrėžimo $N_0 = 0$, $N_1 = 1$.
Tokią lygtį jau sutikote, kai skaičiavote Fibonačio sekos narius.

AVL medžio aukščio priklausomybę nuo viršūnių skaičiaus įvertiname nelygybe ($q_1 = (1 + \sqrt{5})/2$):

$$h_b(N) \leq \frac{\log(N+1)}{\log q_1} = 1.4404 \log N\,.$$

Net ir labiausiai nesubalansuoto AVL paieškos medžio aukštis tik 1.44 karto didesnis už pilnai subalansuoto medžio aukštį.

Duomenų įterpimas į AVL medį

AVL yra ir paieškos medis, tai iš pradžių naują viršūnę įterpiame naudodami dvejetainio paieškos medžio algoritmą.

Duomenų įterpimas į AVL medį

AVL yra ir paieškos medis, tai iš pradžių naują viršūnę įterpiame naudodami dvejetainio paieškos medžio algoritmą.

Pažymėkime viršūnės, į kurios pomedį įterpėme naują elementą, kairiojo ir dešiniojo pomedžių aukščius iki įterpimo, atitinkamai h_L ir h_R.

Duomenų įterpimas į AVL medį

AVL yra ir paieškos medis, tai iš pradžių naują viršūnę įterpiame naudodami dvejetainio paieškos medžio algoritmą.

Pažymėkime viršūnės, į kurios pomedį įterpėme naują elementą, kairiojo ir dešiniojo pomedžių aukščius iki įterpimo, atitinkamai h_L ir h_R.

Tarkime, kad viršūnę įterpėme į kairįjį pomedį, tada po įterpimo jo aukštis padidėja vienetu. Išskirsime tris atvejus:

1. Iki įterpimo galiojo lygybė $h_L = h_R$, tada, įterpus naują viršūnę, pomedžių aukščiai skiriasi $h_L = h_R + 1$, bet AVL medžio subalansuotumo savybė vis dar yra tenkinama.

1. Iki įterpimo galiojo lygybė $h_L = h_R$, tada, įterpus naują viršūnę, pomedžių aukščiai skiriasi $h_L = h_R + 1$, bet AVL medžio subalansuotumo savybė vis dar yra tenkinama.

2. Iki įterpimo kairiojo pomedžio aukštis buvo mažesnis $h_L = h_R - 1$, tada, įterpę naują viršūnę, turime, kad $h_L = h_R$, taigi abiejų pomedžių aukščiai tapo vienodi.

1. Iki įterpimo galiojo lygybė $h_L = h_R$, tada, įterpus naują viršūnę, pomedžių aukščiai skiriasi $h_L = h_R + 1$, bet AVL medžio subalansuotumo savybė vis dar yra tenkinama.

2. Iki įterpimo kairiojo pomedžio aukštis buvo mažesnis $h_L = h_R - 1$, tada, įterpę naują viršūnę, turime, kad $h_L = h_R$, taigi abiejų pomedžių aukščiai tapo vienodi.

3. Iki įterpimo galiojo lygybė $h_L = h_R + 1$, po naujos viršūnės įterpimo $h_L = h_R + 2$, taigi pažeista AVL medžio subalansuotumo savybė.

1. Iki įterpimo galiojo lygybė $h_L = h_R$, tada, įterpus naują viršūnę, pomedžių aukščiai skiriasi $h_L = h_R + 1$, bet AVL medžio subalansuotumo savybė vis dar yra tenkinama.

2. Iki įterpimo kairiojo pomedžio aukštis buvo mažesnis $h_L = h_R - 1$, tada, įterpę naują viršūnę, turime, kad $h_L = h_R$, taigi abiejų pomedžių aukščiai tapo vienodi.

3. Iki įterpimo galiojo lygybė $h_L = h_R + 1$, po naujos viršūnės įterpimo $h_L = h_R + 2$, taigi pažeista AVL medžio subalansuotumo savybė.

Tada paieškos medį reikia papildomai pertvarkyti ir jį subalansuoti.

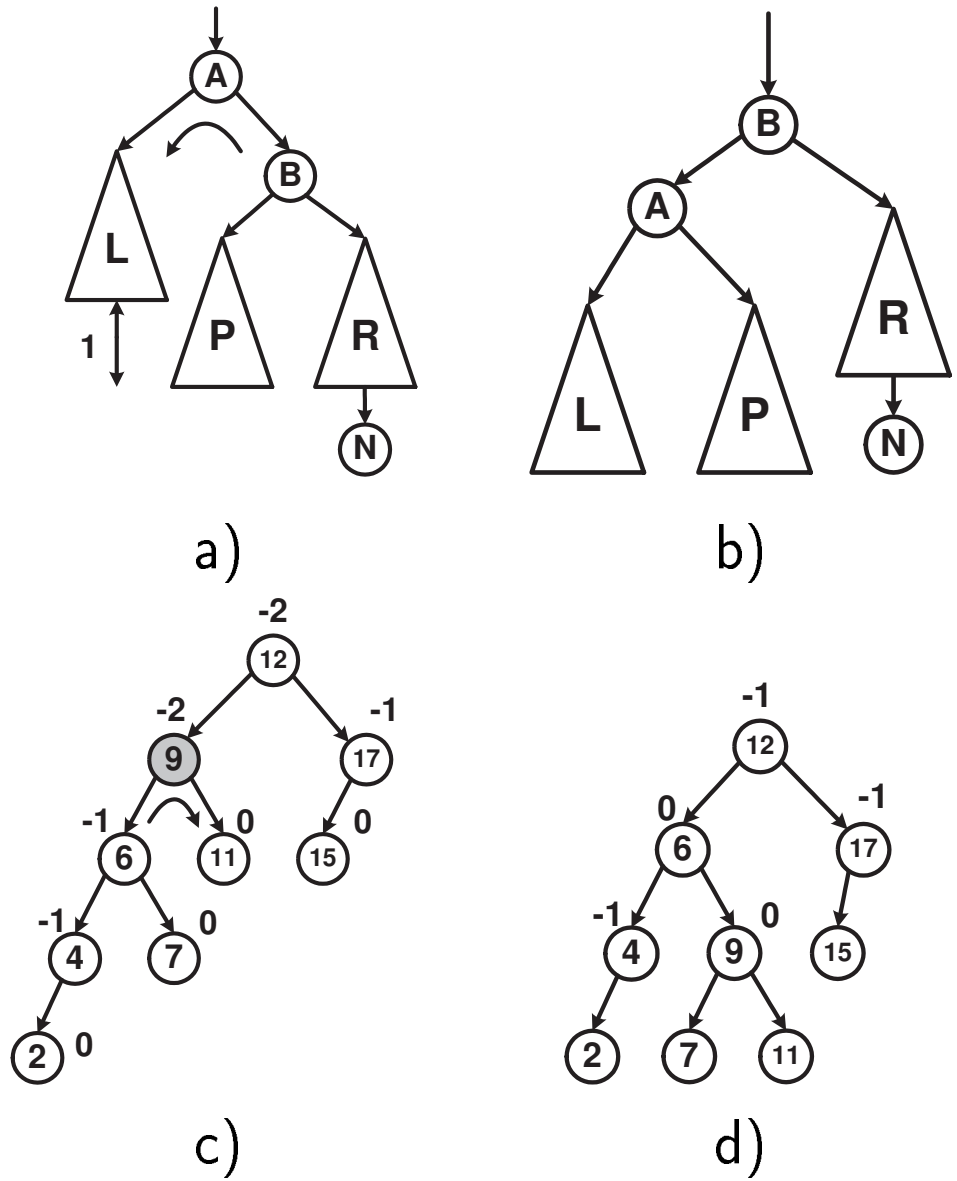


FIGURE: Balansavimas atliekant viengubą pasukimą.

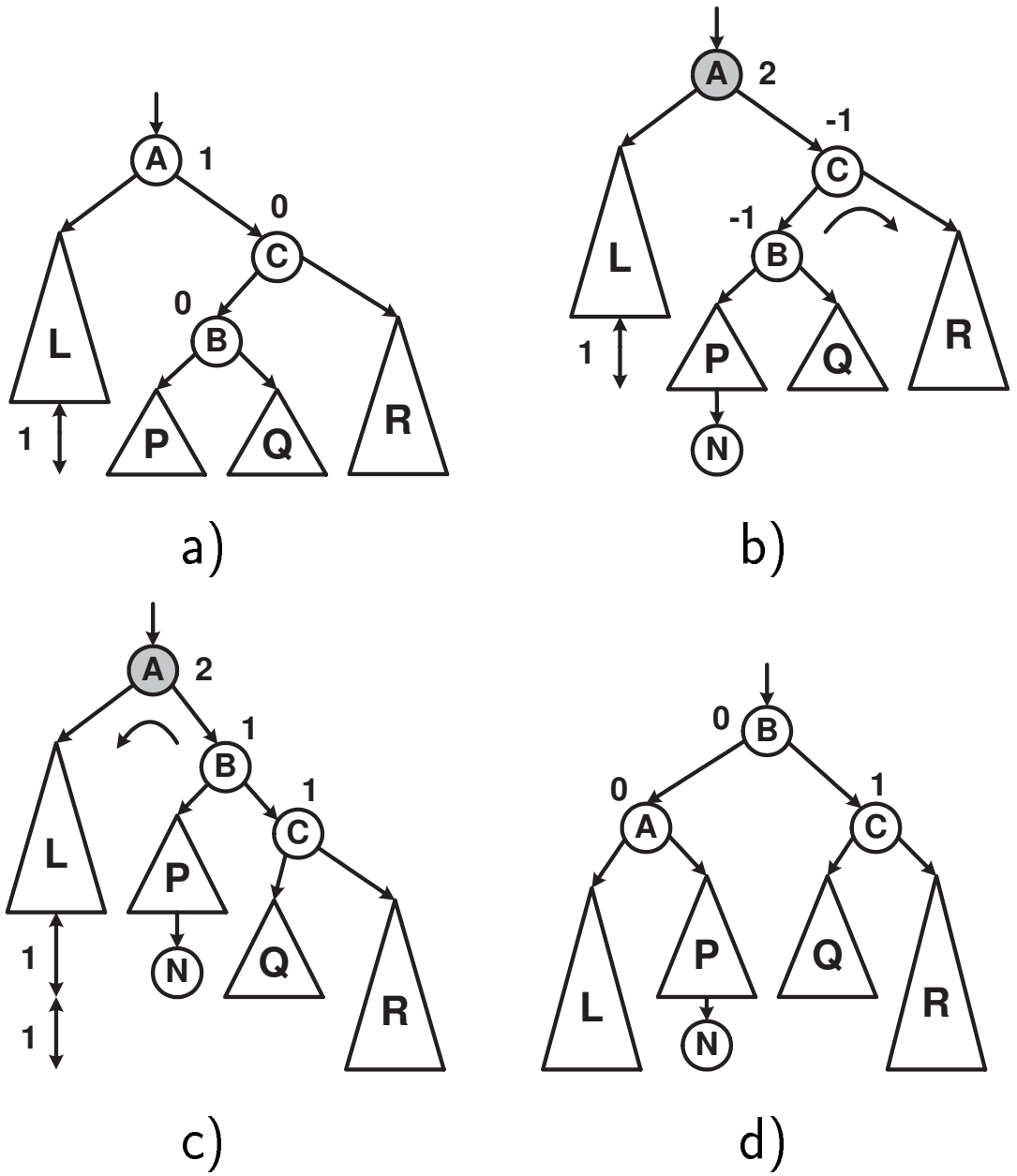


FIGURE: Balansavimas dvigubu pasukimu.

Įterpimo algoritmo bendroji schema

Po viršūnės įterpimo šia AVL medžio šaka vėl einame visą kelią tik dabar jau atvirkštine tvarka (naudojame rodyklę į viršūnės tėvą):

Įterpimo algoritmo bendroji schema

Po viršūnės įterpimo šia AVL medžio šaka vėl einame visą kelią tik dabar jau atvirkštine tvarka (naudojame rodyklę į viršūnės tėvą):

1. Nagrinėjamos viršūnės balansavimo faktorius lygus 0, tada jis keičiamas į (-1) arba 1, atsižvelgiant į tai, kuria medžio briauna į ją ateiname. Taip einame į viršų tol, kol pasiekiame medžio šaknį.

Įterpimo algoritmo bendroji schema

Po viršūnės įterpimo šia AVL medžio šaka vėl einame visą kelią tik dabar jau atvirkštine tvarka (naudojame rodyklę į viršūnės tėvą):

1. Nagrinėjamos viršūnės balansavimo faktorius lygus 0, tada jis keičiamas į (-1) arba 1, atsižvelgiant į tai, kuria medžio briauna į ją ateiname. Taip einame į viršų tol, kol pasiekiame medžio šaknį.

2. Nagrinėjamos viršūnės balansavimo faktorius lygus -1 arba 1.

Įterpimo algoritmo bendroji schema

Po viršūnės įterpimo šia AVL medžio šaka vėl einame visą kelią tik dabar jau atvirkštine tvarka (naudojame rodyklę į viršūnės tėvą):

1. Nagrinėjamos viršūnės balansavimo faktorius lygus 0, tada jis keičiamas į (-1) arba 1, atsižvelgiant į tai, kuria medžio briauna į ją ateiname. Taip einame į viršų tol, kol pasiekiame medžio šaknį.

2. Nagrinėjamos viršūnės balansavimo faktorius lygus -1 arba 1.

   2a. Jei į ją ateiname iš trumpesniojo pomedžio, tai viršūnės balansavimo faktorius tampa 0, ir procedūrą baigiame.

   2b. Jei į viršūnę ateiname iš ilgesniojo pomedžio, tai atliekame vieną iš balansavimo veiksmų.

Kadangi po balansavimo kritinio pomedžio aukštis yra toks pat, kaip ir iki naujos viršūnės įterpimo, tai procedūrą irgi baigiame.

Piramidė

Susipažinsime su dar vienu dvejetainiu medžiu – *piramide* arba *krūva* (angl. *heap*). Jai būdingos šios dvi pagrindinės savybės:

1. Piramidė yra pilnas, subalansuotas medis: pirmiausia užpildoma medžio šaknis, paskui pirmojo lygmens viršūnės, antrojo lygmens viršūnės ir t. t. Kiekvieną lygmenį užpildome iš kairės į dešinę.

Piramidė

Susipažinsime su dar vienu dvejetainiu medžiu – *piramide* arba *krūva* (angl. *heap*). Jai būdingos šios dvi pagrindinės savybės:

1. Piramidė yra pilnas, subalansuotas medis: pirmiausia užpildoma medžio šaknis, paskui pirmojo lygmens viršūnės, antrojo lygmens viršūnės ir t. t. Kiekvieną lygmenį užpildome iš kairės į dešinę.
2. Medis yra sutvarkytas taip, kad kiekvienos viršūnės vaikai yra nedidesni už pačią viršūnę.

Piramidė

Susipažinsime su dar vienu dvejetainiu medžiu – *piramide* arba *krūva* (angl. *heap*). Jai būdingos šios dvi pagrindinės savybės:

1. Piramidė yra pilnas, subalansuotas medis: pirmiausia užpildoma medžio šaknis, paskui pirmojo lygmens viršūnės, antrojo lygmens viršūnės ir t. t. Kiekvieną lygmenį užpildome iš kairės į dešinę.
2. Medis yra sutvarkytas taip, kad kiekvienos viršūnės vaikai yra nedidesni už pačią viršūnę.

Piramidė

Susipažinsime su dar vienu dvejetainiu medžiu – *piramide* arba *krūva* (angl. *heap*). Jai būdingos šios dvi pagrindinės savybės:

1. Piramidė yra pilnas, subalansuotas medis: pirmiausia užpildoma medžio šaknis, paskui pirmojo lygmens viršūnės, antrojo lygmens viršūnės ir t. t. Kiekvieną lygmenį užpildome iš kairės į dešinę.
2. Medis yra sutvarkytas taip, kad kiekvienos viršūnės vaikai yra nedidesni už pačią viršūnę.

Iš antrosios savybės seka išvada, kad piramidės šakninėje viršūnėje saugomas didžiausiasis aibės elementas.

Kadangi piramidė yra pilnas dvejetainis medis, tai jos viršūnes labai patogu saugoti masyve.

Kadangi piramidė yra pilnas dvejetainis medis, tai jos viršūnes labai patogu saugoti masyve.

Piramidės pavyzdys pavaizduotas paveiksle:

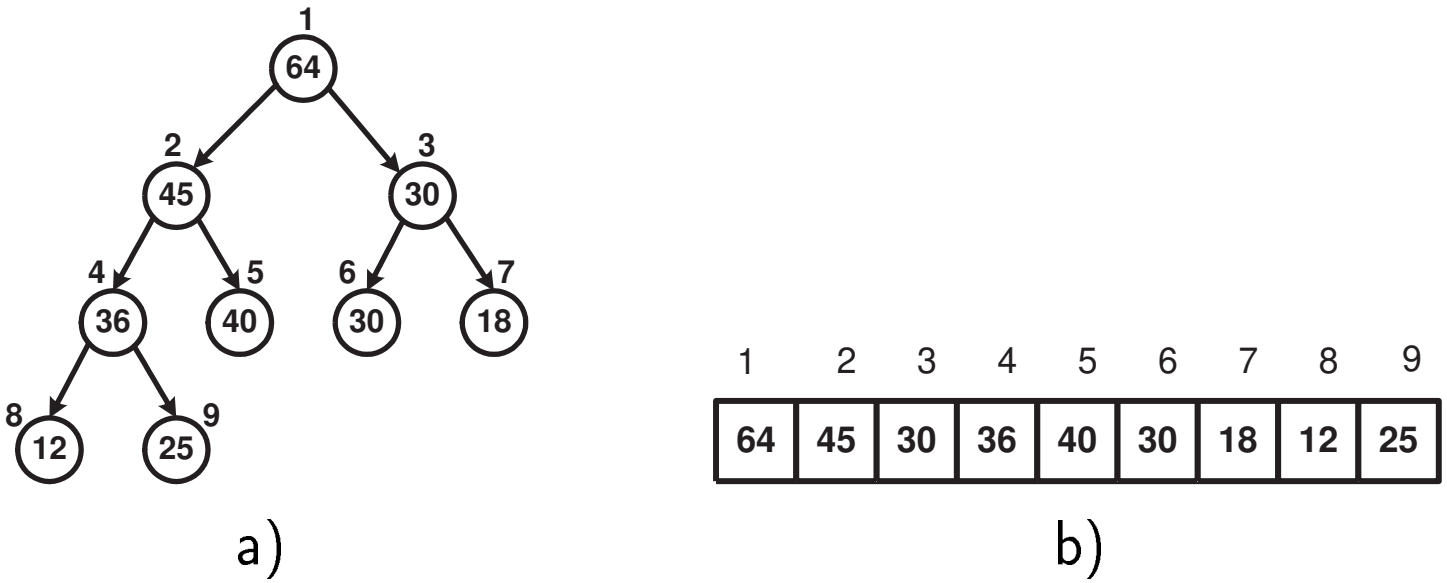


FIGURE: Piramidės pavyzdys: a) dvejetainis medis, b) masyvas

Kadangi piramidė yra pilnas dvejetainis medis, tai jos viršūnes labai patogu saugoti masyve.

Piramidės pavyzdys pavaizduotas paveiksle:

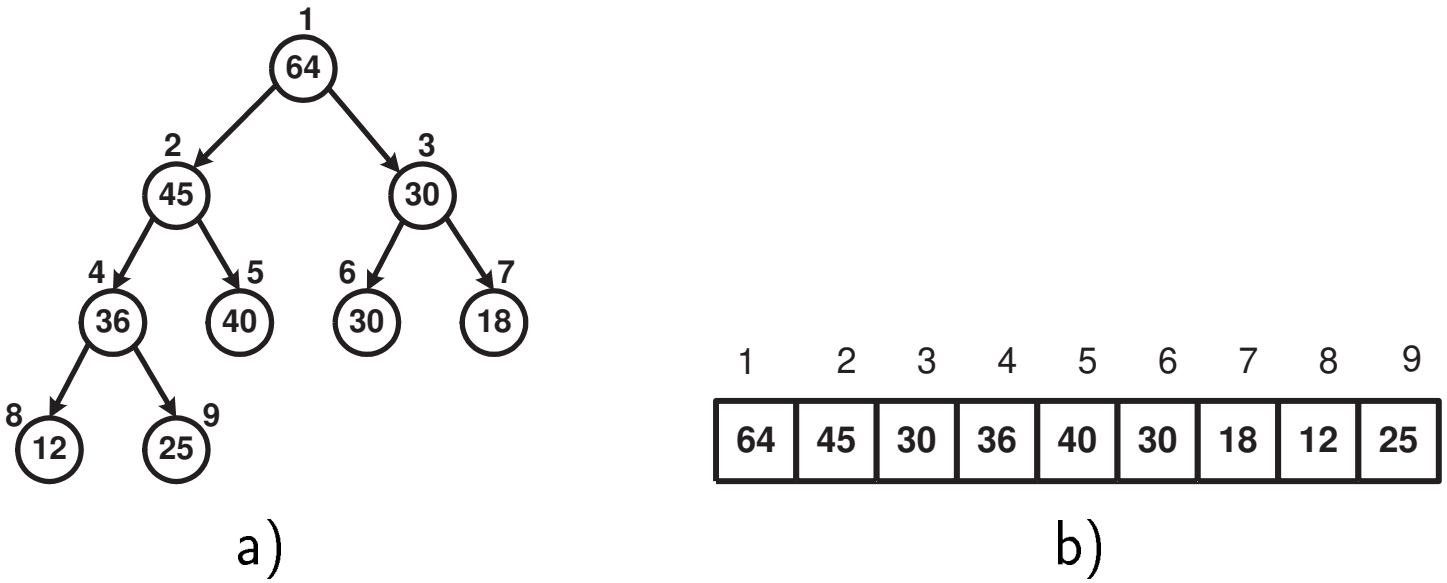


FIGURE: Piramidės pavyzdys: a) dvejetainis medis, b) masyvas

Tada i-tosios viršūnės a_i vaikai yra masyvo elementai a_{2i}, a_{2i+1}. Lengvai randame kiekvienos piramidės viršūnės a_i tėvą: ši viršūnė saugoma $j = \lfloor i/2 \rfloor$-ajame masyvo elemente a_j.

Piramidės formavimas. Turime duomenis $e_1, e_2, \ldots, e_N$, iš kurių reikia sudaryti piramidę. Šiuos elementus paeiliui talpiname į A masyvą.

Piramidės formavimas. Turime duomenis $e_1, e_2, \ldots, e_N$, iš kurių reikia sudaryti piramidę. Šiuos elementus paeiliui talpiname į A masyvą.

Visos dvejetainio medžio viršūnės – lapai jau yra sutvarkyti. Piramidės lapų indeksai kinta nuo $(\frac{N}{2} + 1)$ iki N.

Piramidės formavimas. Turime duomenis $e_1, e_2, \ldots, e_N$, iš kurių reikia sudaryti piramidę. Šiuos elementus paeiliui talpiname į A masyvą.

Visos dvejetainio medžio viršūnės – lapai jau yra sutvarkyti. Piramidės lapų indeksai kinta nuo $(\frac{N}{2} + 1)$ iki N.

Paskui paeiliui imame viršūnes $\frac{N}{2}, \ldots, 1$ ir rekursyviai tikriname piramidės sutvarkymo sąlygą. Jei ji pažeista, sukeičiame vietomis šią viršūnę su didžiausiu jos vaiku.

Piramidės formavimas. Turime duomenis $e_1, e_2, \ldots, e_N$, iš kurių reikia sudaryti piramidę. Šiuos elementus paeiliui talpiname į A masyvą.

Visos dvejetainio medžio viršūnės – lapai jau yra sutvarkyti. Piramidės lapų indeksai kinta nuo $(\frac{N}{2} + 1)$ iki N.

Paskui paeiliui imame viršūnes $\frac{N}{2}, \ldots, 1$ ir rekursyviai tikriname piramidės sutvarkymo sąlygą. Jei ji pažeista, sukeičiame vietomis šią viršūnę su didžiausiu jos vaiku.

Pratęsiame duotosios viršūnės vietos tikrinimą – tam ir naudojame rekursiją

Piramidės formavimo algoritmas

```
MakeHeap ()
begin
  (1)  for ( i=1; i ⩽ N; i++ ) do
  (2)       a_i = e_i;
       end do
  (3)  for ( i=N/2; i > 0; i - - ) do
  (4)       HeapDownOrder ( i, N );
       end do
end MakeHeap
```

Funkciją realizuojame iteraciniu būdu, nes visada tvarkome ne daugiau kaip tik vieną pomedį.
Taip pat labai svarbi savybė, kad pradedant nuo p-tosios viršūnės kiekvienos viršūnės abu pomedžiai tenkina piramidės sąlygas.

Funkciją realizuojame iteraciniu būdu, nes visada tvarkome ne daugiau kaip tik vieną pomedį.
Taip pat labai svarbi savybė, kad pradedant nuo p-tosios viršūnės kiekvienos viršūnės abu pomedžiai tenkina piramidės sąlygas.

```
HeapDownOrder ( p, N )
begin
  (1)   i=p;    j = 2i;
  (2)   while   ( j ⩽ N ) do
  (3)         k = j;
  (4)         if   ( (j+1) ⩽ N ) then
  (5)           if   ( a_{j+1} > a_j ) k = j+1;
  (6)         if   ( a_i < a_k ) then
  (7)           swap (a_i, a_k);
  (8)           i = k;    j= 2i;
  (9)         else
  (10)          j = N+1;
end HeapDownOrder
```

Iš skaičių masyvo $A = (10, 37, 18, 13, 22, 14, 25, 10, 12, 28)$ suformuosime piramidę.

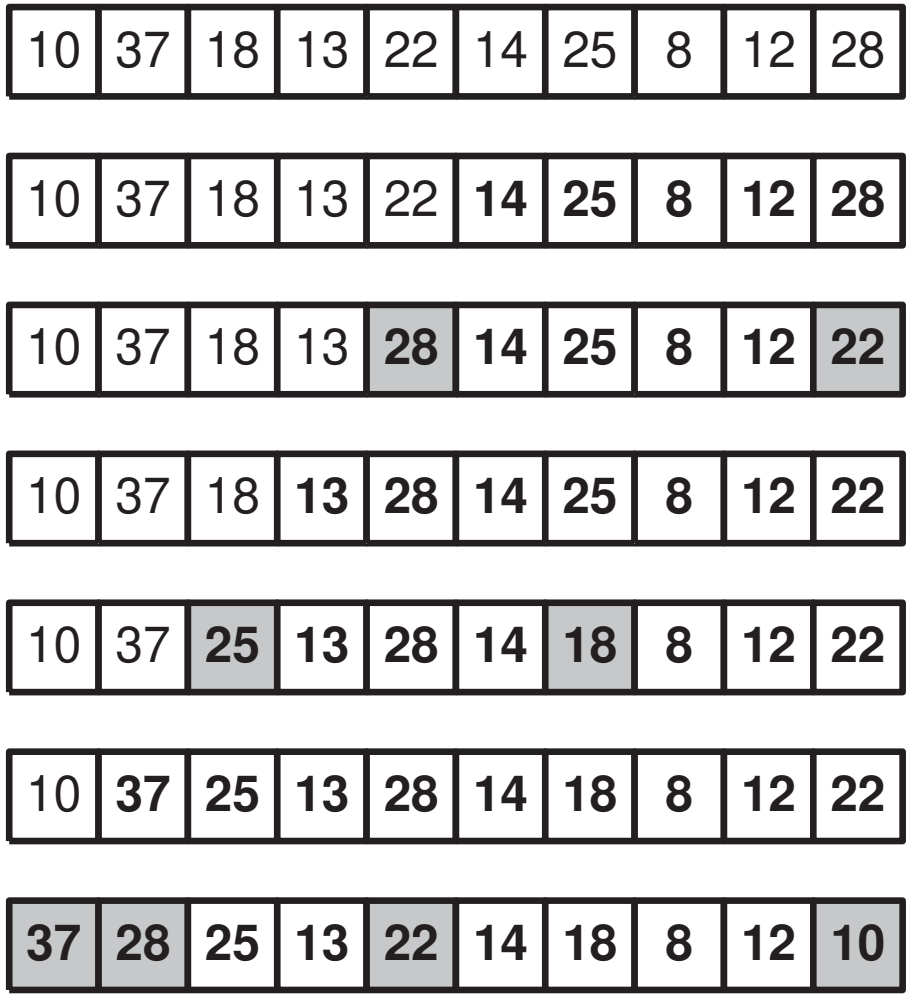


FIGURE: Skaičių masyvo pertvarkymas į piramidės struktūrą.

Įvertinsime piramidės formavimo kaštus.

Kadangi piramidė yra pilnas dvejetainis medis, tai jos aukštis ne didesnis už $\log N$.

Todėl kiekvienos elementų tvarkymo operacijos metu įvykdome ne daugiau kaip $2 \log N$ elementų lyginimo veiksmų ir $\log N$ elementų sukeitimų. Taigi piramidės formavimo algoritmo sudėtingumą galime įvertinti iš viršaus taip

$$L(N) \leq N \log N, \quad S(N) \leq \frac{1}{2} N \log N .$$

Atliksime tikslesnę sudėtingumo analizę. Pirmame lygyje virš lapų yra $N/4$ viršūnių ir su kiekviena iš jų atliekame po vieną koregavimo veiksmą, kurio sudėtingumą vertinsime konstanta c.

Antrame lygyje yra dvigubai mažiau viršūnių bei atliekame po 2 koregavimo veiksmus. Tęsiame procesą iki viršūnės-šaknies, kurios pertvarkymas gali pareikalauti $\log N$ koregavimo veiksmų.

Atliksime tikslesnę sudėtingumo analizę. Pirmame lygyje virš lapų yra $N/4$ viršūnių ir su kiekviena iš jų atliekame po vieną koregavimo veiksmą, kurio sudėtingumą vertinsime konstanta c.

Antrame lygyje yra dvigubai mažiau viršūnių bei atliekame po 2 koregavimo veiksmus. Tęsiame procesą iki viršūnės-šaknies, kurios pertvarkymas gali pareikalauti $\log N$ koregavimo veiksmų.

Taigi bendri piramidės formavimo kaštai yra (skliaustuose esanti suma konverguoja, kai $N \to \infty$, pagal Dalambero požymį):

$$L(N) = \frac{cN}{2}\left(\frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{\log N}{N/2}\right) = \Theta(N).$$